

Categories and Compositionality

with a view to Applications



You are reading a work-in-progress book created for the class Applied Compositional Thinking for Engineers (ACT4E):

applied-compositional-thinking.engineering

Please visit the website to get the last updated edition.

Compiled on 2024-12-09.

Instructor copies: If you are an instructor, please contact us to obtain a copy of the book with the solutions to the graded exercises.

Contents

Contents

	C. ORDER	83
5. Posets	85	
5.1. Trade-offs	86	
5.2. Ordered sets	90	
5.3. Counting orders	95	
5.4. Power poset	96	
5.5. Chains and Antichains	97	
5.6. Measuring posets	99	
6. Constructing posets	101	
6.1. Product of posets	102	
6.2. Disjoint union of posets	104	
6.3. Opposite of a poset	105	
6.4. “Twisted” poset of intervals	106	
6.5. Arrow poset of intervals	108	
7. Monotonicity	109	
7.1. Monotone maps	110	
7.2. Monotone relations and design problems	115	
7.3. Order on monotone maps	121	
8. Poset bounds	123	
8.1. Minimal and maximal elements	124	
8.2. Upper/lower bounds	125	
8.3. Upper and lower sets	127	
8.4. Antichains	128	
	D. ALGEBRA	133
9. Sets with operations	135	
9.1. Magmas	136	
9.2. Semigroups	137	
9.3. Monoids	141	
9.4. Groups	144	
9.5. Rings, fields	149	
10. Morphisms	151	
10.1. Semigroup morphisms	152	
10.2. Encoding as morphism	154	
10.3. Morse coding	155	
10.4. Monoid morphisms	157	
10.5. Group morphisms	159	
10.6. Generators and relations	161	
11. Actions	165	
11.1. Actions	166	
11.2. Modules, Vector spaces	170	

11.3. Linear group actions 11.4. Dynamical systems E. CATEGORIES 177	171 173 F. FUNCTORS 251	
12. Graphs 12.1. Graphs 12.2. Graph homomorphisms 13. (Semi)categories 13.1. Interfaces 13.2. Semicategories 13.3. Categories 13.4. Diagrams 13.5. Categories vs graphs 13.6. Categories from graphs 14. Categories and structures 14.1. Categories of sets and functions . . . 14.2. Categories of relations 14.3. Categories of semigroups, monoids, groups 14.4. Categories from linear algebra . . . 14.5. Categories of posets 14.6. Sets with data 14.7. Categories of graphs 14.8. Preorders as categories 14.9. Monoids as categories 15. Modeling with categories 15.1. Mobility 15.2. Trekking in the Swiss Mountains . . 15.3. Currency categories 15.4. Resources dependencies 15.5. DP as a category 15.6. Procedures 15.7. Software dependencies 16. Constructing categories 16.1. Product of Categories 16.2. Disjoint Union of Categories . . . 16.3. Opposite Category 16.4. Arrow construction 16.5. Twisted arrow construction 16.6. (Co)slice construction 17. Culture 17.1. Definition vs computation	17.2. Things that don't matter 17.3. Choice of symbols 17.4. Typographical conventions 18. (Semi)Category actions 18.1. Moore machines, first version . . . 18.2. The category $\langle \text{Set} \rangle$ 18.3. Moore machines, $\langle \text{Set} \rangle$ version . . 18.4. Standard action of Moore machines . 18.5. Semicategory actions 18.6. More machines 18.7. LTI systems 19. Translation 19.1. Layers of abstraction 19.2. Semifunctors 19.3. Functors 19.4. More examples of functors 19.5. Categorical Databases 20. Specialization 20.1. Subcategories 20.2. Subcategories of endomorphisms . . 20.3. Other examples 20.4. Subcategories of Berg 21. Syntax and semantics 21.1. Specification verses behavior	
		G. NATURALITY 307
		23. Naturality 23.1. Natural transformations 23.2. Morphisms in a category of Functors 23.3. Data migration 23.4. More examples 24. Adjunctions 24.1. Formal concept analysis 24.2. Galois connections 24.3. Adjunctions: hom-set definition . . .

24.4. Adjunctions: (co)unit definition	335	31. Lattices	439
24.5. Product-Hom adjunction	337	31.1. Monoidal posets	440
24.6. Free-forgetful adjunction	338	31.2. Monoidal-time procedures	442
24.7. Relating the two definitions	340	31.3. Lattices	444
H. INTERCONNECTIONS	343	31.4. Lattice homomorphisms	447
25. Parallel composition	345	31.5. Categories Lat and BoundedLat .	448
25.1. Modeling parallelism	346	32. Lattice structure of DPs	449
25.2. Stacking categories	347	32.1. Ordering DPs	450
25.3. Functorial stacking categories	350	32.2. Interaction with series composition	451
25.4. Associative stacking categories	354	32.3. Union of Design Problems	452
25.5. Monoidal categories	357	32.4. Intersection of Design Problems . .	453
25.6. Monoidal functors	364	32.5. Lattice structure of DP hom-sets .	454
25.7. Strictification	366	32.6. Interaction with composition	457
25.8. The case of semicategories	371	33. Constructing design problems	459
26. Crossing wires	377	33.1. Companion and conjoint use	460
26.1. Symmetric monoidal categories	378	33.2. Companions and conjoint	462
26.2. PROPs	381	33.3. Monoidal DPs	464
27. Feedback	383	J. UNIVERSAL PROPERTIES	467
27.1. Traced symmetric monoidal categories	384	34. Sameness	469
27.2. Partial traces	390	34.1. Sameness in category theory	470
27.3. Feedback categories	394	34.2. Isomorphism is not identity	473
27.4. Dual objects and morphisms	396	K. COMPOSITIONAL COMPUTATION	477
27.5. Canonical trace	399	35. DP queries as functors	479
I. CO-DESIGN	403	35.1. Queries are functors from problem statements to solutions	480
28. Design	405	35.2. The Pos_U and Pos_L categories	482
28.1. What is “design”?	406	35.3. Queries as functors	494
28.2. What is “co-design”?	407	36. Solving finite co-design problems	499
28.3. Formal engineering design	409	36.1. Domain theory and fixed points . .	500
28.4. Queries in design	411	36.2. Finite co-design problems	504
29. Monotone Co-Design Theory	413	36.3. Handling loops	505
29.1. DPs	414	36.4. Example: Optimizing over the natural numbers	508
29.2. Examples	417	36.5. Extended Numerical Examples . . .	511
29.3. Queries	423	36.6. Complexity of the solution	520
29.4. Co-design problems	425	36.7. Decomposition of CDPs	522
29.5. The semicategory DPI	429	37. Monads	525
29.6. Sum and intersection of DPs	431	37.1. Generalized objects and operations .	526
30. Feasibility	433		
30.1. From DPs to DPs	434		

37.2. Monads	531	BACK MATTER	545
37.3. The Kleisli construction	535	Example exams	555
37.4. Algebras of a monad	537	1. Exam 1	556
37.5. Monads from Adjunctions	541	REFERENCES	557

INTRODUCTION | PART A.



-
- | | |
|----------------------------|---|
| 1. The view from above | 3 |
| 2. Putting things together | 9 |
-



1. The view from above

In this first chapter, we give some motivation to look at the material in this book.

What we aim to do is to accompany you to the top of the Peak of Abstraction and show you *the view from above*, from which you will be able to see that many things that look different have the same structure.

1.1 Sapiens: a retrospective	4
1.2 Compositionality for engineers	5
1.3 Departures from tradition	6
1.4 Acknowledgments	8

1.1. *Sapiens*: a compositional retrospective

The word *intelligence*, from Latin *intellego*, comes from Proto-Indo-European **h₁entér* (“between”) and **leg-* (“to gather”), and we can translate it as *the ability to gather things together* to obtain some goal; this, for us, is the essence of intelligence.

The *things* to gather could be abstract, such as pieces of evidence to achieve a conclusion, or physical, like ingredients to prepare a tasty meal, or the parts to create machines that will prepare tasty meals.

Intelligence is not unique to *Homo sapiens*; other animals can reason, build, and use language. Some animals excel at things that *sapiens* cannot do. The sapienses will never experience the richness of the smells of things and processes in the world like *canis* does. The sapienses will never have the same spatial awareness of an octopus with eight arm-legs and neurons distributed all along its body.

But sapienses developed a trick no other animal did. They mastered **abstraction and compositionality**. Composing and decomposing is what gave sapiens an edge.

Sapiens decomposed the process of survival and created **societies**. Once, a single sapiens had the ability to survive by themselves, or in a very small pack of sapiens. Eventually they figured out that it was much more efficient to divide up the work, so that some could specialize in hunting, some in gathering, some in fighting, some in rearing children.

About 10,000 years ago, sapiens invented **agriculture**; it was a momentous change, as it was the first time that they could change the world around them and bend nature to their will. Up to that moment, it was the other way around: as they moved beyond Africa, sapienses adapted to the environment; bodies optimized to run after prey in savannas became optimized to fish in tropical seas or to herd cattle on the Alps.

More specialization. Today only 1% work at food production. In fact, you could take most sapiens and put them in the most fruit-rich plains, the most prey-rich savanna, and, alone, they would die in days.

With a sedentary society, while the grains and the rice grew by themselves in the field, they found the time to invent **writing**. They managed to decompose thought into a sequence of symbols, which could be written on clay, and re-composed back by the receiver to reconstruct the original thought. Writing is a teleportation device and a time machine.

Money is abstraction of resources.

Artisans, creating the product from start to finish, are inefficient. In the **industrial age**, workflows are decomposed in steps, and a system is put in place for the product to be assembled from each step.

Lately, Sapiens has cultivated an inclination to create **machines that could help them think**. The most important conceptual shift is that Sapiens needs now to express knowledge *formally* and *computationally*. By “formally”, we mean the choice of a formal system shared by man and machine. By “computationally”, we mean that such knowledge needs to be able to produce actual results to use.

1.2. Compositional thinking for engineers

The thesis of this book is that most engineering fields would benefit from knowing and using the language of applied category theory to address the design and analysis of *complex systems*.

What is a “system”?

Here is a great quote*:

*A system is composed of components;
a component is something you understand.*

Howard Aiken

The first part of the quote, “A system is *composed of components*”, is plain as day as much as it is tautological. We could equally say: “A system is *partitioned in parts*”.

The second part, “a component is something you understand”, is where the insight lies: we call “system” what is too complex to be understood naturally by a human.

Aiken referred to computer engineering, but we find exactly the same sentiment expressed in other fields. In systems engineering, Leveson puts it as “complexity can be defined as intellectual unmanageability” [15].

We will be content of this anthropocentric and slightly circular definition of systems and complexity: “systems” are “complex” and “components” are “simple”.

Whether something is a complex system also depends on the task that we need to do with it. One way to visualize this is to imagine a “phenomenon” as a high-dimensional object that we can see from different angles. For each task, we have a different projection. The decomposition of the system in components can be different according to the task. For example, a system that might be easy to simulate could be very difficult to control.

The tools presented in this book will make it easier to think about different representations of the same system.

* This quote is by Howard Aiken (1900-1973), creator of the MARK I computer, as quoted by Kenneth E. Iverson (1920-2004), creator of programming language APL, as quoted inbut ultimately source-less and probably apocryphal.

1.3. Departures from traditional exposition [experts only]

This section is for experts only. Skip at a first reading.

This section describes the “departures from tradition” in our text. We made several choices to streamline the traditional exposition of category theory, to make it more understandable and relevant to the engineering field.

Induction vs deduction

The greatest difference between this text and a mathematical text is the use of an **inductive exposition** rather than a deductive explanation. In a typical mathematical exposition of category theory, one defines a general mathematical structure, and then give several specific examples [23].

Instead, here we first build up the examples as something that is interesting per se, and then we show how they can all be instances of the same general concept. In this way, the general concept is well motivated. The path laid by the book is one of *spiral learning*.

For example, we look at various constructions from specific to general:

$$\text{set product} \rightarrow \text{poset product} \rightarrow \text{categorical product}. \quad (1)$$

Similarly, we discuss

$$\text{monoid morphisms} \rightarrow \text{category actions} \rightarrow \text{functors}. \quad (2)$$

Materials covered

- ▷ Certain topics (limits, Yoneda’s lemma, *etc.*) that would be traditionally discussed relatively early, are not discussed in this volume. We ordered topics by usefulness in engineering.
- ▷ The main text uses traditional set theory. To ground the exercises, we use slightly more formal **type theory** foundations (setoids, *etc.*). It is in our plan to transition completely to type theory also in the main text. Please contact us if you can help!

Table 1.1.: Use of colors

sets	A, B
posets	P, Q
categories	C, D
objects	X, Y
morphisms	$f : X \rightarrow Y$
functors	$F : C \rightarrow D$
natural	$\alpha : F \Rightarrow G$
transformations	

Use of colors

- ▷ We **use colors** to aid in the parsing of formulas and diagrams (Table 1.1). We also color the composition operations. In this way it is easy to see the types at first glance: $f ; g$, $F ; G$, etc.
- ▷ Color is *not* necessary to infer meaning. The choice of colors is **colorblind-friendly** for red-green color blindness. (One of the authors is colorblind.) Please let us know if this is not the case.

Notation and conventions

- ▷ In general, we use diagrammatic notation $f ; g$ (pronounced “ f then g ”) rather than $g \circ f$ (pronounced “ g after f ”) for function and morphism composition.
- ▷ In the discussion of semigroups, we use “ \circ ” rather than “ \circ ” as the semigroup composition operation. This is because for us a semigroup/monoid is a special (semi)category with only one object.

- ▷ In Chapter 11 we discuss *covariant* and *contravariant* actions. We do not use the terms *left* and *right* actions because they are notation-dependent.
- ▷ We abundantly use *semicategories* (semifunctors, *etc.*). For us, semicategory is the primitive definition. A category is a semicategory with a particular property: having identities at each object.

Extensive use of tuples

We extensively use tuples and tuples concatenation to work directly with strict monoidal categories.

This is a list of standard categories together with their “tupled” definition.

Table 1.2.: Tuple subcategories of well-known categories

original	tuples subcategory
Set (Def. 13.11)	$\langle \text{Set} \rangle$ (Def. 18.3)
Pos (Def. 14.8)	$\langle \text{Pos} \rangle$ (Def. 25.33)
Rel (Def. 14.1)	$\langle \text{Rel} \rangle$ (Def. 25.35)

Treatment of monoidal categories

We noticed that there is a step increase in difficulty associated to natural transformations, without much immediate justification. In our trajectory, natural transformations appear first associated to monoidal categories. The role they play there is associated to very technical checks. Nothing exciting! We made the decision to provide a version of monoidal categories that are strict, so that there is no need for natural transformations.

We also noticed several examples of interest (*e.g.* proper LTI system) that are only semicategories, but they still have a notion of trace.

Furthermore, we noticed that in applications that there are several interesting examples that have a notion of vertical composition but the monoidal structure is not functorial on the nose: for example, systems with states.

Because there is a large part of concrete code exercises, it was not convenient for us to just wave our hands and say things like “let’s just consider the strict / modulo isomorphism version”.

In conclusion, in Chapter 25 we introduce several notions of “stacking” categories, which are defined for semicategories, are strict in the vertical composition operation, and for which the functoriality of the monoidal structure is not a given.

1.4. Acknowledgments

There is a multitude of causes that made it possible for this book to come into creation. Here is a short, incomplete list of the people that contributed to make this happen.

We thank David Spivak (Topos Institute) for infecting us with applied category theory.

We thank Joshua Tan (University of Oxford) to help define some concepts of monotone co-design theory.

We thank all the students at ETH Zürich who followed the class for being guinea pigs and giving constructive feedback.

We thank all the students of the online courses we did on the material for giving us the enthusiasm and the energy to get this done.

Sponsors



myswitzerland.com

Financial support obtained from ETH Zürich and the Swiss National Science Foundation through the NCCR Automation.

So, in a sense, this book is sponsored by Switzerland.

*Switzerland: come for the chocolate,
stay for the direct democracy.*

ETH zürich





2. Putting things together

In this chapter we discuss the various types of “compositions” we find in applications. The basic idea is thinking about *recipes* that produce something given a list of ingredients. We can ask many things about these recipes: does the order of ingredients matter? Can we go from results back to ingredients? Recipes can also be chained. And, we can think about meta-recipes that have other recipes as ingredients or results.

2.1	Stacking blocks	11
2.2	Mixing colors	14
2.3	Recipes as ingredients	16
2.4	Commutativity and associativity	17
2.5	Composing recipes	20
2.6	Isomorphisms	21

10

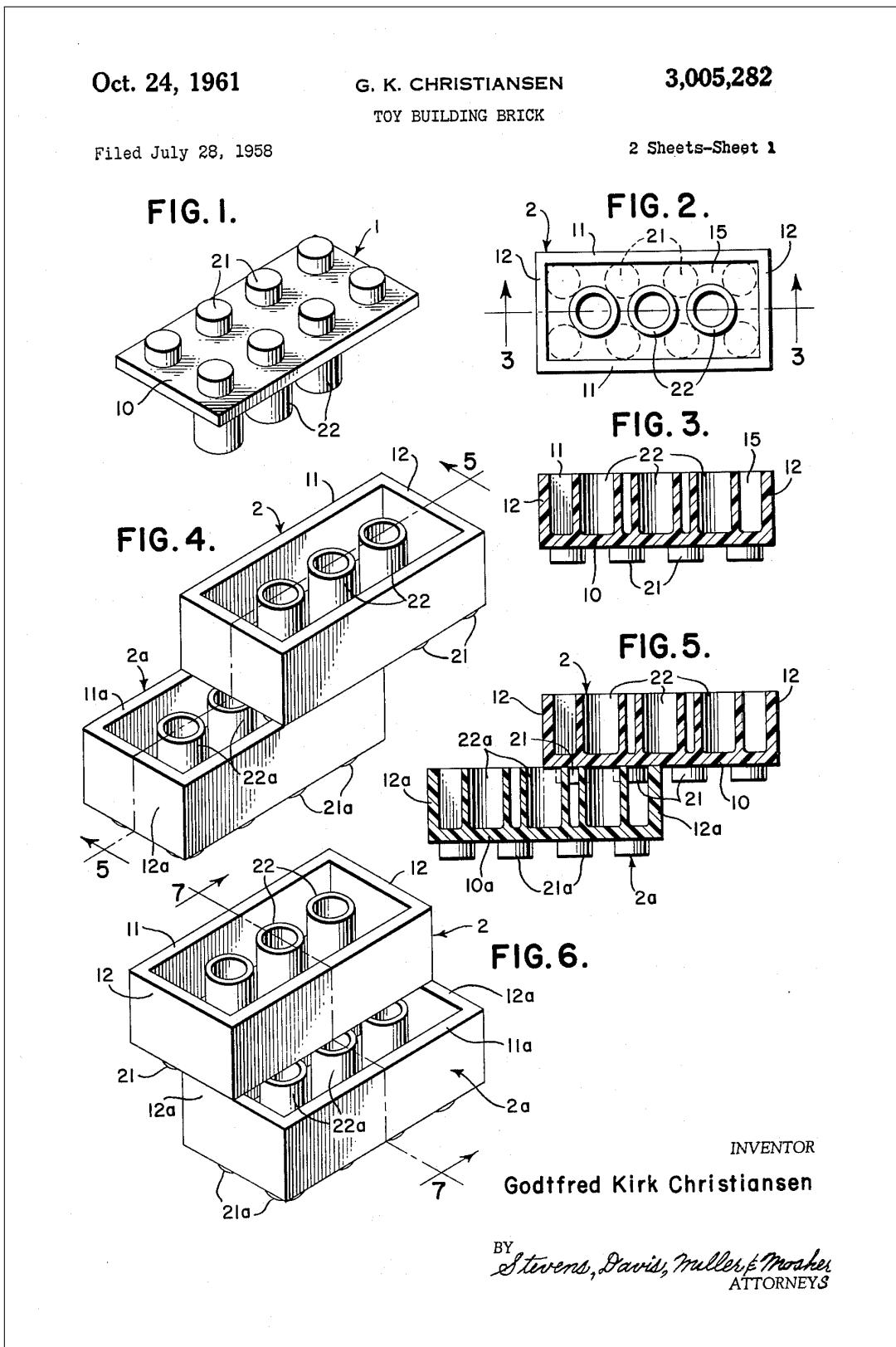


Figure 1.: The 1961 Lego patent.

2.1. Stacking blocks

The first encounter children have with composition is with toy blocks like Lego. It is a coincidence that there is a *lego* in *intellego* (as explained in 1.1); the *lego* in Lego is a contraction from Danish *leg godt*, which means *to play well*.

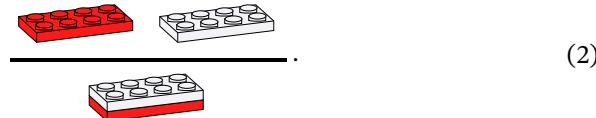
Legos are compositional in this sense: when you put together two blocks, you can treat the ensemble as one block for the purpose of composing it with other blocks.

We are going to use the following graphical notation to talk about composition. We draw a black bar, and we write the *ingredients* at the top, and the *results* at the bottom.

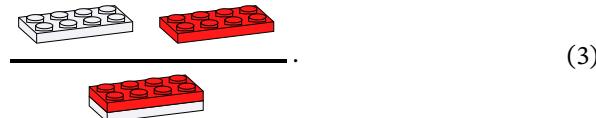
$$\begin{array}{c} \text{ingredient} & \text{ingredient} & \text{ingredient} \\ \hline & \text{result} & \text{result} \end{array} \quad (1)$$

Note that the order of the ingredients matters. For instance, we can have the following recipes for the composition of red and white bricks. We scan the list of ingredients from left to right and then place the bricks on top of what is already on the table.

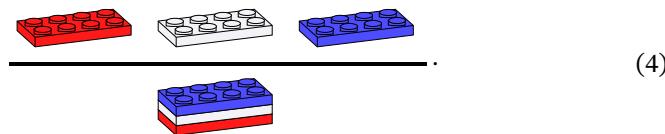
Composing red and white produces a red-white brick:



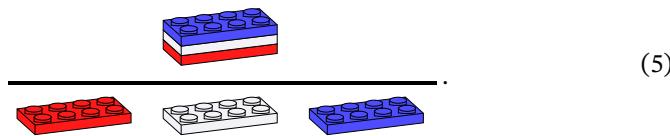
Composing white and red produces a white-red brick:



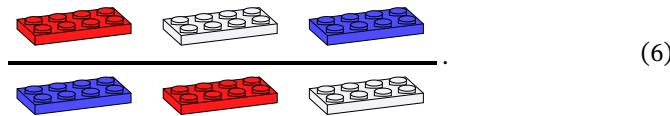
We can compose more than one brick. For example, red, white, blue, make a red-white-blue brick:



In Lego, we can also decompose. If we have a red-white-blue brick, we can also recover the single bricks:

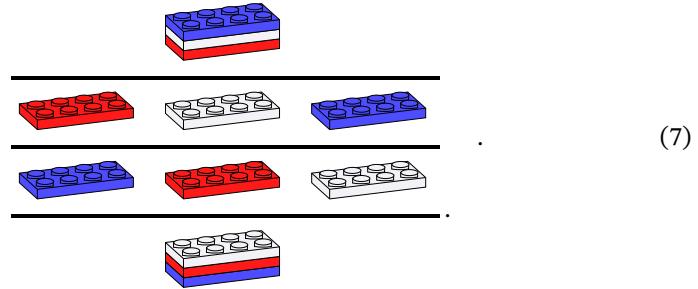


If you have 3 bricks on a table, you can also permute them:



Consequently, if you have a red-white-blue brick, you can disassemble, permute,

and reassemble to obtain a blue-white-red:

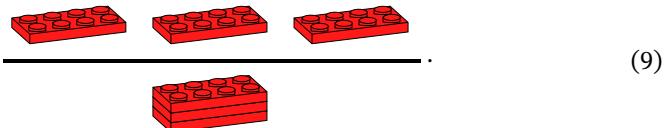


The aforementioned recipe contains several concrete steps to go from the initial ingredient to the final result. If we do not care about the detailed steps, we can summarize the recipe as follows, by eliding the intermediate steps and only remember the ingredient and the results:

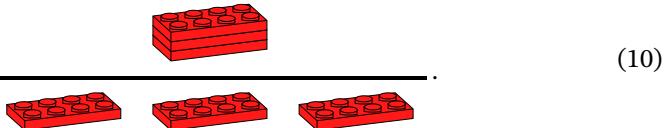


Alternatively, you can think of (8) as the statement of a theorem, and of (7) as the proof of the theorem.

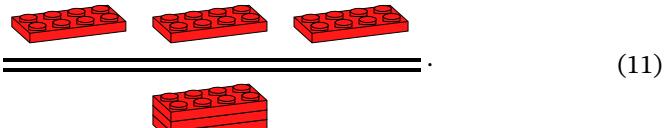
Sometimes we want to think about the transformations that are reversible. For example, we can assemble 3 red bricks into a red-red-red brick:



We can also do the opposite:



To describe the bi-directionality, we use a double line:



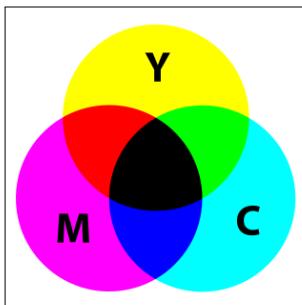
The flat pieces of Lego we have looked at are actually one third shorter than a “regular” piece:



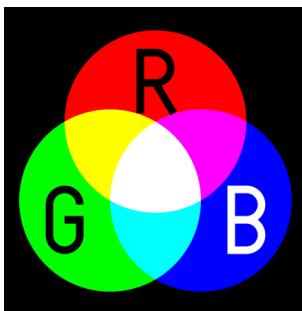
What is the relation between a red-red-red assembly and a full red brick? One point of view that will be very useful is thinking in terms of “substitution”: if I have one of those, can I use it as if I had the other? Lego bricks are very strong when assembled: a red-red-red assembly can certainly substitute a regular brick

in terms of structural functionality. Therefore, given a red-red-red we can treat it as a full block, but not vice versa:





(a) Subtractive composition

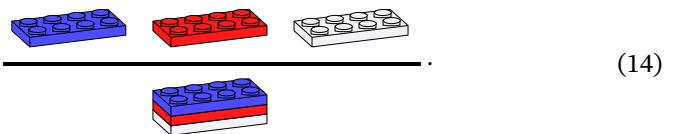


(b) Additive composition

Figure 2.: Additive vs subtractive composition

2.2. Mixing colors

We now look at how we can compose colors. In Denmark there is a small group of **Lego purists**: they are only able to conceive of Lego assemblies where all bricks have the same color. For them, a blue, red, white brick, make a block of a color they call *horrible*:



If you ask a color purist, they will tell you that red and red make red:



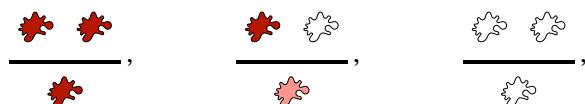
Furthermore, white and white make white:



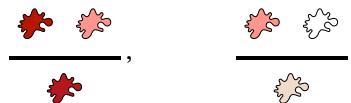
However, white and red make *horrible*:



We can think of many other ways to compose colors. For example, we can think of formalizing what happens when you **mix paint**. Red and white in equal measure give pink. By mixing and mixing we can obtain all the shades that go from red to white:

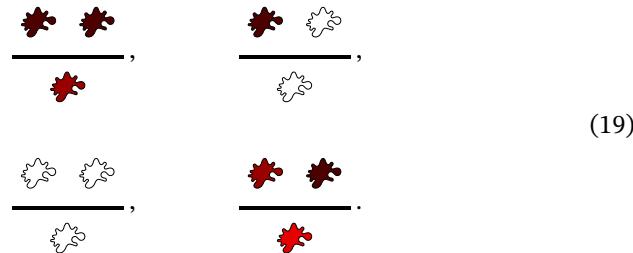


(18)



Colors on a monitor mix in an **additive** way. Two dark reds give a brighter red.

Red and white remains white:



Green, red, blue additively make white:



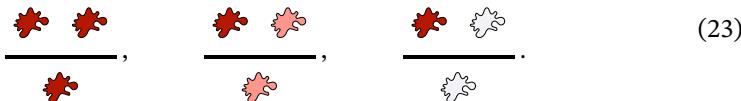
A different way to compose colors is by using the **subtractive** rules in the CMY (cyan, magenta, yellow) color space. These rules formalize the physical process of offset printing: we produce colors by putting pigments that block the other colors:



This is how you produce red, blue, green from CMY:



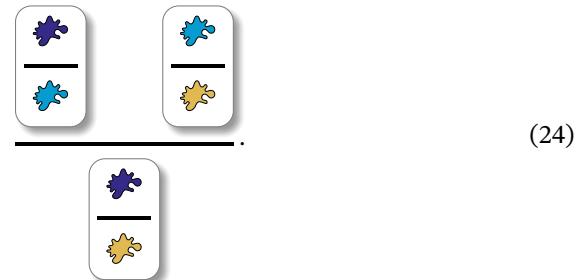
Finally, we can think of a **paint-over-it** composition rule: the first color is replaced by the second:



2.3. Recipes as ingredients

We can think at a higher level, by having recipes as ingredients.

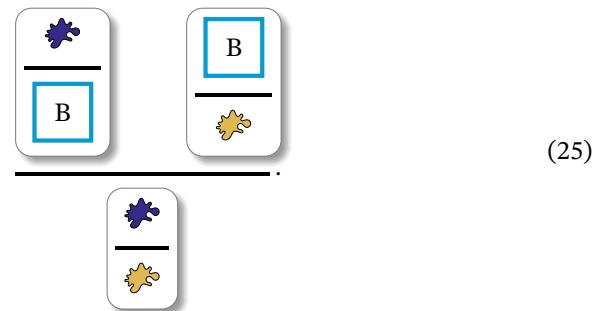
For example, the following shows that if a dark blue stain gives you a light blue stain, and a light blue stain gives you a orange stain, you can produce a orange stain from a dark blue stain:



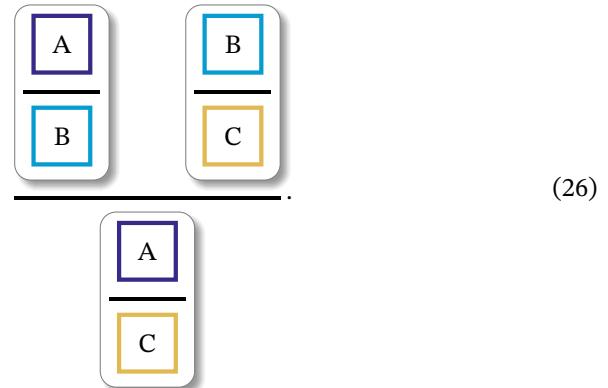
Note that to activate the meta-recipe above, no windsor stain was needed. In fact, for the above to be valid, it is not even necessary to postulate that windsor stains exist.

We can do **abstraction** by replacing some ingredients with *placeholders*. When we write a recipe with placeholders, we mean that the recipe is valid whatever is put in the placeholders, with the constraint that if two placeholders are of a similar color should hold the same thing.

For example, if a windsor stain gives me a B, and B gives me a rob roy stain, then a windsor stain gives me a rob roy stain, not matter what B is:



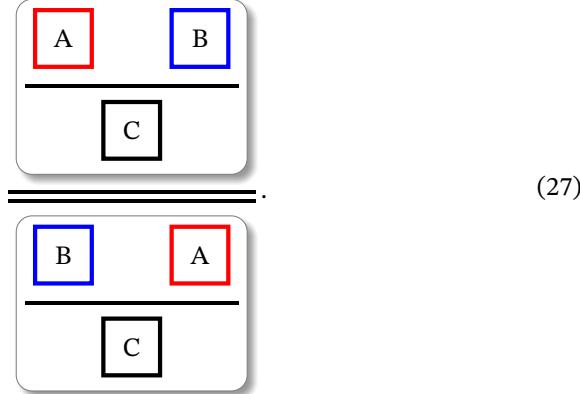
We can abstract further by saying that: if A gives me B, and B gives me C, then A gives me C:



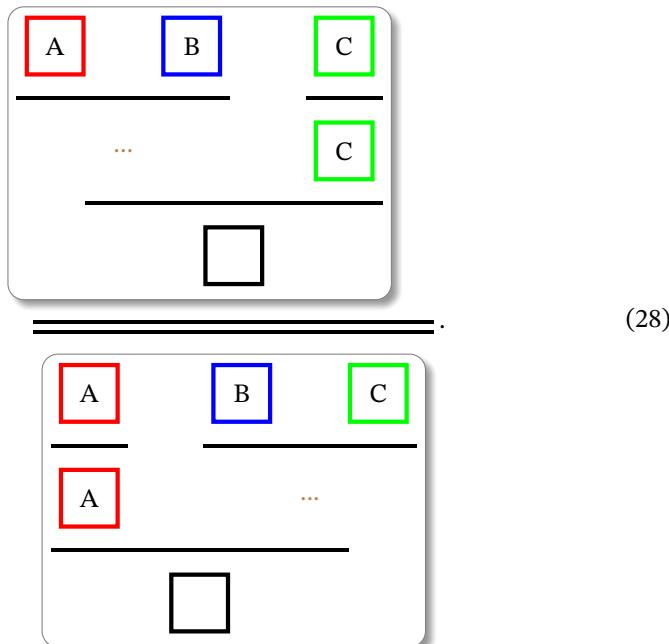
2.4. Commutativity and associativity

With the power of abstraction we can talk about properties of the rules themselves.

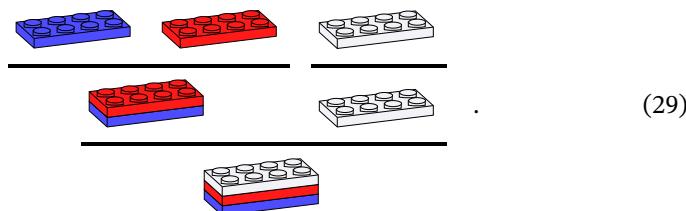
For example, we can define *commutativity* as follows. A composition operation is commutative if getting a C from A and B holds if and only if B and A also give a C:



For associativity, we want to say that, given three things A, B, C, composing A with B and then the result with C is the same thing as composing A with the result of B and C.



It is easy to see that this is valid for Lego composition.



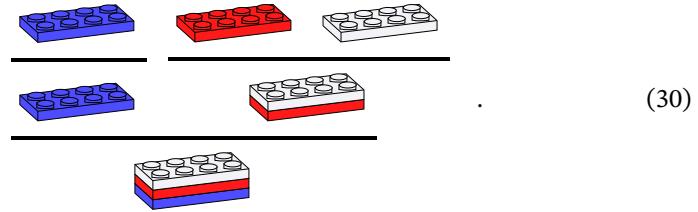
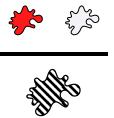
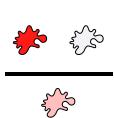
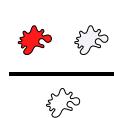
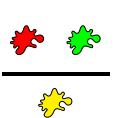
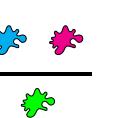
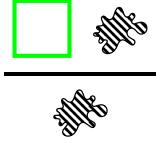
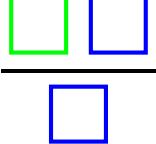
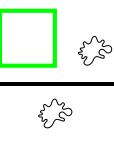
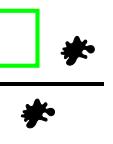
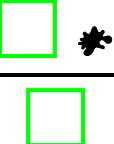
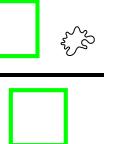


Table 2.1 shows the properties of the 4 composition rules for composing colors that we described earlier.

The table also notes the presence of a *neutral element* and an *annihilating element*. A neutral element, is an element which, when composed with another element, does not change the original color. Using the additive composition rule, for instance, this element is *black*. On the other hand, when considering the lego purists composition rule no neutral element can be found (indeed, composing with any element will result in a color change).

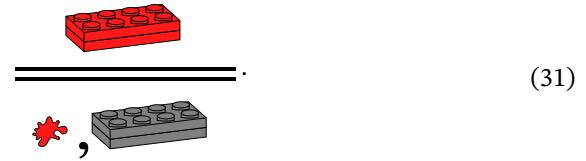
Table 2.1.: Properties of color composition rules

	Lego purist	Mixing paint	Apply paint	Additive	Subtractive
					
Commutative?	✓	✓	✗	✓	✓
Associative?	✓	✗	✓	✓	✓
Annihilative element?		✗			
Neutral element?	✗	✗	✗		

2.5. Composing recipes

We can also compose recipes themselves.

For example, imagine that in our analysis of Lego composition we decompose its color from the shape. Each element is now described by a color and a shape:

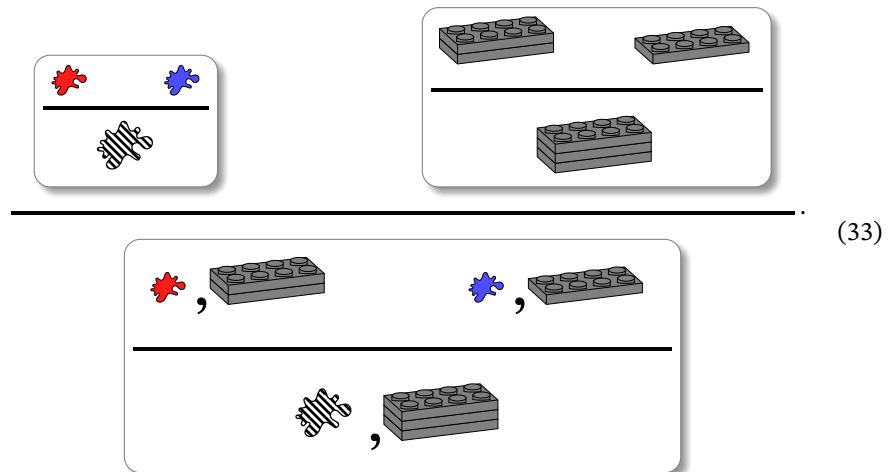


We can now define composition of color-shape pairs by composing the Lego-purist rule for colors with a color-neutral shape composition rule.

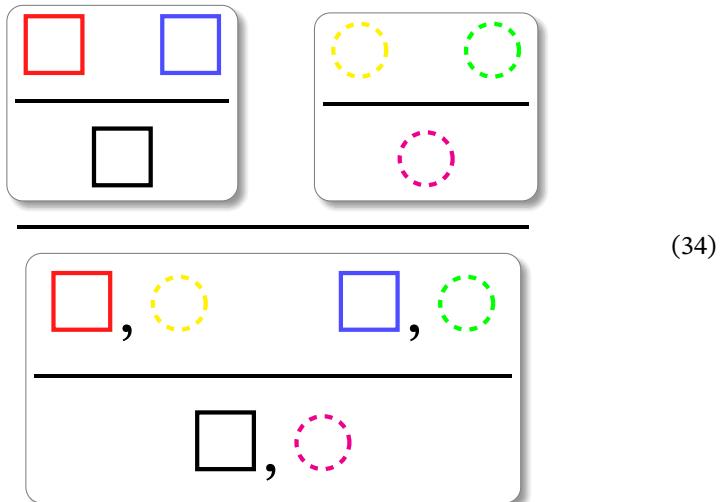
For example, given two pairs



we can find what their composition is by looking at what happens when we compose the components:



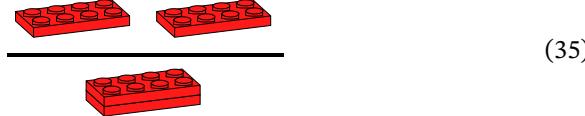
We can generalize this as follows:



2.6. Isomorphisms

Do you know the game “spot the 5 differences”? In this book we are going to play the opposite game, which is “spot how several different things are the same at some level of abstraction”.

Consider two Lego worlds in which all colors are red or all colors are white:



and



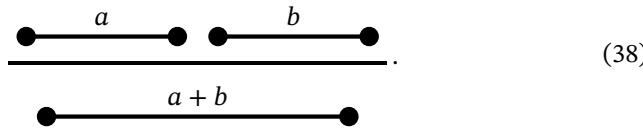
We could create Lego theories for each of the worlds. Although they would describe different worlds, the theories would be *isomorphic*.

If we confine ourselves with composing Lego blocks with the same section, then all it counts is the height of the stacks. The equations above are saying $1 + 1 = 2$:

$$\frac{1 \quad 1}{2}.$$

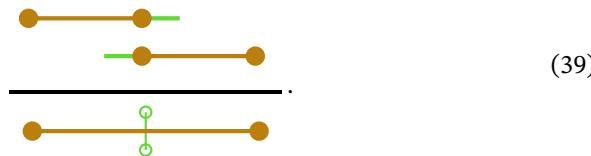
(37)

If we are dealing with addition, then there are many other things that follow the same rules. For example, we might look at composing two pieces of rope. If we have a piece of rope of length a and one of length b , you can tie them together to get a rope of length $a + b$:

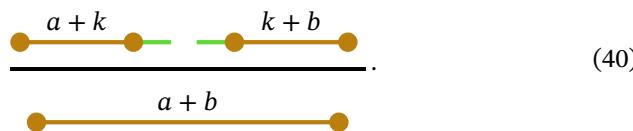


The algebra of ropes captures the algebra of bricks: the bricks are a special case because they have integer height, while ropes can be of any length.

We want to show you a rope trick. Suppose that we want to be more precise than (38) to describe the process of composing ropes, by keeping track of the extra rope that is needed to make a knot:

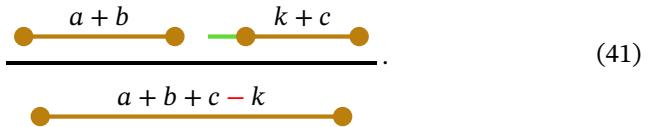


One first attempt would be to call k the extra rope for the knot, and have rules like (40): from $a + k$ and $k + b$ we obtain a piece of rope of $a + b$:

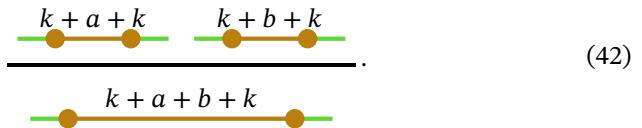


This is fine but not elegant. If you want to compose further, you need to introduce
Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

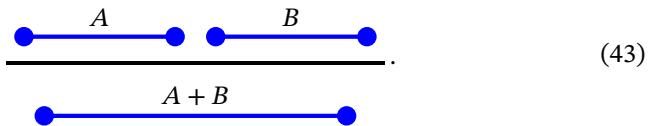
a notion of subtraction:



A more elegant way is the following: consider only ropes of the form $k + a + k$, so that we can account for the rope needed for the knots at either ends:



Now when we compose, the 2 ks on the inside elide, and we are left with 2 ks at either end, ready to be knotted with other pieces of rope. Notice that all ropes so created have the 2 extra ks . We can just remove them from the notation. We obtain new rules for ropes that take into account the knots:



And here's the magic trick: if we don't take into account the knot materials we have the simple rule (38); if we do take into account the knot materials, *for any arbitrary length k* , we obtain (43) which is exactly the same as (38).

Exercise1. Explain the trick: Where did the extra material go?

See solution on page 23.

Solutions to selected exercises

Solution of Exercise 1. Note that (38) describes composition for ropes of any arbitrary size, while (43) describes compositions for ropes whose length can be written as $k + a + k$, hence with a minimum size of $2k$. Therefore, the rope



in the first theory describes a physical rope of length 1, while the rope



in the second theory describes a rope of length $1 + 2k$. They are different theories that happen to have isomorphic rules. Note that the rope $k + k$ acts just like the identity 0. If you connect $k + a + k$ to $k + k$, you obtain $k + a + k$, for all values of a .

PART B. SETS, FUNCTIONS, RELATIONS



3. Sets and functions

27

4. Relations

61

Augusta Raurica is a Roman archaeological site in Switzerland, located on the south bank of the Rhine river (near Basel). It was founded by Lucius Munatius Plancus around 44 B.C.



3. Sets and functions

Sets and functions are fundamental notions in mathematics. In this chapter we give an informal treatment of those set-theoretic notions which are important for the purposes of this book, while avoiding more complicated aspects. The material in this section should be mostly familiar to anyone with some training in engineering, computer science, a natural science, mathematics, *etc.* We suggest nonetheless reading through: we set some conventions, and some contents may be new or rendered from a new perspective.

3.1 Logical preliminaries	28
3.2 Sets	31
3.3 New sets from old	36
3.4 Functions	42
3.5 The categorical perspective	49
3.6 Arithmetic with sets	51

The *Alphorn* is a “labrophone”, consisting of a straight multi-meter-long wooden natural horn and a wooden cup-shaped mouthpiece. It is used by mountain dwellers in Swiss alps as musical instruments and communication tools.

3.1. Logical preliminaries

We assume the reader to have some familiarity with elementary logical concepts and notation, in the way that they are typically used for reasoning and writing proofs in undergraduate mathematics. We recall some basic notions, giving a “naive” treatment intended as a rudimentary foundation and to fix our notation. A more formal treatment is outside the scope of this text.

Deduction

The building blocks for reasoning mathematically are logical statements (sometimes called propositions, assertions, logical formulas, *etc.*) which, in principle, may be evaluated to be either true or false, depending on the situation and the assumptions made. In particular, a statement might depend on variables, and the truth or falsity of the statement might vary depending on how these variables are evaluated.

To make mathematical proofs, we start with some assumptions (statements which we take, for the sake of our argument, to be true), and then we apply rules of reasoning, often called *inference rules*, which allow us to deduce new statements from these. These new statements are the “conclusions”. This process is often iterated many times in order to arrive at a statement that we seek to prove.

If we can infer a statement Q when given a statement P , we write

$$\frac{P}{Q}. \quad (1)$$

Other common ways of phrasing this are ‘ P implies Q ’ or ‘ Q follows from P ’, and another notation for this, often used in logic, is $P \vdash Q$.

If Q can be inferred from statements P_1, \dots, P_n , then we write

$$\frac{P_1 \quad P_2 \quad \dots \quad P_n}{Q}. \quad (2)$$

This notation also allows for combining multiple steps of inference, leading to a “proof tree” such as

$$\begin{array}{c} P_1 \quad P_2 \quad P_3 \\ \hline Q_1 \quad Q_2 \\ \hline R \end{array} . \quad (3)$$

When we write

$$\frac{P}{\equiv}, \quad (4)$$

this means that Q may be inferred from P and vice versa.

If we want to say that a statement Q is simply true – that it can be deduced from zero assumptions – then we write

$$\frac{}{Q}. \quad (5)$$

Connectives

Logical connectives are operations that allow us to construct new logical statements from given ones.

Two familiar logical connectives are “and” and “or”, usually denoted in infix notation by the symbols \wedge and \vee , respectively. We think of each of these as a function that takes two logical statements as its arguments, and returns a new logical statement. If P and Q are logical statements, then

$$P \wedge Q \tag{6}$$

is the new logical statement which is true precisely when both P and Q are true, and otherwise is false. And

$$P \vee Q \tag{7}$$

is the logical statement that is true precisely when either P or Q , or both, are true.

A logical operation that only takes one argument is *negation*: if P is a statement, its negation

$$\neg P \tag{8}$$

is the statement that is true if and only if P is false.

Furthermore, it is useful to include in our logical language the symbols

$$\top \text{ and } \perp \tag{9}$$

for “true” and “false”, respectively, which we think of as connectives taking zero arguments.

Calculus with connectives

Various rules, for example expressed using equations, relate the different logical connectives to each other and dictate how to calculate with them. For instance, rules such as

$$(P \wedge Q) \vee R = (P \vee R) \wedge (Q \vee R) \tag{10}$$

or

$$\neg(P \wedge Q) = (\neg P \vee \neg Q) \tag{11}$$

or

$$\neg(\neg P) = P. \tag{12}$$

Note that we use parentheses above to make clear in which order to evaluate a compound logical formula. Conventions about how strongly different logical operations “bind” allow us to use less parentheses. For example, logical negation is taken to bind at the strongest level, so we can write $\neg P$ instead of $\neg(P)$ in logical formulas, without introducing ambiguity.

Variables and quantifiers

Beyond connectives, our logical language also includes *variables*, as well as the so-called *quantifier* symbols \exists and \forall , read “there exists” and “for all”, respectively.

The quantifiers may be viewed as operations that have two arguments: the first argument is a variable, say x , and the second argument is a logical statement that might depend on x . The result is again a logical statement. For example, given a

statement $P(x)$ possibly depending on x ,

$$\exists x P(x) \quad (13)$$

denotes the statement “there exists an x such that the statement $P(x)$ is true”.
Similarly,

$$\forall x P(x) \quad (14)$$

is the statement “for all x , the statement $P(x)$ is true”.

We will use the notation $\exists!x P(x)$ to say “there exists precisely one x such that $P(x)$ is true”.

Implication as a connective

In addition to expressing the fact that ‘ P implies Q ’ using the notation

$$\frac{P}{Q}, \quad (15)$$

we can also express the statement ‘ P implies Q ’ using implication as a logical connective. We use the notation ‘ \Rightarrow ’ for this connective, and we think of it as a function of two variables: given statements P and Q , it spits out the new statement $P \Rightarrow Q$. It may be expressed (or defined) as

$$(P \Rightarrow Q) = (\neg P \vee Q), \quad (16)$$

depending on whether one wishes to take \Rightarrow as a primitive connective or define it as a compound connective via (16). This is a matter of convention.

The relationship between (15) and “ $P \Rightarrow Q$ ” is that (15) says “the statement $P \Rightarrow Q$ is true”; another way to say this would be to write

$$\overline{\overline{P \Rightarrow Q}}. \quad (17)$$

Another connective that is useful (and commonly used) is called *equivalence*. We use the symbol ‘ \Leftrightarrow ’ for it, and define it by

$$P \Leftrightarrow Q := (P \Rightarrow Q) \wedge (Q \Rightarrow P). \quad (18)$$

When the statement $P \Leftrightarrow Q$ is true we say ‘ P and Q are *equivalent*’ or that ‘ P is true if and only if Q is true’. Also, this is the same as saying

$$\frac{P}{\overline{\overline{Q}}}, \quad (19)$$

3.2. Sets

Intuitively speaking, sets describe “collections of things” – whether it be a collection of people, of objects, of abstract symbols, *etc.* The “things” making up a set are called the *elements* of the set.

There is also one (unique) special set that has *no* elements. It is called the *empty set*, and it is denoted by the symbol \emptyset .

Specifying a set by naming its elements

One way to specify a set is to write out the elements of the set, separated by commas, and surrounded by curled brackets. For example, the set consisting of the symbols “”, “”, “” and “” is indicated by $\{\text{grapes}, \text{apple}, \text{carrot}, \text{beer}\}$.

When we specify a set in this way, the order in which we write the elements does not matter: $\{\text{apple}, \text{carrot}, \text{grapes}, \text{beer}\}$ and $\{\text{beer}, \text{grapes}, \text{apple}, \text{carrot}\}$ and $\{\text{apple}, \text{carrot}, \text{beer}, \text{grapes}\}$ are, for instance, different ways of indicating one and the same set.

However, we do *not* allow repetitions of elements of sets: we do not consider $\{\text{beer}, \text{grapes}, \text{grapes}, \text{apple}, \text{carrot}\}$ as a valid way of indicating a set. At best, $\{\text{beer}, \text{grapes}, \text{grapes}, \text{grapes}, \text{apple}, \text{carrot}\}$ could be interpreted as specifying the set $\{\text{beer}, \text{grapes}, \text{apple}, \text{carrot}\}$. (One might be tempted to read $\{\text{beer}, \text{grapes}, \text{grapes}, \text{grapes}, \text{apple}, \text{carrot}\}$ as meaning that “” appears “three times”. This corresponds to the notion of “multiset” – which is something different from a set – but we are not considering multisets here.)

Because the elements of a set are in general not ordered in any way, we can visualize sets as “clouds” or “bags” of elements, as in Fig. 1.



Figure 1.: We represent sets as “clouds” or “bags” of non-repeating elements.

The size of a set

The usual name for the size of a set is its *cardinality*. If a set has finitely-many elements, then its size or cardinality is just the number of elements it contains. For example, the set $\{\text{apple}, \text{grapes}, \text{carrot}\}$ has cardinality equal to 3.

Sets with finite cardinality are called *finite sets*. If a set is not finite, there are different possible “infinite sizes”. We will delve into distinguishing different infinite cardinalities only later when we need to.

Our notation for the size of a set A will be $\text{card}(A)$ or $|A|$. For now, we will agree that $\text{card}(A)$ can be either zero (for the empty set), a natural number, or simply “infinite”. For sets of infinite size, there are different, distinct possible cardinalities (different infinities); we will discuss a bit later a way of making sense of this.

Set membership

When some “thing” is an element of a set, we also say that that thing is a *member* of that set, or that it *belongs* to that set.

The symbol to indicate membership is “ \in ”: for example,

$$\text{beer} \in \{\text{apple}, \text{carrot}, \text{grapes}, \text{beer}\} \quad (20)$$

is the statement that is an element of the set $\{\text{apple}, \text{carrot}, \text{grapes}, \text{beer}\}$. To indicate that something is *not* an element of a set we use the symbol “ \notin ”.

You may now be wanting to ask: what counts as a “thing” here, anyway? In many rigorous treatments of set theory (there are various different theories), the mathematical world is, roughly speaking, made up of sets.

In other words, a “thing” is a set*. We will not delve into the rabbit holes of formal

* To give a glimpse of how this can work, consider the following formal model for the natural Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

set-theory here, however one basic (and sometimes confusing) point is that sets can be elements of other sets.

For example, we might consider the set

$$\{\text{apple}, \{\text{grapes}\}, \{\text{apple}, \text{carrot}, \text{grapes}, \text{beer}\}\}, \quad (21)$$

which has three elements: namely , , and . Each of , , and is a “thing”, and these three things happen to be assembled together in the set (21). Based on our conventions (curly brackets specify sets), both and are sets (and for our purposes, we do not need to say rigorously what kind of thing might be).

From the above discussion it is hopefully now clear that, given sets **A**, **B**, and **C**, say, we can build for example a new set

$$\{\mathbf{A}, \mathbf{B}, \mathbf{C}\} \quad (22)$$

whose elements are **A**, **B**, and **C**. It is hopefully also clear for example that and are different sets (this is sometimes a confusing case!). The former is the empty set (it has no elements), while the latter is a set which has precisely one element, and that element happens to be the set .

Equality

Given two sets, we say they are equal if and only if “they have the same elements”. For example, we have seen that and and are all equal as sets, because their elements are the same. In particular, this example shows that a given set might have many names or symbolic representations.

From one perspective, the criterion for knowing when sets are equal reduces to knowing when elements (or “things”) are equal. If **A** and **B** are sets, then to check if **A** = **B**, we need to check if respective elements of **A** and **B** are equal.

From another perspective, the question of equality of sets can be expressed in terms of membership: **A** and **B** are equal if and only if the statements $x \in \mathbf{A}$ and $x \in \mathbf{B}$ are logically equivalent (x is a variable that can be instantiated with the elements of **A** or **B**):

$$\begin{array}{c} \mathbf{A} = \mathbf{B} \\ \hline\hline \\ x \in \mathbf{A} \\ \hline\hline \\ x \in \mathbf{B} \end{array} \quad (23)$$

Note that there are two levels of double lines. Because the first set of double lines is wider, we read the statement as stating an equivalence between **A** = **B** and $x \in \mathbf{A}$

$$\begin{array}{c} \hline\hline \\ x \in \mathbf{B} \end{array}$$

Subsets

Consider the set , and compare it with . Each of the elements of the first set is also an element of the second set; in such a case we say

numbers, using sets: define the number zero to be the empty set , define the number one to be the one-element set , define the number two to be the two-element set , etc.

that the first set is a *subset* of or is *included* in the second set. In symbols,

$$\{\text{apple}, \text{carrot}, \text{grapes}\} \subseteq \{\text{apple}, \text{carrot}, \text{grapes}, \text{beer}\}. \quad (24)$$

Generally, given sets \mathbf{A} and \mathbf{B} , the statement $\mathbf{A} \subseteq \mathbf{B}$ (that \mathbf{A} is a subset of \mathbf{B}) is logically equivalent to the statement

$$\frac{x \in \mathbf{A}}{x \in \mathbf{B}} \quad (25)$$

in sequent form and equivalent to the statement

$$\forall x \in \mathbf{A} : x \in \mathbf{B} \quad (26)$$

in terms of the “for all” universal quantifier.

Returning to (24), the second set is, on the other hand, *not included* in the first set, since is an element of the second set, but not the first. If we say a set \mathbf{A} is “strictly included” in another set \mathbf{B} , then we mean “included in and not equal”; the notation for this is $\mathbf{A} \subset \mathbf{B}$.

The notation for inclusion and strict inclusion of sets is analogous to the notation in the context of numbers for “less than or equals”, $x \leq y$, and “strictly less than”, $x < y$, respectively.

In general, if we do not use the adjective “strictly”, then “inclusion” means that equality is also possible. In particular, in our terminology it is true that any set \mathbf{A} is included in itself: $\mathbf{A} \subseteq \mathbf{A}$.

Inclusion and equality are related as follows: given sets \mathbf{A} and \mathbf{B} ,

$$\frac{\mathbf{A} \subseteq \mathbf{B} \quad \mathbf{B} \subseteq \mathbf{A}}{\mathbf{A} = \mathbf{B}}. \quad (27)$$

Many times, in order to prove a statement of the form $\mathbf{A} = \mathbf{B}$ it is a useful strategy to prove the two statements $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{B} \subseteq \mathbf{A}$ each.

With respect to inclusion of sets, the empty set \emptyset has (once again) some slightly tricky behavior: \emptyset is a subset of any other set. To see why this makes sense, consider the formulation (26): when \mathbf{A} is the empty set, this statement is always true, since quantifying “for all” over the empty set poses no condition at all.

Specifying a set via a logical statement

In addition to the “naming the elements” way of specifying a set, many times sets are specified with the help of a logical “statement” or “sentence” which serves as a condition which characterizes its elements.

The idea is this: we start out with some given set \mathbf{B} , and then we consider a statement $S(x)$ which depends on a variable x , which we think of as running over the elements of \mathbf{B} . We can then ask: for which elements x of \mathbf{B} is the statement $S(x)$ true? These elements form a subset of \mathbf{B} , often denoted

$$\{x \in \mathbf{B} \mid S(x)\}. \quad (28)$$

For example, let $\mathbf{B} = \{\text{apple}, \text{grapes}, \text{carrot}\}$ and consider the statement

$$S(x) = "x \text{ is a fruit"}. \quad (29)$$

Then we can form the set

$$\{x \in \mathbf{B} \mid x \text{ is a fruit}\} = \{\text{apple}, \text{grapes}\}. \quad (30)$$

There is an interesting special case of this way of constructing subsets of a set \mathbf{B} : what if, for a given statement $S(x)$, *none* of the elements $x \in \mathbf{B}$ are such that $S(x)$ is true? Then the result is the empty set.

As an example, consider the statement

$$S(x) = "x \text{ is the name of a planet}" \quad (31)$$

Then, for the set $\mathbf{B} = \{\text{apple}, \text{grapes}, \text{carrot}\}$, the set defined as

$$\{x \in \mathbf{B} \mid x \text{ is the name of a planet}\} \quad (32)$$

is equal to the empty set.

Logical statements quantified over a set

The above describes how to define a set using a logical statement. Often times we also conversely use a set to formulate a logical statement. For example a statement of the kind “there exists $x \in \mathbf{A}$, such that the statement $P(x)$ is true”. Our notation for this will be

$$\exists x \in \mathbf{A} : P(x). \quad (33)$$

Similarly,

$$\forall x \in \mathbf{A} : P(x). \quad (34)$$

denotes the statement “for all $x \in \mathbf{A}$, $P(x)$ is true”.

Remark 3.1 (Do you want to be more formal?). The statements (33) and (34) can be formulated in the formats “ $\exists x Q(x)$ ” and “ $\forall x Q(x)$ ”, respectively, that were introduced in Section 3.1 for the logical symbols \exists and \forall :

$$\exists x ((x \in \mathbf{A}) \wedge P(x)), \quad (35)$$

and

$$\forall x ((x \in \mathbf{A}) \Rightarrow P(x)). \quad (36)$$

However, (33) and (34) are easier to read and are common usage.

Some special sets we'll often use

Familiar sets of numbers For us, the set of natural numbers[†] is

$$\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}. \quad (37)$$

These can be extended to the set \mathbb{Z} of integers (or “whole numbers”)

$$\mathbb{Z} = \{0, +1, -1, +2, -2, +3, -3, \dots\}, \quad (38)$$

which in turn may be extended to the rational numbers \mathbb{Q} , the real numbers \mathbb{R} , and the complex numbers \mathbb{C} . In formal set theory, these sets can actually be quite a nuisance to define rigorously; for our purposes this is unnecessary, and we will

[†] Whether zero should be included in the definition of the natural numbers is a question of convention, and there is no clear universal agreement on this. We choose to follow the ISO 80000-2 standard [12] that includes zero as part of the natural numbers.

just work with these sets in the way you are probably used to from high-school or undergraduate mathematics.

Singleton sets In many situations in category theory, it often doesn't matter exactly which of the infinitely-many possible one-element sets we are considering, namely the essential feature that often only matters is the fact that the set has just one element. However, at the same time, it is often choose a specific, explicitly specified one-element set, in oder to be able to “operate” with it directly. For this purpose, we define here a “standard, default” one-element (singleton) set

$$\mathbf{1} := \{\bullet\} \tag{39}$$

whose sole element is the symbol “•”. Here does not really matter which symbol we have chosen as the single element of our default one-element set; we have chosen the symbol “•” simply because it is a fairly “neutral-looking” symbol and one that is not used often to denote things with another mathematical meaning (however we could have instead considered, for example, the singleton sets $\{\star\}$, or $\{\heartsuit\}$, etc.). We will only use this “default” singleton set $\mathbf{1}$ in situations where all we care about it is that it contains exactly one element, and the properties derive from this.

The set of booleans The *set of booleans* is defined as:

$$\mathbf{Bool} = \{\perp, \top\}, \tag{40}$$

where \perp is “false” and \top is “true”.

3.3. New sets from old

In this section we recall some elementary ways of constructing new sets from old. The idea of *constructing* new things from old things is one of the main themes of the book.

Union and intersection

The union of two sets is the set containing precisely those elements which come from either of the two.

Definition 3.2 (Union of sets)

Given sets \mathbf{A} and \mathbf{B} , their *union* is a new set, denoted $\mathbf{A} \cup \mathbf{B}$, characterized by

$$\frac{x \in (\mathbf{A} \cup \mathbf{B})}{x \in \mathbf{A} \vee x \in \mathbf{B}}. \quad (41)$$

For example, if $\mathbf{A} = \{\text{apple}, \text{grapes}\}$ and $\mathbf{B} = \{\text{beer}\}$, then

$$\mathbf{A} \cup \mathbf{B} = \{\text{apple}, \text{grapes}, \text{beer}\}. \quad (42)$$

The intersection of two sets is the set of elements common to both.

Definition 3.3 (Intersection of sets)

The *intersection* of sets \mathbf{A} and \mathbf{B} , denoted $\mathbf{A} \cap \mathbf{B}$, is the set characterized by

$$\frac{x \in (\mathbf{A} \cap \mathbf{B})}{x \in \mathbf{A} \wedge x \in \mathbf{B}}. \quad (43)$$

For example, if $\mathbf{A} = \{\text{apple}, \text{grapes}, \text{carrot}, \text{beer}\}$ and $\mathbf{B} = \{\text{beer}, \text{grapes}, \text{flag}\}$, then

$$\mathbf{A} \cap \mathbf{B} = \{\text{beer}, \text{grapes}\}. \quad (44)$$

Exercise 2. Prove that union and intersection of sets are associative operations.

See solution on page 79.

Exercise 3. Prove that union and intersection of sets are commutative operations.

See solution on page 79.

The definitions of union and intersection above are for two sets, \mathbf{A} and \mathbf{B} . We can also build the union or intersection of any finite number of sets, or even an infinite collection of sets. Let's look at the finite case first.

Given sets $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$, their union $\bigcup_{i \in \{1, \dots, n\}} \mathbf{A}_i$ is defined by

$$\frac{x \in \bigcup_{i \in \{1, \dots, n\}} \mathbf{A}_i}{\exists i \in \{1, \dots, n\}: x \in \mathbf{A}_i} \quad (45)$$

Alternative notations for this union are also $\bigcup_{i=1}^n \mathbf{A}_i$ and $\bigcup \{\mathbf{A}_i \mid i \in \{1, \dots, n\}\}$. The latter notation lends itself well for the generalization of the union operation to any arbitrary collection of sets. If ξ is a collection of sets (it might have two elements (say, sets \mathbf{A} and \mathbf{B}), or it might have 108 elements, or it might have

infinitely many elements) we define $\bigcup \xi$ by

$$\begin{array}{c} x \in \bigcup \xi \\ \hline \hline \exists A \in \xi : x \in A \end{array}. \quad (46)$$

This notation for the union of an arbitrary collection of sets is related to our previous definition for two sets A and B via

$$A \cup B = \bigcup \{A, B\}. \quad (47)$$

Remark 3.4. A slightly confusing thing might be the following; it has to do with how we use variables and the fact that for sets, elements can appear at most once. If we write for example “ A_1, A_2, \dots, A_n ”, then we are a priori syntactically speaking about n distinct sets. However, this notation does exclude the possibility that perhaps $A_1 = A_2 = \dots = A_n = A$, for instance. In this special concrete case, the set of sets $\{A_i \mid i \in \{1, \dots, n\}\}$ will have only one single element, namely A , even though, in terms of notation, it might look like there are more elements.

Remark 3.5. When we write “ A_1, A_2, \dots, A_n ”, we are using the numbers $1, \dots, n$ to *index* these n (no necessarily non-equal) sets. Indexing means giving them distinct names. We don’t necessarily need to use natural numbers to index a collection of sets (or any other things, for that matter) – we can use any other set as an index! The main point is that the index set (let’s call it I) should have precisely as many elements as we wish to have distinct “names” for the sets we are indexing. Let’s look at some examples.

For instance, for a collection of four sets, we might name them A_1, A_2, A_3, A_4 using the index set $I = \{1, 2, 3, 4\}$, but we also might name them $A_{\text{north}}, A_{\text{south}}, A_{\text{east}}, A_{\text{west}}$ using the index set $I = \{\text{north}, \text{south}, \text{east}, \text{west}\}$. With such small index sets, it is easy list all the sets involved; however in general the notation $\{A_i\}_{i \in I}$ is used. For example, if $I = \mathbb{Z}$ or $I = \mathbb{R}$, we denote the respective corresponding indexed collections of sets by $\{A_k\}_{k \in \mathbb{Z}}$ or $\{A_\lambda\}_{\lambda \in \mathbb{R}}$, for example. An indexed collection of sets is also sometimes called a family of sets.

Similar to how the operation of union may be generalized to any arbitrary collection of sets, so too the operation of intersection. Given a collection ξ of sets, we define the intersection $\bigcap \xi$ by

$$\begin{array}{c} x \in \bigcap \xi \\ \hline \hline \forall A \in \xi : x \in A \end{array}. \quad (48)$$

This notation for the intersection of an arbitrary collection of sets is related to our previous definition for two sets A and B via

$$A \cap B = \bigcap \{A, B\}. \quad (49)$$

Powerset

Definition 3.6 (Power set)

Given a set A , we can form a new set whose elements are precisely all the subsets of A . This new set is called the *powerset* of A ; we denote it by $\text{Pow } A$.

For example, if $A = \{\text{apple}, \text{carrot}, \text{grapes}\}$, then its powerset is

$$\text{Pow } A = \{\emptyset, \{\text{apple}\}, \{\text{carrot}\}, \{\text{grapes}\}, \{\text{apple}, \text{carrot}\}, \{\text{apple}, \text{grapes}\}, \{\text{carrot}, \text{grapes}\}, \{\text{apple}, \text{carrot}, \text{grapes}\}\}. \quad (50)$$

Exercise 4. Can you count how many elements the powerset $\text{Pow } \mathbf{A}$ has in the following cases?

1. $\mathbf{A} = \{\text{apple}\}.$
2. $\mathbf{A} = \{\text{apple}, \text{carrot}\}.$
3. $\mathbf{A} = \{\text{apple}, \text{carrot}, \text{grapes}\}.$
4. $\mathbf{A} = \emptyset.$

Can you guess a general formula for the size of the powerset of a finite set?

See solution on page 79.

Now suppose we fix a set \mathbf{A} for a moment. Given $\mathbf{S} \in \text{Pow } \mathbf{A}$, the *complement* of \mathbf{S} with respect to \mathbf{A} is

$$\mathbf{A} \setminus \mathbf{S} = \{x \in \mathbf{A} \mid x \notin \mathbf{S}\}, \quad (51)$$

which is again an element of $\text{Pow } \mathbf{A}$. In situations where it is evident which ambient set \mathbf{A} we are working with, the notation \mathbf{S}^c is sometimes used instead of $\mathbf{A} \setminus \mathbf{S}$.

We also note that the operations of union and intersection, when restricted to $\text{Pow } \mathbf{A}$, again produce elements of $\text{Pow } \mathbf{A}$. That is, if $\mathbf{S}, \mathbf{T} \in \text{Pow } \mathbf{A}$, then $\mathbf{S} \cup \mathbf{T} \in \text{Pow } \mathbf{A}$, and similarly $\mathbf{S} \cap \mathbf{T} \in \text{Pow } \mathbf{A}$.

The operations \cup , \cap , and $(-)^c$ on $\text{Pow } \mathbf{A}$ obey various rules which are useful to be familiar with. For example,

$$(\mathbf{S} \cap \mathbf{T})^c = (\mathbf{S}^c) \cup (\mathbf{T}^c). \quad (52)$$

Can you state more such rules? A useful visual aid for such calculations are so-called Venn diagrams (Fig. 2, Fig. 3, Fig. 4).

Graded exercise B.1 (DistributingSubsets)

Let \mathbf{A} be a set, and let $\mathbf{S}, \mathbf{T}, \mathbf{U} \subseteq \mathbf{A}$ be subsets. Prove that

$$\mathbf{S} \cap (\mathbf{T} \cup \mathbf{U}) = (\mathbf{S} \cap \mathbf{T}) \cup (\mathbf{S} \cap \mathbf{U}). \quad (53)$$

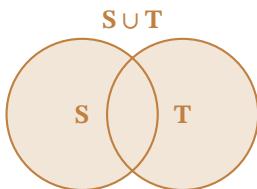


Figure 2.: Venn diagram for union operation.

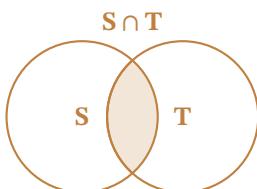


Figure 3.: Venn diagram for intersection operation.

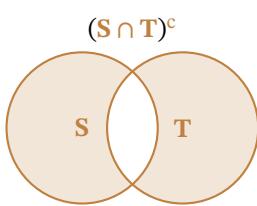


Figure 4.: Venn diagram for complement operation.

Tuples

A “tuple” is a finite sequence or listing of “things” where their order matters and repetitions are allowed. We use the following notation to denote a tuple of numbers, for example:

$$\langle 3.5, -2, 7, 68 \rangle. \quad (54)$$

Tuples always have a *length*, which can be any natural number. A tuple of length $n \in \mathbb{N}$ is called an n -tuple. There is only one possible tuple of length zero; we call it the empty tuple and denote it by $\langle \rangle$.

Tuples of length two are often called *ordered pairs* or just *pairs*; tuples of length three are called *triples*; tuples of length four are called *quadruples*, and so on.

We concatenate tuples using the symbol \circlearrowright :

$$\langle a, b \rangle \circlearrowright \langle c \rangle \circlearrowright \langle \rangle = \langle a, b, c \rangle. \quad (55)$$

The items inside the brackets that indicate a tuple will be called *entries* or *components*. If t is a tuple of length n , then $t[i]$ will refer to the i^{th} entry of t , where $1 \leq i \leq n$. So, for example, if

$$t = \langle 2, 9, -1, 3, 6 \rangle, \quad (56)$$

then $t[1] = 2$, $t[2] = 9$, $t[3] = -1$, etc.

Typically, we use tuples in situations where we also specify, for each entry of the

tuple, a set of which that entry is an element. For example, if $\mathbf{A} = \{\apple, \carrot, \grapes\}$ and $\mathbf{B} = \{\beer, \juice\}$, then sometimes we will want to specify that $\langle \apple, \juice \rangle$ is a 2-tuple where $\apple \in \mathbf{A}$ and $\juice \in \mathbf{B}$.

Lists

Our notion of *list* is similar to that of a tuple, except that we require its entries to all be elements of a single specified set or to be all things of a specified type.

We will use the notation

$$[3, 7, 8]_{\mathbb{N}} \quad (57)$$

to denote a list of natural numbers, for example, and

$$[3, 7, 8]_{\mathbb{R}} \quad (58)$$

to denote a list of real numbers. Often it will be clear which type of things we are dealing with, in which case we drop the subscript in our notation and simply write

$$[3, 7, 8]. \quad (59)$$

We concatenate lists using the symbol $\circ|$:

$$[1, 3] \circ| [] \circ| [5, 7] = [1, 3, 5, 7]. \quad (60)$$

Of course, the type of things we consider need not be numbers. For example, we might work with lists whose entries are from the set $\mathbf{A} = \{\apple, \banana, \carrot, \pretzel, \grapes\}$. Such a list is $[\grapes, \apple, \carrot, \apple]_{\mathbf{A}}$, for instance.

Similar to tuples, lists can have any length $n \in \mathbb{N}$, including zero. The entries of a list l of length n will be denoted $t[i]$, where $1 \leq i \leq n$.

In contrast to tuples, we have a different empty list (zero-length list) for every possible type of list. The empty list of things of a given type T will be denoted $[]_T$.

The set of all lists of elements of \mathbf{A} is written as $\text{List } \mathbf{A}$.

Cartesian product

Definition 3.7

(Cartesian product of two sets)

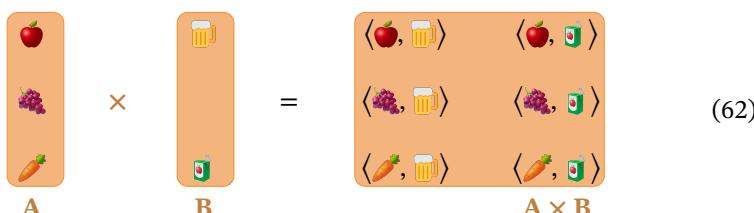
Given sets \mathbf{A} and \mathbf{B} , their *cartesian product* is a new set – denoted $\mathbf{A} \times \mathbf{B}$ – whose elements are precisely all possible 2-tuples $\langle x, y \rangle$ such that the first entry x is an element of \mathbf{A} and the second entry y is an element of \mathbf{B} .

For example, if $\mathbf{A} = \{\apple, \grapes, \carrot\}$ and $\mathbf{B} = \{\beer, \juice\}$, then

$$\mathbf{A} \times \mathbf{B} = \{\langle \apple, \beer \rangle, \langle \apple, \juice \rangle, \langle \grapes, \beer \rangle, \langle \grapes, \juice \rangle, \langle \carrot, \beer \rangle, \langle \carrot, \juice \rangle\}. \quad (61)$$

In the special case where $\mathbf{A} = \emptyset$ or $\mathbf{B} = \emptyset$, then $\mathbf{A} \times \mathbf{B} = \emptyset$.

Another way to represent the cartesian product is the following:



Remark 3.8. For finite sets \mathbf{A} and \mathbf{B} , the size of $\mathbf{A} \times \mathbf{B}$ is the product (multiplication) of the sizes of \mathbf{A} and \mathbf{B} :

$$\text{card}(\mathbf{A} \times \mathbf{B}) = \text{card}(\mathbf{A}) \cdot \text{card}(\mathbf{B}). \quad (63)$$

This is one reason why we think of $\mathbf{A} \times \mathbf{B}$ as a kind of multiplication of sets.

Remark 3.9 (Do you want to be more formal?). In formal set theory, 2-tuples (ordered pairs) are often defined by setting

$$\langle x, y \rangle := \{\{x\}, \{x, y\}\}. \quad (64)$$

In this case, the cartesian product is

$$\mathbf{A} \times \mathbf{B} = \{z \in \text{Pow Pow}(\mathbf{A} \cup \mathbf{B}) \mid \exists x \in \mathbf{A}, \exists y \in \mathbf{B} : z = \langle x, y \rangle\}. \quad (65)$$

We will, however, treat tuples as a primitive construction (*i.e.*, without reference to formal set theory), and hence for us also the construction of “cartesian product of sets” is primitive.

The definition Def. 3.7 is for a *binary* operation of cartesian product: one where there are two factors. We can make an analogous “n-ary” definition for any finite number of factors.

Definition 3.10 (n-ary Cartesian product of sets)

Let $n \in \mathbb{N}$. Given sets $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$, their *cartesian product* is the set $\mathbf{A}_1 \times \mathbf{A}_2 \times \dots \times \mathbf{A}_n$ whose elements are precisely all possible n-tuples $\langle x_1, x_2, \dots, x_n \rangle$ such that each entry x_i is an element of \mathbf{A}_i , for each $i \in \{1, \dots, n\}$.

Disjoint union

Definition 3.11 (Disjoint union of two sets)

Given sets \mathbf{A} and \mathbf{B} , their disjoint union, or sum, is the set

$$\mathbf{A} + \mathbf{B} := (\{1\} \times \mathbf{A}) \cup (\{2\} \times \mathbf{B}). \quad (66)$$

In other words, an element of $\mathbf{A} + \mathbf{B}$ is either a tuple $\langle 1, x \rangle$ for some $x \in \mathbf{A}$ or a tuple $\langle 2, y \rangle$ for some $y \in \mathbf{B}$. The sets $\{1\}$ and $\{2\}$ here simply provide labels which “force” the sets $\{1\} \times \mathbf{A}$ and $\{2\} \times \mathbf{B}$ to be disjoint (even if \mathbf{A} and \mathbf{B} have elements in common).

Consider the sets $\mathbf{A} = \{\text{🥨}, \text{🍺}\}$ and $\mathbf{B} = \{\text{🥩}, \text{🥛}\}$. Their disjoint union can be represented as:

$$\begin{array}{ccc} \text{A} & + & \text{B} \\ & = & \end{array} \quad \begin{array}{c} \langle 1, \text{🥨} \rangle \quad \langle 2, \text{🥩} \rangle \\ \langle 1, \text{🍺} \rangle \quad \langle 2, \text{🥛} \rangle \end{array} \quad \mathbf{A} + \mathbf{B} \quad (67)$$

We can define the disjoint union of a set with itself; this corresponds to having

two distinct copies of the set:

$$\begin{array}{ccc}
 \begin{array}{c} \text{QQ} \\ \text{A} \end{array} & + & \begin{array}{c} \text{QQ} \\ \text{A} \end{array} = \begin{array}{cc} \langle 1, \text{QQ} \rangle & \langle 2, \text{QQ} \rangle \\ \langle 1, \text{beer} \rangle & \langle 2, \text{beer} \rangle \end{array} \\
 & & \text{A} + \text{A}
 \end{array} \tag{68}$$

Let us also think about what happens when the empty set is at play. For example, if $\mathbf{A} = \emptyset$, then $\emptyset + \mathbf{B} = \{2\} \times \mathbf{B}$, and similarly $\mathbf{A} + \emptyset = \{\langle 1, x \rangle \mid x \in \mathbf{A}\}$. Also, $\emptyset + \emptyset = \emptyset$.

Remark 3.12. If \mathbf{A} and \mathbf{B} are finite, then the size of $\mathbf{A} + \mathbf{B}$ is the sum of the sizes of \mathbf{A} and \mathbf{B} :

$$\text{card}(\mathbf{A} + \mathbf{B}) = \text{card}(\mathbf{A}) + \text{card}(\mathbf{B}). \tag{69}$$

This is a reason why we think of $\mathbf{A} + \mathbf{B}$ as a form of addition of sets.

Remark 3.13. Later we will see that the cartesian product of sets is a special case of a very general construction in category theory, called the categorical product, and that the disjoint union, or sum, of sets is a special case of a “dual” construction, called categorical coproduct.

Analogous to the n-ary cartesian product of any finite number of sets, we can define an n-ary version of disjoint union.

Definition 3.14 (n-ary disjoint union of sets)

Let $n \in \mathbb{N}$. Given sets $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$, their disjoint union, or sum, is the set

$$\mathbf{A}_1 + \mathbf{A}_2 + \dots + \mathbf{A}_n := (\{1\} \times \mathbf{A}_1) \cup (\{2\} \times \mathbf{A}_2) \cup \dots \cup (\{n\} \times \mathbf{A}_n) \tag{70}$$

3.4. Functions

Functions are familiar to all of us – to some degree – from our school days. In this section we present some basic terminology and our way of defining functions formally, and we discuss various ways to *think* about functions.

When speaking about functions, we use the words “map” or “mapping” as synonyms.

Specifying functions

x	$f(x)$
1	1
2	4
3	9
...	...

Figure 5.: A function described via a table.

Functions can be specified in a variety of ways.

Sometimes they are indicated with the help of a formula, such as $f(x) = x^2$. Or with the help of a table, as in Fig. 5.

Other times a function might be characterized by equations or properties. For example, the exponential function

$$\begin{aligned} \exp : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto e^x, \end{aligned} \tag{71}$$

is known to be characterized by the fact that it satisfies the differential equation $f'(x) = f(x)$ and the initial condition $f(0) = 1$.

Still other times one might be able to prove the existence (and perhaps also uniqueness) of some function satisfying some given properties, but one might not have any concrete means to “evaluate” or “calculate” that function.

Whatever the route may be by which a function is specified, for us an essential non-negotiable part of specifying a function is to say which set is its *source* (or *domain*), and which set is its *target* (or *co-domain*). That is, a function f is always something that goes from one set \mathbf{A} (the source of f) to another set \mathbf{B} (the target of f). We write this as $f : \mathbf{A} \rightarrow \mathbf{B}$.

For example, the formula “ $f(x) = x^2$ ” does not specify a function yet, because we did not yet say what source and target set we are considering. If we are thinking of x^2 as defining a function $\mathbb{R} \rightarrow \mathbb{R}$, then this is one function, and if we are thinking of x^2 as defining a function $\mathbb{R}_{>0} \rightarrow \mathbb{R}$, then this is another function. And if we are thinking of x^2 as a function $\mathbb{N} \rightarrow \mathbb{N}$, this is yet another.

One way to see why this is important: the function

$$\begin{aligned} f : \mathbb{R}_{>0} &\rightarrow \mathbb{R}, \\ x &\mapsto x^2, \end{aligned} \tag{72}$$

is monotone (increasing), while

$$\begin{aligned} g : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto x^2, \end{aligned} \tag{73}$$

is not. They have different properties.

Or, as another example, the function g can be shown to be a continuous function, while for the function

$$\begin{aligned} h : \mathbb{N} &\rightarrow \mathbb{N}, \\ x &\mapsto x^2, \end{aligned} \tag{74}$$

it is perhaps not immediately clear what the question of continuity even means.

Functions as deterministic machines

One typical way of thinking about functions is in terms of “input” and “output”. Given a function $f : \mathbf{A} \rightarrow \mathbf{B}$ we sometimes speak of “plugging in” an element $x \in \mathbf{A}$ into the function f , and then it will “output” an element $f(x) \in \mathbf{B}$.

One reason for this kind of thinking is that sometimes functions describe things that are like a computational process or a machine: for instance, we might give a software program an input, it might then perform a series of computations, and then output an answer, and all of this might be described by a function.

Another reason for thinking of functions in terms of input and output is because humans often use functions – *as mathematical entities* – in a deterministic “machine” kind of way. Starting with some element $x \in \mathbf{A}$, we can sometimes use the function f to *calculate* or otherwise *determine* what the “output” $f(x) \in \mathbf{B}$ is. For example, if we consider the function

$$\begin{aligned} f : \mathbb{Z} &\rightarrow \mathbb{Z}, \\ x &\mapsto x^2, \end{aligned} \tag{75}$$

then, given any input $x \in \mathbb{Z}$, we can compute the output $f(x) \in \mathbb{Z}$ by multiplying the input x with itself.

Mathematically speaking, functions are *deterministic* in the sense that for any input x , there is *exactly one* output $f(x)$. This is in contrast to the fact that a given output $f(x)$ might arise from various possible inputs: for example $4 \in \mathbb{Z}$ could be the output of (75) for the input $2 \in \mathbb{Z}$ or for the input $-2 \in \mathbb{Z}$.

Functions as relations

Another point of view is that a function $f : \mathbf{A} \rightarrow \mathbf{B}$ defines a certain kind of *relation* between the elements of \mathbf{A} and \mathbf{B} . Given an $x \in \mathbf{A}$, the function f tells us that this x is related to a certain $y \in \mathbf{B}$, which we happen to call $f(x)$. This point of view is fully compatible with thinking of functions as “mathematically deterministic”. However, it is more general than interpreting functions as describing processes which are “physically deterministic” in any sense or where “the input precedes the output”.

As an illustration, consider a large phone book (of personal mobile numbers), which is just a table of names and phone numbers. Let \mathbf{A} be the set of phone numbers in the book, and \mathbf{B} the set of names. There is a function $f : \mathbf{A} \rightarrow \mathbf{B}$ which, given any phone number in \mathbf{A} , will output the name of the person to whom that number is registered. Normally, every number is assigned to a single name, so a name as an “output” of the function is completely determined (mathematically speaking) by the number one “inputs”. However, there is no “physical determinism” here: there is no non-mathematical process by which the name of the person was “computed” or “causally determined” by the phone number. Rather, the function we described arises simply from a table of information.

A formal definition

The idea is that we can formally define a function $f : \mathbf{A} \rightarrow \mathbf{B}$ by the ordered pairs $\langle x, y \rangle \in \mathbf{A} \times \mathbf{B}$ which are the elements of what might be called the “graph” of the function. In other words, those ordered pairs of the form “ $\langle x, f(x) \rangle$ ”.

Definition 3.15 (Function)

Let \mathbf{A} and \mathbf{B} be sets. A *function* $f : \mathbf{A} \rightarrow \mathbf{B}$ is a subset

$$f \subseteq \mathbf{A} \times \mathbf{B}, \quad (76)$$

with the following property:

$$\forall x \in \mathbf{A} \quad \exists! y \in \mathbf{B} : \langle x, y \rangle \in f. \quad (77)$$

Here \mathbf{A} is the source (or domain) and \mathbf{B} is the target (or co-domain) of f .

We emphasize, once again, that the source and target of a function are “baked in” as *part of the definition* of the function.

The property (77) describes the “mathematical determinism” that functions are supposed to have: for any $x \in \mathbf{A}$ there exists *exactly one* element $y \in \mathbf{B}$ that is “the result” of the function f applied to x .

Another important aspect of (77) is that it says that *for every* $x \in \mathbf{A}$ there exists a $y \in \mathbf{B}$ that is related to x by f . In other words, we do not allow functions to be “partially defined”. For example, the formula “ $f(x) = 1/x$ ” could be used to define a function $\mathbb{R} \setminus \{0\} \rightarrow \mathbb{R}$, but it would *not* be valid for defining a function $\mathbb{R} \rightarrow \mathbb{R}$.

Although we take Def. 3.15 as our *formal* definition of functions, we will continue to use the standard kinds of notation for functions, for example usually writing $y = f(x)$ and not $\langle x, y \rangle \in f$. The formal definition above is useful to keep in the back of our minds though. For instance, when thinking about situations involving the empty set.

To and from the empty set

Do there exist functions $\emptyset \rightarrow \mathbf{B}$ for any set \mathbf{B} ? What about $\mathbf{A} \rightarrow \emptyset$?

Consulting Def. 3.15, we can figure out that there is *always* a function $\emptyset \rightarrow \mathbf{B}$ (no matter what set \mathbf{B} is) because the condition “ $\forall x \in \mathbf{A} \dots$ ” in (77) is trivially satisfied, as we are quantifying over $\mathbf{A} = \emptyset$. In this case, $f \subseteq \mathbf{A} \times \mathbf{B}$ corresponds to $\emptyset \subseteq \emptyset \times \mathbf{B} = \emptyset$.

On the other hand, if $\mathbf{A} \neq \emptyset$, there are *no* functions of the type $\mathbf{A} \rightarrow \emptyset$, because the part “ $\exists! y \in \mathbf{B}$ ” of (77) cannot be satisfied, since here $\mathbf{B} = \emptyset$.

Injective, surjective, bijective functions

Even if we don’t know a lot of the specifics of some functions, there is a lot we can still say about how functions between sets can behave *in general*. In the following we review a number of basic observations and properties.

Let $f : \mathbf{A} \rightarrow \mathbf{B}$ be a function.

Definition 3.16 (Injective function)

A f is said to be *injective* if for all $x_1, x_2 \in \mathbf{A}$

$$\frac{f(x_1) = f(x_2)}{x_1 = x_2}, \quad (78)$$

Definition 3.17 (Surjective function)

and f is called *surjective* if the condition

$$\forall y \in \mathbf{B} \exists x \in \mathbf{A} : f(x) = y \quad (79)$$

holds.

Definition 3.18 (Bijective function)

A function which is both injective and surjective is called *bijective*.

Exercise 5. For the following functions, determine whether they are injective, surjective, or bijective, and explain why:

1.

$$\begin{aligned} f : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto x + 10. \end{aligned}$$

2.

$$\begin{aligned} g : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto x^2. \end{aligned}$$

3.

$$\begin{aligned} h : \mathbb{N} &\rightarrow \mathbb{N}, \\ x &\mapsto x^2. \end{aligned}$$

4.

$$\begin{aligned} k : \mathbb{N} &\rightarrow \mathbb{N}, \\ x &\mapsto 3x. \end{aligned}$$

See solution on page 79.

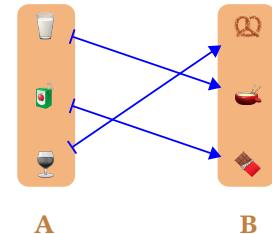


Figure 6.: An injective function.

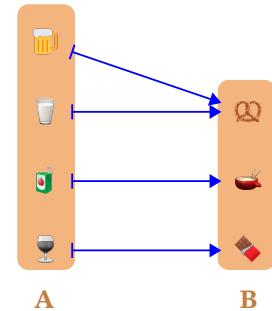


Figure 7.: A surjective function.

Image, preimage, restriction

The *image* of $f : A \rightarrow B$ is the set

$$f(A) := \{y \in B \mid \exists x \in A : f(x) = y\}. \quad (80)$$

More generally, given a subset $S \subseteq A$, its image under f is

$$f(S) := \{y \in B \mid \exists x \in S : f(x) = y\}. \quad (81)$$

From the data of $f : A \rightarrow B$ and $S \subseteq A$ we can define a new function, the *restriction of f to S* ,

$$f|_S : S \rightarrow B, \quad (82)$$

defined by $f|_S(x) = f(x)$ for all $x \in S$.

Given a subset $T \subseteq B$ of the target of $f : A \rightarrow B$, its *preimage* under f is

$$f^{-1}(T) := \{x \in A \mid f(x) \in T\}. \quad (83)$$

An alternative way of phrasing injectivity of f is to say that for every singleton subset $\{y\} \subseteq B$, its preimage under f is either a singleton set or the empty set. Surjectivity of f is equivalent to saying that $f(A) = B$.

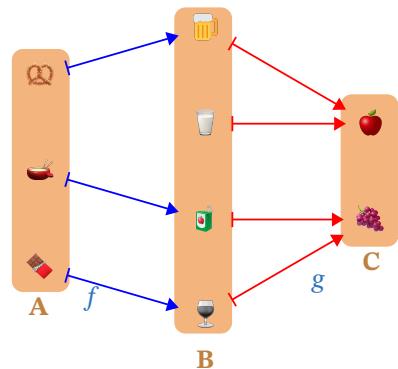


Figure 8.

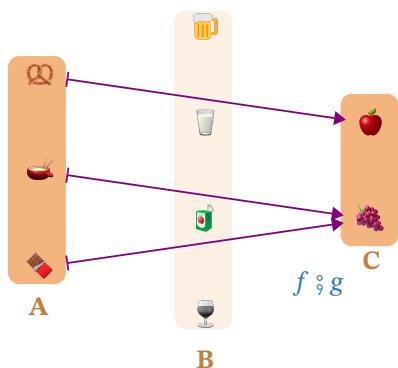


Figure 9.

Given functions $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{B} \rightarrow \mathbf{C}$, we denote their *composition* by

$$\begin{aligned} f \circ g : \mathbf{A} &\rightarrow \mathbf{C}, \\ x &\mapsto g(f(x)). \end{aligned} \tag{84}$$

The notation “ $f \circ g$ ” is different from the more traditional notation “ $g \circ f$ ”. We speak pronounce it as “ f then g ”, which aligns with the fact that, to evaluate the composition $f \circ g$ at an element $x \in \mathbf{A}$, we first apply f to compute $f(x)$, and *then* we apply g to compute the result.

Identity functions

For every set \mathbf{B} there is a special function $\text{id}_{\mathbf{B}} : \mathbf{B} \rightarrow \mathbf{B}$ which “does nothing”.

Definition 3.20 (Identity function on a set)

The *identity function* on a set \mathbf{B} is given by

$$\begin{aligned} \text{id}_{\mathbf{B}} : \mathbf{B} &\rightarrow \mathbf{B}, \\ y &\mapsto y. \end{aligned} \tag{85}$$

Because such a function “does nothing”, it behaves neutrally with respect to the composition of functions: given $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{B} \rightarrow \mathbf{C}$, we have $f \circ \text{id}_{\mathbf{B}} = f$ and $\text{id}_{\mathbf{B}} \circ g = g$.

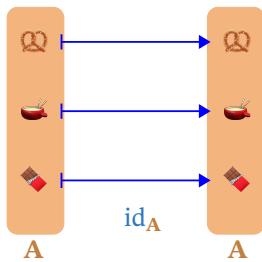


Figure 10.: An identity function

Isomorphisms

Identity functions are used, for example, to say when a function is *invertible* or, synonymously, that it is an *isomorphism*.

Definition 3.21 (Isomorphism)

A function $f : \mathbf{A} \rightarrow \mathbf{B}$ is an isomorphism if there exists an *inverse* to f : a function $g : \mathbf{B} \rightarrow \mathbf{A}$ such that

$$f \circ g = \text{id}_{\mathbf{B}} \quad \text{and} \quad g \circ f = \text{id}_{\mathbf{A}}. \tag{86}$$

Exercise 6. Show that an inverse to f is necessarily unique (so we can speak of “the” inverse).

See solution on page 80.

Remark 3.22. Note that if g is the inverse to f , then also f is the inverse of g .

Exercise 7. Show that a function is an isomorphism if and only if it is bijective.

See solution on page 80.

Definition 3.23 (Isomorphisms of sets)

Given sets \mathbf{A} and \mathbf{B} , we say that they are *isomorphic*, and write $\mathbf{A} \simeq \mathbf{B}$, if there exists an isomorphism $\mathbf{A} \rightarrow \mathbf{B}$ (or $\mathbf{B} \rightarrow \mathbf{A}$).

For a finite set \mathbf{A} , we say it has size $n \in \mathbb{N}$ if there exists an isomorphism between \mathbf{A} and the set $\{1, 2, \dots, n - 1, n\}$.

Graded exercise B.2 (CountingIsos)

Let $\mathbf{A} = \{\star, \circ\}$, $\mathbf{B} = \{1, 2, 3\}$, and $\mathbf{C} = \{a, b, c\}$.

1. How many isomorphisms are there $\mathbf{A} \rightarrow \mathbf{B}$?
2. How many isomorphisms are there $\mathbf{B} \rightarrow \mathbf{C}$?

Sets of functions

So far we have mostly been thinking of functions as a way to relate one set to another. However, in our formal definition, a function $f : \mathbf{A} \rightarrow \mathbf{B}$ is a certain kind of element in $\text{Pow}(\mathbf{A} \times \mathbf{B})$. Consider all those elements of $\text{Pow}(\mathbf{A} \times \mathbf{B})$ which are indeed functions: they form a set, the set of all functions from \mathbf{A} to \mathbf{B} . A notation we will use for this set is $\mathbf{B}^{\mathbf{A}}$, or sometimes also $(\mathbf{A} \rightarrow \mathbf{B})$.

Why the exponent notation?

If \mathbf{A} and \mathbf{B} are finite sets, then $\mathbf{B}^{\mathbf{A}}$ is also a finite set and its size is the size of \mathbf{B} to the power of the size of \mathbf{A} :

$$\text{card}(\mathbf{B}^{\mathbf{A}}) = \text{card}(\mathbf{B})^{\text{card}(\mathbf{A})}. \quad (87)$$

Product, sum, and exponentiation of functions

The following concepts of *product*, *sum*, and *exponentiation* of functions go hand-in-hand with notions of product, sum, and exponentiation of sets.

Definition 3.24 (Product of functions)

Given functions $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{C} \rightarrow \mathbf{D}$, their *product* is the function

$$\begin{aligned} f \times g : \mathbf{A} \times \mathbf{C} &\rightarrow \mathbf{B} \times \mathbf{D}, \\ \langle a, c \rangle &\mapsto \langle f(a), g(c) \rangle. \end{aligned} \quad (88)$$

Example 3.25. Consider $f, g : \mathbb{R} \rightarrow \mathbb{R}$ with $f(a) = a^2$ and $g(b) = b + 1$. We have $(f \times g)(2, 5) = \langle 2^2, 5 + 1 \rangle = \langle 4, 6 \rangle$.

Definition 3.26 (Sum of functions)

Given functions $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{C} \rightarrow \mathbf{D}$, their *sum* is the function

$$\begin{aligned} f + g : \mathbf{A} + \mathbf{C} &\rightarrow \mathbf{B} + \mathbf{D} \\ \langle 1, a \rangle &\mapsto \langle 1, f(a) \rangle, \\ \langle 2, c \rangle &\mapsto \langle 2, g(c) \rangle. \end{aligned} \quad (89)$$

Example 3.27. Consider $f : \mathbb{Z} \rightarrow \mathbb{N}$ with $f(a) = a^2$ and $g : \mathbb{R} \rightarrow \mathbb{R}$ with $g(b) = b^3$. We have $(f + g)(\langle 1, 2 \rangle) = \langle 1, f(2) \rangle = \langle 1, 2^2 \rangle = \langle 1, 4 \rangle$, and $(f + g)(\langle 2, 3 \rangle) = \langle 2, g(3) \rangle = \langle 2, 3^3 \rangle = \langle 2, 27 \rangle$.

Definition 3.28 (Exponentiation of functions)

Given a function $f : \mathbf{A} \rightarrow \mathbf{B}$ and a set \mathbf{C} , the *exponentiation* of f , with base \mathbf{C} , is the function

$$\begin{aligned} \mathbf{C}^f : \mathbf{C}^{\mathbf{B}} &\rightarrow \mathbf{C}^{\mathbf{A}} \\ \varphi &\mapsto f \circ \varphi. \end{aligned} \quad (90)$$

Remark 3.29. Note that under exponentiation, the order in which \mathbf{A} and \mathbf{B} appear becomes “switched”.

Unordered cartesian products and disjoint unions

Given any finite number of sets $\mathbf{A}_1, \dots, \mathbf{A}_n$, we have defined their cartesian product as

$$\mathbf{A}_1 \times \mathbf{A}_2 \times \dots \times \mathbf{A}_n = \{\langle x_1, x_2, \dots, x_n \rangle \mid x_i \in \mathbf{A}_i \text{ for each } i = 1, \dots, n\}. \quad (91)$$

This defines an "ordered" cartesian product because tuples arrange elements in an ordered sequence: the *first* entry is associated with the index "1", the *second* entry with the index "2", and so on, all the way up to n . With this definition, the cartesian product $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$ is not, for example, equal to the cartesian product $\mathbf{C} \times \mathbf{B} \times \mathbf{A}$. The order of the factors matters.

There is, however, an unordered version of the cartesian product. To explain how it works, we start with the observation that we can think of a tuple

$$\langle x_1, x_2, \dots, x_n \rangle \quad (92)$$

(with $x_i \in \mathbf{A}_i$ for each $i = 1, \dots, n$) as encoding (and as encoded by) a function

$$f : \{1, \dots, n\} \rightarrow \bigcup_{i \in I} \mathbf{A}_i \quad (93)$$

such that $f(i) \in \mathbf{A}_i$ for each $i \in \{1, \dots, n\}$. Namely, $f(i) = x_i$, for each $i \in \{1, \dots, n\}$. In this description of the tuple via a function, the "ordering" is encoded via the fact that the elements of $\{1, \dots, n\}$ are ordinal numbers.

We can however, also consider any family of sets $\{\mathbf{A}_i\}_{i \in I}$ for an *arbitrary* set I which serves as an index set (previously we were only using the index set $\mathbf{I} = \{1, \dots, n\}$). In this case, we define the unordered cartesian product of the family of sets $\{\mathbf{A}_i\}_{i \in I}$ by

$$\prod_{i \in I} \mathbf{A}_i := \{f : I \rightarrow \bigcup_{i \in I} \mathbf{A}_i \mid f(i) \in \mathbf{A}_i \forall i \in I\}. \quad (94)$$

We use the symbol \prod instead of the infix notation " \times " (which necessarily requires an ordering). The upper case greek letter pi \prod is meant to stand for "product". Note that this unordered definition does not make any assumptions on the size of I ; it can also be infinite.

There is also an unordered version of the disjoint union, or sum, of sets. Given a family $\{\mathbf{A}_i\}_{i \in I}$, with arbitrary index set I , we define the unordered sum as

$$\sum_{i \in I} \mathbf{A}_i := \bigcup_{i \in I} \{i\} \times \mathbf{A}_i. \quad (95)$$

Also here there is no restriction on the size of I . Note that for the unordered version of the sum of sets we have not needed to switch to using functions – the union of sets is already an "unordered" operation. In the definition of $\mathbf{A}_1 + \dots + \mathbf{A}_n$, the ordering came solely from the ordinals in the index set $\{1, \dots, n\}$.

3.5. The categorical perspective

A central theme in this book is to study how various mathematical structures compose. For instance, what are general patterns of how functions and sets *relate to one another* via composition?

In time, we will see that many features that sets and functions exhibit can be broadly generalized to other kinds of mathematical entities.

Structuralism

One guiding philosophy for creating and understanding such generalizations is to formulate properties of functions in a way that only uses their “external compositional aspects” and does not rely on the fact that we are dealing with sets, which have elements, and so we can “look inside them”. We call this philosophy “structuralism” because

This likely sounds very vague at the moment. Let us illustrate with some examples.

Example 3.30. Consider the property that a function may (or may not) have of being bijective. According to Exercise 7, a function $f : \mathbf{A} \rightarrow \mathbf{B}$ is bijective if and only if it is an isomorphism. The latter means, by definition, that there exists $g : \mathbf{B} \rightarrow \mathbf{A}$ such that $f ; g = \text{id}_{\mathbf{A}}$ and $g ; f = \text{id}_{\mathbf{B}}$. The point is that the equations in the definition of “isomorphism” only make use of the operation of function composition, the notion of quality, and the existence of special identity functions. There is no mention of elements of sets, as there is in the definition of “bijective”.

Example 3.31. The notion of “subset” is traditionally defined, as we did above, by saying that $\mathbf{A} \subseteq \mathbf{B}$ if and only if $\forall x \in \mathbf{A}$:

$$\frac{x \in \mathbf{A}}{x \in \mathbf{B}}. \quad (96)$$

There are, however, alternatives that do not refer to “elements”. To see one way, consider the set $\mathbf{2} = \{0, 1\}$. Any function $f : \mathbf{B} \rightarrow \mathbf{2}$ defines a subset

$$\mathbf{A}_f = \{y \in \mathbf{B} \mid f(y) = 1\} \subseteq \mathbf{B}. \quad (97)$$

Conversely, any subset $\mathbf{A} \subseteq \mathbf{B}$ defines a function $f_{\mathbf{A}} : \mathbf{B} \rightarrow \mathbf{2}$ by setting

$$f_{\mathbf{A}}(y) = \begin{cases} 1 & \text{if } y \in \mathbf{A}, \\ 0 & \text{elsewhere.} \end{cases} \quad (98)$$

It can be checked that this defines a 1-to-1 correspondence between functions $\mathbf{B} \rightarrow \mathbf{2}$ and subsets of \mathbf{B} . In other words, there is a bijection between the set $\mathbf{2}^{\mathbf{B}}$ and the set $\text{Pow } \mathbf{B}$. So, instead of using a definition of subset that involves elements of sets, we could work with functions $\mathbf{B} \rightarrow \mathbf{2}$.

Graded exercise B.3 (SubsetsAsFunctions)

Let \mathbf{B} be any set. Prove that $\mathbf{2}^{\mathbf{B}} \simeq \text{Pow } \mathbf{B}$.

Example 3.32. Even the notions of “element of a set” and “evaluation of a function at an element” can be described purely in terms of functions and their composition, without needing to “look inside” of the sets involved.

To show how this works, let us first define $\mathbf{1} := \{\bullet\}$, a singleton set whose only element is the symbol “•” (any singleton set would do; for concreteness we are fixing one and calling it $\mathbf{1}$).

Now we are ready to make an interesting observation: functions $\mathbf{1} \rightarrow \mathbf{A}$ are in 1-to-1 correspondence with the elements of \mathbf{A} . A function $f : \mathbf{1} \rightarrow \mathbf{A}$ will have to map “•” to some element $f(\bullet) \in \mathbf{A}$, and since $\mathbf{1}$ has no other elements, that is all that f does. So f “picks out” an element of \mathbf{A} . We can work with functions $\mathbf{1} \rightarrow \mathbf{A}$ in place of elements of \mathbf{A} .

Next, let’s talk about function evaluation. Consider a function $g : \mathbf{A} \rightarrow \mathbf{B}$. Given an element $x \in \mathbf{A}$, this element will be mapped by g to an element $g(x) \in \mathbf{B}$. If we use, instead of $x \in \mathbf{A}$, the function $f : \mathbf{1} \rightarrow \mathbf{A}$ to which it corresponds, then the element $g(x) \in \mathbf{B}$ corresponds to the function $f \circ g : \mathbf{1} \rightarrow \mathbf{B}$. In other words, we can talk about evaluation of a function $g : \mathbf{A} \rightarrow \mathbf{B}$ “at an element of \mathbf{A} ” without actually using elements, but rather just using functions and function composition.

Diagrams

Another typical characteristic of “category theory culture” is to often use various kinds of diagrams.

3.6. Arithmetic with sets

In our notation for cartesian product, disjoint union, and function sets we have used notation inspired by basic operations in arithmetic, motivated in part by the following formulas for sizes of finite sets:

$$\text{card}(\mathbf{A} \times \mathbf{B}) = \text{card}(\mathbf{A}) \cdot \text{card}(\mathbf{B}), \quad (99)$$

$$\text{card}(\mathbf{A} + \mathbf{B}) = \text{card}(\mathbf{A}) + \text{card}(\mathbf{B}), \quad (100)$$

$$\text{card}(\mathbf{B}^{\mathbf{A}}) = \text{card}(\mathbf{B})^{\text{card}(\mathbf{A})}. \quad (101)$$

The parallels of these operations to operations in arithmetic go further. For example, consider the following identities which hold for any natural numbers x, y, z :

$$x \cdot y = y \cdot x, \quad (102)$$

$$x + y = y + x, \quad (103)$$

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z, \quad (104)$$

$$x + (y + z) = (x + y) + z; \quad (105)$$

$$z^{x \cdot y} = (z^y)^x, \quad z^{x \cdot y} = (z^x)^y, \quad (106)$$

$$(x \cdot y)^z = x^z \cdot y^z, \quad (107)$$

$$x^{(y+z)} = x^y \cdot x^z, \quad (108)$$

$$(x + y)^z = \sum_{k=0}^z \binom{z}{k} x^k \cdot y^{z-k}; \quad (109)$$

$$1 \cdot x = x, \quad (110)$$

$$0 + x = x, \quad (111)$$

$$x^1 = x, \quad (112)$$

$$x^0 = 1, \quad (113)$$

$$1^x = 1, \quad (114)$$

$$0^x = 0; \quad (115)$$

$$(x \cdot y) + (x \cdot z) = x \cdot (y + z), \quad (116)$$

$$0 \cdot x = 0; \quad (117)$$

These identities also hold on the level of sets, before computing their size; we simply need to replace “=” with the symbol “ \simeq ” for “isomorphic”:

$$\mathbf{A} \times \mathbf{B} \simeq \mathbf{B} \times \mathbf{A}, \quad (118)$$

$$\mathbf{A} + \mathbf{B} \simeq \mathbf{B} + \mathbf{A}, \quad (119)$$

$$(\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \simeq \mathbf{A} \times (\mathbf{B} \times \mathbf{C}), \quad (120)$$

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} \simeq \mathbf{A} + (\mathbf{B} + \mathbf{C}); \quad (121)$$

$$\mathbf{C}^{\mathbf{A} \times \mathbf{B}} \simeq (\mathbf{C}^{\mathbf{B}})^{\mathbf{A}}, \quad \mathbf{C}^{\mathbf{A} \times \mathbf{B}} \simeq (\mathbf{C}^{\mathbf{A}})^{\mathbf{B}}, \quad (122)$$

$$(\mathbf{A} \times \mathbf{B})^{\mathbf{C}} \simeq \mathbf{A}^{\mathbf{C}} \times \mathbf{B}^{\mathbf{C}}, \quad (123)$$

$$\mathbf{A}^{\mathbf{B} + \mathbf{C}} \simeq \mathbf{A}^{\mathbf{B}} \times \mathbf{A}^{\mathbf{C}}, \quad (124)$$

$$(\mathbf{A} + \mathbf{B})^{\mathbf{C}} \simeq \sum_{\mathbf{S} \in \text{Pow } \mathbf{C}} \mathbf{A}^{\mathbf{S}} \times \mathbf{B}^{\mathbf{C} \setminus \mathbf{S}}; \quad (125)$$

$$\mathbf{1} \times \mathbf{A} \simeq \mathbf{A}, \quad (126)$$

$$\emptyset + \mathbf{A} \simeq \mathbf{A} \quad (127)$$

$$\mathbf{A}^1 \simeq \mathbf{A} \quad (128)$$

$$\mathbf{A}^\emptyset \simeq \mathbf{1} \quad (129)$$

$$\mathbf{1}^{\mathbf{A}} \simeq \mathbf{1} \quad (130)$$

$$\emptyset^{\mathbf{A}} = \emptyset; \quad (131)$$

$$(\mathbf{A} \times \mathbf{B}) + (\mathbf{A} \times \mathbf{C}) \simeq \mathbf{A} \times (\mathbf{B} + \mathbf{C}), \quad (132)$$

$$\emptyset \times \mathbf{A} = \emptyset. \quad (133)$$

We can say even more: not only do we have the above statements about certain sets being isomorphic, but in fact in each case there exists a particular, special isomorphism which mathematicians might call "natural" or "canonical". These terms are often used in an informal way to describe situations where some mathematical structure, such as a certain function, exists without the need for any particular "extra" or "ad hoc" choices to be made. Often this situation involves a family of cases, and a "canonical choice" is one which is possible to construct for all cases at once, without reference to the details of any particular case. We'll illustrate this idea by describing a canonical isomorphism for each of the "equations" above. We do this also because we will use many of these isomorphisms over and over again, and hence it makes sense to get to know them and give them names.

Commutativity

The equations $x \cdot y = y \cdot x$ and $x + y = y + x$ describe the *commutativity property* of multiplication and of addition, respectively.

Let us look at the corresponding relationship on the level of sets, for the case of multiplication:

$$\mathbf{A} \times \mathbf{B} \simeq \mathbf{B} \times \mathbf{A}. \quad (134)$$

The canonical isomorphism in this case, which gives proof for this relationship, is

$$\begin{aligned} \text{br} : \mathbf{A} \times \mathbf{B} &\rightarrow \mathbf{B} \times \mathbf{A}, \\ \langle x, y \rangle &\mapsto \langle y, x \rangle. \end{aligned} \quad (135)$$

The name **br** is short for "braiding".

This function is "canonical" because, once again, it's "shape" or "recipe" works for *any* two sets **A** and **B**. To explicate this in more detail, consider, for example, the particular sets $\mathbf{A} = \{0, 1, 2\}$ and $\mathbf{B} = \{\star, \dagger\}$. Then, besides the canonical function $\text{br} : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{B} \times \mathbf{A}$, there are also *other* (non-canonical) isomorphisms possible, such as for instance the function $\mathbf{A} \times \mathbf{B} \rightarrow \mathbf{B} \times \mathbf{A}$ which maps like this:

$$\langle 0, \star \rangle \mapsto \langle \star, 1 \rangle \quad (136)$$

$$\langle 1, \star \rangle \mapsto \langle \star, 2 \rangle \quad (137)$$

$$\langle 2, \star \rangle \mapsto \langle \star, 3 \rangle \quad (138)$$

$$\langle 0, \dagger \rangle \mapsto \langle \dagger, 1 \rangle \quad (139)$$

$$\langle 1, \dagger \rangle \mapsto \langle \dagger, 2 \rangle \quad (140)$$

$$\langle 2, \dagger \rangle \mapsto \langle \dagger, 3 \rangle. \quad (141)$$

It may be written more compactly by the recipe “ $\langle x, y \rangle \mapsto \langle y, x + 1 \bmod 3 \rangle$ ”. However, this recipe still depends on specific features of the set \mathbf{A} and would not work for any two arbitrary sets (namely, it uses the fact that the elements of \mathbf{A} in this example are integers for which the operation “mod 3” makes sense). Thus this function is not “canonical” (while br is).

The above discussion of commutativity regards multiplication. A similar situation holds for addition, with the statement $\mathbf{A} + \mathbf{B} \simeq \mathbf{B} + \mathbf{A}$ as the analogue, on the level of sets, of the commutativity property for the operation of addition for natural numbers. For this relationship there is also proof via a canonical isomorphism. We use the name br for this canonical isomorphism, too.

Exercise8. Can you guess the canonical isomorphism $\text{br} : \mathbf{B} + \mathbf{A} \rightarrow \mathbf{A} + \mathbf{B}$? Prove that your guess is indeed an isomorphism.

See solution on page 81.

Associativity

The equations $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ and $x + (y + z) = (x + y) + z$ describe, respectively, the *associativity* property of multiplication and addition of natural numbers. Correspondingly, there are canonical isomorphisms

$$\text{as} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \rightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C}) \quad (142)$$

and

$$\text{as} : (\mathbf{A} + \mathbf{B}) + \mathbf{C} \rightarrow \mathbf{A} + (\mathbf{B} + \mathbf{C}) \quad (143)$$

(which we both call by the same name). For the case of the cartesian product, the isomorphism as is

$$\begin{aligned} \text{as} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} &\rightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C}), \\ \langle \langle x, y \rangle, z \rangle &\mapsto \langle x, \langle y, z \rangle \rangle. \end{aligned} \quad (144)$$

Exercise9. Guess how as is defined for the sum (disjoint union) of sets, and check that your guess is indeed an isomorphism.

See solution on page 81.

Singletons are like the number one

In - 2 we defined the singleton set $\mathbf{1} = \{\bullet\}$ in order to have – for practical reasons of exposition and readability – a “standard, go-to, default one-element set”. This set (and any other singleton set) behaves like the natural number “1” in the sense that

$$\mathbf{1} \times \mathbf{A} \simeq \mathbf{A} \quad \text{and} \quad \mathbf{A} \times \mathbf{1} \simeq \mathbf{A} \quad (145)$$

is true for any set \mathbf{A} . In each case there is also a *canonical* isomorphism, one which does not involve any ad hoc choices:

$$\begin{aligned} \text{lu} : \mathbf{1} \times \mathbf{A} &\rightarrow \mathbf{A}, \\ \langle \bullet, x \rangle &\mapsto x, \end{aligned} \quad (146)$$

and

$$\begin{aligned} \text{ru} : \mathbf{A} \times \mathbf{1} &\rightarrow \mathbf{A}, \\ \langle x, \bullet \rangle &\mapsto x. \end{aligned} \quad (147)$$

The names lu and ru stand for *left unitor* and *right unitor*, respectively.

These functions are canonical because their "shape" or "recipe" does not depend on the particular set \mathbf{A} involved; the recipe is simply "forget about the element \bullet ". It is in this sense that it works for "all cases at once" and does not depend on any extra or ad hoc choices.

The empty set is like the number zero

Analogous to the equations $0 + x = x$ and $x + 0 = x$, we have

$$\emptyset + \mathbf{A} \simeq \mathbf{A} \quad \text{and} \quad \mathbf{A} + \emptyset \simeq \mathbf{A}. \quad (148)$$

And here again we have canonical isomorphisms

$$\begin{aligned} \mathbf{lu} : \emptyset + \mathbf{A} &\rightarrow \mathbf{A}, \\ \langle 2, \mathbf{x} \rangle &\mapsto \mathbf{x}, \end{aligned} \quad (149)$$

and

$$\begin{aligned} \mathbf{ru} : \mathbf{A} + \emptyset &\rightarrow \mathbf{A}, \\ \langle 1, \mathbf{x} \rangle &\mapsto \mathbf{x}. \end{aligned} \quad (150)$$

which we also call *left unitor* and *right unitor*, respectively.

Functions out of products

Next we look at the equations $z^{x \cdot y} = (z^y)^x$ and $z^{x \cdot y} = (z^x)^y$, and the respective corresponding relationships

$$\mathbf{C}^{\mathbf{A} \times \mathbf{B}} \simeq (\mathbf{C}^{\mathbf{B}})^{\mathbf{A}} \quad (151)$$

and

$$\mathbf{C}^{\mathbf{A} \times \mathbf{B}} \simeq (\mathbf{C}^{\mathbf{A}})^{\mathbf{B}}. \quad (152)$$

We'll only treat the first relationship (151) in detail, since the latter is analogous, except that certain roles are swapped.

What might be a canonical isomorphism proving (151)? The elements of the set $\mathbf{C}^{\mathbf{A} \times \mathbf{B}}$ are functions of two variables, of the type

$$\mathbf{f} : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C}, \quad (153)$$

while elements of the set $(\mathbf{C}^{\mathbf{B}})^{\mathbf{A}}$ are functions of the type

$$\mathbf{g} : \mathbf{A} \rightarrow \mathbf{C}^{\mathbf{B}}; \quad (154)$$

in other words, functions of a single variable but which evaluate, for each input, to a function of the type $\mathbf{B} \rightarrow \mathbf{C}$. A canonical way to turn a function of the type (153) into a function of the type (154) is by "partial evaluation": if we think of \mathbf{f} as having two input slots, we can partially evaluate \mathbf{f} by inserting an element of \mathbf{A} into the first slot of \mathbf{f} , while leaving the second slot "open" or "variable". In other words, for any element $\mathbf{x} \in \mathbf{A}$, we can create, from \mathbf{f} , the function

$$\begin{aligned} \mathbf{f}(\mathbf{x}, -) : \mathbf{B} &\rightarrow \mathbf{C}, \\ y &\mapsto \mathbf{f}(\mathbf{x}, y). \end{aligned} \quad (155)$$

But since we can do this for any $x \in A$, we have also just described a function

$$\begin{aligned} A &\rightarrow C^B, \\ x &\mapsto f(x, -), \end{aligned} \tag{156}$$

or, in other words, an element of $(C^B)^A$. Since this recipe works for any function $f : A \times B \rightarrow C$, we also now have the following function

$$\begin{aligned} cu : C^{A \times B} &\rightarrow (C^B)^A, \\ f &\mapsto (x \mapsto f(x, -)), \end{aligned} \tag{157}$$

which we call **cu**. This function has the desired type to be an isomorphism proving that $C^{A \times B} \simeq (C^B)^A$.

Exercise 10. Prove that $cu : C^{A \times B} \rightarrow (C^B)^A$ is an isomorphism.

See solution on page 81.

The name “cu” stands for “Curry”, which is the last name of the mathematician and logician Haskell Curry (Fig. 11) who did not discover this operation, but likely made it well-known. It is after his first name that the functional programming language Haskell is named.

The operation given by the function (157) is often called “currying”. When needed for clarity, we call this operation *right currying*, in order to distinguish it from the analogous operation of *left currying* given by the function

$$\begin{aligned} cu : C^{A \times B} &\rightarrow (C^A)^B, \\ f &\mapsto (y \mapsto f(-, y)). \end{aligned} \tag{158}$$

We denote the functions (157) and (158) both by the same name “cu” because we trust that it will be clear from context which of the two functions is being used in any given situation.



Figure 11.: Haskell Curry (1900-1982)

Functions into products

Now let's consider the analogue of the equation

$$(x \cdot y)^z = x^z \cdot y^z \tag{159}$$

for natural numbers, namely

$$(A \times B)^C \simeq A^C \times B^C. \tag{160}$$

In order to describe a canonical isomorphism which is a proof of this statement, we make an important observation about the cartesian product of sets. Namely, for any sets A and B , we always have two “projection functions” from the cartesian product $A \times B$ to A and B , respectively. These are defined, respectively, by

$$\begin{aligned} pr_1 : A \times B &\rightarrow A, \\ \langle x, y \rangle &\mapsto x, \end{aligned} \tag{161}$$

and

$$\begin{aligned} pr_2 : A \times B &\rightarrow B, \\ \langle x, y \rangle &\mapsto y, \end{aligned} \tag{162}$$

and are called the *canonical projections* associated to the cartesian product. With the help of these projections, we define a canonical isomorphism

$$\begin{aligned} \textcolor{blue}{f} : (\mathbf{A} \times \mathbf{B})^{\mathbf{C}} &\rightarrow \mathbf{A}^{\mathbf{C}} \times \mathbf{B}^{\mathbf{C}}, \\ \phi &\mapsto \langle \phi \circ \text{pr}_1, \phi \circ \text{pr}_2 \rangle. \end{aligned} \quad (163)$$

Illustrated diagrammatically, what f does is, given a function

$$\begin{array}{ccc} & \mathbf{C} & \\ & \downarrow \phi & \\ \mathbf{A} \times \mathbf{B} & & \end{array} \quad (164)$$

it maps it to the pair of functions $\phi \circ \text{pr}_1$ and $\phi \circ \text{pr}_2$

$$\begin{array}{ccccc} & & \mathbf{C} & & \\ & & \downarrow \phi & & \\ & \phi \circ \text{pr}_1 & \nearrow & \searrow & \phi \circ \text{pr}_2 \\ \mathbf{A} & \xleftarrow{\text{pr}_1} & \mathbf{A} \times \mathbf{B} & \xrightarrow{\text{pr}_2} & \mathbf{B} \end{array} \quad (165)$$

obtained by post-composing ϕ with the projections pr_1 and pr_2 , respectively.

The inverse to f is the function $g : \mathbf{A}^{\mathbf{C}} \times \mathbf{B}^{\mathbf{C}} \rightarrow (\mathbf{A} \times \mathbf{B})^{\mathbf{C}}$ which takes a pair of functions $\langle \psi_1, \psi_2 \rangle \in \mathbf{A}^{\mathbf{C}} \times \mathbf{B}^{\mathbf{C}}$ and maps it to the element of $(\mathbf{A} \times \mathbf{B})^{\mathbf{C}}$ which is the function

$$\begin{aligned} \mathbf{C} &\rightarrow \mathbf{A} \times \mathbf{B}, \\ z &\mapsto \langle \psi_1(z), \psi_2(z) \rangle. \end{aligned} \quad (166)$$

Graded exercise B.4 (CanIsoFunctionsIntoProducts)

Check that f and g are indeed mutually inverse.

Functions out of sums

Here we consider the equation

$$x^{(y+z)} = x^y \cdot x^z \quad (167)$$

and its analogue on the level of sets,

$$\mathbf{A}^{\mathbf{B}+\mathbf{C}} \simeq \mathbf{A}^{\mathbf{B}} \times \mathbf{A}^{\mathbf{C}}. \quad (168)$$

Similar to the observation that cartesian products come along with canonical projection functions, we make the observation here that sums of sets come equipped with canonical *inclusion functions*

$$\begin{aligned} \text{in}_1 : \mathbf{B} &\rightarrow \mathbf{B} + \mathbf{C} \\ y &\mapsto \langle 1, y \rangle, \end{aligned} \quad (169)$$

and

$$\begin{aligned} \text{in}_2 : \mathbf{C} &\rightarrow \mathbf{B} + \mathbf{C}, \\ z &\mapsto \langle 2, z \rangle. \end{aligned} \quad (170)$$

Using these, we construct the following canonical isomorphism:

$$\begin{aligned} f : A^{B+C} &\rightarrow A^B \times A^C, \\ \phi &\mapsto \langle \text{in}_1 \circ \phi, \text{in}_2 \circ \phi \rangle. \end{aligned} \quad (171)$$

Illustrated diagrammatically, f takes any function

$$\begin{array}{c} A \\ \uparrow \phi \\ B + C \end{array} \quad (172)$$

it maps it to the pair of functions $\text{in}_1 \circ \phi$ and $\text{in}_2 \circ \phi$

$$\begin{array}{ccccc} & & C & & \\ & \nearrow \text{in}_1 \circ \phi & \uparrow \phi & \searrow \text{in}_2 \circ \phi & \\ B & \xrightarrow{\text{in}_1} & B + C & \xleftarrow{\text{in}_2} & C \end{array} \quad (173)$$

obtained by pre-composing ϕ with the inclusions in_1 and in_2 , respectively.

The inverse to f is the function $g : A^B \times A^C \rightarrow A^{B+C}$ which maps any pair of functions $\langle \psi_1, \psi_2 \rangle \in A^B \times A^C$ to the element of A^{B+C} given by the function

$$\begin{aligned} B + C &\rightarrow A, \\ u &\mapsto \begin{cases} \psi_1(y) & \text{if } u = \langle 1, y \rangle \text{ for some } y \in B, \\ \psi_2(z) & \text{if } u = \langle 2, z \rangle \text{ for some } z \in C. \end{cases} \end{aligned} \quad (174)$$

Graded exercise B.5 (CanIsoFunctionsOutOfSums)

Check that f and g are indeed mutually inverse.

Functions into sums

The equation

$$(x+y)^z = \sum_{k=0}^z \binom{z}{k} x^k \cdot y^{z-k} \quad (175)$$

is known as the binomial theorem, and the numbers

$$\binom{z}{k} = \frac{z!}{(z-k)! k!} \quad (176)$$

are called the binomial coefficients. On the level of sets and functions, the binomial theorem is analogous to the statement

$$(A + B)^C \simeq \sum_{S \in \text{Pow } C} A^S \times B^{C \setminus S}. \quad (177)$$

The binomial coefficients are related to combinatorics; the number

$$\binom{z}{k} \quad (178)$$

describes the number of different ways that one may choose a k -element subset out of a fixed z -element set. Using this connection as a guide, we can re-write the right-hand side of our statement about sets and functions into a form that is

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

closer to the right-hand side of the binomial theorem:

$$\sum_{S \in \text{Pow } C} A^S \times B^{C \setminus S} \approx \sum_{k=0}^{|C|} \sum_{\substack{S \in \text{Pow } C \\ |S|=k}} A^S \times B^{C \setminus S}. \quad (179)$$

Now, on the right-hand side, for each fixed k , the cardinality of the set

$$\sum_{\substack{S \in \text{Pow } C \\ |S|=k}} A^S \times B^{C \setminus S} \quad (180)$$

is precisely

$$\binom{z}{k} x^k \cdot y^{z-k}, \quad (181)$$

with $z = |C|$.

Next, let's show that there is a canonical isomorphism

$$f : (\mathbf{A} + \mathbf{B})^C \rightarrow \sum_{S \in \text{Pow } C} A^S \times B^{C \setminus S}. \quad (182)$$

Unpacking the definition of unordered sums of sets, the target set of this function is

$$\bigcup_{S \in \text{Pow } C} \{S\} \times (A^S \times B^{C \setminus S}) \quad (183)$$

To define f , we begin by choosing an arbitrary $\phi \in (\mathbf{A} + \mathbf{B})^C$; that is, a function

$$\phi : C \rightarrow \mathbf{A} + \mathbf{B}. \quad (184)$$

Because $\mathbf{A} + \mathbf{B}$ is the *disjoint* union of $\{1\} \times \mathbf{A}$ and $\{2\} \times \mathbf{B}$, their respective pre-images under ϕ have empty intersection, and their union is all of C . In other words, if we set

$$\mathbf{T} := \phi^{-1}(\{1\} \times \mathbf{A}), \quad (185)$$

then

$$\phi^{-1}(\{2\} \times \mathbf{B}) = C \setminus \mathbf{T}. \quad (186)$$

The image $f(\phi)$ should be an element of $\{S\} \times (A^S \times B^{C \setminus S})$ for some $S \in \text{Pow } C$. We will choose $S = \mathbf{T}$. What remains for us to do, then, is to define an element of $\mathbf{A}^{\mathbf{T}}$ and an element of $\mathbf{B}^{C \setminus \mathbf{T}}$, using ϕ . To obtain an element of $\mathbf{A}^{\mathbf{T}}$, we use the composite

$$\mathbf{T} \xrightarrow{\phi|_{\mathbf{T}}} \{1\} \times \mathbf{A} \rightarrow \mathbf{A}, \quad (187)$$

and to obtain an element of $\mathbf{B}^{C \setminus \mathbf{T}}$, we use

$$C \setminus \mathbf{T} \xrightarrow{\phi|_{C \setminus \mathbf{T}}} \{2\} \times \mathbf{B} \rightarrow \mathbf{B}. \quad (188)$$

The intuition behind this definition of f is that any function $\phi : C \rightarrow \mathbf{A} + \mathbf{B}$ essentially amounts to two functions “glued together”: one, $\phi|_{\mathbf{T}}$, which accounts for all the elements of C that are mapped by ϕ to $\{1\} \times \mathbf{A}$, and another, $\phi|_{C \setminus \mathbf{T}}$, which accounts for all the elements of C that are mapped by ϕ to $\{2\} \times \mathbf{B}$.

This intuition can also help us define an inverse

$$g : \sum_{S \in \text{Pow } C} A^S \times B^{C \setminus S} \rightarrow (\mathbf{A} + \mathbf{B})^C \quad (189)$$

to f . For this, fix an $\mathbf{S} \in \text{Pow } \mathbf{C}$. Given an element

$$\langle \mathbf{S}, \psi_1, \psi_2 \rangle \in \{\mathbf{S}\} \times (\mathbf{A}^{\mathbf{S}} \times \mathbf{B}^{\mathbf{C} \setminus \mathbf{S}}), \quad (190)$$

we define $g(\langle \mathbf{S}, \psi_1, \psi_2 \rangle)$ to be the function

$$\begin{array}{rcl} \mathbf{C} & \rightarrow & \mathbf{A} + \mathbf{B} \\ z & \mapsto & \begin{cases} \langle 1, \psi_1(z) \rangle \text{ if } z \in \mathbf{S}, \\ \langle 2, \psi_2(z) \rangle \text{ if } z \in \mathbf{C} \setminus \mathbf{S}. \end{cases} \end{array} \quad (191)$$

Roughly speaking, $g(\langle \mathbf{S}, \psi_1, \psi_2 \rangle) : \mathbf{C} \rightarrow \mathbf{A} + \mathbf{B}$ is just the “gluing together” of ψ_1 and ψ_2 .

Graded exercise B.6 (`CanIsoFunctionsIntoSums`)

Check that f and g are indeed mutually inverse.

Distributivity

The statement

$$(\mathbf{A} \times \mathbf{B}) + (\mathbf{A} \times \mathbf{C}) \simeq \mathbf{A} \times (\mathbf{B} + \mathbf{C}). \quad (192)$$

is analogous to the *distributivity* property of multiplication and addition of numbers:

$$(x \cdot y) + (x \cdot z) = x \cdot (y + z). \quad (193)$$

Exercise 11. Guess the canonical isomorphism $\text{di} : (\mathbf{A} \times \mathbf{B}) + (\mathbf{A} \times \mathbf{C}) \rightarrow \mathbf{A} \times (\mathbf{B} + \mathbf{C})$ and verify that it is indeed an isomorphism.

See solution on page 81.

The empty set is (again) like the number zero

The relationship

$$\emptyset \times \mathbf{A} = \emptyset, \quad (194)$$

which holds for any set \mathbf{A} , is analogous to the equation

$$0 \cdot x = x \quad (195)$$

for any natural number x . (And also $\mathbf{A} \times \emptyset = \emptyset$, just as $x \cdot 0 = 0$.)

These equations follows from Def. 3.7 (or from the formulation given in Remark 3.9).

That this relationship is an equality – and not merely an isomorphism like all the other “arithmetic relationships” between sets that we will address – has to do with the fact that there exists only one unique set having the cardinality zero, namely the empty, while for any non-zero natural number x , there exist infinitely-many sets having the cardinality x .



4. Relations

This chapter describes *relations*, which are a generalization of functions.

4.1 Distribution networks	62
4.2 Relations	65
4.3 Composing relations	66
4.4 Relations and functions	68
4.5 Properties of relations	71
4.6 Transpose of a relation	72
4.7 Endorelations	73
4.8 Equivalence relations	76

4.1. Distribution networks

Consider the type of networks that arise for example in the context of electrical power grids. In a simplified model for a certain region or country, we may have the following kinds of components: power plants (places where electrical power is produced), high voltage transmission lines and nodes, transformers stations, low voltage transmission lines and nodes, and consumers, such as homes and businesses. The situation is depicted in Fig. 1.

To model the connectivity between the components of the power grid, we now draw arrows between components that are connected. We set the direction of the arrows to flow from energy production, via transmission components, to energy consumption, as depicted in Fig. 2.

A possible question one asks about such a power distribution network is: which consumers are serviced by which power sources? For example, power sources such as a solar power plant may fluctuate due to weather conditions, while other power sources, such as a nuclear power plant, may shut down every once in a while due to maintenance work. To see which consumers are connected to which power plants, we can follow “connectivity paths” traced by sequences of arrows, as in Fig. 3. There, two possible connectivity paths are depicted (in red and orange, respectively).

We also will want to know the overall connectivity structure of transmission lines. For example, some lines may go down during a storm, and we want to ensure enough redundancy in our system. In addition to the connections modeled in Fig. 2, we can also include, for example, information about the connectivity of high voltage nodes among themselves, as in Fig. 4.

Power Plants	High Voltage Nodes	Low Voltage Nodes	Consumers
Plant 1	HVN 1	LVN 1	C1
	HVN 2	LVN 2	C2
Plant 2	HVN 3	LVN 3	C3
	HVN 4	LVN 4	C4
Plant 3	HVN 5	LVN 5	C5
		LVN 6	C6
		LVN 7	

Figure 1.: Components of electrical power grids.

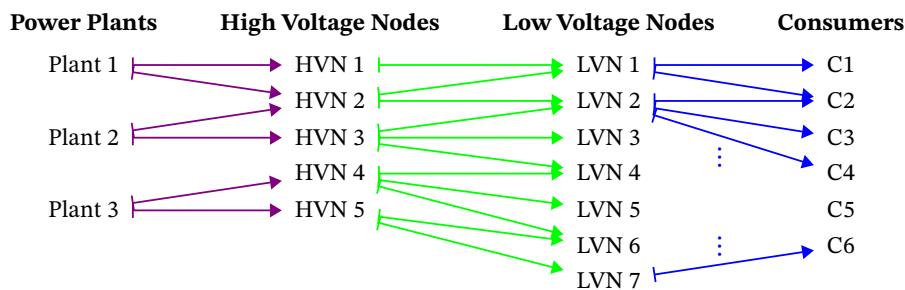


Figure 2.: Connectivity between components in electric power grids.

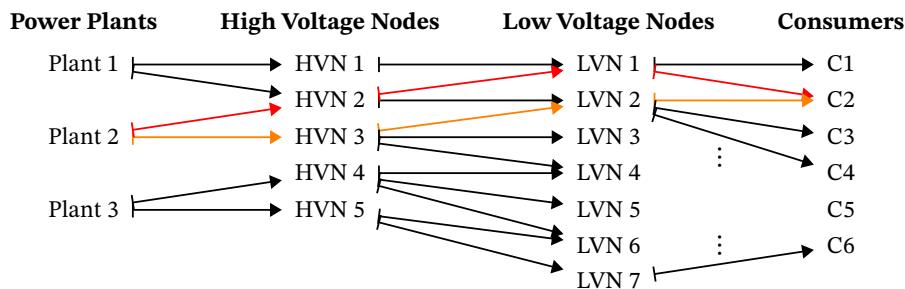


Figure 3.: Connection between consumers and power plants.

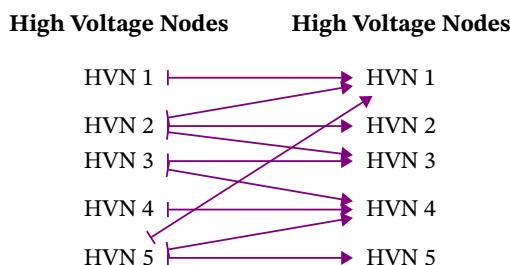


Figure 4.: Connectivity between high voltage nodes.

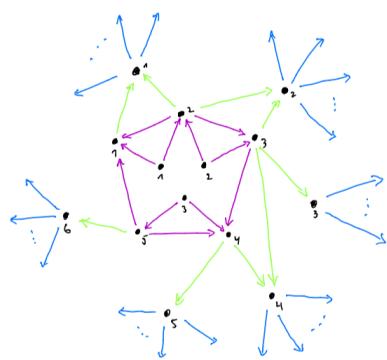


Figure 5.: Alternative visualization for connectivity.

The information encoded in Fig. 4 and Fig. 2 can also be displayed as a single graph, see Figs. 4 and 5.

If we ignore the directionality of the arrows, this is analogous to a depiction of type shown in Fig. 6, which is a schema of a power grid [2]*.

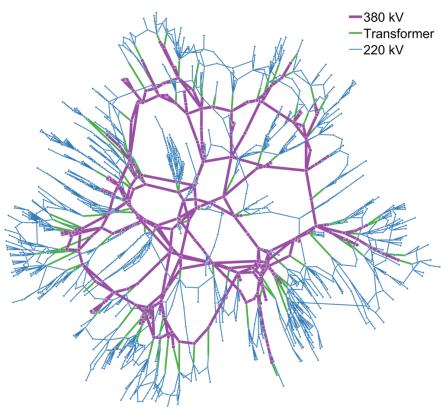


Figure 6.: A schematic view of a power grid.

* See https://en.wikipedia.org/wiki/Electrical_grid

4.2. Relations

A basic mathematical notion which underlies the above discussion is that of a **binary relation**.

Definition 4.1 (Binary relation)

A *binary relation* R from a set \mathbf{A} to a set \mathbf{B} is a subset of the cartesian product $\mathbf{A} \times \mathbf{B}$:

$$R \subseteq \mathbf{A} \times \mathbf{B}. \quad (1)$$

We will often drop the word “binary” and simply use the name “relation”.

We also write

$$R : \mathbf{A} \rightarrow \mathbf{B} \quad (2)$$

to indicate a relation from \mathbf{A} to \mathbf{B} . (\mathbf{A} is the source, and \mathbf{B} is the target).

Example 4.2. Let $\mathbf{A} = \{\text{pretzel}, \text{roast beef}, \text{diamond}\}$ and $\mathbf{B} = \{\text{beer}, \text{water}, \text{juice}, \text{wine}\}$. An example of a relation is the subset

$$R = \{\langle \text{pretzel}, \text{beer} \rangle, \langle \text{roast beef}, \text{juice} \rangle, \langle \text{roast beef}, \text{wine} \rangle\} \subseteq \mathbf{A} \times \mathbf{B}. \quad (3)$$

If \mathbf{A} and \mathbf{B} are finite sets, we can depict a relation $R : \mathbf{A} \rightarrow \mathbf{B}$ graphically as in Fig. 7. For each element $\langle x, y \rangle \in \mathbf{A} \times \mathbf{B}$, we draw an arrow from x to y if and only if $\langle x, y \rangle \in R$.

We can also depict this relation graphically as a subset of $\mathbf{A} \times \mathbf{B}$ in a “coordinate system way”, as in Fig. 8.

The shaded area is the subset R defining the relation.

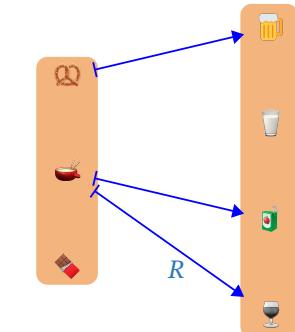


Figure 7.

Remark 4.3 (Notation for relations). From now on we will also use the following notation, where we write

$$x R y := \langle x, y \rangle \in R \quad (4)$$

instead of writing $\langle x, y \rangle \in R$.

Exercise 12. Given an arbitrary set \mathbf{B} , does there always exist a relation $\emptyset \rightarrow \mathbf{B}$?

See solution on page 81.

Exercise 13. Given an arbitrary set \mathbf{A} , does there always exist a relation $\mathbf{A} \rightarrow \emptyset$?

See solution on page 81.

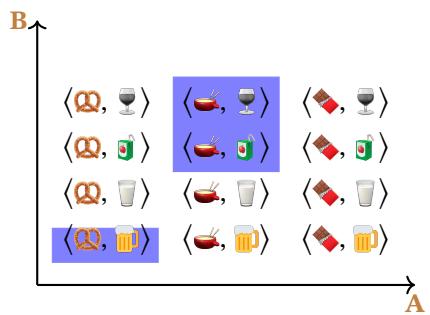


Figure 8.: Relations visualized in “coordinate systems”.

Graded exercise B.7 (visualizeLeqRelation)

Let $\mathbf{A} = \mathbf{B} = \{1, 2, 3, 4\}$ and consider the relation $R : \mathbf{A} \rightarrow \mathbf{B}$ defined by

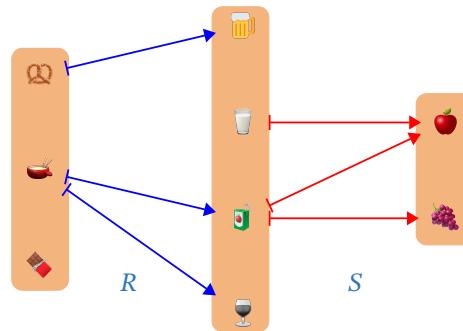
$$R = \{(x, y) \in \mathbf{A} \times \mathbf{B} \mid x \leq y\}. \quad (5)$$

Visualize the relation R via the method in Fig. 7 and Fig. 8 each.

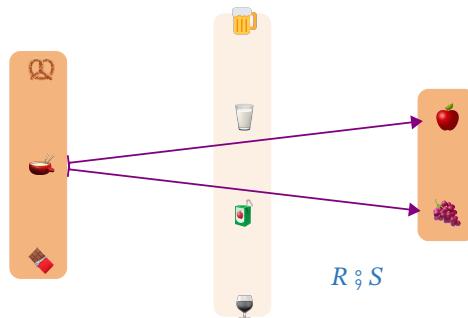
4.3. Composing relations

The visualization in Fig. 7 hints at the fact that we can compose relations if the target of the one is the source of the other.

To illustrate composition of relations, consider a simple example, involving sets **A**, **B**, and **C**, and relations $R : A \rightarrow B$ and $S : B \rightarrow C$, as depicted graphically below in Fig. 9a.



(a) Relations compatible for composition.



(b) Composition of relations.

Figure 9.: Illustrations for relations composition.

The composite relation $R ; S : A \rightarrow C$ is defined to be such that $x(R ; S)z$ if and only if there exists some $y \in B$ such that xRy and ySz . Graphically this means that for $\langle x, z \rangle$ to be an element of the relation $R ; S$, the elements x and z need to be connected by at least one sequence of two arrows such that the target of the first arrow is the source of the second.

For example, in Fig. 9a, there is an arrow from to , and from there on to , and therefore, in the composition $R ; S$ depicted in Fig. 9b, there is an arrow from to .

Definition 4.4 (Relation composition)

Given relations $R : A \rightarrow B, S : B \rightarrow C$, their composition is the relation

$$R ; S := \{ \langle x, z \rangle \in A \times C \mid \exists y \in B : (xRy) \wedge (ySz) \}. \quad (6)$$

Graded exercise B.8 (ComposingRelations)

- Let $A = \mathbb{N}$, $B = \mathbb{Z}$, and $C = \mathbb{R}$. Consider the relation $f : A \rightarrow B$ with

$$f = \{ \langle x, y \rangle \in \mathbb{N} \times \mathbb{Z} \mid x = y^2 \}, \quad (7)$$

and consider the relation $g : \mathbf{B} \rightarrow \mathbf{C}$ with

$$g = \{\langle y, z \rangle \in \mathbb{Z} \times \mathbb{R} \mid y = 2z\}. \quad (8)$$

Calculate the relation $f ; g : \mathbf{A} \rightarrow \mathbf{B}$.

2. Let $\mathbf{A} = \mathbb{N}$, $\mathbf{B} = \mathbb{Z} \times \mathbb{Z}$, and $\mathbf{C} = \mathbb{R}$. Consider the relation $f : \mathbf{A} \rightarrow \mathbf{B}$ with

$$f = \{\langle x, \langle y_1, y_2 \rangle \rangle \in \mathbb{N} \times (\mathbb{Z} \times \mathbb{Z}) \mid x = y_1 - y_2\} \quad (9)$$

and consider the relation $g : \mathbf{B} \rightarrow \mathbf{C}$ with

$$g = \{\langle \langle y_1, y_2 \rangle, z \rangle \in (\mathbb{Z} \times \mathbb{Z}) \times \mathbb{R} \mid y_2 z = y_1\}. \quad (10)$$

Calculate the relation $f ; g : \mathbf{A} \rightarrow \mathbf{B}$.

4.4. Relations and functions

Every function between sets can be thought as a relation: this was the basis of our formal definition of function, Def. 3.15. Let us restate that definition once again here, using notation that we have developed since.

Definition 4.5 (Function as a special type of relation)

Let \mathbf{A} and \mathbf{B} be sets. A relation $R \subseteq \mathbf{A} \times \mathbf{B}$ is a *function* if it satisfies the following two conditions:

1. for all $x \in \mathbf{A}$ there exists an element $y \in \mathbf{B}$ such that $x R y$;
2. for all x, y_1, y_2 , this holds:

$$\frac{x R y_1 \quad x R y_2}{y_1 = y_2}. \quad (11)$$

Although we will mostly continue to think about functions in the “usual” way (as opposed to the perspective of Def. 4.5), it is illuminating – both for understanding relations and functions – to study the relationships between the two points of view.

From functions to relations

Recall how to go from viewing a function in the “usual” way to viewing it as a relation, as in Def. 4.5.

As an illustration, consider the sets $\mathbf{A} = \{\text{🍕, 🍗, ♦}\}$ and $\mathbf{B} = \{\text{🍺, 🍷, 🍹, 🍷}\}$, and the function $f : \mathbf{A} \rightarrow \mathbf{B}$ defined (in the “usual” way) by

$$f(\text{🍕}) = \text{🍺}, \quad f(\text{🍗}) = \text{🍹}, \quad f(\text{♦}) = \text{🍷}. \quad (12)$$

This way of specifying the function f may be depicted graphically as in Fig. 10. The relation that this function defines, in the sense of Def. 4.5, is

$$\{\langle \text{🍕}, \text{🍺} \rangle, \langle \text{🍗}, \text{🍹} \rangle, \langle \text{♦}, \text{🍷} \rangle\} \subseteq \mathbf{A} \times \mathbf{B}. \quad (13)$$

This relation (13) is what is often called the *graph* of f . That is, it is the set of tuples in $\mathbf{A} \times \mathbf{B}$ which are a pairing of an element of the source set \mathbf{A} with the element which is its image under f . In Fig. 11, the graph of (13) is visualized by highlighting the elements of the graph among all the elements of $\mathbf{A} \times \mathbf{B}$.

In general, any function $f : \mathbf{A} \rightarrow \mathbf{B}$ corresponds to the relation

$$\{(x, y) \in \mathbf{A} \times \mathbf{B} \mid y = f(x)\}. \quad (14)$$

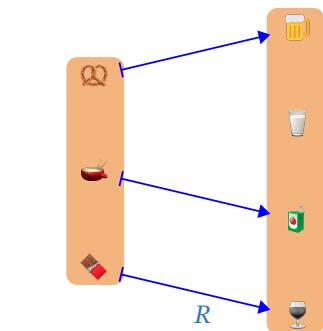


Figure 10.: Visualization of the function (13).

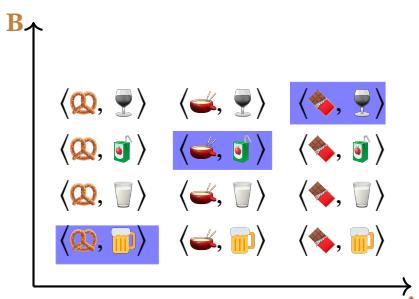


Figure 11.: The graph of the function (13).

From relations to functions

Let’s start now with a relation $R \subseteq \mathbf{A} \times \mathbf{B}$ satisfying the conditions of Def. 4.5 and see how this corresponds to a function $f_R : \mathbf{A} \rightarrow \mathbf{B}$ in the “usual” sense.

Choose an arbitrary $x \in \mathbf{A}$. According to point 1 in Def. 4.5, there exists a $y \in \mathbf{B}$ such that $x R y$. Choose such a y , and call it $f_R(x)$. This gives us recipe to get from any x to a y . But given a specific $x \in \mathbf{A}$, what if we choose y differently each time we apply the recipe? Point 2 guarantees that this can’t happen: it says that the element $f_R(x)$ that we associate to a given $x \in \mathbf{A}$ is in fact uniquely determined by that x . Put another way, the condition 2 says: if $f_R(x_1) \neq f_R(x_2)$, then $x_1 \neq x_2$.

Remark 4.6. Of course, not every relation corresponds to a function – namely precisely those that do not satisfy the conditions in Def. 4.5. For example, the relation in Fig. 7 is not a function. In fact, it does not satisfy *either* of the conditions in Def. 4.5.

Identity relations

We have already discussed how, for any set \mathbf{A} , there is always an identity function

$$\begin{aligned} \text{id}_{\mathbf{A}} : \mathbf{A} &\rightarrow \mathbf{A}, \\ y &\mapsto y. \end{aligned} \tag{15}$$

which “does nothing”. If we turn such functions into relations, we call the result *identity relations*.

Definition 4.7

Let \mathbf{A} be any set. The *identity relation* on \mathbf{A} is

$$\text{id}_{\mathbf{A}} = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{A} \mid x = y\}. \tag{16}$$

Composing functions

If we define functions as special kinds of relations, how is relation composition related to the “usual” way of composing of functions? The answer is that these two apparently different ways of composing functions actually give the same result.

Lemma 4.8. Let $R \subseteq \mathbf{A} \times \mathbf{B}$ and $S \subseteq \mathbf{B} \times \mathbf{C}$ be relations which are functions. Then their composition $R ; S \subseteq \mathbf{A} \times \mathbf{C}$ is again a function, and it corresponds to the “usual” composition of the functions corresponding to R and S .

Proof. First let us check that when R and S are composed as relations, the result is again a function. For this we check that $R ; S$ satisfies the two conditions stated in Def. 4.5.

1. Choose an arbitrary $x \in \mathbf{A}$. We need to show that there exists $z \in \mathbf{C}$ such that $x R ; S z$. Since R is a function, there exists $y \in \mathbf{B}$ such that $x R y$. Choose such a $y \in \mathbf{B}$. Then, because S is a function, there exists $z \in \mathbf{C}$ such that $y S z$. By the definition of composition of relations, we see that z is such that $x R ; S z$.
2. Let $x_1 R ; S z_1, x_2 R ; S z_2$. We need to show that if $x_1 = x_2$, then $z_1 = z_2$. So suppose $x_1 = x_2$. Since $x_1 R ; S z_1, x_2 R ; S z_2$, there exist $y_1, y_2 \in \mathbf{B}$ such that, respectively,

$$x_1 R y_1 \wedge y_1 S z_1, \tag{17}$$

$$x_2 R y_2 \wedge y_2 S z_2. \tag{18}$$

Since $x_1 = x_2$ and R is a function, we conclude that $y_1 = y_2$ must hold. Now, since S is also a function, this implies that $z_1 = z_2$, which is what was to be shown.

Second let us check that relation composition of functions gives the same result as the “usual” composition of functions. Let f_R and g_S denote the relations R and S when we are thinking of them in the “usual” way of thinking about functions. Our goal is to show that $f_R ; g_S$ corresponds to $R ; S$; in other words, that the latter is the graph of former.

Suppose first that $\langle x, z \rangle$ is in the graph of $f_R \circ g_S$, so $z = (f_R \circ g_S)(x)$. In particular $z = g_S(f_R(x))$, which means there exists $y = f_R(x) \in B$ such that $\langle x, y \rangle \in R$ and $\langle y, z \rangle \in S$. This implies that $\langle x, z \rangle \in R; S$.

Conversely, suppose $\langle x, z \rangle \in R; S$. By the definition of relation composition there must exist $y \in B$ such that $\langle x, y \rangle \in R$ and $\langle y, z \rangle \in S$, which means $y = f_R(x)$ and $z = g_S(y)$. Thus, $z = g_S(f_R(x))$. \square

Relations via functions

Though not every relation *is* a function, we can however think about relations *in terms of* functions. Here are three ways:

1. We can think of a relation $R : A \rightarrow B$ as a function $A \times B \rightarrow \text{Bool}$.

Given R we can define a function $\phi_R : A \times B \rightarrow \{\perp, \top\}$ from it by setting

$$\phi_R(\langle x, y \rangle) = \begin{cases} \top & \text{if } x R y, \\ \perp & \text{otherwise.} \end{cases} \quad (19)$$

Conversely, given a function $\phi : A \times B \rightarrow \{\perp, \top\}$ we can define a relation $R_\phi \subseteq A \times B$ from it by setting

$$R_\phi = \{\langle x, y \rangle \in A \times B \mid \phi_R(\langle x, y \rangle) = \top\}. \quad (20)$$

These two constructions are inverse to one-another.

2. We can think of a relation $R : A \rightarrow B$ as a function $A \rightarrow \text{Pow}(B)$.

Given R we can define a function $\hat{\phi}_R : A \rightarrow \text{Pow}(B)$ via

$$\hat{\phi}_R(x) = \{y \in B \mid x R y\}. \quad (21)$$

Conversely, given a function $\hat{\phi} : A \rightarrow \text{Pow}(B)$, we can define

$$R_{\hat{\phi}} = \{\langle x, y \rangle \in A \times B \mid y \in \hat{\phi}_R(x)\}. \quad (22)$$

These two constructions are inverse to one another, too.

3. We can think of a relation $R \subseteq A \times B$ as a function $B \rightarrow \text{Pow}(A)$.

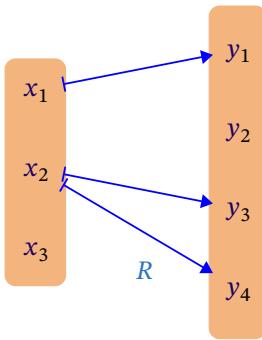
Given R we can define a function $\check{\phi}_R : B \rightarrow \text{Pow}(A)$ via

$$\check{\phi}_R(y) = \{x \in A \mid x R y\}. \quad (23)$$

Conversely, given a function $\check{\phi} : B \rightarrow \text{Pow}(A)$, we can define

$$R_{\check{\phi}} = \{\langle x, y \rangle \in A \times B \mid x \in \check{\phi}_R(y)\}. \quad (24)$$

These two constructions are *also* inverse to one another.



Graded exercise B.9 (Re13Functions)

For the relation R illustrated in Fig. 12, write out the three functions that describe it, respectively, in the three ways outlined in Section 4.4.

Figure 12.

4.5. Properties of relations

We have seen that relations generalize functions – every function defines a relation, via its graph, but not every relation comes from a function in this way. Many notions that we are familiar with for functions also generalize to relations. Here are a few.

Definition 4.9 (Properties of a relation)

We say that a relation $R : A \rightarrow B$ is:

1. *Injective* if

$$\frac{xRy \quad zRy}{x = z}; \quad (25)$$

2. *Single-valued* if

$$\frac{xRy \quad xRu}{y = u}; \quad (26)$$

3. *Surjective* if for all $y \in B$ there exists an element $x \in A$ such that xRy ;

4. *Everywhere-defined* if for all $x \in A$ there exists an element $y \in B$ such that xRy .

Example 4.10. The relation depicted in Fig. 7 is injective but not surjective. It is not single-valued, nor everywhere-defined.

4.6. Transpose of a relation

One can notice a certain duality in the properties listed in Def. 4.9. This is made more precise through the following definition.

Definition 4.11 (Transpose of a relation)

Let $R : \mathbf{A} \rightarrow \mathbf{B}$ be a relation. We define its *transpose* (or *opposite*, or *reverse*) $R^\top : \mathbf{B} \rightarrow \mathbf{A}$ as follows:

$$\frac{x R y}{\hline\hline} \quad (27)$$

$$y R^\top x$$

Remark 4.12. Here are some useful properties of a relation $R : \mathbf{A} \rightarrow \mathbf{B}$ and its opposite $R^\top : \mathbf{B} \rightarrow \mathbf{A}$:

1. $(R^\top)^\top = R$;
2. R is everywhere-defined if and only if R^\top is surjective;
3. R is single-valued if and only if R^\top is injective;
4. R is everywhere-defined if and only if $\text{id}_\mathbf{A} \subseteq R ; R^\top$;
5. R is single-valued if and only if $R^\top ; R \subseteq \text{id}_\mathbf{B}$.

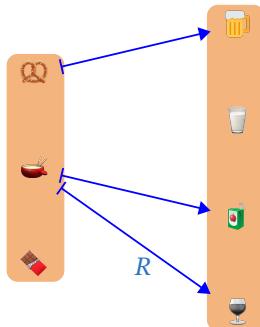


Figure 13.

Remark 4.13. The aforementioned duality can be seen by “reading the relations (arrows) backwards” (Figs. 13 and 14).

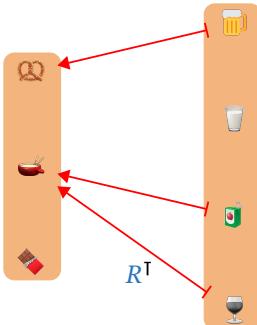


Figure 14.

Graded exercise B.10 (`RelProperties`)

Provide a proof of each of the properties listed in Remark 4.12.

4.7. Endorelations

Definition 4.14 (Endorelation)

An endorelation on a set \mathbf{A} is a relation $R : \mathbf{A} \rightarrow \mathbf{A}$.

Example 4.15. “Equality” on a set \mathbf{A} is an endorelation $=_{\mathbf{A}}$ of the form

$$=_{\mathbf{A}} := \{(x, y) \in \mathbf{A} \times \mathbf{A} \mid x = y\}. \quad (28)$$

Example 4.16. Take $\mathbf{A} = \mathbb{N}$. The relation “less than or equal” is an endorelation of the form

$$\leq := \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid x \leq y\}. \quad (29)$$

Example 4.17. The relation depicted in Fig. 4 is an endorelation between the set of high voltage nodes.

Definition 4.18 (Symmetry, asymmetry, and antisymmetry)

An endorelation $R : \mathbf{A} \rightarrow \mathbf{A}$ is *symmetric* if

$$\frac{x R y}{y R x}, \quad (30)$$

is *asymmetric* if

$$\frac{x R y \quad y R x}{\perp}, \quad (31)$$

and is *antisymmetric* if

$$\frac{x R y \quad y R x}{x = y}. \quad (32)$$

Definition 4.19 (Reflexivity and irreflexivity of endorelations)

An endorelation $R : \mathbf{A} \rightarrow \mathbf{A}$ is *reflexive* if

$$\frac{\top}{x R x}, \quad (33)$$

and is *irreflexive* if

$$\frac{x R x}{\perp}. \quad (34)$$

Table 4.1.: Summary of endorelation properties.

Reflexive	Total	Symmetric	Transitive
$\frac{\top}{x R x}$	$\frac{\top}{x R y \vee y R x}$	$\frac{x R y}{y R x}$	$\frac{x R y \quad y R z}{x R z}$
Irreflexive	Asymmetric	Antisymmetric	
$\frac{x R x}{\perp}$	$\frac{x R y \quad y R x}{\perp}$	$\frac{x R y \quad y R x}{x = y}$	

Definition 4.20 (Total)

An endorelation $R : \mathbf{A} \rightarrow \mathbf{A}$ is *total* if

$$\frac{\top}{(xRy) \vee (yRx)}. \quad (35)$$

Example 4.21. The relation “less than or equal” on \mathbb{N} is not symmetric. It is reflexive since $n \leq n \forall n \in \mathbb{N}$, and it is transitive since $l \leq m$ and $m \leq n$ implies $l \leq m$.

Example 4.22. The relation depicted in Fig. 4 is reflexive (each node is connected to itself).

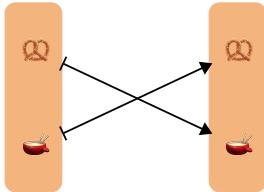


Figure 15.: Example of symmetric endorelation.

Example 4.23. The endorelation depicted in Fig. 15 is a symmetric relation on $\mathbf{A} = \{\text{🥪, 🍲}\}$.

Definition 4.24 (Transitive)

An endorelation $R : \mathbf{A} \rightarrow \mathbf{A}$ is *transitive* if

$$\frac{xRy \ yRz}{xRz}. \quad (36)$$

Example 4.25. The relation “has the same birthday as” is transitive because if Anna has the same birthday as Bob, and Bob has the same birthday as Clara, then Anna has the same birthday as Clara.

Closures

Sometimes an endorelation $R \subseteq \mathbf{A} \times \mathbf{A}$ might not satisfy a property we desire it to have – such as transitivity, for example – but we may be able to find a “best approximation” to R that does have a desired property. In the case of transitivity, this “best approximation” is called the *transitive closure* of the relation R . A similar definition also exists, for example, for the property of symmetry of endorelations.

Definition 4.26 (Transitive closure)

Let R be an endorelation on a set \mathbf{A} , and consider the set

$$\{S \subseteq \mathbf{A} \times \mathbf{A} \mid R \subseteq S \text{ and } S \text{ is transitive}\} \quad (37)$$

of transitive relations on \mathbf{A} containing R . (Note that it is non-empty, because the relation $S = \mathbf{A} \times \mathbf{A}$ contains R and is transitive.) The *transitive closure* R^+ of R is

$$R^+ = \bigcap \{S \subseteq \mathbf{A} \times \mathbf{A} \mid R \subseteq S \text{ and } S \text{ is transitive}\}. \quad (38)$$

Remark 4.27. It is straightforward to check that the intersection of any number of transitive relations on a set \mathbf{A} is again a transitive relation; in particular therefore R^+ is transitive. This is at the core of why the above definition is useful. An analogous definition using inclusion in place of containment would not work well, because in general the union of transitive relations is not necessarily again transitive.

Remark 4.28. The transitive closure of an endorelation R on a set \mathbf{A} is the unique relation R^+ on \mathbf{A} satisfying the following conditions:

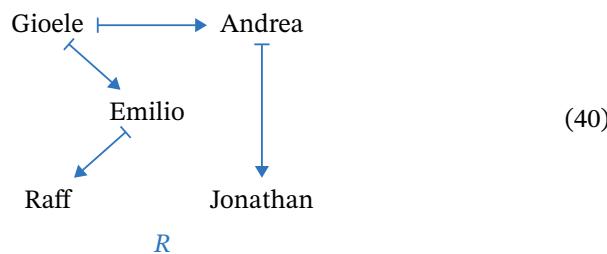
1. $R \subseteq R^+$;
2. R^+ is transitive;
3. if S is a relation on \mathbf{A} that satisfies the previous two points, then $R^+ \subseteq S$.

We might translate these conditions as follows: the first one is saying that R^+ approximates R (via containment); the second one states that R^+ has the property of interest to us here; the third one says that R^+ is the “best” approximation among such relations.

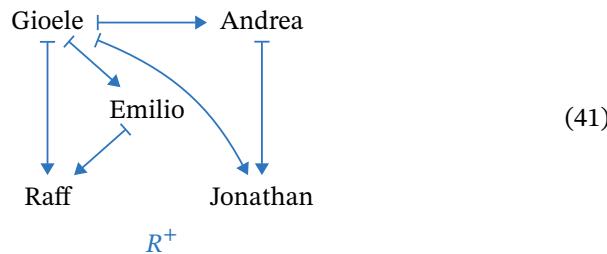
Example 4.29. Consider a relation R on a set of people

$$\mathbf{A} = \{\text{Gioele, Andrea, Jonathan, Emilio, Raff}\}, \quad (39)$$

which describes who invites which friend to a party:



In other words, Gioele invites Andrea and Emilio, Andrea invites Jonathan, and Emilio invites Raff. The transitive closure R^+ of R describes all invitations resulting from transitivity.



In particular, Gioele invites Jonathan and Raff as well, due to the fact that Andrea invites Jonathan, and Emilio invites Raff.

4.8. Equivalence relations

Equivalence relations are a way to group together elements of a set which wish to think of as “the same” in some respect. They appear all over mathematics.

Definition 4.30 (Equivalence relation)

An endorelation $R : \mathbf{A} \rightarrow \mathbf{A}$ is an *equivalence relation* if it is symmetric, reflexive, and transitive.

If R is an equivalence relation, we often write $x \sim_R y$, or simply $x \sim y$, instead of $x R y$.

Example 4.31. The relation “equals” on \mathbb{N} is an equivalence relation. The relation “less than or equal” on \mathbb{N} is not.

Example 4.32. The relation on \mathbb{N} “differing by a multiple of 3”

$$\frac{x R y}{(x - y) \bmod 3 = 0}. \quad (42)$$

is an equivalence relation. Indeed, the relation is reflexive, and symmetric. Furthermore, if x differs by a multiple of 3 from y and y differs by a multiple of 3 from z , then x differs by a multiple of 3 from z (transitivity).

Example 4.33. The relation “has the same birthday as” on the set of all people is an equivalence relation. It is symmetric, because if Anna has the same birthday as Bob, then Bob has the same birthday as Anna. It is reflexive because every person has the same birthday as themselves.

Example 4.34. Let $f : \mathbf{A} \rightarrow \mathbf{B}$ be a function between sets. The following defines an equivalence relation \sim_f on the set \mathbf{A} :

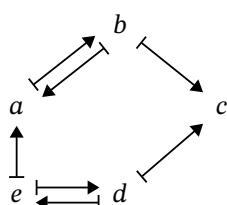
$$\frac{x \sim_f y}{f(x) = f(y)}. \quad (43)$$

Definition 4.35 (Partition)

A *partition* of a set \mathbf{A} is a collection $\{\mathbf{A}_i\}_{i \in I}$ of subsets $\mathbf{A}_i \subseteq \mathbf{A}$ such that

1. $\mathbf{A}_i \cap \mathbf{A}_j = \emptyset \quad \forall i \neq j$;
2. $\bigcup_{i \in I} \mathbf{A}_i = \mathbf{A}$.

Remark 4.36. There is a one-to-one correspondence between equivalence relations on a set \mathbf{A} and partitions on \mathbf{A} .



Example 4.37. An example of partitions can be shown through information networks. An exemplary network is depicted in Fig. 16. Here, nodes represent data centers, and the arrows represent information flows. We say that data centers a and b are equivalent ($x \sim y$) if and only if there is a path from x to y and a path from y to x . In Fig. 16, we have that $a \sim b$, $e \sim d$, and also every center is equivalent with itself.

Figure 16.

Graded exercise B.11 (CountingEquivalenceRelations)

Let $\mathbf{A} = \{1, 2, 3, 4\}$. How many different equivalence relations are there on

A? Explain how you found your answer.

Solutions to selected exercises

Solution of Exercise 2. We start by the union operation \cup . We need to prove

$$A \cup (B \cup C) = (A \cup B) \cup C.$$

We have:

$$\begin{aligned} x &\in A \cup (B \cup C) \\ \Leftrightarrow x &\in A \vee (x \in B \vee x \in C) \\ \Leftrightarrow (x &\in A \vee x \in B) \vee x \in C \\ \Leftrightarrow x &\in (A \cup B) \cup C. \end{aligned}$$

We now continue with the intersection operation \cap . We need to prove

$$A \cap (B \cap C) = (A \cap B) \cap C.$$

We have:

$$\begin{aligned} x &\in A \cap (B \cap C) \\ \Leftrightarrow x &\in A \wedge (x \in B \wedge x \in C) \\ \Leftrightarrow (x &\in A \wedge x \in B) \wedge x \in C \\ \Leftrightarrow x &\in (A \cap B) \cap C. \end{aligned}$$

Essentially, we have used the associativity of the \wedge and \vee connectives.

Solution of Exercise 3. We start by the union operation \cup . We need to prove that $A \cup B = B \cup A$. We have:

$$\begin{aligned} x &\in A \cup B \\ \Leftrightarrow x &\in A \vee x \in B \\ \Leftrightarrow x &\in B \vee x \in A \\ \Leftrightarrow x &\in B \cup A. \end{aligned}$$

We continue with the intersection operation \cap . We need to prove that $A \cap B = B \cap A$. We have:

$$\begin{aligned} x &\in A \cap B \\ \Leftrightarrow x &\in A \wedge x \in B \\ \Leftrightarrow x &\in B \wedge x \in A \\ \Leftrightarrow x &\in B \cap A. \end{aligned}$$

Essentially, we have used the commutativity of the \wedge and \vee connectives.

Solution of Exercise 4. We have:

1. 2.
2. 4.
3. 8.
4. 1.

In general, the size of $\text{Pow } A$ is 2 to the power of the size of A .

Solution of Exercise 5. We analyze the functions one by one.

1. $x + 10 = y + 10$ implies $x = y$, therefore the function is injective. Clearly, for any real number y there is an x such that $f(x) = y$. Therefore the *mapa* is surjective. It follows that f is bijective.

2. g is not injective, since $g(x) = g(-x) = x^2$. Also, it clearly is not surjective, since for any negative real y there is no x such that $g(x) = y$.
3. When $x, y \in \mathbb{N}$, $x^2 = y^2$ implies $x = y$, meaning that h is injective. h is not surjective, since, e.g., there is no $x \in \mathbb{N}$ such that $h(x) = 3$.
4. Clearly, k is injective. It is not surjective, since, e.g., there is no $x \in \mathbb{N}$ such that $k(x) = 2$.

Solution of Exercise 6. *Per absurdum*, assume that a map $f : \mathbf{A} \rightarrow \mathbf{B}$ possesses two *different* inverses $g, h : \mathbf{B} \rightarrow \mathbf{A}$. Following the definition of inverse, we have

$$f \circ g = f \circ h = \text{id}_{\mathbf{A}}, \quad (44)$$

and

$$g \circ f = h \circ f = \text{id}_{\mathbf{B}}. \quad (45)$$

Now, we can write

$$\begin{aligned} h &= \text{id}_{\mathbf{B}} \circ h \\ &= (g \circ f) \circ h \\ &= g \circ (f \circ h) \\ &= g \circ \text{id}_{\mathbf{A}} \\ &= g, \end{aligned} \quad (46)$$

which contradicts the initial assumption.

Solution of Exercise 7. We show the two directions in turn:

$$\frac{f \text{ isomorphism}}{f \text{ bijective}} \text{ and } \frac{f \text{ bijective}}{f \text{ isomorphism}}. \quad (47)$$

Consider an isomorphism $f : \mathbf{A} \rightarrow \mathbf{B}$ and its inverse $g : \mathbf{B} \rightarrow \mathbf{A}$. Take a $y \in \mathbf{B}$ and let $x = g(y)$. We know that $f(x) = f(g(y)) = (g \circ f)(y) = y$. Therefore, f must be *surjective*. To show injectivity, consider $x, x' \in \mathbf{A}$ such that $f(x) = f(x')$. Let $y = f(x)$ and $y'' = g(y)$. Then, we have

$$\begin{aligned} x' &= \text{id}_{\mathbf{A}}(x') \\ &= (f \circ g)(x') \\ &= g(f(x')) \\ &= g(y) \\ &= y''. \end{aligned} \quad (48)$$

However, we also know

$$\begin{aligned} x &= \text{id}_{\mathbf{A}}(x) \\ &= (f \circ g)(x) \\ &= g(f(x)) \\ &= g(y) \\ &= x''. \end{aligned} \quad (49)$$

Therefore, $x = x'$ and f is injective (and therefore bijective).

Now, consider a map $f : \mathbf{A} \rightarrow \mathbf{B}$ which is bijective. One can define $g : \mathbf{B} \rightarrow \mathbf{A}$ in the following way. Take a $y \in \mathbf{B}$, and since f is surjective (it is bijective), there exists a $x \in \mathbf{A}$ such that $f(x) = y$. Let $g(y) = x$. Since f is injective, x must be unique, meaning that g is well-defined. Now we check that indeed g must be the inverse of f . Consider $x \in \mathbf{A}$ and $y = f(x)$. By definition, $g(y) = x$, and hence $(f \circ g)(x) = g(f(x)) = g(y) = x$, implying $f \circ g = \text{id}_{\mathbf{A}}$. Similarly, take $y \in \mathbf{B}$ and $x = g(y)$. Then, by definition we have $f(x) = y$, and hence $(g \circ f)(y) =$

$f(g(y)) = f(x) = y$, implying $g \circ f = \text{id}_B$. Therefore, f is an isomorphism.

Solution of Exercise 8.

Solution of Exercise 9.

$$\begin{aligned} \text{as : } (\mathbf{A} + \mathbf{B}) + \mathbf{C} &\rightarrow \mathbf{A} + (\mathbf{B} + \mathbf{C}), \\ \begin{cases} \langle 1, \langle 1, x \rangle \rangle &\mapsto \langle 1, x \rangle \\ \langle 1, \langle 2, y \rangle \rangle &\mapsto \langle 2, \langle 1, y \rangle \rangle \\ \langle 2, z \rangle &\mapsto \langle 2, \langle 2, z \rangle \rangle. \end{cases} & (50) \end{aligned}$$

Solution of Exercise 10.

Solution of Exercise 11.

Solution of Exercise 12. Yes. Such a relation would be of the form $R \subseteq \emptyset \times B = \emptyset$, where B here is an arbitrary set. In this situation, $R = \emptyset$ is a relation $\emptyset \rightarrow B$.

Solution of Exercise 13. Yes. Such a relation would be of the form $R \subseteq A \times \emptyset = \emptyset$, where A is an arbitrary set. In this situation, $R = \emptyset$ is a relation $A \rightarrow \emptyset$.

PART C. ORDER



5. Posets	85
6. Constructing posets	101
7. Monotonicity	109
8. Poset bounds	123



5. Posets

Life is about trade-offs: there is seldom a uniformly best outcome; rather we need to reason with incomparable attributes. Partially ordered sets (posets) are the mathematical structure used to reason about trade-offs. They are also important as one of the simplest examples of categories.

5.1 Trade-offs	86
5.2 Ordered sets	90
5.3 Counting orders	95
5.4 Power poset	96
5.5 Chains and Antichains	97
5.6 Measuring posets	99

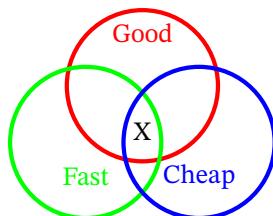


Figure 1.

5.1. Trade-offs

Trade-offs characterize all engineering disciplines.

Do you want to build something?

- ▷ If you want it to be done well and quickly, it won't be cheap.
- ▷ If you want it to be done well and cheaply, it won't be quick.
- ▷ If you want it cheaply and quickly, it won't be done well.

To characterize engineering trade-offs, we will use the mathematical structure of partial orders. In the next section, we will explore some examples, to better contextualize trade-offs.

Functionality and resources

In this section, we introduce concepts which will be important throughout the book, when talking about theories of design. We distinguish semantically between **functionalities** and **requirements/costs**. In general, you prefer **functionalities** to be “large” (Fig. 2b) and **requirements/costs** to be “small” (Fig. 2a).



Figure 2.

We think of three achievable accuracy plots (Fig. 3).

- ▷ In Fig. 3a we plot trade-offs in costs and add a “feasibility” curve. Everything above this curve is feasible and will cost more than what is *on* the curve.
- ▷ In Fig. 3b we plot trade-offs in functionalities and add a “feasibility” curve. Everything below the curve is feasible, but is below the “standards” required by the curve.
- ▷ In Fig. 3c we plot functionality and resource together, representing the trade-offs between “how good a product is” and “how much one needs to pay for it”. Feasible pairs are represented via the feasibility curve. Everything above the curve will be feasible (by paying more).

It is a good exercise to open any engineering book, find the graphs talking about “achievable” performance and “resources” needed, and classify into one of the ones reported in Fig. 3.

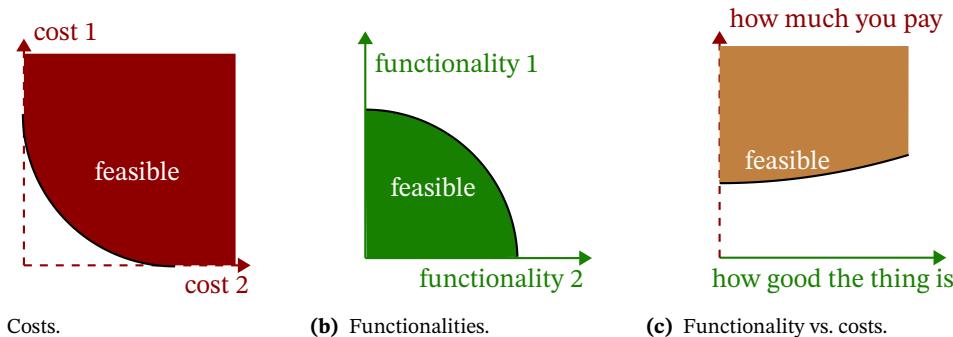


Figure 3.

Trade-offs for the human body

The human body is a great example of trade-offs and adaptability. Consider sports: when looking at different disciplines, various physical abilities are desired and trade-offs between them characterize athletes.

For instance, we can think about trade-offs between **speed** and **strength** for humans (Fig. 4). These are functionalities, which different athletes might want to maximize. Consider Usain Bolt, who owns the 100 meters, 200 meters, and 4×100 meters relay world records. Without doubts, in the human speed-strength trade-off curve he positions himself close to the highest achievable speeds. At the same time, however, Usain Bolt is not among the strongest men in the world. To see the other end of the curve, we need to introduce Oleksii Novikov, who won the 2020 World's Strongest Man competition. Similarly to Bolt, he is among the best in his discipline, reaching very high strength. Again, the speed-strength trade-off implies that Oleksii cannot be among the fastest men in the world, if he wants to be among the strongest ones.

In this case, the resource needed to obtain speed or strength is the amount of **training** (Fig. 5). If we want to relate the invested training and the resulting strength reached by the athletes, we will notice that with a lot of training, Novikov will improve his results, approaching perfection. On the other hand, the kind of training Bolt undergoes is not optimizing strength, and therefore his results will be less effective towards maximizing strength.

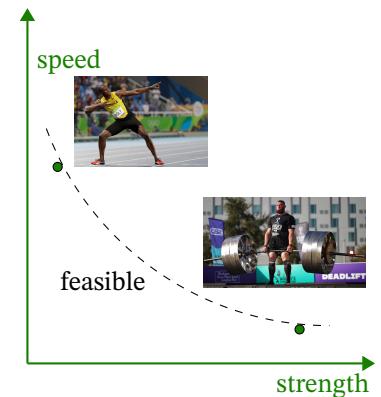


Figure 4.

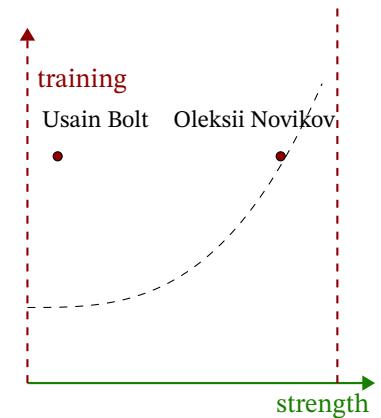


Figure 5.

Masks



Figure 6.: Ordering masks by protection levels

Orders give us a rich way to describe products under various lenses. Recently, we all needed to become experts of protective masks. In this section, we will show various ways in which we can order the latter by functionality.

By first thinking about the effectiveness of the mask in protecting the wearer from a virus, we can order masks as in Fig. 6. In general masks are classified following their filter abilities and inward leakages. The FFP1 class filters at least 80 % of airborne particles and allows less than 22 % inward leakage. The FFP2 class filters at least 96 % of airborne particles and allows less than 8 % inward leakage, and the FFP3 class filters at least 99 % of airborne particles and allows less than 2 % inward leakage.

Obviously, based on the protection level, the most performant in Fig. 6 is FFP3, and the worst is the fashion one. However, this is not the only way in which we can classify masks. If, for instance, we want to consider a functionality “how much does the mask say about the wearer”, we can order the masks differently. Arguably, the ordering could look like the one in Fig. 7a.

Indeed, choosing a fashion mask might say that the wearer cares more about aesthetics than safety, and choosing a FFP3 highlights responsible behaviors, care, and research in masks models.

Similarly, we could order masks based on different performance criteria, adding the functionality “how much does it protect others?” (Fig. 7b).

On the other hand, we could think about the trade-offs between the mask performance and its cost, presenting a functionality-resource plot (Fig. 7c).

More performant masks are typically more expensive, and the fashion mask will be probably the least performance and most expensive.

This example once again highlights the flexibility and richness of the “orders approach”. This will be much more evident in the next example.

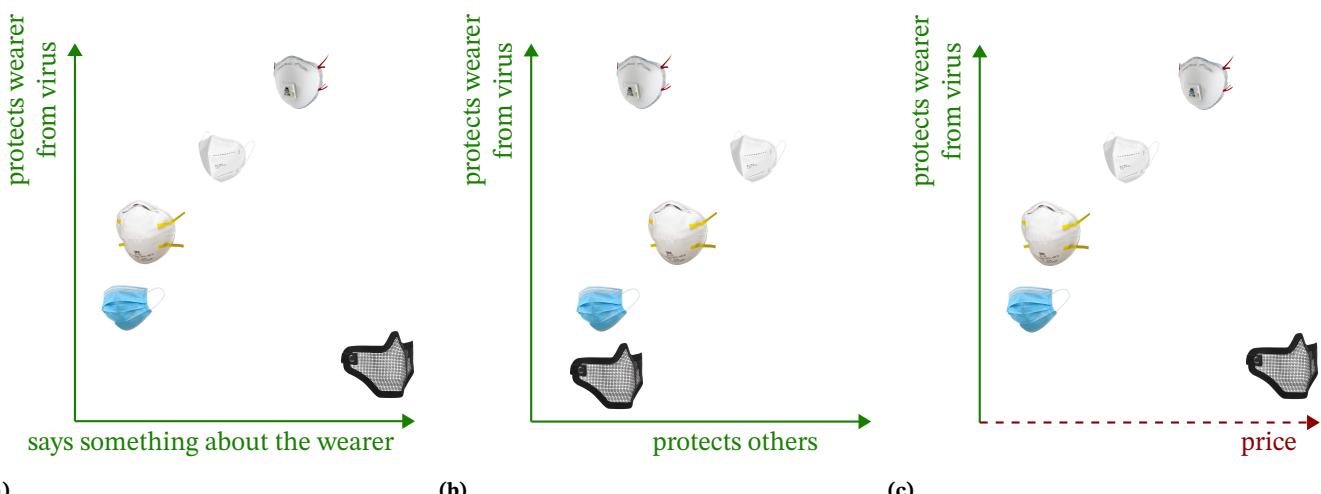


Figure 7.: Ordering masks by other considerations

Hats and headphones

Another good example of ordering of multiple functionalities and costs is the one of headphones. Consider a set of headphones and order them based on their abilities to “keep warm” and to “reproduce music” (Fig. 8a).

Clearly, these two functionalities represent different objectives and diverse product ranges will satisfy them in different ways. For instance, winter hats clearly cannot reproduce music, but keep very warm. On the other hand, large headphones are the best in reproducing music, but cannot keep as warm as winter hats. Functionalities come at a cost. For instance, we could plot the trade-off between “keep warm” and price (Fig. 8b). Other interesting costs could be expressed via the frequency of charging (Fig. 8d) or the hassle of dealing with wires (Fig. 8c).

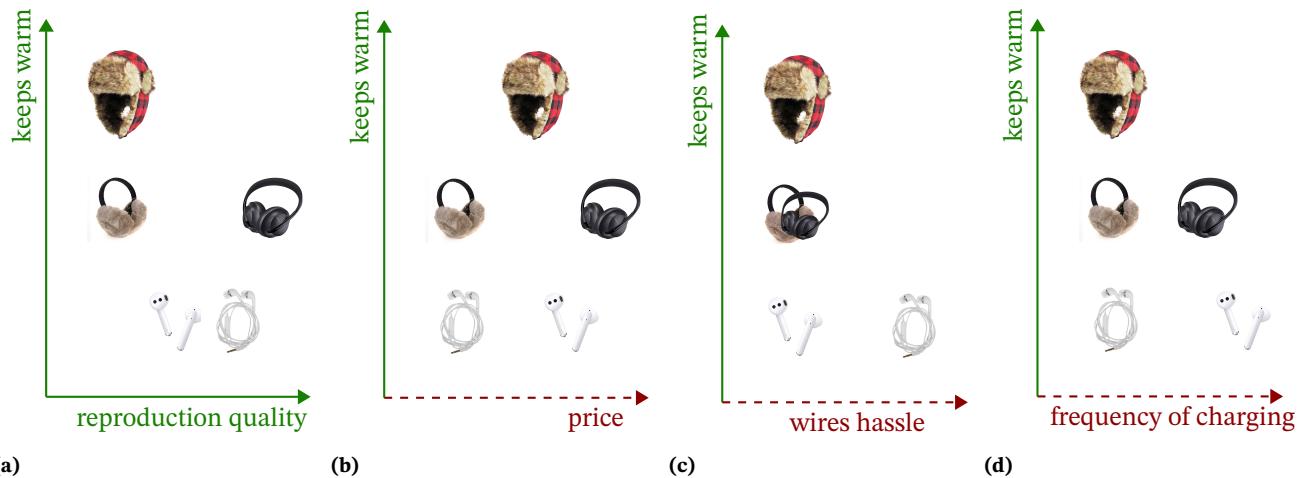


Figure 8.: Ordering hats and headphones

The law of successful products

By considering all the aforementioned characteristics together

$$(\text{keeps warm} \times \text{reproduction quality}) \times (\text{price} \times \text{frequency of charging} \times \text{wires hassle}), \quad (1)$$

no product dominates another.

This is the *law of successful products*. At equilibrium, in an efficient and free market, no product completely dominates another by both functionality and costs. Otherwise, the dominated product would not sell. Once we *specify the design purpose* and the related constraints, we can (partially) order products.

5.2. Ordered sets

So far, the discussion has been purely qualitative. In this section, we introduce pre-orders, partial orders, and total orders. Davey and Priestley [3] and Roman [24] are possible reference texts.

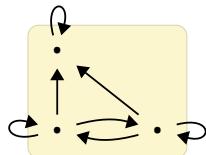


Figure 9.: A pre-order represented as a graph.

Pre-orders

A pre-order is a set together with a binary relation that is both reflexive (Def. 4.19) and transitive (Def. 4.24).

Definition 5.1 (Pre-ordered set)

A *pre-ordered set* is a tuple $P = \langle P, \leq_P \rangle$, where P is a set, called the *carrier set* or *underlying set*, together with a relation \leq_P that is reflexive and transitive.

An example of a pre-ordered set represented as a graph is shown in Fig. 9. In the graph representation of a pre-order P , we draw an arrow between x and y if $x \leq_P y$.

Example 5.2. The reachability relationship in any directed graph (potentially including cycles) is a pre-order. The pre-order P is defined as follows. The set P is the set of nodes of the graph. Take any two nodes $x, y \in P$. One has $x \leq_P y$ if and only if there is a path from x to y in the directed graph. There is always a path from a node to itself (reflexivity), and given a path from x to y , and one from y to z , we know that there is a path from x to z (transitivity).

Exercise 14. Consider the set $P = \{x, y, z\}$. Which of the following are pre-orders? Why?

1. $P = \{\langle x, x \rangle, \langle x, y \rangle, \langle y, x \rangle, \langle y, y \rangle\}$.
2. $Q = \{\langle x, y \rangle, \langle y, z \rangle, \langle z, x \rangle\}$.

See solution on page 131.

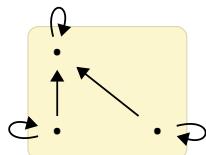


Figure 10.: A partial order represented as a graph.

Partial orders

By adding the condition of *antisymmetry* (Def. 4.18) to a pre-order, we obtain a partially-ordered set.

Definition 5.3 (Partially ordered set)

A pre-ordered set $P = \langle P, \leq_P \rangle$ is a *partially-ordered set* (poset) if the relation \leq_P is antisymmetric.

An example of a partially ordered set represented as a graph is shown in Fig. 10. By comparing this with Fig. 9, we notice that the double-headed arrow is not allowed anymore (indeed, its existence would imply that source and target of the arrow are the same element in the poset).

Exercise 15. Does the reachability relationship in any directed graph define a poset? Why? If not, can you modify the initial statement to make it work?

See solution on page 131.

Example 5.4. The following defines a partial order \leq on the set of natural numbers \mathbb{N} . Define, for all $x, y \in \mathbb{N}$,

$$x \leq y \quad \text{if, and only if} \quad x \text{ divides } y. \quad (2)$$

By definition, a natural number x divides another natural number y if there exists some other natural number z such that $xz = y$. The notation for “ x divides y ” is $x|y$.

Graded exercise C.1

Consider the set \mathbf{A} of natural numbers which divide the number 60, and equipped with the partial order defined by

$$x \leq y \quad \text{if, and only if} \quad x \text{ divides } y \quad (3)$$

for all $x, y \in \mathbf{A}$. Draw the Hasse diagram of this partially ordered set.

Graded exercise C.2 (PolynomialDivisibility)

Let \mathbf{A} be the set of all polynomials with coefficients in \mathbb{R} . Recall that a polynomial p divides a polynomial q if there exists a polynomial m such that $p \cdot m = q$. If p divides q we denote this by $p|q$. Divisibility defines an endorelation on \mathbf{A} by saying p is related to q iff $p|q$. Does this define a pre-order structure on \mathbf{A} ? Does this define a poset structure on \mathbf{A} ? Justify your answer.

Total orders

By imposing *totality* (Def. 4.20), we obtain a total order.

Definition 5.5 (Totally ordered set)

A partially ordered set $P = \langle P, \leq_P \rangle$ is a *totally ordered set* if the relation \leq_P is total.

An example of a totally ordered set represented as a graph is reported in Fig. 11.

Example 5.6 (Reals). The real numbers \mathbb{R} form a totally ordered poset $\langle \mathbb{R}, \leq \rangle$ with order relation given by the usual ordering.

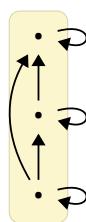


Figure 11.: A total order.

Hasse diagrams

We can represent partial orders in various ways. We now take a proxy partially ordered set and represent it using different conventions. Consider $P = \langle P, \leq_P \rangle$, where $P = \{x, y, z\}$ and $x \leq_P y, y \leq_P z$. First, we could represent this using the same visualization we had for relations (Fig. 12a).

However, this is quite heavy, and does not exploit the fact that partial orders are endorelations. Therefore, we could think to only draw the carrier set once, and to drop the order relations arising from reflexivity (Fig. 12b).

However, the arrow from x to y is implicit in partial orders, because of transitivity.

A *Hasse diagram* is an economical (in terms of arrows) way to visualize a poset. In a Hasse diagram elements are points, and if $p \leq_P q$ then p is drawn lower than q and with an edge connected to it, if no other point is in between (Fig. 12c). Hasse diagrams are directed graphs.

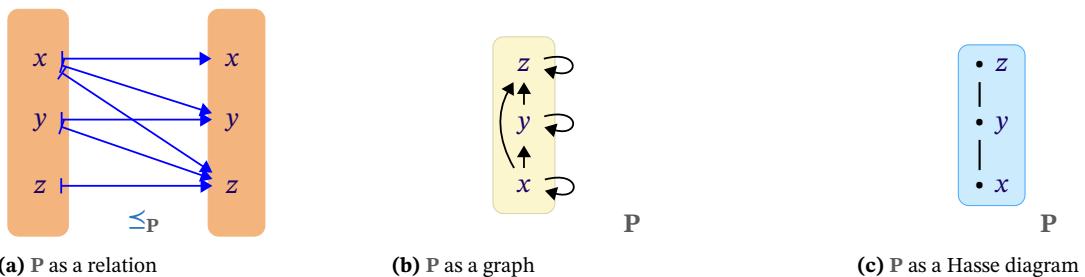


Figure 12.: Three different representations for a poset

Example 5.7 (Discrete partially ordered sets). Every set P can be considered as a discrete poset $P = \langle P, = \rangle$ using equality as the partial order. (Notice that equality is symmetric, transitive, and antisymmetric.) When visualized as a Hasse diagram, discrete posets are a collection of points (Fig. 13).

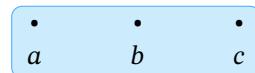


Figure 13.

Definition 5.8 (Boolean poset Bool)

The set of booleans $\text{Bool} = \{\perp, \top\}$ can be made into a poset by choosing the order $\perp \leq_{\text{Bool}} \top$. This is equivalent to using “ \Rightarrow ” as a relation. We obtain the poset

$$\text{Bool} := \langle \text{Bool}, \Rightarrow \rangle. \quad (4)$$



Figure 14.

Example 5.9 (Qualitative information). In the example of the battery choice, both mass and money can be thought of as partially ordered sets. Imagine that you have batteries which are “cheap”, “midrange”, and “expensive”. Clearly, if the partially ordered set represents cost, we can say that cheap \leq midrange \leq expensive. While this is a quantitative judgement (indeed, if I care about cost, I will prefer a cheap battery over a midrange one), it is not a numeric one (cheap could represent a number, but also a range of numbers or just a price category). This can be represented as in Fig. 15.

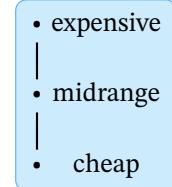
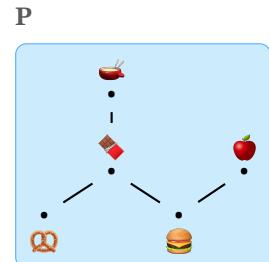
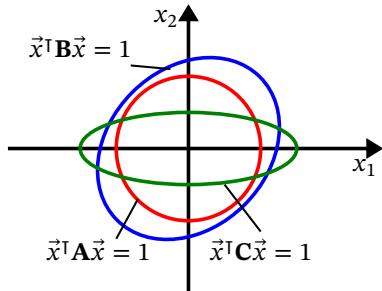


Figure 15.: Possible costs of a battery, epresented as a poset.

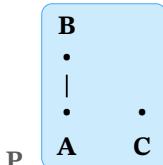
Example 5.10. Consider a poset P representing a person’s food preference over the set $P = \{\text{pretzel}, \text{steak}, \text{diamond}, \text{hamburger}, \text{apple}\}$ with $\text{pretzel} \leq_P \text{diamond}$, $\text{hamburger} \leq_P \text{diamond}$, $\text{diamond} \leq_P \text{steak}$, and $\text{hamburger} \leq_P \text{apple}$. This can be represented with a Hasse diagram as in Fig. 16.

Figure 16.: Example of Hasse diagram of P .

Example: positive definite matrices as ellipsoids



(a) Example of ellipses representing positive definite matrices.



(b) Example of order between positive semi-definite matrices.

Figure 17.

Definition 5.11 (Positive definite matrix)

Definition 3.11 (Positive definite matrix)
A symmetric matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is *positive definite* if $\vec{x}^\top \mathbf{M} \vec{x} > 0$ for all non-zero $\vec{x} \in \mathbb{R}^n$. We call the set of all such matrices PDM(n).

Positive definite matrices have real, positive eigenvalues, which can be interpreted as axes lengths of ellipsoids. Any matrix $\mathbf{A} \in \text{PDM}(n)$ describes an ellipsoid, which can be written as a quadratic equation:

$$\vec{x}^\top \mathbf{A} \vec{x} = 1, \quad \vec{x} \in \mathbb{R}^n. \quad (5)$$

We can define a partial order on n as

$$\overbrace{\vec{x}^\top \mathbf{A} \vec{x}} \leq \vec{x}^\top \mathbf{B} \vec{x} \quad \forall \vec{x} \in \mathbb{R}^n. \quad (6)$$

The order can be interpreted as an inclusion of ellipsoids. Take for instance the matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 3/4 & -1/8 \\ -1/8 & 3/4 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1/2 & 0 \\ 0 & 2 \end{bmatrix}. \quad (7)$$

The order \mathbf{P} on the set $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ is reported in Fig. 17b, and it is easily explained via Fig. 17a. The ellipse representing \mathbf{A} (in red) is included by the one representing matrix \mathbf{B} (in blue), but not by the one representing matrix \mathbf{C} (in green). Furthermore, the one representing \mathbf{B} includes the one representing \mathbf{C} .

5.3. Counting orders

Let's count the number of posets.

If there is only one element, there is only one way to order it (Fig. 18a).

With a 2-elements set, there are 2 posets (panel b), “up to isomorphism”, that is, if we do not care about the labels of points.

On 3-elements sets, we have 5 posets (panel c).

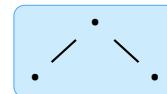
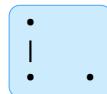
On 4-elements sets, we have 16 posets (panel d).



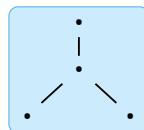
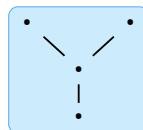
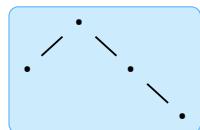
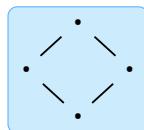
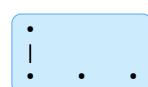
(a) The singleton poset.



(b) All posets on 2-elements sets, up to isomorphism.



(c) All posets on 3-elements sets, up to isomorphisms.



(d) All posets on a 4-element set, up to isomorphism.

Figure 18.

5.4. Power poset

We have introduced the concept of power set in Section 3.3. There is a natural order on subsets, given by set inclusion. We can thus define the *power poset*.

Definition 5.12 (Power poset)

Given a set A , define the *power poset* $\text{Pow } A = \langle \text{Pow } A, \subseteq \rangle$ by ordering the subsets in its power set $\text{Pow } A$ by inclusion.

A subset S precedes T if $S \subseteq T$:

$$\frac{\mathbf{S} \leq_{\text{Pow } A} \mathbf{T}}{\mathbf{S} \subseteq \mathbf{T}}. \quad (8)$$

This is illustrated in Fig. 19 for sets of 0, 1, 2, 3 elements.

Exercise 16. Check formally that $\leq_{\text{Pow } A}$ defined in (8) is a partial order.

See solution on page 131.

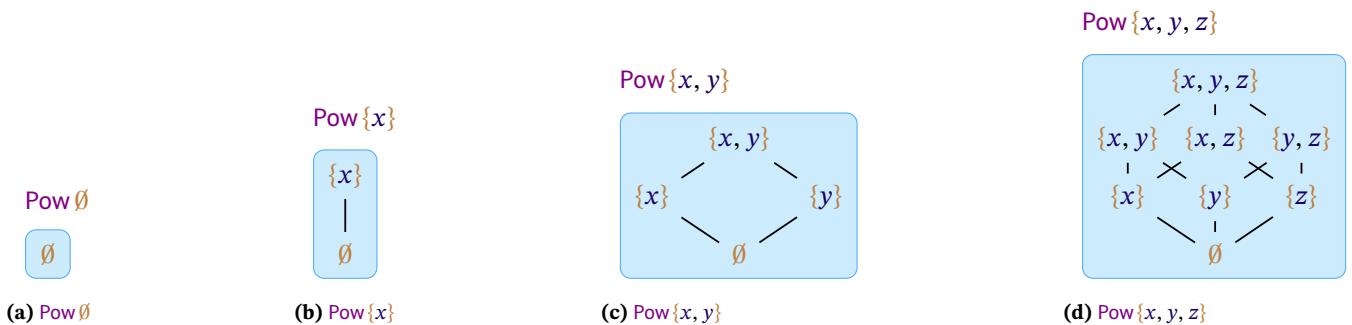


Figure 19. Power set as a poset.

5.5. Chains and Antichains

There are two special types of subsets of a poset: chains and antichains. Their definitions are dual.

Definition 5.13 (Chain in a poset)

Given a poset $P = \langle P, \leq_P \rangle$, a *chain* is a subset $S \subseteq P$ such that any two elements of S are comparable:

$$\frac{x, y \in S}{(x \leq_P y) \vee (y \leq_P x)}. \quad (9)$$

Definition 5.14 (Antichain in a poset)

An *antichain* is a subset S of a poset where no two distinct elements are comparable:

$$\frac{x, y \in S \quad x \leq_P y}{x = y}. \quad (10)$$

Remark 5.15. Note that the empty set \emptyset is both a chain and an antichain.

We denote the set of antichains of a poset P by $\text{Anti } P$.

Example 5.16 (Chains and antichains in a power poset). Consider the poset in Fig. 19d. Examples of chains are

$$\{\emptyset, \{x\}\} \quad \text{and} \quad \{\emptyset, \{y\}, \{y, z\}, \{x, y, z\}\}, \quad (11)$$

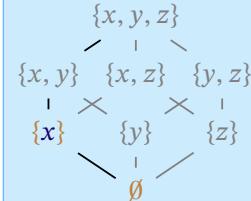
depicted in Fig. 20a and Fig. 20b, respectively.

Examples of antichains are

$$\{\{x\}, \{y\}\} \quad \text{and} \quad \{\{x, y\}, \{x, z\}, \{y, z\}\}, \quad (12)$$

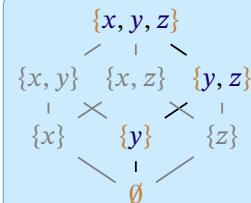
depicted in Fig. 20c and Fig. 20d, respectively.

$\text{Pow}\{x, y, z\}$



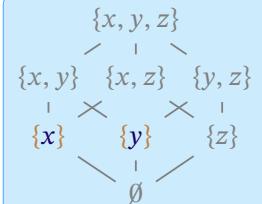
(a) A chain.

$\text{Pow}\{x, y, z\}$



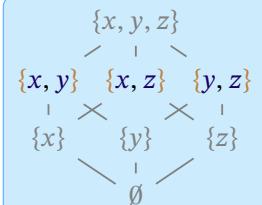
(b) A chain.

$\text{Pow}\{x, y, z\}$



(c) An antichain.

$\text{Pow}\{x, y, z\}$



(d) An antichain.

Figure 20.: Examples of chains (a-b) and antichains (c-d) in the poset $\text{Pow}\{x, y, z\}$.

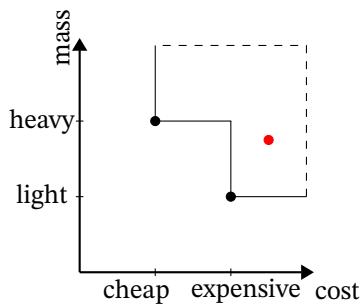


Figure 21.: Example of discrete antichains.

Example 5.17. In the context of battery choices, consider the diagram in Fig. 21. The black markers represent an antichain of choices

$$\{\langle \text{cheap, heavy} \rangle, \langle \text{expensive, light} \rangle\}. \quad (13)$$

It is a set of pairs because they do not dominate each other: one is cheaper, but is heavier, and the other is more expensive, but lighter, making them incomparable. If a battery with the properties as the red marker existed (very expensive, between light and heavy), that would be an element that cannot be part of the antichain, since it would be dominated by $\langle \text{expensive, light} \rangle$.

Similarly, we could think of a continuous law which relates battery cost and mass. Assume that cheap means 10 CHF, expensive means 20 CHF, light means 250 g, and heavy means 500 g. For instance, consider the antichain given by mass = $500 - 25 \cdot \text{cost}$, with maximum possible cost 20 CHF (Fig. 22).

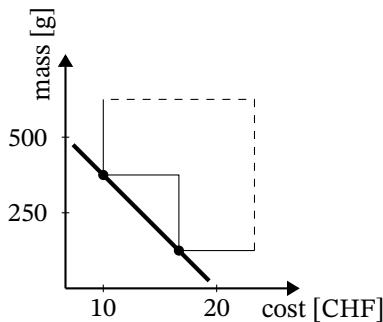


Figure 22.: Example of continuous antichains.

P

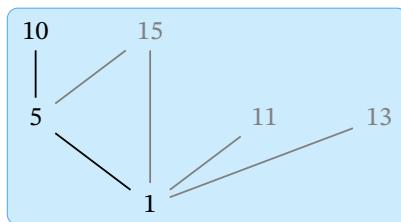


Figure 23.

Example 5.18. Consider the poset $P = \langle P, \leq_P \rangle$ where $(p \leq_P q)$ if p is a divisor of q and $P = \{1, 5, 10, 11, 13, 15\}$. A chain of P is $\{1, 5, 10\}$ (Fig. 23). An antichain of P is $\{10, 11, 13, 15\}$ (Fig. 24).

P

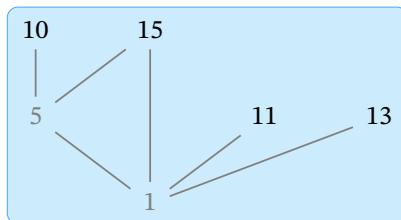


Figure 24.

5.6. Measuring posets

We can define two measurements for a poset: the height and the width. These measurements allow to quantify the performance of several algorithms we will see in the latter parts of the book.

Definition 5.19 (Width of a poset)

The *width of a poset*, denoted $\text{width}(\mathbf{P})$, is the maximum cardinality of an antichain in \mathbf{P} .

Definition 5.20 (Height of a poset)

The *height of a poset*, denoted $\text{height}(\mathbf{P})$, is the maximum cardinality of a chain in \mathbf{P} .

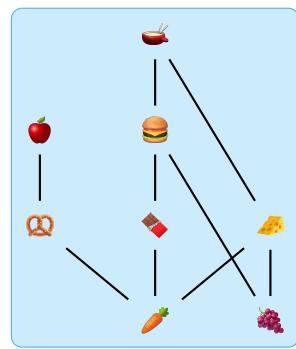
Note that an empty poset has exactly one chain and one antichain: the empty set. Therefore, the height and width are zero.

Example 5.21. Consider the poset \mathbf{P} in Fig. 25a. The longest antichains of \mathbf{P} are $\{\text{apple}, \text{hamburger}, \text{cheese}\}$, $\{\text{apple}, \text{chocolate bar}, \text{cheese}\}$, $\{\text{pretzel}, \text{chocolate bar}, \text{cheese}\}$, $\{\text{apple}, \text{chocolate bar}, \text{grapes}\}$, $\{\text{pretzel}, \text{chocolate bar}, \text{grapes}\}$, and $\{\text{pretzel}, \text{hamburger}, \text{cheese}\}$. Therefore,

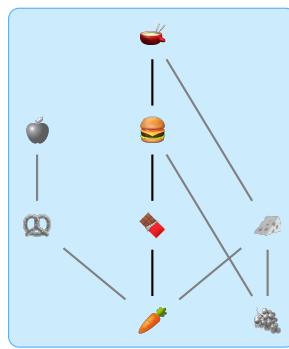
$$\text{width}(\mathbf{P}) = 3. \quad (14)$$

The longest chain in the poset is given by $\{\text{carrot}, \text{chocolate bar}, \text{hamburger}, \text{snack}\}$, and therefore

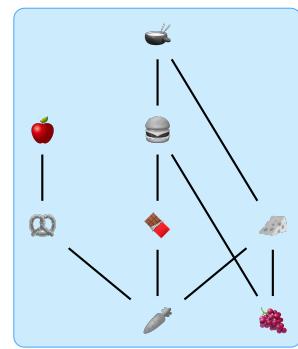
$$\text{height}(\mathbf{P}) = 4. \quad (15)$$



(a)



(b) The longest chain



(c) One of the largest antichains

Figure 25.: Example for height and width of a poset.

Graded exercise C.3 (`MeasurePowerPoset`)

Let \mathbf{A} be a finite set with n elements. Obtain an expression (without proof) for

1. $\text{width}(\text{Pow } \mathbf{A})$;
2. $\text{height}(\text{Pow } \mathbf{A})$.



6. Constructing posets

In this chapter we look at a few standard recipes how we can construct posets from sets or other posets.

6.1 Product of posets	102
6.2 Disjoint union of posets	104
6.3 Opposite of a poset	105
6.4 “Twisted” poset of intervals	106
6.5 Arrow poset of intervals	108

6.1. Product of posets

Just like the product of sets, we can construct the product of posets. That is a poset with the underlying set being the product of the underlying sets.

Definition 6.1 (Product of posets)

Given posets $P = \langle P, \leq_P \rangle$ and $Q = \langle Q, \leq_Q \rangle$, the *product poset*

$$P \times Q = \langle P \times Q, \leq_{P \times Q} \rangle, \quad (1)$$

is the set $P \times Q$ equipped with the order $\leq_{P \times Q}$ given by

$$\frac{\langle p_1, q_1 \rangle \leq_{P \times Q} \langle p_2, q_2 \rangle}{(p_1 \leq_P p_2) \wedge (q_1 \leq_Q q_2)}. \quad (2)$$

Recalling the battery choice example, we have the two posets representing cost and weight. Given that we want to minimize both cost and weight, by considering the cost poset containing elements “cheap”, “midrange”, and “expensive”, and the weight poset containing elements “light”, and “heavy”, we can represent the product as in Fig. 1.

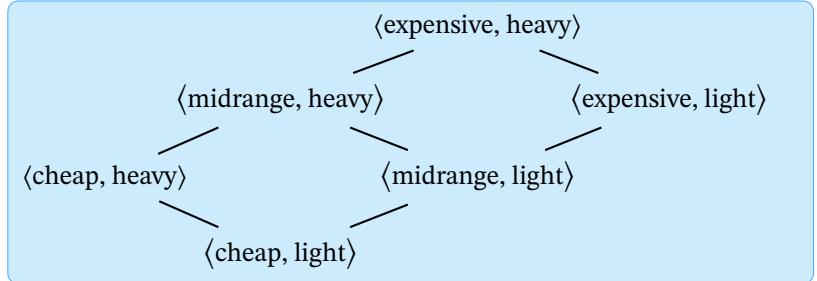


Figure 1.: Product poset of cost and weight for battery choices.

Example 6.2. Consider now two posets and their product, given in Fig. 2.

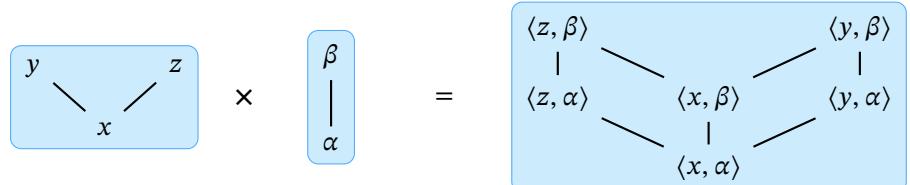


Figure 2.: Product of two posets.

Measuring the product

The following lemma gives expressions for the width and height of the product of two posets.

Lemma 6.3. If P, Q are non-empty finite posets, then we know the height of their product:

$$\text{height}(P \times Q) = \text{height}(P) + \text{height}(Q) - 1 \quad (3)$$

We can derive this bound for the width of the product:

$$\text{width}(P) \cdot \text{width}(Q) \leq \text{width}(P \times Q) \leq \min \{\text{card}(P) \cdot \text{width}(Q), \text{card}(Q) \cdot \text{width}(P)\}. \quad (4)$$

This bound is tight, in the sense that there exist posets that reach this bound.

Proof. The bound (4) can be found in [1]. As for (3), we have the following proof. First, we can construct the longest chain in P :

$$\mathbf{A} = \{p_1, \dots, p_{\text{height}(P)}\}. \quad (5)$$

Furthermore, we can construct the longest chain in Q :

$$\mathbf{B} = \{q_1, \dots, q_{\text{height}(Q)}\}. \quad (6)$$

Out of them, we can construct the chain

$$\mathbf{C} = \{(p_1, q_1), (p_2, q_1), \dots, (p_{\text{height}(P)}, q_1), (p_{\text{height}(P)}, q_2), \dots\}, \quad (7)$$

which has height $\text{height}(P) + \text{height}(Q) - 1$. So we know a lower bound for the height:

$$\text{height}(P \times Q) \geq \text{height}(P) + \text{height}(Q) - 1. \quad (8)$$

Now, consider a chain $\{(p_1, q_1), \dots, (p_n, q_n)\}$ in $P \times Q$. In general, this means that at least a coordinate of (p_i, q_i) must increase in (p_{i+1}, q_{i+1}) . The first coordinate can only increase by $\text{height}(P) - 1$ times, and the second one by $\text{height}(Q) - 1$ times. Summing up, the total number of elements in the chain is *at most* $\text{height}(P) + \text{height}(Q) - 1$:

$$\text{height}(P \times Q) \leq \text{height}(P) + \text{height}(Q) - 1. \quad (9)$$

Because upper and lower bounds are the same, we have an exact expression for the height. Note that this result holds only assuming that P and Q are not empty (for that case, $\text{height}(P \times Q) = 0$). \square

6.2. Disjoint union of posets

Following the pattern, we can define the disjoint union of poset as a poset with the underlying set being the disjoint union of the underlying sets.

Definition 6.4 (Sum of posets)

Given posets $P = \langle P, \leq_P \rangle$ and $Q = \langle Q, \leq_Q \rangle$, their *sum* $P + Q = \langle P + Q, \leq_{P+Q} \rangle$, is the set $P + Q$ equipped with the order \leq_{P+Q} defined by

$$\langle i, x \rangle \leq_{P+Q} \langle j, y \rangle \Leftrightarrow \begin{cases} i = j, \text{ and} \\ \begin{cases} x \leq_P y \text{ if } i = 1, \\ x \leq_Q y \text{ if } i = 2. \end{cases} \end{cases} \quad (10)$$

The expression (10) can be intimidating at first, but all it is saying is that the order relation of the disjoint union is obtained by stitching together the two order relations. No element of P is related to an element of Q , and vice versa.

Example 6.5. Consider the posets P, Q , over the sets $P = \langle \Omega, \leq_P \rangle$ with $\Omega \leq_P \Omega$, and $Q = \langle \text{beer}, \text{cup} \rangle$, with $\text{cup} \leq_Q \text{beer}$. Their disjoint union can be represented as in Fig. 3.

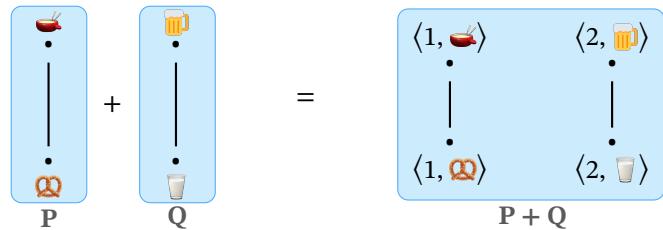


Figure 3.: Disjoint union of posets.

Graded exercise C.4 (MeasurePosetSum)

Prove the following properties:

1. The width of the sum is the sum of the widths:

$$\text{width}(P + Q) = \text{width}(P) + \text{width}(Q). \quad (11)$$

2. The height of the sum is the maximum of the heights:

$$\text{height}(P + Q) = \max(\text{height}(P), \text{height}(Q)). \quad (12)$$

6.3. Opposite of a poset

Definition 6.6 (Opposite of a poset)

The *opposite of a poset* $\mathbf{P} = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ is the poset denoted $\mathbf{P}^{\text{op}} = \langle \mathbf{P}, \leq_{\mathbf{P}}^{\text{op}} \rangle$. It has the same elements as \mathbf{P} , but is equipped with the reverse ordering, in the sense that, for all $x, y \in \mathbf{P}$,

$$\begin{array}{c} x \leq_{\mathbf{P}} y \\ \hline \hline y \leq_{\mathbf{P}}^{\text{op}} x \end{array} \quad (13)$$

For a given $x \in \mathbf{P}$, we will sometimes write x^* do denote its corresponding copy in \mathbf{P}^{op} , in order to emphasize that x and x^* belong to distinct posets. However, often we will not be so pedantic with our notation.

Example 6.7 (Credit and debt). Let us define the set

$$\mathbf{P} = \{0.00, 0.01, 0.02, \dots\} \subseteq \mathbb{R} \quad (14)$$

of all CHF monetary quantities approximated to the cent. From this set we can define two posets, $\mathbf{P}^+ = \langle \mathbf{P}, \leq \rangle$ and $\mathbf{P}^- = \langle \mathbf{P}, \geq \rangle$, that are the opposite of each other. If the context is that, given two quantities 1 CHF and 2 CHF, we prefer 1 CHF to 2 CHF (for example because it is a cost to pay to acquire a component), then we are working in \mathbf{P}^+ , otherwise we are working in \mathbf{P}^- (for example because it represents the price at which we are selling our product). Traditionally, in double-entry ledger systems, the numbers were not written with negative signs, but rather in color: red and black. From this convention we get the idioms “being in the black” and “being in the red”.

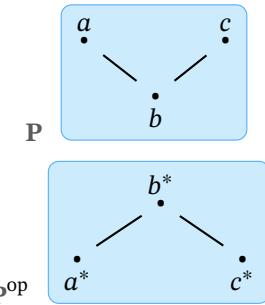


Figure 4.: Opposite of a poset.

6.4. “Twisted” poset of intervals

An interval of time is defined by two numbers: a lower and upper bound. The notation $[a, b]$ usually defines an interval on the real line. We can generalize the notion to an interval of a poset.

Poset intervals

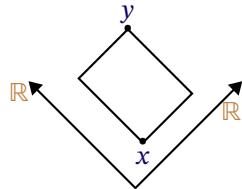


Figure 5.: Poset interval on \mathbb{R}^2 .

Definition 6.8 (Interval)

An *interval* of a poset \mathbb{P} is a pair of elements x, y such that $x \leq_{\mathbb{P}} y$. We also write $[x, y]$, and we identify it with the subset of elements of \mathbb{P} that are bounded above and below by the two elements:

$$[x, y] := \{z \in \mathbb{P} : x \leq_{\mathbb{P}} z \leq_{\mathbb{P}} y\}. \quad (15)$$

Note that, following this definition, the empty set is *not* an interval.

A “twisted” poset of intervals

There are two canonical ways to order poset intervals: a “twisted” version and an “arrow” version. The names are not intuitive at this point: later on, we will see that there exists an “arrow construction” and a “twisted arrow construction” for categories, and they correspond to these constructions when a poset is considered as a category.

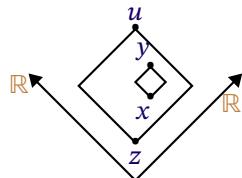


Figure 6.: $[x, y] \leq_{\text{Tw } \mathbb{P}} [z, u]$

Definition 6.9 (“Twisted” poset of intervals $\text{Tw } \mathbb{P}$)

Given a poset \mathbb{P} , we define a “twisted” poset of intervals $\text{Tw } \mathbb{P}$ by ordering the intervals by inclusion:

$$\frac{[x, y] \leq_{\text{Tw } \mathbb{P}} [z, u]}{[x, y] \subseteq [z, u]}. \quad (16)$$

Equivalently we only need to check the bounds:

$$\frac{[x, y] \leq_{\text{Tw } \mathbb{P}} [z, u]}{(z \leq_{\mathbb{P}} x) \wedge (y \leq_{\mathbb{P}} u)}. \quad (17)$$

Exercise 17. Check that the relation defined in Def. 6.9 is indeed a poset.

See solution on page 131.

In general, $\text{Tw } \mathbb{P}$ does not have a top or a bottom.

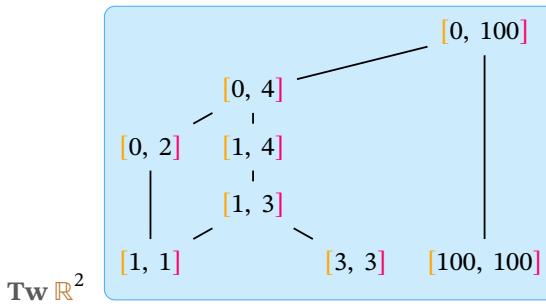


Figure 7.

Set-based filtering

We now look at an example of **set-based filtering**, where filtering refers to online inference (recursive estimation). Suppose that we want to track the value of a quantity $x \in [0, 100]$, without having *a priori* information about x . We are equipped with sensors, which periodically measure the quantity x with some variable precision. At time $t \in \mathbb{R}_{\geq 0}$ they produce an *observation* $y_t : x_t \in [l_t, u_t]$. Also, note that the quantity fluctuates randomly, and we bound its “velocity” to be $\dot{x}_t \in [-1, +1]$ (except at boundaries). At the beginning, our information state \bar{i}_0 could be that $x \in [0, 100]$. At time 0, we get an observation y_0 , that says $x \in [21, 24]$. The new information state can be obtained by “fusing” the two inputs we have received about x . This corresponds to the intersection

$$\begin{aligned} x &\in ([0, 100] \cap [21, 24]) \\ &= [21, 24] \end{aligned} \quad (18)$$

Say we get an observation y_1 which says $x \in [19, 22]$. We now need to take into account the evolution/dynamics of the quantity we are tracking. From the interval $[21, 24]$ we know that the variable could have evolved in $[20, 25]$ (dynamics are bounded with a unit increase/decrease). Therefore, the new information state is given by

$$\begin{aligned} x &\in ([20, 25] \cap [19, 22]) \\ &= [20, 22] \end{aligned} \quad (19)$$

One of the structures which could sustain this kind of inference, is the poset of twisted intervals (Def. 6.9).

The Hasse diagram representing a situation related to this example could be as reported in Fig. 7.

6.5. Arrow poset of intervals

We can order intervals in a second way, which we call “arrow construction”.

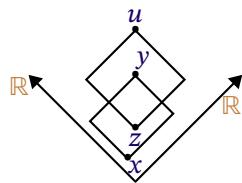


Figure 8.: $[x, y] \leq_{\text{Arr } P} [z, u]$

Definition 6.10 (“Arrow” poset of intervals $\text{Arr } P$)

We define an “Arrow” poset of intervals on the poset P by setting the order:

$$\frac{[x, y] \leq_{\text{Arr } P} [z, u]}{(x \leq_P z) \wedge (y \leq_P u)} \quad (20)$$

This is similar to taking the product of P with itself; however, we are only considering intervals, so we obtain a subposet of $P \times P$.

Exercise 18. Check that the relation defined in Def. 6.10 is indeed a poset.

See solution on page 131.



7. Monotonicity

Life is hard: to obtain more, you need to work more. Monotonicity is the mathematical concept that captures this principle.

7.1 Monotone maps	110
7.2 Monotone relations and design problems	115
7.3 Order on monotone maps	121

7.1. Monotone maps

A monotone map is the generalization to posets of a “non-decreasing” function on real numbers. The function $x \mapsto \max(0, 42x)$ is non-decreasing on the real numbers because

$$\frac{x \leq y}{\max(0, 42x) \leq \max(0, 42y)}. \quad (1)$$

Note that we use “ \leq ” and not “ $<$ ”. “Non-decreasing” is a weaker condition than “increasing”.

The definition of monotone function on a poset is the direct generalization of this concept; the only change is that we use the partial orders at hand, rather than the total order on the reals.

Definition 7.1 (Monotone map)

A *monotone map* between two posets $P = \langle P, \leq_P \rangle$ and $Q = \langle Q, \leq_Q \rangle$ is a function $f : P \rightarrow Q$ that is compatible with the partial-orderings on its source and target in the sense that

$$\frac{x \leq_P y}{f(x) \leq_Q f(y)}. \quad (2)$$

Example 7.2 (The identity is monotone). Given a poset P , the identity function $\text{id}_P : P \rightarrow P$ is a monotone map, since if $x \leq_P y$, then $\text{id}_P(x) = x \leq_P y = \text{id}_P(y)$.

Example 7.3 (Constant functions). Every constant function is a monotone map.

Example 7.4 (Cardinality map). Consider the power poset (Def. 5.12) $\text{Pow } A$ of a finite set A . The cardinality map

$$\text{card} : \text{Pow } A \rightarrow \mathbb{N} \quad (3)$$

is monotone when considered as a map from the poset $\text{Pow } A$ to the poset (\mathbb{N}, \leq) . Figure 1 shows a visualization of this map for the set $A = \{x, y, z\}$. To prove this, recall that in the power poset subsets are ordered by inclusion. Therefore, we need to show that

$$\frac{S \subseteq T}{\text{card}(S) \leq \text{card}(T)}. \quad (4)$$

This is easy to see that, because all elements of S are also in T , the cardinality of S cannot be more than the cardinality of T . Monotonicity depends on the partial

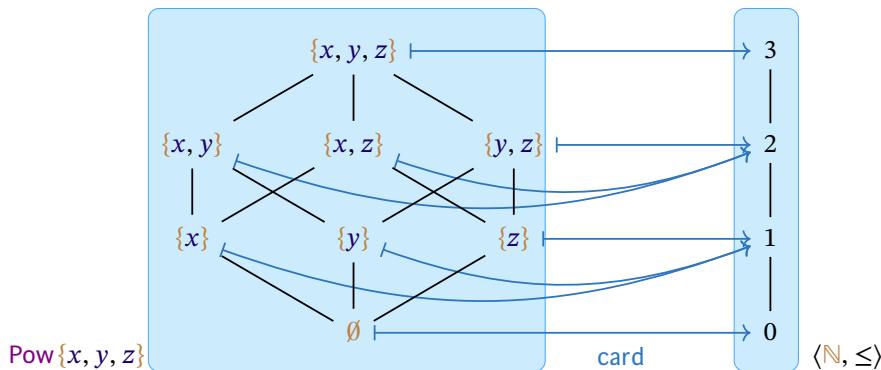


Figure 1.: The cardinality map is a monotone map.

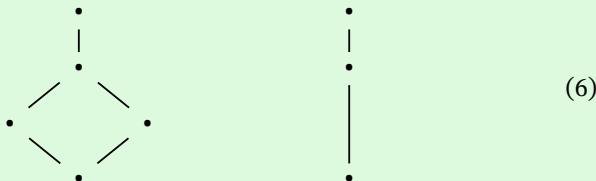
order used on the domain and the codomain. To indicate that a map is monotone, we write it indicating the two posets as the domain/codomain:

$$\text{card} : \langle \text{Pow } \mathbf{A}, \subseteq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle. \quad (5)$$

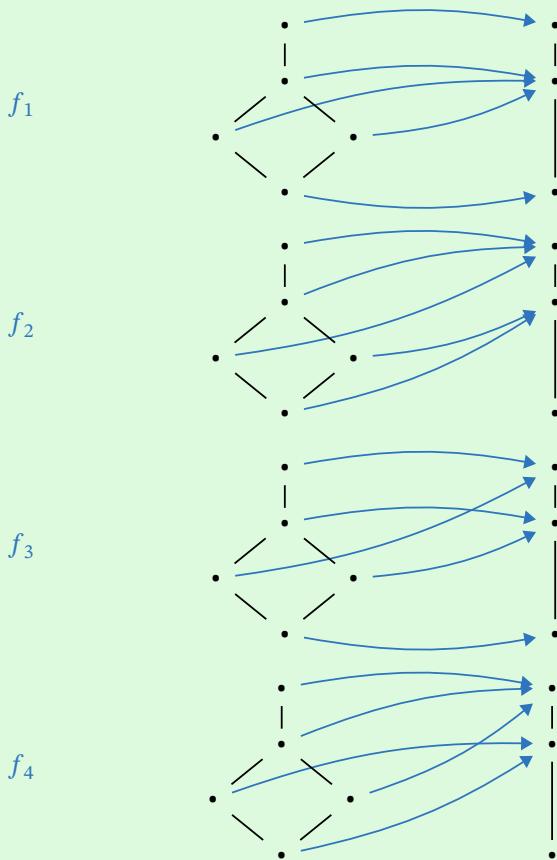
Graded exercise C.5 (`WhichMapsMonotone`)

[4 points]

Consider the posets $\mathbf{P} = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ and $\mathbf{Q} = \langle \mathbf{Q}, \leq_{\mathbf{Q}} \rangle$ described respectively by the following Hasse diagrams.



The following diagrams show functions $\mathbf{P} \rightarrow \mathbf{Q}$. We will call them f_1, f_2, f_3 and f_4 , respectively.



Which of the functions f_1, f_2, f_3 and f_4 are monotone maps?

Lemma 7.5. Consider a discrete poset \mathbf{P} and a poset \mathbf{Q} . Any map $f : \mathbf{P} \rightarrow \mathbf{Q}$ is monotone.

Graded exercise C.6 (FromDiscretePosets)

Prove Lemma 7.5

Graded exercise C.7 (`MonotoneMapCheck`)

Prove your answers to the following questions.

1. Is the function

$$\begin{aligned} f : \langle \mathbb{Z}, \leq \rangle &\rightarrow \langle \mathbb{Z}, \leq \rangle \\ x &\mapsto x^2 \end{aligned}$$

monotone?

2. Let $\mathbf{A} = \{a, b, c\}$ and consider the posets $\langle \text{Pow } \mathbf{A}, \subseteq \rangle$ and $\langle \mathbb{N}, \leq \rangle$. Let

$$\begin{aligned} f : \text{Pow } \mathbf{A} &\rightarrow \mathbb{N} \\ S &\mapsto \text{card}(S) \end{aligned}$$

be the function which calculates the cardinality of any subset of \mathbf{A} . Is f monotone?

3. Consider the set of natural numbers which divide the number 36, equipped with the partial order “ \leq ” such that $x \leq y$ if and only if x divides y . Call this poset $\mathbf{P} = \langle \mathbb{P}, \leq \rangle$, and let $f : \mathbf{P} \rightarrow \{\perp, \top\}$ be defined by

$$f(x) = \begin{cases} \top & \text{if } x \text{ is an even number,} \\ \perp & \text{if } x \text{ is an odd number.} \end{cases} \quad (7)$$

Is f monotone if we equip $\{\perp, \top\}$ with the usual partial order such that $\perp \leq \top$?

Definition 7.6 (Order isomorphism)

A monotone map is an *order isomorphism* if

$$\frac{p \leq_{\mathbf{P}} q}{f(p) \leq_{\mathbf{Q}} f(q)}. \quad (8)$$

Monotonicity is compositional

Monotonicity is a compositional property: the series composition of two monotone maps is monotone.

Lemma 7.7. Given posets $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ and two monotone maps $f : \mathbf{P} \rightarrow \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow \mathbf{R}$, the composite map $f ; g : \mathbf{P} \rightarrow \mathbf{R}$ is monotone as well.

Proof. Consider $p_1, p_2 \in \mathbf{P}$, $q_1, q_2 \in \mathbf{Q}$. By assuming that f and g are monotone, we have

$$\frac{p_1 \leq_{\mathbf{P}} p_2}{f(p_1) \leq_{\mathbf{Q}} f(p_2)} \quad (9)$$

and

$$\frac{q_1 \leq_{\mathbf{Q}} q_2}{g(q_1) \leq_{\mathbf{R}} g(q_2)} \quad (10)$$

By substituting the above in the map composition formula, we have

$$\frac{p_1 \leq_{\mathbf{P}} p_2}{(f ; g)(p_1) \leq_{\mathbf{R}} (f ; g)(p_2)}, \quad (11)$$

which is the monotonicity condition for the composite map $(f \circ g)$. \square

Antitone maps

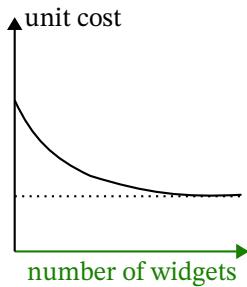
Dually to monotone functions, we can define antitone maps as order *reversing* functions.

Definition 7.8 (Antitone map)

An *antitone map* between two posets $P = \langle P, \leq_P \rangle$ and $Q = \langle Q, \leq_Q \rangle$ is a map f that reverses the ordering, in the sense that

$$\frac{x \leq_P y}{f(x) \geq_Q f(y)}. \quad (12)$$

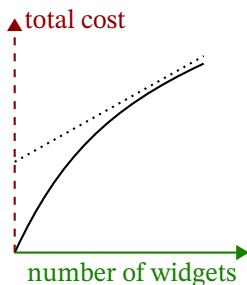
Example 7.9 (Unit cost, total cost). Assume that you want to produce some widgets, and that the manufacturing cost depends on the number of widgets. The function describing the total cost $t : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is a map between the ordered sets \mathbb{N} and $\mathbb{R}_{\geq 0}$, and maps each quantity of widgets to a total manufacturing cost (Fig. 2b). Clearly, t is a monotone function. Conversely, the unit cost function $u : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is antitone (Fig. 2a).



(a) Unit cost vs number of widgets.

It is easy to see that an antitone map $f : P \rightarrow Q$ is the same thing as a monotone map $f : P^{\text{op}} \rightarrow Q$.

Lemma 7.10. An antitone map $f : P \rightarrow Q$ is a monotone map $f : P^{\text{op}} \rightarrow Q$ and a monotone map $f : P \rightarrow Q^{\text{op}}$.



(b) Total cost vs number of widgets.

Figure 2.: Unit and total costs vs. number of widgets.

7.2. Monotone relations and design problems

Definition 7.11 (monotone relation)

Let $P = \langle P, \leq_P \rangle$ and $Q = \langle Q, \leq_Q \rangle$ be posets. A monotone relation $R : P \rightarrow Q$ is a relation $R : P \rightarrow Q$ such that for all $x, x' \in P$ and all $y, y' \in Q$,

1. $\langle x, y \rangle \in R, x' \leq_P x \implies \langle x', y \rangle \in R;$
2. $\langle x, y \rangle \in R, y \leq_Q y' \implies \langle x, y' \rangle \in R.$

Monotone relations and co-design

A monotone relation

$$\mathbf{d} : \mathbf{F} \rightarrow \mathbf{R} \quad (13)$$

can be used to model a relationship of “feasibility” between a poset \mathbf{F} of “functionalities” and a poset \mathbf{R} of “requirements”, in the sense that the relation describes whether a resource $f \in \mathbf{F}$, seen as a functionality or service or product, is *feasible* to obtain given a certain resource $r \in \mathbf{R}$, with r interpreted as a requirement or a cost.

The condition

$$\langle f, r \rangle \in \mathbf{d}, r \leq_{\mathbf{R}} r' \implies \langle f, r' \rangle \in \mathbf{d} \quad (14)$$

says that if f is feasible to obtain using r , then it is also feasible to obtain f if we use more resources, r' .

The condition

$$\langle f, r \rangle \in \mathbf{d}, f' \leq_{\mathbf{F}} f \implies \langle f', r \rangle \in \mathbf{d}, \quad (15)$$

on the other hand, says that if f is feasible to obtain using r amount of resources, then it is also feasible to obtain less, f' , using the same resources r .

Design problems

Here is an alternative way to think about relations of feasibility. Given a particular functionality $f \in \mathbf{F}$ and requirement $r \in \mathbf{R}$, we would like to know a “true” or “false” answer to the question of whether they form a feasible pair of resources. This situation is described by a function

$$g : \mathbf{F} \times \mathbf{R} \rightarrow \mathbf{Bool}. \quad (16)$$

The value $g(f, r)$ is the answer to the question “is the functionality f feasible with resources r ?”.

The conditions (14) and (15) can now be translated into this formulation: they say, respectively, that $g : \mathbf{F} \times \mathbf{R} \rightarrow \mathbf{Bool}$ is monotone in the variable r and antitone in the variable f . Or, equivalently, we can say that we have a monotone map of two variables of the type

$$\mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow \mathbf{Bool}. \quad (17)$$

Definition 7.12 (Design Problem)

Given posets \mathbf{F} and \mathbf{R} , a *design problem* (DP) from \mathbf{F} to \mathbf{R} is a monotone map of the form

$$\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\mathbf{Pos}} \mathbf{Bool}. \quad (18)$$

Definition 7.13 (Feasible set of a design problem)

We define the *feasible set* \mathbf{K}_d of a design problem

$$d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool} \quad (19)$$

as the subset of $\mathbf{F}^{\text{op}} \times \mathbf{R}$ for which d is the *indicator function*, that is

$$\mathbf{K}_d = \{ \langle f^*, r \rangle \in \mathbf{F}^{\text{op}} \times \mathbf{R} \mid d(f^*, r) = \top \}. \quad (20)$$

Note that the feasibility set \mathbf{K}_d of a design problem $d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ is a binary relation $\mathbf{K}_d \subseteq \mathbf{F}^{\text{op}} \times \mathbf{R}$. We saw in Section 4.4 that there is a one-to-one correspondence between functions $g : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{Bool}$ and binary relations $R : \mathbf{A} \rightarrow \mathbf{B}$.

Graded exercise C.8 (DPSAsUpperSets)

In this exercise, your task is to prove that there is a one-to-one correspondence between design problems $d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ and upper sets $\mathbf{K} \subseteq \mathbf{F}^{\text{op}} \times \mathbf{R}$.

In more detail:

Let \mathbf{A} denote the set of all design problems $d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ and let \mathbf{B} denote the set of all upper sets $\mathbf{K} \subseteq \mathbf{F}^{\text{op}} \times \mathbf{R}$.

1. Define a function

$$f : \mathbf{A} \rightarrow \mathbf{B}$$

which to any design problem in \mathbf{A} assigns a corresponding upper set in \mathbf{B} .

2. Define a function

$$g : \mathbf{B} \rightarrow \mathbf{A}$$

which maps any upper set in \mathbf{B} to a corresponding design problem in \mathbf{A} .

3. Prove that f and g are inverses to one another.

Graded exercise C.9 (DPSFromMonotoneMaps)

Given any monotone map $g : \mathbf{F} \rightarrow_{\text{Pos}} \mathbf{R}$, we can turn it into a design problem

$$d_g : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$$

via the following recipe. Set

$$d_g(f^*, r) = \top \quad \text{if and only if} \quad g(f^*) \leq r.$$

Prove that d_g , as defined above, is indeed a design problem when g is a monotone map.

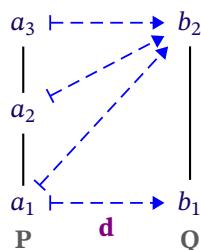
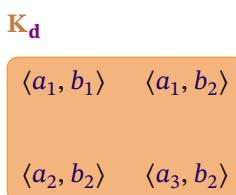


Figure 3.



Recall that when working with a relation $R : \mathbf{A} \rightarrow \mathbf{B}$ between sets, if the sets in question were finite, then we could conveniently draw the relation R using arrows to connect elements of \mathbf{A} to those elements of \mathbf{B} to which they are related via R . Given a design problem $d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ involving finite posets, we can visualize it in a similar fashion. We use Hasse diagrams to visualize the posets involved, and we use dashed arrows to connect those elements which are related via the feasibility set \mathbf{K}_d .

Example 7.14. In Fig. 3 we have illustrated this kind of visualization in the case of a design problem of the type

$$d : \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow_{\text{Pos}} \mathbf{Bool}, \quad (21)$$

Figure 4.

where $\mathbf{P} = \langle \mathbf{P}, \leq \rangle$ and $\mathbf{Q} = \langle \mathbf{Q}, \leq \rangle$ are finite posets, with

$$\mathbf{P} = \{a_1, a_2, a_3\} \quad \text{and} \quad \mathbf{Q} = \{b_1, b_2\}, \quad (22)$$

and ordered as shown in the figure.

The relation described by the design problem is marked with the dashed arrows; The feasibility set

$$\mathbf{K}_d = \{(a_1, b_1), (a_1, b_2), (a_2, b_2), (a_3, b_2)\}, \quad (23)$$

is reported in Fig. 4.

The Boolean-valued design problems we are considering in this section do not distinguish between particular implementations: they only tell us if *any* implementation or solution exists for given functionality and resources.

Diagrammatic notation We represent design problems using a diagrammatic notation. One design problem $d : \mathbf{F} \rightarrow \mathbf{R}$ is represented as a box with functionality \mathbf{F} on the *left* and resources \mathbf{R} on the *right* (Fig. 5).



Figure 5.: Diagrammatic representation of a design problem.

As we did for DPIs, we will connect these diagrams.

Example 7.15. An aerospace company, Jeb's Spaceship Parts, is designing a new rocket engine, the Bucket of Boom X100. The engine requires fuel and provides thrust, and so it can be modeled as a design problem where **fuel** and **thrust** are two totally-ordered sets representing their respective resources.

The corresponding diagram is reported in Fig. 6.

Concretely, “engine” is represented as a monotone map

$$\text{engine} : \text{thrust}^{\text{op}} \times \text{fuel} \rightarrow \text{pos Bool}. \quad (24)$$

Assuming that the posets **fuel**, **thrust**^{op} are finite, we can think of the “engine” design problem as a matrix, where each (i, j) -th entry is the answer to the question, “is the amount of thrust f_i feasible with the amount of fuel r_j ?”:

$$\begin{array}{c} \text{Fuel} \\ \begin{matrix} & r_1 = 0 & r_2 & r_3 & \dots & r_m \\ \text{thrust}^{\text{op}} & f_n^* = 0 & \left(\begin{array}{ccccc} 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & & 1 \\ 0 & 1 & 1 & & 1 \\ & & & \ddots & \\ 1 & 1 & 1 & & 1 \end{array} \right) \\ f_{n-1}^* \\ f_{n-2}^* \\ \vdots \\ f_1^* \end{matrix} \end{array} \quad (25)$$

Suppose we have tested or are given the performance data of a few different engines, as possible solutions to the “engine” design problem, each with a fixed optimal fuel-thrust value. To illustrate the monotonicity assumption, we can render the data of “engine” as a graph, as depicted in Fig. 7.

Note that the shaded regions cover the feasible solution set. This feasible solution set is always an *upper set* (Def. 8.10) in $\text{thrust}^{\text{op}} \times \text{fuel}$, which is another way of characterizing the monotonicity of the design problem. The optimal solutions, indicated by dots, form an *antichain* of solutions. We will come back to antichains when discussing how to compute optimal solutions of design problems.

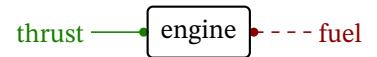


Figure 6.: Diagram of the engine design problem.

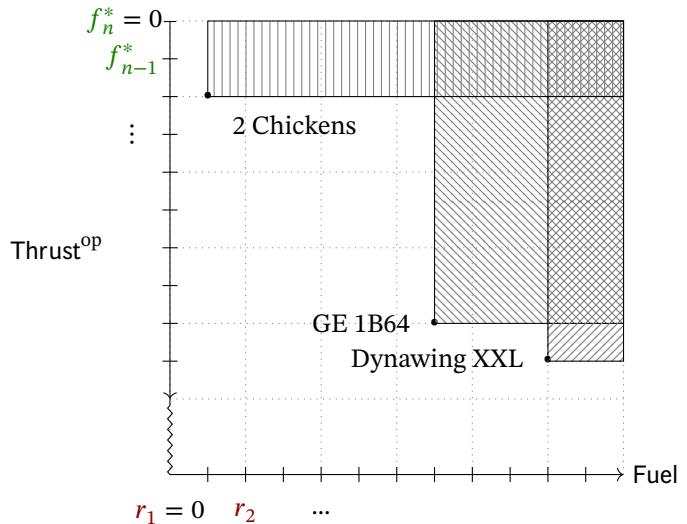


Figure 7.: Graphical representation of the possible solutions of the engine design problem.

Querying

Sometimes we are not focused on whether a specific pair of resources, $\langle \textcolor{violet}{f}, \textcolor{red}{r} \rangle$ is a feasible pair for a given design problem (or monotone relation), but rather, we would like to know, for a fixed functionality $\textcolor{violet}{f}$, what are *all* the requirements $\textcolor{red}{r}$ that render $\langle \textcolor{violet}{f}, \textcolor{red}{r} \rangle$ feasible. Or, analogously, we might want to know, for a fixed requirement $\textcolor{red}{r}$, what are *all* possible functionalities $\textcolor{violet}{f}$ such that $\langle \textcolor{violet}{f}, \textcolor{red}{r} \rangle$ is feasible. We call this type of a question a *query*.

Equation (1) on the one hand, and (2) on the other hand, give two perspectives on the mathematical definition of what we are calling a *design problem*. These two perspectives are analogous to something we already discussed in Section 4.4, when talking about binary relations. There, we said that a binary relation from a set $\textcolor{brown}{A}$ to a set $\textcolor{brown}{B}$ is a subset $\textcolor{blue}{R} \subseteq \textcolor{brown}{A} \times \textcolor{brown}{B}$, but that such a relation $\textcolor{blue}{R}$ can also, equivalently, be viewed as a function $\phi_{\textcolor{blue}{R}} : \textcolor{brown}{A} \times \textcolor{brown}{B} \rightarrow \{\perp, \top\}$. The subset $\textcolor{blue}{R} \subseteq \textcolor{brown}{A} \times \textcolor{brown}{B}$ corresponded to the set

$$\{(x, y) \in \textcolor{brown}{A} \times \textcolor{brown}{B} \mid \phi_{\textcolor{blue}{R}}(x, y) = \top\}. \quad (26)$$

To make the analogy with (20) more precise, note that $\textcolor{brown}{A}$, $\textcolor{brown}{B}$, $\{\perp, \top\}$, and $\phi_{\textcolor{blue}{R}} : \textcolor{brown}{A} \times \textcolor{brown}{B} \rightarrow \{\perp, \top\}$ live in the category of *sets*, and that $\textcolor{brown}{F}$, $\textcolor{brown}{R}$, \textbf{Bool} and $\textcolor{brown}{d} : \textcolor{brown}{F}^{\text{op}} \times \textcolor{brown}{R} \rightarrow \textbf{Pos Bool}$ live in the category of *posets*.

In Section 4.4, we also discussed two further ways to describe a relation $\textcolor{blue}{R} \subseteq \textcolor{brown}{A} \times \textcolor{brown}{B}$: namely, we can transform the function $\phi_{\textcolor{blue}{R}} : \textcolor{brown}{A} \times \textcolor{brown}{B} \rightarrow \{\perp, \top\}$ either into a function

$$\begin{aligned} \hat{\phi}_{\textcolor{blue}{R}} : \textcolor{brown}{A} &\rightarrow \text{Pow}(\textcolor{brown}{B}), \\ x &\mapsto \{y \in \textcolor{brown}{B} \mid x \textcolor{blue}{R} y\}. \end{aligned} \quad (27)$$

or a function

$$\begin{aligned} \check{\phi}_{\textcolor{blue}{R}} : \textcolor{brown}{B} &\rightarrow \text{Pow}(\textcolor{brown}{A}), \\ y &\mapsto \{x \in \textcolor{brown}{A} \mid x \textcolor{blue}{R} y\}. \end{aligned} \quad (28)$$

There are analogous transformations for a design problem $\textcolor{brown}{d} : \textcolor{brown}{F}^{\text{op}} \times \textcolor{brown}{R} \rightarrow \textbf{Pos Bool}$. Can you guess what they would be?

In order to use our “sets to posets” analogy and find an answer, it is useful to express the constructions we used in the setting of sets and relations entirely in terms of constructions from the category of sets, if possible. Then the strategy is to identify what are the analogous constructions in the category of posets, and this will allow us to make analogous definitions for design problems.

The functions $\hat{\phi}_R$ and $\check{\phi}_R$ above have powersets as their target objects. What is the analogue of the powerset operation in the category of posets?

The answer that we will use goes like this. Given a set A , there is a 1-to-1 correspondence between subsets of A and functions $A \rightarrow \{\perp, \top\}$ (similar to above, a set corresponds here to its indicator function). Thus, $\text{Pow}(A)$ can be seen to correspond to $\text{Hom}_{\text{Set}}(A; \{\perp, \top\})$. The latter is definitely an expression we can transfer, by analogy, to the category of posets, namely we can consider $\text{Hom}_{\text{Pos}}(P; \text{Bool})$. And from Graded Exercise F.8 we know that monotone maps $P \rightarrow \text{Bool}$ correspond to *upper* subsets of P . So $\text{Hom}_{\text{Pos}}(P; \text{Bool})$ corresponds to set $\text{USets}(P)$ of upper subsets of P (c.f. Section 8.3 for the definitions of upper and lower sets).

We now can write down the “poset” analogues of the functions $\hat{\phi}_R$ and $\check{\phi}_R$. Namely, given a design problem (19), we have associated functions

$$\hat{d} : \mathbf{F}^{\text{op}} \rightarrow \text{USets}(R) \quad (29)$$

and

$$\check{d} : R \rightarrow \text{USets}(\mathbf{F}^{\text{op}}). \quad (30)$$

However, we are not quite finished: are these monotone functions? Which poset structure can we choose on $\text{USets}(R)$ and $\text{USets}(\mathbf{F}^{\text{op}})$, respectively, so that \hat{d} and \check{d} are monotone?

Graded exercise C.10 (CurryingDesignProblems)

Let $d : \mathbf{F}^{\text{op}} \times R \rightarrow_{\text{Pos}} \text{Bool}$ be a design problem. In this exercise we will show that (29) corresponds to a monotone function

$$\mathbf{F} \rightarrow \langle \text{USets}(R), \sqsupseteq \rangle, \quad (31)$$

and that (30) corresponds to a monotone function

$$R \rightarrow \langle \text{LSets}(\mathbf{F}), \sqsubseteq \rangle. \quad (32)$$

Here $\text{USets}(R)$ denotes the set of upper sets of R and $\text{LSets}(\mathbf{F})$ denotes the set of lower sets of \mathbf{F} .

1. Show that $\hat{d} : \mathbf{F}^{\text{op}} \rightarrow \text{USets}(R)$ and $\check{d} : R \rightarrow \text{USets}(\mathbf{F}^{\text{op}})$ are monotone maps when we consider $\text{USets}(R)$ and $\text{USets}(\mathbf{F}^{\text{op}})$ to have the partial order corresponding to the inclusion of subsets.
2. Show that the poset $\langle \text{USets}(\mathbf{F}^{\text{op}}), \sqsubseteq \rangle$ and the poset $\langle \text{LSets}(\mathbf{F}), \sqsubseteq \rangle$ are isomorphic.
3. Show that there is a 1-to-1 correspondence between monotone functions

$$\mathbf{F}^{\text{op}} \rightarrow \langle \text{USets } R, \sqsupseteq \rangle \quad (33)$$

and monotone functions

$$\mathbf{F} \rightarrow \langle \text{USets } R, \sqsupseteq \rangle, \quad (34)$$

where in the latter poset, the order is given by the relation of “containment” (as opposed to “inclusion”).

4. Explain, in a few words, why the above steps prove the stated goal of this exercise.

A note on pre-orders

The theory of design problems can be easily generalized to pre-orders. This means that there could be two elements p and q such that $p \leq_P q$ and $p \geq_P q$

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

but $p \neq q$.

This is actually common in practice. For example, if the order relation comes from human judgement, such as customer preference, all bets are off regarding the consistency of the relation. We will only refer to posets for two reasons:

1. The exposition is smoother.
2. Given a pre-order, computation will always involve passing to the poset representation.

This means that, given a pre-order, we can consider the poset of its isomorphism classes, by means of the following equivalence relation:

$$p \simeq q \quad \equiv \quad (p \leq_p q) \wedge (q \leq_p p). \quad (35)$$

7.3. Order on monotone maps

Fixed two posets P and Q , the set of monotone maps $P \rightarrow Q$ form a poset themselves. We can order them point wise.

Definition 7.16 (Order on monotone maps)

Consider two monotone maps $f, g : P \rightarrow Q$. We say that f precedes g if, point wise, the output of f precedes the output of g when presented with the same input:

$$\frac{f \leq_{P \rightarrow Q} g}{\forall x \in P : f(x) \leq_Q g(x)}. \quad (36)$$

Example 7.17 (Rounding functions). In this example we look at “rounding functions”: these are functions that truncate a real number to an integer. You might already know the ceiling function `ceil` (Fig. 9a) and the floor function `floor` (Fig. 9b), which are formally defined as

$$\begin{aligned} \text{ceil} : & \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle, \\ x & \mapsto \min \{y \in \mathbb{N} : y \geq x\}, \end{aligned} \quad (37)$$

and

$$\begin{aligned} \text{floor} : & \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle, \\ x & \mapsto \max \{y \in \mathbb{N} : y \leq x\}. \end{aligned} \quad (38)$$

The functions `ceil` and `floor` are monotone, since $x \leq z$ implies both $\text{ceil}(x) \leq \text{ceil}(z)$ and $\text{floor}(x) \leq \text{floor}(z)$.

There exist many other rounding functions, commonly used by computers. For example, the map “round to nearest, ties to even” [11] rounds a number to the closest integer, and in case of ties it rounds to the even one (Fig. 9c). For example, 3.2 is mapped to 3, 1.5 is mapped to 2, and 4.5 is mapped to 4. This is the formal definition:

$$\begin{aligned} \text{rtntte} : & \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle, \\ x & \mapsto \begin{cases} \text{floor}(x), & x < (\text{floor}(x) + \text{ceil}(x))/2 \\ \text{ceil}(x), & x > (\text{floor}(x) + \text{ceil}(x))/2 \\ \text{ceil}(x), & (x = (\text{floor}(x) + \text{ceil}(x))/2) \wedge (\text{ceil}(x) \text{ is even}) \\ \text{floor}(x), & (x = (\text{floor}(x) + \text{ceil}(x))/2) \wedge (\text{floor}(x) \text{ is even}) \end{cases}. \end{aligned} \quad (39)$$

In this example, note that

$$\text{floor} \leq \text{id} \leq \text{ceil} \quad (40)$$

and

$$\text{floor} \leq \text{rtntte} \leq \text{ceil}, \quad (41)$$

and `id` and `rtntte` are not comparable (see Fig. 8).

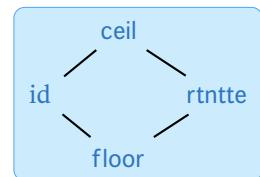


Figure 8.

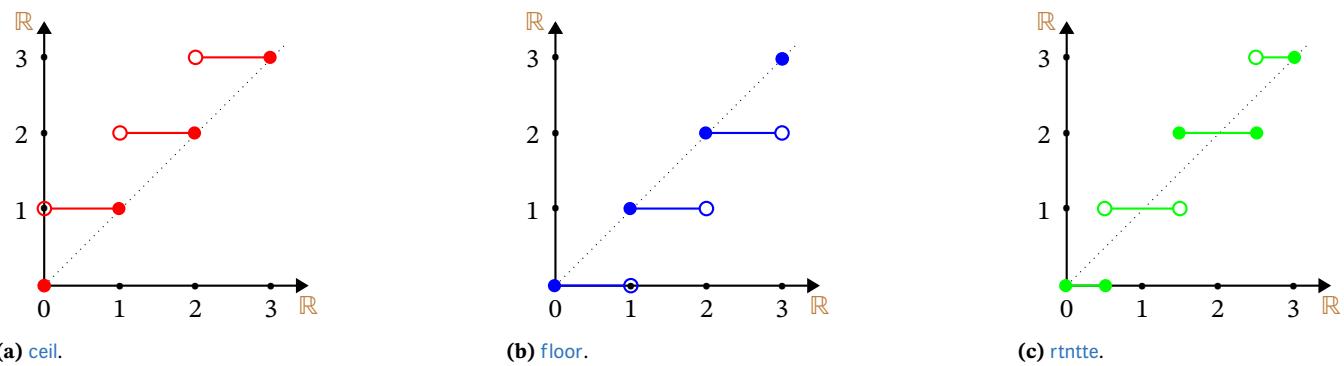


Figure 9.: Comparison of three rounding methods.



8. Poset bounds

This chapter describes some concepts related to posets: upper and lower sets, minimal/maximal elements, *etc.*

8.1 Minimal and maximal elements	124
8.2 Upper/lower bounds	125
8.3 Upper and lower sets	127
8.4 Antichains	128

A gondola lift is a type of cable transport which is supported and propelled by cables from above. This is typical in Swiss alps, where it is used in every season to bring people from one peak to the other.

8.1. Minimal and maximal elements

You know already the operators min/max that give the minimum/maximum values of a set of real numbers. If the set is finite, the minimum and maximum always exist. But for infinite sets, the minimum and maximum might not exist. For example, consider the set of real numbers contained between 0 and 1, excluding the boundaries:

$$\mathbf{A} = \{x \in \mathbb{R} : 0 < x < 1\}. \quad (1)$$

This set does not have a minimum or a maximum.

For a total order, if the minimum and maximum exist, then they are unique. In a partial order, this is not the case. We introduce the operators Min and Max that are the generalization to partial orders of min / max.

Definition 8.1 (Minimal elements)

$\text{Min} : \text{Pow } P \rightarrow \text{Anti } P$ is the map that sends a subset S of a poset to the minimal elements of that subset (those elements $a \in S$ such that $a \leq_P b$ for all $b \in S$). In formulas:

$$\begin{aligned} \text{Min} : \text{Pow } P &\rightarrow \text{Anti } P, \\ S &\mapsto \left\{ c \in S : \frac{d \in S \quad d \leq_P c}{c = d} \right\}. \end{aligned} \quad (2)$$

Note that $\text{Min}(S)$ could be empty.

Definition 8.2 (Maximal elements)

$\text{Max} : \text{Pow } P \rightarrow \text{Anti } P$ is the map that sends a subset S of a poset to the maximal elements of that subset (those elements $a \in S$ such that $a \geq_P b$ for all $b \in S$). In formulas:

$$\begin{aligned} \text{Max} : \text{Pow } P &\rightarrow \text{Anti } P, \\ S &\mapsto \left\{ c \in S : \frac{d \in S \quad d \geq_P c}{c = d} \right\}. \end{aligned} \quad (3)$$

Note that $\text{Max}(S)$ could be empty.

8.2. Upper/lower bounds

Definition 8.3 (Upper bounds in a poset)

The *upper bounds* of a subset S of a poset P are, if they exist, the elements of P which dominate all elements in S . In other words, the upper bounds of S are the elements of the set

$$\text{Upp}_B S := \{y \in P \mid \forall x \in S : x \leq_P y\}. \quad (4)$$

Definition 8.4 (Least upper bound / join / supremum)

A *least upper bound* of $S \subseteq P$, if it exists, is the least element among the upper bounds of S . It is denoted $\vee S$ or $\text{Sup } S$, and also called the *join* or *supremum* of S .

So, given $S \subseteq P$ and $y \in P$, $y = \vee S$ if and only if

1. $x \leq_P y$, $\forall x \in S$, and
2. $x \leq_P z$, $\forall x \in S \Rightarrow y \leq_P z$.

If a least upper bound of a subset $S \subseteq P$ exists, it is unique (can you prove this?), so we speak of “the” least upper bound.

Exercise 19. Let P be a poset and $S \subseteq P$ a subset of the underlying set of P . Show that if $\vee S$ exists, then it is unique. For this, assume that y and z are both least upper bounds of S , and then show that this assumption implies that in fact $y = z$.

See solution on page 131.

Example 8.5. Consider the poset P and its subset S depicted in Fig. 1. The red markers \bullet represent the upper bound of S . For this specific case, there is a *single* least upper bound.

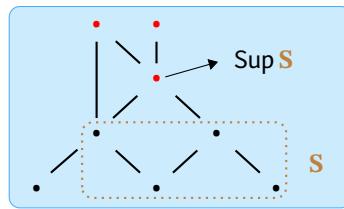


Figure 1.: Example of upper bounds and least upper bound for S .

Example 8.6. Least upper bounds need not necessarily exist even in total orders. For instance, the subset

$$\mathbb{R}_{>0} = \{x \in \mathbb{R} : x > 0\} \quad (5)$$

of the poset \mathbb{R} (with the usual ordering) does not have a least upper bound.

Analogously to the case of (least) upper bounds, we can define lower bounds and greatest lower bounds.

Definition 8.7 (Lower bounds in a poset)

The *lower bounds* of a subset S of a poset P are, if they exist, the elements which are dominated by all elements in S . In other words, the lower bounds of S are the elements of the set

$$\text{Low}_B S := \{y \in P \mid \forall x \in S : y \leq_P x\}. \quad (6)$$

Definition 8.8 (Greatest lower bound / meet / infimum)

The *greatest lower bound*, if it exists, is the greatest among the lower bounds of \mathbf{S} . This is denoted $\wedge \mathbf{S}$ or $\text{Inf } \mathbf{S}$ and also called the *meet* or *infimum* of \mathbf{S} .

Exercise 20. Come up with an example of a subset \mathbf{S} of a poset \mathbf{P} which has lower bounds but no greatest lower bound. Then, modify it to have a greatest lower bound.

See solution on page 131.

Definition 8.9 (Top and bottom)

If there is a least upper bound for the entire lattice \mathbf{P} , it is called the *top* (\top). If a greatest lower bound exists, it is called the *bottom* (\perp).

8.3. Upper and lower sets

Definition 8.10 (Upper set)

An *upper set* \mathbf{U} is a subset of a poset \mathbf{P} such that, if $x \in \mathbf{U}$, then all elements of \mathbf{P} that are above x are also in \mathbf{U} . In other words:

$$\frac{x \in \mathbf{U} \quad x \leq_{\mathbf{P}} y}{y \in \mathbf{U}}. \quad (7)$$

We call $\text{USets } \mathbf{P}$ the set of upper sets of \mathbf{P} .

Definition 8.11 (Lower set)

A *lower set* \mathbf{L} is a subset of a poset \mathbf{P} such that, if $x \in \mathbf{L}$, then all elements of \mathbf{P} that are below x are also in \mathbf{L} . In other words:

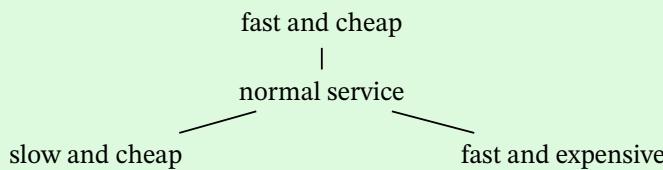
$$\frac{x \in \mathbf{L} \quad y \leq_{\mathbf{P}} x}{y \in \mathbf{L}}. \quad (8)$$

We call $\text{LSets } \mathbf{P}$ the set of lower sets of \mathbf{P} .

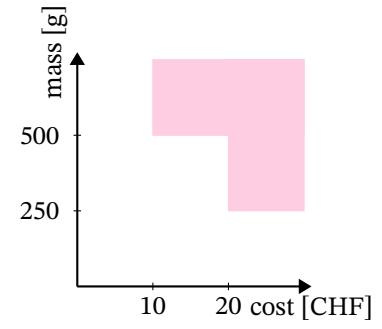
Given the battery choices $\{(10 \text{ CHF}, 500 \text{ g}), (20 \text{ CHF}, 250 \text{ g})\}$, we can represent an upper set as in Fig. 2a. The upper set can be interpreted as all the potential battery choices which are dominated by at least one of the two choices we have (in case we want to minimize mass and cost). Similarly, the lower set in Fig. 2b can be interpreted as all the potential battery choices which dominate at least one of the choices we have. Here when considering “the choices we have” in Fig. 2b, we not only consider the two choices directly presented to us, but also any convex combination of them.

Graded exercise C.11 (`UpperSetsOfPreferences`)

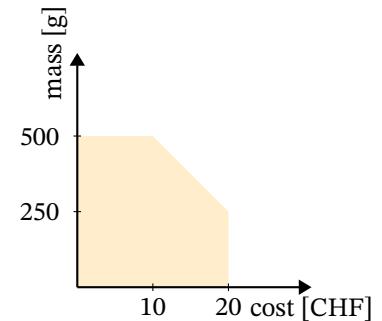
Consider the poset \mathbf{P} described by the following Hasse diagram:



Your task in this exercise is to compute all upper sets of \mathbf{P} .



(a)



(b)

Figure 2.

8.4. Upper and lower closure

Definition 8.12 (Upper closure operator)

The *upper closure operator* \uparrow maps a subset to the smallest upper set that includes it:

$$\begin{aligned} \uparrow : \text{Pow } \mathbf{P} &\rightarrow \text{USets } \mathbf{P}, \\ \mathbf{A} &\mapsto \{y \in \mathbf{P} \mid \exists x \in \mathbf{A} : x \leq_{\mathbf{P}} y\}. \end{aligned} \quad (9)$$

Remark 8.13. Note that, by definition, an upper set is closed to upper closure.

Lemma 8.14. For any $\mathbf{A} \in \text{Pow } \mathbf{P}$, $\uparrow \mathbf{A}$ is in fact an upper set.

Proof. Suppose $y \in \uparrow \mathbf{A}$ and $z \in \mathbf{P}$, and suppose $y \leq_{\mathbf{P}} z$. By definition there exists a x such that $x \leq_{\mathbf{P}} y$, meaning that $x \leq_{\mathbf{P}} z$. Thus, $z \in \uparrow \mathbf{A}$, as was to be shown. \square

Lemma 8.15. The upper closure operator \uparrow is an antitone map.

Proof. Consider the posets $(\text{Pow } \mathbf{P}, \subseteq)$ and $(\text{USets } \mathbf{P}, \supseteq)$, and two sets of sets $\mathbf{A}, \mathbf{B} \in \text{Pow } \mathbf{P}$. It is clear that given $\mathbf{A} \subseteq \mathbf{B}$, we have

$$\{y \in \mathbf{A} \mid \exists x \in \mathbf{A} : x \leq_{\mathbf{P}} y\} \subseteq \{y \in \mathbf{P} \mid \exists x \in \mathbf{B} : x \leq_{\mathbf{P}} y\}. \quad (10)$$

Therefore, $\uparrow \mathbf{A} \subseteq \uparrow \mathbf{B}$. Note that the poset $\text{USets } \mathbf{P}$ is ordered by the relation \supseteq , therefore $\uparrow \mathbf{A} \supseteq_{\text{USets } \mathbf{P}} \uparrow \mathbf{B}$, satisfying the antitone map property for \uparrow . \square

In the example of battery choices (in the numerical case), first, consider the upper closure of a single element of the poset, for instance $p_1 = \langle 10 \text{ CHF}, 500 \text{ g} \rangle$ (Fig. 3, left). Second, we can look at the upper closure when we add the choice $p_2 = \langle 20 \text{ CHF}, 250 \text{ g} \rangle$ (Fig. 3, center).

Note that the upper set of the subset formed by the two elements is the union of the upper sets of the single elements. Finally, we can also define the set

$$\mathbf{S} = \{\langle \text{cost}, \text{mass} \rangle \mid \text{mass} = 750 - 25 \cdot \text{cost}, \forall \text{ cost } \in [0, 20]\}, \quad (11)$$

and find its upper closure (Fig. 3, right).

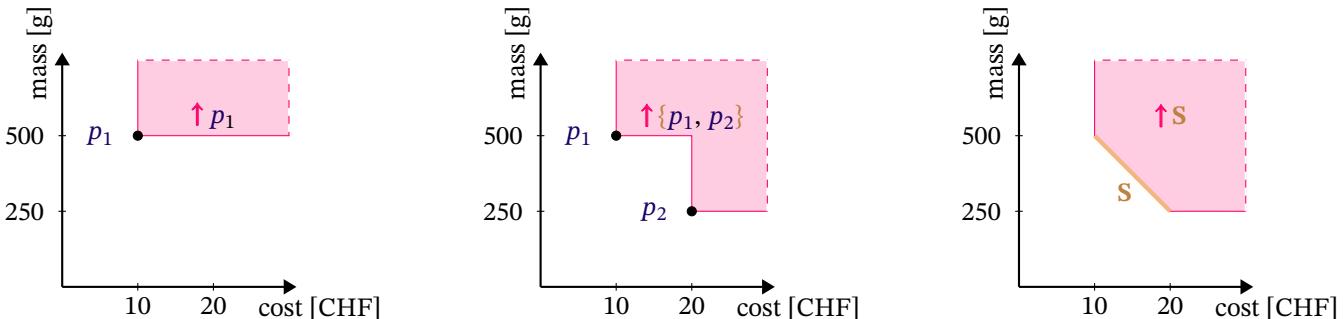


Figure 3. Example of upper closure for different sets of battery choices.

Definition 8.16 (Lower closure operator)

The *lower closure operator* \downarrow maps a subset to the smallest lower set that

includes it:

$$\begin{aligned} \downarrow : \text{Pow } \mathbf{P} &\rightarrow \text{LSets } \mathbf{P}, \\ \mathbf{S} &\mapsto \{y \in \mathbf{P} \mid \exists x \in \mathbf{S} : y \leq_{\mathbf{P}} x\}. \end{aligned} \quad (12)$$

Lemma 8.17. The lower closure operator \downarrow is a monotone map.

Exercise 21. Prove Lemma 8.17.

See solution on page 131.

Consider the battery example, and the antichain given by the battery models $p_1 = \langle 10 \text{ CHF}, 1000 \text{ g} \rangle$, $p_2 = \langle 20 \text{ CHF}, 500 \text{ g} \rangle$, and $p_3 = \langle 30 \text{ CHF}, 250 \text{ g} \rangle$ (Fig. 4, left). The lower closure operator $\downarrow\{p_1, p_2, p_3\}$ represents all the battery models which, if existing, would dominate $\{p_1, p_2, p_3\}$. We could instead consider linear maps between the points getting a poset \mathbf{P} , and obtain the lower closure depicted in Fig. 4 on the right.

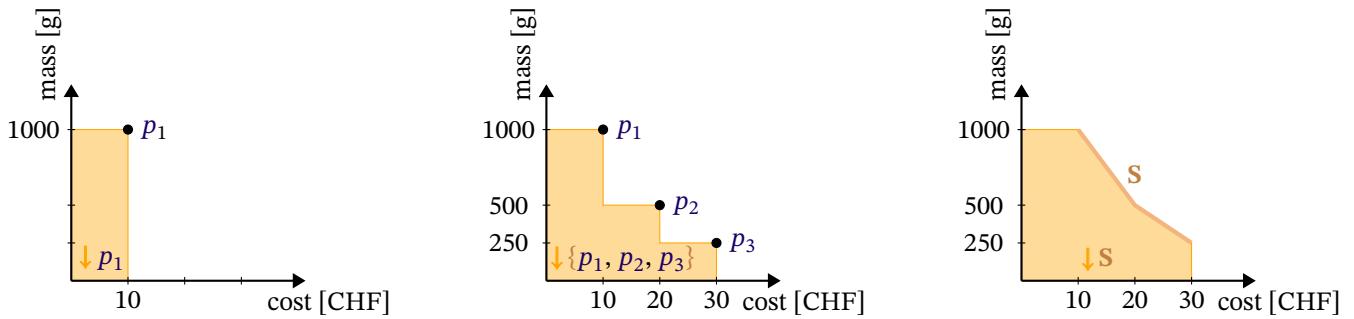


Figure 4.: Example of lower closures.

Antichains and upper sets

Lemma 8.18. Let \mathbf{A} and \mathbf{B} be subsets of \mathbf{P} that are antichains. Then

$$\frac{\uparrow \mathbf{A} = \uparrow \mathbf{B}}{\mathbf{A} = \mathbf{B}}. \quad (13)$$

Proof. Fix an element $a \in \mathbf{A}$. From $\uparrow \mathbf{A} = \uparrow \mathbf{B}$ we know that in particular $\mathbf{A} \subseteq \uparrow \mathbf{B}$. This means that for our fixed $a \in \mathbf{A}$ there exists $b \in \mathbf{B}$ such that $b \leq a$. From $\uparrow \mathbf{A} = \uparrow \mathbf{B}$ it also follows that $\mathbf{B} \subseteq \uparrow \mathbf{A}$, so to the $b \in \mathbf{B}$ given above, there exists $a' \in \mathbf{A}$ such that $a' \leq b$. In total, we have $a' \leq b \leq a$, and since \mathbf{A} is an antichain, we must have $a' = a$. This implies that $a' = b = a$. In particular, we have $a \in \mathbf{B}$.

The above shows that $\mathbf{A} \subseteq \mathbf{B}$. To show $\mathbf{B} \subseteq \mathbf{A}$, we can fix any $b \in \mathbf{B}$ and repeat the above argumentation, now with the roles of \mathbf{A} and \mathbf{B} exchanged. \square

Definition 8.19 (Downward closed set)

An upper set \mathbf{S} is *downward-closed* in a poset \mathbf{P} if

$$\mathbf{S} = \uparrow \text{Min } \mathbf{S}. \quad (14)$$

The set of downward-closed upper sets of \mathbf{P} is denoted $\overline{\text{UpSets}} \mathbf{P}$.

Definition 8.20 (Upward closed set)

A lower set \mathbf{S} is *upward-closed* in a poset \mathbf{P} if

$$\mathbf{S} = \downarrow \text{Max } \mathbf{S}. \quad (15)$$

The set of upward-closed lower sets of \mathbf{P} is denoted $\overline{\text{LowSets}} \mathbf{P}$.

Solutions to selected exercises

Solution of Exercise 14. P is a pre-order, because it satisfies reflexivity and transitivity. Q violates both reflexivity (e.g., $\langle x, x \rangle$ is missing), and transitivity (e.g., $\langle x, z \rangle$ is missing).

Solution of Exercise 15. The reachability relationship in any directed graph does not define a poset. As a simple counterexample, take a graph with nodes $\{x, y, z\}$ and paths x to y , y to z , and z to x . From transitivity, one has $x \leq z$, but from reachability we also have $z \leq x$. Therefore, per antisymmetry one should have $x = z$, but these are actually distinct nodes. To make things work, one needs to consider only acyclic graphs.

Solution of Exercise 16. Consider a set A . Clearly, given $S \in \text{Pow } A$, we have $S \subseteq S$. Furthermore, given also $T \in \text{Pow } A$, we have

$$\frac{S \subseteq T \quad T \subseteq S}{S = T}. \quad (16)$$

Finally, given also $U \in \text{Pow } A$, we have

$$\frac{S \subseteq T \quad T \subseteq U}{S \subseteq U}. \quad (17)$$

Solution of Exercise 17. We check the three conditions.

- ▷ First, we know that $[p_1, q_1] \leq_{\text{Tw } P} [p_1, q_1]$, since $p_1 \leq_P p_1$ and $q_1 \leq_P q_1$.
- ▷ Second, $[p_1, q_1] \leq_{\text{Tw } P} [p_2, q_2]$ and $[p_2, q_2] \leq_{\text{Tw } P} [p_3, q_3]$ imply

$$[p_1, q_1] \leq_{\text{Tw } P} [p_3, q_3]. \quad (18)$$

- ▷ Third, if $[p_1, q_1] \leq_{\text{Tw } P} [p_2, q_2]$ and $[p_2, q_2] \leq_{\text{Tw } P} [p_1, q_1]$, then $p_1 = p_2$ and $q_1 = q_2$.

Solution of Exercise 18. We check the three conditions.

- ▷ First, we know that $[p_1, q_1] \leq_{\text{Arr } P} [p_1, q_1]$, since $p_1 \leq_P p_1$ and $q_1 \leq_P q_1$.
- ▷ Second, $[p_1, q_1] \leq_{\text{Arr } P} [p_2, q_2]$ and $[p_2, q_2] \leq_{\text{Arr } P} [p_3, q_3]$ imply

$$[p_1, q_1] \leq_{\text{Arr } P} [p_3, q_3]. \quad (19)$$

- ▷ Third, if $[p_1, q_1] \leq_{\text{Arr } P} [p_2, q_2]$ and $[p_2, q_2] \leq_{\text{Arr } P} [p_1, q_1]$, then $p_1 = p_2$ and $q_1 = q_2$.

Solution of Exercise 19. Assume that y and z are both least upper bounds of $S \subseteq P$. In other words, one knows $x \leq_P y$ and $x \leq_P z$ for all $x \in S$. However, one also has $y \leq_P z$ and $z \leq_P y$ (from y, z assumed to be both least upper bounds). Because of antisymmetry, this implies $y = z$ and proves the uniqueness of least upper bounds in a poset.

Solution of Exercise 20. In Fig. 5 you find an example of a subset S of a poset P which has incomparable lower bounds. In Fig. 6 instead, there is a greatest lower bound.

Solution of Exercise 21. Consider the posets $(\text{Pow } P, \subseteq)$ and $(\text{LSets } P, \subseteq)$, and let $S_1, S_2 \in \text{Pow } P$. It is clear that given $S_1 \subseteq S_2$, we have

$$\{y \in P \mid \exists x \in S_1 : y \leq_P x\} \subseteq \{y \in P \mid \exists x \in S_2 : y \leq_P x\}. \quad (20)$$

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

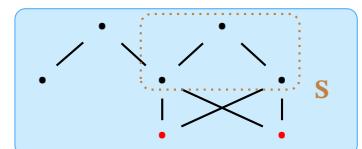


Figure 5.: Example of lower bounds of S .

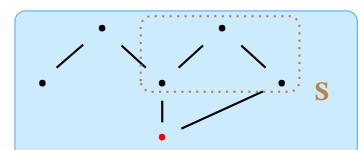


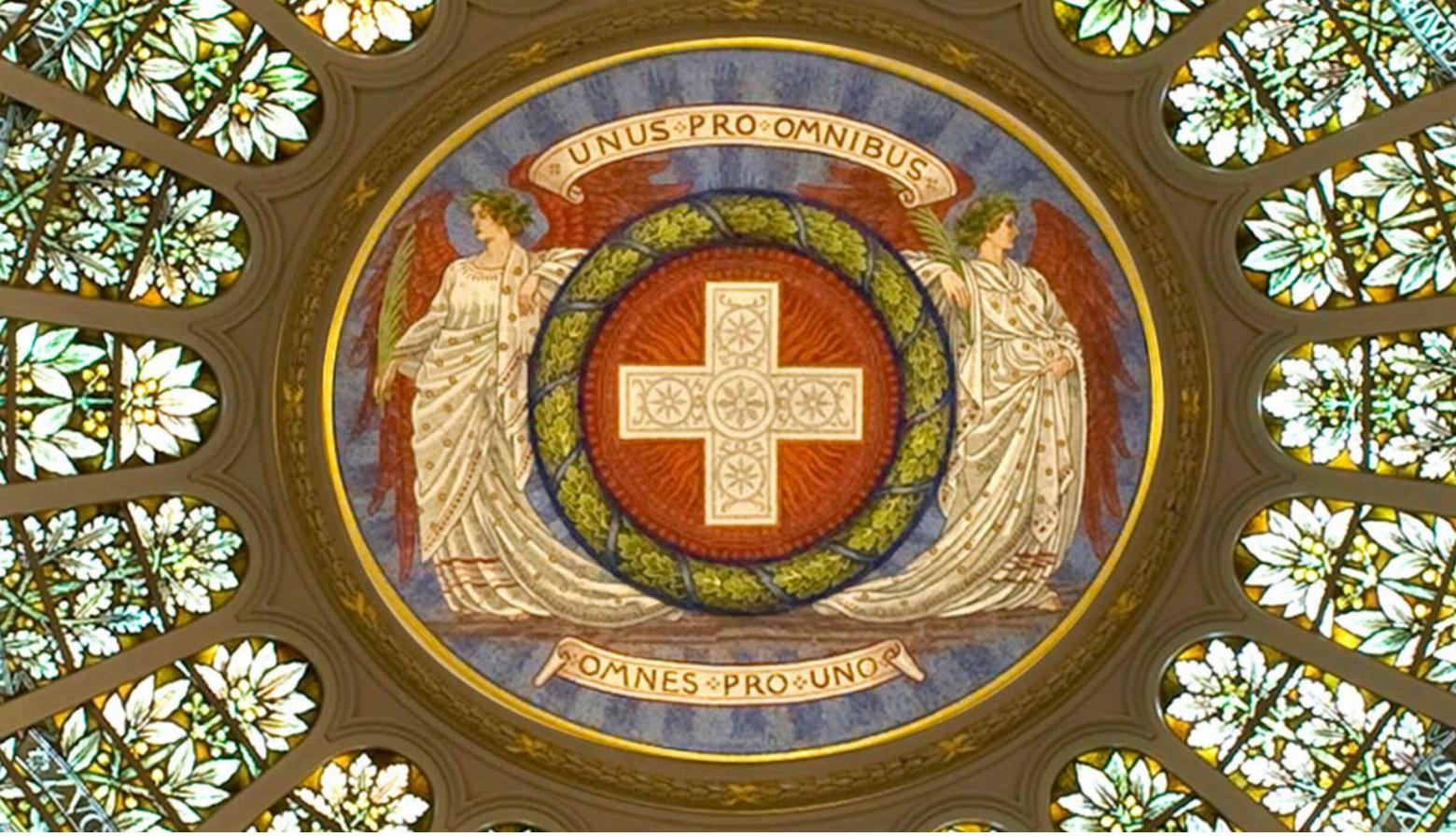
Figure 6.: Example of lower bounds and greatest lower bounds of S .

Therefore, $\downarrow S_1 \subseteq \downarrow S_2$, satisfying the monotonicity property for \downarrow .

PART D. ALGEBRA



9. Sets with operations	135
10. Morphisms	151
11. Actions	165



9. Sets with operations

This chapter introduces three of the most basic algebra structures: semigroups, monoids, and groups. These are sets with one *composition* operation. Additional properties of the operation distinguish among the structures.

9.1 Magmas	136
9.2 Semigroups	137
9.3 Monoids	141
9.4 Groups	144
9.5 Rings, fields	149

“Unus pro omnibus, omnes pro uno” is a Latin phrase that means “One for all, all for one”. Interestingly, this is the unofficial motto of Switzerland. A French version, “Un pour tous, tous pour un”, was made famous by Alexandre Dumas in the 1844 novel “The Three Musketeers”.

9.1. Magmas

Oftentimes we are going to study certain *structures* and then their *refinements*. By *refinement*, we mean another type of structure that has additional properties or constraints.

The simplest algebraic structure that has to do with composition is that of a *magma*: it just assumes that there is a set with a binary operation defined on it.

Definition 9.1 (Magma)

A *magma* \mathbf{S} is a set \mathbf{S} , together with a binary operation

$$\circ : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}, \quad (1)$$

which we refer to as “composition”.

Table 9.1.: Composition table.

Given a finite set \mathbf{A} , one way to specify a composition operation \circ on \mathbf{A} is simply by writing out what it does with each pair of elements of \mathbf{A} . Since \circ is a function of two variables, this can be conveniently displayed as a table, sometimes called a *multiplication table* or a *Cayley table*. We will use the name *composition table*.

Example 9.2. Consider the set

$$\mathbf{A} = \{ \text{Red blob}, \text{Blue blob} \}, \quad (2)$$

representing painting colors. A composition operation \circ is specified in Table 9.1. The rule describes the process of “painting over” another color, meaning that the last color that has been used to paint is the dominant one. We read it as saying

$$\begin{array}{ll} \text{Red blob} \circ \text{Red blob} = \text{Red blob} & \text{Red blob} \circ \text{Blue blob} = \text{Red blob} \\ \text{Blue blob} \circ \text{Red blob} = \text{Red blob} & \text{Blue blob} \circ \text{Blue blob} = \text{Blue blob} \end{array}$$

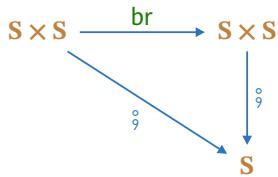


Figure 1.: The commutativity property

Definition 9.3

A magma $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ is called *commutative* (or: Abelian) if

$$x \circ y = y \circ x, \quad (3)$$

for all $x, y \in \mathbf{S}$.

Exercise 22. Is the magma specified in Table 9.1 commutative?

See solution on page 175.

Magmas are quite general and simplistic. There is not all much to say about them.

We can build more interesting structures by considering, for example, properties that the composition operation might have.

In the next sections we will study:

- ▷ *semigroups*, magmas in which the operation is associative;
- ▷ *monoids*, semigroups with an identity;
- ▷ *groups*, monoids with an inverse operation.

9.2. Semigroups

A *semigroup* is a magma for which composition is associative.

Definition 9.4 (Semigroup)

A *semigroup* \mathbf{S} is defined by:

Constituents

1. A set \mathbf{S} ;
2. A binary operation $\circ : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}$ called *composition*.

Conditions

1. Associative law

$$(x \circ y) \circ z = x \circ (y \circ z), \quad (4)$$

for all $x, y, z \in \mathbf{S}$.

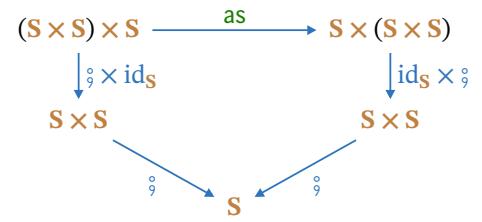


Figure 2.: Semigroup Commutative Diagram

Remark 9.5. Given a fixed set \mathbf{S} , there will in general be many choices of composition operation which make \mathbf{S} into a semigroup. Therefore, technically, a semigroup \mathbf{S} is a pair $\langle \mathbf{S}, \circ \rangle$ consisting of a set \mathbf{S} and a choice of composition \circ . The set \mathbf{S} is the *underlying set* of the semigroup.

Often we will be slightly imprecise and refer to a semigroup simply by the name of its underlying set; this is practical when it is clear from context which composition operation we are considering, or when it is not necessary to refer to the composition operation explicitly.

Also note that any semigroup is in particular a magma $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$, and so it also has an “underlying magma”.

Definition 9.6 (Commutative semigroup)

A semigroup $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ is called *commutative* (or: Abelian) if its underlying magma is commutative.

Some examples

Example 9.7. Consider the semigroup $\langle \mathbb{N}, + \rangle$, which defines composition as

$$x \circ y := x + y. \quad (5)$$

This is a semigroup, since, for all $l, m, n \in \mathbb{N}$, we have

$$(l + m) + n = l + (m + n). \quad (6)$$

It is also a commutative semigroup because addition is commutative.

Example 9.8. Pair-wise average on \mathbb{R} ,

$$x \circ y := \frac{x + y}{2}, \quad (7)$$

does not define semigroup composition, because it is not associative.

For example:

$$(4 \circ 8) \circ 16 = 11 \neq 8 = 4 \circ (8 \circ 16). \quad (8)$$

Example 9.9 (Booleans). Consider the set $\mathbf{Bool} = \{\perp, \top\}$, and $\langle \mathbf{Bool}, \wedge \rangle$, where the operation \wedge (“and”) is defined via Table 9.2.

This forms a semigroup, given the associativity of \wedge .

Table 9.2.: Composition table for booleans.

\wedge	\perp	\top
\perp	\perp	\perp
\top	\perp	\top

Graded exercise D.1 (CompositionTable)

Consider the composition presented in Table 9.1. Does this composition operation define a semigroup?

Exercise 23. [Cross-product] Consider $S = \mathbb{R}^3$ and the operation usually referred to as the “cross-product”:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} \circ \begin{bmatrix} x \\ y \\ z \end{bmatrix} := \begin{bmatrix} bz - cy \\ cx - az \\ ay - bx \end{bmatrix}. \quad (9)$$

This is a binary operation and therefore $\langle \mathbb{R}^3, \circ \rangle$ forms a magma. Show that this does not form a semigroup.

See solution on page 175.

Example 9.10. Consider a finite set A , which we think of as an alphabet. For instance, consider

$$A = \{\bullet, \circ\}. \quad (10)$$

Let $S = \text{List } A$ be the set of non-empty lists of elements of A . For example,

$$[\bullet, \bullet, \bullet, \bullet, \circ, \bullet, \bullet, \bullet, \bullet] \quad (11)$$

is a non-empty list of elements of A .

We may define a composition operation on S simply by concatenating lists. Given the lists

$$[\bullet, \bullet, \bullet, \bullet, \circ, \bullet, \bullet, \bullet, \bullet] \text{ and } [\bullet, \bullet, \bullet, \bullet], \quad (12)$$

their concatenation

$$[\bullet, \bullet, \bullet, \bullet, \circ, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet] \circ [\bullet, \bullet, \bullet, \bullet] \quad (13)$$

is the list

$$[\bullet, \bullet, \bullet, \bullet, \circ, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet]. \quad (14)$$

It is readily seen that concatenation satisfies the associative law, so S , together with this multiplication, forms a semigroup. It is often called *free semigroup* on the set A , a terminology which we will explain later.

Graded exercise D.2 (VariationsOnConcatenation)

Consider the set S of finite non-empty lists of symbols from the alphabet A , as in Example 9.10.

Can you think of other candidates for multiplication operations on S , besides the straightforward concatenation of lists considered above? Do your candidates define semigroup multiplications—that is, do they obey the associative law?

For example, one might consider the operation where, given an ordered pair of lists, one first doubles the last symbol of the first list and then concatenates. Is this operation associative? Justify your answers.

Example 9.11. The function $\max : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ defines a multiplication operation which equips \mathbb{N} with the structure of a semigroup. It is easy to show that it satisfies associativity. Given $x, y, z \in \mathbb{N}$, we have:

$$\max(\max(x, y), z) = \max(x, \max(y, z)). \quad (15)$$

Exercise 24. Verify the statement made in Example 9.11; that is, check that the associative law holds.

Does $\min : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ also define a semigroup structure on \mathbb{N} ?

See solution on page 175.

Example 9.12. Consider the set $\mathbf{A} = \{\text{sprout, young, mature, old, dead}\}$ which describes five possible states of a plant. Let $f : \mathbf{A} \rightarrow \mathbf{A}$ be the function that describes “development” (Fig. 3):

$$f(\text{sprout}) = \text{young}, \quad (16)$$

$$f(\text{young}) = \text{mature}, \quad (17)$$

$$f(\text{mature}) = \text{old}, \quad (18)$$

$$f(\text{old}) = \text{dead}, \quad (19)$$

$$f(\text{dead}) = \text{dead}. \quad (20)$$

In other words, we think of f as the change of state of the plant during a given time interval (say, three months). Composing the function f with itself corresponds to considering multiples of the given time interval. For example, the function

$$(f ; f ; f) : \mathbf{A} \rightarrow \mathbf{A} \quad (21)$$

models the change over the course of nine months. In general, for the n -fold composition of f with itself we write f^n . The set $\mathbf{T} = \{f^n \mid n \in \mathbb{N}\}$, together with the multiplication given by the composition operation, forms a semigroup.

Opposite semigroup

Any semigroup has an opposite: it is obtained by “flipping” the order of composition.

Definition 9.13 (Opposite semigroup)

$\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ be a semigroup. Its *opposite* \mathbf{S}^{op} is the semigroup $\mathbf{S}^{\text{op}} = \langle \mathbf{S}, \circ^{\text{op}} \rangle$ whose composition operation \circ^{op} is the composite

$$\mathbf{S} \times \mathbf{S} \xrightarrow{\text{br}} \mathbf{S} \times \mathbf{S} \xrightarrow{\circ} \mathbf{S} \quad (22)$$

where br is the braiding function

$$\begin{aligned} \text{br} : \mathbf{S} \times \mathbf{S} &\rightarrow \mathbf{S} \times \mathbf{S}, \\ \langle x, y \rangle &\mapsto \langle y, x \rangle. \end{aligned} \quad (23)$$

In other words,

$$x \circ^{\text{op}} y = y \circ x \quad \forall x, y \in \mathbf{S}. \quad (24)$$

Subsemigroups

Definition 9.14 (Subsemigroup)

Let $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ be a semigroup. A *subsemigroup* of \mathbf{S} is:

Constituents

1. A subset $\mathbf{T} \subseteq \mathbf{S}$.

Conditions

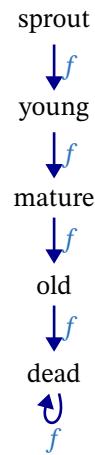


Figure 3.: Graphical representation of plant transitions.

1. The set \mathbf{T} is closed under the composition operation \circ from \mathbf{S} :

$$\frac{x \in \mathbf{T} \quad y \in \mathbf{T}}{(x \circ y) \in \mathbf{T}}. \quad (25)$$

Example 9.15. Consider the semigroup $\langle \mathbb{N}, + \rangle$ introduced in Example 9.7. We can take the subset of even natural numbers $\mathbf{T} \subset \mathbb{N}$. One can show that the sum of two even numbers is always even, satisfying the closure of \mathbf{T} under the composition operation $+$.

Induced n -ary multiplication

Given a semigroup $\langle \mathbf{A}, \circ \rangle$, for each integer $n \geq 1$, we can define an induced n -ary multiplication operation

$$\begin{aligned} \circ^n : \mathbf{A}^n &\rightarrow \mathbf{A}, \\ \langle x_1, x_2, \dots, x_n \rangle &\mapsto x_1 \circ x_2 \dots \circ x_n. \end{aligned} \quad (26)$$

Thanks to the associative law, this is well-defined — that is, we do not need to use parentheses. We will say that an element $x \in \mathbf{A}$ is an n -fold multiplication if it is in the image of this n -ary multiplication operation. At times, we may not wish to specify the arity of the multiplication, in which case we just speak of a multiplication.

9.3. Monoids

Algebraic structures are often defined in *layers*.

For example, in the definition of semigroup, we start with a set S as a basic building block, and we add a layer of structure to it, namely a multiplication operation $\circ : S \times S \rightarrow S$. The multiplication operation for semigroups was not only a new *structure* which we added, but we also required this structure to obey a *condition*, namely that it satisfies the associative law. One might also say that the multiplication operation was a new *constituent* or a new *datum*, and that satisfying the associative law is a *property*.

Mathematicians often use such words in an intuitive, non-rigorous way as a tool for structuring their thinking. We will do the same. For clarity, we will aim to stick with the words *constituents* and *conditions*. Roughly speaking, we think of constituents as building blocks, and we think of conditions as rules for how those blocks fit together and behave.

Using the constituent vs condition distinction we will, in particular, present some definitions in the following succinct, list-like fashion:

Definition 9.16 (Monoid)

A monoid M is given by:

Constituents

1. A set M ;
2. A binary operation $\circ : M \times M \rightarrow M$;
3. A specified element $\text{id}_M \in M$, called *neutral element*.

Conditions

1. Associative law:

$$(x \circ y) \circ z = x \circ (y \circ z) \quad \forall x, y, z \in M; \quad (27)$$

2. Neutrality Laws: $\text{id}_M \circ x = x = x \circ \text{id}_M \quad \forall x \in M$.



Figure 4.: The neutral element, expressed as a function.

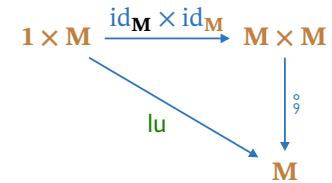


Figure 5.: Monoid Commutative Diagram

Remark 9.17. The way that we presented the definition of a monoid is certainly not unique. For example, we could have done the following.

A monoid M is:

Constituents

1. a semigroup $\langle M, \circ \rangle$;
2. a specified element $\text{id}_M \in M$, called *neutral element*.

Conditions

1. Neutrality laws: $\text{id}_M \circ x = x = x \circ \text{id}_M$.

In this version, two constituents and one condition from Def. 9.16 are “compressed” into the information that we are using here a semigroup as a constituent. This kind of “compression” has its pros and cons; depending on the context will use it to varying degrees.

There is a similar dilemma when considering the software interfaces to describe these structures. In terms of software engineering, the two strategies are *composition* (a monoid has a semigroup as a constituent) and *inheritance* (a monoid is a semigroup with additional data).

Remark 9.18. A monoid is called *commutative* (or: Abelian) if its underlying semigroup is commutative.

Some examples

Example 9.19. Consider $\langle \mathbb{R}, +, 0 \rangle$. This is a monoid, since, for all $x, y, z \in \mathbb{R}$, we have

$$(x + y) + z = x + (y + z), \quad (28)$$

and

$$x + 0 = x = 0 + x. \quad (29)$$

Similarly, $\langle \mathbb{N}, +, 0 \rangle$, $\langle \mathbb{Z}, +, 0 \rangle$, and $\langle \mathbb{Q}, +, 0 \rangle$ are monoids.

Example 9.20. The set \mathbb{Z} , together with the operation of multiplication of whole numbers, forms a monoid. The neutral element is the number 1.

Example 9.21. Given a set \mathbf{A} , the set $\mathbf{End}(\mathbf{A})$ of functions from \mathbf{A} to \mathbf{A} comes “naturally equipped” with a monoid structure: take monoid composition to be function composition, and let the identity element be given by the identity function on \mathbf{A} .

Example 9.22. Consider $\langle \mathbf{Bool}, \wedge \rangle$ as in Example 9.9, and consider \top as neutral element. This forms a monoid, since $b \wedge \top = b = \top \wedge b$, for all $b \in \mathbf{B}$.

Lemma 9.23. Let $\langle \mathbf{S}, \circ \rangle$ be a semigroup. If there exist elements $1 \in \mathbf{S}$ and $1' \in \mathbf{S}$ such that $\langle \mathbf{S}, \circ, 1 \rangle$ and $\langle \mathbf{S}, \circ, 1' \rangle$ are each monoids, then $1 = 1'$ must hold. In other words, the neutral element of a monoid is uniquely determined by the underlying semigroup structure.

Graded exercise D.3 (UniqueNeutralMonoid)

Prove Lemma 9.23.

Example 9.24. Consider $\langle \mathbb{R}_{\geq 0}, \max, 0 \rangle$. This is a monoid, since, for all $x, y \in \mathbb{R}_{\geq 0}$, we have:

$$\max(\max(x, y), z) = \max(x, \max(y, z)), \quad (30)$$

and

$$\max(x, 0) = x = \max(0, x). \quad (31)$$

Remark 9.25. Note that in the above example, we could have just as well instead considered the set $\mathbb{R}_{>7.5}$ of real numbers greater than 7.5, together with “max” as composition and 7.5 as neutral element. In other words, we can choose any real number $a \in \mathbb{R}$ and obtain a monoid $\langle \mathbb{R}_{\geq a}, \max, a \rangle$.

Example 9.26. $\langle \mathbb{N}, \max, 0 \rangle$ forms a monoid.

Definition 9.27

Let \mathbf{A} be a set. We denote by $\mathbf{List} \mathbf{A}$ the set of all lists of elements of \mathbf{A} .

Example 9.28. For any set \mathbf{A} , the set $\mathbf{List} \mathbf{A}$ of lists of elements of \mathbf{A} can naturally be equipped with a monoid structure: composition is concatenation (just like in Example 9.10), and the neutral element is the empty list $[]_{\mathbf{A}}$. This monoid is known as the *free monoid* on the set \mathbf{A} .

Remark 9.29. Just like for semigroups, any monoid has an opposite. It is defined similarly: given a monoid $\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle$, its opposite is $\mathbf{M}^{\text{op}} = \langle \mathbf{M}, \circ^{\text{op}}, \text{id} \rangle$ where \circ^{op} is defined by setting

$$x \circ^{\text{op}} y = y \circ_{\mathbf{M}} x \quad \forall x, y \in \mathbf{M}. \quad (32)$$

Monoids and dynamical systems

Suppose that we are dealing with dynamical systems such that for any point x_0 , there is exactly one trajectory beginning at x_0 . (For this we need to put suitable constraints on the function f .)

We can then ask the following: given a point x , where would its trajectory be after δ ? This question induces a family of functions T_δ , called transition functions. For each particular δ , we have a function

$$T_\delta : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (33)$$

that maps a point to its position δ in the future.

We can spot here a semigroup structure. Suppose we want to know the position of a point $\delta_1 + \delta_2$ in the future. We can take T_{δ_1} and compose it with T_{δ_2} ; or take directly $T_{\delta_1 + \delta_2}$. By construction, we will have that

$$T_{\delta_1 + \delta_2} = T_{\delta_1} \circ T_{\delta_2}. \quad (34)$$

We can also easily prove associativity:

$$\begin{aligned} T_{\delta_1} \circ (T_{\delta_2} \circ T_{\delta_3}) &= T_{\delta_1} \circ T_{\delta_2 + \delta_3} \\ &= T_{\delta_1 + \delta_2 + \delta_3} \\ &= T_{\delta_1 + \delta_2} \circ T_{\delta_3} \\ &= (T_{\delta_1} \circ T_{\delta_2}) \circ T_{\delta_3}. \end{aligned} \quad (35)$$

This shows that the set of transition functions for a particular system with the operation of function composition form a semigroup.

This semigroup is a monoid because there is an identity. The identity is T_0 , the map that tells us what happens after 0 seconds. That is $\text{id}_{\mathbb{R}^n}$, the identity on \mathbb{R}^n . To show that $T_0 = \text{id}_{\mathbb{R}^n}$ is an identity, we can fix any δ and substituting in (34) we have

$$\begin{aligned} T_{\delta+0} &= T_\delta \circ T_0 \\ &= T_\delta. \end{aligned} \quad (36)$$

Submonoids

Definition 9.30 (Submonoids)

Let $\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle$ be a monoid. A *submonoid* of \mathbf{M} is:

Constituents

1. A subset $\mathbf{N} \subseteq \mathbf{M}$.

Conditions

1. The set \mathbf{N} is closed under the composition operation \circ from \mathbf{M} :

$$\frac{x \in \mathbf{N} \quad y \in \mathbf{N}}{x \circ y \in \mathbf{N}}; \quad (37)$$

2. The neutral element $\text{id} \in \mathbf{M}$ is an element of \mathbf{N} .

Example 9.31. $\langle \mathbb{N}, +, 0 \rangle$, $\langle \mathbb{Z}, +, 0 \rangle$, and $\langle \mathbb{Q}, +, 0 \rangle$ are all submonoids of $\langle \mathbb{R}, +, 0 \rangle$.

Example 9.32. $\langle \mathbb{N}, \cdot, 1 \rangle$ is a submonoid of $\langle \mathbb{Z}, \cdot, 1 \rangle$.

Example 9.33. $\{x \in \mathbb{N} \mid x \text{ is even}\}$ is a submonoid of $\langle \mathbb{N}, +, 0 \rangle$, provided that 0 is considered an even natural number.

9.4. Groups

Groups appear in many areas of mathematics, both pure and applied.

One important use of groups is to describe symmetries. Roughly speaking, a symmetry is an invertible transformation or reconfiguration of a figure (or object) that leaves the essential features of that figure invariant.

For example, consider a perfectly square sheet of monochrome paper lying on a table-top. If we rotate the piece of paper by 90 degrees around its center, how it appears to us after this rotation will be essentially indistinguishable from before it was rotated. Thus, this rotation is a symmetry.

Groups of symmetries play an important role in physics and chemistry, for example when studying the repeating patterns of lattices of atoms or molecules in materials, or when studying the geometric patterns of atoms and molecules themselves.

For engineering, a particularly important class of groups are matrix groups, in particular those that describe various types of motion in space. These are discussed in Section 9.4.

Definition 9.34 (Group)

A group is a monoid together with an “inverse” operation. In more detail, a group G is

Constituents

1. a set G ;
2. a binary operation $\circ : G \times G \rightarrow G$, called *composition*;
3. a specified element $\text{id}_G \in G$, called *neutral element*;
4. a map $\text{inv} : G \rightarrow G$, called *inverse*.

Conditions

1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z), \quad \forall x, y, z \in G$;
2. Neutrality laws: $\text{id}_G \circ x = x = x \circ \text{id}_G, \quad \forall x \in G$;
3. Inverse laws:

$$\text{inv}(x) \circ x = \text{id}_G = x \circ \text{inv}(x), \quad \forall x \in G. \quad (38)$$

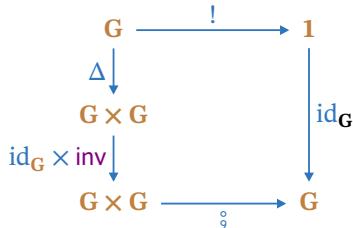


Figure 6.: Group Diagram

Definition 9.35 (Commutative group)

A group is called *commutative* (or: *Abelian*) if its underlying semigroup is commutative.

Remark 9.36. The size of the underlying set of a group is often called the *order* or cardinality of the group. We'll sometimes use this terminology, or just call it the size of the group.

Some examples

Example 9.37. The following is a group: the set \mathbb{Z} , together with addition as the composition operation, the element 0 as neutral element, and “taking the negative” as the inverse operation:

$$\text{inv}(x) := -x, \quad \forall x \in \mathbb{Z}. \quad (39)$$

Example 9.38. The monoid $\langle \mathbb{R}_{\neq 0}, \cdot, 1 \rangle$ becomes a group when equipped with

the inverse operation defined by

$$\text{inv}(x) := \frac{1}{x}, \quad \forall x \in \mathbb{R}. \quad (40)$$

Example 9.39. For the monoids $\langle \mathbb{N}, +, 0 \rangle$ and $\langle \mathbb{N}, \cdot, 1 \rangle$ we cannot find an inverse operation that would turn these monoids into groups.

Exercise 25. Can one find an inverse operation for the monoid $\langle \mathbb{N}, \max, 0 \rangle$?

See solution on page 175.

Example 9.40. The monoid $\langle \text{Bool}, \wedge, \top \rangle$ from Example 9.22 cannot become a group, because there cannot be an inverse for \perp : there is no possible choice for $\text{inv}(\perp)$ such that $\text{inv}(\perp) \wedge \perp = \top$.

Example 9.41. Given a set \mathbf{A} , an invertible function $\mathbf{A} \rightarrow \mathbf{A}$ is called an *automorphism* of \mathbf{A} . There is a “naturally given” group structure on the set $\text{Aut}(\mathbf{A})$ of automorphisms of \mathbf{A} : we can take the composition operation to be the composition of functions, the neutral element is the identity function on \mathbf{A} , and inverses are given by the inverses of functions.

As a sub-example, consider the set

$$\mathbf{A} = \{1, 2, 3, \dots, n-1, n\}, \quad (41)$$

where $n \in \mathbb{N}$. Then $\text{Aut}(\mathbf{A})$ is the group of permutations of n elements. The usual notation for this group is Perms_n . Its size is $n! = n \cdot (n-1) \cdot \dots \cdot 3 \cdot 2 \cdot 1$.

Example 9.42. Consider the set $\{0, 1\}$, equipped with the composition operation \circ defined to be “addition modulo 2”. The composition table for this is Table 9.3

Choose 0 as the neutral element, and let $\text{inv}(0) = 0$ and $\text{inv}(1) = 1$. Then $\langle \{0, 1\}, \circ, 0, \text{inv} \rangle$ is a group.

Table 9.3.: Addition modulo 2 on the set $\{0, 1\}$.

+	0	1
0	0	1
1	1	0

Example 9.43. Consider the following set of complex numbers:

$$\left\{1, e^{\frac{1}{3}2\pi i}, e^{\frac{2}{3}2\pi i}\right\} \subseteq \mathbb{C}. \quad (42)$$

Taking the usual multiplication of complex numbers as the composition operation, these three numbers form a group.

Example 9.44. The set $\{0, 1, 2, 3\}$ may be equipped with a group structure whose composition operation is addition modulo 4, and where 0 is the neutral element. The composition table is Table 9.4.

Table 9.4.: Cyclic group of order 4.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Example 9.45. Consider the shape of a rectangle in the plane, oriented vertically, and assume the rectangle is not a square. Then there are four symmetries of this shape:

1. doing nothing (leaving everything in place);
2. reflecting the shape along the vertical axis;
3. reflecting the shape along the horizontal axis;
4. rotating the shape by 180 degrees.

Table 9.5.: The Klein four group

+	I	V	H	R
I	I	V	H	R
V	V	I	R	H
H	H	R	I	V
R	R	H	V	I

Let us call these symmetries I , V , H , and R , respectively. We can model these for instance using bijective functions of the plane \mathbb{R}^2 . The set $\{I, V, H, R\}$ can be given the structure of a group, with I being its neutral element, and with composition and inverses given as in Table 9.5. We think of composition meaning

that we perform one symmetry transformation and then the other (function composition), and inverses correspond to taking the inverse transformation. This group is known as the Klein four-group, named after the mathematician Felix Klein.

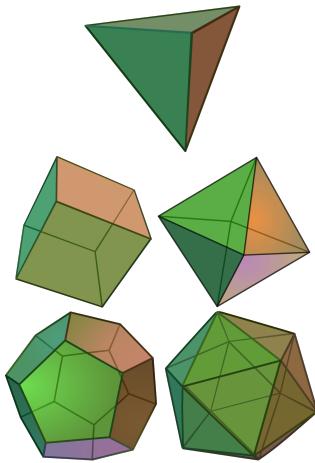


Figure 7.: The Platonic solids

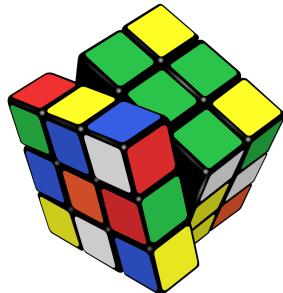


Figure 8.: A Rubik's cube

Example 9.46. Similar to the previous example, all sorts of geometric shapes have groups of symmetries associated with them.

For example, each of the five Platonic solids (Fig. 7) have a group of symmetries associated with them.

It turns out that the cube and the octahedron have the same group of symmetries, and similarly the dodecahedron and the icosahedron have the same symmetry group. This has to do with the fact that these Platonic solids are dual to each other, respectively, in the following sense. An octahedron can be obtained from a cube by replacing faces with vertices and vertices with faces, and the same goes for the dodecahedron and the icosahedron.

The symmetry group of the tetrahedron has order 24, the symmetry group of the cube and octahedron has order 48, and the symmetry group of the dodecahedron and icosahedron has order 120.

Example 9.47. There is a group that describes all the possible manipulations of a Rubik's cube (Fig. 8). The size of this group is

$$43,252,003,274,489,856,000 = 12! \cdot 2^{10} \cdot 8! \cdot 3^7. \quad (43)$$

Matrix groups

There are various *matrix groups* (Table 11.1) that represent linear transformations having special properties. Here we consider only matrices with entries from the real number field \mathbb{R} .

Definition 9.48 (General linear group $GL(n, \mathbb{R})$)

The (real) general linear group of order n , written $GL(n, \mathbb{R})$, is the group of $n \times n$ invertible matrices with entries in \mathbb{R} .

Definition 9.49 (Orthogonal group $O(n, \mathbb{R})$)

The (real) orthogonal group of order n , written $O(n, \mathbb{R})$, is the group of $n \times n$ real matrices that satisfy

$$\mathbf{M}\mathbf{M}^\top = \mathbf{M}^\top\mathbf{M} = \mathbf{1}. \quad (44)$$

Definition 9.50

Special linear group $SL(n, \mathbb{R})$] The (real) special linear group of order n , written $SL(n, \mathbb{R})$, is the group of $n \times n$ invertible real matrices with determinant equal to 1.

Definition 9.51

Special orthogonal group $SO(n, \mathbb{R})$] The (real) special orthogonal group of order n , written $SO(n, \mathbb{R})$, is the group of $n \times n$ real matrices that satisfy

$$\mathbf{M}\mathbf{M}^\top = \mathbf{M}^\top\mathbf{M} = \mathbf{1}, \quad (45)$$

and $\det(\mathbf{M}) = 1$.

Properties of groups

Lemma 9.52. Let $\langle \mathbf{G}, \circ, \text{id}, \text{inv} \rangle$ be a group. Then

1. $\text{inv}(\text{id}) = \text{id}$;
2. $\text{inv}(\text{inv}(x)) = x$, $\forall x \in \mathbf{G}$;
3. $\text{inv}(x \circ y) = \text{inv}(y) \circ \text{inv}(x)$, $\forall x, y \in \mathbf{G}$.

Lemma 9.53. Let $\mathbf{G} = \langle \mathbf{G}, \circ, \text{id}, \text{inv} \rangle$ be a group and let $x, y \in \mathbf{G}$. If x and y satisfy the equation

$$x \circ y = \text{id}, \quad (46)$$

then $y = \text{inv}(x)$ and $x = \text{inv}(y)$.

Proof. If $x \circ y = \text{id}$, then, by composing both sides of this equation with $\text{inv}(x)$ from the left, and using associativity to remove brackets, we find $\text{inv}(x) \circ x \circ y = \text{id} \circ \text{inv}(x)$. Applying the inverse laws on the left-hand side, we obtain $\text{id} \circ y = \text{id} \circ \text{inv}(x)$, and using the neutrality laws on both side of this equation yields $y = \text{inv}(x)$. The fact that $x = \text{inv}(y)$ may be proved similarly. \square

Corollary 9.54. Let $\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle$ be a monoid. If inv_1 and inv_2 are both operations of inverse which make \mathbf{M} into a group, then $\text{inv}_1 = \text{inv}_2$ holds. In other words, if a monoid can be made into a group by adding an inverse operation, then the resulting group is uniquely determined by the underlying monoid.

Proof. Let $x \in \mathbf{M}$. Since $\langle \mathbf{M}, \circ, \text{id}, \text{inv}_2 \rangle$ is a group, we have $x \circ \text{inv}_2(x) = \text{id}$. Also $\langle \mathbf{M}, \circ, \text{id}, \text{inv}_1 \rangle$ is a group, and in terms of this group, the equation $x \circ \text{inv}_2(x) = \text{id}$ implies, by Lemma 9.53, that $\text{inv}_2(x) = \text{inv}_1(x)$, by thinking of $\text{inv}_2(x)$ in the role of y from Lemma 9.53. Since $x \in \mathbf{M}$ was arbitrary, $\text{inv}_2 = \text{inv}_1$ as functions on \mathbf{M} . \square

Example 9.55 (Orthogonal matrices). Fix an integer $n \geq 1$ and consider the set of real *orthogonal* matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$: real, square matrices with orthonormal columns and rows. One way to express orthogonality of a matrix is:

$$\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T = \mathbf{1}, \quad (47)$$

where $\mathbf{1}$ is the *identity matrix*. The set $\mathbb{R}^{n \times n}$ equipped with matrix multiplication as a binary operation, the identity matrix as the neutral element, and the transposition $(\cdot)^T$ (which for this specific type of matrices corresponds to the inverse) forms a group, usually denoted $O(n)$. Any orthogonal matrix \mathbf{A} has the property $\det(\mathbf{A}) \in \{-1, +1\}$. The subset of orthogonal $n \times n$ matrices with determinant 1 forms the so-called *special* orthogonal group $SO(n, \mathbb{R})$.

Graded exercise D.4 (GroupWithThreeElements)

In Example 9.43, what is the neutral element? What is the inverse operation? Draw the composition table for this group.

Graded exercise D.5 (GroupInverseProperties)

Prove Lemma 9.52.

Remark 9.56. Just like for semigroups and monoids, any group has an opposite. It is defined analogously.

Subgroups

Definition 9.57 (Subgroup)

Let $\mathbf{G} = \langle \mathbf{G}, \circ, \text{id}, \text{inv} \rangle$ be a group. A *subgroup* of \mathbf{G} is:

Constituents

1. A subset $\mathbf{H} \subseteq \mathbf{G}$.

Conditions

1. The set \mathbf{H} is closed under the composition operation \circ from \mathbf{G} :

$$\frac{x \in \mathbf{H} \quad y \in \mathbf{H}}{x \circ y \in \mathbf{H}}; \quad (48)$$

2. The neutral element $\text{id} \in \mathbf{G}$ is an element of \mathbf{H} ;

3. The set \mathbf{H} is closed under the inverse operation inv from \mathbf{G} :

$$\frac{x \in \mathbf{H}}{\text{inv}(x) \in \mathbf{H}}. \quad (49)$$

9.5. Rings, fields

Definition 9.58 (Ring)

A *ring* is

Constituents

1. a set \mathbf{R} ;
2. a binary operation $\cdot : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$, called *multiplication*;
3. a binary operation $+ : \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$, called *addition*;
4. a specified element $1 \in \mathbf{R}$ called *one* (or *unit*);
5. a specified element $0 \in \mathbf{R}$ called *zero*.

Conditions

1. $\langle \mathbf{R}, \cdot \rangle$ is a monoid, with neutral element 1;
2. $\langle \mathbf{R}, + \rangle$ is a commutative group, with neutral element 0;
3. Distributive law:

$$\mathbf{x} \cdot (\mathbf{y} + \mathbf{z}) = (\mathbf{x} \cdot \mathbf{y}) + (\mathbf{x} \cdot \mathbf{z}), \quad \forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{R}. \quad (50)$$

Definition 9.59 (Field)

A field is a ring $\langle \mathbf{K}, \cdot, +, 1, 0 \rangle$ such that $\langle \mathbf{K} \setminus \{0\}, \cdot \rangle$ is a commutative group.



10. Morphisms

In this chapter we look at morphisms, which are maps between two semigroups (or monoids, groups) that “preserve the structure”.

10.1 Semigroup morphisms	152
10.2 Encoding as morphism	154
10.3 Morse coding	155
10.4 Monoid morphisms	157
10.5 Group morphisms	159
10.6 Generators and relations	161

A car shuttle train, is a shuttle train used to transport cars, and usually also other types of road vehicles, for a relatively short distance. Switzerland is well known for many such shuttle trains, usually operating through tunnels.

10.1. Semigroup morphisms

A *morphism* is a map between semigroups that “preserves the structure” of composition.

$$\begin{array}{ccc} S \times S & \xrightarrow{F \times F} & T \times T \\ \downarrow \circ_S & & \downarrow \circ_T \\ S & \xrightarrow{F} & T \end{array}$$

Figure 1.: Semigroup Morphism

Definition 10.1 (Semigroup morphism)

A morphism $F : S \rightarrow T$ between semigroups

$$S = \langle S, \circ_S \rangle \quad \text{and} \quad T = \langle T, \circ_T \rangle \quad (1)$$

is a function $F : S \rightarrow T$ such that for all $x, y \in S$,

$$F(x \circ_S y) = F(x) \circ_T F(y). \quad (2)$$

Note that we use $F : S \rightarrow T$ when we want to highlight the function between sets, and we use $F : S \rightarrow T$ when we want to highlight the relation between semigroup structures. We think of (2) as a way of saying that the function $F : S \rightarrow T$ is *compatible* with the multiplication operations on S and T , respectively.

Definition 10.2 (Identity morphism)

Let S be a semigroup. The identity function $\text{id}_S : S \rightarrow S$ is always a morphism of semigroups. We can easily check that (2) is satisfied:

$$\text{id}_S(x \circ_S y) = x \circ_S y = \text{id}_S(x) \circ_S \text{id}_S(y). \quad (3)$$

We call this the *identity morphism* of S .

Definition 10.3 (Semigroup isomorphism)

A morphism of semigroups $F : S \rightarrow T$ is called a *semigroup isomorphism* if there exists a morphism of semigroups $G : T \rightarrow S$ such that

$$F ; G = \text{id}_S \quad \text{and} \quad G ; F = \text{id}_T. \quad (4)$$

Lemma 10.4. The composition of semigroup morphisms is a morphism:

$$\frac{F : S \rightarrow T \quad G : T \rightarrow U}{(F ; G) : S \rightarrow U}. \quad (5)$$

Exercise 26. Prove Lemma 10.4.

See solution on page 175.

Example 10.5 (Logarithms and exponentials). The positive reals with multiplication $\langle \mathbb{R}_{>0}, \cdot \rangle$ is a semigroup. The reals with addition $\langle \mathbb{R}, + \rangle$ is a semigroup. Now consider as a bridge between the two: the logarithmic function. We have

$$\log : \mathbb{R}_{>0} \rightarrow \mathbb{R}, \quad (6)$$

and its inverse

$$\exp : \mathbb{R} \rightarrow \mathbb{R}_{>0}. \quad (7)$$

We already know that these are inverse of each other:

$$\begin{aligned} \exp ; \log &= \text{id}_{\mathbb{R}}, \\ \log ; \exp &= \text{id}_{\mathbb{R}_{>0}}. \end{aligned} \quad (8)$$

We can verify that \log is also a semigroup morphism, because of the following property of the logarithms:

$$\log(a \cdot b) = \log(a) + \log(b). \quad (9)$$

Because \log is a bijection and \exp is its inverse, it already follows that \exp is a morphism in the opposite direction. Alternatively we can see that is the case because of the property of the exponential function:

$$\exp(c + d) = \exp(c) \cdot \exp(d). \quad (10)$$

(9) and (10) are both (2) in disguise.

Example 10.6 (Transition function, continuation of Section 9.3). Consider the map

$$f : \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^n) \quad (11)$$

that associates to a delta δ its transition function T_δ . Re-reading (34), we can see that it is a morphism between the semigroup $\langle \mathbb{R}_{\geq 0}, + \rangle$ and the semigroup of endomorphisms of \mathbb{R}^n .

Graded exercise D.6 ([IsoViaTables](#))

Consider the set $\mathbf{A} = \{\circlearrowleft, \rightsquigarrow\}$ and the following three composition tables, each of which defines a semigroup structure on \mathbf{A} .

\circlearrowleft_1	\circlearrowleft	\rightsquigarrow
\circlearrowleft	\circlearrowleft	\circlearrowleft
\rightsquigarrow	\rightsquigarrow	\rightsquigarrow

\circlearrowleft_2	\circlearrowleft	\rightsquigarrow
\circlearrowleft	\circlearrowleft	\rightsquigarrow
\rightsquigarrow	\rightsquigarrow	\rightsquigarrow

\circlearrowleft_3	\circlearrowleft	\rightsquigarrow
\circlearrowleft	\rightsquigarrow	\rightsquigarrow
\rightsquigarrow	\rightsquigarrow	\rightsquigarrow

Which of the three semigroups defined in this way are isomorphic to each other? Justify your answer.

Graded exercise D.7 ([SemigroupUpToIso](#))

How many different non-isomorphic semigroups are there with precisely one element? How many with precisely two elements? Can you prove your answer?

Graded exercise D.8 ([CharacterizeSemigroupIsos](#))

Let $F : S \rightarrow T$ be a morphism of semigroups. Prove that F is an isomorphism of semigroups if and only if the function $F : S \rightarrow T$ is bijective.

10.2. Encoding as morphism

Example 10.7 (ASCII code). ASCII encoding takes any alphanumerical characters and symbols into a number between 0 and 127 (Fig. 2). Call char the set of those 128 symbols. We can see ASCII encoding as a semigroup morphism of List char to the free semigroup on the integers $\text{List}[0, 127]$:

$$\text{ASCII} : \text{List char} \rightarrow \text{List}[0, 127]. \quad (12)$$

Because we can also go back, by using the inverse function,

$$\text{ASCII}^{-1} : \text{List}[0, 127] \rightarrow \text{List char}, \quad (13)$$

ASCII encoding is also an isomorphism of semigroups.

Example 10.8 (ASCII code to binary). Currently, computers use binary to store data. (There were, in fact, *trinary* computers.) In Fig. 2, you can see represented also the binary encoding of each character. Therefore, we can see ASCII as a morphism between List char and binary lists $\text{List}\{0, 1\}$.

Exercise 27. Show that the morphism

$$\text{ASCII} : \text{List char} \rightarrow \text{List}\{0, 1\} \quad (14)$$

is *not* an isomorphism.

See solution on page 175.

USASCII code chart																
$b_7 b_6 b_5$					$b_4 b_3 b_2 b_1$		Column		Row		0 0 0 0		0 0 1		0 1 0	
											0	1	2	3	4	5
											NUL	DLE	SP	@	P	'
0	0	0	0	0	0	0	0	0	0	0	SOH	DC1	!	1	A	Q
0	0	0	0	1	1	1	1	1	1	1	STX	DC2	"	2	B	R
0	0	0	1	0	2	2	2	2	2	2	ETX	DC3	#	3	C	S
0	0	0	1	1	3	3	3	3	3	3	EOT	DC4	*	4	D	T
0	0	1	0	0	4	4	4	4	4	4	ENQ	NAK	%	5	E	U
0	0	1	0	1	5	5	5	5	5	5	ACK	SYN	8	6	F	V
0	0	1	1	0	6	6	6	6	6	6	BEL	ETB	'	7	G	W
0	0	1	1	1	7	7	7	7	7	7	BS	CAN	(8	H	X
0	0	1	0	0	8	8	8	8	8	8	HT	EM)	9	I	Y
0	0	1	0	1	9	9	9	9	9	9	LF	SUB	*	:	J	Z
0	0	1	0	0	10	10	10	10	10	10	VT	ESC	+	;	K	[
0	0	1	0	1	11	11	11	11	11	11	FF	FS	,	<	L	\
0	0	1	1	0	12	12	12	12	12	12	CR	GS	-	=	M]
0	0	1	1	0	13	13	13	13	13	13	SO	RS	.	>	N	^
0	0	1	1	1	14	14	14	14	14	14	S1	US	/	?	O	-
0	0	1	1	1	15	15	15	15	15	15	DEL				o	~

Figure 2.: 7-bit US-ASCII encoding.

10.3. Morse coding

Example 10.9 (Morse code). Consider the Morse code: a way to encode the letters and numerals to an alphabet of dots (\cdot) and dashes ($-$). The encoding is shown in Table 10.1. Here, the alphabet $mchar$ is the letters A–Z and the numbers 0–9:

$$mchar = (\text{A to Z}) \cup (0 \text{ to } 9). \quad (15)$$

There is no difference between upper and lower case, and there are no punctuation marks.

Transcribing a text in Morse code is not just a matter of creating the right sequence of dots and dashes. The standard also requires a certain timing of the events. These are the rules:

1. If the length of \cdot is ℓ , then the length of $-$ must be 3ℓ .
2. Between dashes and dots of the same letter, there must be a silence of ℓ .
3. Between different letters, there must be a silence of 3ℓ .
4. Between different words, there must be a silence of 7ℓ .

Therefore, there are 5 symbols in the Morse alphabet (Table 10.2); each representing a *signal*.

Define now the extended alphabet $emchar$ to be the union of $mchar$ and the set $\{\textcolor{red}{I}, \textcolor{green}{W}\}$, where $\textcolor{red}{I}$ is inter-letter space, and $\textcolor{green}{W}$ is inter-word space:

$$emchar = (\text{A to Z}) \cup (0 \text{ to } 9) \cup \{\textcolor{red}{I}, \textcolor{green}{W}\}. \quad (16)$$

Therefore, to encode the sentence

$$\text{"I am well"} \quad (17)$$

we first transform it to upper case:

$$\text{"I AM WELL"}. \quad (18)$$

Then we note the inter-letter space and the inter-word spaces:

$$\textcolor{red}{I} \textcolor{green}{A} \textcolor{red}{I} \textcolor{red}{M} \textcolor{green}{W} \textcolor{red}{I} \textcolor{red}{E} \textcolor{red}{I} \textcolor{red}{L} \textcolor{red}{I} \textcolor{green}{L}. \quad (19)$$

At this point we can substitute the Morse code to obtain

$$\cdot s_1 \cdot s_7 \cdot s_1 - s_3 - s_1 - s_7 \cdot s_1 - s_1 - s_3 \cdot s_3 - s_3 -. \quad (20)$$

In signal space—what somebody would hear—this becomes

$$(21)$$

With this representation it is clear that 5 symbols are redundant: if we have a 1-period beep and a 1-period silence, we can obtain the 3-period silence and beeps and the 7-period silence.

In the end, the Morse alphabet is *binary* in the sense that it all reduces to two symbols: not $\{\cdot, -\}$ but rather the alphabet $\{\blacksquare, \blacksquare\}$.

Exercise 28. [Morse Morphism] We have seen that Morse code transforms a word in the alphabet

$$(\text{A to Z}) \cup (0 \text{ to } 9) \cup \{\textcolor{red}{I}, \textcolor{green}{W}\} \quad (22)$$

to the binary alphabet

$$\mathbf{B} = \{\blacksquare, \blacksquare\}. \quad (23)$$

Table 10.1.: Morse encoding

A	$\cdot -$			
B	$- \cdot \cdot \cdot$			
C	$- \cdot \cdot \cdot \cdot$			
D	$- \cdot \cdot \cdot$			
E	\cdot			
F	$\cdot \cdot \cdot \cdot$			
G	$- \cdot \cdot \cdot$			
H	$\cdot \cdot \cdot \cdot$			
I	$\cdot \cdot$	0	-----	
J	$\cdot \cdot \cdot \cdot \cdot$	1	$\cdot \cdot \cdot \cdot \cdot$	
K	$\cdot \cdot \cdot -$	2	$\cdot \cdot \cdot \cdot \cdot$	
L	$\cdot \cdot \cdot \cdot$	3	$\cdot \cdot \cdot \cdot \cdot$	
M	$\cdot \cdot -$	4	$\cdot \cdot \cdot \cdot \cdot$	
N	$\cdot \cdot \cdot$	5	$\cdot \cdot \cdot \cdot \cdot$	
O	$\cdot \cdot \cdot \cdot$	6	$\cdot \cdot \cdot \cdot \cdot$	
P	$\cdot \cdot \cdot \cdot \cdot$	7	$\cdot \cdot \cdot \cdot \cdot$	
Q	$\cdot \cdot \cdot \cdot -$	8	$\cdot \cdot \cdot \cdot \cdot$	
R	$\cdot \cdot \cdot -$	9	$\cdot \cdot \cdot \cdot \cdot$	
S	$\cdot \cdot \cdot$			
T	$-$			
U	$\cdot \cdot \cdot$			
V	$\cdot \cdot \cdot \cdot$			
W	$\cdot \cdot \cdot \cdot$			
X	$\cdot \cdot \cdot \cdot \cdot$			
Y	$\cdot \cdot \cdot \cdot \cdot$			
Z	$\cdot \cdot \cdot \cdot \cdot$			

Table 10.2.: 5 symbols for Morse encoding

\cdot	beep of length ℓ	
$-$	beep of length 3ℓ	
s_1	silence of length ℓ	
s_3	silence of length 3ℓ	
s_7	silence of length 7ℓ	

Is this map a morphism of list semigroups?

See solution on page 175.

10.4. Monoid morphisms

We have defined semigroup morphisms. A monoid morphism has the same properties, and one additional one: the constraint that it be compatible with the identity elements.

Definition 10.10 (Monoid morphism)

A morphism $F : \mathbf{M} \rightarrow \mathbf{N}$ between monoids

$$\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle \quad \text{and} \quad \mathbf{N} = \langle \mathbf{N}, \circ_{\mathbf{N}}, \text{id}_{\mathbf{N}} \rangle \quad (24)$$

is a function $F : \mathbf{M} \rightarrow \mathbf{N}$ such that for all $x, y \in \mathbf{M}$,

$$F(x \circ_{\mathbf{M}} y) = F(x) \circ_{\mathbf{N}} F(y), \quad (25)$$

and

$$F(\text{id}_{\mathbf{M}}) = \text{id}_{\mathbf{N}}. \quad (26)$$

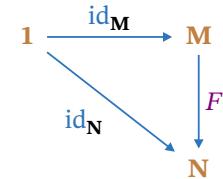


Figure 3.: Compatibility with identity elements

Example 10.11. The set $\mathbf{M} = \{-1, 0, +1\}$, together with multiplication of whole numbers and with 1 as neutral element, forms a monoid. The inclusion map $\mathbf{M} \rightarrow \mathbb{Z}$ is a morphism of monoids.

Example 10.12. Consider the monoid \mathbf{N} in Example 9.28 and the monoid $\mathbf{M} = \langle \mathbb{N}, +, 0 \rangle$. Define a map

$$\text{length} : \mathbf{N} \rightarrow \mathbf{M} \quad (27)$$

that maps each string to its length. This is a monoid morphism, since:

$$\text{length}(x \circ_{\mathbf{N}} y) = \text{length}(x) \circ_{\mathbf{M}} \text{length}(y). \quad (28)$$

In other words, the length of the concatenation of two lists is the sum of the lengths of the two lists.

Definition 10.13 (Identity morphism)

Let \mathbf{M} be a monoid. Similar to the case of semigroups, the identity function induces a morphism of monoids $\text{id}_{\mathbf{M}} : \mathbf{M} \rightarrow \mathbf{M}$.

Definition 10.14 (Monoid isomorphism)

A morphism of monoids $F : \mathbf{M} \rightarrow \mathbf{N}$ is called a *monoid isomorphism* if there is a morphism of monoids $G : \mathbf{N} \rightarrow \mathbf{M}$ such that

$$F \circ G = \text{id}_{\mathbf{N}} \quad \text{and} \quad G \circ F = \text{id}_{\mathbf{M}}. \quad (29)$$

Graded exercise D.9 ([Morphism](#)[Monoid](#)[Isomorphism](#))

Prove: a morphism of monoids $F : \mathbf{M} \rightarrow \mathbf{N}$ is an isomorphism of monoids if and only if the function $F : \mathbf{M} \rightarrow \mathbf{N}$ is bijective.

Graded exercise D.10 ([Trace](#)[And](#)[Determinant](#))

Let $\mathbb{R}^{n \times n}$ denote the set of real $n \times n$ matrices, $n \in \mathbb{N}$. These form a monoid $\langle \mathbb{R}^{n \times n}, \cdot, \mathbf{1} \rangle$, with matrix multiplication as composition, and the identity matrix as neutral element. They also form a monoid $\langle \mathbb{R}^{n \times n}, +, \mathbf{0} \rangle$ with matrix addition as composition, and the zero matrix as neutral element. We also note that that $\langle \mathbb{R}, \cdot, 1 \rangle$ and $\langle \mathbb{R}, +, 0 \rangle$ are two different monoid structures on \mathbb{R} .

Recall that the trace of a real $n \times n$ matrix is the sum of its diagonal entries. This defines a function

$$\text{Tr} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}. \quad (30)$$

Computing the determinant also corresponds to a function

$$\det : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}. \quad (31)$$

1. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbf{1} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
2. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbf{1} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
3. Does \det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbf{1} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
4. Does \det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbf{1} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
5. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, \mathbf{0} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
6. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, \mathbf{0} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
7. Does \det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, \mathbf{0} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
8. Does \det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, \mathbf{0} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?

Short answers (without proof) are fine.

10.5. Group morphisms

After semigroup morphisms and monoid morphisms, we define group morphisms.

Definition 10.15 (Group morphism)

A morphism $F : \mathbf{G} \rightarrow \mathbf{H}$ between groups

$$\mathbf{G} = \langle \mathbf{G}, \circ_{\mathbf{G}}, \text{id}_{\mathbf{G}}, \text{inv}_{\mathbf{G}} \rangle \quad \text{and} \quad \mathbf{H} = \langle \mathbf{H}, \circ_{\mathbf{H}}, \text{id}_{\mathbf{H}}, \text{inv}_{\mathbf{H}} \rangle \quad (32)$$

is a function $F : \mathbf{G} \rightarrow \mathbf{H}$ such that for all $x, y \in \mathbf{G}$,

$$F(x \circ_{\mathbf{G}} y) = F(x) \circ_{\mathbf{H}} F(y). \quad (33)$$

What could be surprising is that, while a group has more structure than a monoid, there are fewer conditions than in the definition of monoid morphism.

Where are the equations

$$F(\text{id}_{\mathbf{G}}) = \text{id}_{\mathbf{H}} \quad (34)$$

and

$$F(\text{inv}_{\mathbf{G}}(x)) = \text{inv}_{\mathbf{H}}(F(x)) ? \quad (35)$$

The answer is that they are not needed, because they can be deduced from the group axioms (and so we omit them, because they don't need to be checked when we want to know if something is a group morphism or not).

Exercise 29. Prove that (34) and (35) are implied by Def. 10.15.

See solution on page 176.

Exercise 30. Let $\mathbf{G} = \{+1, -1, +i, -i\}$ where i is the imaginary unit. Consider the group $\mathbf{G} = \langle \mathbf{G}, \cdot, 1, \text{inv}_{\mathbf{G}} \rangle$ and the group \mathbf{H} of all integers under addition. Prove that $F : \mathbf{H} \rightarrow \mathbf{G}$ such that $f(n) = i^n$ for all $n \in \mathbf{H}$ is a group morphism.

See solution on page 176.

Example 10.16. Consider the group

$$\mathbf{G} = \left\langle \mathbb{R}^{2 \times 2}, +, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, - \right\rangle \quad (36)$$

of real 2×2 matrices, together with sum of matrices as a binary operation, “zero” matrix as the neutral element, and the “-” operation as inverse. Furthermore, consider the group $\langle \mathbb{R}, +, 0, - \rangle$. Taking the *trace* of a matrix corresponds to a group morphism. Indeed, the operation

$$\begin{aligned} \text{Tr} : \mathbb{R}^{2 \times 2} &\rightarrow \mathbb{R}, \\ \begin{bmatrix} a & b \\ c & d \end{bmatrix} &\mapsto a + d, \end{aligned} \quad (37)$$

satisfies the required condition:

$$\begin{aligned} \text{Tr}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix} \circ_{\mathbf{G}} \begin{bmatrix} e & f \\ g & h \end{bmatrix}\right) &= \text{Tr}\left(\begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}\right) \\ &= \text{Tr}\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) \circ_{\mathbf{H}} \text{Tr}\left(\begin{bmatrix} e & f \\ g & h \end{bmatrix}\right) \\ &= (a+d) \circ_{\mathbf{H}} (e+h). \end{aligned} \quad (38)$$

Example 10.17 (Square matrices with full rank). Fix an integer $n \geq 1$ and consider the set of square matrices with full rank $\mathbf{A} \in \mathbb{R}^{n \times n}$, which is to say $\det(\mathbf{A}) \neq 0$. This set, equipped with the usual matrix multiplication as the binary operation $(\mathbf{A} ; \mathbf{B} := \mathbf{AB})$, the identity matrix $\mathbf{1}$ as the neutral element, and matrix inverse as the inverse $(\text{inv}(\mathbf{A}) := \mathbf{A}^{-1})$, forms a group. Furthermore, note that for this type of matrices, we have the properties:

1. $\det(\mathbf{AB}) = \det(\mathbf{A}) \cdot \det(\mathbf{B})$;
2. $\det(\mathbf{A}^{-1}) = (\det(\mathbf{A}))^{-1}$;
3. $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$.

This makes \det a group morphism from the group of invertible square matrices to the real numbers with multiplication.

As before for semigroups and monoids, the identity map on the underlying set of a group is a group morphism.

Definition 10.18 (Identity morphism)

Given a group $\mathbf{G} = \langle \mathbf{G}, ;, \text{id}, \text{inv} \rangle$, the identity function $\mathbf{G} \rightarrow \mathbf{G}$ induces a morphism of groups $\text{id}_{\mathbf{M}} : \mathbf{M} \rightarrow \mathbf{M}$.

Definition 10.19 (Group isomorphism)

A morphism of monoids $F : \mathbf{G} \rightarrow \mathbf{H}$ is called a *group isomorphism* if there is a morphism of groups $G : \mathbf{H} \rightarrow \mathbf{G}$ such that

$$F ; G = \text{id}_{\mathbf{G}} \quad \text{and} \quad G ; F = \text{id}_{\mathbf{H}}. \quad (39)$$

10.6. Generators and relations

Generating subsets

In Example 9.12 we considered a set of states

$$\mathbf{A} = \{\text{sprout, young, mature, old, dead}\}, \quad (40)$$

a function $f : \mathbf{A} \rightarrow \mathbf{A}$, and the semigroup

$$\mathbf{S} = \{f^n \mid n \in \mathbb{N}\}. \quad (41)$$

Note that \mathbf{S} has a special form: all of its elements can be expressed in terms one of its elements, f , and the multiplication operation (which, in this case, is function composition). To describe this state of affairs we say that \mathbf{S} is *generated* by the element f .

Definition 10.20 (Generating subsets)

Let $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ be a semigroup, and let $\mathbf{A} \subseteq \mathbf{S}$ be a subset. We say that \mathbf{S} is *generated* by \mathbf{A} if every element of \mathbf{S} can be expressed as a finite composition of elements of \mathbf{A} .

Remark 10.21. Mutatis mutandis, the same definition also holds for monoids. For groups, we say \mathbf{A} generates the group if every element of the group can be expressed as a finite composition of elements of \mathbf{A} or their inverses.

Example 10.22. Consider Example 9.10, where elements of the semigroup \mathbf{S} were non-empty lists built using the elements of the “alphabet” set $\mathbf{A} = \{\bullet, \circ\}$. In this case, \mathbf{S} is generated by \mathbf{A} .

Example 10.23. Consider the natural numbers (without zero) as a semigroup, where addition is the semigroup composition operation (see Example 9.7). This semigroup is generated by the subset $\{1\}$.

Relations

Let us return to the semigroup (41). Recall that f was defined by

$$f(\text{sprout}) = \text{young}, \quad (42)$$

$$f(\text{young}) = \text{mature}, \quad (43)$$

$$f(\text{mature}) = \text{old}, \quad (44)$$

$$f(\text{old}) = \text{dead}, \quad (45)$$

$$f(\text{dead}) = \text{dead}. \quad (46)$$

Observe that the function f^4 will map all elements of \mathbf{A} to the element “dead”. For example, if we start with the element “sprout”, the result of applying f four times is

$$\text{sprout} \xrightarrow{f} \text{young} \xrightarrow{f} \text{mature} \xrightarrow{f} \text{old} \xrightarrow{f} \text{dead}. \quad (47)$$

Note also that for *any* $n \geq 4$, the function f^n will map all elements of \mathbf{A} to the element “dead”. If we consider f^6 , for example, then, for any $x \in \mathbf{A}$,

$$f^6(x) = f^2(f^4(x)) = f^2(\text{dead}) = f(f(\text{dead})) = f(\text{dead}) = \text{dead}. \quad (48)$$

It follows that all f^n , for $n \geq 4$, are actually *all the same map*: the one that sends

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

every state to the dead state. Thus, $\mathbf{S} = \{\textcolor{blue}{f}^n \mid n \in \mathbb{N}\}$ actually only has at most four elements: $\textcolor{blue}{f}$, $\textcolor{blue}{f}^2$, $\textcolor{blue}{f}^3$, and $\textcolor{blue}{f}^4$.

Graded exercise D.11 (CheckRelations)

Are any of the four maps $\textcolor{blue}{f}$, $\textcolor{blue}{f}^2$, $\textcolor{blue}{f}^3$, and $\textcolor{blue}{f}^4$ actually equal? Justify your answer by argumentation or by explicitly checking via calculation.

When two elements which a priori could be distinct from each other (such as $\textcolor{blue}{f}^6$ and $\textcolor{blue}{f}^4$ above, for example) turn out to be equal, we call this a *relation* between the elements of \mathbf{S} .

Definition 10.24

A *relation* on a semigroup $\langle \mathbf{S}, \circ \rangle$ is an equation between compositions of elements of \mathbf{S} .

Remark 10.25. Again, we have analogous definitions for monoids and groups. In these cases, we interpret the neutral element id as a “zero-fold” multiplication, so it can also be part of equations that express relations.

Remark 10.26. This is not the same notion as that of a (binary) relation, which was the topic of Chapter 4 and takes up a much more important role in this book than the notion that we are discussing here.

Example 10.27. For the semigroup (41), the relations $\textcolor{blue}{f}^5 = \textcolor{blue}{f}^4$, $\textcolor{blue}{f}^6 = \textcolor{blue}{f}^5$, and $\textcolor{blue}{f}^6 = \textcolor{blue}{f}^4$, etc. are satisfied. However, the relation $\textcolor{blue}{f}^3 = \textcolor{blue}{f}$ is not satisfied, for example.

Example 10.28. Consider the semigroup $\langle \mathbb{N}, + \rangle$. The equation $l + k = k + l$ is an example of a relation that holds for all $l, k \in \mathbb{N}$.

Example 10.29. Consider the group \mathbf{G} discussed in Example 9.43, where

$$\mathbf{G} = \left\{ 1, e^{\frac{1}{3}2\pi i}, e^{\frac{2}{3}2\pi i} \right\} \subseteq \mathbb{C} \quad (49)$$

and the composition operation is multiplication of complex numbers. The element $x := e^{\frac{1}{3}2\pi i}$ satisfies the relation $x^3 = \text{id}_{\mathbf{G}}$.

Example 10.30. Consider the group \mathbf{G} given in Example 9.45 which describe symmetries of a rectangle. We had

$$\mathbf{G} = \{I, V, H, R\} \quad (50)$$

where $I = \text{id}$ corresponds to “doing nothing”, V is reflecting the rectangle along its long axis, H is reflecting on the short axis, and R is rotation by 180 degrees. In this group, the relations $V^2 = I$, $H^2 = I$, $R^2 = I$ are satisfied, for example.

Freeness

When, in Def. 10.20 we spoke about the semigroup $\mathbf{S} = \langle \mathbf{S}, \circ \rangle$ being generated by a subset $\mathbf{A} \subseteq \mathbf{S}$, we supposed that we already had a semigroup \mathbf{S} to work with. However, if we start with just a set, say $\mathbf{A} = \{\bullet, \bullet\}$, then we saw in Example 9.10 that we can build a semigroup from this set by considering lists of elements of \mathbf{A} , with concatenation as the composition operation. The resulting semigroup in that example has a special characteristic: its elements do not satisfy any relations

other than the ones that are required by the definition of a semigroup, namely those relations dictated by the associative law. Such a semigroup is called *free*. If we think of relations as “constraints” (they are equations) between the elements of a semigroup, then free semigroups are “free of constraints”.

For a given set, say $\mathbf{A} = \{\bullet, \bullet\}$, there will in general be different ways of formally building a free semigroup from it. For instance, instead of considering lists of elements of \mathbf{A}

$$[\bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet], \quad (51)$$

we could instead consider strings of elements

$$\bullet \bullet \bullet \bullet \bullet, \quad (52)$$

or tuples of elements

$$\langle \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet, \bullet \rangle, \quad (53)$$

both of which could also be composed in a way which is analogous to concatenation.

A common feature of all three of these formalizations is that we are writing a finite sequence of elements of \mathbf{A} , keeping account of the ordering. Both approaches will in fact build a semigroup from the set \mathbf{A} which is *free*. And in both cases there is a natural way of seeing \mathbf{A} as *generating* the resulting semigroup. However, the two set-ups are *formally* distinct because we are using a different way of writing things down with symbols. We will see in a later chapter that the resulting semigroups are essentially “the same” (they are isomorphic) and their “freeness” can be given an elegant characterization.

Because in fact *all* free semigroups constructed from \mathbf{A} are “the same”, independent of the formal symbolic specifics of how they are constructed, we refer to them all as *the* free semigroup generated by \mathbf{A} . Furthermore, if we were to work with a set $\mathbf{B} = \{a, b\}$ instead of $\mathbf{A} = \{\bullet, \bullet\}$, and generate a free semigroup from \mathbf{B} , then this would also produce a semigroup which is “the same”. Therefore, sometimes one speaks simply of “the free group on two generators”, in view of the fact that the essential feature is that both \mathbf{A} and \mathbf{B} have two elements.

Note that although \mathbf{A} and \mathbf{B} each only have two elements, the free semigroups that they generate will have infinitely many elements. Indeed, there are infinitely many lists that we can build from the elements of \mathbf{A} . As we concatenate lists, the resulting compositions grow longer and longer, and there are no relations which would allow us to “simplify” a string to one which is shorter.



11. Actions

We have talked about semigroups, monoids, and groups as mathematical structures in isolation. In this chapter we discuss the concept of *action* of a semigroup on *another set*. For example, we discuss how linear transformations are a semigroup that *act* on vectors.

11.1 Actions	166
11.2 Modules, Vector spaces	170
11.3 Linear group actions	171
11.4 Dynamical systems	173

11.1. Actions

Monoid actions

$$\begin{array}{ccc}
 M \times (M \times A) & \xrightarrow{id_M \times act} & M \times A \\
 \downarrow as & & \downarrow act \\
 (M \times M) \times A & & \\
 \downarrow \circ_S \times id_A & & \\
 M \times A & \xrightarrow{act} & A
 \end{array}$$

Figure 1.: Compatibility of left-action operation with composition.

Definition 11.1 (Monoid left action operation)

A *left action operation* of a monoid $\mathbf{M} = \langle M, \circ_M, id_M \rangle$ on a set A is:

Constituents

1. a function

$$act : M \times A \rightarrow A. \quad (1)$$

Conditions

1. for all $a \in A$ and $x, y \in M$,

$$act(x, act(y, a)) = act(x \circ_M y, a); \quad (2)$$

2. for all $a \in A$,

$$act(id_M, a) = a. \quad (3)$$

Equation (2) says that if we first apply the action of x to obtain $act(x, a)$, and then we apply the action of y $act(y, -)$, it is the same thing as applying the action of $x \circ_M y$.

Remark 11.2. Our above definition of a monoid left action operation corresponds to what is often called a “left action”. Such action operations are often denoted using infix notation of the kind

$$M \times A \rightarrow A, \langle x, a \rangle \mapsto x \cdot a \quad (4)$$

or even simply

$$M \times A \rightarrow A, \langle x, a \rangle \mapsto xa \quad (5)$$

(in regard to the latter, think of the notation often used for multiplication of matrices with vectors). If we write our notation “ $x \circ_M y$ ” as “ $x \star y$ ” (or even simply “ xy ”), then (2) reads as

$$x \cdot (y \cdot a) = (x \star y) \cdot a, \quad (6)$$

or simply

$$x(ya) = (xy)a. \quad (7)$$

In other words, we can view (2) as a kind of “associative law”, but where two of the three elements come from M , while one of the elements comes from A .

Definition 11.3 (Monoid right action operation)

A *right action operation* of a monoid $\mathbf{M} = \langle M, \circ_M, id_M \rangle$ on a set A is:

Constituents

1. a function

$$act : A \times M \rightarrow A. \quad (8)$$

Conditions

1. for all $a \in A$ and $x, y \in M$,

$$act(act(a, x), y) = act(a, x \circ_M y); \quad (9)$$

2. for all $a \in A$,

$$\text{act}(a, \text{id}_M) = a. \quad (10)$$

There is a way to reformulate the notions of left and right monoid action operations which leads to a compact definition which we will generalize later. To derive this definition, we'll start with the definition Def. 11.3 of a right action operation, and then apply “left currying”.

Recall that left currying is done by applying the canonical isomorphism

$$\text{cu} : C^{A \times B} \rightarrow (C^A)^B, \quad (11)$$

which transforms any function of the type

$$A \times B \rightarrow C \quad (12)$$

into one of the type

$$B \rightarrow (A \rightarrow C). \quad (13)$$

Applying this to $\text{act} : A \times M \rightarrow A$ we obtain a function

$$\text{cu}(\text{act}) : M \rightarrow (A \rightarrow A). \quad (14)$$

Since we can tell the difference between act and $\text{cu}(\text{act})$ based on their respective domains and codomains, so we now simply use the same name act for both functions, and drop the “ cu ”. Furthermore, we will usually write

$$\text{act} : M \rightarrow \text{End}(A), \quad (15)$$

using the notation $\text{End}(A)$ for functions of type $A \rightarrow A$. These are the endomorphisms of A .

Now, so far, we have just used currying to produce the function (15) from the a monoid action operation. Is (15) really also a homomorphisms of monoids?

To see that this is the case, consider first the second condition in the definition of a right action operation, given by (10). Translating this to the function (15), this means that

$$\text{act}(\text{id}_M)(a) = a \quad \forall a \in A, \quad (16)$$

or, in other words, that

$$\text{act}(\text{id}_M) = \text{id}_A. \quad (17)$$

Since the identity function id_A is the identity element of the monoid $\text{End}(A)$, this shows that the function (15) respects the identity elements of the monoids which are its source and target.

Now let's take a look at the first condition in the definition of a right action operation. For any $x, y \in M$, we have

$$\text{act}(\text{act}(a, x), y) = \text{act}(a, x \circ_M y) \quad \forall a \in A. \quad (18)$$

If we translate this, via currying, into an equality of functions in $\text{End}(A)$, we obtain

$$\text{act}(x) \circ_{\text{End}(A)} \text{act}(y) = \text{act}(x \circ_S y). \quad (19)$$

In other words, act respects the composition operations defined on its source and target. In sum, we have found that act is indeed monoid homomorphism (Def. 10.10), and this leads us to the the following compact definition.

Definition 11.4 (Monoid action)

An *action* of a monoid \mathbf{M} on a set \mathbf{A} is a monoid homomorphism

$$\text{act} : \mathbf{M} \rightarrow \mathbf{End}(\mathbf{A}). \quad (20)$$

Observe that this definition makes no reference to “left” and “right”, even though we derived it from the definition of a *right* action operation. What about left action operations?

Graded exercise D.12 (`LeftActionCurry`)

Show that a left action operation of a monoid \mathbf{M} on a set \mathbf{A} corresponds to a monoid homomorphism

$$\text{act} : \mathbf{M}^{\text{op}} \rightarrow \mathbf{End}(\mathbf{A}). \quad (21)$$

Group actions

For defining a group action, we must adapt the definition of a monoid action. The endomorphisms $\mathbf{End}(\mathbf{A})$ are not a group, because they also contain non-invertible maps. Recall that an invertible endomorphism is called an *automorphism*, and that $\mathbf{Aut}(\mathbf{A})$, the set of automorphisms of \mathbf{A} , comes naturally equipped with a group structure.

Definition 11.5 (Group right action operation)

A *right action operation* of a group $\mathbf{G} = \langle \mathbf{G}, \circ_{\mathbf{G}}, \text{id}_{\mathbf{G}} \rangle$ on a set \mathbf{A} is:

Constituents

1. a function

$$\text{act} : \mathbf{A} \times \mathbf{G} \rightarrow \mathbf{A}. \quad (22)$$

Conditions

1. for all $a \in \mathbf{A}$ and $x, y \in \mathbf{G}$,

$$\text{act}(\text{act}(a, x), y) = \text{act}(a, x \circ_{\mathbf{G}} y); \quad (23)$$

2. for all $a \in \mathbf{A}$,

$$\text{act}(a, \text{id}_{\mathbf{G}}) = a. \quad (24)$$

Definition 11.6 (Group action)

An *action* of a group \mathbf{G} onto a set \mathbf{A} is a group morphism

$$\text{act} : \mathbf{G} \rightarrow \mathbf{Aut}(\mathbf{A}). \quad (25)$$

Graded exercise D.13 (`MatrixMultAction`)

Let $\mathbf{A} = \mathbb{R}^n$, and let $\text{GL}(n, \mathbb{R})$ be the group of invertible $n \times n$ matrices. Let

$$\begin{aligned} \alpha : \text{GL}(n, \mathbb{R}) \times \mathbf{A} &\rightarrow \mathbf{A} \\ \langle \mathbf{M}, \mathbf{v} \rangle &\mapsto \mathbf{M}\mathbf{v} \end{aligned} \quad (26)$$

be the usual multiplication of matrices with vectors. Check that (26) defines a left group action operation of $\text{GL}(n, \mathbb{R})$ on \mathbf{A} .

Semigroup actions

Definition 11.7 (Semigroup action operation)

A *right action operation* of a semigroup $\mathbf{S} = \langle \mathbf{S}, \circ_{\mathbf{S}} \rangle$ on a set \mathbf{A} is a function

$$\text{act} : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{A} \quad (27)$$

such that, for all $a \in \mathbf{A}$ and $x, y \in \mathbf{S}$,

$$\text{act}(\text{act}(a, x), y) = \text{act}(a, x \circ_{\mathbf{S}} y). \quad (28)$$

Definition 11.8 (Semigroup action)

An *action* of a semigroup \mathbf{S} on a set \mathbf{A} is a semigroup morphism

$$\text{act} : \mathbf{S} \rightarrow \mathbf{End}(\mathbf{A}). \quad (29)$$

Actions of sets

Definition 11.9 (Set left action operation)

A *left action operation* of a set \mathbf{S} on a set \mathbf{A} is a function

$$\text{act} : \mathbf{S} \times \mathbf{A} \rightarrow \mathbf{A}. \quad (30)$$

Definition 11.10 (Set right action operation)

A *right action operation* of a set \mathbf{S} on a set \mathbf{A} is a function

$$\text{act} : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{A}. \quad (31)$$

11.2. Modules, Vector spaces

Vector spaces

Definition 11.11 (Real vector space)

A *real vector space* is

Constituents

1. a set \mathbf{V} , the elements of which are called *vectors*;
2. a binary operation $+$: $\mathbf{V} \times \mathbf{V} \rightarrow \mathbf{V}$, called *vector addition*;
3. an element $0 \in \mathbf{V}$;
4. an operation $\cdot : \mathbb{R} \times \mathbf{V} \rightarrow \mathbf{V}$, called *scalar multiplication*.

Conditions

1. $\langle \mathbf{V}, + \rangle$ is a commutative group, with neutral element 0 ;
2. Scalar multiplication is a left action operation of the ring \mathbb{R} :
 - a) $(\lambda\mu) \cdot \mathbf{x} = \lambda \cdot (\mu \cdot \mathbf{x}) \quad \forall \lambda, \mu \in \mathbb{R}, \forall \mathbf{x} \in \mathbf{U}$;
 - b) $1 \cdot \mathbf{x} = \mathbf{x} \quad \forall \mathbf{x} \in \mathbf{U}$;
 - c) $(\lambda + \mu) \cdot \mathbf{x} = (\lambda \cdot \mathbf{x}) + (\mu \cdot \mathbf{x}) \quad \forall \lambda, \mu \in \mathbb{R}, \forall \mathbf{x} \in \mathbf{V}$;
 - d) $\lambda \cdot (\mathbf{x} + \mathbf{y}) = (\lambda \cdot \mathbf{x}) + (\lambda \cdot \mathbf{y}), \quad \forall \lambda \in \mathbb{R}, \forall \mathbf{x}, \mathbf{y} \in \mathbf{V}$.

Remark 11.12. The general definition of a vector space over any field \mathbf{K} is obtained by replacing \mathbb{R} in the above definition with an arbitrary field \mathbf{K} .

Example 11.13. $\langle \mathbb{R}^n, +, \cdot \rangle$

Graded exercise D.14 (RealPolynomials)

Let \mathbf{A} denote the set of polynomials in one variable and with coefficients in \mathbb{R} . In other words, elements of \mathbf{A} are polynomials of the form

$$p(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0, \quad (32)$$

where $n \in \mathbb{N}$ and $a_n, a_{n-1}, \dots, a_1, a_0 \in \mathbb{R}$ may vary.

1. Check (prove) that \mathbf{A} forms a commutative monoid when equipped with with the usual addition operation for polynomials and the zero polynomial $p(X) = 0$ as neutral element.
2. Is the commutative monoid $\langle \mathbf{A}, +, 0 \rangle$ in fact a group?
3. Check that $\langle \mathbf{A}, +, 0 \rangle$ forms a real vector space when we equip it with the following (usual) scalar multiplication:

$$\lambda \cdot (a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0) := \lambda a_n X^n + \lambda a_{n-1} X^{n-1} + \cdots + \lambda a_1 X + \lambda a_0 \quad (33)$$

for any $\lambda \in \mathbb{R}$.

11.3. Linear group actions

Special Euclidean group

Definition 11.14 (General Euclidean group $E(n, \mathbb{R})$)

The general (real) Euclidean group of order n , written $E(n, \mathbb{R})$, is the group of $(n+1) \times (n+1)$ real matrices of the form

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \quad (34)$$

where $\mathbf{R} \in O(n, \mathbb{R})$ and $\mathbf{t} \in \mathbb{R}^n$.

Definition 11.15

Special Euclidean group $SE(n, \mathbb{R}^n)$] The special (real) Euclidean group of order n , written $SE(n, \mathbb{R}^n)$, is the group of $(n+1) \times (n+1)$ real matrices of the form

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}, \quad (35)$$

where $\mathbf{R} \in SO(n, \mathbb{R})$ and $\mathbf{t} \in \mathbb{R}^n$.

The groups $SE(2, \mathbb{R})$ and $SE(3, \mathbb{R})$ are particular important in robotics because they represent the roto-translations of the plane and 3D space, respectively.

From (34) we know we can represent one by a pair $\langle \mathbf{R}, \mathbf{t} \rangle$, with $\mathbf{R} \in SO(n, \mathbb{R})$ and $\mathbf{t} \in \mathbb{R}^n$.

If we look at how matrices compose, we get

$$\begin{bmatrix} \mathbf{R}_2 & \mathbf{t}_2 \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 & \mathbf{t}_1 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_2 \mathbf{R}_1 & \mathbf{R}_2 \mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{0} & 1 \end{bmatrix}. \quad (36)$$

The formula for composition is

$$\langle \mathbf{R}_1, \mathbf{t}_1 \rangle \circ_{SE(n, \mathbb{R}^n)} \langle \mathbf{R}_2, \mathbf{t}_2 \rangle = \langle \mathbf{R}_2 \mathbf{R}_1, \mathbf{R}_2 \mathbf{t}_1 + \mathbf{t}_2 \rangle. \quad (37)$$

The group $SE(n, \mathbb{R}^n)$ induces a transformation on the points of \mathbb{R}^n . We are going to call this an *action*.

The action is the following function:

$$\begin{aligned} \text{apply: } SE(n, \mathbb{R}^n) \times \mathbb{R}^n &\rightarrow \mathbb{R}^n, \\ \langle \langle \mathbf{R}, \mathbf{t} \rangle, \mathbf{p} \rangle &\mapsto \mathbf{R}\mathbf{p} + \mathbf{t}. \end{aligned} \quad (38)$$

Given a roto-translation and a point, the function returns the roto-translated point. We can also see this in matrix form as follows. We need to substitute for a

Table 11.1.: Matrix groups

$GL(n, \mathbb{R})$	General linear group	arbitrary linear transformations
$SL(n, \mathbb{R})$	Special linear group	invertible linear transformations
$O(n, \mathbb{R})$	Orthogonal group	preserve length of vectors
$SO(n, \mathbb{R})$	Special orthogonal group	rotations
$E(n, \mathbb{R})$	Euclidean group	preserve distances and angles
$SE(n, \mathbb{R}^n)$	Special Euclidean group	rigid motions

point $\mathbf{p} \in \mathbb{R}^n$ a homogenous point $\begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1}$.

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}\mathbf{p} + \mathbf{t} \\ 1 \end{bmatrix}. \quad (39)$$

If we apply two rototranslations, first $\mathbf{x} = \langle \mathbf{R}_x, \mathbf{t}_x \rangle$ and then $\mathbf{y} = \langle \mathbf{R}_y, \mathbf{t}_y \rangle$, we find:

$$\begin{aligned} \text{apply}(\langle \mathbf{R}_y, \mathbf{t}_y \rangle, \text{apply}(\langle \mathbf{R}_x, \mathbf{t}_x \rangle, \mathbf{p})) &= \text{apply}(\langle \mathbf{R}_y, \mathbf{t}_y \rangle, \mathbf{R}_x \mathbf{p} + \mathbf{t}_x) \\ &= \mathbf{R}_y \mathbf{R}_x \mathbf{p} + \mathbf{R}_y \mathbf{t}_x + \mathbf{t}_y. \end{aligned} \quad (40)$$

It is easy to see that it is equal to compose the two transformations in the inverse order

$$\langle \mathbf{R}_x, \mathbf{t}_x \rangle \circ_{\text{SE}(n, \mathbb{R}^n)} \langle \mathbf{R}_y, \mathbf{t}_y \rangle = \langle \mathbf{R}_y \mathbf{R}_x, \mathbf{R}_y \mathbf{t}_x + \mathbf{t}_y \rangle, \quad (41)$$

and then apply it to the object

$$\text{apply}(\langle \mathbf{R}_y \mathbf{R}_x, \mathbf{R}_y \mathbf{t}_x + \mathbf{t}_y \rangle, \mathbf{p}) = \mathbf{R}_y \mathbf{R}_x \mathbf{p} + \mathbf{R}_y \mathbf{t}_x + \mathbf{t}_y. \quad (42)$$

Thus, we have proved this property

$$\text{apply}(\mathbf{y}, \text{apply}(\mathbf{x}, \mathbf{p})) = \text{apply}(\mathbf{x} \circ \mathbf{y}, \mathbf{p}), \quad (43)$$

which is graphically reported in Fig. 2.

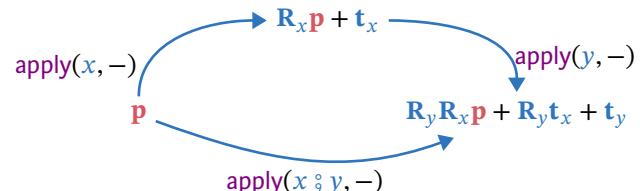


Figure 2.: Graphical representation of roto-translation action.

The notion of semigroup action generalizes this property.

11.4. Dynamical systems

Dynamical systems are ubiquitous in engineering, and there are various ways to formulate this concept mathematically. At a very basic level, there are usually the following ingredients:

1. a *state space* which comprises the possible states of the system in question
2. a *dynamics rule* which governs how states of the system may change

Typically, there is a model of *time* involved (e.g. either in discrete time-steps, or as continuous time) for describing how states change over time, or, instead of a model of time, one may use a set of *events* as the triggers for changes of state.

A common way to model dynamics is by a right action operation of a monoid $\mathbf{T} = \langle \mathbf{T}, \circ_{\mathbf{T}}, \text{id}_{\mathbf{T}} \rangle$ on a set \mathbf{X} which models the state space,

$$\text{dyn} : \mathbf{X} \times \mathbf{T} \rightarrow \mathbf{X}. \quad (44)$$

Discrete-time systems

Consider for example the monoid $\mathbf{T} = \langle \mathbb{N}, +, 0 \rangle$ as a discrete model of time. Observe that \mathbf{T} is generated by the number $1 \in \mathbb{N}$: every natural number n can be written as a n -fold sum of 1 with itself. If we model a dynamical system by a left action operation of \mathbf{T} on a state space \mathbf{X} ,

$$\text{dyn} : \mathbb{N} \times \mathbf{X} \rightarrow \mathbf{X}, \quad (45)$$

then in particular this dynamics function is compatible with the summing of natural numbers (which is the composition operation in the monoid \mathbf{T}):

$$\text{dyn}(n, \text{dyn}(m, x)) = \text{dyn}(n + m, x). \quad (46)$$

Because $\mathbf{T} = \langle \mathbb{N}, +, 0 \rangle$ is generated by the number 1, the function (45) is then completely determined by the function

$$\text{dyn}(1, -) : \mathbf{X} \rightarrow \mathbf{X}. \quad (47)$$

If we start ‘running’ the dynamical system at some point x_0 at time $t = 0$, then iteratively applying the function (47) will compute that trajectory $x_0, x_1, x_2, x_3, \dots$ of the initial point x_0 as time progresses. Often a discrete-time dynamical system is specified by defining the function (47) in the following notation,

$$x_{n+1} = \text{f}(x_n), \quad (48)$$

where $\text{f} = \text{dyn}(1, -)$ and x_n denotes $\text{dyn}(n, x_0)$, the n -th element of a trajectory starting at some x_0 .

Example 11.16. The logistic equation

$$x_{n+1} = rx_n(x_n - 1) \quad (49)$$

is of the form (48), with $\mathbf{X} = [0, 1] \subseteq \mathbb{R}$ and where $r \in \mathbb{R}$ is a parameter. Despite being only 1-dimensional and deterministic, this dynamical system can exhibit extremely complex behavior (more precisely: deterministic chaos), depending on which value for the parameter r is chosen.

Continuous-time systems

Consider the initial value problem

$$\begin{cases} \dot{x}(t) = f(x(t)) \\ x(0) = x_0 \end{cases} \quad (50)$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is a differentiable function, $x : \mathbb{R} \rightarrow \mathbb{R}^n$ is a continuously differentiable function, and $x_0 \in \mathbb{R}^n$. Under suitable assumptions on f , for each choice of x_0 , there exists a unique solution $s_{x_0} : \mathbb{R} \rightarrow \mathbb{R}^n$ of this initial value problem. Using all of these solutions, it is possible to define the following map, which is called a *flow*:

$$\begin{aligned} \varphi : \mathbb{R} \times \mathbb{R}^n &\rightarrow \mathbb{R}^n, \\ \langle t, x_0 \rangle &\mapsto s_{x_0}(t). \end{aligned} \quad (51)$$

In this case, the flow map φ will necessarily satisfy the following conditions

1. $\varphi(t+s, x) = \varphi(t, \varphi(s, x)) \quad \forall t, s \in \mathbb{R}, x \in \mathbb{R}^n,$
2. $\varphi(0, x) = x \quad \forall x \in \mathbb{R}^n,$

which means that is is a left action operation on \mathbb{R}^n by the monoid $\langle \mathbb{R}, +, 0 \rangle$.

Definition 11.17

A continuous-time dynamical system on \mathbb{R}^n is a continuously differentiable function

$$\text{dyn} : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (52)$$

which is a left action operation of the monoid $\langle \mathbb{R}, +, 0 \rangle$

Event-based systems

Machines

Linear time-invariant systems

Definition 11.18 (LTI System)

A *linear time-invariant dynamical (LTI) system*, in a state-space representation, is specified by real vector spaces $\mathbf{U} = \mathbb{R}^l$ (input space), $\mathbf{Y} = \mathbb{R}^m$ (output space), and $\mathbf{X} = \mathbb{R}^n$ (state space), along with a system of equations of the form

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (53)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \quad (54)$$

and an *initial state* $\mathbf{x}_0 \in \mathbf{X}$, where $t \in \mathbb{R}_{\geq 0}$, $\mathbf{u}(t) \in \mathbf{U}$, $\mathbf{y}(t) \in \mathbf{Y}$, $\mathbf{x}(t) \in \mathbf{X}$, and where \mathbf{A} , \mathbf{B} , \mathbf{C} , \mathbf{D} are real matrices of appropriate dimension.

We think of (53) as defining dynamics

$$\begin{aligned} \text{dyn} : \mathbf{U} \times \mathbf{X} &\rightarrow \mathbf{X}, \\ \langle u, x \rangle &\mapsto \mathbf{Ax} + \mathbf{Bu}, \end{aligned} \quad (55)$$

and (54) as defining a read-out function

$$\begin{aligned} \text{ro} : \mathbf{U} \times \mathbf{X} &\rightarrow \mathbf{Y}, \\ \langle u, x \rangle &\mapsto \mathbf{Cx} + \mathbf{Du}. \end{aligned} \quad (56)$$

Solutions to selected exercises

Solution of Exercise 22. No.

Solution of Exercise 23. We show a counterexample. Clearly we have

$$\left(\begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} \circledast \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right) \circledast \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -2 \\ -1 \end{bmatrix}. \quad (57)$$

However,

$$\begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} \circledast \left(\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \circledast \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \quad (58)$$

violating the associative law.

Solution of Exercise 24. Given $x, y, z \in \mathbb{N}$, we have:

$$\min(\min(x, y), z) = \min(x, \min(y, z)). \quad (59)$$

Solution of Exercise 25. No. Consider the condition $0 = \max(\text{inv}(x), x)$. In general, this is true only if $x = \text{inv}(x) = 0$.

Solution of Exercise 26. We have:

$$\begin{aligned} (F \circledast G)(x \circledast_S y) &= G(F(x \circledast_S y)) \\ &= G(F(x) \circledast_T F(y)) \\ &= G(F(x)) \circledast_U G(F(y)) \\ &= (F \circledast G)(x) \circledast_U (F \circledast G)(y). \end{aligned} \quad (60)$$

Solution of Exercise 27. We can show that we cannot find an inverse morphism

$$\text{ASCII}^{-1} : \text{List}\{\text{char}\} \rightarrow \text{List char}. \quad (61)$$

At first sight everything seems in order: if we can find an isomorphism to $\text{List}[0, 127]$, and we can express integers in binary, what could hold us back?

What fails here is something so simple it could go unnoticed: the hypothetical function g is not well-defined for all points of its domain. We know how to translate a binary string of length 7, 14, 21, ... back to symbols; but what would be the output of g on the string 111?

The function g is a left inverse for ASCII , in the sense that $\text{ASCII} \circledast g = \text{id}_{\text{List char}}$, but it is not a right inverse.

Solution of Exercise 28. The answer is **no** because the encoding is context dependent; I don't know if a single letter is followed by a space or another letter. For example, take the string

$$\text{I } \text{A } \text{M } \text{A } \text{X}. \quad (62)$$

We can decompose it as follows

$$\text{I } \text{A } \circledast \text{M } \circledast \text{A } \circledast \text{X}. \quad (63)$$

If Morse encoding was a morphism F then we would be able to encode the string as follows:

$$\text{morse}(\text{I } \text{A}) \circledast \text{morse}(\text{M}) \circledast \text{morse}(\text{A}) \circledast \text{morse}(\text{X}). \quad (64)$$

However, this cannot work, because in the second instance of M we would need to output a letter separator, while in the first case we don't.

Can you find a way to fix it?

For example, consider the alphabet obtained by taking the *product* of the letters and numbers with the set of spaces $\{\text{I}, \text{L}\}$:

$$((\text{A to Z}) \cup (\text{0 to 9})) \times \{\text{I}, \text{L}\}, \quad (65)$$

where we annotate if each symbol is followed by a letter or by a space.

In this representation, the string can be written as

$$\langle I, \text{I} \rangle \langle A, \text{L} \rangle \langle M, \text{I} \rangle \langle M, \text{L} \rangle \langle A, \text{L} \rangle \langle X, \text{I} \rangle. \quad (66)$$

Based on this representation we can define context-independent rules that make a morphism.

Solution of Exercise 29. We start with the first one. Consider $x \in \mathbf{G}$. We know that

$$F(\text{id}_{\mathbf{G}} \circ_{\mathbf{G}} x) = F(x). \quad (67)$$

On the other hand, we know:

$$F(\text{id}_{\mathbf{G}} \circ_{\mathbf{G}} x) = F(\text{id}_{\mathbf{G}}) \circ_{\mathbf{H}} F(x). \quad (68)$$

These two are equivalent if and only if $F(\text{id}_{\mathbf{G}}) = \text{id}_{\mathbf{H}}$.

For the second statement, consider again $x \in \mathbf{G}$. We now that

$$\begin{aligned} F(\text{inv}_{\mathbf{G}}(x) \circ_{\mathbf{G}} x) &= F(\text{id}_{\mathbf{G}}) \\ &= \text{id}_{\mathbf{H}}, \end{aligned} \quad (69)$$

and

$$F(\text{inv}_{\mathbf{G}}(x) \circ_{\mathbf{G}} x) = F(\text{inv}_{\mathbf{G}}(x)) \circ_{\mathbf{H}} F(x). \quad (70)$$

These two are equivalent if and only if $F(\text{inv}_{\mathbf{G}}(x)) = \text{inv}_{\mathbf{H}}(F(x))$.

Solution of Exercise 30. We have:

$$\begin{aligned} F(m + n) &= i^{m+n} \\ &= i^m \cdot i^n \\ &= F(m) \cdot F(n). \end{aligned} \quad (71)$$

PART E. CATEGORIES



12. Graphs	179
13. (Semi)categories	185
14. Categories and structures	201
15. Modeling with categories	211
16. Constructing categories	235
17. Culture	243

Hiking (“wandern” in german) is one of the main sport activities in Switzerland, often referred to as the “national sport”. On average, 520 million kilometres (in 130 million hours) are travelled every year by the Swiss. In Switzerland, the total hiking trail network is about 65,000 kilometres.



12. Graphs

In this chapter we give a formal description of graphs. Graphs are data structures with “points” and “arrows”: it is the first time we encounter a data structure with two kinds of elements.

12.1 Graphs	180
12.2 Graph homomorphisms	182

12.1. Graphs

A graph is a data structure with “points” and “arrows”, usually called *nodes/vertices* and *arcs/edges*.

Graphs are widely used in many engineering disciplines, to represent, formulate, and solve complex problems. For instance, we can represent the intermodal mobility network in a city as a directed graph (Fig. 1). The word “intermodal” means that one can jump from a mobility option to another. For instance, in the figure, you can spot the graphs for an autonomous vehicle mobility service, a micromobility service, the subway service, and roads on which you can walk. The vertices represent locations, and the edges represent different travel routes connecting the locations.

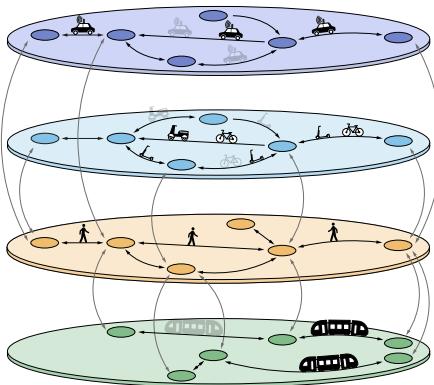


Figure 1.: Intermodal mobility network of a city.

Defining graphs

The usual definition of directed graph in engineering, which we will *not* use, is as follows:

Definition 12.1 (Directed Graph)

A *directed graph* is a pair $\mathcal{G} = \langle \mathbf{V}, \mathbf{E} \rangle$, where \mathbf{V} is a set of vertices and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is a set of edges.

In this definition, an edge is a pair of vertices $\langle x, y \rangle$ where x is the source and y is the target. One limitation of this notion of graph is that we can only have *one* edge between two vertices in either direction.

The following definition is more expressive, though a bit more abstract.

Definition 12.2 (Directed Multigraph)

A *directed multigraph* $\mathcal{G} = \langle \mathbf{V}, \mathbf{E}, \text{src}, \text{tgt} \rangle$ consists of a set of vertices \mathbf{V} , a set of edges \mathbf{E} , and two functions $\text{src}, \text{tgt} : \mathbf{E} \rightarrow \mathbf{V}$, called the *source* and *target* functions, respectively. Given $a \in \mathbf{E}$ with $\text{src}(a) = v$ and $\text{tgt}(a) = w$, we say that a is an edge (or arrow) from v to w .

Both directed graphs and undirected graphs play a prominent role in many kinds of mathematics. In this text, we work primarily with directed multigraphs and so, from now on, we drop the “directed” and the “multi”: unless indicated otherwise, the word “graph” will mean “directed multigraph”.

Paths

Definition 12.3 (Paths)

A *path* in a graph $\mathcal{G} = \langle \mathbf{V}, \mathbf{E}, \text{src}, \text{tgt} \rangle$ is:

Constituents

▷ a list of edges $[e_1, \dots, e_n]_{\mathbf{E}}$, with $n \in \mathbb{N}$.

- If $n \neq 0$, the source of a path $[e_1, \dots, e_n]_{\mathbf{E}}$ is defined as $\text{src}(e_1)$ and its target is $\text{tgt}(e_n)$.
- If $n = 0$, we speak of a “trivial path” or an “empty path” and we must additional specify an element $x \in \mathbf{V}$ which is designated as both the source and target of the path. If paths describe a journey, then trivial paths correspond to “not going anywhere”.

Conditions

- ▷ if $n \geq 2$, we require that, for any two subsequent edges e_i and e_{i+1} in $[e_1,$

$\dots, e_n]_{\text{E}}$,
 $\text{tgt}(e_i) = \text{src}(e_{i+1})$. (1)

The length of $[e_1, \dots, e_n]_{\text{E}}$ is called the *length* of the path.

12.2. Graph homomorphisms

Definition 12.4 (Graph homomorphism)

Given graphs $\mathcal{G}_1 = \langle \mathbf{V}_1, \mathbf{E}_1, \text{src}_1, \text{tgt}_1 \rangle$ and $\mathcal{G}_2 = \langle \mathbf{V}_2, \mathbf{E}_2, \text{src}_2, \text{tgt}_2 \rangle$, a *graph homomorphism* $F: \mathcal{G}_1 \rightarrow \mathcal{G}_2$ is given by two maps

$$\mathbf{F}_*: \mathbf{V}_1 \rightarrow \mathbf{V}_2, \quad (2)$$

$$\mathbf{F}_\rightarrow: \mathbf{E}_1 \rightarrow \mathbf{E}_2, \quad (3)$$

such that

$$\mathbf{F}_\rightarrow; \text{src}_2 = \text{src}_1; \mathbf{F}_*, \quad (4)$$

$$\mathbf{F}_\rightarrow; \text{tgt}_2 = \text{tgt}_1; \mathbf{F}_*, \quad (5)$$

or, in other words, that the following diagrams commute:

$$\begin{array}{ccc} \mathbf{E}_1 & \xrightarrow{\mathbf{F}_\rightarrow} & \mathbf{E}_2 \\ \text{src}_1 \downarrow & & \downarrow \text{src}_2 \\ \mathbf{V}_1 & \xrightarrow{\mathbf{F}_*} & \mathbf{V}_2 \end{array} \quad \begin{array}{ccc} \mathbf{E}_1 & \xrightarrow{\mathbf{F}_\rightarrow} & \mathbf{E}_2 \\ \text{tgt}_1 \downarrow & & \downarrow \text{tgt}_2 \\ \mathbf{V}_1 & \xrightarrow{\mathbf{F}_*} & \mathbf{V}_2 \end{array}$$

Remark 12.5. Intuitively, all this is saying is that “arrows are bound to their vertices”, meaning that if a vertex v_1 is connected to v_2 via an arrow a , the vertices resulting from the application of the maps on nodes $\mathbf{F}_*(v_1)$ and $\mathbf{F}_*(v_2)$ have to be connected via the arrow resulting from the application of the map on arrows $\mathbf{F}_\rightarrow(a)$.

Example 12.6. Consider the two graphs, \mathcal{G}_1 and \mathcal{G}_2 depicted in Fig. 2.

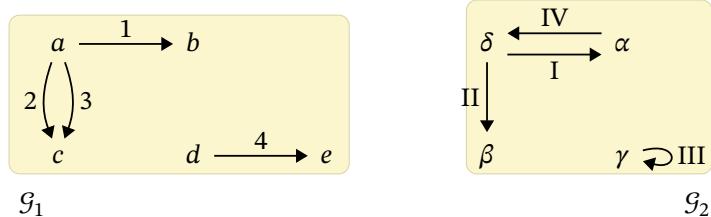


Figure 2.: Example of graphs for graph homomorphism.

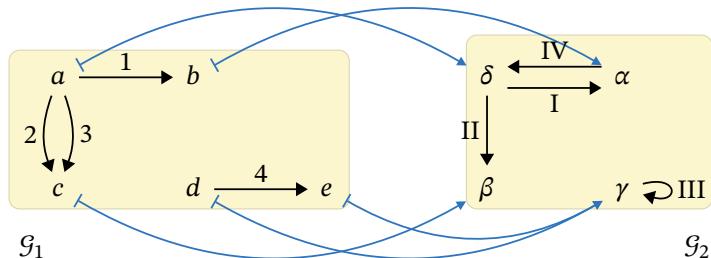


Figure 3.: \mathbf{F}_* for the presented graph homomorphism.

A possible graph homomorphism between the two is given by \mathbf{F}_* , \mathbf{F}_\rightarrow graphically defined as in Fig. 3 and Fig. 4, respectively.

Example 12.7 (Counterexample). By considering the graphs in Example 12.6, we could define \mathbf{F}_* , \mathbf{F}_\rightarrow in the same way, exception made for $\mathbf{F}_*(e) = \alpha$. Clearly, this would violate the commuting diagrams condition.

Exercise 31. Consider the two graphs depicted in Fig. 5.

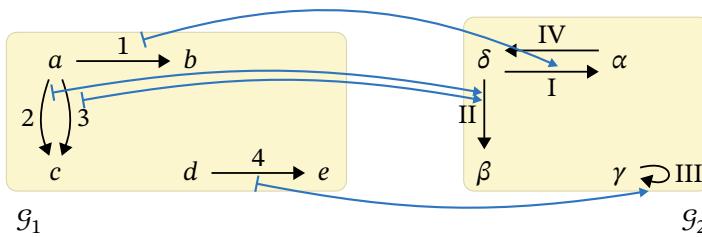


Figure 4.: F_\rightarrow for the presented graph homomorphism.

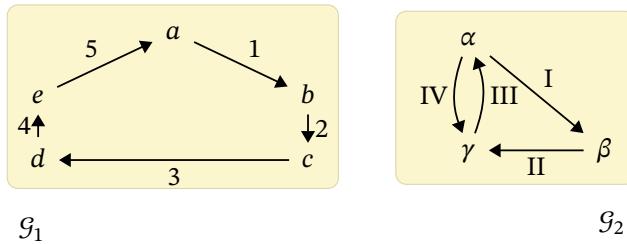


Figure 5.

Furthermore, consider the map F_\bullet depicted in Fig. 6.

Find a map F_\rightarrow such that F_\bullet, F_\rightarrow describe a graph homomorphism between \mathcal{G}_1 and \mathcal{G}_2 .

See solution on page 249.

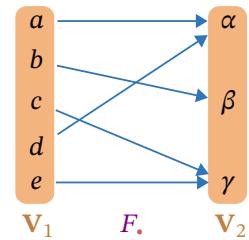


Figure 6.



13. (Semi)categories

In this chapter we look at the fundamental notion of a category, and also its cousin, the notion of semicategory. We will see that categories generalize many of mathematical structures that we have studied in this book so far.

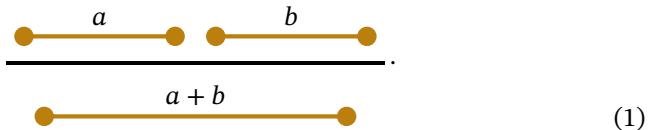
Although categories are more of a protagonist in this book, we introduce semicategories first because they are more rudimentary. The step from semicategories to categories will then be very similar to the step from semigroups to monoids.

13.1 Interfaces	186
13.2 Semicategories	188
13.3 Categories	192
13.4 Diagrams	195
13.5 Categories vs graphs	197
13.6 Categories from graphs	198

13.1. Interfaces

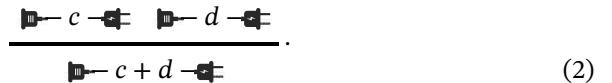
One way to understand semicategories is to see them as a generalization of semigroups. In semigroups, monoids, and groups we could take any two elements and compose them: the elements always had a “compatible” interface.

To motivate the need for interfaces, consider the ropes of Chapter 2, which had this composition rule:



Two chapters later, we can recognize that we were describing the monoid $\langle \mathbb{R}_{\geq 0}, +, 0 \rangle$. Being a monoid, all pieces of rope are compatible and can be composed.

A first step towards discussing interfaces is to think of things that have a direction. For example, consider extension cords. Let $\blacksquare - c - \blacksquare$ be an extension cord of length c . If you have an extension cord of length c and another of length d , you can plug them together to get an extension cord of length $c + d$:



In this form, this is still the same monoid.

But suppose now that, reading this book, you fall in love with Switzerland and want to visit. As you start to plan your trip, at some point you need to think about electrical adapters. Switzerland uses the connector of type N (Fig. 1). If you come from Ireland, your appliances use type G. Now when we think of extension cords, we might allow either end to have a plug type. These would be Irish and Swiss extension cords of length ℓ :



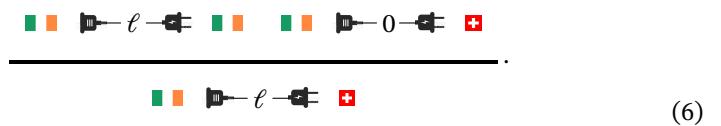
You might want a cord that has a Swiss male end and an Irish female end:



Unfortunately these devices don't exist. What you can buy are adapters, which we can think of extension cord of length zeros:



If you have an adapter, then you can attach an extension cord to it to obtain (4):



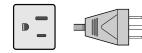
The general formula to compose cords with generic types X, Y, Z is

$$\begin{array}{c} X \quad \square \rightarrow a \rightarrow \blacksquare \quad Y \quad Y \quad \square \rightarrow b \rightarrow \blacksquare \quad Z \\ \hline X \quad \square \rightarrow a + b \rightarrow \blacksquare \quad Z \end{array} \quad (7)$$

This kind of composition of things that have an input and an output interface, like cords, can be modeled by the notions of semicategory and category.



A



B



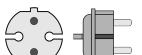
C



D



E



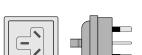
F



G



H



I



L



M



N

Figure 1: Plug/socket types used in the world

13.2. Semicategories

We begin to introduce the concept of a semicategory by listing key aspects of the electrical cords example in the previous section which each correspond to a part of the formal definition of semicategory.

Objects, morphisms, composition

Firstly, in the cords example, we have a set of types of electrical interfaces. In a semicategory, the things that play the role of interfaces are called *objects*. (In this text, we often denote generic objects by letters X, Y, Z, \dots .)

Secondly, in the electrical cords example, the cords themselves connect two interfaces and have a directionality: one end has a socket, the other end has a plug. In a semicategory, the things that play the role of cords are called *morphisms*. (This is a word we've already gotten to know, and the connection here is not accidental.) The object that denotes the type of the “socket end” of the morphism is called the *domain* or *source* of the morphism. The object that denotes the “plug end” of a morphism is called its *codomain* or *target*. So morphisms are directed: they go from their source to their target.

To visualize morphisms, we often draw arrows. For example, if X and Y are objects in some semicategory, then we draw a morphism from X to Y (call it f) like this:

$$f : X \rightarrow Y. \quad (8)$$

So here X is the source of f , and Y is its target.

Thirdly, a key feature of the cords example is that we can compose two cords and the result is again a chord. However, for this to work, the plug-end of the first chord must match the socket-end of the other. Similarly, in a semicategory we specify a way to compose two morphisms, provided that the target object of the first morphism matches the source object of the second morphism. (When this is the case, the morphisms are said to be *composable*.)

In other words, if $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ are morphisms in some semicategory (note: they are composable), then the semicategory has an operation for composing them, and the result is another morphism. Our notation for composition of f and g is $f ; g$, read “ f then g ”. (Once again: this is in contrast to the more traditional notation $g \circ f$.) Thus, we have

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{(f ; g) : X \rightarrow Z}, \quad (9)$$

which is analogous to (7) in the cords example.

Associativity

So far we have described the building blocks, or constituents, of a semicategory: objects, morphisms, and composition operations. We also want these to obey a certain condition called the *associativity law*. This condition says that if we are given a string of three composable morphisms, then it doesn't matter in which order we choose to compose them:

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z \quad h : Z \rightarrow U}{(f ; g) ; h = f ; (g ; h)}. \quad (10)$$

This is analogous to the fact, in the electrical cords example, that if we look at

three cords connected together, we cannot tell if the first two were connected together first and then the result was connected to the third, or if the connecting happened the other way around.

The definition of a semicategory

Here is the full formal definition of a semicategory. Have a read through, then we illustrate it further with examples.

Definition 13.1 (Semicategory)

A *semicategory* \mathbf{C} is specified by:

Constituents

1. Objects: A collection* $\text{Ob}_{\mathbf{C}}$ whose elements are called *objects*.
2. Morphisms: For every pair of objects X, Y in $\text{Ob}_{\mathbf{C}}$, there is a set called a “hom-set” and indicated as $\text{Hom}_{\mathbf{C}}(X; Y)$, elements of which are called *morphisms* and denoted $f : X \rightarrow Y$.
For such an f , we call X its *source* and Y its *target*.
3. Composition operations: For every three objects X, Y, Z in $\text{Ob}_{\mathbf{C}}$ there is a composition map

$$\circ_{X,Y,Z} : \text{Hom}_{\mathbf{C}}(X; Y) \times \text{Hom}_{\mathbf{C}}(Y; Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Z). \quad (11)$$

We usually just write \circ instead of $\circ_{X,Y,Z}$:

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{(f \circ g) : X \rightarrow Z}. \quad (12)$$

The morphism $f \circ g$ is called the *composition* of f and g .

Conditions

1. Associativity: it holds that

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z \quad h : Z \rightarrow U}{(f \circ g) \circ h = f \circ (g \circ h)}. \quad (13)$$

Remark 13.2. We denote composition of morphisms using the symbol “ \circ ” (pronounced “then”), as already introduced for functions in Section 3.4. This is in contrast to the more common notation for composition, namely $g \circ f$, or simply gf , which reads as “ g after f ”. As usual, f^2 denotes $f \circ f$, f^3 denotes $f \circ f \circ f$, and so on.

Remark 13.3. 1. When we want to emphasize which semicategory we are working with, we will sometimes write

$$f : X \rightarrow_{\mathbf{C}} Y \quad (14)$$

to indicate

$$f \in \text{Hom}_{\mathbf{C}}(X; Y). \quad (15)$$

2. Sometimes we will use the notation $\text{Mor}_{\mathbf{C}}$ to denote the collection of *all* morphisms in a semicategory \mathbf{C} , not just a certain hom-set.

Remark 13.4. We will often visualize objects and morphisms using diagrams where symbols or dots indicate objects and arrows indicate morphisms. For

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.



Figure 2.

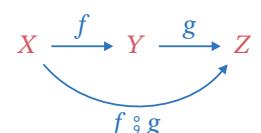


Figure 3.

instance, if X, Y, Z are objects in a semicategory \mathbf{C} , and $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ are morphisms, we draw this as in Fig. 2, for example. The composition $f ; g$ of f with g corresponds to yet another arrow, as in Fig. 3.

Example 13.5. There is a semicategory whose objects are all sets, morphisms are functions between sets, and the composition operations are the usual composition of functions.

Let us also check that composition of functions does indeed satisfy the associativity law. Suppose $f : X \rightarrow Y$, $g : Y \rightarrow Z$, and $h : Z \rightarrow U$ are three composable functions, and let $x \in X$ be an arbitrary element. On the one hand

$$((f ; g) ; h)(x) = h((f ; g)(x)) = h(g(f(x))), \quad (16)$$

while on the other hand

$$(f ; (g ; h))(x) = (g ; h)(f(x)) = h(g(f(x))). \quad (17)$$

So

$$(f ; g) ; h = f ; (g ; h) \quad (18)$$

holds.

Example 13.6. This example is an extension of Example 9.12. We will describe a semicategory \mathbf{C} with two objects, $\text{Ob}_{\mathbf{C}} = \{\mathbf{A}, \mathbf{B}\}$. Each of the objects is a set which describes possible states of a plant. Let

$$\mathbf{A} = \{\text{sprout, young, mature, old, dead}\}, \quad (19)$$

and

$$\mathbf{B} = \{\text{alive, dead}\}. \quad (20)$$

Also, let $f : \mathbf{A} \rightarrow \mathbf{A}$ be the function with

$$\begin{aligned} f(\text{sprout}) &= \text{young}, \\ f(\text{young}) &= \text{mature}, \\ f(\text{mature}) &= \text{old}, \\ f(\text{old}) &= \text{dead}, \\ f(\text{dead}) &= \text{dead}; \end{aligned} \quad (21)$$

let $g : \mathbf{B} \rightarrow \mathbf{B}$ be the function with

$$\begin{aligned} g(\text{alive}) &= \text{dead}, \\ g(\text{dead}) &= \text{dead}; \end{aligned} \quad (22)$$

and let $h : \mathbf{A} \rightarrow \mathbf{B}$ be the function with

$$\begin{aligned} h(\text{sprout}) &= \text{alive}, \\ h(\text{young}) &= \text{alive}, \\ h(\text{mature}) &= \text{alive}, \\ h(\text{old}) &= \text{alive}, \end{aligned} \quad (23)$$

For our sets of morphisms, we let

$$\begin{aligned}\text{Hom}_{\mathbf{C}}(\mathbf{A}; \mathbf{A}) &= \{\textcolor{blue}{f}, \textcolor{blue}{f}^2, \textcolor{blue}{f}^3, \textcolor{blue}{f}^4\}, \\ \text{Hom}_{\mathbf{C}}(\mathbf{B}; \mathbf{B}) &= \{\textcolor{blue}{g}\}, \\ \text{Hom}_{\mathbf{C}}(\mathbf{A}; \mathbf{B}) &= \{\textcolor{blue}{h}, \textcolor{blue}{f} \circ \textcolor{blue}{h}, \textcolor{blue}{f}^2 \circ \textcolor{blue}{h}, \textcolor{blue}{f}^3 \circ \textcolor{blue}{h}, \textcolor{blue}{f}^4 \circ \textcolor{blue}{h}\}, \\ \text{Hom}_{\mathbf{C}}(\mathbf{B}; \mathbf{A}) &= \emptyset;\end{aligned}\tag{24}$$

and for the composition operations, we define these to be the usual composition of functions, which we know obeys the associative law. Thus, we have a semicategory.

13.3. Categories

Now we come to one of the central protagonists of this book: the concept of a category. Categories are like semicategories, but with one more ingredient added: identity morphisms. A good analogy is that categories are to semicategories as monoids are to semigroups. A monoid is a semigroup that additionally has an identity element, and similarly a category is a semicategory that additionally has identity morphisms.

Identity morphisms

One might say that identity morphisms are morphisms that “do nothing”: they do not have any effect when we compose with them. This is analogous to how the identity element of a monoid “does nothing” when we multiply it with other elements of the monoid.

Definition 13.7 (Identity morphisms)

Let \mathbf{C} be a semicategory. An *identity morphism*, or just *identity*, for an object X of \mathbf{C} is a morphism

$$\text{id}_X : X \rightarrow X \quad (25)$$

in \mathbf{C} that acts neutrally with respect to composition with any morphism in the category with which it is composable:

$$\frac{f : W \rightarrow X}{f \circ \text{id}_X = f}, \quad (26)$$

and

$$\frac{g : X \rightarrow Y}{\text{id}_X \circ g = g}. \quad (27)$$

Remark 13.8. If an identity morphism id_X for an object X exists, then it is unique. To see this, observe that $\text{Hom}_{\mathbf{C}}(X; X)$ is a semigroup, and id_X is a neutral element for this semigroup, making $\text{Hom}_{\mathbf{C}}(X; X)$ a monoid. We have seen earlier that neutral elements for semigroups are necessarily unique.

Categories

Definition 13.9 (Category)

A *category* \mathbf{C} is a semicategory in which there is an identity morphism for every object.

Example 13.10. The semicategory of sets and functions described above in Example 13.5 is in fact a category. Given a set X , the identity morphism for this set is the identity function

$$\begin{aligned} \text{id}_X : X &\rightarrow X, \\ x &\mapsto x. \end{aligned} \quad (28)$$

Let us check that the conditions (26) and (27) are satisfied. Given a function $f : W \rightarrow X$, the function composition $f \circ \text{id}_X$ is the same function as just f on its own:

$$(f \circ \text{id}_X)(x) = \text{id}_X(f(x)) = f(x). \quad (29)$$

Given a function $g : X \rightarrow Y$, we can show similarly that $\text{id}_X \circ g = g$.

Definition 13.11 (Category of sets)

The category **Set** of sets is defined by:

1. *Objects*: all sets.
2. *Morphisms*: given sets X and Y , the hom-set $\text{Hom}_{\text{Set}}(X; Y)$ is the set of all functions from X to Y .
3. *Composition*: the usual composition of functions.
4. *Identity morphisms*: given a set X , its identity morphism id_X is the identity function $X \rightarrow X$, $\text{id}_X(x) = x$.

Graded exercise E.1 (LinearMaps)

Morphisms between real vector spaces are called *linear maps*. Given real vector spaces $U = \langle \mathbf{U}, +_U, 0_U, \cdot_U \rangle$ and $V = \langle \mathbf{V}, +_V, 0_V, \cdot_V \rangle$, a linear map $f : U \rightarrow V$ is a function

$$f : U \rightarrow V \quad (30)$$

which satisfies the following conditions:

1. $f(x +_U y) = f(x) +_V f(y) \quad \forall x, y \in U$;
2. $f(\lambda \cdot_U x) = \lambda \cdot_V f(x) \quad \forall x \in U, \forall \lambda \in \mathbb{R}$.

Your task in this exercise is to prove that the composition of linear maps is again a linear map. Concretely, suppose that you are given linear maps $f : U \rightarrow V$ and $g : V \rightarrow W$ and check that the composition of functions $f \circ g : U \rightarrow W$ is again a linear map.

Graded exercise E.2 (CategoryRealVect)

Show that real vector spaces and linear maps between them form a category. This means:

1. State what are the objects, the morphisms, and the composition operations (and check that the latter are well-defined).
2. Check that the associative law holds.
3. State what the identity morphisms are, and prove that they are neutral for composition.

Graded exercise E.3 (PointedEuclideanSpaces)

In this exercise we will define a category **Euc_{*}** of “pointed Euclidean spaces” and your task is to check that it is in fact a category.

The objects of **Euc_{*}** are pairs $\langle \mathbb{R}^n, x \rangle$ where $x \in \mathbb{R}^n$ (and $n \in \mathbb{N}$ varies).

A morphism $f : \langle \mathbb{R}^n, x \rangle \rightarrow \langle \mathbb{R}^m, y \rangle$ is a smooth (infinitely differentiable) function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that $f(x) = y$.

Composition is given by composition of smooth functions, and identity morphisms are identity functions.

Graded exercise E.4 (Endofunctions)

In this exercise we will define a category **EndSet** of “endofunctions” and it is your task to check that it really is a category.

The objects of **EndSet** are pairs $\langle A, \varphi \rangle$ where A is a set and $\varphi : A \rightarrow A$ is a function.

A morphism $f : \langle A, \varphi \rangle \rightarrow \langle B, \psi \rangle$ is a function $f : A \rightarrow \text{Set} B$ with the property that $f \circ \psi = \varphi \circ f$.

Composition is given by composition of functions, and identity morphisms are identity functions.

Graded exercise E.5 (HwkGSet)

Consider the definition below of the category **GSet**. Your task: check/justify that this does indeed define a category.

Definition 13.12 (GC)

An object of **GSet** is a tuple

$$\langle Q, A, R \rangle, \quad (31)$$

where Q and A are sets, and $R : Q \rightarrow \text{Rel } A$ is a relation.

A morphism $\mathbf{r} : \langle Q_1, A_1, R_1 \rangle \rightarrow_{\text{GSet}} \langle Q_2, A_2, R_2 \rangle$ is a pair of maps

$$\mathbf{r} = \langle r_b, r^\# \rangle, \quad (32)$$

$$r_b : Q_1 \xleftarrow{\text{Set}} Q_2, \quad (33)$$

$$r^\# : A_1 \xrightarrow{\text{Set}} A_2, \quad (34)$$

that satisfy the property

$$\forall q_2 : Q_2 \quad \forall a_1 : A_1 \quad r_b(q_2)R_1 a_1 \Rightarrow q_2 R_2 r^\#(a_1). \quad (35)$$

Morphism composition is defined component-wise

$$(\mathbf{r} \circ \mathbf{s})_b = s_b \circ r_b, \quad (36)$$

$$(\mathbf{r} \circ \mathbf{s})^\# = r^\# \circ s^\#. \quad (37)$$

The identity at $\langle Q, A, R \rangle$ is $\text{id}_{\langle Q, A, R \rangle} = \langle \text{id}_Q, \text{id}_A \rangle$.

Isomorphisms

What are identity morphisms good for? One thing we can do with them is define, for any category, the important notion of isomorphism. This concept describes a way of saying when two objects are “the same”, even if they are not equal.

Definition 13.13

Let **C** be a category. A morphism $f : X \rightarrow Y$ in **C** is an *isomorphism* if there exists a morphism $g : Y \rightarrow X$ in **C** such that

$$f \circ g = \text{id}_X \quad (38)$$

and

$$g \circ f = \text{id}_Y. \quad (39)$$

Remark 13.14. Note that the above definition coincides, for the category **Set** of sets and functions, with Def. 3.21. We saw in Exercise 7 that an isomorphism in the category **Set** is the same thing as a bijective function.

13.4. Diagrams

When working with (semi)categories, it is typical to use diagrams that look like directed graphs, with nodes representing objects and directed arrows representing morphisms. Diagrams are usually used as a tool to speak and think about specific situations, where one is focusing on certain objects and morphisms of a given (semi)category. We typically don't draw every single object and morphism in the (semi)category, we just draw the ones that we want to refer to.

For example, we might draw a diagram as in Fig. 4 because we are considering the morphisms f , g , and h . The diagram encodes their sources and targets, and is suggestive of how they may be composed. Composition corresponds to following paths in the diagram. Based on Fig. 4, we could build all the morphisms depicted in Fig. 5. By the associative law for semicategories, however, we know that $(f \circ g) \circ h = f \circ (g \circ h)$; this morphism corresponds to the path along f , g , and h .

Commutative diagrams

Often we will be interested in knowing whether two given morphisms in a hom-set are *equal* or not. For instance, in Fig. 5 we have $(f \circ g) \circ h = f \circ (g \circ h)$. Or consider for example the situation in Fig. 6. It could be that the morphisms $f \circ g$ and h are two distinct elements of $\text{Hom}_C(X; Z)$, or it could be that they are in fact equal, $f \circ g = h$. When the latter is the case, we encode this information compactly by drawing just the diagram in Fig. 7 and saying that it is *commutative*. This is a shorthand way of saying that both possible paths from X to Z in the diagram (namely via h or via f then g) give rise to the same morphism.

Definition 13.15 (Commutative diagram)

A diagram in a (semi)category is commutative if, for any two objects X and Y in the diagram, all morphisms that arise from following paths in the diagram from X to Y are in fact equal.

As a further example, if we say that the diagram in Fig. 8 is commutative, then this means in particular that the morphisms $f \circ g$, $h \circ i$, and j are all equal. By Def. 13.15 it also means for instance that all paths between Y and U give rise to the same morphism, but since there is only one path, namely via g , this doesn't give us any new information.

Remark 13.16. When drawing a diagram in a category (as opposed to just a semicategory), we usually do not draw the identity morphisms. By the definition of a category, we know the identity morphisms are there, and because they act neutrally for composition, they do not alter the computation of morphisms via following paths in a diagram.

Sometimes, however, it is more clear or simply needed to draw identity morphisms in a diagram. For instance, saying that the diagram in Fig. 9 is commutative is a way of saying that $f \circ g = \text{id}_X$.

Graded exercise E.6 ([InventingCommDiagrams](#))

Come up with three different examples of a commutative diagram in the category of sets and functions. In each example, be sure to define clearly all of the functions involved.

Graphical calculi

Later in the book we will get to know another kind of diagram: *string diagrams*. These are a different visual tool for reasoning “diagrammatically” about situations



Figure 4.

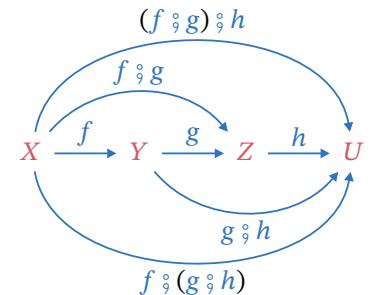


Figure 5.

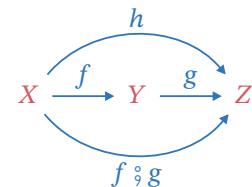


Figure 6.

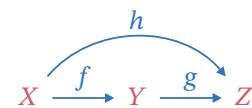


Figure 7.

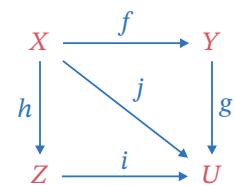


Figure 8.

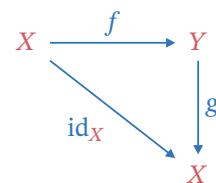


Figure 9.

in category theory. In fact, the string diagrams we will introduce are just one example in a whole zoo of visual tools, “graphical calculi”, that combine visual intuition with formal rigor. For now though, the word “diagram” will refer to the kind discussed above.

13.5. Semicategories vs directed graphs

One might now be led to ask: what is actually the difference between a semicategory and a directed graph?

With directed graphs we have nodes and directed edges, and with semicategories we have objects and morphisms. These are the same ingredients, apart from name differences.

A further essential ingredient in the definition of a semicategory is the composition operation: for any two morphisms where the target of one is the source of the other, we can compose them to obtain a further morphism.

One might thus say: a semicategory corresponds to a special kind of directed graph, where for any two adjacent directed edges there must exist a third edge corresponding to the “composite” of those edges. This is technically a true statement, however it does not emphasize the key point that, in category theory, we are often interested in comparing composite morphisms which might turn out to be equal, or not.

In Section 24.6, we will spell out an elegant and formally detailed way of thinking about the relationship between directed graphs and (semi)categories.

13.6. Generating categories from graphs

We can turn any graph into a (semi)category.

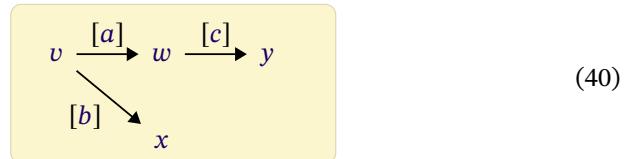
Definition 13.17 (Semicategory generated by a graph)

Let $\mathcal{G} = \langle \mathbf{V}, \mathbf{E}, \text{src}, \text{tgt} \rangle$ be a graph. The *free semicategory on \mathcal{G}* , denoted $\mathbf{SC}(\mathcal{G})$, has as objects the vertices \mathbf{V} of \mathcal{G} , and given vertices $v \in \mathbf{V}$ and $w \in \mathbf{V}$, morphisms $\text{Hom}_{\mathbf{SC}(\mathcal{G})}(v; w)$ are the non-trivial paths from v to w . The composition of morphisms is given by concatenation of paths.

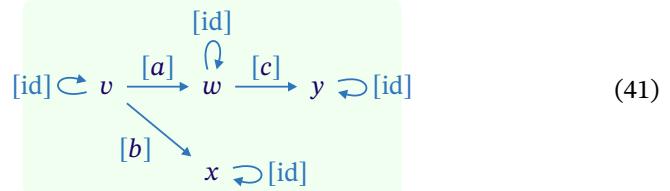
Definition 13.18 (Category generated by a graph)

Given a graph $\mathcal{G} = \langle \mathbf{V}, \mathbf{E}, \text{src}, \text{tgt} \rangle$, the *free category on \mathcal{G}* , denoted $\mathbf{C}(\mathcal{G})$, is defined analogously to $\mathbf{SC}(\mathcal{G})$ but with the modification that $\text{Hom}_{\mathbf{C}(\mathcal{G})}(v; w)$ is equal to *all* paths from v to w . Identity morphisms are the trivial paths.

For instance, consider the graph



The free category on this graph is given by

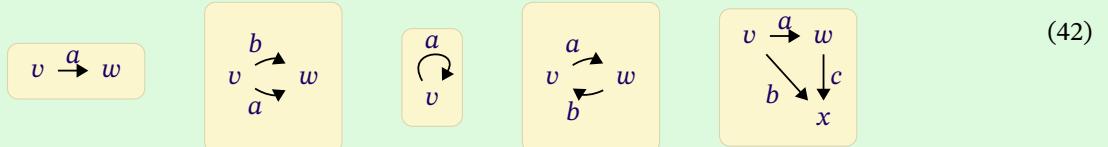


and has 8 morphisms (each vertex/object has identity morphisms, a, b, c give rise to three morphisms, and the composition of a, c gives rise to a morphism, omitted from the drawing).

Does Def. 13.18 define a category? We can check it ourselves. For it to define a category, unitality and associativity need to be satisfied. Given our definition of path, this is easy. The concatenation of paths is just list concatenation (which we already proved to be associative). Furthermore, a trivial path can be expressed via an empty list, which acts as an identity when composed to any other path.

Graded exercise E.7 (`HowManyMorphisms`)

Consider the following five graphs. For each graph \mathcal{G} , how many morphisms in total are there in the associated category $\mathbf{SC}(\mathcal{G})$?





14. Categories and structures

In this chapter we will take a tour of various categories whose objects are sets equipped with additional structures, and whose morphisms are maps between those sets that preserve the given structures.

14.1 Categories of sets and functions	202
14.2 Categories of relations	203
14.3 Categories of semigroups, monoids, groups	204
14.4 Categories from linear algebra	205
14.5 Categories of posets	206
14.6 Sets with data	207
14.7 Categories of graphs	208
14.8 Preorders as categories	209
14.9 Monoids as categories	210

14.1. Categories of sets and functions

We have already introduced in Example 13.10 the category of all sets and functions (it is a very large category). A close relative of this category is the category **FinSet**, where we only consider *finite* sets as objects, but otherwise, everything is the same as in the category **Set**.

Other categories of sets and functions can be obtained by restricting what type of functions we consider. For example, there is a category **InjSet** where the objects are all sets and where morphisms are injective functions. Similarly, there is a category **Surj** of surjective functions, and also a category **Bij** of bijective functions.

Exercise32. Spell out a definition of the category **InjSet** of injective functions, and check that it is indeed a category. In particular:

1. Specify what the composition operations are and check if the composition of two composable injective functions is again injective;
2. Specify what the identity morphisms are and check that they are indeed injective functions;
3. Argue why the associativity condition is satisfied.

See solution on page 249.

14.2. Categories of relations

Recall that a (binary) relation from a set \mathbf{A} to a \mathbf{B} is a subset $R \subseteq \mathbf{A} \times \mathbf{B}$. We have already seen that relations can be composed, so it is natural now to think of a relation $R \subseteq \mathbf{A} \times \mathbf{B}$ as a *morphism* from \mathbf{A} to \mathbf{B} .

Definition 14.1 (Category \mathbf{Rel})

The category \mathbf{Rel} of relations is defined by:

1. *Objects*: all sets.
2. *Morphisms*: for sets X, Y , $\text{Hom}_{\mathbf{Rel}}(X; Y)$ is the set of all relations $R \subseteq X \times Y$.
3. *Composition*: for relations $R : X \rightarrow Y$ and $S : Y \rightarrow Z$, their composition is

$$R ; S := \{\langle x, z \rangle \in X \times Z \mid \exists y \in Y : (x R y) \wedge (y S z)\}. \quad (1)$$

4. *Identity morphisms*: for a set X , its identity morphism is

$$\text{id}_X := \{\langle x, y \rangle \in X \times X \mid x = y\}. \quad (2)$$

Graded exercise E.8 (`IsosInRel`)

Prove that isomorphisms in the category \mathbf{Rel} are precisely those relations which correspond to bijective functions.

14.3. Categories of semigroups, monoids, groups

Definition 14.2 (Category of semigroups)

The category **SGrp** of semigroups is:

1. *Objects*: all semigroups.
2. *Morphisms*: for semigroups X and Y , the hom-set $\text{Hom}_{\text{SGrp}}(X; Y)$ is the set of all semigroup morphisms from X to Y .
3. *Composition*: composition of semigroup morphisms (composition of the underlying functions).
4. *Identity morphisms*: for a semigroup $X = \langle S, \circ_X \rangle$, its identity morphism id_X is given by the identity function $S \rightarrow S$.

Graded exercise E.9 (CategorySemigroups)

Check explicitly that the above definition does indeed define a category. Is the composition of composable morphisms again a morphism? Does the associative law hold? Are the conditions for identity morphisms satisfied?

Remark 14.3. An isomorphism (in the sense of category theory) in the category **SGrp** of semigroups is the same thing as an isomorphism of semigroups (in the sense of algebra).

Definition 14.4 (Category of monoids)

The category **Mon** of monoids is:

1. *Objects*: all monoids.
2. *Morphisms*: for monoids X and Y , the hom-set $\text{Hom}_{\text{Mon}}(X; Y)$ is the set of all monoid morphisms from X to Y .
3. *Composition*: composition of monoid morphisms.
4. *Identity morphisms*: for a monoid $X = \langle M, \circ, \text{id} \rangle$, its identity morphism id_X is given by the identity function $M \rightarrow M$.

Definition 14.5 (Category of groups)

The category **Grp** of groups is:

1. *Objects*: all groups.
2. *Morphisms*: for groups X, Y , the hom-set $\text{Hom}_{\text{Grp}}(X; Y)$ is the set of all group morphisms from X to Y .
3. *Composition*: composition of group morphisms.
4. *Identity morphisms*: for a group X , its identity morphism id_X is given by the identity function $G \rightarrow G$.

14.4. Categories from linear algebra

Definition 14.6 (Category of real matrices)

The category $\mathbf{Mat}_{\mathbb{R}}$ of real matrices is:

1. *Objects*: natural numbers \mathbb{N} .
2. *Morphisms*: for any $m, n \in \mathbb{N}$, $\text{Hom}_{\mathbf{Mat}_{\mathbb{R}}}(m; n)$ is the set of $n \times m$ real matrices.
3. *Composition*: matrix multiplication.
4. *Identity morphisms*: identity matrices.

Definition 14.7 (Category of real vector spaces)

The category $\mathbf{Vect}_{\mathbb{R}}$ of real vector spaces is:

1. *Objects*: all real vector spaces.
2. *Morphisms*: $\text{Hom}_{\mathbf{Vect}_{\mathbb{R}}}(X; Y)$ is the set of real linear maps $X \rightarrow Y$.
3. *Composition*: the usual composition of linear maps.
4. *Identity morphisms*: for any real vector space $X = \langle \mathbb{X}, +, \cdot \rangle$, the identity morphism id_X is given by the identity function $X \rightarrow X$.

14.5. Categories of posets

Definition 14.8 (Category \mathbf{Pos})

The category \mathbf{Pos} is:

1. *Objects*: all posets.
2. *Morphisms*: for posets $X = \langle P, \leq_X \rangle$ and $Y = \langle P, \leq_Y \rangle$, $\text{Hom}_{\mathbf{Pos}}(X; Y)$ is the set of all monotone maps from X to Y .
3. *Composition*: composition of monotone maps.
4. *Identity morphisms*: for a poset $X = \langle P, \leq \rangle$, its identity morphism id_X is given by the identity function $P \rightarrow P$.

Occasionally we will write $f : X \rightarrow_{\mathbf{Pos}} Y$ to emphasize that a monotone map between posets is a morphism in \mathbf{Pos} .

14.6. Sets with data

There are various simple constructions where we can build categories whose objects are not just sets, but sets together with some extra data. Morphisms are then functions which are compatible with the extra data. Below we give a few examples and we encourage the reader to imagine further variations.

Definition 14.9 (Pointed sets)

The category **Set_{*}** of pointed sets is:

1. *Objects*: pairs $\langle \mathbf{A}, x \rangle$ where \mathbf{A} is a set and $x \in \mathbf{A}$ is an element of \mathbf{A} .
2. *Morphisms*: a morphism $f : \langle \mathbf{A}, x \rangle \rightarrow \text{Set}_* \langle \mathbf{B}, y \rangle$ is a function $f : \mathbf{A} \rightarrow \text{Set} \mathbf{B}$ such that $f(x) = y$.
3. *Composition*: the usual composition of functions.
4. *Identity morphisms*: identity functions.

Exercise 33. Prove that Def. 14.9 really is a category.

See solution on page 249.

Definition 14.10 (Endofunctions)

The category **EndSet** of endofunctions is:

1. *Objects*: pairs $\langle \mathbf{A}, \varphi \rangle$ where \mathbf{A} is a set and $\varphi : \mathbf{A} \rightarrow \mathbf{A}$ is a function.
2. *Morphisms*: a morphism $f : \langle \mathbf{A}, \varphi \rangle \rightarrow \text{EndSet} \langle \mathbf{B}, \psi \rangle$ is a function $f : \mathbf{A} \rightarrow \text{Set} \mathbf{B}$ with the property that $f \circ \psi = \varphi \circ f$.
3. *Composition*: the usual composition of functions.
4. *Identity morphisms*: identity functions.

Exercise 34. Prove that Def. 14.10 is indeed a category.

See solution on page 249.

Definition 14.11 (Equivalence relations)

The category **EquivRel** of equivalence relations is:

1. *Objects*: pairs $\langle \mathbf{A}, \sim_{\mathbf{A}} \rangle$ where \mathbf{A} is a set and $\sim_{\mathbf{A}} : \mathbf{A} \rightarrow \text{Rel} \mathbf{A}$ is an equivalence relation.
2. *Morphisms*: a morphism $f : \langle \mathbf{A}, \sim_{\mathbf{A}} \rangle \rightarrow \text{EquivRel} \langle \mathbf{B}, \sim_{\mathbf{B}} \rangle$ is a function $f : \mathbf{A} \rightarrow \text{Set} \mathbf{B}$ such that

$$x \sim_{\mathbf{A}} y \implies f(x) \sim_{\mathbf{B}} f(y). \quad (3)$$

3. *Composition*: the usual composition of functions.
4. *Identity morphisms*: identity functions.

Exercise 35. We can visualize an equivalence relation on a set \mathbf{A} as a partition of \mathbf{A} . Can you visualize the condition (3) in terms of sets and partitions of them?

See solution on page 249.

Remark 14.12. The above example with equivalence relations is very similar to the category of posets and monotone maps; we are simply considering equivalence relations instead of relations which are partial orders. The category of posets, and as well as most of our other examples of categories of algebraic structures (semigroups, monoids, groups, etc.), can all be thought of as categories built from “sets with extra data”.

14.7. Categories of graphs

Before introducing the category of graphs **Grph**, we show how we can compose graph homomorphisms. Given graph homomorphisms $F : \mathcal{G}_1 \rightarrow \mathcal{G}_2$ and $G : \mathcal{G}_2 \rightarrow \mathcal{G}_3$, their composition $F ; G : \mathcal{G}_1 \rightarrow \mathcal{G}_3$ acts on vertices with $F ; G$ and on edges with $F_{\rightarrow} ; G_{\rightarrow}$.

Lemma 14.13. The composition of graph homomorphisms is a graph homomorphism.

Proof. Consider graphs $\mathcal{G}_1 = \langle \mathbf{V}_1, \mathbf{E}_1, \text{src}_1, \text{tgt}_1 \rangle$, $\mathcal{G}_2 = \langle \mathbf{V}_2, \mathbf{E}_2, \text{src}_2, \text{tgt}_2 \rangle$, and $\mathcal{G}_3 = \langle \mathbf{V}_3, \mathbf{E}_3, \text{src}_3, \text{tgt}_3 \rangle$ and graph homomorphisms $F : \mathcal{G}_1 \rightarrow \mathcal{G}_2$ and $G : \mathcal{G}_2 \rightarrow \mathcal{G}_3$. The composition of the graph homomorphisms is $F ; G : \mathcal{G}_1 \rightarrow \mathcal{G}_3$, and acts on vertices with $F ; G$ and on edges with $F_{\rightarrow} ; G_{\rightarrow}$. Then, $F ; G$ is a graph homomorphism, since we have:

$$\begin{aligned} (F_{\rightarrow} ; G_{\rightarrow}) ; \text{src}_3 &= F_{\rightarrow} ; G_{\rightarrow} ; \text{src}_3 && \text{associativity in } \mathbf{Set} \\ &= F_{\rightarrow} ; \text{src}_2 ; G_{\rightarrow} && G \text{ is a graph homom.} \\ &= \text{src}_1 ; F_{\rightarrow} ; G_{\rightarrow} && F \text{ is a graph homom.} \\ &= \text{src}_1 ; (F_{\rightarrow} ; G_{\rightarrow}) && \text{associativity in } \mathbf{Set}, \end{aligned} \tag{4}$$

and

$$\begin{aligned} (F_{\rightarrow} ; G_{\rightarrow}) ; \text{tgt}_3 &= F_{\rightarrow} ; G_{\rightarrow} ; \text{tgt}_3 && \text{associativity in } \mathbf{Set} \\ &= F_{\rightarrow} ; \text{tgt}_2 ; G_{\rightarrow} && G \text{ is a graph homom.} \\ &= \text{tgt}_1 ; F_{\rightarrow} ; G_{\rightarrow} && F \text{ is a graph homom.} \\ &= \text{tgt}_1 ; (G_{\rightarrow} ; G_{\rightarrow}) && \text{associativity in } \mathbf{Set}. \end{aligned} \tag{5}$$

These are precisely the conditions for a graph homomorphism $\mathcal{G}_1 \rightarrow \mathcal{G}_3$ \square

Definition 14.14 (Category **Grph**)

The category **Grph** is defined by:

1. *Objects:* all graphs.
2. *Morphisms:* for graphs X and Y , $\text{Hom}_{\mathbf{Grph}}(X; Y)$ is the set of graph homomorphisms (Def. 12.4) from X to Y .
3. *Composition:* composition of graph homomorphisms.
4. *Identity morphisms:* for a graph $X = \langle \mathbf{V}, \mathbf{E}, \text{src}, \text{tgt} \rangle$, its identity morphism id_X is the graph homomorphism F with $F_{\bullet} = \text{id}_{\mathbf{V}}$ and $F_{\rightarrow} = \text{id}_{\mathbf{E}}$.

Lemma 14.15. **Grph** is indeed a category.

Proof. First, from Lemma 14.13 we know that the composition of graph homomorphisms is a graph homomorphism. We check unitality and associativity. We start with unitality. Consider graphs $\mathcal{G}_1, \mathcal{G}_2$ and a graph homomorphism $F : \mathcal{G}_1 \rightarrow \mathcal{G}_2$. For $\text{id}_{\mathcal{G}_1} ; F$ we have, for every $a \in \mathbf{E}_1$ that $\text{id} ; F_{\bullet} = F_{\bullet}$ and $\text{id} ; F_{\rightarrow} = F_{\rightarrow}$. Similarly, for $F ; \text{id}_{\mathcal{G}_2}$ we have, for every $a \in \mathbf{E}_2$ that $F_{\bullet} ; \text{id} = F_{\bullet}$ and $F_{\rightarrow} ; \text{id} = F_{\rightarrow}$. (In short, unitality follows from the unitality in **Set**). Similarly, associativity follows from associativity in **Set**. \square

14.8. Preorders as categories

Definition 14.16 (Categorification of a preorder)

Given a preorder $P = \langle P, \leq_P \rangle$, its *categorification* $C(P)$ is a category with

1. *Objects*: the elements of P ;
2. *Morphisms*: given $X, Y \in P$, we define the homset $\text{Hom}_{C(P)}(X; Y)$ to be

$$\text{Hom}_{C(P)}(X; Y) = \begin{cases} \mathbf{1} & \text{if } X \leq_P Y \\ \emptyset & \text{else.} \end{cases} \quad (6)$$

3. *Composition*: should be given by functions of the type

$$\text{Hom}_{C(P)}(X; Y) \times \text{Hom}_{C(P)}(Y; Z) \rightarrow \text{Hom}_{C(P)}(X; Z). \quad (7)$$

When either of the factors in the source set are the empty set, then there is a unique function of the desired type. When both factors are equal to the set $\mathbf{1}$, then thanks to the *transitivity* of the preorder \leq_P the target set $\text{Hom}_{C(P)}(X; Z)$ must also be the set $\mathbf{1}$, and there is a unique function of the type $\mathbf{1} \times \mathbf{1} \rightarrow \mathbf{1}$.

4. *Identities*: given any $X \in \text{Ob}_{C(P)}$, we always have $X \leq_P X$, by the *reflexivity* of the preorder \leq_P . Hence $\text{Hom}_{C(P)}(X; X) = \mathbf{1}$ always. We define the single element of $\text{Hom}_{C(P)}(X; X) = \mathbf{1}$ to be the *identity morphism* of X .

Remark 14.17. A *thin* category is one in which there is at most one morphism in any homset. Categorifications of preorders are examples of thin categories. Conversely, every thin category can be interpreted as defining a preorder.

Remark 14.18. If we consider a preorder which is actually a poset, then its categorification is an example of a *skeletal* category. These are categories where, if any two objects are isomorphic, then they are necessarily equal.

Example 14.19. We revisit Def. 5.12, in which we had a poset P on $\text{Pow}\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ with order given by inclusion (Fig. 1). Its categorification $C(P)$ is a category, with $\text{Ob}_{C(P)} = \text{Pow}(\{\mathbf{a}, \mathbf{b}, \mathbf{c}\})$, and morphisms given by the inclusions. Note that we omit self-arrows for the identity morphisms, taking these to be tacitly implied. Composition is given by the transitivity law of posets. For instance, since $\{\mathbf{a}\} \subseteq \{\mathbf{a}, \mathbf{b}\}$ and $\{\mathbf{a}, \mathbf{b}\} \subseteq \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, we can say that $\{\mathbf{a}\} \subseteq \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$.

$\text{Pow}\{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$

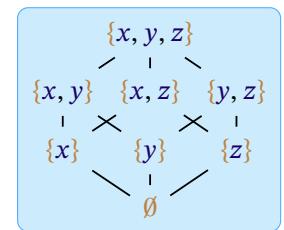


Figure 1.: Power set $\text{Pow}\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ as a poset.

14.9. Monoids as categories

Definition 14.20 (Categorification of a monoid)

Given a monoid $\mathbf{M} = \langle \mathbf{M}, \circ, \text{id} \rangle$, its *categorification* $\mathbf{C}(\mathbf{M})$ is the category with

1. *Objects*: a single object, which we denote by \star (this is arbitrarily chosen);
2. *Morphisms*: $\text{Hom}_{\mathbf{C}(\mathbf{M})}(\star; \star) = \mathbf{M}$;
3. *Composition*: is defined by the composition operation \circ of the monoid;
4. *Identity morphism*: is defined by the neutral element id of the monoid.



15. Modeling with categories

In this chapter we provide some examples of modeling real-world phenomena using categories.

15.1 Mobility	212
15.2 Trekking in the Swiss Mountains	214
15.3 Currency categories	216
15.4 Resources dependencies	219
15.5 DP as a category	223
15.6 Procedures	230
15.7 Software dependencies	233

Fondue is a Swiss dish consisting of melted cheese, served in a *caquelon* (communal pot) over a *réchaud* (portable stove). It is best enjoyed with boiled potatoes, bread, and pickled vegetables.

15.1. Mobility

For a specific mode of transportation, say a car, we can define a graph

$$\mathcal{G}_c = \langle \mathbf{V}_c, \mathbf{E}_c, \text{src}_c, \text{tgt}_c \rangle, \quad (1)$$

where \mathbf{V}_c represents geographical locations which the car can reach and \mathbf{E}_c represents the paths it can take, such as roads. Similarly, we consider a graph $\mathcal{G}_s = \langle \mathbf{V}_s, \mathbf{E}_s, \text{src}_s, \text{tgt}_s \rangle$, representing the subway system of a city, with stations \mathbf{V}_s and subway lines going through paths \mathbf{E}_s , and a graph $\mathcal{G}_b = \langle \mathbf{V}_b, \mathbf{E}_b, \text{src}_b, \text{tgt}_b \rangle$, representing onboarding and off boarding at airports. In the following, we want to express intermodality: the phenomenon that someone might travel to a certain intermediate location in a car and then take the subway to reach their final destination. By considering the graph $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \text{src}, \text{tgt})$ with $\mathbf{V} = \mathbf{V}_c \cup \mathbf{V}_s \cup \mathbf{V}_b$ and $\mathbf{E} = \mathbf{E}_c \cup \mathbf{E}_s \cup \mathbf{E}_b$, we obtain the desired intermodality graph. Graph \mathcal{G} can be seen as a new category, with objects \mathbf{V} and morphisms \mathbf{E} .

Example 15.1. Consider the **Car** category, describing your road trip through Italy and Switzerland, with

$$\mathbf{V}_c = \{FCO_c, Florence, Bologna, MPX_c, Gotthard, ZRH_c\}, \quad (2)$$

and arrows as in Fig. 1. The nodes represent typical touristic road-trip checkpoints in Italy and Switzerland and the arrows represent highways connecting them.



Figure 1.: The **Car** category.

Furthermore, consider the **Flight** category with

$$\mathbf{V}_f = \{FCO_f, LIN, MPX_f, ZRH_f\} \quad (3)$$

and arrows as in Fig. 2. The nodes represent airports in Italy and Switzerland and the arrows represent connections, offered by specific flight companies.

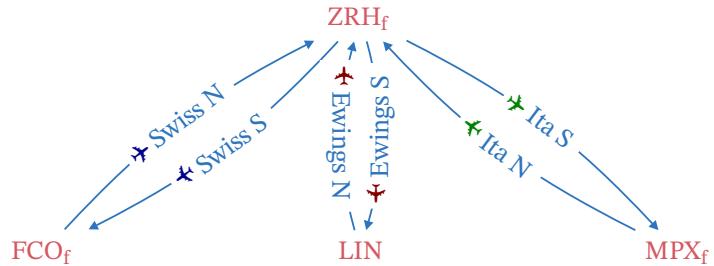


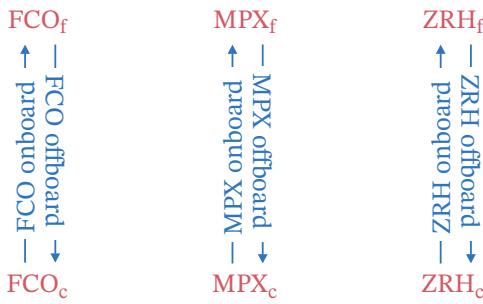
Figure 2.: The **Flight** category.

We then consider the **Board** category, with nodes

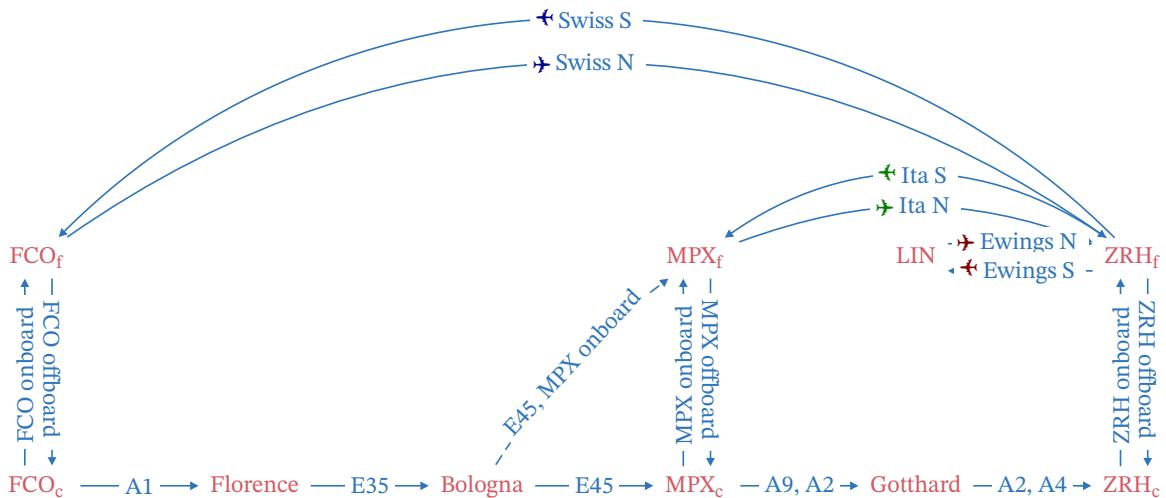
$$\mathbf{V}_b = \{FCO_f, FCO_c, MPX_f, MPX_c, ZRH_f, ZRH_c\} \quad (4)$$

and arrows as in Fig. 3. Nodes represent airports and airport parking lots, and arrows represent the onboarding and off boarding paths we have to walk to get from the parking lot to the airport and vice-versa.

The combination of the three, which we call the *intermodal graph*, can be represented as a graph, in which we use dashed arrows for intermodal morphisms,

**Figure 3.: The Board category.**

arising from composition of morphisms involving multiple modes (Fig. 4). Imagine that you are in the parking lot of ZRH airport and you want to reach Florence . From there, you can onboard to a Swiss flight to FCO_f , will then offboard reaching the parking lot FCO_c , and drive on highway $\text{A}1$ reaching Florence . This is intermodality.

**Figure 4.: Intermodal graph.** The dashed arrows represent intermodal morphisms, and we depict just one of them for simplicity.

The intermodal network category **Intermodal** is the free category on the graph illustrated in Fig. 4.

15.2. Trekking in the Swiss Mountains

In the section we discuss a more “continuum-flavored”(as opposed to “discrete-flavored”) example of how one might describe “connectedness” using a category.

Suppose we are planning a hiking tour in the Swiss Alps. In particular, we wish to consider various routes for hikes. We have a map of the relevant region which uses coordinates $\langle x, y, z \rangle$. We assume the z -th coordinate is given by an “elevation function”, $z = h(x, y)$, and that h is C^1 (a continuously differentiable function). We think of the graph of h as describing the surface of the landscape; call this surface L .

We will now define a category where the morphisms are built from C^1 paths through the landscape, and such that these paths can be composed, essentially, by concatenation. We take paths which are C^1 so that we can speak of the slope (steepness) of a path in any given point, as given by its derivative.

To set things up, we need to have a way to compose C^1 paths such that their composition is again C^1 . For this, the derivative (velocity) at the end of one path must match the starting velocity of the subsequent path.

Definition 15.2 (Berg)

Let **Berg** be the category defined as follows:

- ▷ *Objects*: Objects are tuples $\langle p, v \rangle$, where
 - $p \in L$,
 - $v \in \mathbb{R}^3$ (we think of this as a tangent vector to L at p).
- ▷ *Morphisms*: A morphism $\langle p_1, v_1 \rangle \rightarrow \langle p_2, v_2 \rangle$ is $\langle \gamma, T \rangle$, where
 - $T \in \mathbb{R}_{\geq 0}$,
 - $\gamma : [0, T] \rightarrow L$ is a C^1 function with $\gamma(0) = p_1$ and $\gamma(T) = p_2$, as well as $\dot{\gamma}(0) = v_1$ and $\dot{\gamma}(T) = v_2$ (we take one-sided derivatives at the boundaries).
- ▷ *Identity morphism*: For any object $\langle p, v \rangle$, we define its identity morphism

$$\text{id}_{\langle p, v \rangle} = \langle \gamma, 0 \rangle \quad (5)$$

formally: its path γ is defined on the closed interval $[0, 0]$, (with $T = 0$ and $\gamma(0) = p$). We declare this path to be C^1 by convention, and declare its derivative at 0 to be v .

- ▷ *Composition of morphisms*: Given morphisms

$$\langle \gamma_1, T_1 \rangle : \langle p_1, v_1 \rangle \rightarrow \langle p_2, v_2 \rangle \quad (6)$$

and

$$\langle \gamma_2, T_2 \rangle : \langle p_2, v_2 \rangle \rightarrow \langle p_3, v_3 \rangle, \quad (7)$$

their composition is $\langle \gamma, T \rangle$ with $T = T_1 + T_2$ and

$$\gamma(t) = \begin{cases} \gamma_1(t) & 0 \leq t \leq T_1 \\ \gamma_2(t - T_1) & T_1 \leq t \leq T_1 + T_2. \end{cases} \quad (8)$$

Since we are only amateurs, we don’t feel comfortable hiking on paths that are too steep in some places. We want to only consider paths that have a certain maximum inclination. Mathematically speaking, for any path – as described by a morphism $\langle \gamma, T \rangle$ in the category **Berg**– we can compute its vertical inclination (vertical slope) and renormalize it to give a number in the interval $[-1, +1]$, say. (Here -1 represents vertical descent, and $+1$ represents vertical ascent.) Taking

absolute values of inclinations – call the resulting quantity “steepness” – we can compute the maximum steepness that a path γ obtains over its domain $[0, T]$. This gives, for every hom-set $\text{Hom}_{\mathbf{Berg}}(\langle p_1, v_1 \rangle; \langle p_2, v_2 \rangle)$, a function

$$\text{MaxSteepness} : \text{Hom}_{\mathbf{Berg}}(\langle p_1, v_1 \rangle; \langle p_2, v_2 \rangle) \longrightarrow [0, 1]. \quad (9)$$

Now, suppose we decide that we don’t want to traverse paths which have a maximal steepness greater than $1/2$. Paths which satisfy this condition we call *feasible*. Consider only the feasible paths in **Berg**. If we keep the same objects as **Berg**, but only consider feasible path, will the resulting structure still form a category? Should we restrict the set of objects for this to be true? We’ll let you ponder here; this type of question leads to the notion of a *subcategory*, which we’ll introduce soon in a subsequent chapter.

15.3. Currency categories

In this section, we introduce a kind of category for describing currency exchangers. Our idea is to model currencies as objects of a category, and morphisms will describe ways of exchanging between those currencies. As an example, currency exchangers offer this service.

We start with a set \mathbf{C} of labels for all the currencies we wish to consider:

$$\mathbf{C} = \{\text{EUR, USD, CHF, SGD, ...}\}. \quad (10)$$

Now consider two currencies, say USD and EUR . How can we describe the process of changing an amount of USD to an amount of EUR ? We model this using two numbers: an exchange rate r and a commission c for the transaction. A morphism from one currency to another is given by this pair of numbers $\langle r, c \rangle$. Now, for each morphism, there is a function which takes an amount of the source currency and transforms it into an amount of the target currency. Given an amount $x \in \mathbb{R}$ of USD , this function is:

$$\begin{aligned} f_{\langle r, c \rangle} : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto rx - c. \end{aligned} \quad (11)$$

Note that the commission c is to be intended in the target currency. Of course, for changing USD to EUR , there may be various different banks or agencies which each offer different exchange rates and/or different commissions. Each of these corresponds to a different morphism from USD to EUR .

To build our category, we also need to specify how currency exchangers compose. Given currencies X, Y, Z , and given currency exchangers $\langle r, c \rangle$ from X to Y and $\langle s, d \rangle$ from Y to Z , we define the composition $\langle r, c \rangle \circ \langle s, d \rangle$ to be the currency exchanger from X to Z given by the pair of numbers

$$\langle rs, sc + d \rangle. \quad (12)$$

In other words, we compose currency exchangers as one would expect: we multiply the first and the second exchange rates together, and we add the commissions (paying attention to first transform the first commission into the units of the final target currency).

Finally, we also need to specify identity morphisms for our category. These are currency exchangers which “do nothing”. For any object X , its identity morphism is

$$\langle 1, 0 \rangle, \quad (13)$$

the currency exchanger from X to X with exchange rate “1” and commission “0”.

We now want to check that the composition of currency exchangers as defined above obeys unitality and associativity.

Given $\langle 1, 0 \rangle$ from X to X , $\langle 1, 0 \rangle$ from Y to Y , and $\langle r, c \rangle$ from X to Y we have:

$$\begin{aligned} \langle 1, 0 \rangle \circ \langle r, c \rangle &= \langle 1 \cdot r, r \cdot 0 + c \rangle \\ &= \langle r, c \rangle, \end{aligned} \quad (14)$$

and

$$\begin{aligned} \langle r, c \rangle \circ \langle 1, 0 \rangle &= \langle r \cdot 1, 1 \cdot c + 0 \rangle \\ &= \langle r, c \rangle, \end{aligned} \quad (15)$$

This is unitality. Furthermore, given $\langle s, d \rangle$ from Y to Z , and $\langle t, e \rangle$ from Z to U we

have:

$$\begin{aligned}
 (\langle r, c \rangle \circ \langle s, d \rangle) \circ \langle t, e \rangle &= \langle rs, sc + d \rangle \circ \langle t, e \rangle \\
 &= \langle rst, t(sc + d) + e \rangle \\
 &= \langle r, c \rangle \circ \langle st, te + e \rangle \\
 &= \langle r, c \rangle \circ (\langle s, d \rangle \circ \langle t, e \rangle).
 \end{aligned} \tag{16}$$

This is associativity. Thus, we indeed have a category.

We can formally define the category of currencies **Curr**.

Definition 15.3 (Category Curr)

The *category of currencies Curr* is specified by:

1. *Objects*: a collection of currencies.
2. *Morphisms*: given two currencies X, Y , morphisms between them are currency exchangers $\langle r, c \rangle$ from X to Y .
3. *Identity morphism*: given a currency X , its identity morphism is the currency exchanger $\langle 1, 0 \rangle$. We also call such morphisms “trivial currency exchangers”.
4. *Composition of morphisms*: the composition of morphisms is given by the formula (12), describing a composed currency exchanger.

As an illustration, consider three currency exchange companies ExchATM, MoneyLah, and Frankurrencies, which operate on several currencies (Table 15.1).

Table 15.1. Three currency exchange companies operating different currencies.

Company name	Exchanger label	Direction	Exchange rate a	Fixed commission b
ExchATM	A	USD → CHF	0.95 (in CHF/USD)	2.0 (in CHF)
ExchATM	B	CHF → USD	1.05 (in USD/CHF)	1.5 (in USD)
ExchATM	C	USD → SGD	1.40 (in SGD/USD)	1.0 (in SGD)
MoneyLah	D	USD → CHF	1.00 (in CHF/USD)	1.0 (in CHF)
MoneyLah	E	SGD → USD	0.72 (in USD/SGD)	3.0 (in USD)
Frankurrencies	F	EUR → CHF	1.20 (in CHF/EUR)	0.0 (in CHF)
Frankurrencies	G	CHF → EUR	1.00 (in EUR/CHF)	1.0 (in EUR)

We can represent this information as a graph, where the nodes are the currencies and the edges are particular exchange operations (Fig. 5).

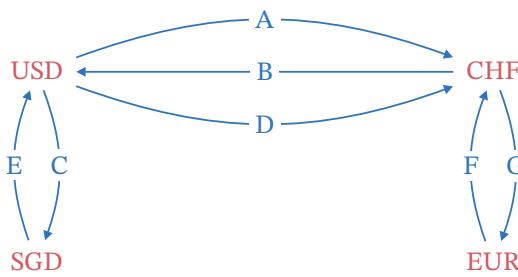


Figure 5. Three currency exchange companies operating different currencies as a graph.

There is a currency category built from the information in Table 15.1 and the graph in Fig. 5. Its collection of objects is the set {EUR, USD, CHF, SGD}, and its morphisms are, in total:

- ▷ the trivial currency exchanger (identity morphism) $\langle 1, 0 \rangle$ for each of the four currencies (which are the objects),
- ▷ the currency exchangers corresponding to each item in Table 15.1,
- ▷ all possible compositions of the currency exchangers listed in Table 15.1.

The phrase “all possible compositions” is a bit vague. What we mean here can be made more precise. It corresponds to a general recipe for starting with a graph G , such as in Fig. 5, and obtaining from it an associated category, called the *free category on G* .

We introduce this concept in Section 13.6.

Exercise36. [Temperatures] Define a category of temperature converters, where the objects are *Celsius*, *Kelvin*, *Fahrenheit*, and the morphisms are the rules to transform a measurement from one unit to another.

Prove that this forms a category.

See solution on page 249.

15.4. Resources dependencies

In engineering design, one creates *systems* out of *components*. Each component has a reason to be in there. We will show how category theory can help in formalizing the chains of causality that underlie a certain design.

We will need to reason at the level of abstraction where we consider the “function”, or “functionality”, which each component provides, and the “requirements” that are needed to provide the function.

We will start with a simple example of the functioning principle of an electric car.

In an electric car, there is a battery, a store of the electric energy resource. We can see the production of motion as the series of two transformations:

- ▷ The **motor** transmutes the **electricity** into **rotation**.
- ▷ The **rotation** is converted into **translation** by the **wheels** and their friction with the road.

We see that there are two types of things in this example:

1. The “transmuted”: the **electricity**, the **rotation**, the **translation**: these are objects in a category of transmuted resources.
2. The “transmuters”: the **motor** and **wheels**: these are morphisms in a category of transmuted resources.

For a first qualitative description of the scenario, we might choose to just keep track of what is transmuted into what. We can draw a diagram in which each resource is a point (Fig. 6).



Figure 6.: Resources in the electric car example.

Now, we can draw arrows between two points if there is a transmuter between them.

We choose the direction of the arrow such that

$$X \xrightarrow{\text{transmuter}} Y \tag{17}$$

means that “using **transmuter**, having **Y** is sufficient to produce **X**”.

Remark 15.4 (Are we going the wrong direction?). The chosen direction for the arrows is completely the opposite of what you would expect if you thought about “input and outputs”. There is a good reason to use this convention, though it will be apparent only a few chapters later. In the meantime, it is a good exercise to liberate your mind about the preconception of what an arrow means; in category theory there will be categories where the arrows represent much more abstract concepts than input/output.

Another way to write (17) would be as follows:

$$\text{transmuter} : X \rightarrow Y. \tag{18}$$

With these conventions, we can describe the two transmuters as these arrows:

$$\begin{aligned} \text{motor} &: \text{rotation} \rightarrow \text{electricity}, \\ \text{wheels} &: \text{translation} \rightarrow \text{rotation}. \end{aligned} \tag{19}$$

We can put these arrows in the diagrams, and obtain the following (Fig. 7).

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

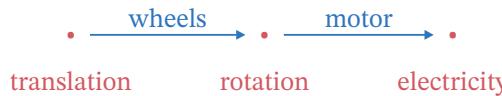


Figure 7.: Transmutes are arrows between resources.

In this representation, the arrows are the components of the system. The basic rule is *composition*. If we use the semantics that an arrow from resource X to resource Y means “having Y is enough to obtain X ”, then, since Y is enough for Y per definition, we can add a self-loop for each resource. We will call the self-loops *identities* (Fig. 8).

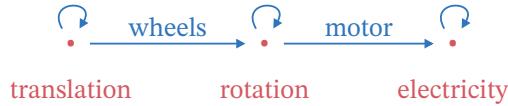


Figure 8.: System components and identities.

Furthermore, we might consider the idea of composition of arrows. Suppose that we know that

$$X \xrightarrow{f} Y \quad \text{and} \quad Y \xrightarrow{g} Z, \quad (20)$$

that is, using a g we can get a Y from a Z , and using a f we can get a X from a Y , then we conclude that using a f and a g we can get an X from a Z .

In our example, if the arrows `wheels` and `motor` exist, then also the arrow “`wheels` then `motor`” exists (Fig. 9).

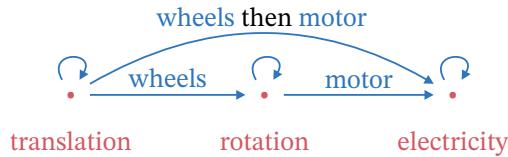


Figure 9.: Composition of system components.

So far, we have drawn only one arrow between two points, but we can draw as many as we want. If we want to distinguish between different brands of motors, we would just draw one arrow for each model. For example, Fig. 10 shows two models of motors (`motor A`, and `motor B`) and two models of wheels (`wheels U` and `wheels V`).

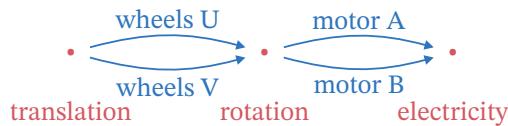


Figure 10.: Multiple models for wheels and motors.

The figure implies now the existence of *four* composed arrows: “`wheels U` then `motor A`”, “`wheels U` then `motor B`”, “`wheels V` then `motor A`”, and “`wheels V` then `motor B`”, all going from `translation` to `electricity`;

Note that we may save some ink when drawing diagrams of morphisms:

- ▷ We do not need to draw the identity arrows from one object to itself, because, by Def. 13.9, they always exist.
- ▷ Given arrows $X \rightarrow Y$ and $Y \rightarrow Z$, we do not need to draw their composition because, by Def. 13.9, this composition is guaranteed to exist.

With these conventions, we can just draw the arrows `motor` and `wheels` in the diagram, and the rest of the diagram is implied (Fig. 11).

In particular, the electric car example corresponds to the category **C** specified by

- ▷ *Objects:* $\text{Ob}_C = \{\text{electricity}, \text{rotation}, \text{translation}\}$.

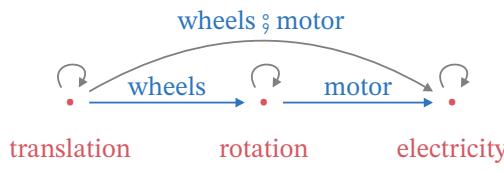


Figure 11.: Electric car example. The gray arrows are implied by the properties of a category.

▷ **Morphisms:** The system components are the morphisms. For instance, we have **motor**, **wheels**, and the morphism **wheels ; motor**, implied by the properties of the category.

We can slightly expand this example by noting the reverse transformations. In an electric car it is possible to regenerate power; that is, we can obtain **rotation** of the **wheels** from **translation** (via the morphism **move**), and then convert the **rotation** into **electricity** (via the morphism **dynamo**) (Fig. 12, Fig. 13).

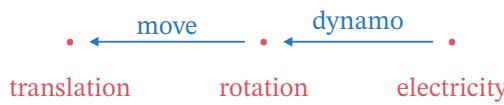


Figure 12.: Electric power can be produced from motion.

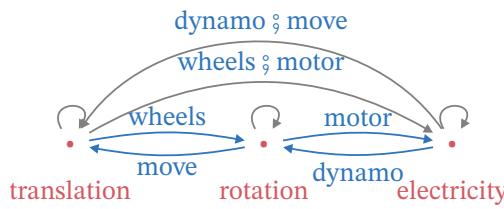


Figure 13.: Electric car example: forward and backward transformations.

Given the semantics of the arrows in a category, all compositions of arrows exist, even if they are not drawn explicitly. For example, we can consider the composition

$$\text{wheels ; motor ; dynamo ; move}, \quad (21)$$

which converts **translation** into **rotation**, into **electricity**, then back to **rotation** and **translation**. Note that this is an arrow that has the same head and tail as the identity arrow on **translation** (Fig. 14). However, these two arrows are not necessarily the same. In this example we are representing physical systems, so we would in fact not expect them to be the same, since there will be some losses during the many conversions.

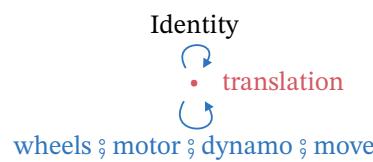


Figure 14.: There can be multiple morphisms from an object to itself.

The directionality of the arrows is also important. While the convention of which resource is the tail and which the head is just a typographic convention, it might be the case that we know how to convert one resource into another, but not vice versa. Figure 15 shows an example of a diagram that describes a process which is definitely not invertible.

Example 15.5. Given any category **C**, and any object $X \in \mathbf{C}$, the set of *endomorphisms* $\text{Hom}_{\mathbf{C}}(X; X)$ is a monoid. The category depicted in Fig. 16 has three objects **X**, **Y**, **Z** and several morphisms. **X** has four endomorphisms, **Y** two, and **Z**

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.



Figure 15.: An example of a process which is not invertible.

three (including identity morphisms). Take the binary operation \circ to be the composition \circ in \mathbf{C} , and the neutral element to be the identity id_X . The associativity and unitality laws of the category \mathbf{C} coincide with the ones of the monoid's definition, and are satisfied. Therefore, we can identify a monoid as a one-object category.

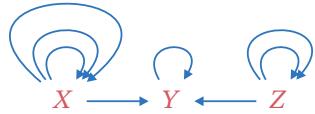


Figure 16.

15.5. Design problems form a category

Series composition

We will define several ways to connect design problems together. The first and most basic way is series composition, or just “composition”.

Definition 15.6 (Series composition)

Let $\mathbf{d} : \mathbf{P} \rightarrow \mathbf{Q}$ and $\mathbf{e} : \mathbf{Q} \rightarrow \mathbf{R}$ be design problems. We define their *series composition* $(\mathbf{d} ; \mathbf{e}) : \mathbf{P} \rightarrow \mathbf{R}$ as:

$$\begin{aligned} (\mathbf{d} ; \mathbf{e}) : \mathbf{P}^{\text{op}} \times \mathbf{R} &\rightarrow \text{Pos Bool}, \\ \langle p^*, r \rangle &\mapsto \bigvee_{q \in Q} \mathbf{d}(p^*, q) \wedge \mathbf{e}(q^*, r). \end{aligned} \quad (22)$$

The series composition $(\mathbf{d} ; \mathbf{e})$ judges a pair $\langle p^*, r \rangle$ as feasible if and only if there exists a $q \in Q$ such that $\mathbf{d}(p^*, q)$ and $\mathbf{e}(q^*, r)$ are feasible.

Given a set \mathbf{A} and a map $s : \mathbf{A} \rightarrow \text{Bool}$, we can define the boolean $\bigvee_{a \in A} s(a)$ by

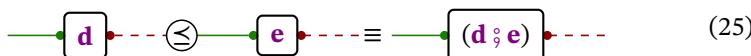
$$\bigvee_{a \in A} s(a) := \begin{cases} \top & \text{if there exists } a \in A \text{ for which } s(a) = \top, \\ \perp & \text{if there exists no } a \in A \text{ for which } s(a) = \top. \end{cases} \quad (23)$$

In (22) we could have written “ $\exists_{q \in Q}$ ” instead of “ $\bigvee_{q \in Q}$ ”:

$$\exists_{q \in Q} \mathbf{d}(p^*, q) \wedge \mathbf{e}(q^*, r). \quad (24)$$

Using \bigvee form highlights the connection with an integration operation \int_q .

We use the same diagrammatic notation for DPs as for DPIs. We represent series composition as



One can notice the “co-design constraint” \leq , which can be interpreted as follows. The **resource** required by a component is limited by the **functionality** produced by another component.

When viewing compositions (and larger diagrams) formed from these boxes, it is tempting to interpret the boxes as input-output processes. However, that would be misleading. The arrows do not represent information flow, materials flow, or energy flow. Design problems do not represent input-output processes but rather a static calculus of requirements—a requirements flow.

When the posets involved are finite, the series composition of design problems can be calculated visually, using the kind of representation discussed in Example 7.14.

To explain how this works, consider the design problem

$$\mathbf{d} : \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow \text{Pos Bool}, \quad (26)$$

from Example 7.14, visualized again for convenience in the first row of Fig. 17.

And consider another design problem of the type

$$\mathbf{e} : \mathbf{Q}^{\text{op}} \times \mathbf{R} \rightarrow \text{Pos Bool}, \quad (27)$$

as given by the visualization in the first row of Fig. 17.

We can calculate the series composition $\mathbf{d} ; \mathbf{e}$ by tracing paths in the “composite”

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

visualization given in the second row of Fig. 17. Namely, a pair $\langle a, c \rangle$ is in the feasibility set of $\mathbf{d} ; \mathbf{e}$ if and only if we can trace a path from a to c by only moving upwards in the posets P , Q , and R , or crossing from one poset to another following dashed arrows in the direction they are pointing. Thus, the visualization of the composite $\mathbf{d} ; \mathbf{e}$ is as in the third row of Fig. 17.

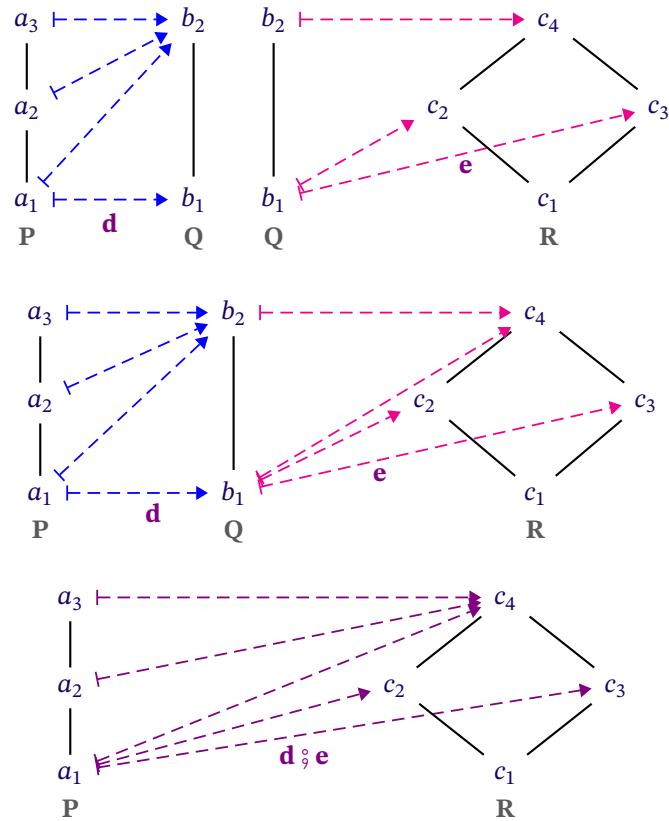


Figure 17.

Let us check that, given design problems \mathbf{d} and \mathbf{e} , their series composition $(\mathbf{d} ; \mathbf{e})$ is in fact a design problem.

Lemma 15.7. Series composition as in (22) is monotone in p and r .

Proof. We need to show that $(\mathbf{d} ; \mathbf{e})(p^*, r)$ is monotone in p^* and r . Because \mathbf{d} represents a design problem, $\mathbf{d}(p^*, q)$ is monotone in p^* , and similarly $\mathbf{e}(q^*, r)$ is monotone in r . The conjunction “ \wedge ” is monotone in both variables, and likewise the “ \vee ” operation. \square

We can show two important properties for the “ \circ ” operation: associativity and unitality.

Lemma 15.8. The series composition operation as in (22) is associative:

$$(\mathbf{d} ; \mathbf{e}) ; \mathbf{g} = \mathbf{d} ; (\mathbf{e} ; \mathbf{g}). \quad (28)$$

Proof. Consider $\mathbf{d} : \mathbf{P} \rightarrowtail \mathbf{Q}$, $\mathbf{e} : \mathbf{Q} \rightarrowtail \mathbf{R}$, $\mathbf{g} : \mathbf{R} \rightarrowtail \mathbf{S}$. To show that the operation is associative, we can use distributivity and commutativity in

Bool:

$$\begin{aligned}
 ((\mathbf{d} ; \mathbf{e}) ; \mathbf{g})(p^*, s) &= \bigvee_{r \in R} \left(\bigvee_{q \in Q} \mathbf{d}(p^*, q) \wedge \mathbf{e}(q^*, r) \right) \wedge \mathbf{g}(r^*, s) \\
 &= \bigvee_{r \in R} \left(\bigvee_{q \in Q} \mathbf{d}(p^*, q) \wedge \mathbf{e}(q^*, r) \wedge \mathbf{g}(r^*, s) \right) \\
 &= \bigvee_{q \in Q} \mathbf{d}(p^*, q) \wedge \left(\bigvee_{r \in R} \mathbf{e}(q^*, r) \wedge \mathbf{g}(r^*, s) \right) \\
 &= (\mathbf{d} ; (\mathbf{e} ; \mathbf{g}))(p^*, s).
 \end{aligned} \tag{29}$$

□

Because of associativity, we can write $\mathbf{d} ; \mathbf{e} ; \mathbf{g}$ without ambiguity. Associativity of composition is represented as:

$$\begin{aligned}
 \text{---} \bullet \boxed{\mathbf{d}} \bullet \textcircled{S} \bullet \boxed{\mathbf{e}} \bullet \textcircled{S} \bullet \boxed{\mathbf{g}} \bullet \text{---} \equiv \text{---} \bullet \boxed{(\mathbf{d} ; \mathbf{e})} \bullet \textcircled{S} \bullet \boxed{\mathbf{g}} \bullet \text{---} \\
 \equiv \text{---} \bullet \boxed{\mathbf{d}} \bullet \textcircled{S} \bullet \boxed{(\mathbf{e} ; \mathbf{g})} \bullet \text{---}
 \end{aligned} \tag{30}$$

Identity for DP

There exists an identity for the “ \circ ” operation. We define the identity $\mathbf{id}_P : P \rightarrow P$ as follows.

Definition 15.9 (Identity design problem)

For any poset P , the *identity design problem* $\mathbf{id}_P : P \rightarrow P$ is a monotone map

$$\begin{aligned}
 \mathbf{id}_P : P^{\text{op}} \times P &\rightarrow_{\text{Pos}} \text{Bool}, \\
 \langle p_1^*, p_2 \rangle &\mapsto p_1 \leq_P p_2.
 \end{aligned} \tag{31}$$

Remark 15.10 (Monotonicity of the identity). Let's consider $p_1' \leq_P p_1$. If it holds $p_1 \leq_P p_2$, then it also holds $p_1' \leq_P p_2$. Similarly, now consider $p_2 \leq_P p_2'$. If it holds $p_1 \leq_P p_2$, then it also holds $p_1 \leq_P p_2'$.

In the diagrammatic notation, we represent \mathbf{id}_P as:

$$P \xrightarrow{\quad \boxed{\mathbf{id}_P} \quad} P \tag{32}$$

Lemma 15.11. The series composition operation as in (22) satisfies the left and right unit laws ((33)).

$$\text{---} \bullet \boxed{\mathbf{id}_P} \bullet \textcircled{S} \bullet \boxed{\mathbf{d}} \bullet \textcircled{S} \bullet \boxed{\mathbf{id}_Q} \bullet \text{---} \equiv \text{---} \bullet \boxed{\mathbf{d}} \bullet \text{---} \tag{33}$$

Proof. Given $\mathbf{d} : P \rightarrow Q$, we need to show:

$$\mathbf{id}_P ; \mathbf{d} = \mathbf{d} = \mathbf{d} ; \mathbf{id}_Q. \tag{34}$$

In the following, we prove $\mathbf{id}_P ; \mathbf{d} = \mathbf{d}$. Proving $\mathbf{d} ; \mathbf{id}_Q = \mathbf{d}$ is similar. Consider

the poset **Bool**. Since for $x, y \in \text{Bool}$,

$$\frac{x \cong y}{x = y}, \quad (35)$$

(also referred to as skeletality [5]), we just need to show that $\mathbf{d} \leq \text{id}_P ; \mathbf{d}$ and $\text{id}_P ; \mathbf{d} \leq \mathbf{d}$. Here, $\mathbf{d} \leq \mathbf{e}$ means $\mathbf{d}(p^*, q) \leq_{\text{Bool}} \mathbf{e}(p^*, q)$ for all $p \in P, q \in Q$. We have

$$\begin{aligned} \mathbf{d}(p^*, q) &= \top \wedge \mathbf{d}(p^*, q) \\ &= \text{id}_P(p^*, p) \wedge \mathbf{d}(p^*, q) \\ &\leq \bigvee_{p' \in P} \text{id}_P(p^*, p') \wedge \mathbf{d}(p'^*, q) \\ &= (\text{id}_P ; \mathbf{d})(p^*, q). \end{aligned} \quad (36)$$

For the other direction, we need to show that $\text{id}_P ; \mathbf{d} \leq \mathbf{d}$:

$$\bigvee_{p' \in P} \text{id}_P(p^*, p') \wedge \mathbf{d}(p'^*, q) \leq \mathbf{d}(p^*, q). \quad (37)$$

This holds if and only if $\text{id}_P(p^*, p') \wedge \mathbf{d}(p'^*, q) \leq \mathbf{d}(p^*, q)$ for some $p' \in P$. If there is no such p' , then the inequality holds ($\perp \leq \perp$ and $\perp \leq \top$). If there is such an element p' , it means that $\text{id}_P(p^*, p') = \top$ and $\mathbf{d}(p'^*, q) = \top$. We know that

$$\frac{\text{id}_P(p^*, p') = \top}{\mathbf{d}(p^*, q) = \top} \quad (38)$$

and hence $\mathbf{d}(p^*, q) = \top$. \square

The category DP

Finally, we can declare that the design problems so defined are morphisms in a category that we call **DP**.

Definition 15.12 (Category of design problems **DP**)

The category of design problems **DP** consists of the following constituents:

1. *Objects*: The objects of **DP** are posets.
2. *Morphisms*: The morphisms of **DP** are design problems (Def. 7.12).
3. *Identity morphism*: The identity morphism $\text{id}_P : P \rightarrow P$ is given by Def. 15.9.
4. *Composition operation*: Given morphisms $\mathbf{d} : P \rightarrow Q$ and $\mathbf{e} : Q \rightarrow R$, their composition $\mathbf{d} ; \mathbf{e} : P \rightarrow R$ is given by Def. 15.6.

We have already shown that the composition operator “ $;$ ” is associative and unital, and that the composition of two design problems is a design problem (closure). Therefore, **DP** is a category.

DP is called **Feas** or **Prof_{Bool}** in [5].

Graded exercise E.10 (ComposingDesignProblems)

Consider the following posets, given in terms of Hasse diagrams:

▷ **Speed**:

fast
|
slow

▷ **Size:**

large
|
small

▷ **Money:**

120K
|
40K
|
20K

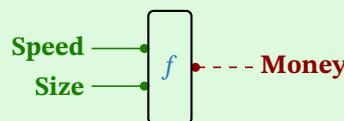
▷ **Time:**

long
|
short

Furthermore let the poset $\mathbf{Speed} \times \mathbf{Size}$ be equipped with the standard product partial ordering.

Consider the design problem given by the monotone function

$$f : (\mathbf{Speed} \times \mathbf{Size})^{\text{op}} \times \mathbf{Money} \rightarrow \mathbf{Bool} \quad (39)$$

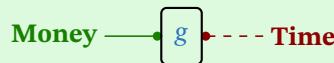


with

$$\begin{aligned} f^{-1}(\top) = & \{ \langle \langle \text{slow}, \text{small} \rangle, \text{20K} \rangle, \langle \langle \text{slow}, \text{small} \rangle, \text{40K} \rangle, \langle \langle \text{slow}, \text{small} \rangle, \text{120K} \rangle, \\ & \langle \langle \text{slow}, \text{large} \rangle, \text{40K} \rangle, \langle \langle \text{slow}, \text{large} \rangle, \text{120K} \rangle \\ & \langle \langle \text{fast}, \text{small} \rangle, \text{120K} \rangle \} \end{aligned}$$

along with the design problem given by the monotone function

$$g : \mathbf{Money}^{\text{op}} \times \mathbf{Time} \rightarrow \mathbf{Bool} \quad (40)$$



with

$$g^{-1}(\top) = \{ \langle \text{20K}, \text{short} \rangle, \langle \text{20K}, \text{long} \rangle, \langle \text{40K}, \text{long} \rangle \}.$$

1. Compute the series composition $f ;_{\text{DP}} g$ in the category of design problems. The result should be a design problem described in terms of a monotone function $(\mathbf{Speed} \times \mathbf{Size})^{\text{op}} \times \mathbf{Time} \rightarrow \mathbf{Bool}$.
2. We interpret elements of $\mathbf{Speed} \times \mathbf{Size}$ as properties of cars that Alice considers buying. \mathbf{Money} represents the amounts of money that she would need to buy one of said cars, and \mathbf{Time} represents the amounts of time Alice could spend saving money. The feasibility relations f and g describe what is possible for Alice. According to $f ;_{\text{DP}} g$, is it feasible for

Alice to buy a fast small car? If yes, will she have to work, at minimum, a long or a short amount of time in order to save enough money to buy it?

Graded exercise E.11 (DPComposition)

Consider the following posets, given in terms of Hasse diagrams:

▷ **Time:**



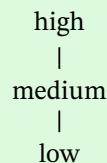
▷ **Money:**



▷ **Tech complexity:**



▷ **Task complexity:**

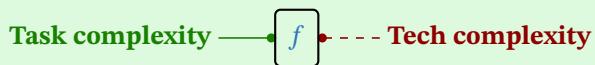


Let the poset **Time** × **Money** be equipped with the standard product partial ordering.

You are in charge of an engineering team that should develop a robotic system. It is not yet clear what strategy you wish follow for realizing the project, and you wish to do a preliminary feasibility study. You will potentially need to make some tradeoffs between the *time* that your team works on the project and the *money* that you invest. Furthermore, you will need to decide between implementing either a cutting edge *fancy* robotic system, or a *simple* one. This will have an impact on the *complexity of the tasks* that the system will be able to handle.

Design problems capture feasibility relations. In your case, the feasibility relation between **Task complexity** and **Tech complexity** is given by the design problem

$$f : \mathbf{Task\ complexity}^{\text{op}} \times \mathbf{Tech\ complexity} \rightarrow \mathbf{Bool} \quad (41)$$

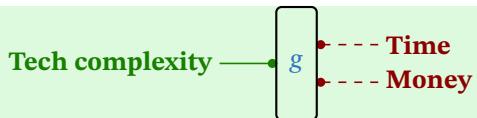


with

$$f^{-1}(\top) = \{ \langle \text{high, fancy} \rangle, \langle \text{medium, fancy} \rangle, \langle \text{low, fancy} \rangle, \langle \text{low, simple} \rangle \}$$

and the feasibility relation between **Tech complexity** and **Time** × **Money** is given by the design problem

$$g : \mathbf{Tech\ complexity}^{\text{op}} \times (\mathbf{Time} \times \mathbf{Money}) \rightarrow \mathbf{Bool} \quad (42)$$



with

$$g^{-1}(\top) = \{\langle \text{fancy}, \langle \text{long, 100K} \rangle \rangle, \langle \text{simple}, \langle \text{long, 100K} \rangle \rangle, \langle \text{simple}, \langle \text{long, 20K} \rangle \rangle, \langle \text{simple}, \langle \text{short, 100K} \rangle \rangle\}.$$

Your tasks in this exercise:

1. Compute the series composition $f ;_{\text{DP}} g$ in the category of design problems.
2. Based on the previous calculation, would it be feasible to build a robotic system that can handle medium-complexity tasks if your team works only a short time on the project but invests 100K?

15.6. Procedures

In programming, it is common to use the name *function*; we keep that word to denote mathematical functions: morphisms of the category **Set**. We use the word *procedure* to refer to pieces of code that run on a computer or virtual machine.

Procedures are richer than functions:

- ▷ They might be non-deterministic: not always return the same value.
- ▷ They might have side effects: change the world in some way.
- ▷ They have resource consumption associated to them: they need memory and computation to produce the result.

The category **Set** is not sufficient to describe these properties, but we can easily invent categories that are built on top of **Set** to add these properties.

We are going to do this in stages. First, we are going to define a category of procedures that keeps track of running time. Then we are going to keep track of time depending on the size of the input. Finally, we are going to keep track of memory usage.

Modeling execution time

We can model execution time by keeping track of an additional real number in the morphism.

Definition 15.13 (Semicategory **ProcTime**)

The semicategory **ProcTime** consists of the following constituents:

1. *Objects*: The objects of **ProcTime** are the objects of **Set**.
2. *Morphisms*: A morphism

$$f : X \rightarrow_{\text{ProcTime}} Y \quad (43)$$

is a pair $\langle f_e, t \rangle$, where $f_e : X \rightarrow \text{Set} Y$ is a regular function that describes what the procedure computes and $t > 0$ is a real number representing “execution time”.

3. *Composition operation*: Given two morphisms

$$f : X \rightarrow_{\text{ProcTime}} Y \quad \text{and} \quad g : Y \rightarrow_{\text{ProcTime}} Z, \quad (44)$$

represented by $\langle f_e, t_1 \rangle$ and $\langle g_e, t_2 \rangle$, their composition is given by $\langle f_e \circ \text{Set } g_e, t_1 + t_2 \rangle$.

This category allows distinguishing between different implementations of the same functions with different computational requirements. For example, there could be two morphisms $\langle f_e, 1 \rangle$ and $\langle f_e, 10 \rangle$ which compute the same function, but the second takes 10 times as much.

Modeling sized data

The category **ProcTime** models execution time, but it does not model how the execution time depends on the size of the input.

If we wanted to capture such ideas, we need to have an explicit way to talk about the size of the input. For this we introduce “sized sets”.

Definition 15.14 (Sized set)

A sized set is a pair $\langle \mathbf{A}, \text{size} \rangle$, where \mathbf{A} is a set, and $\text{size} : \mathbf{A} \rightarrow \mathbb{N} \cup \{\infty\}$ is the size function.

For example, we could have $\mathbf{A} = \text{lists of integers}$ and $\text{size} = \text{length of list}$.

Now we can create a category whose objects are sized sets.

The procedures are morphisms that keep track of how the size of the input influences the size of the output. For example, a sorting function produces an output that is the same size as the input, while a function that chooses an element of a list produces an output of size 1, no matter how large the input is.

Definition 15.15 (Semicategory **ProcSize**)

The semicategory **ProcSize** consists of the following constituents:

1. *Objects*: The objects are sized sets.
2. *Morphisms*: A morphism

$$f : X \rightarrow_{\text{ProcSize}} Y \quad (45)$$

between the two objects

$$X = \langle \mathbf{A}, \text{size}_{\mathbf{A}} \rangle \quad \text{and} \quad Y = \langle \mathbf{B}, \text{size}_{\mathbf{B}} \rangle \quad (46)$$

is a pair

$$\langle f_e, \sigma \rangle, \quad (47)$$

where:

- a) $f_e : \mathbf{A} \rightarrow \mathbf{B}$ is the function computed;
- b) $\sigma : \mathbb{N} \rightarrow \mathbb{N}$ keeps track of how the size changes.

3. *Composition*: The composition of

$$\langle f_e, \sigma_f \rangle \quad \text{and} \quad \langle g_e, \sigma_g \rangle \quad (48)$$

is given by

$$\langle f_e \circ g_e, \sigma_f \circ \sigma_g \rangle. \quad (49)$$

Modeling data-size-dependent running times

Now we can create a category that keeps track of both data size and execution time, possibly dependent on data size.

Definition 15.16 (Semicategory **ProcSizeTime**)

The semicategory **ProcSizeTime** consists of the following constituents:

1. *Objects*: The objects are sized sets.
2. *Morphisms*: A morphism

$$f : X \rightarrow_{\text{ProcSize}} Y \quad (50)$$

between the two objects

$$X = \langle \mathbf{A}, \text{size}_{\mathbf{A}} \rangle \quad \text{and} \quad Y = \langle \mathbf{B}, \text{size}_{\mathbf{B}} \rangle \quad (51)$$

is a tuple

$$\langle f_e, \sigma, \text{dur} \rangle, \quad (52)$$

where:

- a) $f_e : \mathbf{A} \rightarrow \mathbf{B}$ is the function computed;
- b) $\sigma : \mathbb{N} \rightarrow \mathbb{N}$ keeps track of how the size changes.
- c) $\text{dur} : \mathbb{N} \rightarrow \mathbb{R}$ says what is the computation time as a function of instance size;

3. *Composition:* The composition of

$$\langle f_1, \sigma_1, \text{dur}_1 \rangle \quad \text{and} \quad \langle g_2, \sigma_2, \text{dur}_2 \rangle \quad (53)$$

is given by

$$\langle f_1 ; g_2, \sigma_1 ; \sigma_2, \text{dur}_{1,2} \rangle, \quad (54)$$

where $\text{dur}_{1,2}$ is defined as

$$\begin{aligned} \text{dur}_{1,2} : \mathbb{N} &\rightarrow \mathbb{R}, \\ n &\mapsto \text{dur}_1(n) + \text{dur}_2(\sigma_1(n)). \end{aligned} \quad (55)$$

Exercise37. Check that associativity holds for the composition in **ProcSizeTime**.

See solution on page 250.

Graded exercise E.12 (Asymptotics)

The category **ProcSizeTime** as defined thinks of time as real numbers, and size of the data as integers. In computer science, it is convenient to use *asymptotic analysis*. For example, we know that, in the general case, ordering a list of n elements takes $O(n \log n)$ steps.

Discuss how you can extend or modify **ProcSizeTime** to be able to capture asymptotic analysis.

15.7. Software dependencies

We will discuss two examples:

- ▷ Makefiles.
- ▷ Python packages dependencies.





16. Constructing categories

In this chapter we discuss various ways of building new categories from old ones. Given categories **C** and **D**, we will see how to build their cartesian product **C** × **D** and their direct sum **C** + **D**, respectively. In addition to these constructions, we will see how to start with a category **C** and construct its *opposite category* **C**^{op}, as well as various other categories derived from **C** by thinking of kinds of diagrams in **C** as objects themselves.

16.1 Product of Categories	236
16.2 Disjoint Union of Categories . . .	237
16.3 Opposite Category	238
16.4 Arrow construction	239
16.5 Twisted arrow construction	240
16.6 (Co)slice construction	242

16.1. Product of Categories

Definition 16.1 (Cartesian product of categories)

Given two categories \mathbf{C} and \mathbf{D} , their *cartesian product* $\mathbf{C} \times \mathbf{D}$ is the category specified as follows:

1. *Objects*: Objects are pairs $\langle X, Y \rangle$, with $X \in \text{Ob}_{\mathbf{C}}$ and $Y \in \text{Ob}_{\mathbf{D}}$.
2. *Morphisms*: A morphism from $\langle X, Y \rangle$ to $\langle Z, U \rangle$ is a pair of morphisms

$$\langle f, g \rangle : \langle X, Y \rangle \rightarrow \langle Z, U \rangle, \quad (1)$$

with $f : X \rightarrow_{\mathbf{C}} Z, g : Y \rightarrow_{\mathbf{D}} U$.

3. *Composition*: The composition of morphisms is given by composing each component of the pair separately:

$$\langle f, g \rangle ;_{\mathbf{C} \times \mathbf{D}} \langle h, i \rangle = \langle f ;_{\mathbf{C}} h, g ;_{\mathbf{D}} i \rangle. \quad (2)$$

4. *Identity morphisms*: Given objects $X \in \text{Ob}_{\mathbf{C}}$ and $Y \in \text{Ob}_{\mathbf{D}}$, the identity morphism on $\langle X, Y \rangle$ is the pair $\langle \text{id}_X, \text{id}_Y \rangle$.

Remark 16.2. In a manner analogous to Def. 16.1 we can also define the cartesian product of any finite number of categories. For example, if $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are three categories, their triple cartesian product $\mathbf{A} \times \mathbf{B} \times \mathbf{C}$ is a category whose objects are triples $\langle X, Y, Z \rangle$, with $X \in \text{Ob}_{\mathbf{A}}, Y \in \text{Ob}_{\mathbf{B}}, Z \in \text{Ob}_{\mathbf{C}}$. For any $n \in \mathbb{N}$, the n -fold cartesian product of a category \mathbf{C} with itself is denoted \mathbf{C}^n .

16.2. Disjoint Union of Categories

Definition 16.3 (Disjoint union category)

Given two categories **C** and **D**, their *disjoint union* **C + D** is the category specified as follows:

1. *Objects*: $\text{Ob}_{\mathbf{C}+\mathbf{D}} := \text{Ob}_{\mathbf{C}} + \text{Ob}_{\mathbf{D}}$. That is, objects of **C + D** are tuples of the form $\langle i, X \rangle$, with $i = 1$ and $X \in \text{Ob}_{\mathbf{C}}$ or $i = 2$ and $X \in \text{Ob}_{\mathbf{D}}$.

2. *Morphisms*: Given objects $\langle i, X \rangle, \langle j, Y \rangle \in \text{Ob}_{\mathbf{C}+\mathbf{D}}$,

$$\text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle i, X \rangle; \langle j, Y \rangle) := \begin{cases} \text{Hom}_{\mathbf{C}}(X; Y) & \text{if } i = j = 1, \\ \text{Hom}_{\mathbf{D}}(X; Y) & \text{if } i = j = 2, \\ \emptyset & \text{else.} \end{cases} \quad (3)$$

3. *Composition*: The composition operations $\circ_{\mathbf{C}+\mathbf{D}}$, which are functions from

$$\text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle i, X \rangle; \langle j, Y \rangle) \times \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle j, Y \rangle; \langle k, Z \rangle)$$

to

$$\text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle i, X \rangle; \langle k, Z \rangle),$$

are equal to

$$\begin{cases} \circ_{\mathbf{C}} & \text{if } i = j = k = 1, \\ \circ_{\mathbf{D}} & \text{if } i = j = k = 2, \end{cases}$$

and equal to the unique function

$$\emptyset \rightarrow \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle i, X \rangle; \langle k, Z \rangle)$$

in all other cases.

4. *Identity morphisms*: The identities are copied from either category:

$$\text{id}_{\langle 1, X \rangle}^{\mathbf{C}+\mathbf{D}} := \text{id}_X^{\mathbf{C}}, \quad (4)$$

$$\text{id}_{\langle 2, X \rangle}^{\mathbf{C}+\mathbf{D}} := \text{id}_X^{\mathbf{D}}. \quad (5)$$

Remark 16.4. If you think about categories in diagrammatic form, this operation corresponds to placing two categories side-by-side, without connecting them.

Remark 16.5. A remark similar to Remark 16.2 holds here, too: we can form the disjoint union of any finite number of categories. For example, **A + B + C** for three categories **A, B, C**.

Exercise 38. Show that the disjoint union of two categories is indeed again a category.

See solution on page 250.

16.3. Opposite Category

Definition 16.6 (Opposite category)

Given a category \mathbf{C} , its *opposite category* \mathbf{C}^{op} is specified by:

1. *Objects*: $\text{Ob}_{\mathbf{C}^{\text{op}}} = \text{Ob}_{\mathbf{C}}$.
2. *Morphisms*: Given objects $X, Y \in \text{Ob}_{\mathbf{C}^{\text{op}}} = \text{Ob}_{\mathbf{C}}$,

$$\text{Hom}_{\mathbf{C}^{\text{op}}}(X; Y) := \text{Hom}_{\mathbf{C}}(Y; X). \quad (6)$$

For each morphism $f : X \rightarrow_{\mathbf{C}} Y$, there is a morphism $f^{\text{op}} : Y \rightarrow_{\mathbf{C}^{\text{op}}} X$. Graphically, given

$$X \xrightarrow{f} Y \quad (7)$$

we have

$$Y \xrightarrow{f^{\text{op}}} X \quad (8)$$

3. *Composition*: Given morphisms

$$f^{\text{op}} : Z \rightarrow_{\mathbf{C}^{\text{op}}} Y \quad \text{and} \quad g^{\text{op}} : Y \rightarrow_{\mathbf{C}^{\text{op}}} X \quad (9)$$

their composition is defined as

$$f^{\text{op}} ;_{\mathbf{C}^{\text{op}}} g^{\text{op}} := (g ;_{\mathbf{C}} f)^{\text{op}}. \quad (10)$$

4. *Identity morphisms*: given by the identities of the original category \mathbf{C} .

Given $X \in \text{Ob}_{\mathbf{C}}$, we will sometimes (though not always) write X^{op} to signify when we are thinking of X as an object of $\text{Ob}_{\mathbf{C}^{\text{op}}}$.

Graded exercise E.13 (`OppositeCat`)

Verify that Def. 16.6 defines a category. In other words, check that its constituents satisfy the conditions of associativity and unitality.

Example 16.7 (Opposite of a poset). A single poset $\mathbf{P} = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ can be described as a category, in which each point is an object, and there is a morphism between two objects x and y if and only if $x \leq_{\mathbf{P}} y$. We have defined the opposite of a poset in Section 6.3. The opposite category of a category for a poset, is the category for the opposite poset.

$$(\mathbf{C}(\mathbf{P}))^{\text{op}} \simeq \mathbf{C}((\mathbf{P}^{\text{op}})). \quad (11)$$

16.4. Arrow construction

Definition 16.8 (Arrow category)

Given any category \mathbf{C} , its *arrow category* $\mathbf{Arr} \mathbf{C}$ is:

1. *Objects*: morphisms of \mathbf{C} .
2. *Morphisms*: Given objects $f : X \rightarrow_{\mathbf{C}} Y$ and $g : Z \rightarrow_{\mathbf{C}} U$, a morphism $\varphi : f \rightarrow_{\mathbf{Arr} \mathbf{C}} g$ in $\mathbf{Arr} \mathbf{C}$ is a pair of morphisms $\langle \varphi_s, \varphi_t \rangle$ in \mathbf{C} that make the following diagram

$$\begin{array}{ccc} X & \xrightarrow{\varphi_s} & Z \\ f \downarrow & & \downarrow g \\ Y & \xrightarrow{\varphi_t} & U \end{array} \quad (12)$$

a commutative square in \mathbf{C} .

3. *Composition*: Composition in $\mathbf{Arr} \mathbf{C}$ is given by placing commutative squares side by side. Consider $\langle \varphi_s, \varphi_t \rangle : f \rightarrow g$ and $\langle \psi_s, \psi_t \rangle : g \rightarrow h$ in $\mathbf{Arr} \mathbf{C}$ giving rise to the following composite commutative diagram

$$\begin{array}{ccccc} X & \xrightarrow{\varphi_s} & Z & \xrightarrow{\psi_s} & V \\ f \downarrow & & \downarrow g & & \downarrow h \\ Y & \xrightarrow{\varphi_t} & U & \xrightarrow{\psi_t} & W \end{array} \quad (13)$$

Since this diagram is again commutative, we define

$$\langle \varphi_s, \varphi_t \rangle ;_{\mathbf{Arr} \mathbf{C}} \langle \psi_s, \psi_t \rangle := \langle \varphi_s ;_{\mathbf{C}} \psi_s, \varphi_t ;_{\mathbf{C}} \psi_t \rangle. \quad (14)$$

4. *Identities*: given an object $f : X \rightarrow Y$ of $\mathbf{Arr} \mathbf{C}$, its identity morphism is $\langle \text{id}_X, \text{id}_Y \rangle$.

Example 16.9 (Intervals). Consider a poset \mathbf{P} . The arrow category $\mathbf{Arr}(\mathbf{C}(\mathbf{P}))$ is isomorphic to the poset (viewed as a category) $\mathbf{C}((\mathbf{Arr} \mathbf{P}))$ of nonempty intervals in \mathbf{P} :

$$\mathbf{Arr}(\mathbf{C}(\mathbf{P})) \simeq \mathbf{C}((\mathbf{Arr} \mathbf{P})). \quad (15)$$

16.5. Twisted arrow construction

Definition 16.10 (Twisted arrow category)

Given a category \mathbf{C} , its *twisted arrow category* $\mathbf{Tw} \mathbf{C}$ is:

1. *Objects*: morphisms in \mathbf{C} .
2. *Morphisms*: A morphism in $\mathbf{Tw} \mathbf{C}$ from $f : X \rightarrow_{\mathbf{C}} Y$ to $g : Z \rightarrow_{\mathbf{C}} U$ is given by a pair of morphisms (φ_s, φ_t) in \mathbf{C} such that the following diagram commutes:

$$\begin{array}{ccc} X & \xleftarrow{\varphi_s} & Z \\ f \downarrow & & \downarrow g \\ Y & \xrightarrow{\varphi_t} & U \end{array} \quad (16)$$

3. *Composition*: Composition in $\mathbf{Tw} \mathbf{C}$ is given by placing commutative squares side by side. Consider $(\varphi_s, \varphi_t) : f \rightarrow g$ and $(\psi_s, \psi_t) : g \rightarrow h$ in $\mathbf{Tw} \mathbf{C}$, giving rise to the following composite diagram

$$\begin{array}{ccccc} X & \xleftarrow{\varphi_s} & Z & \xleftarrow{\psi_s} & V \\ f \downarrow & & \downarrow g & & \downarrow h \\ Y & \xrightarrow{\varphi_t} & U & \xrightarrow{\psi_t} & W \end{array} \quad (17)$$

It is commutative because its component squares are, and hence it defines a morphism $f \rightarrow h$ in $\mathbf{Tw} \mathbf{C}$, namely

$$\langle \varphi_s, \varphi_t \rangle :_{\mathbf{Tw} \mathbf{C}} \langle \psi_s, \psi_t \rangle := \langle \psi_s \circ_{\mathbf{C}} \varphi_s, \varphi_t \circ_{\mathbf{C}} \psi_t \rangle. \quad (18)$$

4. *Identitites*: given an object $f : X \rightarrow Y$ of $\mathbf{Tw} \mathbf{C}$, its identity morphism is $\langle \text{id}_X, \text{id}_Y \rangle$.

Remark 16.11. The above construction might be more precisely called the “source twisted arrow category” of \mathbf{C} , because it is a modification of the arrow construction where we are “twisting” the arrow between source objects of morphisms of \mathbf{C} by reversing its direction. An analogous construction exists where instead we twist the arrow construction by reversing the arrow between targets of morphisms of \mathbf{C} . This latter construction might be called the “target twisted arrow category”. For brevity, we have only spelled out the source twisted variant here.

Graded exercise E.14 (TwistedCat)

Let \mathbf{C} be a category, and let $\mathbf{Tw} \mathbf{C}$ be the associated twisted arrow category. Check that the definition of $\mathbf{Tw} \mathbf{C}$ does indeed define a category. Specifically, check that for $\mathbf{Tw} \mathbf{C}$

1. composition of composable morphism does again define a morphism of $\mathbf{Tw} \mathbf{C}$;
2. composition is associative;
3. identity morphisms satisfy the identity laws (that they behave neutrally for composition).

Example 16.12 (Twisted construction in posets and categories). Consider a poset \mathbf{P} . The twisted arrow category $\mathbf{Tw} \mathbf{C}(\mathbf{P})$ is isomorphic to the poset (viewed as a category) $\mathbf{C}((\mathbf{Tw} \mathbf{P}))$ of nonempty intervals in \mathbf{P} :

$$\mathbf{Tw} (\mathbf{C}(\mathbf{P})) \simeq \mathbf{C}((\mathbf{Tw} \mathbf{P})). \quad (19)$$

Exercise39. Prove the statement in Example 16.12. Recall that, given elements $x, y \in P$, the interval $[x, y]$ is

$$[x, y] := \{z \in P \mid x \leq_P z \leq_P y\}. \quad (20)$$

Start to show that the partial order is equivalent to a twisted morphism.

See solution on page 250.

16.6. (Co)slice construction

Definition 16.13 (Slice categories)

Let \mathbf{C} be a category and fix an object $T \in \text{Ob}_{\mathbf{C}}$. The *slice category* \mathbf{C}/T of \mathbf{C} over T is:

1. *Objects*: morphisms in \mathbf{C} of the type $X \rightarrow T$, where X ranges over all objects of \mathbf{C} .

2. *Morphisms*: given objects $X \xrightarrow{f} T$ and $Y \xrightarrow{g} T$, a morphism

$$\varphi_{\mathbf{C}/T} : (X \xrightarrow{f} T) \rightarrow_{\mathbf{C}/T} (Y \xrightarrow{g} T) \quad (21)$$

is specified by a morphism $\varphi : X \rightarrow_{\mathbf{C}} Y$ such that the diagram

$$\begin{array}{ccc} X & \xrightarrow{\varphi} & Y \\ & \searrow f & \swarrow g \\ & T & \end{array} \quad (22)$$

commutes.

3. *Composition*: defined via the composition in \mathbf{C} . Concretely, given composable morphisms $\varphi_{\mathbf{C}/T}$ and $\psi_{\mathbf{C}/T}$ of \mathbf{C}/T , we define

$$\varphi_{\mathbf{C}/T} ;_{\mathbf{C}/T} \psi_{\mathbf{C}/T} := (\varphi ;_{\mathbf{C}} \psi)_{\mathbf{C}/T}. \quad (23)$$

4. *Identities*: defined by the identities in \mathbf{C} .

Graded exercise E.15 (*SliceCat*)

Let \mathbf{C} be a category, fix $T \in \text{Ob}_{\mathbf{C}}$, and consider the slice category \mathbf{C}/T . Your task is to check that the composition of two composable morphisms in \mathbf{C}/T is again in fact a morphism in \mathbf{C}/T .

Definition 16.14 (Coslice categories)

Let \mathbf{C} be a category and fix an object $S \in \text{Ob}_{\mathbf{C}}$. The *coslice category* S/\mathbf{C} of \mathbf{C} under S is:

1. *Objects*: morphisms in \mathbf{C} of the type $S \rightarrow X$, where X ranges over all objects of \mathbf{C} .

2. *Morphisms*: given objects $S \xrightarrow{f} X$ and $S \xrightarrow{g} Y$, a morphism

$$\varphi_{S/\mathbf{C}} : (S \xrightarrow{f} X) \rightarrow_{S/\mathbf{C}} (S \xrightarrow{g} Y) \quad (24)$$

is specified by a morphism $\varphi : X \rightarrow_{\mathbf{C}} Y$ such that the diagram

$$\begin{array}{ccc} X & \xrightarrow{\varphi} & Y \\ & \swarrow f & \searrow g \\ & S & \end{array} \quad (25)$$

commutes.

3. *Composition*: defined by the composition in \mathbf{C} .

4. *Identities*: defined by the identities in \mathbf{C} .



17. Culture

Culture is what you don't think about. Mathematics has a quite different culture than engineering. Engineering is about getting things to work *in practice*. Conventions are important. Specifics of protocols are important. In mathematics, certain things just not matter: one studies what is true regardless of conventions.

17.1 Definition vs computation	244
17.2 Things that don't matter	245
17.3 Choice of symbols	246
17.4 Typographical conventions	247

Schwingen is the traditional Swiss wrestling native to the pre-alpine parts of German-speaking Switzerland. Wrestlers wear *Schwingerhosen* that can be used to hold and grapple the adversary.

17.1. Definitional impetus vs computational goals

The category **Curr** represents the set of all possible currency exchangers that could ever exist. However, in this set there would be very irrational agents. For example, there is a currency exchanger that, given 1 USD, will give you back 2 USD; there is one currency exchanger that corresponds to converting USD to CHF back and forth 21 times before getting you the money. There is even one that will not give you back any money.

Moreover, using the composition operations we could produce many more morphisms. In fact, if there are loops, we could traverse the loops multiple times, and, depending on the numbers, finding new morphisms, possibly infinitely many more.

This highlights a recurring topic: often mathematicians will be happy to define a broader category of objects, while, in practice, the engineer will find herself thinking about a more constrained set of objects. In particular, while the mathematician is more concerned with defining categories as hypothetical universes of things, the engineer is typically interested in representing concrete things, and solve some computational problem on the represented structure.

For example, in the case of the currency exchangers, the problem might be that of finding the sequence of the best conversions between a source and a target currency.

First, the engineer would add more constraints to the definition to work with more well-behaved objects. For example, it is reasonable to limit the universe of morphisms in such a way that the action of converting back and forth the same currency to have a cost (through the commission) higher than 0.

In that case, we will find that the optimal paths of currencies never pass through a currency more than once. To see this, consider three currencies **A**, **B**, **C**, a currency exchanger $\langle a, b \rangle$ from **A** to **B**, a currency exchanger $\langle c, d \rangle$ from **B** to **C**, and a currency exchanger $\langle e, f \rangle$ from **C** to **A**. The composition of the currency exchangers reads:

$$\langle eca, ecb + ed + f \rangle = \langle g, h \rangle. \quad (1)$$

Assuming $e = a^{-1}$ (in words, an exchange rate direction is not more profitable than the other), and $h \neq 0$, because of the commissions we can show that there are multiple morphisms from **A** to **A**, and that the identity morphism is the most “convenient” one. If we only pass through each currency at most once, there are only a finite amount of paths to check, and this might simplify the computational problem.

Second, the engineer might be interested in keeping track only of the “dominant” currency exchangers. For example, if we have two exchangers with the same rate but different commission, we might want to keep track only of the one with the lowest commission.

In the next chapters we will see that there are concepts that will be useful to model these situations:

- ▷ There is a concept of *subcategory* that allows to define more specific categories of a parent one, in a way that still satisfies the axioms.
- ▷ There is a concept called *locally posetal* categories, in which the set of morphisms between two objects is assumed to be a *poset* rather than a *set*, that is, we assume that there is an order, and that this order will be compatible with the operation of composition.

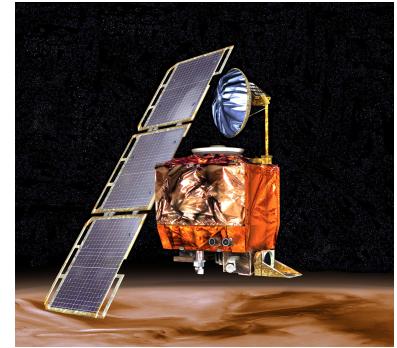
17.2. Things that don't matter

In engineering we know that **using the right conventions is essential**.

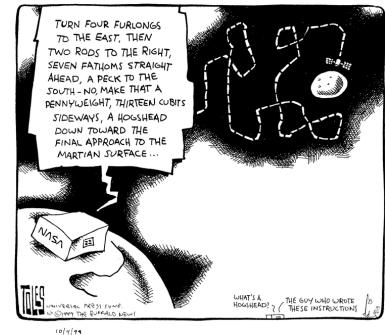
There are many famous examples of unit mismatches causing disasters or near-disasters:

- ▷ The loss of the Mars Climate Orbiter in 1999 was due to the fact that NASA used the metric system, while contractor Lockheed Martin used (by mistake) imperial units (Fig. 1).
- ▷ In 1983, an Air Canada's Boeing 767 jet ran out of fuel in mid-flight because there was a miscalculation of the fuel needed for the trip. In the end, the pilot managed to successfully land the “Gimli Glider”.
- ▷ Going back in history, Columbus wound up in the Bahamas because he miscalculated the Earth’s circumference, due to several mistakes, and one of them was assuming that his sources were using the *Roman mile* rather than the *Arabic mile* [20]. Columbus’ mathematical mistakes led to a happy incident for him, but not so great outcomes for many others.

However, in category theory, we look at the “essence” of things, and we consider **what is true regardless of conventions**.



(a) Mars climate orbiter



(b)

Figure 1.

17.3. The choice of symbols does not matter

For example, it doesn't matter what symbols we use to represent numerals. It also

•	۱	۲	۳	۴	۵	۶	۷	۸	۹
0	1	2	3	4	5	6	7	8	9
sifr	eeth-nayn	arba'a	khamsa	sitta	thamaaneeya	sab'a	tis'a		

should not matter that we use base-10 numerals—certainly mathematical truths do not depend on how many fingers humans have.

Just like this book is written in rather plain English, and could be translated to another language while preserving the meaning, in category theory we look at what is not changed by a 1:1 translation that can be reversed.

This will be covered later in a section on “isomorphisms”; but for now we can look at this intuitively.



Figure 2.: A page from Muhammad ibn Mūsā al-Khwārizmī's *Algebra*. The word *algorithm* comes from the name *al-Khwārizmī*.

17.4. Typographical conventions don't matter

Some of you might have objected to the conventions that we used in this chapter for the notation for composition of morphisms. We have used the notation $f ; g$ (“ f then g ”) while usually in the rest of mathematics we would have used $g \circ f$ (“ g after f ”). However, any concept we will use is “invariant” to the choice of notation. We can decide to rewrite the book using the other convention and still all the theorems would remain true, and all the falsities will remain false. More technically, we can take any formula written in one convention and rewrite it with the other convention, and vice versa using a specific mechanical rule. For example, the formula

$$(f ; g) ; h = f ; (g ; h) \quad (2)$$

would be transformed in

$$h \circ (g \circ f) = (h \circ g) \circ f. \quad (3)$$

(A bit more advanced category theory can describe this transformation more precisely.)

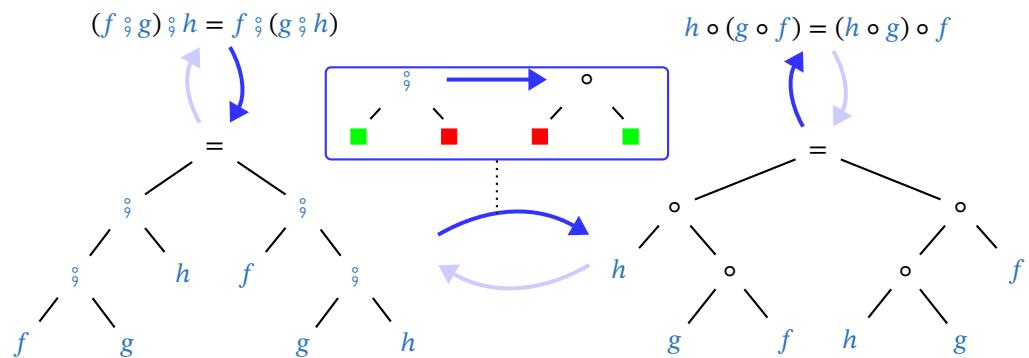


Figure 3.: Mechanical rule to transform one convention to another.

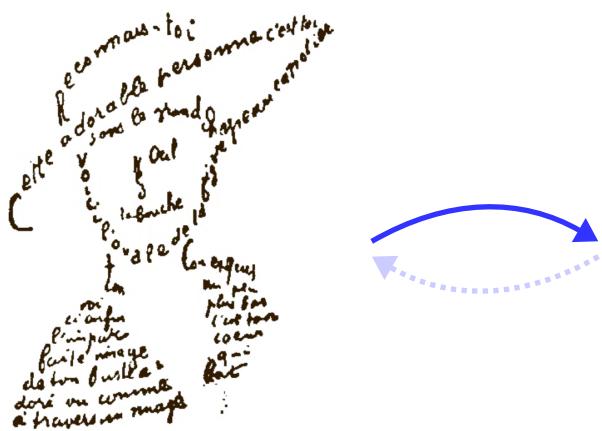
This invariance to mechanical invertible transformations only holds for mathematical and technical writing. In other contexts it might fail.

For example, “Veni Vidi Vici” has a sound to it that the English translation does not have (Fig. 4).

VENI VIDI VICI  I came; I saw; I conquered.

Figure 4.

Sometimes, the meaning is in the typography, as in Apollinaire's poem “Poème à Lou” (Fig. 5).



Reconnais-toi
 Cette adorable personne c'est toi
 Sous le grand chapeau canotier
 Oeil
 Nez
 La bouche
 Voici l'ovale de ta figure
 Ton cou exquis
 Voici enfin l'imparfaite image de ton buste adoré
 vu comme à travers un nuage
 Un peu plus bas c'est ton coeur qui bat

Apollinaire - 1915 - Poème à Lou

Figure 5.

Solutions to selected exercises

Solution of Exercise 31. We know:

- ▷ $\text{src}(1) = a$ and $\text{tgt}(1) = b$: we have $F_*(a) = \alpha$ and $F_*(b) = \beta$, meaning that $F_*(1) = \text{I}$.
- ▷ $\text{src}(2) = b$ and $\text{tgt}(2) = c$: we have $F_*(b) = \beta$ and $F_*(c) = \gamma$, meaning that $F_*(2) = \text{II}$.
- ▷ $\text{src}(3) = c$ and $\text{tgt}(3) = d$: we have $F_*(c) = \gamma$ and $F_*(d) = \alpha$, meaning that $F_*(3) = \text{III}$.
- ▷ $\text{src}(4) = d$ and $\text{tgt}(4) = e$: we have $F_*(d) = \alpha$ and $F_*(e) = \gamma$, meaning that $F_*(4) = \text{IV}$.
- ▷ $\text{src}(5) = e$ and $\text{tgt}(5) = a$: we have $F_*(e) = \gamma$ and $F_*(a) = \alpha$, meaning that $F_*(5) = \text{III}$.

Therefore, the map F_* is as reported in Fig. 6.

Solution of Exercise 32. We define the category **InjSet** to be such that its objects are all sets, its morphisms are injective functions, composition is the usual composition of functions, and identity morphisms are the usual identity functions.

1. We show that the composition of two injective functions is injective. Given functions $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ injective, suppose that $g(f(x_1)) = g(f(x_2))$ for some $x_1, x_2 \in X$. By the injectivity of g it follows that $f(x_1) = f(x_2)$, and then using the injectivity of f , we can conclude that $x_1 = x_2$.
2. Identity functions are clearly injective.
3. Associativity of composition holds because it holds for all functions, so in particular also for injective ones.

Solution of Exercise 33.

Solution of Exercise 34.

Solution of Exercise 35.

Solution of Exercise 36. The concept of this exercise is very similar to the one of **Curr**. In general, we can write a temperature converter (morphism) from e to f as a pair of numbers $\langle a, d \rangle$, $a, d \in \mathbb{R}$. For each morphism we have a map which actually transform an amount of the first temperature into an amount of the second temperature:

$$\begin{aligned} f_{\langle a, d \rangle} : \mathbb{R} &\rightarrow \mathbb{R} \\ x &\mapsto ax + d. \end{aligned} \tag{4}$$

Now, all the possible conversions between the three temperature conventions feature specific values for a, d , listed in Table 17.1 (rows are to be intended as the source, and columns as the target convention).

Table 17.1.: Temperature conversion factors.

	Celsius	Kelvin	Fahrenheit
Celsius	$a = 1, d = 0$	$a = 1, d = 273$	$a = 9/5, d = 32$
Kelvin	$a = 1, d = -273$	$a = 1, d = 0$	$a = 9/5, d = -459.4$
Fahrenheit	$a = 5/9, d = -17.7$	$a = 5/9, d = 255.2$	$a = 1, d = 0$

We now define the category **Temp** as being constituted of:

- ▷ *Objects*: $\text{Ob}_{\text{Temp}} = \{\text{Celsius}, \text{Kelvin}, \text{Fahrenheit}\}$;
- ▷ *Morphisms*: There is a single morphism from e to f for any $e, f \in \text{Ob}_{\text{Temp}}$, given by $\langle a, d \rangle$, with a, d as in Table 17.1;

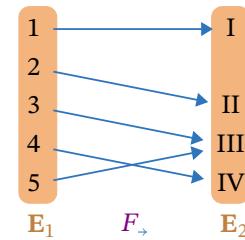


Figure 6.

▷ *Composition of morphisms:* Analogous to the one in **Curr**;

▷ *Identity morphisms:* Analogous to the ones in **Curr**.

This indeed forms a category, as we have shown for **Curr**.

Solution of Exercise 37. The only nontrivial step is checking that associativity holds for composition of **dur**. Consider 3 compatible morphisms f_1, f_2, f_3 . We know from (55) that

$$\text{dur}_{1;2}(n) = \text{dur}_1(n) + \text{dur}_2(\sigma_1(n)). \quad (5)$$

Now we compose $f_{1;2}$ with f_3 . For **dur** we obtain

$$\text{dur}_{(1;2);3}(n) = \text{dur}_{1;2}(n) + \text{dur}_3(\sigma_{1;2}(n)) \quad (6)$$

$$= \text{dur}_1(n) + \text{dur}_2(\sigma_1(n)) + \text{dur}_3(\sigma_2(\sigma_1(n))) \quad (7)$$

Instead, if we first compute $f_2 \circ f_3$, we have

$$\text{dur}_{2;3}(n) = \text{dur}_2(n) + \text{dur}_3(\sigma_2(n)). \quad (8)$$

If we now compose f_1 with $f_2 \circ f_3$ we get:

$$\text{dur}_{1;(2;3)}(n) = \text{dur}_1(n) + \text{dur}_{2;3}(\sigma_1(n)) \quad (9)$$

$$= \text{dur}_1(n) + \text{dur}_2(n) + \text{dur}_3(\sigma_2(\sigma_1(n))), \quad (10)$$

which is the same as (7).

Solution of Exercise 38. We check the two conditions. First, consider a morphism $f : X \rightarrow Y \in \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle X, i \rangle; \langle Y, i \rangle)$ (the index i is repeated, because following the definition of morphisms, no morphism connects objects of one category to objects of the other one). We have

$$\text{id}_{\mathbf{C}+\mathbf{D}} \circ f = \begin{cases} \text{id}_C \circ f = f & \text{if } i = 1, \\ \text{id}_D \circ f = f & \text{if } i = 2, \end{cases} \quad (11)$$

and

$$f \circ \text{id}_{\mathbf{C}+\mathbf{D}} = \begin{cases} f \circ \text{id}_C = f & \text{if } i = 1, \\ f \circ \text{id}_D = f & \text{if } i = 2. \end{cases} \quad (12)$$

Second, consider the morphisms $f : X \rightarrow Y \in \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle X, i \rangle; \langle Y, i \rangle)$, $g : Y \rightarrow Z \in \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle X, j \rangle; \langle Y, j \rangle)$, and $h : Z \rightarrow U \in \text{Hom}_{\mathbf{C}+\mathbf{D}}(\langle X, k \rangle; \langle Y, k \rangle)$. We have

$$(f \circ_{\mathbf{C}+\mathbf{D}} g) \circ_{\mathbf{C}+\mathbf{D}} h := \begin{cases} (f \circ_C g) \circ_C h = f \circ_C g \circ_C h & \text{if } i = j = k = 1, \\ (f \circ_D g) \circ_D h = f \circ_D g \circ_D h & \text{if } i = j = k = 2, \\ \text{does not exist} & \text{else.} \end{cases} \quad (13)$$

and

$$f \circ_{\mathbf{C}+\mathbf{D}} (g \circ_{\mathbf{C}+\mathbf{D}} h) := \begin{cases} f \circ_C (g \circ_C h) = f \circ_C g \circ_C h & \text{if } i = j = k = 1, \\ f \circ_D (g \circ_D h) = f \circ_D g \circ_D h & \text{if } i = j = k = 2, \\ \text{does not exist} & \text{else.} \end{cases} \quad (14)$$

Solution of Exercise 39. Consider $\mathbf{C}((\text{Tw P}))$. Take two morphisms in \mathbf{P} : $f : X \rightarrow Y$ (from $X \leq Y$) and $g : Z \rightarrow U$ (from $Z \leq U$). These are two objects in $\mathbf{C}((\text{Tw P}))$. Now, a morphism in $\mathbf{C}((\text{Tw P}))$ is a pair $\langle h, i \rangle$ where $h : Z \rightarrow X$ (from $Z \leq X$) and $i : Y \rightarrow U$ (from $Y \leq U$). Therefore, we have $Z \leq X \leq Y \leq U$, which corresponds to $[X, Y] \leq_{\text{Tw P}} [Z, U]$. Therefore, morphisms in $\mathbf{C}((\text{Tw P}))$ between arrows (intervals) correspond to order relations between intervals.

PART F. FUNCTORS



18. (Semi)Category actions	253
19. Translation	277
20. Specialization	289
21. Syntax and semantics	295
22. Up the ladder of abstraction	299

The “Stoosbahn”, also known as “Schwyz-Stoos” funicular, is a Swiss funicular railway. On a length of 1.7 kilometers, it climbs a height difference of 744 metres: it is the steepest funicular railway in Europe, second in the world (surpassed by the “Katoomba Scenic Railway” in Australia).



18. (Semi)Category actions

Categories are well suited to represent processes: both physical processes that move and transform matter, and software processes that communicate and compute.

18.1 Moore machines, first version	254
18.2 The category $\langle \text{Set} \rangle$	258
18.3 Moore machines, $\langle \text{Set} \rangle$ version	260
18.4 Standard action of Moore machines	263
18.5 Semicategory actions	266
18.6 More machines	268
18.7 LTI systems	270

18.1. Moore machines, first version

In the following we consider **Moore machines***, which are a type of basic model for describing certain dynamical systems. Its characteristic features are:

- ▷ a **state space** which describes all possible states that the system can possibly be in;
- ▷ an **input space** and an **output space**;
- ▷ a **dynamics** which describes how, given an input, the system's state changes according to that input;
- ▷ a **read-out** which relates the current state of the system to the output space.

We are in particular interested in the ways that such a machine can transform a sequence of inputs into a sequence of outputs.

A first mathematical model

Our first version for formalizing the idea of a Moore machine looks as follows. We model the input space, state space, and output space as sets \mathbf{U} , \mathbf{X} , and \mathbf{Y} respectively, and we model the dynamics and read-out as functions

$$\begin{cases} \mathbf{dyn} : \mathbf{U} \times \mathbf{X} \rightarrow \mathbf{X}, \\ \mathbf{ro} : \mathbf{X} \rightarrow \mathbf{Y}. \end{cases} \quad (1)$$

We will also choose an element $\mathbf{st} \in \mathbf{X}$ of the state space as an initial state; we will use this when specifying how a Moore machine acts on sequences of inputs.

Thus, in total, a Moore machine is specified by a tuple of the following type:

$$\langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \mathbf{dyn}, \mathbf{ro}, \mathbf{st} \rangle. \quad (2)$$

Example 18.1 (Robot on a grid). We consider a robot moving on a two-dimensional unit grid, represented as $\mathbb{N}_{[0,100]}^2$ (Fig. 1).

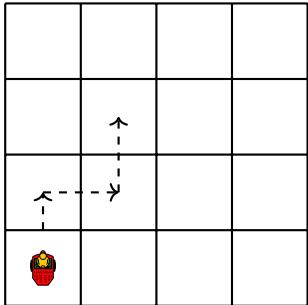


Figure 1.: Robot on a grid example.

The state of the robot is described by its position coordinates, as $x \in \mathbb{N}_{[0,100]}^2$ (for convenience, consider the first element of the state to be the horizontal coordinate and the second element to be the vertical one). The robot can choose from a set of input actions $\mathbf{U} = \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$, each representing a movement of 1 unit in the corresponding direction on the grid. We can write the dynamics as

$$\begin{aligned} \mathbf{dyn} : \mathbf{U} \times \mathbf{X} &\rightarrow \mathbf{X} \\ (\uparrow, (x_1, x_2)) &\mapsto (x_1, \min(x_2 + 1, 100)), \\ (\downarrow, (x_1, x_2)) &\mapsto (x_1, \max(x_2 - 1, 0)), \\ (\rightarrow, (x_1, x_2)) &\mapsto (\min(x_1 + 1, 100), x_2), \\ (\leftarrow, (x_1, x_2)) &\mapsto (\max(x_1 - 1, 0), x_2), \end{aligned} \quad (3)$$

where the “min” and “max” ensure that the robot stays in within the boundaries of the grid. The readout can be thought of as a sensor measuring the position of the robot (the state). Assuming a perfect sensor, we have:

$$\begin{aligned} \mathbf{ro} : \mathbf{X} &\rightarrow \mathbf{Y}, \\ (x_1, x_2) &\mapsto (x_1, x_2). \end{aligned}$$

The starting position of the robot can be specified as any $\mathbf{st} \in \mathbb{N}_{[0,100]}^2$.

* They are named after Edward F. Moore.

Moore machines acting on sequences

Given a Moore machine $f = \langle \mathbf{U}, \mathbf{Y}, \mathbf{X}, \text{dyn}, \text{ro}, \text{st} \rangle$, the following is a standard way to think of it as acting on a sequence of inputs, transforming it into a sequence of outputs.

Given an infinite sequence of inputs u_0, u_1, u_2, \dots , we use the following recipe

$$\begin{cases} x_{k+1} = \text{dyn}(u_k, x_k) \\ y_k = \text{ro}(x_k), \end{cases} \quad (4)$$

to produce an infinite sequence y_0, y_1, y_2, \dots of outputs. For the very first step, when $k = 0$, we need the initial state $x_0 = \text{st}$ in order to compute $x_1 = \text{dyn}(u_0, x_0)$ and $y_0 = \text{ro}(x_0)$.

Example 18.2. To continue Example 18.1, we can consider a robot with starting position $\langle 0, 0 \rangle$. We consider a sequence of inputs $\rightarrow, \uparrow, \rightarrow, \downarrow, \leftarrow$. Using the recipe given by the Moore machine in Example 18.1, one can find the sequences of states and outputs, which in this case coincide, given the identity readout:

$$\begin{aligned} x_0 &= \text{st} = y_0 = \langle 0, 0 \rangle \\ x_1 &= y_1 = \langle 1, 0 \rangle, \\ x_2 &= y_2 = \langle 1, 1 \rangle, \\ x_3 &= y_3 = \langle 2, 1 \rangle, \\ x_4 &= y_4 = \langle 2, 0 \rangle, \\ x_5 &= y_5 = \langle 1, 0 \rangle, \end{aligned}$$

An infinite sequence u_0, u_1, u_2, \dots of elements of \mathbf{U} can be also thought as a function $u : \mathbb{N} \rightarrow \mathbf{U}$ with $u(0) = u_0, u(1) = u_1, u(2) = u_2, \dots$ etc. To denote an infinite sequence of elements of \mathbf{U} we use the two notations $\mathbf{U}^{\mathbb{N}}$ and **Stream** \mathbf{U} :

$$\text{Stream } \mathbf{A} := (\mathbb{N} \rightarrow \mathbf{A}) = \mathbf{A}^{\mathbb{N}}. \quad (5)$$

For a fixed Moore machine f , the recipe (4) defines a function act_f which maps any given sequence u of elements of \mathbf{U} to a corresponding sequence $y = \text{act}_f(u)$ of elements of \mathbf{Y} . In other words, from $\langle \mathbf{U}, \mathbf{Y}, \mathbf{X}, \text{dyn}, \text{ro}, \text{st} \rangle$ and (4) we obtain

$$\text{act}_f : \text{Stream } \mathbf{U} \rightarrow \text{Stream } \mathbf{Y}. \quad (6)$$

We think of this function act_f as describing the external behavior of the Moore machine f , because it encodes what is externally observable in terms of how the Moore machine is used to relate inputs to outputs.

Composing Moore machines

Let us consider composing Moore machines serially by letting the output of one machine be the input of the next. We'd like the result to again be a Moore machine.

In other words, given Moore machines

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{st}_f \rangle \quad (7)$$

and

$$g = \langle \mathbf{U}_g, \mathbf{X}_g, \mathbf{Y}_g, \text{dyn}_g, \text{ro}_g, \text{st}_g \rangle, \quad (8)$$

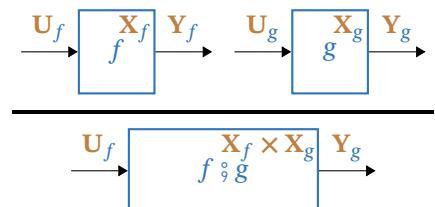


Figure 2.: Composition of Moore machines (first version).

with $\mathbf{Y}_f = \mathbf{U}_g$, we'd like to define their composition as a Moore machine

$$f ; g = \langle \mathbf{U}_{f ; g}, \mathbf{X}_{f ; g}, \mathbf{Y}_{f ; g}, \text{dyn}_{f ; g}, \text{ro}_{f ; g}, \text{st}_{f ; g} \rangle. \quad (9)$$

The situation is illustrated in Fig. 2.

Here is one way to do it: we set

$$\begin{aligned} \mathbf{U}_{f ; g} &:= \mathbf{U}_f, \\ \mathbf{X}_{f ; g} &:= \mathbf{X}_f \times \mathbf{X}_g, \\ \mathbf{Y}_{f ; g} &:= \mathbf{Y}_g, \\ \text{st}_{f ; g} &:= \langle \text{st}_f, \text{st}_g \rangle, \end{aligned} \quad (10)$$

we define the composite dynamics to be

$$\begin{aligned} \text{dyn}_{f ; g} : \mathbf{U}_f \times (\mathbf{X}_f \times \mathbf{X}_g) &\rightarrow (\mathbf{X}_f \times \mathbf{X}_g), \\ \langle u, \langle x_f, x_g \rangle \rangle &\mapsto \langle \text{dyn}_f(u, x_f), \text{dyn}_g(\text{ro}_f(x_f), x_g) \rangle, \end{aligned} \quad (11)$$

and we define the composite readout to be

$$\begin{aligned} \text{ro}_{f ; g} : (\mathbf{X}_f \times \mathbf{X}_g) &\rightarrow \mathbf{Y}_g, \\ \langle x_f, x_g \rangle &\mapsto \text{ro}_g(x_g). \end{aligned} \quad (12)$$

This formalization works very nicely – it models composition using the output of one machine as the input of the next, and result of composition is again a Moore machine.

However, there is one aspect which we wish were different: this composition operation is not associative. It *almost* is, but not quite. We explain why below.

The reason we wish it *were* associative is that Moore machines would then form a semicategory, and we could integrate them nicely into the mathematics we have been developing thus far.

The culprit for associativity failing is the cartesian product that we use to define the state space $\mathbf{X}_f \times \mathbf{X}_g$ of a composite Moore machine $f ; g$. Consider three composable systems f , g , and h . If we compose them in the two ways $(f ; g) ; h$ and $f ; (g ; h)$, then their respective state spaces are $(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h$ and $\mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h)$. These sets are *isomorphic*, but they are not equal on the nose, and hence also the Moore machines $(f ; g) ; h$ and $f ; (g ; h)$ are very close to being equal, but are not quite.

To see why

$$(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h \neq \mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h), \quad (13)$$

recall that the elements of $(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h$ are nested tuples of the form

$$\langle \langle x_f, x_g \rangle, x_h \rangle \quad (14)$$

while the elements of $\mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h)$ are nested tuples of the form

$$\langle x_f, \langle x_g, x_h \rangle \rangle. \quad (15)$$

An isomorphism between the two sets is given by the following function

$$\begin{aligned} (\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h &\rightarrow \mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h) \\ \langle \langle x_f, x_g \rangle, x_h \rangle &\mapsto \langle x_f, \langle x_g, x_h \rangle \rangle \end{aligned} \quad (16)$$

which simply re-brackets the tuples.

In the next section we will introduce a technical construction for defining a product similar to the cartesian product, but which is associative on the nose - not just “up to an isomorphism”. This construction will allow us to make a new, modified formalization of Moore machines which form a bona fide semicategory, and it will prove useful in other respects further down the road.

18.2. The category $\langle \text{Set} \rangle$

We will define a category $\langle \text{Set} \rangle$ whose objects are “sets of tuples”.

Given sets $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$, where $n \in \mathbb{N}$, we define

$$\langle \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n \rangle := \{\langle x_1, x_2, \dots, x_n \rangle \mid x_1 \in \mathbf{A}_1, x_2 \in \mathbf{A}_2, \dots, x_n \in \mathbf{A}_n\}. \quad (17)$$

So for example,

$$\langle \mathbb{Z}, \mathbb{N}, \mathbb{R} \rangle = \{\langle x, y, z \rangle \mid x \in \mathbb{Z}, y \in \mathbb{N}, z \in \mathbb{R}\}. \quad (18)$$

Note the case $n = 0$, where

$$\langle \rangle = \{\langle \rangle\} \quad (19)$$

is a singleton set whose element is the empty tuple.

All the objects of $\langle \text{Set} \rangle$ are in particular just sets, so we can package them into a category whose morphisms are simply functions between such sets.

Definition 18.3 ($\langle \text{Set} \rangle$)

The category $\langle \text{Set} \rangle$ is the category whose objects are those sets which are of the form $\langle \mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n \rangle$, for any $n \in \mathbb{N}$ and any sets $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$. Morphisms in $\langle \text{Set} \rangle$ are any functions between such sets, and the composition operations and identities are the usual ones for functions.

From Set to $\langle \text{Set} \rangle$

Observe that we can turn any set into an object of $\langle \text{Set} \rangle$: given a set \mathbf{A} , there is the associated set

$$\langle \mathbf{A} \rangle = \{\langle x \rangle \mid x \in \mathbf{A}\} \quad (20)$$

whose elements are precisely all the tuples of length one whose entry is an element of \mathbf{A} . This defines a function

$$\begin{aligned} \text{Ob}_{\text{Set}} &\rightarrow \text{Ob}_{\langle \text{Set} \rangle} \\ \mathbf{A} &\mapsto \langle \mathbf{A} \rangle \end{aligned} \quad (21)$$

which one might call “bracket”.

Similarly, given a function $f : \mathbf{A} \rightarrow \mathbf{B}$ between sets, there is an associated function $\langle f \rangle : \langle \mathbf{A} \rangle \rightarrow \langle \mathbf{B} \rangle$ given by

$$\langle f \rangle(\langle x \rangle) := \langle f(x) \rangle. \quad (22)$$

This defines a function

$$\text{Hom}_{\text{Set}}(\mathbf{A}, \mathbf{B}) \rightarrow \text{Hom}_{\langle \text{Set} \rangle}(\langle \mathbf{A} \rangle, \langle \mathbf{B} \rangle) \quad (23)$$

for any sets \mathbf{A} and \mathbf{B} ; one might also call it “bracket”.

Concatenation of tuples

Given tuples $\langle x, y, z \rangle$ and $\langle u, v \rangle$ we can stick them together to make the longer tuple $\langle x, y, z, u, v \rangle$, using the already defined concatenation of tuples.

Multiplication in $\langle \text{Set} \rangle$

We define a multiplication for objects of $\langle \text{Set} \rangle$. The symbol \circledast will denote this multiplication in infix notation.

Given $\langle A_1, \dots, A_m \rangle$ and $\langle B_1, \dots, B_n \rangle$ we define the operation

$$\circledast : \text{Ob}_{\langle \text{Set} \rangle} \times \text{Ob}_{\langle \text{Set} \rangle} \rightarrow \text{Ob}_{\langle \text{Set} \rangle} \quad (24)$$

by

$$\langle A_1, \dots, A_m \rangle \circledast \langle B_1, \dots, B_n \rangle := \langle A_1, \dots, A_m, B_1, \dots, B_n \rangle \quad (25)$$

for any $n, m \in \mathbb{N}$.

The elements of $\langle A_1, \dots, A_m \rangle \circledast \langle B_1, \dots, B_n \rangle$ are all possible concatenations of elements of $\langle A_1, \dots, A_m \rangle$ with elements of $\langle B_1, \dots, B_n \rangle$.

Equation (25) holds in particular also in the cases when $m = 0$ or $n = 0$:

$$\langle \rangle \circledast \langle B_1, \dots, B_n \rangle = \langle B_1, \dots, B_n \rangle \quad (26)$$

and

$$\langle A_1, \dots, A_m \rangle \circledast \langle \rangle = \langle A_1, \dots, A_m \rangle. \quad (27)$$

The multiplication of objects in $\langle \text{Set} \rangle$ is associative (which was our goal with making this construction). Indeed, both

$$(\langle A_1, \dots, A_l \rangle \circledast \langle B_1, \dots, B_m \rangle) \circledast \langle C_1, \dots, C_n \rangle \quad (28)$$

and

$$\langle A_1, \dots, A_l \rangle \circledast (\langle B_1, \dots, B_m \rangle \circledast \langle C_1, \dots, C_n \rangle) \quad (29)$$

are equal to

$$\langle A_1, \dots, A_l, B_1, \dots, B_m, C_1, \dots, C_n \rangle. \quad (30)$$

Remark 18.4. Note that, for any sets A, B , we have

$$\langle A, B \rangle = A \times B. \quad (31)$$

In other words, the multiplication operation in $\langle \text{Set} \rangle$ “coincides” with the cartesian product operation in the special case when we are multiplying two sets together.

Because of this, the operation of cartesian product is well-defined for $\langle \text{Set} \rangle$. Namely, given tuples sets $\langle A_1, \dots, A_l \rangle$ and $\langle B_1, \dots, B_m \rangle$, their cartesian product is again a tuple set:

$$\langle A_1, \dots, A_l \rangle \times \langle B_1, \dots, B_m \rangle = \langle \langle A_1, \dots, A_l \rangle \langle B_1, \dots, B_m \rangle \rangle, \quad (32)$$

(which is a “nested” tuple set).

18.3. Moore machines, $\langle \text{Set} \rangle$ version

We can now slightly rework the definition of Moore machines using morphisms in $\langle \text{Set} \rangle$ rather than Set .

Definition 18.5 (Moore machine, $\langle \text{Set} \rangle$ version)

A *Moore machine* is a tuple

$$\langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle, \quad (33)$$

where \mathbf{U} , \mathbf{X} , and \mathbf{Y} are objects of $\langle \text{Set} \rangle$,

$$\text{dyn} : \mathbf{U} \circ \mathbf{X} \rightarrow \mathbf{X}, \quad (34)$$

and

$$\text{ro} : \mathbf{X} \rightarrow \mathbf{Y}, \quad (35)$$

are morphisms in $\langle \text{Set} \rangle$ (so they are functions), and $\text{st} \in \mathbf{X}$.

Composition

Consider two Moore machines

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{st}_f \rangle \quad (36)$$

$$g = \langle \mathbf{U}_g, \mathbf{X}_g, \mathbf{Y}_g, \text{dyn}_g, \text{ro}_g, \text{st}_g \rangle, \quad (37)$$

that are compatible for composition, in the sense that $\mathbf{Y}_f = \mathbf{U}_g$. Their composition is given by

$$f ; g = \langle \mathbf{U}_{f ; g}, \mathbf{X}_{f ; g}, \mathbf{Y}_{f ; g}, \text{dyn}_{f ; g}, \text{ro}_{f ; g}, \text{st}_{f ; g} \rangle, \quad (38)$$

where

$$\begin{aligned} \mathbf{U}_{f ; g} &:= \mathbf{U}_f, \\ \mathbf{X}_{f ; g} &:= \mathbf{X}_f \circ \mathbf{X}_g, \\ \mathbf{Y}_{f ; g} &:= \mathbf{Y}_g, \\ \text{st}_{f ; g} &:= \text{st}_f \circ \text{st}_g, \end{aligned} \quad (39)$$

and $\text{dyn}_{f ; g}$ and $\text{ro}_{f ; g}$ are defined as

$$\begin{aligned} \text{dyn}_{f ; g} : \mathbf{U}_f \circ \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \mathbf{X}_f \circ \mathbf{X}_g, \\ u \circ x_f \circ x_g &\mapsto \text{dyn}_f(u \circ x_f) \circ \text{dyn}_g(\text{ro}_f(x_f) \circ x_g), \end{aligned} \quad (40)$$

and

$$\begin{aligned} \text{ro}_{f ; g} : \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \mathbf{Y}_g, \\ x_f \circ x_g &\mapsto \text{ro}_g(x_g). \end{aligned} \quad (41)$$

Composition is associative

Let three composable Moore machines f , g , and h be given. We check that each of the six entries in the definition Def. 18.5 coincide for $(f ; g) ; h$ and $f ; (g ; h)$.

Clearly,

$$\mathbf{U}_{(f ; g) ; h} = \mathbf{U}_f = \mathbf{U}_{f ; (g ; h)} \quad (42)$$

and

$$\mathbf{Y}_{(f ; g) ; h} = \mathbf{Y}_h = \mathbf{Y}_{f ; (g ; h)}. \quad (43)$$

Furthermore,

$$\mathbf{X}_{(f \circ g) \circ h} = (\mathbf{X}_f \circ \mathbf{X}_g) \circ \mathbf{X}_h = \mathbf{X}_f \circ (\mathbf{X}_g \circ \mathbf{X}_h) = \mathbf{X}_{f \circ (g \circ h)} \quad (44)$$

since concatenation of lists is associative. Similarly,

$$\begin{aligned} \mathbf{st}_{(f \circ g) \circ h} &= \mathbf{st}_{f \circ g} \circ \mathbf{st}_h \\ &= (\mathbf{st}_f \circ \mathbf{st}_g) \circ \mathbf{st}_h \\ &= \mathbf{st}_f \circ (\mathbf{st}_g \circ \mathbf{st}_h) \\ &= \mathbf{st}_f \circ \mathbf{st}_{g \circ h} \\ &= \mathbf{st}_{f \circ (g \circ h)}. \end{aligned} \quad (45)$$

Next we show that $\mathbf{dyn}_{(f \circ g) \circ h} = \mathbf{dyn}_{f \circ (g \circ h)}$.

On the one hand,

$$\begin{aligned} \mathbf{dyn}_{(f \circ g) \circ h} : \mathbf{U}_f \circ \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h &\rightarrow \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h \\ u \circ x_f \circ x_g \circ x_h &\mapsto \mathbf{dyn}_{f \circ g}(u \circ x_f \circ x_g) \circ \mathbf{dyn}_h(\mathbf{ro}_{f \circ g}(x_f \circ x_g) \circ x_h) \\ &= \mathbf{dyn}_f(u \circ x_f) \circ \mathbf{dyn}_g(\mathbf{ro}_f(x_f) \circ x_g) \circ \mathbf{dyn}_h(\mathbf{ro}_g(x_g) \circ x_h), \end{aligned} \quad (46)$$

while on the other hand

$$\begin{aligned} \mathbf{dyn}_{f \circ (g \circ h)} : \mathbf{U}_f \circ \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h &\rightarrow \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h \\ u \circ x_f \circ x_g \circ x_h &\mapsto \mathbf{dyn}_f(u \circ x_f) \circ \mathbf{dyn}_{g \circ h}(\mathbf{ro}_f(x_f) \circ x_g \circ x_h) \\ &= \mathbf{dyn}_f(u \circ x_f) \circ \mathbf{dyn}_g(\mathbf{ro}_f(x_f) \circ x_g) \circ \mathbf{dyn}_h(\mathbf{ro}_g(x_g) \circ x_h). \end{aligned} \quad (47)$$

So, these two functions are indeed the same.

Finally, we verify that $\mathbf{ro}_{(f \circ g) \circ h} = \mathbf{ro}_{f \circ (g \circ h)}$:

$$\begin{aligned} \mathbf{ro}_{(f \circ g) \circ h} : \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h &\rightarrow \mathbf{Y}_h \\ x_f \circ x_g \circ x_h &\mapsto \mathbf{ro}_h(x_h) \end{aligned} \quad (48)$$

while

$$\begin{aligned} \mathbf{ro}_{f \circ (g \circ h)} : \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h &\rightarrow \mathbf{Y}_h, \\ x_f \circ x_g \circ x_h &\mapsto \mathbf{ro}_{g \circ h}(x_g \circ x_h) = \mathbf{ro}_h(x_h). \end{aligned} \quad (49)$$

The semicategory of Moore machines

Now that we have shown that composition of Moore machines is associative (with our new definition), we can organize Moore machines as a semicategory.

Definition 18.6 (Moo)

The semicategory of Moore machines **Moo** is given by:

1. *Objects*: objects of $\langle \text{Set} \rangle$.
2. *Morphisms*: A morphism from \mathbf{U} to \mathbf{Y} is a Moore machine

$$\langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \mathbf{dyn}, \mathbf{ro}, \mathbf{st} \rangle, \quad (50)$$

where

- ▷ $\mathbf{U}, \mathbf{X}, \mathbf{Y}$ are objects of $\langle \text{Set} \rangle$;
- ▷ $\mathbf{dyn} : \mathbf{U} \circ \mathbf{X} \rightarrow \mathbf{X}$ and $\mathbf{ro} : \mathbf{X} \rightarrow \mathbf{Y}$ are morphisms in $\langle \text{Set} \rangle$;

▷ $\mathbf{s} \in \mathbf{X}$.

3. *Composition*: as defined above in this section.

18.4. Standard action of Moore machines

Given a Moore machine $f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{st}_f \rangle$ we saw that we can use it to transform an infinite sequence of inputs u_0, u_1, u_2, \dots into an infinite sequence of outputs y_0, y_1, y_2, \dots using the following recipe

$$\begin{cases} x_{k+1} = \text{dyn}_f(u_k \circ x_k) \\ y_k = \text{ro}_f(x_k). \end{cases} \quad (51)$$

Rephrased mathematically, this means that (51), together with f , defines a function

$$\text{act}_f : \text{Stream } \mathbf{U}_f \rightarrow \text{Stream } \mathbf{Y}_f, \quad (52)$$

which takes any sequence of elements of \mathbf{U}_f and maps it to a corresponding sequence of elements of \mathbf{Y}_f . We call this the *standard action* of f on sequences.

Remark 18.7. One way to think about the function act_f is to imagine it being calculated in two steps.

1. Given a sequence $u = u_0, u_1, u_2, \dots$, there is a unique solution $s = s_0, s_1, s_2, \dots$ of the recursion $x_{k+1} = \text{dyn}_f(u_k, x_k)$ with $s_0 = \text{st}_f$. (Clearly, this solution can be computed iteratively.)
2. Given the solution s , the sequence $y = \text{act}_f(u)$ is simply $y = s \circ \text{ro}_f$.

Example 18.8. Consider a Moore machine f with $\mathbf{U} = \mathbf{X} = \mathbf{Y} = \langle \mathbb{N} \rangle$ and let

$$\begin{aligned} \text{dyn}_f : \mathbf{U} \circ \mathbf{X} &\rightarrow \mathbf{X} \\ \langle m \rangle \circ \langle n \rangle &\mapsto \langle m + n \rangle \end{aligned} \quad (53)$$

and

$$\begin{aligned} \text{ro}_f : \mathbf{X} &\rightarrow \mathbf{Y} \\ \langle n \rangle &\mapsto \langle n + 1 \rangle \end{aligned} \quad (54)$$

and $\text{st} = \langle 0 \rangle$.

Given a sequence of inputs of the form $u = \langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle, \langle 5 \rangle, \dots$, what are the first five entries of the corresponding sequence of outputs?

To compute this, let us first calculate the first five entries of the sequence of states $s = s_0, s_1, s_2, \dots$ that solves (51). We have

$$s_0 = \text{st} = \langle 0 \rangle, \quad (55)$$

$$s_1 = \langle 1 + 0 \rangle = \langle 1 \rangle, \quad (56)$$

$$s_2 = \langle 2 + 1 \rangle = \langle 3 \rangle, \quad (57)$$

$$s_3 = \langle 3 + 3 \rangle = \langle 6 \rangle, \quad (58)$$

$$s_4 = \langle 4 + 6 \rangle = \langle 10 \rangle. \quad (59)$$

Now, applying ro_f to the entries of this sequence of state, we obtain the first five entries of the output sequence:

$$y_0 = \langle 0 + 1 \rangle = \langle 1 \rangle, \quad (60)$$

$$y_1 = \langle 1 + 1 \rangle = \langle 2 \rangle, \quad (61)$$

$$y_2 = \langle 3 + 1 \rangle = \langle 4 \rangle, \quad (62)$$

$$y_3 = \langle 6 + 1 \rangle = \langle 7 \rangle, \quad (63)$$

$$y_4 = \langle 10 + 1 \rangle = \langle 11 \rangle. \quad (64)$$

Graded exercise F.1 (ComputingMooreActions)

Consider the Moore machine f with $\mathbf{U} = \mathbf{X} = \mathbf{Y} = \langle \mathbb{Z} \rangle$, $\mathbf{st} = \langle 0 \rangle$, and

$$\begin{aligned} \mathbf{dyn}_f : \mathbf{U} \circ \mathbf{X} &\rightarrow \mathbf{X} \\ \langle m \rangle \circ \langle n \rangle &\mapsto \langle n - m \rangle, \end{aligned} \quad (65)$$

$$\begin{aligned} \mathbf{ro}_f : \mathbf{X} &\rightarrow \mathbf{Y} \\ \langle n \rangle &\mapsto \langle n + 2 \rangle. \end{aligned} \quad (66)$$

Given a stream $u \in \text{Stream } \mathbf{U}$ of inputs of the form $u = \langle 1 \rangle, \langle 2 \rangle, \langle 3 \rangle, \langle 4 \rangle, \langle 5 \rangle, \dots$, compute the first five entries of the corresponding stream of outputs for the following two actions:

1. The standard action defined via the recursion

$$\begin{cases} x_{k+1} = \mathbf{dyn}_f(u_k \circ x_k) \\ y_k = \mathbf{ro}_f(x_k) \end{cases} \quad (67)$$

with $x_0 = \mathbf{st}$.

2. The action defined via the recursion

$$\begin{cases} x_{k+1} = \mathbf{dyn}_f(u_k \circ x_k) \\ \tilde{x}_{k+1} = \mathbf{dyn}_f(u_k \circ x_{k+1}) \\ y_k = \mathbf{ro}_f(\tilde{x}_k) \end{cases} \quad (68)$$

with $x_0 = \tilde{x}_0 = \mathbf{st}$.

Compositionality

Proposition 18.9. The standard action of Moore machines on signal sequences is compatible with Moore machine composition in the sense that

$$\mathbf{act}_{f \circ g} = \mathbf{act}_f \circ \mathbf{act}_g \quad (69)$$

for any composable Moore machines f and g .

Proof. Suppose we have Moore machines

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \mathbf{dyn}_f, \mathbf{ro}_f, \mathbf{st}_f \rangle \quad (70)$$

and

$$g = \langle \mathbf{U}_g, \mathbf{X}_g, \mathbf{Y}_g, \mathbf{dyn}_g, \mathbf{ro}_g, \mathbf{st}_g \rangle, \quad (71)$$

with $\mathbf{Y}_f = \mathbf{U}_g$.

We will first calculate the right-hand side of (69), following the procedure of Remark 18.7.

Given a sequence $u \in \text{Stream } \mathbf{U}_f$, to calculate $\mathbf{act}_f(u)$ we first map u to the unique solution $r_u \in \text{Stream } \mathbf{X}_f$ of the recursion

$$x_{k+1} = \mathbf{dyn}_f(u_k, x_k) \quad k \in \mathbb{N}, \quad (72)$$

such that $r_u(0) = \mathbf{st}_f$. Then $\mathbf{act}_f(u) = r_u \circ \mathbf{ro}_f$, an element of $\text{Stream } \mathbf{Y}_f = \text{Stream } \mathbf{U}_g$.

Next, to calculate $(\mathbf{act}_f \circ \mathbf{act}_g)(u)$, we first map $\mathbf{act}_f(u)$ to the unique solu-

tion $s_{\text{act}_f(u)} \in \text{Stream } \mathbf{X}_g$ of the recursion

$$x_{k+1} = \text{dyn}_g(\text{act}_f(u_k, x_k)) \quad k \in \mathbb{N} \quad (73)$$

such that $s_{\text{act}_f(u)}(0) = \text{st}_g$, and then we have

$$(\text{act}_f \circ \text{act}_g)(u) = s_{\text{act}_f(u)} \circ \text{ro}_g. \quad (74)$$

We calculate the left-hand side of (69). Given $u \in \text{Stream } \mathbf{U}_f$, we map it to the unique solution $q_u \in \text{Stream } \mathbf{X}_{f \circ g} = \text{Stream}(\mathbf{X}_f \circ \mathbf{X}_g)$ of the recursion

$$x_{k+1} = \text{dyn}_{f \circ g}(u_k, x_k) \quad k \in \mathbb{N} \quad (75)$$

with $q_u(0) = \text{st}_f \circ \text{st}_g$. Using the definition of $\text{dyn}_{f \circ g}$ and the fact that we can write any $x_k \in \mathbf{X}_f \circ \mathbf{X}_g$ as

$$x_k = x_k^f \circ x_k^g \quad \text{with } x_k^f \in \mathbf{X}_f, x_k^g \in \mathbf{X}_g, \quad (76)$$

we can rewrite (75) as

$$x_{k+1}^f \circ x_{k+1}^g = \text{dyn}_f(u_k, x_k^f) \circ \text{dyn}_g(\text{ro}_f(x_k^f), x_k^g) \quad k \in \mathbb{N}. \quad (77)$$

Now observe that the sequence $r_u(k) \circ s_{\text{act}_f(u)}(k)$ solves this recursion: by definition r_u solves the recursion

$$x_{k+1} = \text{dyn}_f(u_k, x_k) \quad k \in \mathbb{N}, \quad (78)$$

and, recalling that $\text{act}_f(u)_k = \text{ro}_f(r_u(k))$, we see that $s_{\text{act}_f(u)}$ solves the recursion

$$x_{k+1} = \text{dyn}_g(\text{ro}_f(r_u(k)), x_k) \quad k \in \mathbb{N}. \quad (79)$$

Since $r_u(0) \circ s_{\text{act}_f(u)}(0) = \text{st}_f \circ \text{st}_g$, this implies that the unique solution q_u above is precisely $q_u(k) = r_u(k) \circ s_{\text{act}_f(u)}(k)$.

Finally, we have

$$\text{act}_{f \circ g}(u) = q_u \circ \text{ro}_{f \circ g}. \quad (80)$$

Evaluating at any $k \in \mathbb{N}$ we find

$$(q_u \circ \text{ro}_{f \circ g})(k) = \text{ro}_{f \circ g}(q_u(k)) \quad (81)$$

$$= \text{ro}_{f \circ g}(r_u(k) \circ s_{\text{act}_f(u)}(k)) \quad (82)$$

$$= \text{ro}_g(s_{\text{act}_f(u)}(k)) \quad (83)$$

$$= (s_{\text{act}_f(u)} \circ \text{ro}_g)(k). \quad (84)$$

Comparing with (74), we conclude that

$$\text{act}_{f \circ g}(u) = (\text{act}_f \circ \text{act}_g)(u). \quad (85)$$

□

18.5. Semicategory actions

In this section we generalize from actions of a semigroup to actions of a semicategory, using the example of Moore machines acting on signal sequences.

To motivate our story, let us first consider Moore machines of the form

$$\langle \mathbf{U}, \mathbf{X}, \mathbf{U}, \text{dyn}, \text{ro}, \text{st} \rangle \quad (86)$$

where the input and output sets are equal. In other words, we are considering the set $\text{Hom}_{\mathbf{Moo}}(\mathbf{U}; \mathbf{U})$, which is a semigroup under morphism composition.

In Section 18.4 we defined a standard action which associates to every morphism $f \in \text{Hom}_{\mathbf{Moo}}(\mathbf{U}; \mathbf{U})$ a function

$$\text{act}_f : \text{Stream } \mathbf{U} \rightarrow \text{Stream } \mathbf{U}. \quad (87)$$

This action defines a function

$$\text{act}_\rightarrow : \text{Hom}_{\mathbf{Moo}}(\mathbf{U}; \mathbf{U}) \rightarrow \text{End}(\text{Stream } \mathbf{U}) \quad (88)$$

where

$$\text{act}_\rightarrow(f) = \text{act}_f. \quad (89)$$

And this function act_\rightarrow is in fact a semigroup morphism, and so a semigroup action, since we proved in Prop. 18.9 that

$$\text{act}_{f; g} = \text{act}_f \circ \text{act}_g. \quad (90)$$

(Compare with Def. 11.8 of semigroup action.)

Now consider the general situation of Moore machines acting on signals. Given any Moore machine of the general form

$$f = \langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle \quad (91)$$

(the input and output spaces are no longer necessarily equal) we again have an associated function on signals

$$\text{act}_f : \text{Stream } \mathbf{U} \rightarrow \text{Stream } \mathbf{Y}. \quad (92)$$

We can assemble this data as a family of functions (all which we call act_\rightarrow)

$$\text{act}_\rightarrow : \text{Hom}_{\mathbf{Moo}}(\mathbf{U}; \mathbf{Y}) \rightarrow \text{Hom}_{\text{Set}}(\text{Stream } \mathbf{U}; \text{Stream } \mathbf{Y}), \quad (93)$$

where \mathbf{U} and \mathbf{Y} range over all objects of \mathbf{Moo} . Or, if you will,

$$\text{act}_\rightarrow : \text{Mor}_{\mathbf{Moo}} \rightarrow \text{Mor}_{\text{Set}}. \quad (94)$$

From Prop. 18.9 we have that this function is compatible with the composition operations in \mathbf{Moo} and Set :

$$\text{act}_\rightarrow(f; g) = \text{act}_\rightarrow(f) \circ \text{act}_\rightarrow(g). \quad (95)$$

Note that the sets $\text{Stream } \mathbf{U}$ and $\text{Stream } \mathbf{Y}$ involved in the right-hand side of (93) depend on the objects \mathbf{U} and \mathbf{Y} on the left-hand side. We will encode this also with a function

$$\begin{aligned} \text{act}_\rightarrow : \text{Ob}_{\mathbf{Moo}} &\rightarrow \text{Ob}_{\text{Set}}, \\ \mathbf{U} &\mapsto \text{Stream } \mathbf{U}, \end{aligned} \quad (96)$$

in which case (93) becomes

$$\text{act}_\rightarrow : \text{Hom}_{\text{Moo}}(\mathbf{U}; \mathbf{Y}) \rightarrow \text{Hom}_{\text{Set}}(\text{act}_\bullet \mathbf{U}; \text{act}_\bullet \mathbf{Y}). \quad (97)$$

In summary, we have reformulated Moore machine actions as consisting of a pair of functions

$$\text{act}_\rightarrow : \text{Mor}_{\text{Moo}} \rightarrow \text{Mor}_{\text{Set}} \quad \text{and} \quad \text{act}_\bullet : \text{Ob}_{\text{Moo}} \rightarrow \text{Ob}_{\text{Set}} \quad (98)$$

which work together as in (97) and such that act_\rightarrow is compatible with composition as in (95).

Now we formalize this situation as a general definition.

Definition 18.10 (Semicategory action)

A *semicategory action* of a semicategory \mathbf{C} is

Constituents

1. A map $\text{act}_\bullet : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\text{Set}}$;
2. For every two objects $X, Y \in \text{Ob}_{\mathbf{C}}$, a map

$$\text{act}_\rightarrow : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\text{Set}}(\text{act}_\bullet(X); \text{act}_\bullet(Y)). \quad (99)$$

Conditions

1. For all composable morphisms f and g ,

$$\text{act}_\rightarrow(f \circ g) = \text{act}_\rightarrow(f) \circ \text{act}_\rightarrow(g). \quad (100)$$

The compatibility of the action with composition is illustrated in Fig. 3.

For reference, let us also fix the following definition.

Definition 18.11 (Standard action of Moore machines)

The *standard action of Moore machines* on sequences is given by

$$\begin{aligned} \text{act}_\bullet : \text{Ob}_{\text{Moo}} &\rightarrow \text{Ob}_{\text{Set}}, \\ \mathbf{U} &\mapsto \text{Stream } \mathbf{U}, \end{aligned} \quad (101)$$

on the level of objects, and on the level of morphisms, the functions

$$\text{act}_\rightarrow : \text{Hom}_{\text{Moo}}(\mathbf{U}; \mathbf{Y}) \rightarrow \text{Hom}_{\text{Set}}(\text{Stream } \mathbf{U}; \text{Stream } \mathbf{Y}) \quad (102)$$

are defined via the recursion equations

$$\begin{cases} x_{k+1} = \text{dyn}_f(u_k \circ_< x_k) \\ y_k = \text{ro}_f(x_k) \end{cases} \quad (103)$$

as in Section 18.4.

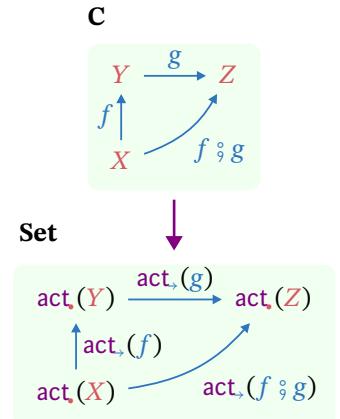


Figure 3.

18.6. More machines

A Moore machine outputs 1 element at each step; what if a machine was able to output more than one or zero output? We will call these *More* machines.

We define their dynamics and readout to be of the form

$$\begin{cases} \text{dyn} : \mathbf{U} \circ \mathbf{X} \rightarrow \mathbf{X}, \\ \text{ro} : \mathbf{X} \rightarrow \text{List } \mathbf{Y}, \end{cases} \quad (104)$$

where the output set is not just \mathbf{Y} but $\text{List } \mathbf{Y}$, the set of finite list of elements of \mathbf{Y} . In other words, for a given state, the machine can produce zero or more outputs in the form of a list.

We'll specify More machines as a tuple $\langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle$, like we did for Moore machines.

Composition

Given More machines $f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{st}_f \rangle$ and $g = \langle \mathbf{U}_g, \mathbf{X}_g, \mathbf{Y}_g, \text{dyn}_g, \text{ro}_g, \text{st}_g \rangle$ with $\mathbf{Y}_f = \mathbf{U}_g$, their composition is the More machine with

$$\begin{aligned} \mathbf{U}_{f \circ g} &= \mathbf{U}_f, \\ \mathbf{X}_{f \circ g} &= \mathbf{X}_f \circ \mathbf{X}_g, \\ \text{st}_{f \circ g} &= \text{st}_f \circ \text{st}_g, \\ \mathbf{Y}_{f \circ g} &= \mathbf{Y}_g. \end{aligned} \quad (105)$$

The dynamics of the composite $f \circ g$ is

$$\begin{aligned} \text{dyn}_{f \circ g} : \mathbf{U}_f \circ \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \mathbf{X}_f \circ \mathbf{X}_g, \\ u \circ x_f \circ x_g &\mapsto \text{dyn}_f(u \circ x_f) \circ \text{dyn}_g(y_f[n] \circ \text{dyn}_g(y_f[n-1] \circ \dots \circ \text{dyn}_g(y_f[1] \circ x_g))). \end{aligned} \quad (106)$$

where $y_f = \text{ro}_f(x_f) \in \text{List } \mathbf{Y}_f$ and n is its length.

The readout of $f \circ g$ is

$$\begin{aligned} \text{ro}_{f \circ g} : \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \text{List } \mathbf{Y}_g, \\ x_f \circ x_g &\mapsto \text{ro}_g(x_g). \end{aligned} \quad (107)$$

Definition 18.12 (Mor)

The semicategory of More machines **Mor** is given by:

1. *Objects*: objects of $\langle \mathbf{Set} \rangle$.
2. *Morphisms*: A morphism is a tuple

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{st}_f \rangle, \quad (108)$$

where:

- ▷ $\mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f$ are objects of $\langle \mathbf{Set} \rangle$;
- ▷ $\text{st}_f \in \mathbf{X}_f$;
- ▷ $\text{dyn}_f : \mathbf{U}_f \circ \mathbf{X}_f \rightarrow \mathbf{X}_f$;
- ▷ $\text{ro}_f : \mathbf{X}_f \rightarrow \text{List } \mathbf{Y}_f$.

3. *Composition of morphisms*: Composition is given by (106) and (107).

Examples

Example 18.13 (Duplicator). We consider an example of a More machine d which we call a *duplicator*. Practically, this machine takes an input, and duplicates it. The machine is written as

$$\langle \mathbf{U}_d, \mathbf{X}_d, \mathbf{Y}_d, \text{dyn}_d, \text{ro}_d, \text{st}_d \rangle, \quad (109)$$

where $\mathbf{U}_d = \mathbf{X}_d = \mathbf{Y}_d$, and

$$\begin{aligned} \text{dyn}_d : \mathbf{U}_d \nparallel \mathbf{X}_d &\rightarrow \mathbf{X}_d, \\ u \circ \langle x \rangle &\mapsto u, \end{aligned} \quad (110)$$

$$\begin{aligned} \text{ro}_d : \mathbf{X}_d &\rightarrow \text{List } \mathbf{Y}_d, \\ x &\mapsto [x, x]_{\mathbf{Y}_d}, \end{aligned} \quad (111)$$

and $\text{st} = \langle \rangle \in \mathbf{X}_d$.

Example 18.14 (Discarder). Here is an example of a More machine e which we call a *discarder*. Practically, this machine discards every other input. The machine is written as:

$$\langle \mathbf{U}_e, \mathbf{X}_e, \mathbf{Y}_e, \text{dyn}_e, \text{ro}_e, \text{st}_e \rangle, \quad (112)$$

where $\mathbf{X}_e = \langle \{\perp, \top\} \rangle \nparallel \mathbf{U}_e$, and

$$\begin{aligned} \text{dyn}_e : \mathbf{U}_e \nparallel \langle \{\perp, \top\} \rangle \nparallel \mathbf{U}_e &\rightarrow \langle \{\perp, \top\} \rangle \nparallel \mathbf{U}_e \\ u \circ \langle \perp \rangle \circ \langle x \rangle &\mapsto \langle \top \rangle \circ \langle u \rangle \\ u \circ \langle \top \rangle \circ \langle x \rangle &\mapsto \langle \perp \rangle \circ \langle u \rangle, \end{aligned} \quad (113)$$

$$\begin{aligned} \text{ro}_e : \langle \{\perp, \top\} \rangle \nparallel \mathbf{U}_e &\rightarrow \text{List } \mathbf{Y}_e \\ \langle \perp \rangle \circ \langle x \rangle &\mapsto []_{\mathbf{Y}_e} \\ \langle \top \rangle \circ \langle x \rangle &\mapsto [x]_{\mathbf{Y}_e} \end{aligned} \quad (114)$$

and $\text{st}_e = \langle \perp \rangle \circ \langle x \rangle \in \mathbf{X}_e$, for an arbitrary $x \in \mathbf{U}_e$.

Example 18.15 (Terminator). We describe a More machine t which we call a *terminator*. Practically, this machine terminates any input, outputting an empty list. The machine is written as

$$\langle \mathbf{U}_t, \mathbf{X}_t, \mathbf{Y}_t, \text{dyn}_t, \text{ro}_t, \text{st}_t \rangle, \quad (115)$$

where:

$$\begin{aligned} \text{dyn}_t : \mathbf{U}_t \nparallel \mathbf{X}_t &\rightarrow \mathbf{X}_t, \\ u \circ \langle x \rangle &\mapsto x, \end{aligned} \quad (116)$$

$$\begin{aligned} \text{ro}_t : \mathbf{X}_t &\rightarrow \text{List } \mathbf{Y}_t, \\ \mathbf{X}_t &\mapsto []_{\mathbf{Y}_t}, \end{aligned} \quad (117)$$

and $\text{st}_t \in \mathbf{X}_t$ can be any element.

18.7. LTI systems

Definition 18.16 (LTI System)

A *linear time-invariant dynamical (LTI) system*, in a so-called state-space representation, is specified by real vector spaces $\mathbf{U} = \mathbb{R}^l$ (input space), $\mathbf{Y} = \mathbb{R}^m$ (output space), and $\mathbf{X} = \mathbb{R}^n$ (state space), along with a system of equations of the form

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (118)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \quad (119)$$

and an *initial state* $\mathbf{st} \in \mathbf{X}$, where $t \in \mathbb{R}_{\geq 0}$, $\mathbf{u}(t) \in \mathbf{U}$, $\mathbf{y}(t) \in \mathbf{Y}$, $\mathbf{x}(t) \in \mathbf{X}$, and where $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are real matrices of appropriate dimension.

Remark 18.17. Def. 18.16 includes the particular case of matrices with 0 dimension. For instance, in a system with $\mathbf{U} = \mathbb{R}^l$ and $\mathbf{Y} = \mathbb{R}^m$, we could have $\mathbf{A} \in \mathbb{R}^{0 \times 0}$ (in other words, no state). This would imply $\mathbf{B} \in \mathbb{R}^{0 \times l}$, $\mathbf{C} \in \mathbb{R}^{m \times 0}$, and $\mathbf{D} \in \mathbb{R}^{m \times l}$. Matrices with zero rows and/or zero columns multiply exactly as other matrices. For instance, the multiplication of a $\mathbb{R}^{0 \times 0}$ matrix with a $\mathbb{R}^{0 \times l}$ matrix, will return a $\mathbb{R}^{0 \times l}$ matrix.

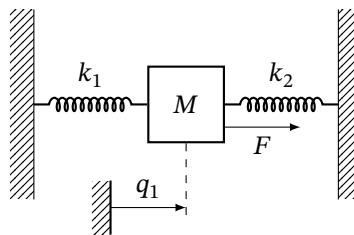
We compactly refer to an LTI system by writing it as a tuple $\langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$. The equations (118) describe the dynamics. The equations (119) describe the output, or, one might say, the variables that are “exposed” or externally visible. The matrix \mathbf{D} is called the *feedthrough term*.

Definition 18.18 (Proper LTI System)

We will call an LTI system $\langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$ *proper* if the matrix \mathbf{D} is the zero matrix (understood as having the appropriate dimensions).

Remark 18.19. When using state-space LTI systems as models, it is typical that the equations (118) are chosen using physical laws and first principles reasoning, while the matrices \mathbf{C} and \mathbf{D} in (119) are rather chosen based on what information from our model is explicitly relevant or accessible.

Example 18.20. Consider a mass m lying on a frictionless surface, and attached to two springs as depicted in Example 18.20.



The coordinate $q(t)$ describes position of the mass along one horizontal dimension, F denotes a force that is applied to m in that horizontal direction, and k_1 and k_2 are the spring constants of the respective springs.

The dynamics of the position coordinate q as a function of time is described by the differential equation

$$(k_1 + k_2)q + m\ddot{q} = F. \quad (120)$$

We may rewrite this as a proper LTI system in a state-space representation by choosing the state-variable to be

$$\mathbf{x} = \begin{bmatrix} q(t) \\ \dot{q}(t) \end{bmatrix}. \quad (121)$$

Then

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ \frac{-(k_1+k_2)}{m} & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} F \quad (122)$$

describes the dynamics, and as output we might choose

$$\mathbf{y} = [1 \ 0] \mathbf{x}(t) + [0] F. \quad (123)$$

Equivalent LTI systems

Definition 18.21 (Equivalence of LTI systems)

Two systems $\langle \mathbf{st}_f, \mathbf{A}_f, \mathbf{B}_f, \mathbf{C}_f, \mathbf{D}_f \rangle$ and $\langle \mathbf{st}_g, \mathbf{A}_g, \mathbf{B}_g, \mathbf{C}_g, \mathbf{D}_g \rangle$ are *equivalent* if and only if there exists an invertible linear transformation $\mathbf{x}_g(t) = \mathbf{T}\mathbf{x}_f(t)$ such that

$$\mathbf{A}_g = \mathbf{T}\mathbf{A}_f\mathbf{T}^{-1}, \mathbf{B}_g = \mathbf{T}\mathbf{B}_f, \mathbf{C}_g = \mathbf{C}_f\mathbf{T}^{-1}, \mathbf{D}_g = \mathbf{D}_f, \mathbf{st}_g = \mathbf{T}\mathbf{st}_f. \quad (124)$$

\mathbf{T} is called an *equivalence transformation*.

We think of equivalent LTI-systems as different ways of specifying what is essentially “the same system”. What is different in each specification is only different by a change of coordinates.

Category of LTI systems

We define a category of LTI systems **LTI**.

Definition 18.22 (Category LTI)

The category **LTI** of LTI systems is defined by:

1. *Objects*: natural numbers.
2. *Morphisms*: A morphism in **LTI** from $l \in \mathbb{N}$ to $m \in \mathbb{N}$ is a continuous time LTI system $\langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$ such that l is the dimension of the input space, and m the dimension of the output space.
3. *Composition*: Given morphisms $f : a \rightarrow b$ and $g : b \rightarrow c$, described by the LTI systems

$$\begin{aligned} &\langle \mathbf{st}_f, \mathbf{A}_f, \mathbf{B}_f, \mathbf{C}_f, \mathbf{D}_f \rangle \\ &\langle \mathbf{st}_g, \mathbf{A}_g, \mathbf{B}_g, \mathbf{C}_g, \mathbf{D}_g \rangle, \end{aligned} \quad (125)$$

their composition $(f ; g) : a \rightarrow c$ is the LTI system $\langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$, where

$$\begin{aligned} \mathbf{st} &= \begin{bmatrix} \mathbf{st}_f \\ \mathbf{st}_g \end{bmatrix}, & \mathbf{A} &= \begin{bmatrix} \mathbf{A}_f & \mathbf{0} \\ \mathbf{B}_g \mathbf{C}_f & \mathbf{A}_g \end{bmatrix}, & \mathbf{B} &= \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_g \mathbf{D}_f \end{bmatrix}, \\ \mathbf{C} &= [\mathbf{D}_g \mathbf{C}_f \quad \mathbf{C}_g], & \mathbf{D} &= \mathbf{D}_g \mathbf{D}_f. \end{aligned} \quad (126)$$

4. *Identities*: the identity for $l \in \mathbb{N}$ is the system $\langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times l}, \mathbf{0}^{m \times 0}, \mathbf{0}^{m \times l} \rangle$.

Remark 18.23. Again, all of this works with matrices with zero rows and/or zero columns. Let’s see practically how via a simple instance. Consider the system

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

$\underline{f} : a \rightarrow b$ given by $\langle \text{st}_{\underline{f}}, \mathbf{A}_{\underline{f}}, \mathbf{B}_{\underline{f}}, \mathbf{C}_{\underline{f}}, \mathbf{D}_{\underline{f}} \rangle$ and the system $\underline{g} : b \rightarrow c$ given by $\langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times b}, \mathbf{0}^{c \times 0}, \mathbf{0}^{c \times b} \rangle$. Their composition $\underline{f} ; \underline{g} : a \rightarrow c$ is a system $\langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$ with:

$$\text{st} = \begin{bmatrix} \text{st}_{\underline{f}} \\ \mathbf{0}^{0 \times 1} \end{bmatrix} = \text{st}_{\underline{f}}.$$

How could we write the above equation? $\text{st}_{\underline{f}}$ has s rows and 1 column, and $\mathbf{0}^{0 \times 1}$ has 0 rows and 1 column (it has to have 1 column for us to be able to write the above block matrix), allowing us to write the expression as $\text{st}_{\underline{f}}$. We can now write:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{\underline{f}} & \mathbf{0}^{s \times 0} \\ \mathbf{0}^{0 \times b} \mathbf{C}_{\underline{f}} & \mathbf{0}^{0 \times 0} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\underline{f}} & \mathbf{0}^{s \times 0} \\ \mathbf{0}^{0 \times s} & \mathbf{0}^{0 \times 0} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{\underline{f}} \\ \mathbf{0}^{0 \times s} \end{bmatrix} = \mathbf{A}_{\underline{f}},$$

The above equality teaches us how to manipulate matrices with zero rows and/or zero columns. First, since $\mathbf{C}_{\underline{f}}$ has b rows and s columns, the multiplication $\mathbf{0}^{0 \times b} \mathbf{C}_{\underline{f}}$ is well defined and gives the zero matrix with 0 rows and s columns. Once we realize this, we see that we end up with a matrix made of four block matrices. Given their zero rows/columns, one can then simplify as shown. Similar arguments can be made for the other matrices resulting from the composition.

Exercise 40. Prove that LTI is indeed a category.

See solution on page 303.

Example 18.24. Consider the LTI for the spring-mass system from Example 18.20, and define the LTI

$$\dot{\mathbf{z}}(t) = p(t) + Cq(t)$$

$$\mathbf{w}(t) = p(t),$$

taking as input the output produced by the spring-mass system (the position of the mass along the horizontal dimension) and transforms it by a factor C . We can compose the two systems, obtaining the system $\langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle$ with

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{k_1+k_2}{m} & 0 & 0 \\ C & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{C} = [0 \ 0 \ 1].$$

This can be visualized more intuitively with the explicit composed dynamics:

$$\begin{aligned} \begin{bmatrix} \dot{q}(t) \\ \ddot{q}(t) \\ \dot{p}(t) \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ -\frac{k_1+k_2}{m} & 0 & 0 \\ C & 0 & 1 \end{bmatrix} \begin{bmatrix} q(t) \\ \dot{q}(t) \\ p(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} F \\ w(t) &= [0 \ 0 \ 1] \begin{bmatrix} q(t) \\ \dot{q}(t) \\ p(t) \end{bmatrix} \end{aligned}$$

Given an input force, we will get a transformed horizontal position as an output.

Standard action of LTI systems

Definition 18.25 (Category action)

A *category action* of a category \mathbf{C} is

Constituents

1. A map $\text{act}_{\mathbf{C}} : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\text{Set}}$;

2. For every two objects $X, Y \in \text{Ob}_{\mathbf{C}}$, a map

$$\text{act}_\rightarrow : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\text{Set}}(\text{act}_\rightarrow(X); \text{act}_\rightarrow(Y)). \quad (127)$$

Conditions

1. For all composable morphisms f and g ,

$$\text{act}_\rightarrow(f \circ g) = \text{act}_\rightarrow(f) \circ \text{act}_\rightarrow(g). \quad (128)$$

2. For all objects $X \in \text{Ob}_{\mathbf{C}}$,

$$\text{act}_\rightarrow(\text{id}_X) = \text{id}_{\text{act}_\rightarrow(X)} \quad (129)$$

Definition 18.26 (LTI standard action)

We define a standard action of **LTI** via:

▷ A map

$$\begin{aligned} \text{act}_\rightarrow : \text{Ob}_{\text{LTI}} &\rightarrow \text{Ob}_{\text{Set}}, \\ n &\mapsto C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^n). \end{aligned} \quad (130)$$

▷ A map

$$\text{act}_\rightarrow : \text{Hom}_{\text{LTI}}(m; n) \rightarrow \text{Hom}_{\text{Set}}(C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^m); C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^n)) \quad (131)$$

where act_\rightarrow takes an LTI system $f : m \rightarrow n$ given by

$$\langle \text{st}_f, \mathbf{A}_f, \mathbf{B}_f, \mathbf{C}_f, \mathbf{D}_f \rangle, \quad (132)$$

and returns the function

$$\begin{aligned} \text{act}_\rightarrow(f) : C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^m) &\rightarrow C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^n), \\ \mathbf{u}(t) &\mapsto \mathbf{C}_f \mathbf{s}_f(t) + \mathbf{D}_f \mathbf{u}(t), \end{aligned} \quad (133)$$

where \mathbf{s}_f is the unique solution of the initial value problem

$$\begin{cases} \dot{\mathbf{x}}(t) &= \mathbf{A}_f \mathbf{x}(t) + \mathbf{B}_f \mathbf{u}(t) \\ \mathbf{x}(0) &= \text{st}. \end{cases} \quad (134)$$

Remark 18.27. The initial value problem

$$\begin{cases} \dot{\mathbf{x}}(t) &= \mathbf{A} \mathbf{x}(t) + \mathbf{B} \mathbf{u}(t) \\ \mathbf{x}(0) &= \text{st} \end{cases} \quad (135)$$

is a system of linear first-order non-homogenous differential equations. It does *not* have constant coefficients, because the inhomogenous term $\mathbf{B}_f \mathbf{u}(t)$ is not constant (it depends on the independent variable t).

The theorem of Picard-Lindelöf guarantees that initial value problems of the form (135) always have a unique solution. There is general formula for the solution \mathbf{s} of (135), namely

$$\mathbf{s}(t) = e^{\mathbf{A}t} \text{st} + \int_0^t e^{\mathbf{A}(t-\tau)} \mathbf{B} \mathbf{u}(\tau) d\tau. \quad (136)$$

Lemma 18.28. Def. 18.26 indeed defines a category action.

Proof. We need to prove that

$$\text{act}_\rightarrow(f \circ_{\text{LTI}} g) = \text{act}_\rightarrow(f) \circ_{\text{Set}} \text{act}_\rightarrow(g). \quad (137)$$

Consider $f : m \rightarrow n$ and $g : n \rightarrow o$. We first look at the map

$$\begin{aligned} \text{act}_\rightarrow(f \circ_{\text{LTI}} g) : C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^m) &\rightarrow C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^o), \\ \mathbf{u}(t) &\mapsto \mathbf{C}_{f \circ g} \mathbf{s}_{f \circ g}(t) + \mathbf{D}_{f \circ g} \mathbf{u}(t), \end{aligned} \quad (138)$$

where $\mathbf{s}_{f \circ g}$ is the unique solution of the initial value problem

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_{f \circ g} \mathbf{x}(t) + \mathbf{B}_{f \circ g} \mathbf{u}(t), \\ \mathbf{x}(0) &= \mathbf{s}_{f \circ g}, \end{aligned} \quad (139)$$

and

$$\mathbf{C}_{f \circ g} = [\mathbf{D}_g \mathbf{C}_f \quad \mathbf{C}_g], \quad \mathbf{D} = \mathbf{D}_g \mathbf{D}_f. \quad (140)$$

From the definition of composition of LTI systems (Def. 18.22), we know that we can expand (139) into

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \begin{bmatrix} \dot{\mathbf{x}}_f(t) \\ \dot{\mathbf{x}}_g(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} \\ \mathbf{B}_g \mathbf{C}_f & \mathbf{A}_g \end{bmatrix} \begin{bmatrix} \mathbf{x}_f(t) \\ \mathbf{x}_g(t) \end{bmatrix} + \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_g \mathbf{D}_f \end{bmatrix} \mathbf{u}(t) \\ \mathbf{s}_{f \circ g} &= \begin{bmatrix} \mathbf{s}_f \\ \mathbf{s}_g \end{bmatrix}. \end{aligned} \quad (141)$$

By instead looking at $\text{act}_\rightarrow(f)$ we have

$$\begin{aligned} \text{act}_\rightarrow(f) : C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^m) &\rightarrow C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^n), \\ \mathbf{u} &\mapsto \mathbf{C}_f \mathbf{s}_f + \mathbf{D}_f \mathbf{u}, \end{aligned} \quad (142)$$

where \mathbf{s}_f is the unique solution of the initial value problem

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_f \mathbf{x}(t) + \mathbf{B}_f \mathbf{u}(t), \\ \mathbf{x}(0) &= \mathbf{s}_f, \end{aligned} \quad (143)$$

and by looking at $\text{act}_\rightarrow(g)$ we have

$$\begin{aligned} \text{act}_\rightarrow(g) : C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^n) &\rightarrow C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^o), \\ \mathbf{u} &\mapsto \mathbf{C}_g \mathbf{s}_g + \mathbf{D}_g \mathbf{u}, \end{aligned} \quad (144)$$

where \mathbf{s}_g is the unique solution of the initial value problem

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_g \mathbf{x}(t) + \mathbf{B}_g \mathbf{u}(t), \\ \mathbf{x}(0) &= \mathbf{s}_g. \end{aligned} \quad (145)$$

Clearly, considering $\text{act}_\rightarrow(f) \circ \text{act}_\rightarrow(g)$ is equivalent to considering $\mathbf{u}_g = \text{act}_\rightarrow(f)(\mathbf{u}_f)$. By substitution into (144) we obtain

$$\begin{aligned} \text{act}_\rightarrow(f) \circ \text{act}_\rightarrow(g) : C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^m) &\rightarrow C^1(\mathbb{R}_{\geq 0}, \mathbb{R}^o), \\ \mathbf{u} &\mapsto [\mathbf{D}_g \mathbf{C}_f \quad \mathbf{C}_g] \begin{bmatrix} \mathbf{s}_f \\ \mathbf{s}_g \end{bmatrix} + \mathbf{D}_g \mathbf{D}_f \mathbf{u}, \end{aligned} \quad (146)$$

proving the statement. \square

Lemma 18.29. Two equivalent systems have the same standard LTI action.

Proof. Consider two equivalent LTI systems

$$\langle \mathbf{s}_{\text{f}}, \mathbf{A}_{\text{f}}, \mathbf{B}_{\text{f}}, \mathbf{C}_{\text{f}}, \mathbf{D}_{\text{f}} \rangle, \quad \langle \mathbf{s}_{\text{g}}, \mathbf{A}_{\text{g}}, \mathbf{B}_{\text{g}}, \mathbf{C}_{\text{g}}, \mathbf{D}_{\text{g}} \rangle. \quad (147)$$

The initial value problem $\langle \mathbf{s}_{\text{f}}, \mathbf{A}_{\text{f}}, \mathbf{B}_{\text{f}}, \mathbf{C}_{\text{f}}, \mathbf{D}_{\text{f}} \rangle$ poses reads:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_{\text{f}} \mathbf{x}(t) + \mathbf{B}_{\text{f}} \mathbf{u}(t), \\ \mathbf{x}(0) &= \mathbf{s}_{\text{f}}. \end{aligned} \quad (148)$$

Let $\mathbf{s}(t)$ be the solution of (148). Consider the equivalence transformation $\mathbf{r}(t) = \mathbf{T}\mathbf{s}(t)$. Then, we have

$$\begin{aligned} \dot{\mathbf{r}}(t) &= \mathbf{T}\dot{\mathbf{s}}(t) \\ &= \mathbf{T}\mathbf{A}_{\text{f}}\mathbf{s}(t) + \mathbf{T}\mathbf{B}_{\text{f}}\mathbf{u}(t) \\ &= \mathbf{T}\mathbf{A}_{\text{f}}\mathbf{T}^{-1}\mathbf{r}(t) + \mathbf{T}\mathbf{B}_{\text{f}}\mathbf{u}(t) \\ &= \mathbf{A}_{\text{g}}\mathbf{r}(t) + \mathbf{B}_{\text{g}}\mathbf{u}(t), \end{aligned} \quad (149)$$

and $\mathbf{s}_{\text{g}} = \mathbf{T}\mathbf{s}_{\text{f}}$. Therefore, the action of the system $\langle \mathbf{s}_{\text{g}}, \mathbf{A}_{\text{g}}, \mathbf{B}_{\text{g}}, \mathbf{C}_{\text{g}}, \mathbf{D}_{\text{g}} \rangle$ is:

$$\begin{aligned} \text{act}_{\text{g}}(\mathbf{u}(t)) &= \mathbf{C}_{\text{g}}\mathbf{r}(t) + \mathbf{D}_{\text{g}}\mathbf{u}(t) \\ &= \mathbf{C}_{\text{f}}\mathbf{T}^{-1}\mathbf{T}\mathbf{s}(t) + \mathbf{D}_{\text{f}}\mathbf{u}(t) \\ &= \mathbf{C}_{\text{f}}\mathbf{s}(t) + \mathbf{D}_{\text{f}}\mathbf{u}(t) \\ &= \text{act}_{\text{f}}(\mathbf{u}(t)). \end{aligned} \quad (150)$$

□

Remark 18.30. Two LTI systems with the same LTI category action are not necessarily equivalent.

Proof. For a simple counterexample, consider the LTI system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u}(t) \\ \mathbf{y}(t) &= [1 \ 0] \mathbf{x}(t) + \mathbf{u}(t). \end{aligned} \quad (151)$$

The LTI category action of this system will be the same as the one of any system

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \begin{bmatrix} 1 & 0 \\ 0 & \alpha \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u}(t) \\ \mathbf{y}(t) &= [1 \ 0] \mathbf{x}(t) + \mathbf{u}(t), \end{aligned} \quad (152)$$

where $\alpha \in \mathbb{R}$, because the output \mathbf{y} generation does not care about the second component of \mathbf{x} . However, there is no linear, invertible transformation which relates these systems. We note that control theory has developed tools to deal with hidden parts of the dynamics. Examples are the notion of controllability (ability of an actuator/input to control all states of a system) and observability (ability to estimate all states of the system). □



19. Translation

In this chapter we discuss *functors*, which map from one category to another and “preserve the structure”.

19.1 Layers of abstraction	278
19.2 Semifunctors	279
19.3 Functors	280
19.4 More examples of functors	283
19.5 Categorical Databases	287

19.1. Layers of abstraction

We can think of a given category **C** as a “compositional world”: inside **C** we have objects, morphisms between them, and a way to talk about composing morphisms. Now we will zoom out a level, and consider different categories – different worlds – simultaneously, and how to relate them to each other.

The most basic notion of how to “map” one category to another is given by the concept of a *functor*.

Just like a morphism

$$f : \textcolor{red}{X} \rightarrow \textcolor{red}{Y} \tag{1}$$

is an arrow between objects in a category, a functor

$$\textcolor{violet}{F} : \mathbf{C} \rightarrow \mathbf{D} \tag{2}$$

is an arrow between two categories. In fact, we will see in the next chapters that functors are morphisms in a *category of categories*.

19.2. Semifunctors

Definition 19.1 (Semifunctor)

Given semicategories \mathbf{C} and \mathbf{D} , a *semifunctor* $F : \mathbf{C} \rightarrow \mathbf{D}$ from \mathbf{C} to \mathbf{D} is:

Constituents

1. A function

$$F_\bullet : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{D}}. \quad (3)$$

2. For every pair of objects $X, Y \in \text{Ob}_{\mathbf{C}}$, a function

$$F_\rightarrow : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{D}}(F_\bullet(X); F_\bullet(Y)). \quad (4)$$

Conditions

1. The function F_\rightarrow is compatible with the composition operations in the source and target category, respectively:

$$\frac{f : X \rightarrow_{\mathbf{C}} Y \quad g : Y \rightarrow_{\mathbf{C}} Z}{F_\rightarrow(f \circ_{\mathbf{C}} g) = F_\rightarrow(f) \circ_{\mathbf{D}} F_\rightarrow(g)}. \quad (5)$$

This situation is depicted graphically in Fig. 1a. It is common to overload the notation and use F to denote not only the whole functor, both also both of its constituent functions F_\bullet and F_\rightarrow . The diagram with this overloaded “synthetic notation” is in Fig. 1b.

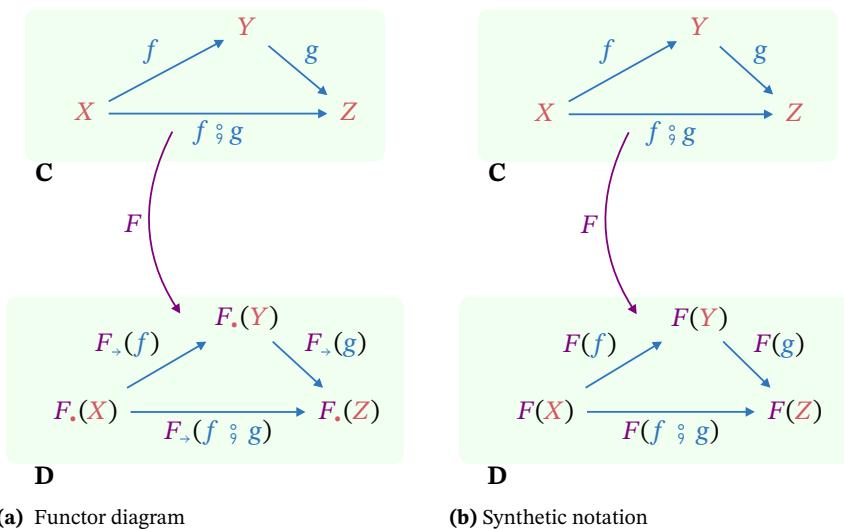


Figure 1.: Commuting diagrams for semifunctors, with verbose notation (left) and synthetic notation (right).

19.3. Functors

For categories, we have the stronger concept of functor. Categories have identities, and functors are required to preserve the identities.

Definition 19.2 (Functor)

A *functor* from category **C** to a category **D** is a semifunctor $F : \mathbf{C} \rightarrow \mathbf{D}$ that satisfies the condition

$$F_{\rightarrow}(\text{id}_{\mathbf{X}}) = \text{id}_{F_{\rightarrow}(\mathbf{X})} \quad (6)$$

for all objects \mathbf{X} in **C**.

Example 19.3 (Powerset functor). We define a functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ which maps each set to its power set. In other words, $F_{\cdot}(\mathbf{A}) = \text{Pow } \mathbf{A}$ for any set \mathbf{A} . On the level of morphisms, given a function $f : \mathbf{A} \rightarrow \mathbf{B}$, we define

$$\begin{aligned} F_{\rightarrow}(f) : \text{Pow } \mathbf{A} &\rightarrow \text{Pow } \mathbf{B} \\ \mathbf{C} &\mapsto \{f(c) \mid c \in \mathbf{C}\}. \end{aligned} \quad (7)$$

Here is a concrete illustration. Consider the two sets $\mathbf{A} = \{\text{☕}, \text{🥨}, \text{♦}\}$ and $\mathbf{B} = \{\text{🧀}, \text{🍇}, \text{🍎}\}$. Applying the functor to \mathbf{A} gives

$$F_{\cdot}(\mathbf{A}) = \{\emptyset, \{\text{☕}\}, \{\text{🥨}\}, \{\text{♦}\}, \{\text{☕}, \text{🥨}\}, \{\text{☕}, \text{♦}\}, \{\text{🥨}, \text{♦}\}, \{\text{☕}, \text{🥨}, \text{♦}\}\} \quad (8)$$

and applying it to \mathbf{B} gives

$$F_{\cdot}(\mathbf{B}) = \{\emptyset, \{\text{🧀}\}, \{\text{🍇}\}, \{\text{🍎}\}, \{\text{🧀}, \text{🍇}\}, \{\text{🧀}, \text{🍎}\}, \{\text{🍇}, \text{🍎}\}, \{\text{🧀}, \text{🍇}, \text{🍎}\}\}. \quad (9)$$

Furthermore, consider the map

$$\begin{aligned} f : \mathbf{A} &\rightarrow \mathbf{B}, \\ \text{☕} &\mapsto \text{🧀}, \\ \text{🥨} &\mapsto \text{🍇}, \\ \text{♦} &\mapsto \text{🍎}. \end{aligned} \quad (10)$$

This would for instance give $F_{\rightarrow}(f)(\{\text{☕}, \text{🥨}\}) = \{f(\text{☕}), f(\text{🥨})\} = \{\text{🧀}, \text{🍇}\}$.

Now let us check that F so-defined really is a functor.

First let us check that it is compatible with composition. Consider functions $f : \mathbf{A} \rightarrow \mathbf{B}$, $g : \mathbf{B} \rightarrow \mathbf{C}$. On the one hand we have

$$F(f \circ g)(\mathbf{C}) = \{g(f(c)) \mid c \in \mathbf{C}\}, \quad (11)$$

and on the other hand

$$\begin{aligned} (F(f) \circ F(g))(\mathbf{C}) &= F(g)(\{f(c) \mid c \in \mathbf{C}\}) \\ &= \{g(d) \mid d \in \{f(c) \mid c \in \mathbf{C}\}\} \\ &= \{g(f(c)) \mid c \in \mathbf{C}\}. \end{aligned} \quad (12)$$

Second, let us check that F is compatible with identity morphisms. We have

$$\begin{aligned} F(\text{id}_{\mathbf{A}})(\mathbf{C}) &= \{\text{id}_{\mathbf{A}}(c) \mid c \in \mathbf{C}\} \\ &= \text{id}_{F(\mathbf{C})}. \end{aligned} \quad (13)$$

Example 19.4. There is a functor

$$\text{List} : \mathbf{Set} \rightarrow \mathbf{Mon} \quad (14)$$

from the category of sets to the category of monoids, defined as follows.

Given a set \mathbf{A} , the functor returns a specific monoid

$$\text{List}(\mathbf{A}) := \langle \text{List } \mathbf{A}, [\]_{\mathbf{A}}, \circ_{[\]} \rangle. \quad (15)$$

Given a map $f : \mathbf{A} \rightarrow \mathbf{B}$, we have

$$\begin{aligned} \text{List}(f) : \text{List } \mathbf{A} &\rightarrow \text{List } \mathbf{B}, \\ [\dots, a_i, \dots] &\mapsto [\dots, f(a_i), \dots], \end{aligned} \quad (16)$$

which applies f entry-wise in the list. The empty list in $\text{List } \mathbf{A}$ is mapped to the empty list in $\text{List } \mathbf{B}$. List is a functor, because identity functions in \mathbf{Set} are mapped to the corresponding identity morphisms on lists, and

$$\begin{aligned} \text{List}(f \circ g)([\dots, a_i, \dots]) &= [\dots, (f \circ g)(a_i), \dots] \\ &= [\dots, g(f(a_i)), \dots] \\ &= \text{List}(g)([\dots, f(a_i), \dots]) \\ &= (\text{List}(f) \circ \text{List}(g))([\dots, a_i, \dots]). \end{aligned} \quad (17)$$

Graded exercise F.2 (DifferentiationFunctor)

Consider the category \mathbf{Euc}_* from Graded Exercise E.3, as well as the category $\mathbf{Vect}_{\mathbb{R}}$ of real vector spaces studied in Graded Exercise E.2. In this exercise we will define a functor $F : \mathbf{Euc}_* \rightarrow \mathbf{Vect}_{\mathbb{R}}$ corresponding to differentiation and it is your task to check that it is in fact a functor.

F on objects: $F_*(\langle \mathbb{R}^n, x \rangle) = \mathbb{R}^n$.

F on morphisms: given a morphism $f : \langle \mathbb{R}^n, x \rangle \rightarrow \langle \mathbb{R}^m, y \rangle$ in \mathbf{Euc}_* , the linear map $F_*(f)$ is the derivative

$$Df|_x : \mathbb{R}^n \rightarrow \mathbb{R}^m, \quad (18)$$

which is typically represented by the Jacobian matrix.

The intuition for the on-objects part of this functor is that $\langle \mathbb{R}^n, x \rangle$ is mapped to the tangent space of “vectors starting at x ”, which is isomorphic to \mathbb{R}^n . On the level of morphisms, the differential of a function f at x maps vectors starting at x to vectors starting at $f(x)$.

Graded exercise F.3 (FixedPointFunctor)

We will propose a functor $F : \mathbf{EndSet} \rightarrow \mathbf{Set}$ from the category \mathbf{EndSet} defined in Graded Exercise E.4 to the category of sets and functions. Your task is to check if this is a functor.

On objects: given an object $\langle \mathbf{A}, \varphi \rangle$ of \mathbf{EndSet} , we define $F_*(\langle \mathbf{A}, \varphi \rangle) = \text{Fix}(\varphi)$, where

$$\text{Fix}(\varphi) = \{x \in \mathbf{A} \mid \varphi(x) = x\} \quad (19)$$

is the set of fixed points of φ .

On morphisms: given a morphism $f : \langle \mathbf{A}, \varphi \rangle \rightarrow \langle \mathbf{B}, \psi \rangle$ of \mathbf{EndSet} , we define

$$F_*(f) = f|_{\text{Fix}(\varphi)}. \quad (20)$$

In other words, we restrict f to the subset $\text{Fix}(\varphi) \subseteq \mathbf{A}$.

A functor from a category to itself is called an *endofunctor*. The simplest example of an endofunctor is the identity functor.

Identity functor

Definition 19.5 (Identity (semi)functor)

For any (semi)category \mathbf{C} , we can define the *identity (semi)functor*

$$\text{id}_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbf{C}, \quad (21)$$

which maps each object to itself and each morphism to itself.

Exercise 41. Check that the identity functor is a functor.

See solution on page 303.

Functors generalize monoid morphisms.

We have seen that monoids can be viewed as categories, where the elements of a monoid play the role of morphisms. From this point of view, a functor corresponds to a morphism of monoids.

Functors generalize monotone maps.

Recall that a single poset $\mathbf{P} = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ can be viewed as a category $\mathbf{C}(\mathbf{P})$, in which each element of the poset is an object, and there is a morphism between two objects x and y if and only if $x \leq_{\mathbf{P}} y$ (Def. 14.16).

Lemma 19.6. A monotone map between posets \mathbf{P}, \mathbf{Q} is the same thing as a functor between the “posetal categories” $\mathbf{C}(\mathbf{P})$ and $\mathbf{C}(\mathbf{Q})$.

Proof. We start by specifying the functor F and two posetal categories $\mathbf{C}(\mathbf{P})$ and $\mathbf{C}(\mathbf{Q})$. We first specify the action of F on objects (elements of a poset considered as objects of the posetal category) and on morphisms (order relations considered as morphisms of the posetal category). A monotone function maps each element of a poset $x \in \mathbf{P}$ to $F(x) \in \mathbf{Q}$, and it guarantees that for every $x, y \in \mathbf{P}$, if $x \leq_{\mathbf{P}} y$ then $F(x) \leq_{\mathbf{Q}} F(y)$. We now need to check the two conditions that a functor must satisfy. First, consider the identity morphism for $x \in \mathbf{P}$, namely $x \leq_{\mathbf{P}} x$. The application of the map F results in the condition $F(x) \leq_{\mathbf{Q}} F(x)$, which is the identity morphism on \mathbf{Q} . Second, morphisms $x \leq_{\mathbf{P}} y$ and $y \leq_{\mathbf{P}} z$ in \mathbf{P} , by applying the map F to the morphism composition $x \leq_{\mathbf{P}} z$ we obtain $F(x) \leq_{\mathbf{Q}} F(z)$, which is the same as the composition of $F(x) \leq_{\mathbf{Q}} F(y)$ and $F(y) \leq_{\mathbf{Q}} F(z)$. \square

Functors generalize semicategory actions

Semi-functors are a generalization of the various semigroup morphisms that we saw in the previous chapter.

In particular, they are a generalization of semicategory actions (Def. 18.10), which we can re-define as follows.

Definition 19.7 (Semicategory actions, redefined)

A semicategory action of \mathbf{C} is a semifunctor $F : \mathbf{C} \rightarrow \mathbf{Set}$.

19.4. More examples of functors

Example 19.8 (Equivalence Classes). Consider the category **EquivRel** from Def. 14.11. We define a functor

$$\mathbf{F} : \mathbf{EquivRel} \rightarrow \mathbf{Set} \quad (22)$$

as follows.

Given an object $\langle \mathbf{A}, \sim_{\mathbf{A}} \rangle$ of **EquivRel** (that is, a set equipped with an equivalence relation), we define $\mathbf{F}_*(\langle \mathbf{A}, \sim_{\mathbf{A}} \rangle) = \mathbf{A}/\sim_{\mathbf{A}}$ to be the quotient of \mathbf{A} by $\sim_{\mathbf{A}}$.

Recall that $\mathbf{A}/\sim_{\mathbf{A}}$ is the set of equivalence classes of $\sim_{\mathbf{A}}$. Its elements are the subsets of \mathbf{A} which form the partition of \mathbf{A} induced by $\sim_{\mathbf{A}}$. Each such subset is the set of all elements of \mathbf{A} which are mutually equivalent to each other according to $\sim_{\mathbf{A}}$. For any element $x \in \mathbf{A}$, the equivalence class it belongs to is denoted $[x]$ and in this case x is called a *representative* of the equivalence class $[x]$.

To define \mathbf{F}_* , let $f : \langle \mathbf{A}, \sim_{\mathbf{A}} \rangle \rightarrow \langle \mathbf{B}, \sim_{\mathbf{B}} \rangle$ be a morphism in **EquivRel**. We let

$$\mathbf{F}_*(f) : \mathbf{A}/\sim_{\mathbf{A}} \rightarrow \mathbf{Set}\mathbf{B}/\sim_{\mathbf{B}}, [x] \mapsto [f(x)]. \quad (23)$$

It may be readily checked that this function is well-defined, irrespective of the (arbitrary) choice of element x used to represent a given equivalence class $[x]$.

Exercise42. Prove that the functor defined in Example 19.8 is in fact a functor.

See solution on page 304.

Example 19.9 (Double dual). Let **Vect**_ℝ be the category whose objects are all real vector spaces and whose morphisms are ℝ-linear maps. Composition is the usual composition of linear maps.

There is an endofunctor $\mathbf{F} : \mathbf{Vect}_{\mathbb{R}} \rightarrow \mathbf{Vect}_{\mathbb{R}}$ whose action on objects is

$$\mathbf{F}(V) = V^{**}. \quad (24)$$

(Recall that $V^{**} = \{\text{linear maps } V^* \rightarrow \mathbb{R}\} = \text{Hom}_{\mathbf{Vect}_{\mathbb{R}}}(V^*; \mathbb{R})$). The action of \mathbf{F} on morphisms is as follows. Given a linear map $f : V \rightarrow W$, we can write

$$\begin{aligned} \mathbf{F}(f) : V^{**} &\rightarrow W^{**}, \\ \xi &\mapsto [l \mapsto \xi(f \circ l)]. \end{aligned} \quad (25)$$

Graded exercise F.4 (DoubleDualFunctor)

Prove that \mathbf{F} as defined in Example 19.9 is in fact a functor.

Example 19.10. Let \mathbf{G} be a group. We've seen that we can view \mathbf{G} as a category with just a single object (denote that object by \star) and where the morphisms are the elements of \mathbf{G} (with composition given by the group's composition operation).

We claim that specifying a functor $\mathbf{G} \rightarrow \mathbf{Set}$ is "the same thing" as specifying a set \mathbf{A} together with a (covariant) action of \mathbf{G} on \mathbf{A} .

Exercise43. Prove the claim made in Example 19.10.

See solution on page 304.

Graded exercise F.5 (MultiplicationWithASet)

Let \mathbf{S} be a fixed set. We define a functor $\mathbf{F} : \mathbf{Set} \rightarrow \mathbf{Set}$ which acts on objects by

$$\mathbf{F}_*(\mathbf{A}) = \mathbf{A} \times \mathbf{S} \quad (26)$$

and on morphisms by

$$\textcolor{violet}{F}_*(\textcolor{blue}{f}) = f \times \text{id}_{\textcolor{brown}{S}}. \quad (27)$$

Prove that this is indeed a functor.

Graded exercise F.6 (ExponentiationWithASet)

Let $\textcolor{brown}{S}$ be a fixed set. We can define a functor $\textcolor{violet}{F} : \textbf{Set} \rightarrow \textbf{Set}$ which acts on objects by

$$\textcolor{violet}{F}_*(\textcolor{brown}{A}) = \textcolor{brown}{A}^{\textcolor{brown}{S}}. \quad (28)$$

(Recall that $\textcolor{brown}{A}^{\textcolor{brown}{S}}$ denotes the set of functions from $\textcolor{brown}{S}$ to $\textcolor{brown}{A}$.) For the action on morphisms, suppose we have a function $\textcolor{violet}{F} : \textcolor{brown}{A} \rightarrow \textcolor{brown}{B}$. Then

$$\begin{aligned} \textcolor{violet}{F}_*(\textcolor{blue}{f}) &: \textcolor{brown}{A}^{\textcolor{brown}{S}} \rightarrow \textcolor{brown}{B}^{\textcolor{brown}{S}}, \\ \varphi &\mapsto \varphi \circ \textcolor{blue}{f}. \end{aligned} \quad (29)$$

Prove that this does really define a functor.

Graded exercise F.7 (GraphsViaFunctors)

Consider the following category, which has two objects, $\textcolor{red}{V}$ and $\textcolor{brown}{A}$, and four morphisms: besides the identity morphisms, there are two morphisms, s and t , from $\textcolor{brown}{A}$ to $\textcolor{red}{V}$. See Fig. 2. Call this category \textbf{G} .

Can you explain the following statement? “Specifying a functor $\textbf{G} \rightarrow \textbf{Set}$ is the “same thing” as specifying a directed graph”.

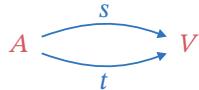


Figure 2.

Graded exercise F.8 (UpperSetsViaFunctors)

Recall that **Bool** denotes the category with two objects, \top and \perp , and with precisely one non-identity morphism which goes from \perp to \top . Let \textbf{P} be a poset. View it as a category \textbf{P} , and let $\textcolor{violet}{F} : \textbf{P} \rightarrow \textbf{Bool}$ be a functor. In other words, $\textcolor{blue}{F} = \textcolor{violet}{F}_*$ is a monotone function. Prove that the set

$$\textcolor{brown}{S} := \{p \in \textbf{P} \mid \textcolor{blue}{F}(p) = \top\} \subseteq \textbf{P} \quad (30)$$

is an upper set.

Graded exercise F.9 (CartProdAsFunctor)

Recall that, given categories \textbf{C} and \textbf{D} , we can form the product category $\textbf{C} \times \textbf{D}$. In this exercise we will use this construction in a situation where we consider the product of the category **Set** with itself.

Your task in this exercise is to show that there is a functor $\textcolor{violet}{F} : \textbf{Set} \times \textbf{Set} \rightarrow \textbf{Set}$ which is defined by the operation of “taking the cartesian product of sets and functions”.

Concretely, given any two sets $\textcolor{brown}{A}, \textcolor{brown}{B}$, let

$$\textcolor{violet}{F}_*((\textcolor{brown}{A}, \textcolor{brown}{B})) := \textcolor{brown}{A} \times \textcolor{brown}{B}$$

(this defines $\textcolor{violet}{F}$ on the level of objects) and given any two functions $\textcolor{blue}{f} : \textcolor{brown}{A} \rightarrow \textcolor{brown}{C}, \textcolor{blue}{g} : \textcolor{brown}{B} \rightarrow \textcolor{brown}{D}$, let

$$\textcolor{violet}{F}_*((\textcolor{blue}{f}, \textcolor{blue}{g})) := \textcolor{blue}{f} \times \textcolor{blue}{g}$$

(this defines $\textcolor{violet}{F}$ on the level of morphisms).

To show that $\textcolor{violet}{F}$ is in fact a functor, check the two conditions in the defintion of

a functor, namely that F is compatible with composition and with identities.

Graded exercise F.10 (ProbabilityFunctor)

In this exercise we will define a functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ and your task is to check that it is in fact a functor.

F on objects: for any set \mathbf{A} we define $F_*(\mathbf{A})$ to be the set of *finitely supported probability measures* on \mathbf{A} . These are functions $p : \mathbf{A} \rightarrow [0, 1]$ with only finitely-many non-zero values and such that these sum to 1:

$$\sum_{x \in \mathbf{A}} p(x) = 1. \quad (31)$$

F on morphisms: for any function $f : \mathbf{A} \rightarrow \mathbf{B}$, the function $F_*(f) : F_*(\mathbf{A}) \rightarrow F_*(\mathbf{B})$ is defined by

$$\begin{aligned} F_*(f)(p) &: \mathbf{B} \rightarrow [0, 1], \\ y &\mapsto \sum_{x \in f^{-1}(y)} p(x). \end{aligned} \quad (32)$$

The following is a helpful visualization. We may think of finitely-supported probability measures on \mathbf{A} as “finite normalized histograms over the elements of \mathbf{A} ”, and the functor F moves the columns of a histogram on \mathbf{A} “along” f to make a histogram on \mathbf{B} , stacking columns on top of each other whenever they end up over the same element. (Credit: this nice description is taken from Paolo Perrone’s *Notes on category theory*.)

Example 19.11. Let \mathbf{C} be any category. There is a functor

$$F : \mathbf{C}^{\text{op}} \times \mathbf{C} \rightarrow \mathbf{Set} \quad (33)$$

defined on objects by

$$F_*(\langle X, Y \rangle) = \text{Hom}_{\mathbf{C}}(X, Y). \quad (34)$$

To see how F is defined on morphisms, let $f^{\text{op}} : U \rightarrow_{\mathbf{C}^{\text{op}}} X$ and $g : Y \rightarrow_{\mathbf{C}} Z$ be morphisms in \mathbf{C}^{op} and \mathbf{C} , respectively, and recall that $f^{\text{op}} : U \rightarrow_{\mathbf{C}^{\text{op}}} X$ is, by definition, a morphism $f : X \rightarrow_{\mathbf{C}} U$. We define $F_*(\langle f^{\text{op}}, g \rangle)$ to be the function

$$\begin{aligned} F_*(\langle f^{\text{op}}, g \rangle) &: \text{Hom}_{\mathbf{C}}(X, Y) \rightarrow \text{Hom}_{\mathbf{C}}(U, Z), \\ \varphi &\mapsto f \circ \varphi \circ g. \end{aligned} \quad (35)$$

Graded exercise F.11 (HomFunctor)

Prove that the hom-functor defined in Example 19.11 is in fact a functor.

Planning as the search for a functor

Example 19.12. Recall the category **Berg** introduced in Section 15.2 and define a category **Plans** where objects are activities and morphisms describe activities order constraints, illustrated in Fig. 3 (left).

For instance, there is a morphism from “mountain lodge” to “panoramic lake”, which describes the plan of going from the lodge area to the lake area. We call such morphisms *plans*. Plans can be composed via concatenation. For instance, given a plan to go from “mountain lodge” to “panoramic lake”, and a plan to go

from “panoramic lake” to “peak”, their composition is the plan of going from “mountain lodge” to the “peak”, passing through the “panoramic lake”.

When we talk about “planning” in this context, we refer to the action of finding a functor F from **Plans** to **Berg**. The objects of **Berg** are tuples $\langle p, v \rangle$, where p represent coordinates of a specific location and $v \in \mathbb{R}^3$ represents velocities. Morphisms in **Berg** are paths that connect locations. For the sake of our planning, we can identify areas of the mountain as sets of locations. Such areas are, for instance, the “mountain lodge”, “panoramic lake”, and the “peak” (note that the “peak” represents an area corresponding to a single location). Given some plans as in Fig. 3 (left), we want to find a map P which maps each object in **Plans** (activity) to an object of **Berg** (specific location and velocity). Similarly, it must map each morphism in **Plans** (activity order constraints) to a morphism in **Berg** (specific paths). This is illustrated in Fig. 3.

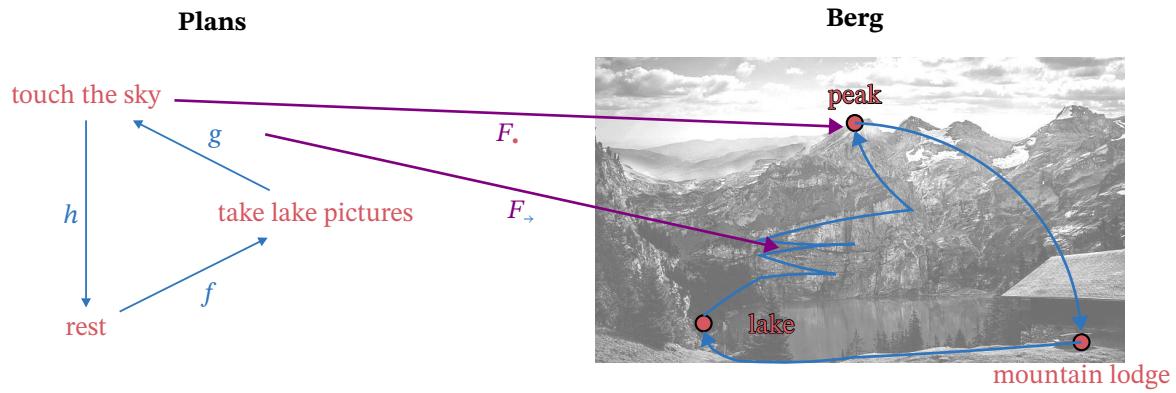


Figure 3.: Planning functor.

19.5. Categorical Databases

In this section we look at how a relational database can be modeled using the notion of functors. This view of databases is due to Spivak [27, 28].

To model a database, we need to model two things:

1. The *schema* of the database is its structure. This includes which tables are present, what fields are included in each table, and what constraints there are among tables.
2. The *data* that resides in the database.

Spivak's idea is that categories can be used for modeling the schema, not the data. The data can be modeled as a *functor*.



20. Specialization

Oftentimes, two categories are in a special relation: one is a specialization of the other. There are two interesting situations.

First, a category can be a *subcategory* of another if it contains a subset of the objects and morphisms of the other, in such a way that it is closed to composition.

Second, there is a generalization called *embedding*: objects and morphisms are different, but we can find a functor that “draws” the first category in the second.

20.1 Subcategories	290
20.2 Subcategories of endomorphisms	291
20.3 Other examples	292
20.4 Subcategories of Berg	293

20.1. Subcategories

Definition 20.1 (Subcategory)

A *sub(semi)category* \mathbf{D} of a (semi)category \mathbf{C} is a category for which:

1. All the objects in $\text{Ob}_{\mathbf{D}}$ are in $\text{Ob}_{\mathbf{C}}$;
2. For any two objects $X, Y \in \text{Ob}_{\mathbf{D}}$, the morphisms of \mathbf{D} between them are a subset of the morphisms of \mathbf{C} :

$$\text{Hom}_{\mathbf{D}}(X; Y) \subseteq \text{Hom}_{\mathbf{C}}(X; Y); \quad (1)$$

3. If $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ in \mathbf{D} , then the composite $f ; g$ in \mathbf{C} is in \mathbf{D} and represents the composite in \mathbf{D} .
4. (Categories) If $X \in \text{Ob}_{\mathbf{D}}$, then the identity id_X in $\text{Hom}_{\mathbf{C}}(X; X)$ is also in $\text{Hom}_{\mathbf{D}}(X; X)$ and acts as its identity morphism.

Subcategories of Rel and Set

Two important examples of subcategory are the following.

Example 20.2 (**FinSet**). **FinSet** is the category of finite sets and all functions between them. It is a subcategory of the category **Set** of sets and functions. While an object $X \in \text{Ob}_{\text{Set}}$ is a set with arbitrary cardinality, $\text{Ob}_{\text{FinSet}}$ only includes sets which have finitely many elements. Objects of **FinSet** are in **Set**, but the converse is not true. Furthermore, given $X, Y \in \text{Ob}_{\text{FinSet}}$, we take $\text{Hom}_{\text{FinSet}}(X; Y) = \text{Hom}_{\text{Set}}(X; Y)$.

Example 20.3 (**Set** and **Rel**). The category **Set** is a subcategory of **Rel**. To show this, we need to prove the conditions presented in Def. 20.1.

1. In both **Rel** and **Set**, the collection of objects is all sets.
2. Given $X, Y \in \text{Ob}_{\text{Set}}$, we know that $\text{Hom}_{\text{Set}}(X; Y) \subseteq \text{Hom}_{\text{Rel}}(X; Y)$, since all functions between sets X, Y are a particular subset of all relations between X, Y .
3. Let $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ be relations which are functions. We need to show that their composition in **Rel**, expressed as $R ; S \subseteq X \times Z$, is again a function. This was proven in Lemma 4.8.
4. For each $X \in \text{Ob}_{\text{Set}}$, the identity relation $\text{id}_X = \{(x, x') \in X \times X \mid x = x'\}$ corresponds to the identity function $\text{id}_X : X \rightarrow X$ in **Set**.

20.2. Subcategories of endomorphisms

Definition 20.4 (Drawings)

There exists a category **Draw** in which:

1. An object in $\alpha \in \text{Ob}_{\text{Draw}}$ is a black-and-white drawing, that is a function $\alpha : \mathbb{R}^2 \rightarrow \text{Bool}$.
2. A morphism in $\text{Hom}_{\text{Draw}}(\alpha; \beta)$ between two drawings α and β is an invertible map $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ such that $\alpha(x) = \beta(f(x))$.
3. The identity function at any object α is the identity map on \mathbb{R}^2 .
4. Composition is given by function composition.

Exercise 44. Check whether just considering

- ▷ affine invertible transformations, or
- ▷ rototranslations, or
- ▷ scalings, or
- ▷ translations, or
- ▷ rotations,

as morphisms forms a subcategory of **Draw**.

See solution on page 304.

20.3. Other examples of subcategories in engineering

In engineering, it is very common to look at specific types of functions; in many cases, the properties of a certain type of function are preserved by function composition, and so they form a category.

InjSet is a subcategory of Set

Example 20.5. We can define a category **InjSet** that has the same objects as **Set** but restricts the morphisms to be *injective functions* (Def. 3.16). We want to show that **InjSet** is a subcategory of **Set**. Composition and identity morphisms are defined as in **Set**.

Since $\text{Ob}_{\text{InjSet}} = \text{Ob}_{\text{Set}}$, the first condition of Def. 20.1 is satisfied. Injective functions are a particular type of functions: this satisfies the second condition. Given $X \in \text{Ob}_{\text{InjSet}}$, the identity morphism $\text{id}_X \in \text{Hom}_{\text{Set}}(X; X)$ corresponds to the identity morphism in $\text{Hom}_{\text{InjSet}}(X; X)$: the identity function is injective. This proves the fourth condition. To check the third condition, consider two morphisms $f \in \text{Hom}_{\text{Set}}(X; Y)$, $g \in \text{Hom}_{\text{Set}}(Y; Z)$ such that $f \in \text{Hom}_{\text{InjSet}}(X; Y)$ and $g \in \text{Hom}_{\text{InjSet}}(Y; Z)$. From the injectivity of f, g , we know that given $x, x' \in X$,

$$\frac{\underline{\underline{f(x) = f(x')}}}{x = x'}, \quad (2)$$

and $y, y' \in Y$,

$$\frac{\underline{\underline{g(y) = g(y')}}}{y = y'}. \quad (3)$$

Furthermore, we have:

$$\begin{aligned} &\underline{\underline{(f \circ g)(x) = (f \circ g)(x')}} \\ &\underline{\underline{f(x) = f(x')}} \\ &x = x' \end{aligned} \quad (4)$$

which proves the third condition of Def. 20.1: the composition of injective functions is injective.

20.4. Subcategories of Berg

Recall the category **Berg** presented in Section 15.2. In the following, we want to give both a positive and a negative example of subcategories related to **Berg**.

We start our discussion by introducing a *limited* version of **Berg**, called **Berg**_α, which only considers paths (morphisms) in **Berg**, whose steepness does not exceed the critical value $\alpha \in [0, 1]$. Is **Berg**_α a subcategory of **Berg**? We check the different conditions:

1. The constraint on the maximum steepness restricts the objects which are acceptable in **Berg**_α via the identity morphisms of **Berg**. Indeed, recall that given an object $\langle p, v \rangle \in \text{Ob}_{\text{Berg}}$, the identity morphism is defined as $1_{\langle p, v \rangle} = \langle \gamma, 0 \rangle$, with $\gamma(0) = p$ and $\dot{\gamma}(0) = v$. The steepness is computed via v . In particular, **Berg**_α will only contain objects whose identity morphisms do not exceed the steepness constraint. In other words $\text{Ob}_{\text{Berg}} \subseteq \text{Ob}_{\text{Berg}}$.
2. For $X, Y \in \text{Ob}_{\text{BergAma}}$, we know that paths satisfying the steepness constraint are specific paths in **Berg**: $\text{Hom}_{\text{Berg}} \subseteq \text{Hom}_{\text{Berg}}$.
3. Given two morphisms f, g which can be composed in **Berg**_α, the maximum steepness of their composition $f ; g$ is given by:

$$\text{MaxSteep}(f ; g) = \max \{\text{MaxSteep}(f), \text{MaxSteep}(g)\} < \alpha. \quad (5)$$

4. The identity morphisms in **Berg** which satisfy the steepness constraint are, by definition, in **Berg**_α and they act as identities there.

This shows that **Berg**_α is a subcategory of **Berg**.

What would an example of non-subcategory of **Berg** be? We could try defining a new category **BergLazy**, which now discriminates morphisms based on the lengths of the paths they represent. For instance, assume that as amateur hikers, we don't want to consider morphisms which are more than 1 km long. By concatenating two paths (morphisms) of length 0.6 km in **BergLazy**, the resulting composition will be 1.2 km, violating the posed constraint and hence not being in **BergLazy**. This violates the third property of Def. 20.1.



21. Syntax and semantics

To write...

21.1 Specification verses behavior . . . 296

Raclette is a Swiss dish, based on heating cheese and scraping off the melted part. In Switzerland, raclette is typically served with potatoes, cornichons, pickled onions, and Fendant wine.

21.1. Specification verses behavior Systems

In the following discussion we will often use the word “system”. This term here is not a precise one; however, it is often a way of saying something like “engineering-flavored morphism” or “system component” or “dynamical system”, but without necessarily getting precise about which kind of things we are exactly referring to or which (semi)categories might be involved.

We will think of a system as something that has input and output ports with which it can interact with other systems (or its broader environment), and that a system in some way establishes a relationship between input and output signals. This relationship might be a deterministic, causal relationship – inputs determining outputs – or it might be another form of lawfulness.

A typical notation to depict a system diagrammatically is to draw it as a box, with externally extended wires indicating the input and output ports. We will usually orient such diagrams horizontally, and assume that the left-hand wires indicate input ports, and right-hand wires indicate output ports.

Composing systems

We assume that our concept of system is compositional: systems can be connected together to build larger, composite systems.

An illustrative example of systems are Moore machines, which we discussed in some length in Chapter 18. Given a Moore machine

$$\langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle \quad (1)$$

with an input set \mathbf{U} and an output \mathbf{Y} . When the output set of one machine matches the input set of another, we have already seen how to compose Moore machines in series such that the result is again a Moore machine.

System specification vs. system behavior

We will make a distinction between ways of specifying a system, and ways of describing how a system might behave.

To see what we mean, consider the example of Moore machines. A way to specify a Moore machine, according to our Def. 18.5, is to specify a tuple of the form (1).

On the other hand, we saw that Moore machines can act on sequences or lists of signals in various ways. We think of these actions as encoding “behaviors” that a Moore machine can exhibit. The idea is that an action encodes a way that a Moore machine “does something” – how it relates input signals to output signals.

Strictly speaking, we will think of just the specific relation between inputs and outputs as a behavior, and an action is a way of associating specified machines to specified behaviors.

For example, given a machine

$$f : \mathbf{U} \rightarrow \mathbf{Y} \quad (2)$$

specified by (1), the standard action Def. 18.11 associates to it a behavior

$$\text{act}(f) : \text{Stream } \mathbf{U} \rightarrow \text{Stream } \mathbf{Y}. \quad (3)$$

We can study specifications of machines and possible behaviors of machines each in their own right, and we can study ways that specifications and behaviors can be connected.

In Section 18.7 below on LTI systems, we will see an example of how a single system might be specified in different ways.



22. Up the ladder of abstraction

In this section we “zoom out” one level and describe how functors are morphisms in a category of categories.

22.1 Functor composition	300
22.2 A category of categories	301
22.3 Products and sums of functors .	302

22.1. Functor composition

Definition 22.1 (Functor composition)

Consider categories \mathbf{A} , \mathbf{B} , \mathbf{C} and functors $F : \mathbf{A} \rightarrow \mathbf{B}$, $G : \mathbf{B} \rightarrow \mathbf{C}$. Functor composition is given by $F \circ G : \mathbf{A} \rightarrow \mathbf{C}$, where:

- ▷ Given $X \in \text{Ob}_{\mathbf{A}}$, we have

$$(F \circ G)(X) := G(F(X)). \quad (1)$$

- ▷ Given $f \in \text{Hom}_{\mathbf{A}}(X; Y)$, we have

$$(F \circ G)(f) := G(F(f)). \quad (2)$$

Lemma 22.2. The composition of functors is a functor.

Exercise 45. Prove Lemma 22.2.

See solution on page 305.

22.2. A category of categories

Given the existence of an identity functor and the ability of functors to compose, we can define a category of categories **Cat**. In order to avoid set-theoretic technicalities, we restrict our attention to so-called “small” categories: these are categories whose collection of objects form a set (and not a proper class).

Definition 22.3 (Category of small categories)

There is a category, called **Cat**, which is constituted of

- ▷ Objects: small categories;
- ▷ Morphisms: functors;
- ▷ Identity morphisms: identity functors;
- ▷ Composition: composition of functors.

Graded exercise F.12 (`CatProductCategorical`)

Prove that the product category $\mathbf{C} \times \mathbf{D}$ of two small categories “is” the categorical product of \mathbf{C} and \mathbf{D} within the category of small categories.

22.3. Products and sums of functors

Definition 22.4 (Product of two functors)

Given functors $F : \mathbf{A} \rightarrow \mathbf{B}$ and $G : \mathbf{C} \rightarrow \mathbf{D}$, their product is the functor

$$F \times G : \mathbf{A} \times \mathbf{C} \rightarrow \mathbf{B} \times \mathbf{D}$$

defined on objects by

$$(F \times G)(\langle X, Y \rangle) = \langle F_*(X), G_*(Y) \rangle \quad (3)$$

and on morphisms by

$$(F \times G)(\langle f, g \rangle) = \langle F_{\rightarrow}(f), G_{\rightarrow}(g) \rangle. \quad (4)$$

Definition 22.5 (Sum of two functors)

Given functors $F : \mathbf{A} \rightarrow \mathbf{B}$ and $G : \mathbf{C} \rightarrow \mathbf{D}$, their sum is the functor

$$F + G : \mathbf{A} + \mathbf{C} \rightarrow \mathbf{B} + \mathbf{D}$$

defined on objects by

$$\begin{aligned} (F + G)(\langle 1, X \rangle) &= \langle 1, F_*(X) \rangle, \\ (F + G)(\langle 2, Y \rangle) &= \langle 2, G_*(Y) \rangle, \end{aligned} \quad (5)$$

and on morphisms by

$$\begin{aligned} (F + G)(\langle 1, f \rangle) &= \langle 1, F_{\rightarrow}(f) \rangle, \\ (F + G)(\langle 2, g \rangle) &= \langle 2, G_{\rightarrow}(g) \rangle. \end{aligned} \quad (6)$$

Solutions to selected exercises

Solution of Exercise 40. We start with associativity. Consider morphism $f : l \rightarrow m$ given by $\langle \text{st}_f, \mathbf{A}_f, \mathbf{B}_f, \mathbf{C}_f, \mathbf{D}_f \rangle$, and $g : m \rightarrow n$ given by $\langle \text{st}_g, \mathbf{A}_g, \mathbf{B}_g, \mathbf{C}_g, \mathbf{D}_g \rangle$, and $h : n \rightarrow o$ given by $\langle \text{st}_h, \mathbf{A}_h, \mathbf{B}_h, \mathbf{C}_h, \mathbf{D}_h \rangle$. The morphism $f \circ g$ is described by the LTI

$$\langle \text{st}_{f,g}, \mathbf{A}_{f,g}, \mathbf{B}_{f,g}, \mathbf{C}_{f,g}, \mathbf{D}_{f,g} \rangle, \quad (7)$$

where

$$\text{st}_{f,g} = \begin{bmatrix} \text{st}_f \\ \text{st}_g \\ \text{st}_h \end{bmatrix}, \quad \mathbf{A}_{f,g} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} \\ \mathbf{B}_g \mathbf{C}_f & \mathbf{A}_g \end{bmatrix}, \quad \mathbf{B}_{1,2} = \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_g \mathbf{D}_f \end{bmatrix}, \quad \mathbf{C}_{f,g} = [\mathbf{D}_g \mathbf{C}_f \quad \mathbf{C}_g], \quad \mathbf{D}_{f,g} = \mathbf{D}_g \mathbf{D}_f. \quad (8)$$

The morphism $(f \circ g) \circ h$ is described by

$$\langle \text{st}_{(f,g),h}, \mathbf{A}_{(f,g),h}, \mathbf{B}_{(f,g),h}, \mathbf{C}_{(f,g),h}, \mathbf{D}_{(f,g),h} \rangle, \quad (9)$$

where:

$$\begin{aligned} \text{st}_{(f,g),h} &= \begin{bmatrix} \text{st}_f \\ \text{st}_g \\ \text{st}_h \end{bmatrix}, \quad \mathbf{A}_{(f,g),h} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_g \mathbf{C}_f & \mathbf{A}_g & \mathbf{0} \\ \mathbf{B}_h \mathbf{D}_g \mathbf{C}_f & \mathbf{B}_h \mathbf{C}_g & \mathbf{A}_h \end{bmatrix}, \quad \mathbf{B}_{(f,g),h} = \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_g \mathbf{D}_f \\ \mathbf{B}_h \mathbf{D}_g \mathbf{D}_f \end{bmatrix}, \\ \mathbf{C}_{(f,g),h} &= [\mathbf{D}_h \mathbf{D}_g \mathbf{C}_f \quad \mathbf{D}_h \mathbf{C}_g \quad \mathbf{C}_h], \quad \mathbf{D}_{(f,g),h} = \mathbf{D}_h \mathbf{D}_g \mathbf{D}_f. \end{aligned} \quad (10)$$

On the other hand, the morphism $g \circ h$ is described by

$$\langle \text{st}_{g,h}, \mathbf{A}_{g,h}, \mathbf{B}_{g,h}, \mathbf{C}_{g,h}, \mathbf{D}_{g,h} \rangle, \quad (11)$$

where:

$$\text{st}_{g,h} = \begin{bmatrix} \text{st}_g \\ \text{st}_h \end{bmatrix}, \quad \mathbf{A}_{g,h} = \begin{bmatrix} \mathbf{A}_g & \mathbf{0} \\ \mathbf{B}_h \mathbf{C}_g & \mathbf{A}_h \end{bmatrix}, \quad \mathbf{B}_{g,h} = \begin{bmatrix} \mathbf{B}_g \\ \mathbf{B}_h \mathbf{D}_g \end{bmatrix}, \quad \mathbf{C}_{g,h} = [\mathbf{D}_h \mathbf{C}_g \quad \mathbf{C}_h], \quad \mathbf{D}_{g,h} = \mathbf{D}_h \mathbf{D}_g. \quad (12)$$

Furthermore, the morphism $f \circ (g \circ h)$ is described by

$$\langle \text{st}_{f,(g,h)}, \mathbf{A}_{f,(g,h)}, \mathbf{B}_{f,(g,h)}, \mathbf{C}_{f,(g,h)}, \mathbf{D}_{f,(g,h)} \rangle, \quad (13)$$

where:

$$\begin{aligned} \text{st}_{(f,g),h} &= \begin{bmatrix} \text{st}_f \\ \text{st}_g \\ \text{st}_h \end{bmatrix}, \quad \mathbf{A}_{(f,g),h} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_g \mathbf{C}_f & \mathbf{A}_g & \mathbf{0} \\ \mathbf{B}_h \mathbf{D}_g \mathbf{C}_f & \mathbf{B}_h \mathbf{C}_g & \mathbf{A}_h \end{bmatrix}, \quad \mathbf{B}_{(f,g),h} = \begin{bmatrix} \mathbf{B}_f \\ \mathbf{B}_g \mathbf{D}_f \\ \mathbf{B}_h \mathbf{D}_g \mathbf{D}_f \end{bmatrix}, \\ \mathbf{C}_{(f,g),h} &= [\mathbf{D}_h \mathbf{D}_g \mathbf{C}_f \quad \mathbf{D}_h \mathbf{C}_g \quad \mathbf{C}_h], \quad \mathbf{D}_{(f,g),h} = \mathbf{D}_h \mathbf{D}_g \mathbf{D}_f. \end{aligned} \quad (14)$$

Clearly, the matrices in (10) and (14) coincide, showing associativity.

We now show unitality. Consider a morphism $f : l \rightarrow m$, described by $\langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$. The morphism $\text{id}_l \circ f$ is a morphism $l \rightarrow m$ still given by $\langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$. Similarly, the morphism $f \circ \text{id}_m$ is a morphism $l \rightarrow m$, given by $\langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$. Therefore, **LTI** is a category.

Solution of Exercise 41. To show that this is a valid functor, we need to show that it preserves identities and composition:

▷ Given any $X \in \text{Ob}_{\mathbf{C}}$, we have:

$$\begin{aligned} \text{id}_{\mathbf{C}}(\text{id}_X) &= \text{id}_X \\ &= \text{id}_{\text{id}_{\mathbf{C}}(X)} \end{aligned} \quad (15)$$

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

Furthermore, given composable morphisms f, g in \mathbf{C} , we have:

$$\begin{aligned}\text{id}_{\mathbf{C}}(f \circ g) &= f \circ g \\ &= \text{id}_{\mathbf{C}}(f) \circ \text{id}_{\mathbf{C}}(g).\end{aligned}\quad (16)$$

Solution of Exercise 42.

Solution of Exercise 43.

We check the specializations one by one. In all specializations, we consider the same objects as in **Draw**.

- ▷ **Scalings.** Let $s, t \in \mathbb{R}$. Scalings can be represented as functions of the form

$$\begin{aligned}\text{sc}_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2, \\ \langle x, y \rangle &\mapsto \langle sx, ty \rangle.\end{aligned}\quad (17)$$

By just considering morphisms which are scalings, we are considering a subset of all morphisms. Furthermore, the composition of two scalings is again a scaling. Indeed, consider scalings $\text{sc}_{s,t}, \text{sc}_{u,v}$. We have

$$\begin{aligned}(\text{sc}_{s,t} \circ \text{sc}_{u,v})(x, y) &= \text{sc}_{u,v}(sx, ty) \\ &= \langle usx, vty \rangle \\ &= \text{sc}_{us,vt}.\end{aligned}\quad (18)$$

Finally, the identity morphism in **Draw** corresponds to a scaling of the form $\text{sc}_{1,1}$.

- ▷ **Translations.** Let $s, t \in \mathbb{R}$. Translations are functions of the form

$$\begin{aligned}\text{tra}_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2, \\ \langle x, y \rangle &\mapsto \langle x + s, y + t \rangle.\end{aligned}\quad (19)$$

By just considering morphisms which are translations, we are considering a subset of all morphisms. Furthermore, the composition of two translations is again a translation. Indeed, consider scalings $\text{tra}_{s,t}, \text{tra}_{u,v}$. We have

$$\begin{aligned}(\text{tra}_{s,t} \circ \text{tra}_{u,v})(x, y) &= \text{tra}_{u,v}(x + s, y + t) \\ &= \langle x + s + u, y + t + v \rangle \\ &= \text{tra}_{s+u, t+v}.\end{aligned}\quad (20)$$

Finally, the identity morphism in **Draw** corresponds to a translation of the form $\text{tra}_{0,0}$.

- ▷ **Rotations.** Let $\theta \in [0, 2\pi)$. Rotations are functions of the form

$$\begin{aligned}\text{rot}_\theta : \mathbb{R}^2 &\rightarrow \mathbb{R}^2, \\ \langle x, y \rangle &\mapsto \langle x \cos(\theta) + y \sin(\theta), y \cos(\theta) - x \sin(\theta) \rangle.\end{aligned}\quad (21)$$

By just considering morphisms which are rotations, we are considering a subset of all morphisms. Furthermore, the composition of two rotations is again a rotation. Indeed, consider rotations $\text{rot}_\theta, \text{rot}_\phi$. We have

$$\begin{aligned}(\text{rot}_\theta \circ \text{rot}_\phi)(x, y) &= \text{rot}(\langle x \cos(\theta) + y \sin(\theta), y \cos(\theta) - x \sin(\theta) \rangle) \\ &= \langle x \cos(\theta + \phi) + y \sin(\theta + \phi), y \cos(\theta + \phi) - x \sin(\theta + \phi) \rangle \\ &= \text{rot}_{\theta+\phi}.\end{aligned}\quad (22)$$

Finally, the identity morphism in **Draw** corresponds to a rotation of the form

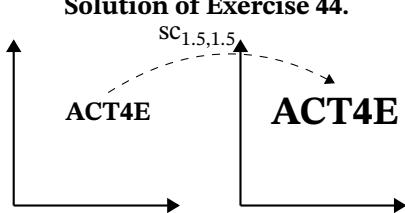


Figure 1.: Example of scaling.

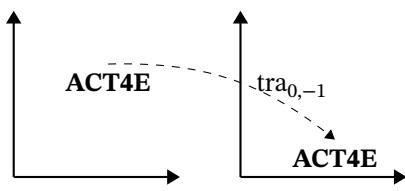


Figure 2.: Example of translation.

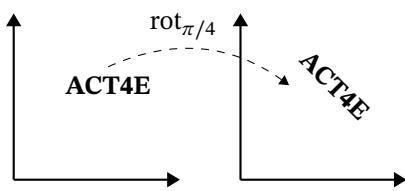


Figure 3.: Example of rotation.

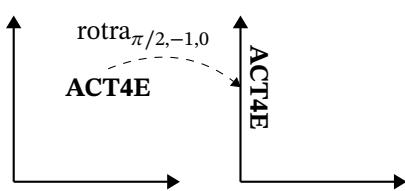


Figure 4.: Example of rototranslation.

rot_0 .

- ▷ **Rototranslations.** Let $s, t \in \mathbb{R}$ and $\theta \in [0, 2\pi)$. Rototranslations are functions arising from the combination of rotations and translations, and are therefore of the form:

$$\begin{aligned} \text{rotra}_{\theta,s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2, \\ \langle x, y \rangle &\mapsto \langle x \cos(\theta) + y \sin(\theta) + s, y \cos(\theta) - x \sin(\theta) + t \rangle. \end{aligned} \quad (23)$$

By just considering morphisms which are rotations, we are considering a subset of all morphisms. Furthermore, the composition of two rototranslations is again a rototranslation. Consider rototranslations $\text{rotra}_{\theta,s,t}$, $\text{rotra}_{\phi,u,v}$. We have:

$$\begin{aligned} (\text{rotra}_{\theta,s,t} \circ \text{rotra}_{\phi,u,v})(x, y) &= \text{rotra}_{\phi,u,v}(x \cos(\theta) + y \sin(\theta) + s, y \cos(\theta) - x \sin(\theta) + t) \\ &= \langle (x \cos(\theta) + y \sin(\theta) + s) \cos(\phi) + (y \cos(\theta) - x \sin(\theta) + t) \sin(\phi) + u, \\ &\quad (y \cos(\theta) - x \sin(\theta) + t) \cos(\phi) - (x \cos(\theta) + y \sin(\theta) + s) \sin(\phi) + v \rangle \\ &= \langle x \cos(\theta + \phi) + y \sin(\theta + \phi) + s \cos(\phi) + t \sin(\phi) + u, \\ &\quad y \cos(\theta + \phi) - x \sin(\theta + \phi) + t \cos(\phi) - s \sin(\phi) + v \rangle \\ &= \text{rotra}_{\theta+\phi,s \cos(\phi)+t \sin(\phi)+u,t \cos(\phi)-s \sin(\phi)+v}(x, y). \end{aligned} \quad (24)$$

Finally, the identity morphism in **Draw** corresponds to a rotation of the form $\text{rotra}_{0,0,0}$.

- ▷ **Affine transformations.** Let $\mathbf{A} \in \mathbb{R}^{2 \times 2}$ and $\mathbf{b} \in \mathbb{R}^{2 \times 1}$. Affine transformations are functions that could arise from the combination of rotations and translations, and scalings, and are therefore of the form:

$$\begin{aligned} \text{aff}_{\mathbf{A}, \mathbf{b}} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2, \\ \langle x, y \rangle &\mapsto \langle a_{11}x + a_{12}y + b_{11}, a_{21}x + a_{22}y + b_{21} \rangle, \end{aligned} \quad (25)$$

where $*_{ij}$ represents the element at the i -th row and j -th column of $*$.

Some special cases are:

- With $\mathbf{A} = \mathbb{I}$ we obtain translations $\text{tra}_{b_{11}, b_{21}}$;
- With $\mathbf{b} = [0 \ 0]^\top$ and $A = \begin{bmatrix} s & 0 \\ 0 & t \end{bmatrix}$ we obtain scalings $\text{sc}_{s,t}$;
- With $\mathbf{b} = [0 \ 0]^\top$ and $A = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$ we obtain rotations rot_θ .

By just considering morphisms which are affine transformations, we are considering a subset of all morphisms. Furthermore, the composition of two affine transformations is again an affine transformation. Clearly, the composition of affine transformations $\text{aff}_{\mathbf{A}, \mathbf{b}}$, $\text{aff}_{\mathbf{C}, \mathbf{d}}$ is $\text{aff}_{\mathbf{CA}, \mathbf{Cb} + \mathbf{d}}$. Finally, the identity morphism in **Draw** corresponds to an affine transformation of the form $\text{aff}_{\mathbb{I}, \mathbf{0}^{2 \times 1}}$.

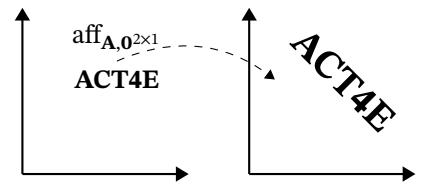


Figure 5.: Example of affine transformation with $A = 1.5 \begin{bmatrix} \cos(\pi/4) & \sin(\pi/4) \\ -\sin(\pi/4) & \cos(\pi/4) \end{bmatrix}$.

Solution of Exercise 45. In the following, we want to show that functors compose. Given categories **A**, **B**, **C** and functors $F : \mathbf{A} \rightarrow \mathbf{B}$, $G : \mathbf{B} \rightarrow \mathbf{C}$, we want to show that $F \circ G$ is a functor. To do this, we show that $F \circ G$ preserves identities and compositions.

- ▷ Given an object X in **A**, we have:

$$\begin{aligned} (F_\rightarrow \circ G_\rightarrow)(\text{id}_X) &= G_\rightarrow(F_\rightarrow(\text{id}_X)) \\ &= G_\rightarrow(\text{id}_{F_\rightarrow(X)}) \end{aligned} \quad (26)$$

where we used that $\textcolor{violet}{F}$ and $\textcolor{violet}{G}$ are functors (they preserve identities).

▷ Furthermore, given composable morphisms f, g in \mathbf{A} , we have:

$$\begin{aligned} (\textcolor{violet}{F}_\rightarrow \circ \textcolor{violet}{G}_\rightarrow)(\textcolor{blue}{f} \circ g) &= \textcolor{violet}{G}_\rightarrow(\textcolor{violet}{F}_\rightarrow(\textcolor{blue}{f}) \circ \textcolor{violet}{F}_\rightarrow(g)) \\ &= \textcolor{violet}{G}_\rightarrow(\textcolor{violet}{F}_\rightarrow(\textcolor{blue}{f})) \circ \textcolor{violet}{G}_\rightarrow(\textcolor{violet}{F}_\rightarrow(g)) \\ &= (\textcolor{violet}{F}_\rightarrow \circ \textcolor{violet}{G}_\rightarrow)(\textcolor{blue}{f}) \circ (\textcolor{violet}{F}_\rightarrow \circ \textcolor{violet}{G}_\rightarrow)(g), \end{aligned} \tag{27}$$

where again we used that $\textcolor{violet}{F}, \textcolor{violet}{G}$ are functors (they preserve composition).

PART G.NATURALITY



23. Naturality 309

24. Adjunctions 325

The *Sechseläuten* is a traditional spring holiday in the Zurich, Switzerland, usually happening on the 3rd monday of April. The old city guilds meet in the city center for a parade, climax of which is the burning of the “Böögg”, a snowman prepared with explosives, considered a weather oracle for the summer.

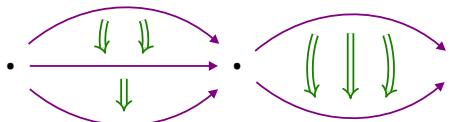


23. Naturality

We have seen that functors are “morphisms between categories”. It turns out that there is an important third layer to the world of categories: there are also “morphisms between functors”, and these are known as *natural transformations*.

To represent the three layers of structure involved in the world of categories, it is common to draw diagrams like this: Points repre-

23.1 Natural transformations	310
23.2 Morphisms in a category of Functors	315
23.3 Data migration	319
23.4 More examples	321



sent categories, single arrows represent functors, and double arrows represent natural transformations.

When one thinks about Switzerland, one of the symbols that comes to mind is usually cows. Recent statistics show that Switzerland has around 1.6 million cows (roughly one cow per five residents). The canton of Bern leads the rankings, being the one owning the most cows. Interestingly, in the canton of Appenzell Innerrhoden, the ratio of cows and humans is close to 1:1.

23.1. Natural transformations

To formally define natural transformations, the general situation we will start from is when we have two functors $F : \mathbf{C} \rightarrow \mathbf{D}$ and $G : \mathbf{C} \rightarrow \mathbf{D}$, sharing the same source and target, respectively.

A natural transformation from F to G is then a kind of “map” that relates the two functors. How might one define such a thing?

First, let’s look at the situation only on the level of objects. Each object X of \mathbf{C} is mapped by F and G to an object $F(X)$ and $G(X)$ of \mathbf{D} , respectively. One straightforward way to relate $F(X)$ to $G(X)$ is to choose a morphism $F(X) \rightarrow G(X)$ in \mathbf{D} . We call this morphism α_X , using the subscript X since α_X relates the respective images of the object X under F and G . If we choose such a morphism for each object in \mathbf{C} , then we have collection $\{\alpha_X\}_{X \in \text{Ob}_C}$ of morphisms in \mathbf{D} , indexed by the objects of \mathbf{C} .

Next, consider a morphism $f : X \rightarrow Y$ in the category \mathbf{C} . Under the functor F it will be mapped to some morphism $F(f) : F(X) \rightarrow F(Y)$ in \mathbf{D} , and under the functor G it will be mapped to some other morphism $G(f) : G(X) \rightarrow G(Y)$, also in \mathbf{D} .

We can think of $F(f) : F(X) \rightarrow F(Y)$ and $G(f) : G(X) \rightarrow G(Y)$ as each being very small diagrams (directed graphs) in \mathbf{D} . If we have already chosen morphisms $\alpha_X : F(X) \rightarrow G(X)$ and $\alpha_Y : F(Y) \rightarrow G(Y)$ in \mathbf{D} , then these will connect the two diagrams, as depicted in the figure to the side.

In \mathbf{D} , this gives rise to the square diagram shown to the side. We’ll require, as a condition on the morphisms α_X and α_Y , that they make the diagram commutative:

$$F(f) ; \alpha_Y = \alpha_X ; G(f). \quad (1)$$

Now consider not only a single morphism $f : X \rightarrow Y$ in \mathbf{C} being mapped by F and G , respectively, but *all* of the category \mathbf{C} . Under F , the category \mathbf{C} is mapped to a – possibly very complicated – diagram in \mathbf{D} (a directed graph of objects and morphism comprising the image of F), and similarly, under G , the category \mathbf{C} is mapped to another diagram in \mathbf{D} (the image of G).

To relate the image of F to the image of G we can proceed in the same way as above: for each object X in \mathbf{C} , we choose a morphism $\alpha_X : F(X) \rightarrow G(X)$ in \mathbf{D} . In other words, we have a collection of morphisms $(\alpha_X), X \in \text{Ob}_C$ indexed by the objects of \mathbf{C} . These give rise to lots of squares of the kind in Fig. 1, which we will require to be commutative. It is because of this commutativity condition, which is a condition on the collection $(\alpha_X), X \in \text{Ob}_C$, that some mathematicians would say that the collection $(\alpha_X), X \in \text{Ob}_C$ is a “coherent” or “natural” way to relate the image of F to the image of G . (This does not mean, however, that there is at most one natural transformation between any two given functors—on the contrary, there might be many!)

In Fig. 2 we have illustrated a situation involving three objects and two morphisms in \mathbf{C} , giving rise to two squares. We have “glued” the two squares together since they share an edge (this a more compact way of drawing them). Note that because each of the two component squares in the diagram commute, so does the entire diagram.

Definition 23.1 (Natural transformation)

Let \mathbf{C} and \mathbf{D} be categories, and let $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors. A *natural transformation* $\alpha : F \Rightarrow G$ is specified by:

Constituents

- For each object $X \in \text{Ob}_C$, a morphism $\alpha_X : F(X) \rightarrow G(X)$ in \mathbf{D} , called

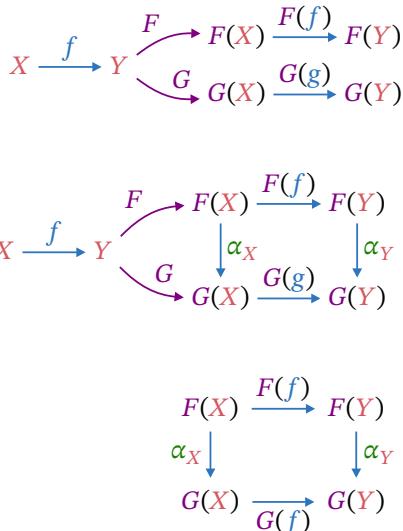


Figure 1.

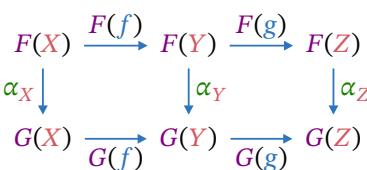


Figure 2.

the X -component of α .

Conditions

- For every morphism $f : X \rightarrow Y$ in \mathbf{C} , the components of α must satisfy the *naturality condition*

$$F(f) \circ \alpha_Y = \alpha_X \circ G(f). \quad (2)$$

In other words, the following diagram must commute:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(f)} & F(Y) \\ \alpha_X \downarrow & & \downarrow \alpha_Y \\ G(X) & \xrightarrow{G(f)} & G(Y) \end{array} \quad (3)$$

To reiterate: a natural transformation α is a *collection* $(\alpha_X)_{X \in \text{Ob}_{\mathbf{C}}}$ of morphisms (called the *components* of the natural transformation) which satisfy the naturality conditions. The name “components” is analogous to how a vector $v = (v_1, \dots, v_n)$ has *components* or a sequence $a = (a_n)_{n \in \mathbb{N}}$ has *terms*.

The diagrams (3) are often called *naturality squares*, and a natural transformation $\alpha : F \Rightarrow G$ is often depicted concisely in this manner:

$$\begin{array}{ccc} & \overset{F}{\curvearrowright} & \\ C & \Downarrow \alpha & D \\ & \underset{G}{\curvearrowright} & \end{array} \quad (4)$$

Figure 3 shows a diagram that describes the property of functors and of natural transformations. The diagram in \mathbf{D} is a “commuting prism”: all faces of the prism commute.

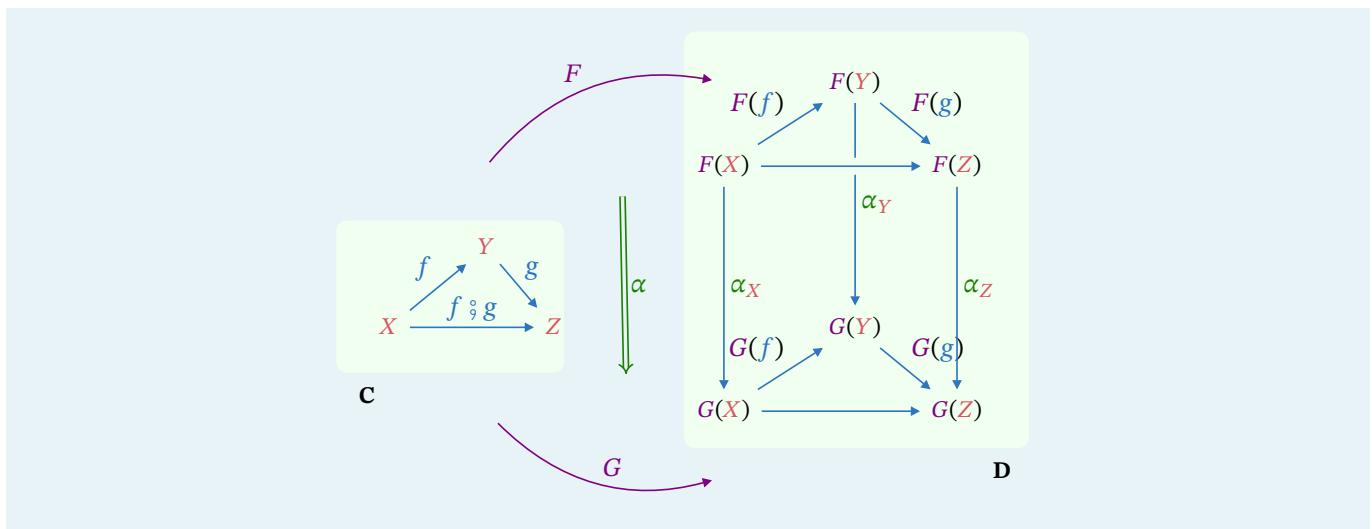


Figure 3.

Examples

Example 23.2. Consider the following two functors $\mathbf{F}, \mathbf{G} : \mathbf{Set} \times \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}$. We define \mathbf{F} on objects by

$$\mathbf{F}(\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle) = (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \quad (5)$$

and define \mathbf{G} on objects by

$$\mathbf{G}(\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle) = \mathbf{A} \times (\mathbf{B} \times \mathbf{C}). \quad (6)$$

For their actions on morphisms, consider a morphism

$$\langle f, g, h \rangle : \langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle \rightarrow \langle \mathbf{A}', \mathbf{B}', \mathbf{C}' \rangle \quad (7)$$

in $\mathbf{Set} \times \mathbf{Set} \times \mathbf{Set}$. Its image under \mathbf{F} is

$$\langle\langle f, g, h \rangle \rangle : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \rightarrow (\mathbf{A}' \times \mathbf{B}') \times \mathbf{C}' \quad (8)$$

and its image under \mathbf{G} is

$$\langle\langle f, g, h \rangle \rangle : \mathbf{A} \times (\mathbf{B} \times \mathbf{C}) \rightarrow \mathbf{A}' \times (\mathbf{B}' \times \mathbf{C}'). \quad (9)$$

One way to see that \mathbf{F} (and similarly \mathbf{G}) is indeed a functor is to note that it is equal to the following composition of functors

$$\begin{aligned} & \mathbf{Set} \times \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}, \\ & \langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle \mapsto \langle \mathbf{A} \times \mathbf{B}, \mathbf{C} \rangle \mapsto (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \end{aligned} \quad (10)$$

and recall from Graded Exercise F.9 that “ \times ” is a functor.

Now we define a natural transformation $\alpha : \mathbf{F} \Rightarrow \mathbf{G}$ by specifying its components to be the functions

$$\begin{aligned} \alpha_{\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} & \rightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C}), \\ \langle\langle x, y \rangle, z \rangle & \mapsto \langle x, \langle y, z \rangle \rangle, \end{aligned} \quad (11)$$

indexed by triples of sets $\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle$.

For the family of morphisms $\alpha_{\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle}$ to be a natural transformation, we need to check that the diagrams

$$\begin{array}{ccc} (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} & \xrightarrow{\langle\langle f, g, h \rangle \rangle} & (\mathbf{A}' \times \mathbf{B}') \times \mathbf{C}' \\ \alpha_{\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle} \downarrow & & \downarrow \alpha_{\langle \mathbf{A}', \mathbf{B}', \mathbf{C}' \rangle} \\ \mathbf{A} \times (\mathbf{B} \times \mathbf{C}) & \xrightarrow{\langle\langle f, g, h \rangle \rangle} & \mathbf{A}' \times (\mathbf{B}' \times \mathbf{C}') \end{array} \quad (12)$$

in \mathbf{Set} commute for all morphisms $\langle f, g, h \rangle$ in $\mathbf{Set} \times \mathbf{Set} \times \mathbf{Set}$. It is easily checked that this is true.

This natural transformation is an example of something called an *associator*, which we will discuss later when we define monoidal categories. The idea here is that the cartesian product of sets is not quite an associative operation, but almost: instead of an “equality” symbol in the usual equation for the associative law, we have the components of this associator natural transformation.

Example 23.3. Let $\mathbf{F} : \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}$ be the functor which on objects maps any pair of sets $\langle \mathbf{A}, \mathbf{B} \rangle$ to their cartesian product $\mathbf{A} \times \mathbf{B}$. On functions, it maps any

pair of functions $\langle f, g \rangle$ to their cartesian product $f \times g$.

Consider another functor $G : \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}$ which on objects maps any pair of sets $\langle A, B \rangle$ to their cartesian product $B \times A$ (and similarly for morphisms). In other words, G is very similar to F , however it is different in that, when forming the cartesian product, the order of the factors is swapped.

There is a natural transformation $\alpha : F \Rightarrow G$ which expresses explicitly the relationship between F and G . Its components are these functions:

$$\begin{aligned}\alpha_{\langle A, B \rangle} : A \times B &\rightarrow B \times A, \\ \langle x, y \rangle &\mapsto \langle y, x \rangle.\end{aligned}\tag{13}$$

Graded exercise G.1 (NaturalBraiding)

Consider the functor $F : \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}$ defined on objects and morphisms by

$$\langle A, B \rangle \mapsto A \times B \quad \langle f, g \rangle \mapsto f \times g,\tag{14}$$

and consider as well the similar (but different!) functor $G : \mathbf{Set} \times \mathbf{Set} \rightarrow \mathbf{Set}$ defined on objects and morphisms by

$$\langle A, B \rangle \mapsto B \times A \quad \langle f, g \rangle \mapsto g \times f.\tag{15}$$

Now consider, for each ordered pair of sets $\langle A, B \rangle$, the function

$$\begin{aligned}\text{br}_{\langle A, B \rangle} : A \times B &\rightarrow B \times A, \\ \langle x, y \rangle &\mapsto \langle y, x \rangle.\end{aligned}\tag{16}$$

Your task is to prove that the family of functions $\{\text{br}_{\langle A, B \rangle}\}_{\langle A, B \rangle}$ defines a natural transformation $\text{br} : F \Rightarrow G$.

Example 23.4. Consider the powerset functor \mathbf{Pow} from Example 19.3. As a reminder, the powerset functor \mathbf{Pow} maps a set A to its powerset $\mathbf{Pow}(A)$, and a function $f : A \rightarrow B$ to the function $\mathbf{Pow}(f) : \mathbf{Pow}(A) \rightarrow \mathbf{Pow}(B)$ which sends each subset of A to its image under f , which is a subset of B .

There is a natural transformation $\alpha : \text{id}_{\mathbf{Set}} \Rightarrow \mathbf{Pow}$ whose components are the functions

$$\begin{aligned}\alpha_A : A &\rightarrow \mathbf{Pow}(A) \\ a &\mapsto \{a\}.\end{aligned}\tag{17}$$

In other words, the natural transformation embeds each element of A into the power set $\mathbf{Pow}(A)$. To check that this is a natural transformation, consider an arbitrary function between sets $f : A \rightarrow B$. On the one hand,

$$(\alpha_A ; \mathbf{Pow}(f))(a) = \mathbf{Pow}(f)(\{a\}) = \{f(a)\},\tag{18}$$

while on the other hand

$$(f ; \alpha_B)(a) = \alpha_B(f(a)) = \{f(a)\}.\tag{19}$$

Thus, the condition for α to be a natural transformation is satisfied.

Graded exercise G.2 (ListUnitNatural)

Recall the list functor $\mathbf{List} : \mathbf{Set} \rightarrow \mathbf{Set}$ which, on objects, assigns to each set A the set $\mathbf{List}(A)$ of finite lists in elements of A , and to each function

$f : \mathbf{A} \rightarrow \mathbf{B}$ it assigns the induced function

$$\begin{aligned}\text{List}(f) : \text{List } \mathbf{A} &\rightarrow \text{List } \mathbf{B}, \\ [\dots, a_i, \dots] &\mapsto [\dots, f(a_i), \dots].\end{aligned}\tag{20}$$

(You do not need to prove that this is a functor; we take that fact as given in this exercise.)

Consider now the family of functions $\{\alpha_{\mathbf{A}}\}_{\mathbf{A}}$ defined by

$$\begin{aligned}\alpha_{\mathbf{A}} : \mathbf{A} &\rightarrow \text{List } \mathbf{A}, \\ x &\mapsto [x]_{\mathbf{A}}.\end{aligned}\tag{21}$$

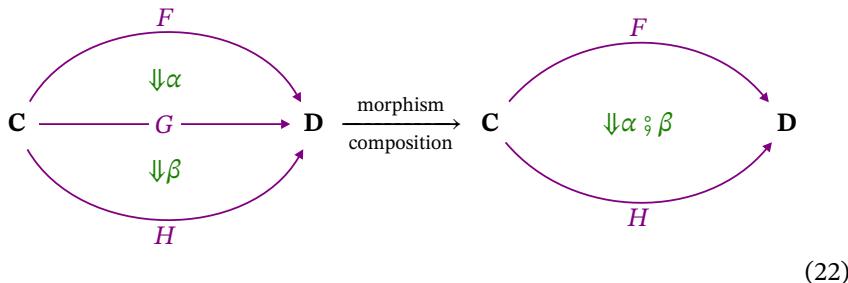
In other words, the function $\alpha_{\mathbf{A}}$ maps each element x of \mathbf{A} to a list of length 1 whose single entry is the element x . Does this family of functions define a natural transformation? Prove your answer.

23.2. Morphisms in a category of Functors

We have seen that natural transformations between two functors $F, G : \mathbf{C} \rightarrow \mathbf{D}$ map objects in \mathbf{C} to morphisms in \mathbf{D} , and map morphisms in \mathbf{C} to commutative diagrams. This is quite similar to the effect of functors on category objects and morphisms. What if there were a category where objects are functors, and morphisms are natural transformations?

Vertical Composition

As for any category, we would need to define the morphism composition law and the identity morphisms. The former would look like this:



Due to its diagrammatic form, we call this type of composition vertical composition.

Definition 23.5 (Vertical Composition of natural transformations)

Let \mathbf{C}, \mathbf{D} be categories and let $F, G, H : \mathbf{C} \rightarrow \mathbf{D}$ be functors from \mathbf{C} to \mathbf{D} . Suppose we are given natural transformations

$$\alpha : F \Rightarrow G, \quad (23)$$

$$\beta : G \Rightarrow H. \quad (24)$$

Their (vertical) composition $\alpha ; \beta$ is the natural transformation

$$\alpha ; \beta : F \Rightarrow H \quad (25)$$

defined in components by

$$(\alpha ; \beta)_X := \alpha_X ; \beta_X \quad \forall X \in \text{Ob}_{\mathbf{C}}. \quad (26)$$

Definition 23.6 (Identity natural transformation)

Let \mathbf{C}, \mathbf{D} be categories and let $F : \mathbf{C} \rightarrow \mathbf{D}$ be a functor. The identity natural transformation at F is the natural transformation $\text{id}_F : F \Rightarrow F$ defined in components by

$$(\text{id}_F)_X := \text{id}_{F(X)} \quad \forall X \in \text{Ob}_{\mathbf{C}}. \quad (27)$$

Definition 23.7

Let \mathbf{C}, \mathbf{D} be categories. The category $[\mathbf{C}, \mathbf{D}]$ of functors from \mathbf{C} to \mathbf{D} is given by

1. *Objects*: functors $\mathbf{C} \rightarrow \mathbf{D}$.
2. *Morphisms*: natural transformations between functors $\mathbf{C} \rightarrow \mathbf{D}$.
3. *Composition*: (vertical) composition of natural transformations.

4. Identities: identity natural transformations.

Natural isomorphisms**Definition 23.8** (Natural isomorphism)

A natural transformation $\alpha : F \Rightarrow G$ is called a *natural isomorphism* if each component morphism α_X in \mathbf{D} is an isomorphism.

Lemma 23.9. Let \mathbf{C}, \mathbf{D} be categories and let $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors. A natural transformation $\alpha : F \Rightarrow G$ is an isomorphism in the category of functors $[\mathbf{C}, \mathbf{D}]$ if and only if α is a natural isomorphism.

Horizontal Composition

Suppose we have three categories: \mathbf{C}, \mathbf{D} and \mathbf{E} with functors $F_1, F_2 : \mathbf{C} \rightarrow \mathbf{D}$ and $G_1, G_2 : \mathbf{D} \rightarrow \mathbf{E}$ and finally two natural transformations $\alpha : F_1 \Rightarrow F_2$ and $\beta : G_1 \Rightarrow G_2$. We would then have the following situation.

$$\begin{array}{ccc} & F_1 & F_2 \\ A & \Downarrow \alpha & \Downarrow \beta \\ & G_1 & G_2 \end{array} \quad (28)$$

This looks suspiciously composable.

Definition 23.10 (Horizontal Composition of natural transformations)

Let $\mathbf{C}, \mathbf{D}, \mathbf{E}$ be categories and let $F_1, F_2 : \mathbf{C} \rightarrow \mathbf{D}$ and $G_1, G_2 : \mathbf{D} \rightarrow \mathbf{E}$ be functors. Suppose we are given the natural transformations

$$\alpha : F_1 \Rightarrow G_1, \quad (29)$$

$$\beta : F_2 \Rightarrow G_2. \quad (30)$$

Then their horizontal composition is given by the natural transformation

$$\alpha * \beta : (F_1 ; G_1) \Rightarrow (G_1 ; G_2) \quad (31)$$

defined in components by

$$(\alpha * \beta)_X := \alpha_X * \beta_X \quad \forall X \in \text{Ob}_{\mathbf{C}} \quad (32)$$

as the composition

$$(\alpha * \beta)_X : (F_2(F_1(X))) \Rightarrow (G_2(F_1(X))) \Rightarrow (G_2(G_1(X))) \quad (33)$$

Interchange Law

The following statement is quite powerful. Although perhaps obvious-looking, it allows us to presume associativity of natural transformations.

Proposition 23.11 (Interchange). Let $\mathbf{C}, \mathbf{D}, \mathbf{E}$ be categories, $F_1, G_1, H_1 : \mathbf{C} \rightarrow \mathbf{D}$, $F_2, G_2, H_2 : \mathbf{D} \rightarrow \mathbf{E}$ be functors and $\alpha_1 : F_1 \Rightarrow G_1$, $\alpha_2 : F_2 \Rightarrow G_2$, $\beta_1 : G_1 \Rightarrow H_1$, $\beta_2 : G_2 \Rightarrow H_2$ be natural transformations. Then,

$$(\alpha_1 ; \beta_1) * (\alpha_2 ; \beta_2) = (\alpha_1 * \alpha_2) ; (\beta_1 * \beta_2) \quad (34)$$

Proof. Consider the case where we have categories **C**, **D**, **E**, with functors $F_1, G_1 : \mathbf{C} \rightarrow \mathbf{D}$ and $F_2, G_2 : \mathbf{D} \rightarrow \mathbf{E}$ relating them. We can thus have the following four diagrams:

$$\begin{array}{ccc} & F_1 & \\ A & \Downarrow \alpha_1 & B & B & \Downarrow \alpha_2 & C \\ & G_1 & & & G_2 & \\ \end{array} \quad (35)$$

$$\begin{array}{ccc} & G_1 & \\ A & \Downarrow \beta_1 & B & B & \Downarrow \beta_2 & C \\ & H_1 & & & H_2 & \\ \end{array} \quad (36)$$

We are now faced with the choice of either vertically composing first, or horizontally composing. By vertically composing, we get

$$\begin{array}{ccc} & F_1 & \\ A & \Downarrow \alpha_1 ; \beta_1 & B & B & \Downarrow \alpha_2 ; \beta_2 & C \\ & H_1 & & & H_2 & \\ \end{array} \quad (37)$$

Subsequently applying horizontal composition yields

$$\begin{array}{ccc} & F_1 ; F_2 & \\ A & \Downarrow (\alpha_1 ; \beta_1) * (\alpha_2 ; \beta_2) & C \\ & H_1 ; H_2 & \\ \end{array} \quad (38)$$

Otherwise, we apply horizontal composition first:

$$\begin{array}{ccc} & F_1 ; F_2 & \\ C & \Downarrow \alpha_1 ; \alpha_2 & D \\ & G_1 ; G_2 & \\ & \Downarrow \beta_1 ; \beta_2 & \\ & H_1 ; H_2 & \\ \end{array} \quad (39)$$

thus proving that vertical and horizontal composition are interchangeable. \square

Remark 23.12. The proof is also interesting to do by observing the commuting squares, and putting them together.

Whiskering

Two special cases of horizontal composition are known as “left-whiskering” and “right-whiskering”.

Definition 23.13 (Right Whiskering)

Let \mathbf{C} , \mathbf{D} and \mathbf{E} be categories, $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors from \mathbf{C} to \mathbf{D} , $H : \mathbf{D} \rightarrow \mathbf{E}$ be a functor from \mathbf{D} to \mathbf{E} and $\alpha : F \Rightarrow G$ be a natural transformation from F to G .

The right whiskering of H and α is given by the natural transformation

$$H\alpha : (F ; H) \Rightarrow (G ; H) \quad (40)$$

Definition 23.14 (Left Whiskering)

Let \mathbf{C} , \mathbf{D} and \mathbf{E} be categories, $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors from \mathbf{C} to \mathbf{D} , $H : \mathbf{B} \rightarrow \mathbf{C}$ be a functor from \mathbf{B} to \mathbf{C} and $\beta : F \Rightarrow G$ be a natural transformation from F to G .

The left whiskering of β and H is given by the natural transformation

$$\beta H : (H ; F) \Rightarrow (H ; G) \quad (41)$$

23.3. Data migration

Now that we have seen some definitions and toy examples, let's look at an (toy) example a real-world application. As mentioned in Categorical Databases (see 19.5 for the full context), we can model instances of databases as functors from the category of the architecture of the database to the category of sets.

An alumni database The architecture of the database in our example might be encoded by a category with four objects and three non-identity morphisms, as depicted in Fig. 4. The object S stands for *Student*, D stands for *Discipline*, N stands for *Name*, and Y stands for *Year*. Call this category C . A database instance $F : C \rightarrow \text{Set}$ entails specifying a set $F(S)$ of all student IDs, a set $F(D)$ of university disciplines such as mechanical engineering, civil engineering, applied mathematics, pure mathematics, etc. It also entails defining functions for each of the arrows in C . For example, $F(\text{studied}) : F(S) \rightarrow F(D)$ is the function that assigns to each student ID the name of the discipline that that student studied.

Updating the database For simplicity, we focus on two aspects of the data: student ID numbers and the disciplines of study. We assume that the university updates its alumni database once a year. This means, for example, adding the graduates of that year to the total list of graduates.

To model the situation, let $F : C \rightarrow \text{Set}$ be the database instance for the year 2021, and let $G : C \rightarrow \text{Set}$ be the database instance for the year 2022. For concreteness, suppose the student IDs are in some standardized format, for example a code of the kind 17-371-802, where each of the three parts of the code are calculated/assigned by some rule (e.g., the “17” here stands for 2017, the year the student registered with the university, etc.).

Since new students register to the university each year, the set $F(S)$ of all student IDs registered up to the end of 2020 is a subset of the set $G(S)$ of student IDs up to the end of 2022. This means there is an inclusion function $\alpha_S : F(S) \rightarrow G(S)$.

Now suppose that in 2022 the university decides to simplify the way it attributes disciplines to students in the database.

For instance, instead of the discipline names $F(D) = \{\text{mechanical engineering, civil engineering, applied physics, theoretical physics, pure math, applied math}\}$, the new discipline names are just

$$G(D) = \{\text{engineering, physics, math}\}. \quad (42)$$

In order to implement these changes, we use a function $\alpha_D : F(D) \rightarrow G(D)$ which maps the old discipline names to the corresponding new ones in an obvious way:

$$\alpha_D(\text{civil engineering}) = \text{engineering}, \alpha_D(\text{applied physics}) = \text{physics}, \text{etc.} \quad (43)$$

The functions α_S and α_D allow us to check whether the new database instance G relates coherently with the older database instance F . Concretely, we want that if a student ID in $G(S)$ is inherited from $F(S)$ – in other words, if it is in the image of α_S – then we want that its associated discipline in the database instance G is the same as if we first computed the student's discipline in the older database instance F , and then mapped it to G using the function α_D .

This can be formulated succinctly by saying that we want the diagram in figure Fig. 5 to commute.

What we have defined is a collection of morphisms: for each object X in C , we have a morphism $\alpha_X : F(X) \rightarrow G(X)$ that obeys the commutativity property

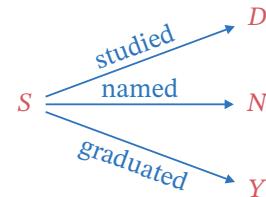


Figure 4.: The schema of an alumni database.

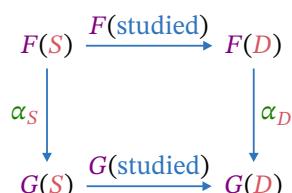


Figure 5.

corresponding to the diagram. Such transformations appear in many places and are formalized by the notion of natural transformations.

23.4. More examples

Example 23.15. Consider the category $\mathbf{Vect}_{\mathbb{R}}$ whose objects are real vector spaces and whose morphisms are linear maps. (For convenience, in the following we sometimes omit reference to the ground field.) Recall that the *dual* of a vector space V is the vector space describing all linear maps from V to \mathbb{R} :

$$V^* := \mathbf{Hom}_{\mathbf{Vect}_{\mathbb{R}}}(V; \mathbb{R}), \quad (44)$$

Also, recall that if $f : V \rightarrow W$ is a linear map, then its dual is the linear map $f^* : W^* \rightarrow V^*$ which maps any $\xi \in W^*$ to the element of V^* given by

$$f^*(\xi) : V \rightarrow \mathbb{R}, v \mapsto \xi(f(v)). \quad (45)$$

Applying the above duality construction twice to a vector space or a linear map gives their double dual. It turns out that this is a functorial operation. That is, there is a functor

$$\text{Double dual} : \mathbf{Vect}_{\mathbb{R}} \rightarrow \mathbf{Vect}_{\mathbb{R}} \quad (46)$$

that maps every vector space and every linear map to its double dual.

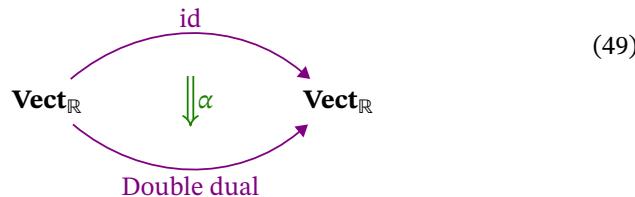
Furthermore, for any vector space V , there is a “canonical” or “natural” map

$$\alpha_V : V \Rightarrow V^{**} \quad (47)$$

defined by

$$\alpha_V(v)(l) = l(v), \quad v \in V, l \in V^*. \quad (48)$$

These form the components of a natural transformation from the identity functor on $\mathbf{Vect}_{\mathbb{R}}$ to the double dual functor.



Example 23.16. Fix a set S . There are functors $F, G : \mathbf{Set}^{\text{op}} \times \mathbf{Set} \rightarrow \mathbf{Set}$ whose respective actions on objects are

$$F_* : \langle A, B \rangle \mapsto \mathbf{Hom}_{\mathbf{Set}}(A \times S, B) \quad (50)$$

and

$$G_* : \langle A, B \rangle \mapsto \mathbf{Hom}_{\mathbf{Set}}(A, B^S). \quad (51)$$

These functors may be understood as built up using compositions of functors of the kind discussed in Graded Exercise F.5, Graded Exercise F.6 and Example 19.11.

Recall that we can “curry” any function $f : A \times S \rightarrow B$ to get a function $\hat{f} : A \rightarrow B^S$, where $\hat{f}(x)$ may be thought of as a partial evaluation of f .

There is a natural transformation $\alpha : F \Rightarrow G$ whose components are the functions

$$\alpha_{\langle A, B \rangle} : \mathbf{Hom}_{\mathbf{Set}}(A \times S, B) \rightarrow \mathbf{Hom}_{\mathbf{Set}}(A, B^S), f \mapsto \hat{f}, \quad (52)$$

where \hat{f} is the “curried” version of f .

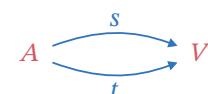


Figure 6.

Graded exercise G.3 ([NatTrafosGraphs](#))

This exercise builds on Graded Exercise F.7. There, we defined a category \mathbf{G} which has precisely two objects and four morphisms, see Fig. 6 (the two identity morphisms are not drawn). The task there was to understand how specifying a functor from this category CatSymbolG into the category of sets is “the same thing” as specifying a directed graph.

Now consider two functors $F_1, F_2 : \mathbf{G} \rightarrow \mathbf{Set}$. Spell out what it means to have a natural transformation $\alpha : F_1 \Rightarrow F_2$. What does this correspond to in the language of directed graphs?

Graded exercise G.4 (UpperSetsNatTrafos)

This exercise builds on Graded Exercise F.8. There we fixed a poset \mathbf{P} , viewed it as a category \mathbf{P} , and saw that functors $\mathbf{P} \rightarrow \mathbf{Bool}$ encode upper sets in \mathbf{P} . Suppose we have two functors $F_1, F_2 : \mathbf{P} \rightarrow \mathbf{Bool}$. What does a natural transformation $\alpha : F_1 \Rightarrow F_2$ correspond to in terms of the upper sets encoded by F_1 and F_2 , respectively?

Graded exercise G.5 (DoubleDualNatTrafo)

This exercise builds on Graded Exercise F.4. Consider the category $\mathbf{Vect}_{\mathbb{R}}$ whose objects are real vector spaces and whose morphisms are linear maps. For any vector space V , there is a “canonical” or “natural” map

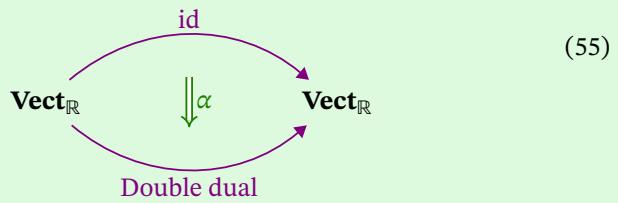
$$\alpha_V : V \Rightarrow V^{**} \quad (53)$$

defined by

$$\alpha_V(v)(l) = l(v), \quad v \in V, l \in V^*. \quad (54)$$

Your tasks in this exercise:

1. Check that the operation of “taking the double dual” (see (46) above) defines a functor.
2. Verify that these components define a natural transformation from the identity functor on $\mathbf{Vect}_{\mathbb{R}}$ to the double dual functor.



Graded exercise G.6 (NaturalCurry)

Fix a set \mathbf{S} . There are functors $F, G : \mathbf{Set}^{\text{op}} \times \mathbf{Set} \rightarrow \mathbf{Set}$ whose respective actions on objects are

$$F_{\bullet} : \langle \mathbf{A}, \mathbf{B} \rangle \mapsto \text{Hom}_{\mathbf{Set}}(\mathbf{A} \times \mathbf{S}, \mathbf{B}) \quad (56)$$

and

$$G_{\bullet} : \langle \mathbf{A}, \mathbf{B} \rangle \mapsto \text{Hom}_{\mathbf{Set}}(\mathbf{A}, \mathbf{B}^{\mathbf{S}}). \quad (57)$$

On morphisms, F acts as follows. Given a morphism $\langle f^{\text{op}}, g \rangle : \langle \mathbf{A}, \mathbf{B} \rangle \rightarrow \langle \mathbf{A}', \mathbf{B}' \rangle$ in $\mathbf{Set}^{\text{op}} \times \mathbf{Set}$, the function

$$F(\langle f^{\text{op}}, g \rangle) : \text{Hom}_{\mathbf{Set}}(\mathbf{A} \times \mathbf{S}, \mathbf{B}) \rightarrow \text{Hom}_{\mathbf{Set}}(\mathbf{A}' \times \mathbf{S}, \mathbf{B}')$$

takes any function $\varphi : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{B}$ and maps it to the function

$$\langle f, \text{id}_{\mathbf{S}} \rangle ; \varphi ; g : \mathbf{A}' \times \mathbf{S} \rightarrow \mathbf{B}'. \quad (58)$$

Let us also define the action of \mathbf{G} on morphisms. Given again a morphism $\langle f^{\text{op}}, g \rangle : \langle \mathbf{A}, \mathbf{B} \rangle \rightarrow \langle \mathbf{A}', \mathbf{B}' \rangle$ in $\mathbf{Set}^{\text{op}} \times \mathbf{Set}$, the function

$$\mathbf{G}(\langle f^{\text{op}}, g \rangle) : \text{Hom}_{\mathbf{Set}}(\mathbf{A}, \mathbf{B}^{\mathbf{S}}) \rightarrow \text{Hom}_{\mathbf{Set}}(\mathbf{A}', \mathbf{B}'^{\mathbf{S}})$$

takes any function $\varphi : \mathbf{A} \rightarrow \mathbf{B}^{\mathbf{S}}$ and maps it to the function

$$f ; \varphi ; g_* : \mathbf{A}' \rightarrow \mathbf{B}'^{\mathbf{S}} \quad (59)$$

where g_* is the function

$$g_* : \mathbf{B}^{\mathbf{S}} \rightarrow \mathbf{B}'^{\mathbf{S}}, \psi \mapsto \psi ; g. \quad (60)$$

Now, recall that we can “curry” any function $f : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{B}$ to get a function $\bar{f} : \mathbf{A} \rightarrow \mathbf{B}^{\mathbf{S}}$, where $\bar{f}(x)$ may be thought of as a partial evaluation of f .

Your task in this exercise: show that the functions

$$\alpha_{\langle \mathbf{A}, \mathbf{B} \rangle} : \text{Hom}_{\mathbf{Set}}(\mathbf{A} \times \mathbf{S}, \mathbf{B}) \rightarrow \text{Hom}_{\mathbf{Set}}(\mathbf{A}, \mathbf{B}^{\mathbf{S}}), f \mapsto \bar{f}, \quad (61)$$

where \bar{f} is the “curried” version of f , are the components of a natural transformation $\alpha : \mathbf{F} \Rightarrow \mathbf{G}$.



24. Adjunctions

24.1 Formal concept analysis	326
24.2 Galois connections	332
24.3 Adjunctions: hom-set definition	334
24.4 Adjunctions: (co)unit definition	335
24.5 Product-Hom adjunction	337
24.6 Free-forgetful adjunction	338
24.7 Relating the two definitions	340

The *Battle of Surfaces* was a men's tennis exhibition match that was held on May 2, 2007, between the Swiss Roger Federer and Rafael Nadal, respectively number 1 and 2 in the world in men's singles. Federer preferred grass—he was 5 years unbeaten on that terrain. Nadal preferred clay—he was 3 years unbeaten. To check who would win when averaging out the terrain, the match was played on a unique court with a clay surface on one side of the net and grass on the other.

24.1. Formal concept analysis

Formal concept analysis (FCA) is a mathematical theory which formalizes the relationships, and in particular hierarchies, that appear when we consider a set of *things* – these are called *objects* in FCA – together with a set of *attributes* that these things may or may not have. (We will use the word “things” instead of “objects”, because we are already using the word “objects” in the category-theory sense.)

The most basic set-up for formal concept analysis is to start with a triple $\langle \mathbf{G}, \mathbf{M}, \mathbf{I} \rangle$, where \mathbf{G} is a set of things (“G” stands for the German word “Gegenstände”), \mathbf{M} is a set of attributes (“M” stands for the German word “Merkmale”), and $\mathbf{I} \subseteq \mathbf{G} \times \mathbf{M}$ is a relation that encodes which objects are associated with which attributes (“I” stands for the German word “Inzidenz”). The triple $\langle \mathbf{G}, \mathbf{M}, \mathbf{I} \rangle$ is called a *formal context*.

Here is a (very simplified) example in the context of “private means of transportation”. We consider the set of things to be the following means of transportation

$$\mathbf{G} = \{\text{classic car, hybrid car, electric car, classic bike, hybrid bike, electric scooter, skateboard}\} \quad (1)$$

and we consider the set of attributes

$$\mathbf{M} = \{\text{fast, electric, gas, muscle, cheap}\}, \quad (2)$$

which describe aspects such how the means of transportation are powered, their relative cost, or if they can go fast enough to move on a highway, for example. We define the relation $\mathbf{I} \subseteq \mathbf{G} \times \mathbf{M}$ via the following table

	fast	electric	gas	muscle	cheap
classic car	×		×		
hybrid car	×	×	×		
electric car	×	×			
classic bike				×	×
hybrid bike		×		×	
electric scooter		×			×
skateboard				×	×

where a cross “ \times ” indicates when a thing and an attribute are associated with each other.

Induced monotone maps

For each element x of \mathbf{G} , we can consider the set $I_{\sharp}(\{x\})$ of attributes that are associated with x . This corresponds to reading off in the above table where there are crosses “ \times ” in the row labeled by x . For example

$$I_{\sharp}(\{\text{hybrid car}\}) = \{\text{fast, electric, gas}\}. \quad (3)$$

More generally, given a subset $\mathbf{A} \subseteq \mathbf{G}$, we can consider the largest set $I_{\sharp}(\mathbf{A})$ of attributes which all elements of \mathbf{A} have in common. For example

$$I_{\sharp}(\{\text{hybrid car, electric car}\}) = \{\text{fast, electric}\}. \quad (4)$$

Here, “gas” is not an element of $I_{\sharp}(\{\text{hybrid car, electric car}\})$ because “gas” is associated with hybrid car but not with electric car.

Similarly,

$$I_{\sharp}(\{\text{classic car, hybrid car, electric car}\}) = \{\text{fast}\}. \quad (5)$$

Also, we have, for example,

$$\text{I}_{\sharp}(\{\text{classic car, classic bike}\}) = \emptyset \quad (6)$$

because “classic car” and “classic bike” have *no* attributes from the set \mathbf{M} in common. In general we have

$$\text{I}_{\sharp}(\mathbf{A}) = \bigcap_{x \in \mathbf{A}} \text{I}_{\sharp}(\{x\}). \quad (7)$$

and the operation “ I_{\sharp} ” defines a function

$$\text{I}_{\sharp} : \text{Pow } \mathbf{G} \rightarrow \text{Pow } \mathbf{M}. \quad (8)$$

Observe that the larger \mathbf{A} is, the smaller $\text{I}_{\sharp}(\mathbf{A})$ will be. Formulated more mathematically, we have

$$\mathbf{A} \subseteq \mathbf{A}' \implies \text{I}_{\sharp}(\mathbf{A}) \supseteq \text{I}_{\sharp}(\mathbf{A}'). \quad (9)$$

Another way of saying this is to say that I_{\sharp} is a monotone map of posets

$$\text{I}_{\sharp} : (\text{Pow } \mathbf{G}, \subseteq) \rightarrow (\text{Pow } \mathbf{M}, \supseteq). \quad (10)$$

Or, equivalently, we can say that I_{\sharp} is a monotone map

$$\text{I}_{\sharp} : (\text{Pow } \mathbf{G}, \subseteq) \rightarrow (\text{Pow } \mathbf{M}, \subseteq)^{\text{op}}. \quad (11)$$

Note that we can also define a similar map in the other direction: there is a function

$$\text{I}_{\flat} : \text{Pow } \mathbf{M} \rightarrow \text{Pow } \mathbf{G} \quad (12)$$

defined such that for any subset $\mathbf{B} \subseteq \mathbf{M}$, the set $\text{I}_{\flat}(\mathbf{B})$ is the largest set of elements of \mathbf{G} such that the attributes in \mathbf{B} apply to all of the elements of $\text{I}_{\flat}(\mathbf{B})$. For example,

$$\text{I}_{\flat}(\{\text{muscle, cheap}\}) = \{\text{classic bike, skateboard}\}. \quad (13)$$

The map I_{\flat} is also order-reversing with respect to inclusion of sets: if we start with a larger set of attributes, then set of things that these all apply to will be smaller. Thus we have a monotone map

$$\text{I}_{\flat} : (\text{Pow } \mathbf{M}, \subseteq) \rightarrow (\text{Pow } \mathbf{G}, \subseteq). \quad (14)$$

We will also want to use its opposite, the monotone map

$$\text{I}_{\flat}^{\text{op}} : (\text{Pow } \mathbf{M}, \subseteq) \rightarrow (\text{Pow } \mathbf{G}, \subseteq)^{\text{op}}. \quad (15)$$

In the following, we will try to keep track of when there is an superscript “ $(-)^{\text{op}}$ ”; however sometimes it will be convenient to use the notations I_{\sharp} and I_{\flat} both for these maps *and* their opposites (in particular, on the level of objects they are the same function).

A key observation is that I_{\sharp} and I_{\flat} are “complementary” in the following sense. For any $\mathbf{A} \subseteq \mathbf{G}$ and any $\mathbf{B} \subseteq \mathbf{M}$ we have

$$\text{I}_{\sharp}(\mathbf{A}) \supseteq \mathbf{B} \Leftrightarrow \mathbf{A} \subseteq \text{I}_{\flat}(\mathbf{B}). \quad (16)$$

This equivalence formalizes the (nearly tautological-seeming) statement that a set \mathbf{B} is contained in the largest set of attributes which apply to all members of \mathbf{A} (meaning: the attributes \mathbf{B} apply to all elements of \mathbf{A}) if, and only if, \mathbf{A} is contained in the largest set of things to which all attributes \mathbf{B} apply (again meaning: the

attributes \mathbf{B} apply to all elements of \mathbf{A}).

Despite seeming tautological, we can use the equivalence (16) to make non-trivial observations. One consequence of (16) is that the monotone maps

$$I_{\sharp} \circ I_b : \langle \text{Pow } \mathbf{G}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{G}, \subseteq \rangle \quad (17)$$

and

$$I_b^{\text{op}} \circ I_{\sharp}^{\text{op}} : \langle \text{Pow } \mathbf{M}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{M}, \subseteq \rangle. \quad (18)$$

are examples of what are called a *closure operator* and *interior operator*, respectively.

Closure and interior operators

Definition 24.1

Let $\mathbf{P} = \langle \mathbf{P}, \leq \rangle$ be poset. A *closure operator* on \mathbf{P} is

Constituents

1. a monotone map $f : \mathbf{P} \rightarrow \mathbf{P}$;

Conditions

1. Extensivity: $x \leq f(x) \quad \forall x \in \mathbf{P}$;
2. Idempotence: $f(f(x)) = f(x) \quad \forall x \in \mathbf{P}$.

Definition 24.2

Let $\mathbf{P} = \langle \mathbf{P}, \leq \rangle$ be poset. An *interior operator* on \mathbf{P} is

Constituents

1. a monotone map $f : \mathbf{P} \rightarrow \mathbf{P}$;

Conditions

1. Intensivity: $f(x) \leq x \quad \forall x \in \mathbf{P}$;
2. Idempotence: $f(f(x)) = f(x) \quad \forall x \in \mathbf{P}$.

The notions of closure and interior operator are dual in the following sense.

Lemma 24.3. If $f : \mathbf{P} \rightarrow \mathbf{P}$ is a closure (interior) operator, then $f^{\text{op}} : \mathbf{P}^{\text{op}} \rightarrow \mathbf{P}^{\text{op}}$ is an interior (closure) operator.

In this section, for simplicity, we will work mainly in terms of closure operators.

Lemma 24.4. The monotone maps

$$I_{\sharp} \circ I_b : \langle \text{Pow } \mathbf{G}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{G}, \subseteq \rangle \quad (19)$$

and

$$I_b^{\text{op}} \circ I_{\sharp}^{\text{op}} : \langle \text{Pow } \mathbf{M}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{M}, \subseteq \rangle. \quad (20)$$

are closure operators.

Proof. Let's check that $I_{\sharp} \circ I_b : \langle \text{Pow } \mathbf{G}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{G}, \subseteq \rangle$ is a closure operator, using (16). We omit the proof for $I_b^{\text{op}} \circ I_{\sharp}^{\text{op}}$, which may be done analogously.

To show the first condition in the definition of closure operator, fix a set of things $\mathbf{A} \subseteq \mathbf{G}$. In the situation of (16), choose $\mathbf{B} = I_{\sharp}(\mathbf{A})$. Since $I_{\sharp}(\mathbf{A}) \supseteq I_{\sharp}(\mathbf{A})$ is true, (16) implies that $\mathbf{A} \subseteq I_b(I_{\sharp}(\mathbf{A})) = (I_{\sharp} \circ I_b)(\mathbf{A})$.

Now let's consider the second condition. Applying the monotone map $I_{\sharp} \circ I_b$

to the relation $\mathbf{A} \subseteq (I_{\sharp} ; I_b)(\mathbf{A})$, we have

$$(I_{\sharp} ; I_b)(\mathbf{A}) \subseteq (I_{\sharp} ; I_b ; I_{\sharp} ; I_b)(\mathbf{A}). \quad (21)$$

Thus we are finished when we show the inclusion in the other direction. By the first condition, we know that

$$I_{\sharp}(\mathbf{A}) \subseteq I_{\sharp}(I_b(I_{\sharp}(\mathbf{A}))) = (I_{\sharp} ; I_b ; I_{\sharp})(\mathbf{A}). \quad (22)$$

Applying the order-reversing map I_b to both sides of this inclusion then gives

$$(I_{\sharp} ; I_b)(\mathbf{A}) \supseteq (I_{\sharp} ; I_b ; I_{\sharp} ; I_b)(\mathbf{A}) \quad (23)$$

as desired. \square

Closure and interior operators arise in various contexts in mathematics. Often we are interested in the elements which are in the images of these operators. These are called *closed elements* and *open elements*, respectively. We will use the term *fixed-points* to refer to both of these cases without needing to specify whether we are working with a closure or an interior operator.

Definition 24.5

Let $\mathbf{P} = \langle \mathbf{P}, \leq \rangle$ be a poset, $f : \mathbf{P} \rightarrow \mathbf{P}$ a monotone map, and $x \in \mathbf{P}$ an arbitrary element of \mathbf{P} .

If f is a closure (interior) operator, then $f(x) \in \mathbf{P}$ is called the *closure* (*interior*) of x , and x is called *closed* (*open*) if $f(x) = x$. In both cases, when $f(x) = x$, we say that x is a fixed-point of f .

The set of fixed-points of f will be denoted \mathbf{P}_f , or by \mathbf{P}_{fix} when the operator f in question is clear.

Remark 24.6. Note that if $f : \mathbf{P} \rightarrow \mathbf{P}$ is a closure or interior operator, then the set of fixed points \mathbf{P}_{fix} coincides with the image of f .

On the one hand, any element y of the form $y = f(x)$ is a fixed-point, because

$$f(f(x)) = f(x) \quad (24)$$

by the idempotence property.

On the other hand, if $y \in \mathbf{P}$ is a fixed point, then by definition $y = f(y)$ is in the image of f .

Remark 24.7. If x is a fixed-point of a closure/interior operator $f : \mathbf{P} \rightarrow \mathbf{P}$, then x is also a fixed-point of $f^{\text{op}} : \mathbf{P}^{\text{op}} \rightarrow \mathbf{P}^{\text{op}}$.

Returning now to formal concept analysis, let's look at closures and closed elements for the closure operators

$$I_{\sharp} ; I_b : \langle \text{Pow } \mathbf{G}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{G}, \subseteq \rangle$$

and

$$I_b^{\text{op}} ; I_{\sharp}^{\text{op}} : \langle \text{Pow } \mathbf{M}, \subseteq \rangle \rightarrow \langle \text{Pow } \mathbf{M}, \subseteq \rangle$$

in terms of our simple example.

For example, let

$$\mathbf{A} = \{\text{classic car, electric car}\}. \quad (25)$$

Then

$$I_{\sharp}(\mathbf{A}) = \{\text{fast}\} \quad (26)$$

and

$$I_b(I_\sharp(\text{A})) = I_b(\{\text{fast}\}) = \{\text{classic car, hybrid car, electric car}\}. \quad (27)$$

So $\text{A} = \{\text{classic car, electric car}\}$ is not a closed element of $\langle \text{Pow } \mathbf{G}, \subseteq \rangle$. Its closure contains the element “hybrid car” which is not in A .

Or consider

$$\text{B} = \{\text{electric, muscle}\}. \quad (28)$$

Then

$$I_b(\text{B}) = \{\text{hybrid bike}\} \quad (29)$$

and

$$I_\sharp(I_b(\text{B})) = I_\sharp(\{\text{hybrid bike}\}) = \{\text{electric, muscle}\} = \text{B}. \quad (30)$$

We find here that $\{\text{hybrid bike}\}$ is a closed element of $\langle \text{Pow } \mathbf{M}, \subseteq \rangle$.

In general, given a set A of things, its closure $(I_\sharp \circ I_b)(\text{A})$ is the largest set of things that share the attributes in $I_\sharp(\text{A})$. And $I_\sharp(\text{A})$ is the largest set of attributes shared by A . Thus we may say:

$(I_\sharp \circ I_b)(\text{A})$ is the maximal set of things that share the same attributes as are shared by A .

Or, put another way, taking the closure of A is a way of enlarging A without decreasing the set of associated shared attributes. Closing A is adding those things to A that come “for free” in the sense that, by adding them, we are not losing shared attributes.

A similar point of view of course also applies to closing sets of attributes with respect to the closure operator $I_b \circ I_\sharp$.

Concepts

Definition 24.8

Let $\langle \mathbf{G}, \mathbf{M}, \mathcal{I} \rangle$ be a formal context in the sense of formal concept analysis. A *concept* is a pair $\langle \text{A}, \text{B} \rangle \in \text{Pow } \mathbf{G} \times \text{Pow } \mathbf{M}$ such that

$$I_\sharp(\text{A}) = \text{B} \quad \text{and} \quad I_b(\text{B}) = \text{A}. \quad (31)$$

For a concept $\langle \text{A}, \text{B} \rangle$, the set A of things is called the *extent* of the concept, and the set B of attributes is called the *intent* of the concept.

We denote the set of all concepts for the context $\langle \mathbf{G}, \mathbf{M}, \mathcal{I} \rangle$ by $\mathcal{B}(\mathbf{G}, \mathbf{M}, \mathcal{I})$. (Here “B” comes from the German term “Begriffe”.)

The set $\mathcal{B}(\mathbf{G}, \mathbf{M}, \mathcal{I})$ of concepts for a formal context has a natural partial order structure. We set

$$\langle \text{A}_1, \text{B}_1 \rangle \leq \langle \text{A}_2, \text{B}_2 \rangle \quad (32)$$

if $\text{A}_1 \subseteq \text{A}_2$ and $\text{B}_1 \supseteq \text{B}_2$. (In fact, by the definition of a concept, if one of the latter inclusions holds, then so must the other, so we only need to require one of them.) When (32) holds, we say that $\langle \text{A}_1, \text{B}_1 \rangle$ is a *subconcept* of $\langle \text{A}_2, \text{B}_2 \rangle$.

Lemma 24.9. If $\langle \text{A}, \text{B} \rangle$ is a concept, then A and B are closed elements of $\langle \text{Pow } \mathbf{G}, \subseteq \rangle$ and $\langle \text{Pow } \mathbf{M}, \subseteq \rangle$, respectively.

Proof. For A we have

$$I_b(I_\sharp(\text{A})) = I_b(\text{B}) = \text{A} \quad (33)$$

using both the equations (31). The case for B is analogous. \square

Lemma 24.10. If $\mathbf{A} \in \langle \text{Pow } \mathbf{G}, \subseteq \rangle$ is closed, then $I_{\sharp}(\mathbf{A})$ is closed and $\langle \mathbf{A}, I_{\sharp}(\mathbf{A}) \rangle$ is a concept.

Similarly, if $\mathbf{B} \in \langle \text{Pow } \mathbf{M}, \subseteq \rangle$ is closed, then $I_b(\mathbf{B})$ is closed and $\langle I_b(\mathbf{B}), \mathbf{B} \rangle$ is a concept.

Proof. We show only the first statement. We have

$$(I_b \circ I_{\sharp})(I_{\sharp}(\mathbf{A})) = I_{\sharp}(I_b(I_{\sharp}(\mathbf{A}))) = I_{\sharp}((I_b \circ I_{\sharp})(\mathbf{A})) = I_{\sharp}(\mathbf{A}), \quad (34)$$

so $I_{\sharp}(\mathbf{A})$ is closed. That $\langle \mathbf{A}, I_{\sharp}(\mathbf{A}) \rangle$ is a concept is clear, since $I_b(I_{\sharp}(\mathbf{A})) = \mathbf{A}$. \square

Lemma 24.11. The posets of fixed points $\langle \text{Pow } \mathbf{G}_{\text{fix}}, \subseteq \rangle$ and $\langle \text{Pow } \mathbf{M}_{\text{fix}}, \subseteq \rangle^{\text{op}}$ are isomorphic via the restrictions of I_{\sharp} and I_b , and each is isomorphic to the poset $\langle \mathcal{B}(\mathbf{G}, \mathbf{M}, I), \leq \rangle$ via its projections onto its first and second factors, respectively.

Proof. This follows from Lemma 24.9, Lemma 24.10, and the definition of the ordering on $\langle \mathcal{B}(\mathbf{G}, \mathbf{M}, I), \leq \rangle$. \square

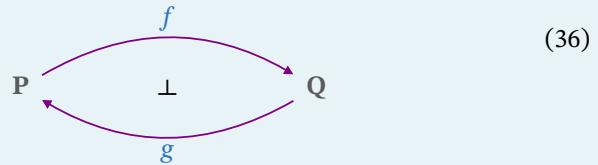
24.2. Galois connections

Definition 24.12 (Monotone Galois Connection)

A (monotone) *Galois connection* between posets P and Q is a pair of monotone maps $f : P \rightarrow Q$ and $g : Q \rightarrow P$ such that for all $p \in P, q \in Q$:

$$\frac{f(p) \leq_Q q}{p \leq_P g(q)}. \quad (35)$$

In this case f is called the *left adjoint* (or *lower adjoint*) and g is called the *right adjoint* (or *upper adjoint*). We use the short-hand notation $f \dashv g$ to say that f and g form a Galois connection, or we draw a globular diagram like so:



Lemma 24.13. Monotone maps $f : P \rightarrow Q$ and $g : Q \rightarrow P$ form a Galois connection if and only if the following hold:

1. $p \leq_P g(f(p)) \quad \forall p \in P;$
2. $f(g(q)) \leq_Q q \quad \forall q \in Q.$

Definition 24.14 (Antitone Galois Connection)

An *antitone Galois connection* between P and Q is a pair of antitone maps $f : P \rightarrow Q$ and $g : Q \rightarrow P$ such that for all $p \in P, q \in Q$:

$$\frac{q \leq_Q f(p)}{p \leq_P g(q)}. \quad (37)$$

Remark 24.15. The underlying function of an antitone map $f : P \rightarrow Q$ defines a monotone map $f : P \rightarrow Q^{\text{op}}$ (or a monotone map $f : P^{\text{op}} \rightarrow Q$). Every antitone Galois connection $f : P \rightarrow Q$ and $g : Q \rightarrow P$ defines a Galois connection $f : P \rightarrow Q^{\text{op}}$ and $g : Q^{\text{op}} \rightarrow P$.

Because of the above remark, and because we prefer to work with monotone maps (since they are morphisms of posets), we will mainly focus on (monotone) Galois connections. However, it is useful to be aware of the antitone definition, since it is sometimes used in the literature and sometimes more natural in the context of certain examples.

Lemma 24.16. Antitone maps $f : P \rightarrow Q$ and $g : Q \rightarrow P$ form an antitone Galois connection if and only if the following hold:

1. $p \leq_P g(f(p)) \quad \forall p \in P;$
2. $q \leq_Q f(g(q)) \quad \forall q \in Q.$

Examples

Induced closure and interior operators

Lemma 24.17. If $f : P \rightarrow Q$ and $g : Q \rightarrow P$ form a Galois connection, then

$$f \circ g : P \rightarrow P \tag{38}$$

is a closure operator and

$$g \circ f : Q \rightarrow Q \tag{39}$$

is an interior operator.

24.3. Adjunctions: hom-set definition

In this section we give a definition of adjunction which can be viewed as an analogy with the following situation in linear algebra. Suppose V and W are finite-dimensional real vector spaces, equipped with inner products $(-, -)_V$ and $(-, -)_W$, respectively. The adjoint of a linear map $F : V \rightarrow W$ is a linear map $F^* : W \rightarrow V$ such that

$$(Fv, w)_W = (v, F^*w)_V, \quad \forall v \in V, w \in W. \quad (40)$$

Definition 24.18 (Adjunction, Version 1)

Let \mathbf{C} and \mathbf{D} be categories. An *adjunction* from \mathbf{C} to \mathbf{D} is given by the following data:

1. A functor $L : \mathbf{C} \rightarrow \mathbf{D}$, called the *left adjoint*;
2. A functor $R : \mathbf{D} \rightarrow \mathbf{C}$, called the *right adjoint*;
3. A natural isomorphism $\tau : \text{Hom}_{\mathbf{D}}(L-; -) \Rightarrow \text{Hom}_{\mathbf{C}}(-; R-)$ between functors $\mathbf{C}^{\text{op}} \times \mathbf{D} \rightarrow \mathbf{Set}$.

We use the notation $L \dashv R$ to indicate that L and R form an adjunction, with L the left adjoint and R the right adjoint.

24.4. Adjunctions: (co)unit definition

Recall from Def. 34.1: in a category \mathbf{C} , a morphism $f : X \rightarrow Y$ is an isomorphism if there exists a morphism $g : Y \rightarrow X$ such that

$$f \circ g = \text{id}_X \quad \text{and} \quad g \circ f = \text{id}_Y. \quad (41)$$

Now let's think about this definition in the case where \mathbf{C} is the category \mathbf{Cat} of categories. We will consider weakenings of the notion of isomorphism in this setting, and this will lead to a second (but equivalent) definition of adjunction. The precise relationship between the two definitions will be spelled out Section 24.7.

The idea of “weakening” the notion of isomorphism of categories is as follows. Given functors



(42)

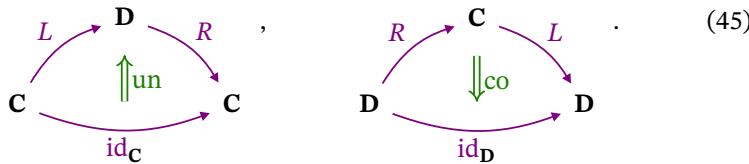
instead of requiring the equations

$$\text{id}_{\mathbf{C}} = L \circ R \quad \text{and} \quad R \circ L = \text{id}_{\mathbf{D}}, \quad (43)$$

we replace the equality symbols with 2-morphisms, in this way:

$$\text{id}_{\mathbf{C}} \xrightarrow{\text{un}} L \circ R \quad \text{and} \quad R \circ L \xrightarrow{\text{co}} \text{id}_{\mathbf{D}}. \quad (44)$$

The last two relationships can also be depicted in the following more geometric manner:



(45)

Definition 24.19 (Equivalence of categories)

Let \mathbf{C} and \mathbf{D} be categories. An *equivalence of categories* between \mathbf{C} and \mathbf{D} is the following data:

1. A functor $L : \mathbf{C} \rightarrow \mathbf{D}$;
2. A functor $R : \mathbf{D} \rightarrow \mathbf{C}$;
3. Natural isomorphisms $\text{un} : \text{id}_{\mathbf{C}} \Rightarrow L \circ R$ and $\text{co} : R \circ L \Rightarrow \text{id}_{\mathbf{D}}$.

Definition 24.20 (Adjunction, Version 2)

Let \mathbf{C} and \mathbf{D} be categories. An *adjunction* from \mathbf{C} to \mathbf{D} is given by the following data, satisfying the following conditions.

Data:

1. A functor $L : \mathbf{C} \rightarrow \mathbf{D}$ (the *left adjoint*);
2. A functor $R : \mathbf{D} \rightarrow \mathbf{C}$ (the *right adjoint*);
3. Two natural transformations $\text{un} : \text{id}_{\mathbf{C}} \Rightarrow L \circ R$ and $\text{co} : R \circ L \Rightarrow \text{id}_{\mathbf{D}}$

Conditions:

1. For all objects X of \mathbf{C} , it holds that

$$\text{Lun}_X \circ \text{co}_{LX} = \text{id}_{LX} \quad \text{and} \quad \text{un}_{RY} \circ R\text{co}_Y = \text{id}_{RY}, \quad (46)$$

which means that the following diagrams commute:

$$\begin{array}{ccc} LX & \xrightarrow{\text{Lun}_X} & LRLX \\ & \searrow \text{id}_{LX} & \downarrow \text{co}_{LX} \\ & LX & \end{array} \quad \begin{array}{ccc} RY & \xrightarrow{\text{un}_{RY}} & RLRY \\ & \searrow \text{id}_{RY} & \downarrow R\text{co}_Y \\ & RY & \end{array} \quad (47)$$

The natural transformations un and co are called the *unit* and *counit* of the adjunction.

Definition 24.21 (Adjoint equivalence)

An adjunction is called an *adjoint equivalence* if the unit and counit are natural isomorphisms.

Remark 24.22. The conditions (triangle identities) from Def. 24.20 are “hidden” in Def. 24.18 in the condition that τ be a natural isomorphism. In Section 24.7 we spell out how the two definitions are related.

24.5. Example of a “Product-Hom” adjunction

We will consider an adjunction between the category **Set** and itself which is a basic representative of a certain “type” of adjunction that appears all over mathematics. This type of adjunction might be called a “Product-Hom” adjunction.

Fix a set **B** and consider the functors **F** and **G** which act as follows. Given a set **A**,

$$\mathbf{F}(\mathbf{A}) = \mathbf{B} \times \mathbf{A} \quad (48)$$

and

$$\mathbf{G}(\mathbf{A}) = \text{Hom}_{\mathbf{Set}}(\mathbf{B}; \mathbf{A}) =: \mathbf{A}^{\mathbf{B}}. \quad (49)$$

Given a morphism $f : \mathbf{A} \rightarrow \mathbf{A}'$,

$$\mathbf{F}(f) = \text{id}_{\mathbf{B}} \times f \quad (50)$$

and

$$\begin{aligned} \mathbf{G}(f) : \mathbf{A}^{\mathbf{B}} &\rightarrow \mathbf{A}'^{\mathbf{B}} \\ g &\mapsto g \circ f. \end{aligned} \quad (51)$$

These functors are part of an adjunction

$$\begin{array}{ccc} & \mathbf{B} \times - & \\ \text{Set} & \begin{array}{c} \swarrow \curvearrowright \\ \perp \end{array} & \text{Set} \\ & (-)^{\mathbf{B}} & \end{array} \quad (52)$$

In terms of Def. 24.18, there is a natural isomorphism

$$\tau : \text{Hom}_{\mathbf{Set}}(\mathbf{F}(-); -) \Rightarrow \text{Hom}_{\mathbf{Set}}(-; \mathbf{G}(-)) \quad (53)$$

whose component at $\langle \mathbf{A}, \mathbf{C} \rangle$ is the isomorphism

$$\tau_{\mathbf{A}, \mathbf{C}} : \text{Hom}_{\mathbf{Set}}(\mathbf{B} \times \mathbf{A}; \mathbf{C}) \rightarrow \text{Hom}_{\mathbf{Set}}(\mathbf{A}; \mathbf{C}^{\mathbf{B}}) \quad (54)$$

given by “partial evaluation”. Namely, given $f : \mathbf{B} \times \mathbf{A} \rightarrow \mathbf{C}$, this is mapped by $\tau_{\mathbf{A}, \mathbf{C}}$ to the function $\tau f : \mathbf{A} \rightarrow \mathbf{C}^{\mathbf{B}}$, $a \mapsto f(-, a)$.

In terms of Def. 24.20, the component at \mathbf{A} of the unit and co-unit, respectively, are

$$\begin{aligned} \text{un}_{\mathbf{A}} : \mathbf{A} &\rightarrow (\mathbf{B} \times \mathbf{A})^{\mathbf{B}} \\ a &\mapsto (b \mapsto \langle a, b \rangle) \end{aligned} \quad (55)$$

and

$$\begin{aligned} \text{co}_{\mathbf{A}} : \mathbf{B} \times (\mathbf{A}^{\mathbf{B}}) &\rightarrow \mathbf{A} \\ \langle b, f \rangle &\mapsto f(b) \end{aligned} \quad (56)$$

24.6. Example of a “Free-Forgetful” adjunction

Another “type” of adjunction that appears frequently can be called a “Free-Forgetful” adjunction. Such adjunctions are composed of a “free functor” and a “forgetful functor”. These terms are informal, but the idea is as follows.

A free functor $\mathbf{C} \rightarrow \mathbf{D}$ typically takes an object X of \mathbf{C} and “freely” adds some structure to it. “Free” means that only those structures and conditions are added that are absolutely necessary to make X an object of \mathbf{D} , and otherwise the functor does not impose any constraints or relations.

Conversely, a “forgetful functor” usually starts from an object Y on \mathbf{D} which has some structure, and “forgets” some of this structure, which results in us being able to view Y as an object in \mathbf{C} .

Example 24.23. Any real vector space is built from an underlying set, together with extra structure given by operations (vector addition and scalar multiplication). There is a forgetful functor

$$\mathbf{Vect}_{\mathbb{R}} \rightarrow \mathbf{Set} \quad (57)$$

which maps any vector space to its underlying set of vectors. On the other hand, there is a “free” functor

$$\mathbf{Set} \rightarrow \mathbf{Vect}_{\mathbb{R}}. \quad (58)$$

Given a set A , we can build the “free real vector space generated by A ”. To do this, we think of the elements of A as basis vectors, and we build a vector space by taking formal finite \mathbb{R} -linear combinations of them.

In the following we consider an example in detail where we “freely” generate a category from a directed graph.

Example 24.24. Let \mathbf{Grph} be the category of directed graphs and \mathbf{Cat} the category of (small) categories.

There is a functor $F : \mathbf{Grph} \rightarrow \mathbf{Cat}$ which turns any directed graph $D = \langle V, E, s, t \rangle$ into a category whose objects are the vertices V and whose morphisms are finite directed paths between vertices. This is called the *free category generated by the graph D* (Section 13.6).

There is also a functor $G : \mathbf{Cat} \rightarrow \mathbf{Grph}$ which turns a category C into a graph where the set of vertices is Ob_C and there is a directed edge between vertices for every morphism in C between the corresponding vertices.

Let’s first describe this adjunction via Def. 24.18. The natural isomorphism

$$\tau : \text{Hom}_{\mathbf{Cat}}(F(-); -) \Rightarrow \text{Hom}_{\mathbf{Grph}}(-; G(-)) \quad (59)$$

is the one whose component at $\langle D, C \rangle$ is the isomorphism

$$\tau_{D,C} : \text{Hom}_{\mathbf{Cat}}(F(D); C) \Rightarrow \text{Hom}_{\mathbf{Grph}}(D; G(C)) \quad (60)$$

which assigns to any functor $F : F(D) \rightarrow C$ the morphism of graphs $D : G(C)$ given by restricting F to D and only keeping track of its action on vertices and edges (in other words, we ignore its compositional properties and think of it just as a graph morphism).

Now let’s consider this adjunction from the perspective of Def. 24.20. The component at D of the counit is the morphism of graphs

$$\text{un}_D : D \rightarrow G(F(D)) \quad (61)$$

which includes D into the graph $G(F(D))$. The latter has an edge from the source

to the target of every finite path in D . The paths of length zero are what corresponded to identity morphisms in $F(D)$, and the paths of length one constitute a copy of D inside $G(F(D))$.

What does the unit look like? Its component at \mathbf{C} is a functor

$$\text{co}_{\mathbf{C}} : F(G(\mathbf{C})) \rightarrow \mathbf{C}. \quad (62)$$

The category $F(G(\mathbf{C}))$ is larger than \mathbf{C} : starting with \mathbf{C} , the graph $G(\mathbf{C})$ will contain edges for all the morphisms in \mathbf{C} , but it will forget their compositional interlinking. In particular, for example, it will forget which loops denote identity morphisms (in other words, which morphisms act neutrally) and, more generally, it will forget when different compositions of morphism give the same result. In $F(G(\mathbf{C}))$, then, morphism compositions that might have given the same result in \mathbf{C} will now be distinct. The functor $\text{co}_{\mathbf{C}}$ in a sense “remembers” those relations that were true in \mathbf{C} and it “implements” them by “projecting” $F(G(\mathbf{C}))$ back to \mathbf{C} .

24.7. Relating the two definitions

We start first with the “hom-set definition” of adjunction, and show how to obtain the “(co)unit definition”. Given an adjunction $F \dashv G$ from a category \mathbf{C} to a category \mathbf{D} , we have, by Def. 24.18 a natural isomorphism τ with components

$$\tau_{X,Y} : \text{Hom}_{\mathbf{D}}(F(X); Y) \rightarrow \text{Hom}_{\mathbf{C}}(X; G(Y)). \quad (63)$$

From this data we can construct the unit and counit of the adjunction as follows.

Given an object X of \mathbf{C} , we define

$$\eta_X : X \rightarrow G(F(X)) \quad (64)$$

to be the image under $\tau_{X,F(X)}$ of $\text{id}_{F(X)} \in \text{Hom}_{\mathbf{D}}(F(X); F(X))$.

Given an object Y of \mathbf{D} , we define

$$\varepsilon_Y : F(G(Y)) \rightarrow Y \quad (65)$$

to be the image under $\tau_{G(Y),Y}^{-1}$ of $\text{id}_{G(Y)} \in \text{Hom}_{\mathbf{D}}(G(Y); G(Y))$.

Exercise46. Show that if we define η and ε in terms of their components as above, then they do indeed define natural transformations

$$\eta : \text{id}_{\mathbf{C}} \Rightarrow F \circ G \quad (66)$$

and

$$\varepsilon : G \circ F \Rightarrow \text{id}_{\mathbf{D}} \quad (67)$$

respectively. In other words, check the naturality conditions for η and ε .

See solution on page 341.

Exercise47. Show that η and ε , as defined above, satisfy the triangle identities stated in Def. 24.20.

See solution on page 341.

Now let's start with the “(co)unit definition” of adjunction and see how to obtain the “hom-set definition”.

Given the unit η and counit ε , we can construct the components $\tau_{X,Y}$ of the natural transformation τ as follows. Given $f \in \text{Hom}_{\mathbf{D}}(F(X), Y)$, we define

$$\tau_{X,Y}(f) = \eta_X \circ G(f). \quad (68)$$

Similarly, given $g \in \text{Hom}_{\mathbf{C}}(X, G(Y))$, the inverse component is given by

$$\tau_{X,Y}^{-1}(g) = F(g) \circ \varepsilon_Y. \quad (69)$$

Exercise48. Show that $\tau_{X,Y}$ and $\tau_{X,Y}^{-1}$ are indeed functions which are inverses of each other.

See solution on page 341.

Exercise49. Show that the functions $\tau_{X,Y}$ do assemble to a natural transformation

$$\tau : \text{Hom}_{\mathbf{D}}(F(-), -) \Rightarrow \text{Hom}_{\mathbf{C}}(-, G(-)) \quad (70)$$

between functors $\mathbf{C}^{\text{op}} \times \mathbf{D} \rightarrow \mathbf{Set}$.

See solution on page 341.

Solutions to selected exercises

Solution of Exercise 46.

Solution is missing.



Solution of Exercise 47.

Solution is missing.



Solution of Exercise 48.

Solution is missing.



Solution of Exercise 49.

Solution is missing.



PART H. INTERCONNECTIONS



25. Parallel composition	345
26. Crossing wires	377
27. Feedback	383

The *Sechseläuten* is a traditional spring holiday in the Zurich, Switzerland, usually happening on the 3rd monday of April. The old city guilds meet in the city center for a parade, climax of which is the burning of the “Böögg”, a snowman prepared with explosives, considered a weather oracle for the summer.



25. Parallel composition

25.1 Modeling parallelism	346
25.2 Stacking categories	347
25.3 Functorial stacking categories .	350
25.4 Associative stacking categories .	354
25.5 Monoidal categories	357
25.6 Monoidal functors	364
25.7 Strictification	366
25.8 The case of semicategories	371

Raclette is a Swiss dish, based on heating cheese and scraping off the melted part. In Switzerland, raclette is typically served with potatoes, cornichons, pickled onions, and Fendant wine.

25.1. Modeling parallelism

We have talked a lot about composition, and considered many examples. However, the types of compositions we studied were, so far, mostly of the “in series composition” kind. For instance, we considered the series composition of travel routes (Example 15.1) and trekking routes (Section 15.2), functions (Section 3.4) and relations (Section 4.2), engineering component dependencies (Section 15.4) and Moore machines (Section 18.1), *etc.*

In this chapter, we will consider composition both in series and *in parallel*. For example, given functions $f : A \rightarrow B$ and $g : B \rightarrow C$, because the target set of the function f matches the source set of g they may be composed in series to obtain a function $f \circ g : A \rightarrow C$. On the other hand, any two functions $f_1 : A \rightarrow B$ and $f_2 : C \rightarrow D$ may be composed “in parallel” by taking their cartesian product: we obtain the function $f_1 \times f_2 : A \times C \rightarrow B \times D$. This parallel composition of f_1 and f_2 does not rely on any match-up of target and source sets, but it does rely on the “additional structure” provided by the cartesian product.

Such “additional structure” will be formalized in this chapter using the notion of a *monoidal structure*.

Composing components in parallel is of course a very familiar notion in engineering, and the mathematical concepts we develop here will, in particular, model parallel composition in this engineering sense. In the context of co-design of complex systems, for example, we have seen that series composition corresponds to relating the functionalities of one component to the required resources of a next component.

Parallel composition, on the other hand, will correspond to taking two components and thinking of them as a single component whose functionality and resource space are given by the cartesian products of the respective constituent functionality and resource spaces of the original two components.

In general, a monoidal structure will be a notion of “product” and “neutral element” that a category may be equipped with, in which case such is called a *monoidal category*. One thing that could potentially be confusing at this point is the following. At the beginning of this book, we studied monoids as a basic kind of algebraic gadget whose composition operation (also called multiplication) was generalized to the series composition encoded in the definition of a category. In this chapter, we will also use the basic pattern of a monoid as inspiration, but now for parallel composition! Thus, parallel composition is also “monoid-like”, and hence the name *monoidal structure*.

Types of stacking operations

There are various properties that we can consider for categories equipped with an operation of parallel composition. This leads to a number of definitions; here is a short overview.

1. *Stacking category*: a category in which it is possible to stack two morphisms.
2. *Functorial stacking category*: a stacking category in which the stacking operation is a functor.
3. *Associative stacking category*: a functorial stacking category in which the stacking operation is associative (either strictly, or up to isomorphism).
4. *Monoidal category*: an associative stacking category in which there is a special object, called the monoidal unit, which is neutral for the stacking operation.
5. *Symmetric monoidal category*: a monoidal category equipped with a symmetric way to “cross wires”.

25.2. Stacking categories

So far we have seen how we can compose morphisms “horizontally”:

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{f ; g : X \rightarrow Z}. \quad (1)$$

There are other notions of composition that allow us to compose morphisms by “stacking them vertically”. Given two morphisms

$$f : X \rightarrow Y, \quad (2)$$

$$g : Z \rightarrow U, \quad (3)$$

we will obtain by parallel stacking a morphism

$$(f \otimes g) : (X \otimes Z) \rightarrow (Y \otimes U), \quad (4)$$

where “ \otimes ” and “ \otimes ” are operations to be defined. Note that while in the case of sequential composition there was a compatibility condition to be defined, as the target of the first morphism must be the source of the second morphism, here instead we can stack arbitrary morphisms.

We also expect to be able to stack any number of morphisms. Having a collection of morphisms

$$f_i : X_i \rightarrow Y_i, \quad 1 \leq i \leq n, \quad (5)$$

we expect to be able to obtain the composed morphism

$$(\otimes_{i=1}^n f_i) : (\otimes_{i=1}^n X_i) \rightarrow (\otimes_{i=1}^n Y_i). \quad (6)$$

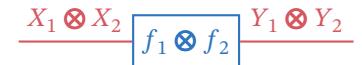


Figure 1.: Stacked morphisms

Definition 25.1 (Stacking category)

A *stacking category* is a category \mathbf{C} with the following additional constituents and properties.

Constituents

- ▷ A stacking operation $\otimes : \text{Ob}_{\mathbf{C}} \times \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{C}}$.
- ▷ A stacking operation $\otimes : \text{Mor}_{\mathbf{C}} \times \text{Mor}_{\mathbf{C}} \rightarrow \text{Mor}_{\mathbf{C}}$.

Conditions

- ▷ The two operations \otimes and \otimes are compatible in the sense that

$$\frac{f_1 : X_1 \rightarrow Y_1 \quad f_2 : X_2 \rightarrow Y_2}{f_1 \otimes f_2 : X_1 \otimes X_2 \rightarrow Y_1 \otimes Y_2}. \quad (7)$$

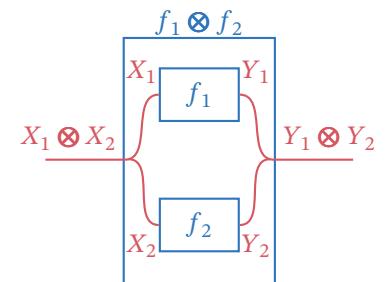


Figure 2.: Stacking string diagrams

In Fig. 1 we have depicted a string diagram of two stacked morphisms. Alternatively, in Fig. 2 we depict the stacking of the string diagrams for f_1 and f_2 , respectively, by stacking their diagrams vertically and drawing a box around them, merging their respective input and output terminals. The outer box denotes $f_1 \otimes f_2$; we think of Fig. 1 as a “black-boxed” version of Fig. 2.

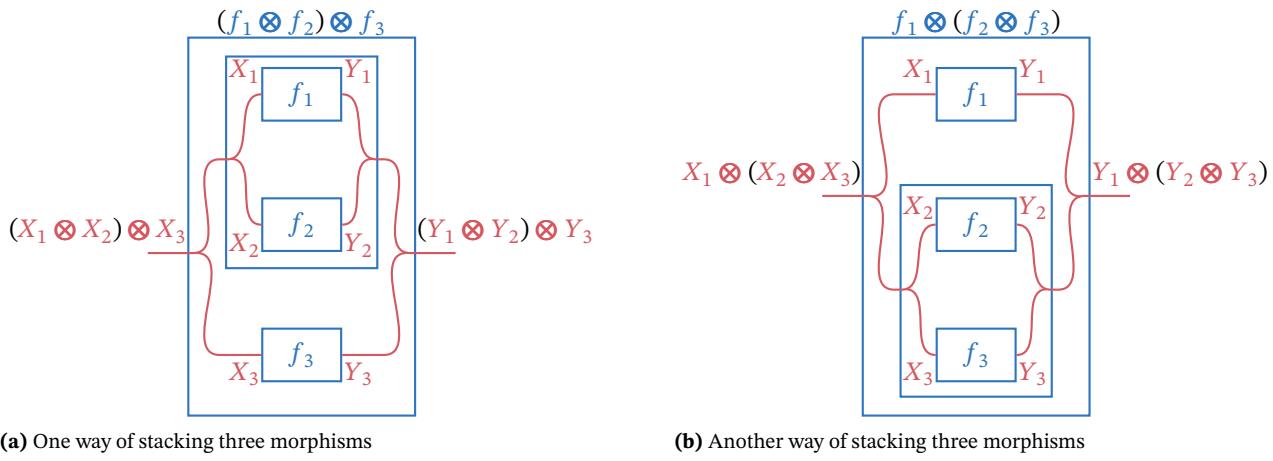
Example 25.2.

The cartesian product of sets and functions

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{A} \times \mathbf{B} \quad f \otimes g = f \times g \quad (8)$$

defines a stacking operation on the category **Set**. Indeed, by the definition of the

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

**Figure 3.**: Stacking three morphisms.

cartesian product of functions, the two stacking layers are compatible:

$$\frac{f : \mathbf{A} \rightarrow \mathbf{C} \quad g : \mathbf{B} \rightarrow \mathbf{D}}{f \times g : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C} \times \mathbf{D}}. \quad (9)$$

$$\langle a, c \rangle \mapsto \langle f(a), g(c) \rangle$$

Example 25.3. The sum of sets and functions

$$\mathbf{A} \otimes \mathbf{B} = \mathbf{A} + \mathbf{B} \quad f \otimes g = f + g \quad (10)$$

also defines a stacking operation on the category **Set**.

Example 25.4. The category of real matrices admits a stacking operation defined by summing dimensions

$$n \otimes m = n + m \quad (11)$$

and combining block matrices like this:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}. \quad (12)$$

Example 25.5. The category **DP** of design problems admits a stacking operation which is defined on objects by taking the product of posets,

$$\mathbf{P} \otimes \mathbf{Q} = \mathbf{P} \times \mathbf{Q}, \quad (13)$$

and on morphisms it is defined by

$$\frac{\mathbf{d} : \mathbf{P}^{\text{op}} \times \mathbf{R} \rightarrow \mathbf{Bool} \quad \mathbf{e} : \mathbf{Q}^{\text{op}} \times \mathbf{S} \rightarrow \mathbf{Bool}}{\mathbf{d} \otimes \mathbf{e} : (\mathbf{P} \otimes \mathbf{Q})^{\text{op}} \times (\mathbf{R} \otimes \mathbf{S}) \rightarrow \mathbf{Bool}}. \quad (14)$$

$$\langle \langle a, c \rangle, \langle b, d \rangle \rangle \mapsto \mathbf{d}(a, c) \wedge \mathbf{e}(c, d)$$

Example 25.6. The following defines a stacking operation for the category **LTI** of LTI systems. On objects the stacking is just addition on the natural numbers (which represent dimensions of input and output spaces):

$$\otimes : \mathbf{Ob}_{\mathbf{LTI}} \times \mathbf{Ob}_{\mathbf{LTI}} \rightarrow \mathbf{Ob}_{\mathbf{LTI}}, \quad (15)$$

$$\langle l, m \rangle \mapsto l + m.$$

On morphisms, the stacking is*

$$\frac{f : l \rightarrow_{\text{LTI}} m \quad g : n \rightarrow_{\text{LTI}} o}{f \otimes g = \langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle : l + n \rightarrow_{\text{LTI}} m + o}. \quad (16)$$

with

$$\text{st} = \begin{bmatrix} \text{st}_f \\ \text{st}_g \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_g \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_g \end{bmatrix}, \quad (17)$$

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_g \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} \mathbf{D}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_g \end{bmatrix}. \quad (18)$$

* For the control engineers out there: the resulting LTI system will have a Relative Gain Array (RGA) matrix corresponding to the identity matrix.

25.3. Functorial stacking categories

Definition 25.7 (Functorial stacking category)

A *functorial stacking category* is a stacking category where the two stacking operations \otimes and \boxtimes are the two components of a functor

$$\boxtimes : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}. \quad (19)$$

In infix notation, this means that

$$(f ; h) \boxtimes (g ; i) = (f \boxtimes g) ; (h \boxtimes i) \quad (20)$$

for all morphisms f, g, h and i (where respectively f and h , and g and i are composable), and that

$$\text{id}_X \boxtimes \text{id}_Y = \text{id}_{X \boxtimes Y} \quad (21)$$

for all objects X, Y of \mathbf{C} .

This describes a sort of commutativity property: we can either first compose horizontally and then vertically, or vice versa; either way, we obtain the same result (Fig. 4).

Example 25.8. The cartesian product of sets and functions is a functorial stacking operation. Suppose we are given functions f, g, h and i (where respectively f and h , and g and i are composable). On the one hand,

$$((f ; g) \times (h ; k))(\langle a, b \rangle) = \langle (f ; g)(a), (h ; k)(b) \rangle \quad (22)$$

$$= \langle g(f(a)), k(h(b)) \rangle \quad (23)$$

while on the other hand,

$$((f \times h) ; (g \times k))(\langle a, b \rangle) = (g \times k)((f \times h)(\langle a, b \rangle)) \quad (24)$$

$$= (g \times k)(\langle f(a), h(b) \rangle) \quad (25)$$

$$= \langle g(f(a)), k(h(b)) \rangle \quad (26)$$

Example 25.9. The sum of sets and functions is a functorial stacking operation.

Example 25.10. Consider the stacking operation defined previously for the category of real matrices. It is functorial stacking:

$$(\mathbf{A} ; \mathbf{B}) \boxtimes (\mathbf{C} ; \mathbf{D}) = (\mathbf{B}\mathbf{A}) \boxtimes (\mathbf{D}\mathbf{C}) \quad (27)$$

$$= \begin{bmatrix} \mathbf{B}\mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}\mathbf{C} \end{bmatrix} \quad (28)$$

$$(\mathbf{A} \boxtimes \mathbf{C}) ; (\mathbf{B} \boxtimes \mathbf{D}) = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} ; \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \quad (29)$$

$$= \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix} \quad (30)$$

$$= \begin{bmatrix} \mathbf{B}\mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}\mathbf{C} \end{bmatrix} \quad (31)$$

Example 25.11. The stacking operation previously defined for the category \mathbf{DP} of design problems is functorial.

Stacking for LTI is almost functorial

Example 25.12. We want to show that **LTI**, equipped with the defined stacking operations, is almost a functorial stacking semicategory, but not quite. Given morphisms $f : l \rightarrow m, h : m \rightarrow n, g : o \rightarrow p, i : p \rightarrow q$, we would need to have

$$(f \circ h) \otimes (g \circ i) = (f \otimes g) \circ (h \otimes i).$$

This, however, is not true. Let's see this by looking at the first matrix component of the LTI system. On one hand we have:

$$\mathbf{A}_{(f \circ h) \otimes (g \circ i)} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_h \mathbf{C}_f & \mathbf{A}_h & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_g & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_i \mathbf{C}_g & \mathbf{A}_i \end{bmatrix}. \quad (32)$$

On the other hand we have:

$$\mathbf{A}_{(f \otimes g) \circ (h \otimes i)} = \begin{bmatrix} \mathbf{A}_f & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_g & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_h \mathbf{C}_f & \mathbf{0} & \mathbf{A}_h & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_i \mathbf{C}_g & \mathbf{0} & \mathbf{A}_i \end{bmatrix}. \quad (33)$$

These two are different, and will therefore describe different systems. However, the two matrices just differ by two permutations, which can be expressed via an invertible linear transformation \mathbf{T} as follows:

$$\mathbf{A}_{(f \otimes g) \circ (h \otimes i)} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \cdot \mathbf{A}_{(f \circ h) \otimes (g \circ i)} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (34)$$

It turns out that $(f \circ h) \otimes (g \circ i)$ and $(f \otimes g) \circ (h \otimes i)$ are equivalent systems (Def. 18.21), even though they are not equal. In particular, although **LTI** is not a functorial stacking semicategory, by Lemma 18.29 $(f \circ h) \otimes (g \circ i)$ and $(f \otimes g) \circ (h \otimes i)$ have the same action, and hence **LTI** is, in one sense, “morally” functorial.

Graded exercise H.1 (StackingLTI)

Consider the category of finite-dimensional linear time-invariant systems defined in Def. 18.22 with the stacking defined above.

Your task: supposing that morphisms $f : l \rightarrow m, h : m \rightarrow n, g : o \rightarrow p, i : p \rightarrow q$ are given, compute the matrices

$$\mathbf{A}_{(f \circ h) \otimes (g \circ i)} \quad (35)$$

and

$$\mathbf{A}_{(f \otimes g) \circ (h \otimes i)} \quad (36)$$

associated with the morphisms $(f \circ h) \otimes (g \circ i)$ and $(f \otimes g) \circ (h \otimes i)$, respectively.

A very non-functorial stacking

We describe here a category that will serve as an example of a stacking category that is not functorial.

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

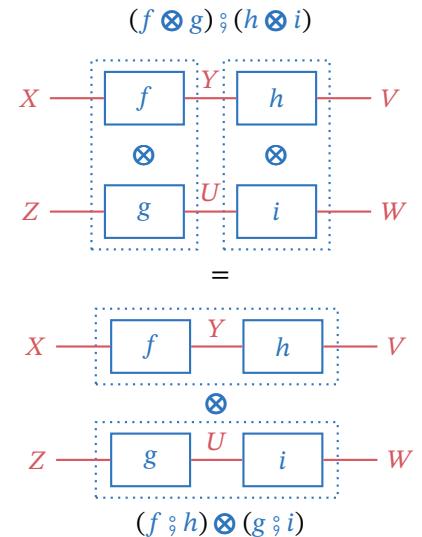


Figure 4.: Commutation of stacking and composition in a functorial stacking semicategory.

There are two types of computation: on the one hand, “functional”, or “pure”, or “free of side effects”, and on the other hand there is *effectful* computation.

In functional programming, functions in the programming language are “pure” and are very much like mathematical functions: they need an input and produce and output. And they don’t interfere with other functions.

Effectful procedures, instead, can “change the world”: for example, printing a page, sending an email, or placing an order of pizza. The order in which effectful procedures are evaluated might change the result. For example, the result of the sequence of operations

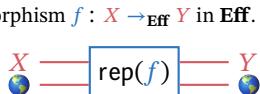
1. Order a pizza;
 2. Cancel the last order;
- is different from the result of the sequence of operations
1. Cancel the last order;
 2. Order a pizza;

A very elegant way to treat side effects mathematically is using *linear types* [31]. We will mention those in a successive part on linear logic. For now, we stick to a simple treatment.

We are going to define a category **Eff**. The idea is to add another variable that represents “the world” that can be affected. An effectful function

$$f : X \rightarrow_{\text{Eff}} Y, \quad (37)$$

(a) A morphism $f : X \rightarrow_{\text{Eff}} Y$ in **Eff**.



(b) Its representation in $\langle \text{Set} \rangle$ as a morphism $\text{rep}(f) : \langle X, \text{ } \rangle \rightarrow_{\langle \text{Set} \rangle} \langle Y, \text{ } \rangle$.

Figure 5.

which could have some unknown side effects on the world, can be represented by a pure function

$$\text{rep}(f) : X \times \text{ } \rightarrow_{\text{Set}} Y \times \text{ }, \quad (38)$$

or, in other words, as a morphism

$$\text{rep}(f) : \langle X, \text{ } \rangle \rightarrow_{\langle \text{Set} \rangle} \langle Y, \text{ } \rangle, \quad (39)$$

where is the set of all possible worlds (Fig. 5).

The second input to $\text{rep}(f)$ is the state of the world before the execution of the function. The second output of $\text{rep}(f)$ is the state of the world after the execution of the function.

We can now interconnect different effectful functions, with some precautions. We cannot “split the world”, by creating a function of type $\text{ } \rightarrow \text{ } \times \text{ }$. We will re-state this formally when we get to linear logic.

We can extend usual function composition to composition of effectful functions as in Fig. 6. The second effectful function operates on the world after it was possibly modified by the first effectful function.

We have all the ingredients to define the category **Eff** of effectful computation.

Definition 25.13 (Category of effectful procedures **Eff**)

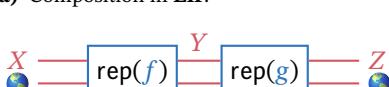
Fix a set of all possible worlds. The category **Eff** is defined by the following:

- ▷ *Objects*: same as the objects of $\langle \text{Set} \rangle$;
- ▷ *Morphisms*: a morphism $f : X \rightarrow_{\text{Eff}} Y$ is a function

$$\text{rep}(f) : \langle X, \text{ } \rangle \rightarrow_{\langle \text{Set} \rangle} \langle Y, \text{ } \rangle. \quad (40)$$

- ▷ *Composition*: The composition of $f : X \rightarrow_{\text{Eff}} Y$ and $g : Y \rightarrow_{\text{Eff}} Z$ is the

(a) Composition in **Eff**.



(b) Its representation in $\langle \text{Set} \rangle$.

Figure 6.

morphism $f \circ g : X \rightarrow_{\text{Eff}} Z$ with $\text{rep}(f \circ g)$ given by

$$\text{rep}(f \circ g) = \text{rep}(f) \circ_{\downarrow \text{Set}\Downarrow} \text{rep}(g) \quad (41)$$

as illustrated in Fig. 6.

- ▷ *Identities:* for any object X , its identity morphism $\text{id}_X : X \rightarrow_{\text{Eff}} X$ is the identity function

$$\langle X, \text{Earth} \rangle \rightarrow_{\downarrow \text{Set}\Downarrow} \langle X, \text{Earth} \rangle. \quad (42)$$

We can now make **Eff** into a stacking category by deciding how to evaluate a stack of functions. We define this in the way shown in Fig. 7.

Lemma 25.14. **Eff** is *not* a functorial stacking semicategory.

Proof. In general, we have

$$(f \otimes g) \circ (h \otimes i) \neq (f \circ h) \otimes (g \circ i).$$

This is shown graphically in Fig. 8.

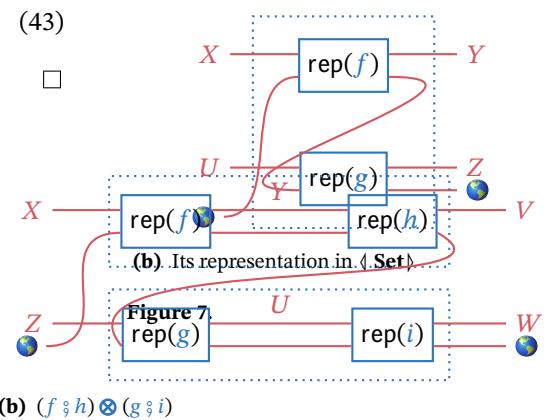
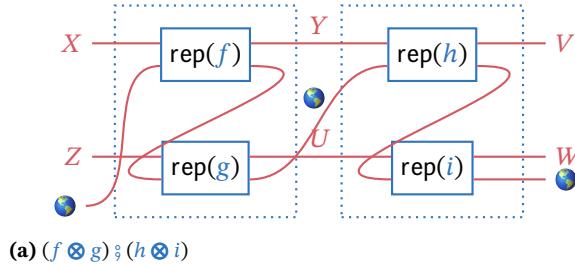


Figure 8.: Proof that **Eff** is not a functorial stacking category by showing that the two morphisms above have different representations in $\langle \downarrow \text{Set}\rangle$.

25.4. Associative stacking categories

Definition 25.15 (Strict associative stacking category)

An *strict associative stacking category* is

Constituents

1. a functorial stacking category $\langle \mathbf{C}, \otimes \rangle$;

Conditions

1. the two composite functors $((-) \otimes (-)) \otimes (-)$ and $(-) \otimes ((-) \otimes (-))$ are equal as functors $\mathbf{C} \times \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$.

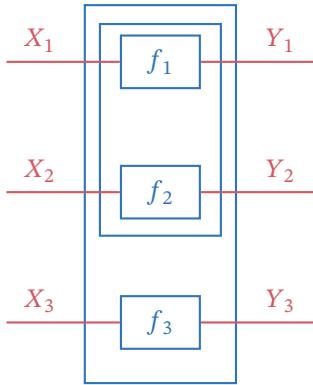


Figure 9.: Associative stacking of three morphisms, in one order.

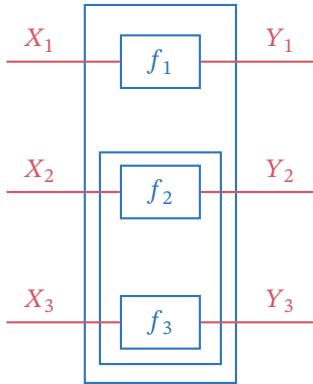


Figure 10.: Associative stacking of three morphisms, in another order.

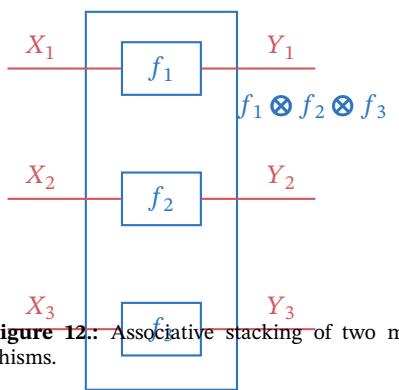


Figure 12.: Associative stacking of two morphisms.

Definition 25.16 (Associative stacking category)

An *associative stacking category* is a

Constituents

1. functorial stacking category $\langle \mathbf{C}, \otimes \rangle$;
2. a natural isomorphism

$$\text{as} : ((-) \otimes (-)) \otimes (-) \Rightarrow (-) \otimes ((-) \otimes (-)) \quad (44)$$

Conditions

1. Pentagon identities: for all $X, Y, Z, U \in \text{Ob}_{\mathbf{C}}$, the following diagram must commute

$$\begin{array}{ccccc}
 & & (X \otimes Y) \otimes (Z \otimes U) & & \\
 & \nearrow \text{as}_{X \otimes Y, Z, U} & & \searrow \text{as}_{X, Y, Z \otimes U} & \\
 ((X \otimes Y) \otimes Z) \otimes U & & & & (X \otimes (Y \otimes (Z \otimes U))) \\
 & \downarrow \text{as}_{X, Y, Z} \otimes \text{id}_U & & & \uparrow \text{id}_X \otimes \text{as}_{Y, Z, U} \\
 (X \otimes (Y \otimes Z)) \otimes U & \xrightarrow{\text{as}_{X, Y \otimes Z, U}} & X \otimes ((Y \otimes Z) \otimes U) & &
 \end{array} \quad (45)$$

If a semicategory is associative stacking, then the two ways of stacking three morphisms – as depicted in Fig. 3a and Fig. 3b, respectively – give the same result. For associative stacking semicategories we will use a simpler diagrammatic notation, where the diagrams in Fig. 3a and Fig. 3b instead look like the ones in Fig. 9 and Fig. 10. Since these two diagrams depict that same morphism, when it is convenient we will also simply depict them as in Fig. 11 (and similarly for any number of stacked morphisms). In particular, in an associative stacking semicategory, and stacking of two morphisms will be depicted as in Fig. 12.

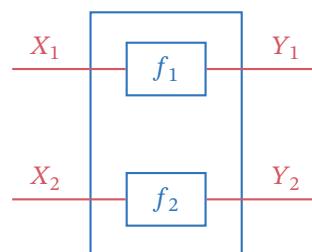


Figure 11.: Our string diagram notation for a triple stack.

Remark 25.17. In an associative stacking semicategory it follows that we can

stack any number of morphisms without needing to bracket using parentheses:

$$\frac{f_i : X_i \rightarrow Y_i, \quad 1 \leq i \leq n,}{\otimes_{i=1}^n f_i : \otimes_{i=1}^n X_i \rightarrow \otimes_{i=1}^n Y_i.} \quad (46)$$

Example 25.18. We let the integers \mathbb{Z} to be the set of objects of an associative stacking semicategory, and we say that there exists a unique morphism $X \rightarrow Y$ if and only if $X \leq Y$. (We have already seen that this forms a semicategory; in fact, a category.) As our stacking operation for objects we set

$$X \otimes Y := \max(X, Y) \quad (47)$$

and for stacking morphisms we say that if $f : X \rightarrow Y$ and $g : Z \rightarrow U$ exist, then there exists a unique morphism

$$f \otimes g : X \otimes Z \rightarrow Y \otimes U \quad (48)$$

which corresponds to (and is consistent with) the inequality

$$\max(X, Z) \leq \max(Y, U). \quad (49)$$

Example 25.19. Let \mathbf{A} be a non-empty set and consider a semicategory where the collection of objects is $\text{List } \mathbf{A}$, the set of non-empty lists of elements of \mathbf{A} . We can define a pre-order on $\text{List } \mathbf{A}$ by setting, for any lists X, Y ,

$$X \leq Y := \text{length}(X) \leq \text{length}(Y). \quad (50)$$

Then we view this pre-order as a semicategory, and define the following stacking operations. Given lists X, Y , let $X \otimes Y$ be the concatenation of X and Y , and given morphisms $f : X \rightarrow Y$ and $g : Z \rightarrow U$ representing inequalities, we let $f \otimes g : X \otimes Z \rightarrow Y \otimes U$ represent the inequality

$$\text{length}(X \otimes Z) \leq \text{length}(Y \otimes U). \quad (51)$$

One-object associative stacking semicategories

Consider a special kind of a stacking semicategory $\langle \mathbf{C}, \otimes, \otimes \rangle$ where the semicategory \mathbf{C} has only one object (call it X , say).

In this special case, we have $\otimes_{i=1}^n X_i = X$ for any $n \in \mathbb{N}$. The only hom-set is $\text{Hom}_{\mathbf{C}}(X, X)$ and this is equipped with the stacking operation

$$\otimes : \text{Hom}_{\mathbf{C}}(X, X) \times \text{Hom}_{\mathbf{C}}(X, X) \rightarrow \text{Hom}_{\mathbf{C}}(X, X) \quad (52)$$

which makes $\text{Hom}_{\mathbf{C}}(X, X)$ into a semigroup. In other words, this means that an associative stacking semicategory with one object may equivalently be described as set together with two operations – serial composition and stacking – that equip said set with two semigroup structures.

Example 25.20. The integers \mathbb{Z} , equipped with addition and multiplication as serial composition and stacking respectively, specify an associative stacking category with one object. Alternatively, we may also choose multiplication as our serial composition, and addition as our stacking.

Design problems

Example 25.21. As long as the propulsion and life support systems below do not interact, we can simply tensor the two design problems representing these systems into one, big co-design problem (Fig. 13).

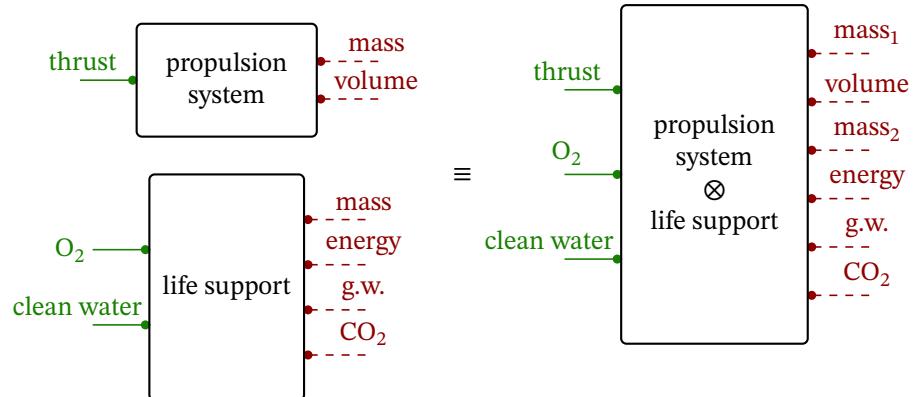


Figure 13.: Example of tensor of design problems.

25.5. Monoidal categories

Strict monoidal categories

Definition 25.22 (Strict monoidal category) A strict monoidal category is

Constituents

1. a strict associative stacking category $\langle \mathbf{C}, \otimes \rangle$,
2. an object $\mathbf{1} \in \text{Ob}_{\mathbf{C}}$, called the *monoidal unit*;

Conditions

1. for any object X of \mathbf{C} ,

$$X \otimes \mathbf{1} = X \quad \text{and} \quad \mathbf{1} \otimes X = X; \quad (53)$$

2. for any morphism $f : X \rightarrow Y$,

$$f \otimes \text{id}_{\mathbf{1}} = f \quad \text{and} \quad \text{id}_{\mathbf{1}} \otimes f = f. \quad (54)$$

Example 25.23. Consider the associative stacking category from Example 25.18, where objects are integers and stacking of objects is taking their maximum. There is no possible monoidal unit here: it would have to be a neutral element for the operation “max”, but such does not exist for \mathbb{Z} . However, we could modify this example, and replace \mathbb{Z} with a bounded set of numbers, such as \mathbb{N} . Then the smallest number in \mathbb{N} , namely 0, serves a neutral element for “max” and provides a monoidal unit $\mathbf{1}$.

Example 25.24. Consider the associative stacking category from Example 25.19. A monoidal unit would need to be a neutral element for list concatenation. This would be the empty list. In Example 25.19 we specified that objects are only non-empty lists, hence we do not have a strict monoidal semicategory. However, this example can be easily adjusted to include also the empty list, in which case we do obtain a strict monoidal semicategory.

Example 25.25. **LTI**, equipped with previously described stacking operations and an appropriate unit, is a strict monoidal stacking category. The unit is given by the object 0, and its identity morphism is given by the LTI system

$$\text{id}_{\mathbf{1}} = \langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 0} \rangle.$$

On the side of objects, clearly $l + 0 = 0 + l = l$ for any object $l \in \text{Ob}_{\text{LTI}}$. Consider $f : l \rightarrow m$. On the side of morphisms we have:

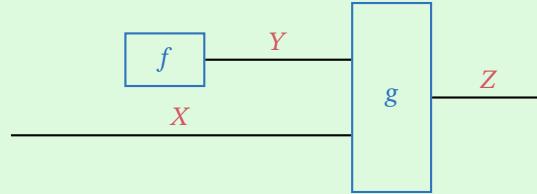
$$\begin{aligned} f \otimes \text{id}_{\mathbf{1}} &= \langle \begin{bmatrix} \text{st} \\ \mathbf{0}^{0 \times 1} \end{bmatrix}, \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}^{0 \times 0} \end{bmatrix}, \begin{bmatrix} \mathbf{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}^{0 \times 0} \end{bmatrix}, \begin{bmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}^{0 \times 0} \end{bmatrix}, \begin{bmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}^{0 \times 0} \end{bmatrix} \rangle \\ &= \langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle \end{aligned}$$

Similarly:

$$\begin{aligned} \text{id}_{\mathbf{1}} \otimes f &= \langle \begin{bmatrix} \mathbf{0}^{0 \times 1} \\ \text{st} \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{0 \times 0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{0 \times 0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{0 \times 0} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{bmatrix}, \begin{bmatrix} \mathbf{0}^{0 \times 0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D} \end{bmatrix} \rangle \\ &= \langle \text{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle \end{aligned}$$

Graded exercise H.2 (StringDiagrams)

The following is a string diagram which can represent a composition of morphisms (having certain types) in any given monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$ (We read the diagram left-to-right for series composition, and top-to-bottom for parallel composition). The resulting morphism described by the total diagram – call it h – is one of the type $h : X \rightarrow Z$.



In each part of this exercise, we will specify a monoidal category and specific objects and morphisms to plug into the variables X, Y, Z and f, g in this diagram. Your task is to compute the respective resulting morphism h as dictated by the diagram.

1. In this part, let $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$ be the monoidal category where \mathbf{C} is the category **Set** of sets and functions, \otimes is the cartesian product of sets and functions, and $\mathbf{1}$ is a chosen 1-element set that we denote by $\mathbf{1}$. In the string diagram above, let $X = \mathbb{Z}$, $Y = \mathbb{N}$, and $Z = \mathbb{Z}$. Furthermore, let

$$f : \mathbf{1} \rightarrow Y \quad (55)$$

be the function with $f(\bullet) = 5$, and let

$$\begin{aligned} g : \mathbb{N} \times \mathbb{Z} &\rightarrow \mathbb{Z}, \\ \langle y, x \rangle &\mapsto y + x. \end{aligned} \quad (56)$$

Compute the composite morphism h described by the string diagram in this case.

2. In this part, let $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$ be the monoidal category where \mathbf{C} is the category **Rel** of sets and relations, \otimes is the cartesian product of sets and relations, and $\mathbf{1}$ is a chosen 1-element set that we again denote by $\mathbf{1}$. In the string diagram, let $X = \mathbb{Z}$, $Y = \mathbb{Z}$, and $Z = \mathbb{Z}$. Furthermore, let $f : \mathbf{1} \rightarrow \mathbb{Z}$ be the relation

$$f = \{\langle \bullet, y \rangle \in \mathbf{1} \times \mathbb{Z} \mid y \text{ is an even number}\} \quad (57)$$

and let $g : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ be the relation

$$g = \{\langle \langle y, x \rangle, z \rangle \in (\mathbb{Z} \times \mathbb{Z}) \times \mathbb{Z} \mid y = x = z\}. \quad (58)$$

Compute the composite morphism h described by the string diagram in this case.

3. In this part, let $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$ be the monoidal category where \mathbf{C} is the category of real vector spaces and real linear maps, \otimes is the direct sum, and $\mathbf{1}$ is the 0-dimensional real vector space $\{0\}$. In the string diagram, let $X = Y = Z = \mathbb{R}^3$. Furthermore, let

$$f : \{0\} \rightarrow \mathbb{R}^3 \quad (59)$$

be the linear function with $f(0) = 0$, and let

$$\begin{aligned} g: \mathbb{R}^3 \oplus \mathbb{R}^3 &\rightarrow \mathbb{R}^3, \\ \langle y, x \rangle &\mapsto y + x. \end{aligned} \quad (60)$$

Compute the composite morphism h described by the string diagram in this case.

Monoidal categories

Definition 25.26 (Monoidal category)

A *monoidal category* is

Constituents

1. a strict associative stacking category $\langle \mathbf{C}, \otimes \rangle$,
2. An object $\mathbf{1} \in \text{Ob}_{\mathbf{C}}$, called the *monoidal unit*.
3. A natural isomorphism, called the *associator*, whose components are of the type

$$\text{as}_{X,Y,Z}: (X \otimes Y) \otimes Z \xrightarrow{\cong} X \otimes (Y \otimes Z) \quad X, Y, Z \in \text{Ob}_{\mathbf{C}}. \quad (61)$$

4. A natural isomorphism, called the *left unit*, whose components are of the type

$$\text{lu}_X: \mathbf{1} \otimes X \xrightarrow{\cong} X \quad X \in \text{Ob}_{\mathbf{C}}. \quad (62)$$

5. A natural isomorphism, called the *right unit*, whose components are of the type

$$\text{ru}_X: X \otimes \mathbf{1} \xrightarrow{\cong} X \quad X \in \text{Ob}_{\mathbf{C}}. \quad (63)$$

Conditions

For all $X, Y, Z, U \in \text{Ob}_{\mathbf{C}}$, the following diagrams must commute:

1. Pentagon identities.

$$\begin{array}{ccccc} & & (X \otimes Y) \otimes (Z \otimes U) & & \\ & \text{as}_{X \otimes Y, Z, U} \nearrow & & \searrow \text{as}_{X, Y, Z \otimes U} & \\ ((X \otimes Y) \otimes Z) \otimes U & & & & (X \otimes (Y \otimes (Z \otimes U))) \\ \downarrow \text{as}_{X, Y, Z} \otimes \text{id}_U & & & & \text{id}_X \otimes \text{as}_{Y, Z, U} \uparrow \\ (X \otimes (Y \otimes Z)) \otimes U & \xrightarrow{\text{as}_{X, Y \otimes Z, U}} & X \otimes ((Y \otimes Z) \otimes U) & & \end{array} \quad (64)$$

2. Triangle identities.

$$\begin{array}{ccc} (X \otimes \mathbf{1}) \otimes Y & \xrightarrow{\text{as}_{X, \mathbf{1}, Y}} & X \otimes (\mathbf{1} \otimes Y) \\ \text{ru}_X \otimes \text{id}_Y \swarrow & & \searrow \text{id}_X \otimes \text{lu}_Y \\ X \otimes Y & & \end{array} \quad (65)$$

A category equipped with a monoidal structure is called a *monoidal category*. If the components of the associator, left unit, and right unit are all equalities, one calls the category *strict monoidal*.

Remark 25.27. Note that in the constituents listed in Def. 25.26 we specified natural isomorphisms as , lu , and ru simply in terms of their components. You may be wondering: which functors are the respective source and target of these natural transformations? Since it is a mouthful to write, this information is often left to be inferred from the components given. Let us quickly illustrate how to see, from the components, which functors are involved. Take, for example, the left unit. Its components are

$$\text{lu}_X : \mathbf{1} \otimes X \xrightarrow{\cong} X \quad X \in \text{Ob}_{\mathbf{C}}, \quad (66)$$

so, if F and G denote the functors which are the source and target of lu , the functor F must act on objects by $F(X) = \mathbf{1} \otimes X$ and G must act by $G(X) = X$. The “obvious” or “canonical” choice then (given that we are considering *any* monoidal category) is that G is the identity functor and that F is the functor which acts on morphisms by mapping $f : X \rightarrow Y$ to

$$\text{id}_{\mathbf{1}} \otimes f : \mathbf{1} \otimes X \rightarrow \mathbf{1} \otimes Y. \quad (67)$$

Note that the components of the left unit lu are indexed by one variable $X \in \text{Ob}_{\mathbf{C}}$, while the associator as is indexed by *three* variables! The associator is therefore a natural transformation between two functors of the type

$$\mathbf{C} \times \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}. \quad (68)$$

Can you guess which functors of this type are meant in Def. 25.26 to be the source and target of as ?

Example 25.28. We digest the definition of monoidal category with an explanatory example. We consider the structure $\langle \text{Set}, \times, \mathbf{1} \rangle$ and show that it indeed forms a monoidal category. First, we specify how the monoidal product (cartesian product here) acts on objects and morphisms in Set (it is a functor). Given $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\text{Set}}$, $\mathbf{A} \times \mathbf{B}$ is the cartesian product of sets, and given $f : \mathbf{A} \rightarrow \mathbf{A}'$, $g : \mathbf{B} \rightarrow \mathbf{B}'$, we have:

$$\begin{aligned} (f \times g) : \mathbf{A} \times \mathbf{B} &\rightarrow \mathbf{A}' \times \mathbf{B}' \\ (a, b) &\mapsto \langle f(a), g(b) \rangle. \end{aligned} \quad (69)$$

Furthermore, given any $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \text{Ob}_{\text{Set}}$, we specify the associator $\text{as}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$:

$$\begin{aligned} \text{as}_{\mathbf{A}, \mathbf{B}, \mathbf{C}} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} &\rightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C}) \\ \langle \langle a, b \rangle, c \rangle &\mapsto \langle a, \langle b, c \rangle \rangle \end{aligned} \quad (70)$$

This defines an isomorphism (I can go “back and forth”, by switching the tuple separation). We now need to check whether as is natural. We check this graphically, using the commutative diagram in Fig. 14.

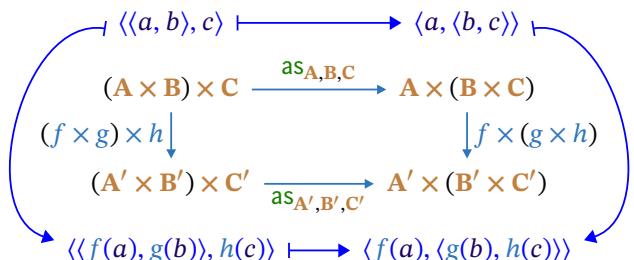


Figure 14.

Given an object $\mathbf{A} \in \text{Ob}_{\text{Set}}$, the unitor $\text{lu}_{\mathbf{A}}$ is given by:

$$\begin{aligned}\text{lu}_{\mathbf{A}} : \mathbf{1} \times \mathbf{A} &\rightarrow \mathbf{A} \\ \langle \bullet, a \rangle &\mapsto a.\end{aligned}\tag{71}$$

Again, this defines an isomorphism. Consider a morphism $f : \mathbf{A} \rightarrow \mathbf{A}'$. We now prove naturality graphically (Fig. 15).

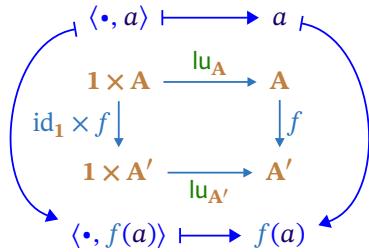


Figure 15.

Analogously, given an object $\mathbf{A} \in \text{Ob}_{\text{Set}}$, the unit isomorphism $\text{ru}_{\mathbf{A}}$ is given by:

$$\begin{aligned}\text{ru}_{\mathbf{A}} : \mathbf{A} \times \mathbf{1} &\rightarrow \mathbf{A} \\ \langle a, \bullet \rangle &\mapsto a.\end{aligned}\tag{72}$$

The proof for naturality is analogous to the one of $\text{lu}_{\mathbf{A}}$. We now need to check whether the triangle and pentagon identities are satisfied. We start by the triangle. Given $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\text{Set}}$, the proof is displayed in Fig. 16.

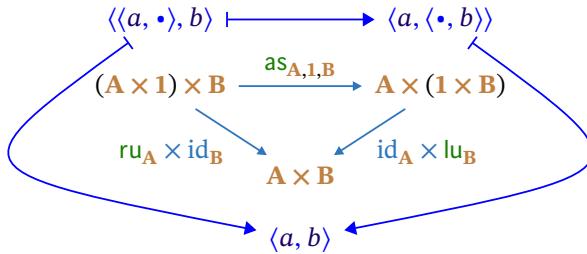


Figure 16.

We now prove the pentagon identity. Given $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \in \text{Ob}_{\text{Set}}$, the proof is reported in Fig. 17.

Example 25.29. The category $\text{Vect}_{\mathbb{R}}$ is can be equipped with a monoidal structure where the monoidal product is the tensor product of real vector spaces. It can also be equipped with a different monoidal structure where the monoidal product is the direct sum of real vector spaces.

Example 25.30 (Robot configurations). Consider \mathbb{R}^2 , discretized as a two-dimensional grid, representing locations (cells) which a robot can reach. The configuration space of the robot is $\mathbb{R}^2 \times \Theta$, where $\Theta = [0, 2\pi]$. A specific configuration $\langle x, y, \theta \rangle$ is characterized at each time by the position of the robot $x, y \in \mathbb{R}$ and its orientation $\theta \in \Theta$. The action space of the robot is $\mathcal{A} = \{\text{stay}, \leftarrow, \rightarrow, \uparrow, \downarrow\}$. This is a category, where each configuration of the robot is an object, and morphisms are robot actions which change configurations. Each configuration has the identity morphism which does not change it (stay). Composition of morphisms is given by concatenation of actions (Fig. 18). Assuming the existence of multiple robots $r_i = \langle x_i, y_i, \theta_i \rangle$, it is possible to define a product $r_i \otimes r_j$, which is to be intended as “we have a robot at configuration r_i and another one at configuration r_j ”.

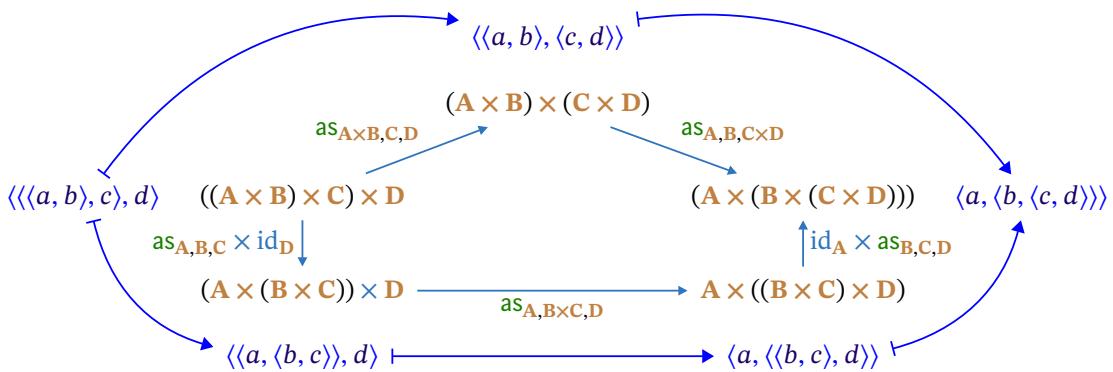


Figure 17.

However, this cannot be a proper monoidal product, because two robots cannot have the same configuration (physically, they cannot lie on each other), and hence $r_i \otimes r_i$ does not exist. By assuming that two robots could share the same configuration, this would be a valid monoidal product.

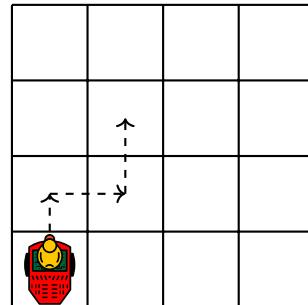


Figure 18.: Example of the robot category.

Graded exercise H.3 ($\text{VectTensorMonStructure}$)

What are straightforward choices of monoidal unit, associator, and left/right unitors which, together with the tensor product as monoidal product, equip $\text{Vect}_{\mathbb{R}}$ with a monoidal structure?

In this exercise, simply write down how you think each of these pieces of data would be defined – it is not asked that you prove that they do indeed form a monoidal structure (that would be much more involved).

Graded exercise H.4 ($\text{MonoidalProductVectDirectSum}$)

Let \mathbf{C} denote the category of real vector spaces. Given real vector spaces $\mathbf{X} = \langle \mathbf{X}, +_{\mathbf{X}}, \cdot_{\mathbf{X}} \rangle$ and $\mathbf{Y} = \langle \mathbf{Y}, +_{\mathbf{Y}}, \cdot_{\mathbf{Y}} \rangle$, their *direct sum* $\mathbf{X} \oplus \mathbf{Y}$ is defined as follows. The underlying set of $\mathbf{X} \oplus \mathbf{Y}$ is the cartesian product $\mathbf{X} \times \mathbf{Y}$ of the underlying sets of \mathbf{X} and \mathbf{Y} . For $\mathbf{X} \oplus \mathbf{Y}$, vector addition is defined (using infix notation) by

$$\langle x_1, y_1 \rangle + \langle x_2, y_2 \rangle := \langle x_1 +_{\mathbf{X}} x_2, y_1 +_{\mathbf{Y}} y_2 \rangle \quad x_1, x_2 \in \mathbf{X}; y_1, y_2 \in \mathbf{Y}; \quad (73)$$

and scalar multiplication is defined (using infix notation) by

$$\lambda \cdot \langle x, y \rangle := \langle \lambda \cdot_{\mathbf{X}} x, \lambda \cdot_{\mathbf{Y}} y \rangle, \quad \lambda \in \mathbb{R}; x \in \mathbf{X}; y \in \mathbf{Y}. \quad (74)$$

There is a monoidal structure $\langle \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru} \rangle$ on \mathbf{C} where the monoidal

product \otimes , on objects, is the direct sum of real vector spaces. Your tasks:

1. Guess what the definition of \otimes on morphisms is, and check that \otimes really does define a functor.
2. Guess what the definitions of the monoidal unit, associator, and the unitors are for this monoidal product. Are the components of the associator and unitors really isomorphisms in \mathbf{C} ? Justify, briefly, why.
3. Check that the left-unit is indeed a natural transformation. What are the functors that it maps between?
4. Check the coherence condition in the definition of a monoidal category that involves the unitors.

25.6. Monoidal functors

Definition 25.31 (Strong monoidal functor)

Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}} \rangle$ and $\langle \mathbf{D}, \otimes_{\mathbf{D}}, \mathbf{1}_{\mathbf{D}} \rangle$ be monoidal categories.

A *strong monoidal functor* between \mathbf{C} and \mathbf{D} is

Constituents

1. A functor

$$\mathcal{F} : \mathbf{C} \rightarrow \mathbf{D}; \quad (75)$$

2. A natural isomorphism μ

$$\mu_{X,Y} : \mathcal{F}(X) \otimes_{\mathbf{D}} \mathcal{F}(Y) \rightarrow \mathcal{F}(X \otimes_{\mathbf{C}} Y), \quad \forall X, Y \in \mathbf{C}, \quad (76)$$

3. An isomorphism

$$u : \mathbf{1}_{\mathbf{D}} \rightarrow \mathcal{F}(\mathbf{1}_{\mathbf{C}}); \quad (77)$$

Conditions

1. *Associativity*: For all objects $X, Y, Z \in \mathbf{C}$, there are *associators* $\text{as}^{\mathbf{C}}$ and $\text{as}^{\mathbf{D}}$ such that the diagram in Fig. 19a commutes.
2. *Unitality*: For all $X \in \mathbf{C}$, there exist left and right *unitors* $\text{lu}^{\mathbf{C}}$ and $\text{ru}^{\mathbf{C}}$, the diagram in Fig. 19b commutes.

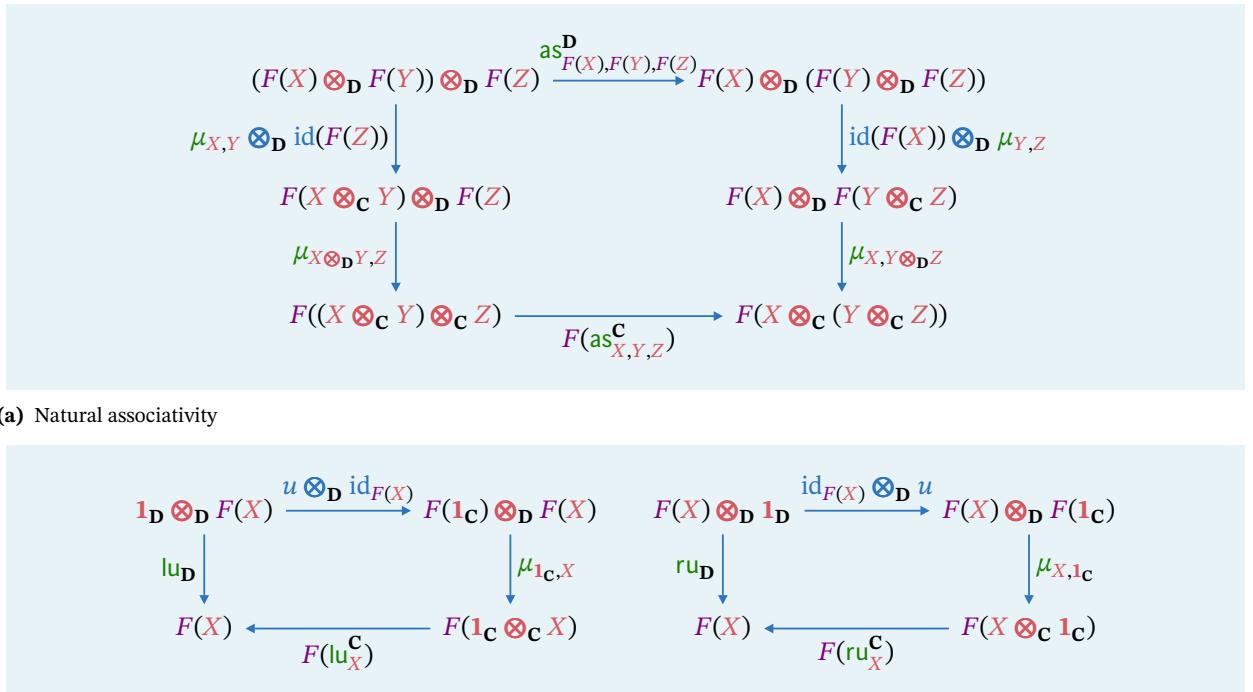


Figure 19.: Commuting diagrams using in Def. 25.31

25.7. Strictification

Lemma 25.32. $\langle \text{Set} \rangle$ is associative stacking using the structure that arises from tuple concatenation.

Proof. For the stacking operation on objects \otimes we use the operation $\circ\backslash$ defined in Section 18.2 which was referred to as the “multiplication in $\langle \text{Set} \rangle$ ”:

$$\langle A_1, \dots, A_m \rangle \otimes \langle B_1, \dots, B_n \rangle := \langle A_1, \dots, A_m \rangle \circ\backslash \langle B_1, \dots, B_n \rangle \quad (78)$$

$$= \langle A_1, \dots, A_m, B_1, \dots, B_n \rangle. \quad (79)$$

It was shown there that this operation is associative.

As for \otimes , we define it as follows:

$$\frac{f : A \rightarrow \langle \text{Set} \rangle B \quad g : C \rightarrow \langle \text{Set} \rangle D}{(f \otimes g) : A \circ\backslash C \rightarrow \langle \text{Set} \rangle B \circ\backslash D}.$$

$$x \circ\backslash z \mapsto f(x) \circ\backslash g(z) \quad (80)$$

The two operations \otimes, \otimes so defined satisfy the compatibility conditions required by Def. 25.41.

To show associativity, consider three morphisms

$$f : A \rightarrow \langle \text{Set} \rangle B, \quad g : C \rightarrow \langle \text{Set} \rangle D, \quad h : E \rightarrow \langle \text{Set} \rangle F. \quad (81)$$

We compute $f \otimes (g \otimes h)$ and $(f \otimes g) \otimes h$ following the recipe (80) to obtain

$$(f \otimes g) \otimes h : (A \circ\backslash C) \circ\backslash E \rightarrow \langle \text{Set} \rangle (B \circ\backslash D) \circ\backslash F,$$

$$(x \circ\backslash z) \circ\backslash v \mapsto (f(x) \circ\backslash g(z)) \circ\backslash h(v), \quad (82)$$

$$f \otimes (g \otimes h) : A \circ\backslash (C \circ\backslash E) \rightarrow \langle \text{Set} \rangle B \circ\backslash (D \circ\backslash F),$$

$$x \circ\backslash (z \circ\backslash v) \mapsto f(x) \circ\backslash (g(z) \circ\backslash h(v)). \quad (83)$$

Notice that the operations $\circ\backslash$ and $\circ\backslash$ are associative; therefore, we can remove all the light parentheses that appear in the formulas. This implies that both functions are equal to

$$f \otimes g \otimes h : A \circ\backslash C \circ\backslash E \rightarrow \langle \text{Set} \rangle B \circ\backslash D \circ\backslash F,$$

$$x \circ\backslash z \circ\backslash v \mapsto f(x) \circ\backslash g(z) \circ\backslash h(v). \quad (84)$$

□

The category $\langle \text{Pos} \rangle$

We define a category analogous to $\langle \text{Set} \rangle$, but its objects are “tuple posets”.

Given posets P_1, \dots, P_n we define the poset

$$\langle P_1, \dots, P_n \rangle := \langle \langle P_1, \dots, P_n \rangle, \leq_{\langle P_1, \dots, P_n \rangle} \rangle, \quad (85)$$

where $\langle P_1, \dots, P_n \rangle$ is a set of tuples, and we use the product order:

$$\frac{\langle x_1, \dots, x_n \rangle \leq_{\langle P_1, \dots, P_n \rangle} \langle y_1, \dots, y_n \rangle}{\overline{x_i \leq_{P_i} y_i \text{ for all } i \in \{1, \dots, n\}}}. \quad (86)$$

Definition 25.33 (The category $\langle \mathbf{Pos} \rangle$)

The category $\langle \mathbf{Pos} \rangle$ is the subcategory of \mathbf{Pos} where the objects are tuple posets, posets of the form

$$\langle P_1, \dots, P_n \rangle = \langle \langle P_1, \dots, P_n \rangle, \leq_{\langle P_1, \dots, P_n \rangle} \rangle. \quad (87)$$

Lemma 25.34. $\langle \mathbf{Pos} \rangle$ is associative stacking using the structure induced by tuple concatenation.

Proof. Analogously to what we did for $\langle \mathbf{Set} \rangle$, we can define a multiplication operation \otimes in $\langle \mathbf{Pos} \rangle$.

Given two objects $P = \langle P_1, \dots, P_n \rangle$ and $Q = \langle Q_1, \dots, Q_n \rangle$, we define

$$\langle P_1, \dots, P_n \rangle \otimes \langle Q_1, \dots, Q_n \rangle := \langle P_1, \dots, P_n, Q_1, \dots, Q_n \rangle. \quad (88)$$

For the multiplication on morphisms, we define

$$\frac{\begin{array}{c} f : P \rightarrow Q \\ f : R \rightarrow S \end{array}}{(f \otimes g) : P \otimes R \rightarrow Q \otimes S} \quad (89)$$

$$p \circledast r \mapsto f(p) \circledast g(r)$$

We need to check that the expression $f(p) \circledast g(r)$ is monotone. This can be easily seen because the order on $Q \otimes S$ is akin to a product order. The proof for associativity of \otimes is the same as in the proof of $\langle \mathbf{Set} \rangle$ (Lemma 25.32). \square

The category $\langle \mathbf{Rel} \rangle$

We define a category $\langle \mathbf{Rel} \rangle$ where the objects are sets of tuples (as in $\langle \mathbf{Set} \rangle$).

Definition 25.35 (The category $\langle \mathbf{Rel} \rangle$)

The category $\langle \mathbf{Rel} \rangle$ is the subcategory of \mathbf{Rel} where the objects are tuples sets (objects of $\langle \mathbf{Set} \rangle$).

Lemma 25.36. $\langle \mathbf{Rel} \rangle$ is associative stacking with the structure induced by tuple concatenation.

Proof. The multiplication \otimes is \circledast , the same as the one defined for $\langle \mathbf{Set} \rangle$.

The multiplication \otimes is defined as follows:

$$\frac{\begin{array}{c} R \subseteq A \times B \\ S \subseteq C \times D \end{array}}{(R \otimes S) \subseteq (A \circledast C) \times (B \circledast D)} \quad (90)$$

where

$$(R \otimes S) = \{ \langle a \circledast c, b \circledast d \rangle \mid (a R b) \wedge (c S d) \}. \quad (91)$$

The rest of the proof is left as an exercise. \square

Exercise 50. Consider the stacking operations on objects and on morphisms introduced in this section. Prove that $\langle \mathbf{Rel} \rangle$ is associative stacking.

See solution on page 401.

subsectionThe category $\langle \mathbf{DP} \rangle$

Analogously, we define the category $\langle \mathbf{DP} \rangle$.

Definition 25.37 (The category $\langle \mathbf{DP} \rangle$)

The category $\langle \mathbf{DP} \rangle$ is the subcategory of \mathbf{DP} where the objects are posets of tuple posets (objects of $\langle \mathbf{Pos} \rangle$).

Lemma 25.38. $\langle \mathbf{DP} \rangle$ is associative stacking using the structure induced by tuple concatenation.

Proof. For the stacking operation \otimes on objects, we use $\otimes_{\langle \mathbf{DP} \rangle} := \otimes_{\langle \mathbf{Pos} \rangle}$. For stacking morphisms, we define \otimes by

$$\begin{array}{c} \mathbf{d}: P^{\text{op}} \times R \rightarrow \mathbf{Bool} \quad \mathbf{e}: Q^{\text{op}} \times S \rightarrow \mathbf{Bool}, \\ \hline \mathbf{d} \otimes \mathbf{e}: (P \otimes Q)^{\text{op}} \times (R \otimes S) \rightarrow \mathbf{Bool} \\ \langle a^* \between c^*, b \between d \rangle \mapsto \mathbf{d}(a^*, c) \wedge \mathbf{e}(c^*, d) \end{array} \quad (92)$$

Note that this is a valid definition of a design problem because the expression $\mathbf{d}(a^*, c) \wedge \mathbf{e}(c^*, d)$ is monotone, as may readily be checked.

To show associativity, consider three DPs

$$\mathbf{d}: P^{\text{op}} \times R \rightarrow_{\mathbf{Pos}} \mathbf{Bool}, \quad \mathbf{e}: Q^{\text{op}} \times S \rightarrow_{\mathbf{Pos}} \mathbf{Bool}, \quad \mathbf{g}: T^{\text{op}} \times U \rightarrow_{\mathbf{Pos}} \mathbf{Bool}. \quad (93)$$

We compute $(\mathbf{d} \otimes \mathbf{e}) \otimes \mathbf{g}$ and $\mathbf{d} \otimes (\mathbf{e} \otimes \mathbf{g})$ according to the recipe in (92):

$$\begin{array}{c} (\mathbf{d} \otimes \mathbf{e}) \otimes \mathbf{g}: ((P \otimes Q) \otimes T)^{\text{op}} \times ((R \otimes S) \otimes U) \rightarrow_{\mathbf{Pos}} \mathbf{Bool}, \\ \langle \langle a^* \between c^* \rangle \between e^*, \langle b \between d \rangle \between f \rangle \mapsto (\mathbf{d}(a^*, c) \wedge \mathbf{e}(c^*, d)) \wedge \mathbf{g}(e^*, f), \end{array} \quad (94)$$

$$\begin{array}{c} \mathbf{d} \otimes (\mathbf{e} \otimes \mathbf{g}): (P \otimes (Q \otimes T))^{\text{op}} \times (R \otimes (S \otimes U)) \rightarrow_{\mathbf{Pos}} \mathbf{Bool}, \\ \langle a^* \between (c^* \between e^*), b \between (d \between f) \rangle \mapsto \mathbf{d}(a^*, c) \wedge (\mathbf{e}(c^*, d) \wedge \mathbf{g}(e^*, f)). \end{array} \quad (95)$$

Because the operations \otimes and \wedge are associative, we can erase all the light parentheses in the formulas, and we find that $(\mathbf{d} \otimes \mathbf{e}) \otimes \mathbf{g}$ and $\mathbf{d} \otimes (\mathbf{e} \otimes \mathbf{g})$ are equal to the design problem

$$\begin{array}{c} \mathbf{d} \otimes \mathbf{e} \otimes \mathbf{g}: (P \otimes Q \otimes T)^{\text{op}} \times (R \otimes S \otimes U) \rightarrow_{\mathbf{Pos}} \mathbf{Bool}, \\ \langle a^* \between c^* \between e^*, b \between d \between f \rangle \mapsto \mathbf{d}(a^*, c) \wedge \mathbf{e}(c^*, d) \wedge \mathbf{g}(e^*, f). \end{array} \quad (96)$$

□

Example 25.39 ($\langle \mathbf{Set} \rangle$ is a functorial stacking semicategory). We want to show that $\langle \mathbf{Set} \rangle$ is a functorial stacking semicategory.

Consider four morphisms

$$\begin{array}{l} f: A \rightarrow B, \quad h: B \rightarrow C, \\ g: D \rightarrow E, \quad i: E \rightarrow F. \end{array} \quad (97)$$

We want to show that

$$(f \otimes g) ; (h \otimes i) = (f ; h) \otimes (g ; i), \quad (98)$$

We show this by showing that, for any $a \between d \in A \between D$:

$$\begin{aligned} ((f \otimes g) ; (h \otimes i))(a \between d) &= (h \otimes i)(f(a) \between g(d)) \\ &= h(f(a)) \between i(g(d)) \\ &= (f ; h)(a) \between (g ; i)(d) \\ &= ((f ; h) \otimes (g ; i))(a \between d). \end{aligned} \quad (99)$$

Exercise 51. Prove that $\langle \mathbf{Pos} \rangle$ is a functorial stacking category. Hint: You have to define the functor \otimes and check that it satisfies the two equations in Def. 25.7.
See solution on page 401.

Graded exercise H.5 (`RelFunStack`)

Prove that the structure defined in Exercise 50 makes $\langle \mathbf{Rel} \rangle$ a functorial stacking semicategory.

Lemma 25.40 ($\langle \mathbf{DP} \rangle$ is a functorial stacking semicategory). $\langle \mathbf{DP} \rangle$, equipped with the aforementioned stacking operations on objects and morphisms, is functorial stacking semicategory.

Proof. Consider

$$\begin{aligned} \mathbf{d} &: \mathbf{P}^{\text{op}} \times \mathbf{R} \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \mathbf{e} &: \mathbf{Q}^{\text{op}} \times \mathbf{S} \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \mathbf{g} &: \mathbf{R}^{\text{op}} \times \mathbf{T} \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \mathbf{h} &: \mathbf{S}^{\text{op}} \times \mathbf{U} \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \end{aligned} \quad (100)$$

We want to prove that

$$(\mathbf{d} \circ \mathbf{g}) \otimes (\mathbf{e} \circ \mathbf{h}) = (\mathbf{d} \otimes \mathbf{e}) \circ (\mathbf{g} \otimes \mathbf{h}). \quad (101)$$

We start from the left-hand side. We have

$$(\mathbf{d} \circ \mathbf{g})(p^*, t) = \bigvee_{r \in \mathbf{R}} \mathbf{d}(p^*, r) \wedge \mathbf{g}(r^*, t) \quad (102)$$

and

$$(\mathbf{e} \circ \mathbf{h})(q^*, u) = \bigvee_{s \in \mathbf{S}} \mathbf{e}(q^*, s) \wedge \mathbf{h}(s^*, u) \quad (103)$$

Therefore, we know

$$\begin{aligned} &((\mathbf{d} \circ \mathbf{g}) \otimes (\mathbf{e} \circ \mathbf{h}))((p \circ q)^*, t \circ u) \\ &= \bigvee_{r \in \mathbf{R}} \mathbf{d}(p^*, r) \wedge \mathbf{g}(r^*, t) \wedge \bigvee_{s \in \mathbf{S}} \mathbf{e}(q^*, s) \wedge \mathbf{h}(s^*, u). \end{aligned} \quad (104)$$

On the other hand, we have

$$(\mathbf{d} \otimes \mathbf{e})((p \circ q)^*, r \circ s) = \mathbf{d}(p^*, r) \wedge \mathbf{e}(q^*, s) \quad (105)$$

and

$$(\mathbf{g} \otimes \mathbf{h})((r \circ s)^*, t \circ u) = \mathbf{g}(r^*, t) \wedge \mathbf{h}(s^*, u) \quad (106)$$

Therefore, we know

$$\begin{aligned} &((\mathbf{d} \otimes \mathbf{e}) \circ (\mathbf{g} \otimes \mathbf{h}))((p \circ q)^*, t \circ u) \\ &= \bigvee_{r \circ s \in \{\mathbf{R}, \mathbf{S}\}} (\mathbf{d} \otimes \mathbf{e})((p \circ q)^*, r \circ s) \wedge (\mathbf{g} \otimes \mathbf{h})((r \circ s)^*, t \circ u) \\ &= \bigvee_{r \circ s \in \{\mathbf{R}, \mathbf{S}\}} \mathbf{d}(p^*, r) \wedge \mathbf{e}(q^*, s) \wedge \mathbf{g}(r^*, t) \wedge \mathbf{h}(s^*, u) \\ &= \bigvee_{r \in \mathbf{R}} \mathbf{d}(p^*, r) \wedge \mathbf{g}(r^*, t) \wedge \bigvee_{s \in \mathbf{S}} \mathbf{e}(q^*, s) \wedge \mathbf{h}(s^*, u), \end{aligned} \quad (107)$$

proving the statement for any posets $\mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{S}, \mathbf{T}, \mathbf{U}$ (and hence, also for posets of tuples).



25.8. The case of semicategories

Stacking

Definition 25.41 (Stacking semicategory)

A *stacking semicategory* is a semicategory \mathbf{C} with the following additional constituents and properties.

Constituents

- ▷ A stacking operation $\otimes : \text{Ob}_{\mathbf{C}} \times \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{C}}$.
- ▷ A stacking operation $\boxtimes : \text{Mor}_{\mathbf{C}} \times \text{Mor}_{\mathbf{C}} \rightarrow \text{Mor}_{\mathbf{C}}$.

Conditions

- ▷ The two operations \otimes and \boxtimes are compatible in the sense that

$$\frac{f_1 : X_1 \rightarrow Y_1 \quad f_2 : X_2 \rightarrow Y_2}{f_1 \otimes f_2 : X_1 \otimes X_2 \rightarrow Y_1 \otimes Y_2}. \quad (108)$$

Functorial stacking

Definition 25.42 (Functorial stacking semicategory)

A *functorial stacking semicategory* is a stacking semicategory where the two stacking operations \otimes and \boxtimes are the two components of a functor

$$\otimes : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}. \quad (109)$$

In infix notation, this means that

$$(f ; h) \otimes (g ; i) = (f \otimes g) ; (h \otimes i) \quad (110)$$

for all morphisms f, g, h , and i (where respectively f and h , and g and i are composable), and that

$$\text{id}_X \otimes \text{id}_Y = \text{id}_{X \otimes Y} \quad (111)$$

for all objects X, Y of \mathbf{C} , whenever all three of these identity morphisms exist.

Example 25.43. We want to show that **Moo**, equipped with the defined stacking operations, is almost a functorial stacking semicategory, but not quite. Consider four Moore machines:

$$f : \mathbf{U}_f \rightarrow \mathbf{Y}_f, \quad g : \mathbf{U}_g \rightarrow \mathbf{Y}_g, \quad h : \mathbf{Y}_f \rightarrow \mathbf{Y}_h, \quad i : \mathbf{Y}_g \rightarrow \mathbf{Y}_i.$$

We want to check if the equation

$$(f ; h) \otimes (g ; i) = (f \otimes g) ; (h \otimes i)$$

holds. Let's start from the left-hand side. First, we have:

$$\begin{aligned} f ; h &= \langle \mathbf{U}_f, \mathbf{X}_f \circ_{\mathbf{f}} \mathbf{X}_h, \mathbf{Y}_h, \text{dyn}_{f ; h}, \text{ro}_{f ; h}, \text{st}_f \circ_{\mathbf{f}} \text{st}_h \rangle, \\ g ; i &= \langle \mathbf{U}_g, \mathbf{X}_g \circ_{\mathbf{g}} \mathbf{X}_i, \mathbf{Y}_i, \text{dyn}_{g ; i}, \text{ro}_{g ; i}, \text{st}_g \circ_{\mathbf{g}} \text{st}_i \rangle, \end{aligned}$$

with

$$\text{dyn}_{f ; h} : \mathbf{U}_f \circ_{\mathbf{f}} \mathbf{X}_f \circ_{\mathbf{f}} \mathbf{X}_h \rightarrow \mathbf{X}_f \circ_{\mathbf{f}} \mathbf{X}_h,$$

$$u_f \circ_{\mathbf{f}} x_f \circ_{\mathbf{f}} x_h \mapsto \text{dyn}_f(u_f \circ_{\mathbf{f}} x_f) \circ_{\mathbf{f}} \text{dyn}_h(\text{ro}_f(x_f) \circ_{\mathbf{f}} x_h),$$

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

$$\begin{aligned}\text{dyn}_{g \circ i} : \mathbf{U}_g \circ \mathbf{X}_g \circ \mathbf{X}_i &\rightarrow \mathbf{X}_g \circ \mathbf{X}_i, \\ u_g \circ x_g \circ x_i &\mapsto \text{dyn}_g(u_g \circ x_g) \circ \text{dyn}_i(\text{ro}_g(x_g) \circ x_i), \\ \text{ro}_{f \circ h} : \mathbf{X}_f \circ \mathbf{X}_h &\rightarrow \mathbf{Y}_h, \\ x_f \circ x_h &\mapsto \text{ro}_h(x_h),\end{aligned}$$

and

$$\begin{aligned}\text{ro}_{g \circ i} : \mathbf{X}_g \circ \mathbf{X}_i &\rightarrow \mathbf{Y}_i, \\ x_g \circ x_i &\mapsto \text{ro}_i(x_i).\end{aligned}$$

Furthermore:

$$(f \circ h) \otimes (g \circ i) = \langle \mathbf{U}_f \circ \mathbf{U}_g, \mathbf{X}_f \circ \mathbf{X}_h \circ \mathbf{X}_g \circ \mathbf{X}_i, \mathbf{Y}_h \circ \mathbf{Y}_i, \text{dyn}_{(f \circ h) \otimes (g \circ i)}, \text{ro}_{(f \circ h) \otimes (g \circ i)}, \text{st}_f \circ \text{st}_h \circ \text{st}_g \circ \text{st}_i \rangle,$$

with

$$\begin{aligned}\text{dyn}_{(f \circ h) \otimes (g \circ i)} : \mathbf{U}_f \circ \mathbf{U}_g \circ \mathbf{X}_f \circ \mathbf{X}_h \circ \mathbf{X}_g \circ \mathbf{X}_i &\rightarrow \mathbf{X}_h \circ \mathbf{X}_g \circ \mathbf{X}_i, \\ u_f \circ u_g \circ x_f \circ x_h \circ x_g \circ x_i &\mapsto \underbrace{\text{dyn}_{f \circ h}(u_f \circ x_f \circ x_h) \circ \text{dyn}_{g \circ i}(u_g \circ x_g \circ x_i)}_{(1)}\end{aligned}$$

where

$$(1) = \text{dyn}_f(u_f \circ x_f) \circ \text{dyn}_h(\text{ro}_f(x_f) \circ x_h) \circ \text{dyn}_g(u_g \circ x_g) \circ \text{dyn}_i(\text{ro}_g(x_g) \circ x_i),$$

and

$$\begin{aligned}\text{ro}_{(f \circ h) \otimes (g \circ i)} : \mathbf{X}_f \circ \mathbf{X}_h \circ \mathbf{X}_g \circ \mathbf{X}_i &\rightarrow \mathbf{Y}_h \circ \mathbf{Y}_i, \\ x_f \circ x_h \circ x_g \circ x_i &\mapsto \underbrace{\text{ro}_{f \circ h}(x_f \circ x_h) \circ \text{ro}_{g \circ i}(x_g \circ x_i)}_{(2)}\end{aligned}$$

where

$$(2) = \text{ro}_h(x_h) \circ \text{ro}_i(x_i).$$

On the other hand, we have:

$$\begin{aligned}f \otimes g &= \langle \mathbf{U}_f \circ \mathbf{U}_g, \mathbf{X}_f \circ \mathbf{X}_g, \mathbf{Y}_f \circ \mathbf{Y}_g, \text{dyn}_{f \otimes g}, \text{ro}_{f \otimes g}, \text{st}_f \circ \text{st}_g \rangle, \\ h \otimes i &= \langle \mathbf{U}_h \circ \mathbf{U}_i, \mathbf{X}_h \circ \mathbf{X}_i, \mathbf{Y}_h \circ \mathbf{Y}_i, \text{dyn}_{h \otimes i}, \text{ro}_{h \otimes i}, \text{st}_h \circ \text{st}_i \rangle,\end{aligned}$$

with

$$\begin{aligned}\text{dyn}_{f \otimes g} : \mathbf{U}_f \circ \mathbf{U}_g \circ \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \mathbf{X}_f \circ \mathbf{X}_g, \\ u_f \circ u_g \circ x_f \circ x_g &\mapsto \text{dyn}_f(u_f \circ x_f) \circ \text{dyn}_g(u_g \circ x_g), \\ \text{dyn}_{h \otimes i} : \mathbf{U}_h \circ \mathbf{U}_i \circ \mathbf{X}_h \circ \mathbf{X}_i &\rightarrow \mathbf{X}_h \circ \mathbf{X}_i, \\ u_h \circ u_i \circ x_h \circ x_i &\mapsto \text{dyn}_h(u_h \circ x_h) \circ \text{dyn}_i(u_i \circ x_i), \\ \text{ro}_{f \otimes g} : \mathbf{X}_f \circ \mathbf{X}_g &\rightarrow \mathbf{Y}_f \circ \mathbf{Y}_g, \\ x_f \circ x_g &\mapsto \text{ro}_f(x_f) \circ \text{ro}_g(x_g),\end{aligned}$$

and

$$\begin{aligned}\text{ro}_{h \otimes i} : \mathbf{X}_h \circ \mathbf{X}_i &\rightarrow \mathbf{Y}_h \circ \mathbf{Y}_i, \\ x_h \circ x_i &\mapsto \text{ro}_h(x_h) \circ \text{ro}_i(x_i).\end{aligned}$$

Furthermore:

$\mathbf{U}_g, \mathbf{X}_f \circ_{\mathbb{A}} \mathbf{X}_g \circ_{\mathbb{A}} \mathbf{X}_h \circ_{\mathbb{A}} \mathbf{X}_i, \mathbf{Y}_h \circ_{\mathbb{A}} \mathbf{Y}_i, \text{dyn}_{(f \otimes g) \otimes (h \otimes i)}, \text{ro}_{(f \otimes g) \circ (h \otimes i)}, \text{st}_f \circ_{\mathbb{A}} \text{st}_g \circ_{\mathbb{A}} \text{st}_h \circ_{\mathbb{A}} \text{st}_i,$

with

$$\begin{aligned} \text{dyn}_{(f \otimes g) \circ (h \otimes i)} : \mathbf{U}_f \circ_{\mathbb{A}} \mathbf{U}_g \circ_{\mathbb{A}} \mathbf{X}_f \circ_{\mathbb{A}} \mathbf{X}_g \circ_{\mathbb{A}} \mathbf{X}_h \circ_{\mathbb{A}} \mathbf{X}_i &\rightarrow \mathbf{X}_f \circ_{\mathbb{A}} \mathbf{X}_g \circ_{\mathbb{A}} \mathbf{X}_h \circ_{\mathbb{A}} \mathbf{X}_i, \\ u_f \circ_{\mathbb{A}} u_g \circ_{\mathbb{A}} x_f \circ_{\mathbb{A}} x_g \circ_{\mathbb{A}} x_h \circ_{\mathbb{A}} x_i &\mapsto \underbrace{\text{dyn}_{f \otimes g}(u_f \circ_{\mathbb{A}} u_g \circ_{\mathbb{A}} x_f \circ_{\mathbb{A}} x_g) \circ_{\mathbb{A}} \text{dyn}_{h \otimes i}(\text{ro}_{f \otimes g}(x_f \circ_{\mathbb{A}} x_g) \circ_{\mathbb{A}} x_h \circ_{\mathbb{A}} x_i)}_{(3)}, \end{aligned}$$

with

$$\begin{aligned} (3) &= \text{dyn}_f(u_f \circ_{\mathbb{A}} x_f) \circ_{\mathbb{A}} \text{dyn}_g(u_g \circ_{\mathbb{A}} x_g) \circ_{\mathbb{A}} \text{dyn}_{h \otimes i}(\text{ro}_f(x_f) \circ_{\mathbb{A}} \text{ro}_g(x_g) \circ_{\mathbb{A}} x_h \circ_{\mathbb{A}} x_i) \\ &= \text{dyn}_f(u_f \circ_{\mathbb{A}} x_f) \circ_{\mathbb{A}} \underbrace{\text{dyn}_g(u_g \circ_{\mathbb{A}} x_g) \circ_{\mathbb{A}} \text{dyn}_h(\text{ro}_f(x_f) \circ_{\mathbb{A}} x_h)}_{(*)} \circ_{\mathbb{A}} \underbrace{\text{dyn}_i(\text{ro}_g(x_g) \circ_{\mathbb{A}} x_i)}_{(**)} \end{aligned}$$

and

$$\begin{aligned} \text{ro}_{(f \otimes g) \circ (h \otimes i)} : \mathbf{X}_f \circ_{\mathbb{A}} \mathbf{X}_g \circ_{\mathbb{A}} \mathbf{X}_h \circ_{\mathbb{A}} \mathbf{X}_i &\rightarrow \mathbf{Y}_h \circ_{\mathbb{A}} \mathbf{Y}_i, \\ x_f \circ_{\mathbb{A}} x_g \circ_{\mathbb{A}} x_h \circ_{\mathbb{A}} x_i &\mapsto \text{ro}_{h \otimes i}(x_h \circ_{\mathbb{A}} x_i) = \text{ro}_h(x_h) \circ_{\mathbb{A}} \text{ro}_i(x_i), \end{aligned}$$

As one can see from the expression for (3), the two terms (*) and (**) are switched compared to (1). Apart from this switch (and the corresponding switch in the signatures of the dynamics maps), we can see that there is a “moral correspondence” between the Moore machines in the functorial stacking axiom.

Associative stacking

Definition 25.44 (Strict associative stacking semicategory)
An *strict associative stacking category* is

Constituents

1. a functorial stacking semicategory $\langle \mathbf{C}, \otimes \rangle$;

Conditions

1. the two composite functors $((-) \otimes (-)) \otimes (-)$ and $(-) \otimes ((-) \otimes (-))$ are equal as functors $\mathbf{C} \times \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$.

Moo is associative stacking

When considering Moore machines, we can define stacking operations and show that **Moo** forms a stacking semicategory (Def. 25.41). The objects of **Moo** are objects of $\downarrow \mathbf{Set} \uparrow$, and therefore the stacking operation for objects corresponds to the “multiplication in $\downarrow \mathbf{Set} \uparrow$ ”, denoted by $\circ_{\mathbb{A}}$.

The operation on morphisms “stacks” Moore machines onto each other. Formally:

$$\frac{f : \mathbf{U}_f \rightarrow_{\mathbf{Moo}} \mathbf{Y}_f \quad f : \mathbf{U}_g \rightarrow_{\mathbf{Moo}} \mathbf{Y}_g}{f \otimes g = \langle \mathbf{U}_f \circ_{\mathbb{A}} \mathbf{U}_g, \mathbf{X}_f \circ_{\mathbb{A}} \mathbf{X}_g, \mathbf{Y}_f \circ_{\mathbb{A}} \mathbf{Y}_g, \text{dyn}_{f \otimes g}, \text{ro}_{f \otimes g}, \text{st}_f \circ_{\mathbb{A}} \text{st}_g \rangle} \quad (112)$$

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

with

$$\begin{aligned}\text{dyn}_{f \otimes g} : \mathbf{U}_f \circledast \mathbf{U}_g \circledast \mathbf{X}_f \circledast \mathbf{X}_g &\rightarrow \mathbf{X}_f \circledast \mathbf{X}_g, \\ u_f \circledast u_g \circledast x_f \circledast x_g &\mapsto \text{dyn}_f(u_f \circledast x_f) \circledast \text{dyn}_g(u_g \circledast x_g),\end{aligned}$$

and

$$\begin{aligned}\text{ro}_{f \otimes g} : \mathbf{X}_f \circledast \mathbf{X}_g &\rightarrow \mathbf{Y}_f \circledast \mathbf{Y}_g, \\ x_f \circledast x_g &\mapsto \text{ro}(x_f) \circledast \text{ro}(x_g).\end{aligned}$$

While we have already proved that the operation \circledast is associative, it is also easy to see that the stacking of Moore machines is associative. Therefore, **Moo** equipped with the described stacking operations forms an associative stacking semicategory.

Monoidal stacking

Definition 25.45 (Strict monoidal stacking semicategory)

A strict monoidal stacking semicategory is a stacking semicategory $\langle \mathbf{C}, \otimes, \otimes \rangle$ with

Constituents

1. an object $\mathbf{1} \in \text{Ob}_{\mathbf{C}}$, called the *monoidal unit*

Conditions

1. For any object X of \mathbf{C} ,

$$X \otimes \mathbf{1} = X \quad \text{and} \quad \mathbf{1} \otimes X = X. \quad (113)$$

2. The monoidal unit $\mathbf{1}$ has an identity morphism $\text{id}_{\mathbf{1}}$, and for any morphism

$$f : X \rightarrow Y, \quad f \otimes \text{id}_{\mathbf{1}} = f \quad \text{and} \quad \text{id}_{\mathbf{1}} \otimes f = f. \quad (114)$$

Example 25.46. We can look at **Moo** and ask whether it is a strict monoidal semicategory. The monoidal unit is given by the object

$$\mathbf{1} = \langle \rangle.$$

Its identity morphism is the Moore machine

$$\text{id}_{\mathbf{1}} = \langle \langle \rangle, \langle \rangle, \langle \rangle, \text{dyn}_{\mathbf{1}}, \text{ro}_{\mathbf{1}}, \langle \rangle \rangle,$$

where

$$\begin{aligned}\text{dyn}_{\mathbf{1}} : \langle \rangle \circledast \langle \rangle &\rightarrow \langle \rangle, \\ \langle \rangle \circledast \langle \rangle &\mapsto \langle \rangle,\end{aligned}$$

and

$$\begin{aligned}\text{ro}_{\mathbf{1}} : \langle \rangle &\rightarrow \langle \rangle, \\ \langle \rangle &\mapsto \langle \rangle.\end{aligned}$$

Clearly, $\mathbf{A} \circledast \langle \rangle = \langle \rangle \circledast \mathbf{A} = \mathbf{A}$ for every $\mathbf{A} \in \text{Ob}_{\text{Moo}}$. Furthermore, consider a Moore machine $f : \mathbf{U} \rightarrow \mathbf{Y}$ with

$$f = \langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle.$$

One has:

$$\begin{aligned} f \otimes \text{id}_1 &= \langle \mathbf{U} \circ \langle \rangle, \mathbf{X} \circ \langle \rangle, \mathbf{Y} \circ \langle \rangle, \text{dyn}_{f \otimes \text{id}_1}, \text{ro}_{f \otimes \text{id}_1}, \text{st} \circ \langle \rangle \rangle \\ &= \langle \mathbf{U}, \mathbf{X}, \mathbf{Y}, \text{dyn}, \text{ro}, \text{st} \rangle = f, \end{aligned}$$

where we used

$$\begin{aligned} \text{dyn}_{f \otimes \text{id}_1} : \mathbf{U} \circ \langle \rangle \circ \mathbf{X} \circ \langle \rangle \rightarrow \mathbf{X} \circ \langle \rangle \\ u \circ \langle \rangle \circ x \circ \langle \rangle \mapsto \text{dyn}(u, x) \circ \text{dyn}_1(\langle \rangle, \langle \rangle) = \text{dyn}(u, x) \end{aligned}$$

and

$$\begin{aligned} \text{ro}_{f \otimes \text{id}_1} : \mathbf{X} \circ \langle \rangle \rightarrow \mathbf{Y} \circ \langle \rangle \\ x \circ \langle \rangle \mapsto \text{ro}(x) \circ \text{ro}_1(\langle \rangle) = \text{ro}(x) \end{aligned}$$

to show the equivalences $\text{dyn} = \text{dyn}_{f \otimes \text{id}_1}$ and $\text{ro} = \text{ro}_{f \otimes \text{id}_1}$. The argument for $\text{id}_1 \otimes f$ follows analogously.



26. Crossing wires

26.1 Symmetric monoidal categories	378
26.2 PROPs	381

26.1. Symmetric monoidal categories

Symmetric strict monoidal categories

Definition 26.1

A *symmetric strict monoidal category* is

Constituents

1. a strict monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$,
2. for any two objects $X, Y \in \text{Ob}_{\mathbf{C}}$, an isomorphism

$$\text{br}_{X,Y} : X \otimes Y \rightarrow Y \otimes X, \quad (1)$$

called the *braiding*;

Conditions

1. *Naturality*: For any morphisms $f : X \rightarrow Z, g : Y \rightarrow U$, the diagram

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{\text{br}_{X,Y}} & Y \otimes X \\ f \otimes g \downarrow & & \downarrow g \otimes f \\ Z \otimes U & \xrightarrow{\text{br}_{Z,U}} & U \otimes Z \end{array} \quad (2)$$

commutes.

2. *Compatibility with nesting*:

$$\begin{array}{ccc} & (X \otimes Y) \otimes Z & \\ & \swarrow \text{br}_{X,Y \otimes Z} \quad \searrow \text{br}_{X,Y} \otimes \text{id}_Z & \\ X \otimes (Y \otimes Z) & & (Y \otimes X) \otimes Z \\ \downarrow \text{br}_{X,Y \otimes Z} & & \parallel \\ (Y \otimes Z) \otimes X & & Y \otimes (X \otimes Z) \\ & \swarrow \text{id}_Y \otimes \text{br}_{X,Z} & \\ & Y \otimes (Z \otimes X) & \end{array} \quad (3)$$

3. *Symmetry*: For all $X, Y \in \text{Ob}_{\mathbf{C}}$,

$$\text{br}_{X,Y} \circ \text{br}_{Y,X} = \text{id}_{X \otimes Y}. \quad (4)$$

Definition 26.2 (Symmetric monoidal category)

A *symmetric monoidal category* is

Constituents

1. A monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru} \rangle$,
2. for any two objects $X, Y \in \text{Ob}_{\mathbf{C}}$, an isomorphism

$$\text{br}_{X,Y} : X \otimes Y \rightarrow Y \otimes X, \quad (5)$$

called the *braiding*;

Conditions

1. *Naturality*: For any morphisms $f : X \rightarrow Z, g : Y \rightarrow U$, the diagram

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{\text{br}_{X,Y}} & Y \otimes X \\ f \otimes g \downarrow & & \downarrow g \otimes f \\ Z \otimes U & \xrightarrow{\text{br}_{Z,U}} & U \otimes Z \end{array} \quad (6)$$

commutes.

2. *Hexagon identities*: for any $X, Y, Z \in \text{Ob}_C$, the following diagrams must commute.

$$\begin{array}{ccccc} (X \otimes Y) \otimes Z & \xrightarrow{\text{br}_{X,Y} \otimes \text{id}_Z} & (Y \otimes X) \otimes Z & \xrightarrow{\text{as}_{Y,X,Z}} & Y \otimes (X \otimes Z) \\ \text{as}_{X,Y,Z} \downarrow & & & & \downarrow \text{id}_Y \otimes \text{br}_{X,Z} \\ X \otimes (Y \otimes Z) & \xrightarrow{\text{br}_{X,Y \otimes Z}} & (Y \otimes Z) \otimes X & \xrightarrow{\text{as}_{Y,Z,X}} & Y \otimes (Z \otimes X) \end{array} \quad (7)$$

$$\begin{array}{ccccc} X \otimes (Y \otimes Z) & \xrightarrow{\text{id}_X \otimes \text{br}_{Y,Z}} & X \otimes (Z \otimes Y) & \xrightarrow{\text{as}_{Y,X,Z}^{-1}} & (X \otimes Z) \otimes Y \\ \text{as}_{X,Y,Z}^{-1} \downarrow & & & & \downarrow \text{br}_{X,Z} \otimes \text{id}_Y \\ (X \otimes Y) \otimes Z & \xrightarrow{\text{br}_{X \otimes Y,Z}} & Z \otimes (X \otimes Y) & \xrightarrow{\text{as}_{Z,X,Y}^{-1}} & (Z \otimes X) \otimes Y \end{array} \quad (8)$$

3. *Symmetry*: for any $X, Y \in \text{Ob}_C$,

$$\text{br}_{X,Y} ; \text{br}_{Y,X} = \text{id}_{X \otimes Y} \quad (9)$$

Remark 26.3. In the presence of the symmetry condition (9), the two hexagon identities are actually redundant and only one of them is needed. However, if one drops the condition (9), then the above (with both hexagon identities) gives the definition of a *braided monoidal category*.

Remark 26.4. If $\langle C, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ is a symmetric monoidal category (or even a braided monoidal category), we can show that the following diagram commutes for all $X \in \text{Ob}_C$.

$$\begin{array}{ccc} \mathbf{1} \otimes X & \xrightarrow{\text{br}_{\mathbf{1},X}} & X \otimes \mathbf{1} \\ \text{lu}_X \searrow & & \swarrow \text{ru}_X \\ & X & \end{array} \quad (10)$$

DP is a symmetric monoidal category

We define a monoidal product \otimes for **DP** on objects by

$$P \otimes Q = P \times Q \quad (11)$$

and on morphisms by

$$\begin{array}{c} \mathbf{d} : P^{\text{op}} \times R \rightarrow \text{Pos} \text{ Bool} \quad \mathbf{e} : Q^{\text{op}} \times S \rightarrow \text{Pos} \text{ Bool} \\ \hline \mathbf{d} \otimes \mathbf{e} : (P \times Q)^{\text{op}} \times (R \times S) \rightarrow \text{Pos} \text{ Bool}, \\ \langle \langle a, c \rangle^*, \langle b, d \rangle \rangle \mapsto \mathbf{d}(a^*, b) \wedge \mathbf{e}(c^*, d), \end{array} \quad (12)$$

Lemma 26.5. There is a symmetric monoidal category $\langle \mathbf{DP}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ where the braiding is given by the design problem $\text{br}_{P,Q} : P \times Q \rightarrow Q \times P$ with

$$\text{br}_{P,Q}((p_1, q_1)^*, (q_2, p_2)) := (p_1 \leq_P p_2) \wedge (q_1 \leq_Q q_2) \quad (13)$$

for any $P, Q \in \text{Ob}_{\mathbf{DP}}$.

26.2. PROPs

Definition 26.6

A *prop* is a symmetric strict monoidal category **C** where the collection of objects is the natural numbers \mathbb{N} , the monoidal product on objects is addition of natural numbers, and the monoidal unit is $0 \in \mathbb{N}$.

Example 26.7. There is a prop **FinSet** where

- ▷ the set of morphisms from m to n (for $m, n \in \mathbb{N}$) is defined to be the set of functions from $\{1, \dots, m\}$ to $\{1, \dots, n\}$;
- ▷ composition is the usual composition of functions;
- ▷ identity morphisms are identity functions;
- ▷ the monoidal product of functions $f : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ and $f' : \{1, \dots, m'\} \rightarrow \{1, \dots, n'\}$ is the “disjoint union”

$$f + f' : \{1, \dots, m + m'\} \rightarrow \{1, \dots, n + n'\}. \quad (14)$$

Example 26.8. We define a prop **Mat_R** where:

- ▷ morphisms from m to n are $n \times m$ matrices with entries in \mathbb{R} (we also allow zero-dimensional matrices);
- ▷ composition is matrix multiplication ;
- ▷ identity morphisms are identity matrices;
- ▷ the monoidal product of matrices $\mathbf{A} : m \rightarrow n$ and $\mathbf{B} : m' \rightarrow n'$ is

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix} : m + m' \rightarrow n + n'. \quad (15)$$

Example 26.9. We define a prop **LinRel_R** where:

- ▷ morphisms from m to n are linear relations $\mathbb{R}^m \rightarrow \mathbb{R}^n$ (in other words, linear subspaces of $\mathbb{R}^m \oplus \mathbb{R}^n$);
- ▷ composition is composition of relations;
- ▷ identity morphisms are identity relations;
- ▷ the monoidal product of linear relations $R : m \rightarrow n$ and $S : m' \rightarrow n'$ is $R \oplus S : \mathbb{R}^m \oplus \mathbb{R}^{m'} \rightarrow \mathbb{R}^n \oplus \mathbb{R}^{n'}$, where

$$R \oplus S = \{\langle \langle v, v' \rangle, \langle w, w' \rangle \rangle \mid \langle v, w \rangle \in R \text{ and } \langle v', w' \rangle \in S\}. \quad (16)$$



27. Feedback

Some symmetric monoidal categories have, in addition to serial composition and parallel composition, also an operation that describes “feedback” or “recursion” or “iteration”. In terms of string diagram, this operation allows us to create loops.

27.1 Traced symmetric monoidal categories	384
27.2 Partial traces	390
27.3 Feedback categories	394
27.4 Dual objects and morphisms	396
27.5 Canonical trace	399

27.1. Traced symmetric monoidal categories

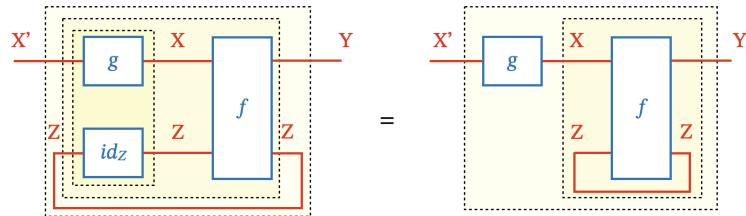


Figure 1.: Naturality in X

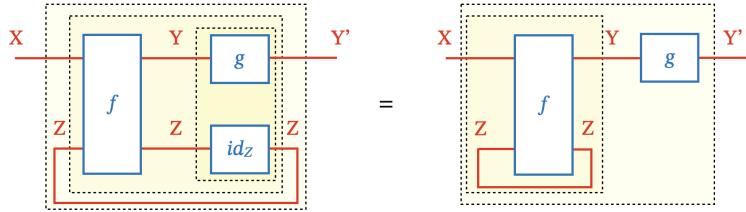


Figure 2.: Naturality in Y

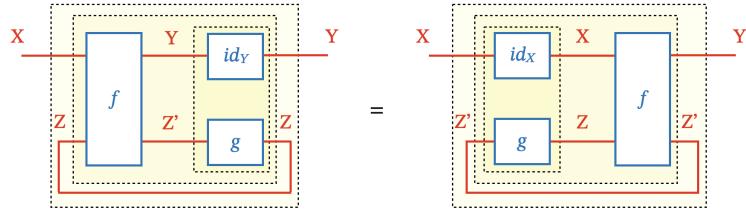


Figure 3.: Dinaturality in Z

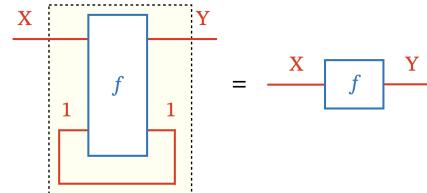


Figure 4.: Vanishing I

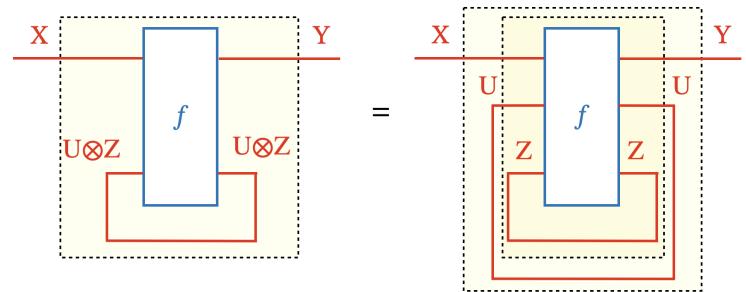


Figure 5.: Vanishing II

Definition 27.1 (Traced monoidal category)

We say that a symmetric monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ is *traced* if it is equipped with a *a trace operator*: a family of functions

$$\text{Tr}_{X,Y}^Z : \text{Hom}_{\mathbf{C}}(X \otimes Z; Y \otimes Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Y), \quad (1)$$

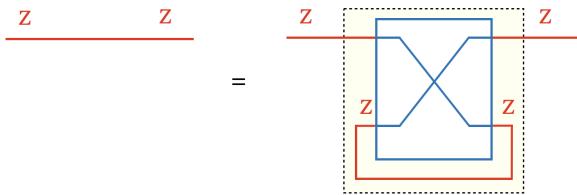


Figure 6.: Yanking

satisfying the following axioms:

1. *Naturality in X*: For any morphisms $f : X \otimes Z \rightarrow Y \otimes Z$ and $g : X' \rightarrow X$,

$$\text{Tr}_{X',Y}^Z((g \otimes \text{id}_Z) ; f) = g ; \text{Tr}_{X,Y}^Z(f) \quad (2)$$

2. *Naturality in Y*: For any morphisms $f : X \otimes Z \rightarrow Y \otimes Z$ and $g : Y \rightarrow Y'$,

$$\text{Tr}_{X,Y'}^Z(f ; (g \otimes \text{id}_Z)) = \text{Tr}_{X,Y}^Z(f) ; g \quad (3)$$

3. *Dinaturality in Z*: For any morphisms $f : X \otimes Z \rightarrow Y \otimes Z'$ and $g : Z' \rightarrow Z$,

$$\text{Tr}_{X,Y}^Z(f ; (\text{id}_Y \otimes g)) = \text{Tr}_{X,Y'}^{Z'}((\text{id}_X \otimes g) ; f). \quad (4)$$

4. *Vanishing I*: For any morphisms $f : X \otimes \mathbf{1} \rightarrow Y \otimes \mathbf{1}$ in \mathbf{C} ,

$$\text{Tr}_{X,Y}^{\mathbf{1}}(f) = \text{ru}_X^{-1} ; f ; \text{ru}_Y. \quad (5)$$

5. *Vanishing II*: For any morphism $f : (X \otimes Z) \otimes U \rightarrow (Y \otimes Z) \otimes U$ in \mathbf{C} ,

$$\text{Tr}_{X,Y}^Z(\text{Tr}_{X \otimes Z, Y \otimes Z}^U(f)) = \text{Tr}_{X,Y}^{Z \otimes U}(\text{as}_{X,Z,U} ; f ; \text{as}_{Y,Z,U}^{-1}). \quad (6)$$

6. *Superposing*: For any morphism $f : X \otimes Z \rightarrow Y \otimes Z$ in \mathbf{C} ,

$$\text{Tr}_{V \otimes X, V \otimes Y}^Z(\text{as}_{V,X,Z} ; \text{id}_V \otimes f ; \text{as}_{V,Y,Z}^{-1}) = \text{id}_V \otimes \text{Tr}_{X,Y}^Z(f). \quad (7)$$

7. *Yanking*:

$$\text{Tr}_{Z,Z}^Z(\text{br}_{Z,Z}) = \text{id}_Z. \quad (8)$$

Remark 27.2. Other variants of the definition of a traced monoidal category can be found in the literature. For instance, some include a more general version of the superposing law, see Lemma 27.3 below.

Lemma 27.3. Let $\langle \mathbf{C}, \otimes, \mathbf{1}_C, \text{br}, \text{Tr} \rangle$ be a traced monoidal category. Then a more general version of the superposing law holds: for any morphisms $f : X \otimes Z \rightarrow Y \otimes Z$ and $g : U \rightarrow V$,

$$\text{Tr}_{U \otimes X, V \otimes Y}^Z(g \otimes f) = g \otimes \text{Tr}_{X,Y}^Z(f). \quad (9)$$

Definition 27.4

For the symmetric monoidal category $\langle \mathbf{Rel}, \times, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$, a trace is defined as follows. Given a relation

$$R : \mathbf{A} \times \mathbf{C} \rightarrow \mathbf{B} \times \mathbf{C}, \quad (10)$$

we set

$$\text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathcal{R}) = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid \exists z \in \mathbf{C} : \langle \langle x, z \rangle, \langle y, z \rangle \rangle \in \mathcal{R}\} \quad (11)$$

Definition 27.5

Consider the symmetric monoidal category $\langle \mathbf{Rel}, +, \emptyset, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$. In \mathbf{Rel} , the disjoint union of any two sets always comes with canonical inclusion relations

$$\begin{aligned} \text{in}_{\mathbf{A}} : \mathbf{A} &\rightarrow \mathbf{A} + \mathbf{B} & \text{in}_{\mathbf{B}} : \mathbf{B} &\rightarrow \mathbf{A} + \mathbf{B} \\ \{\langle a, \langle 1, a \rangle \rangle\}, && \{\langle b, \langle 2, b \rangle \rangle\}, & \end{aligned} \quad (12)$$

and canonical projection relations

$$\begin{aligned} \text{pr}_{\mathbf{A}} : \mathbf{A} + \mathbf{B} &\rightarrow \mathbf{A} & \text{pr}_{\mathbf{B}} : \mathbf{A} + \mathbf{B} &\rightarrow \mathbf{B} \\ \{\langle \langle 1, a \rangle, a \rangle\}, && \{\langle \langle 1, a \rangle, b \rangle\}. & \end{aligned} \quad (13)$$

Given any relation of type

$$\mathcal{R} : \mathbf{A} + \mathbf{C} \rightarrow \mathbf{B} + \mathbf{D}, \quad (14)$$

we can pre-compose it with the inclusion $\text{in}_{\mathbf{A}} : \mathbf{A} \rightarrow \mathbf{A} + \mathbf{C}$ and post-compose it with the relation $\text{pr}_{\mathbf{A}} : \mathbf{A} + \mathbf{C} \rightarrow \mathbf{B}$ to obtain a relation

$$\mathcal{R}_{\mathbf{AB}} : \mathbf{A} \xrightarrow{\text{in}_{\mathbf{A}}} \mathbf{A} + \mathbf{C} \xrightarrow{\mathcal{R}} \mathbf{B} + \mathbf{C} \xrightarrow{\text{pr}_{\mathbf{B}}} \mathbf{B}. \quad (15)$$

In an analogous manner we also obtain relations

$$\mathcal{R}_{\mathbf{CB}} : \mathbf{A} \rightarrow \mathbf{B}, \quad \mathcal{R}_{\mathbf{AC}} : \mathbf{C} \rightarrow \mathbf{B}, \quad \mathcal{R}_{\mathbf{CC}} : \mathbf{C} \rightarrow \mathbf{C}, \quad (16)$$

induced from \mathcal{R} .

Now we set

$$\text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathcal{R}) = \mathcal{R}_{\mathbf{AB}} \cup \bigcup_{n \geq 0} \mathcal{R}_{\mathbf{AC}} ; \mathcal{R}_{\mathbf{CC}}^n ; \mathcal{R}_{\mathbf{CB}}. \quad (17)$$

Graded exercise H.6 (TracingRelations)

Given the symmetric monoidal category $\langle \mathbf{Rel}, \times, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$, consider the trace operation

$$\text{Tr}_{X, Y}^Z : \text{Hom}_{\mathbf{Rel}}(X \otimes Z; Y \otimes Z) \rightarrow \text{Hom}_{\mathbf{Rel}}(X; Y)$$

which is defined, for a morphism $R \in \text{Hom}_{\mathbf{C}}(X \otimes Z; Y \otimes Z)$, by

$$\text{Tr}_{X, Y}^Z(R) = \{\langle x, y \rangle \in X \times Y \mid \exists z \in Z : \langle x \otimes z, y \otimes z \rangle \in R\}. \quad (18)$$

Your task is to check that this definition satisfies the following two trace axioms:

1. *Vanishing II*:

For any relation $R : X \otimes Z \otimes U \rightarrow Y \otimes Z \otimes U$ in \mathbf{Rel} ,

$$\text{Tr}_{X, Y}^{Z \otimes U}(R) = \text{Tr}_{X, Y}^Z \left(\text{Tr}_{X \otimes Z, Y \otimes Z}^U(R) \right). \quad (19)$$

2. *Superposing*:

For any relations $R : X \circ Z \rightarrow Y \circ Z$ and $S : V \rightarrow W$ in $\langle \text{Rel} \rangle$,

$$\text{Tr}_{V \circ X, W \circ Y}^Z(S \otimes R) = S \otimes \text{Tr}_{X, Y}^Z(R). \quad (20)$$

Trace in co-design

Suppose that we are given a design problem with a resource and a functionality of the same type \mathbf{R} (Fig. 7). Can we “close the loop”, as in the diagram reported in Fig. 8?

It turns out that we can give a well-defined semantics to this loop-closing operation, which coincides with the notion of a *trace* in category theory.

The following is the formal definition of the trace operation for design problems.

Definition 27.6 (Trace of a design problem)

Given a design problem $\mathbf{d} : \mathbf{P} \otimes \mathbf{R} \rightarrow \mathbf{Q} \otimes \mathbf{R}$, its *trace*

$$\text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{R}}(\mathbf{d}) : \mathbf{P} \rightarrow \mathbf{Q} \quad (21)$$

is defined as follows:

$$\begin{aligned} \text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{R}}(\mathbf{d}) : \mathbf{P}^{\text{op}} \times \mathbf{Q} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle p^*, q \rangle &\mapsto \bigvee_{r \in \mathbf{R}} \mathbf{d}(\langle p, r \rangle^*, \langle q, r \rangle). \end{aligned} \quad (22)$$

Think of drawing a loop as a way of writing down the following requirement: Something that produces \mathbf{R} should not use up more of \mathbf{R} than it produces.

Lemma 27.7. Trace as in Def. 27.6 satisfies the trace axioms. In other words, $\langle \mathbf{DP}, \otimes, \mathbf{1}, \sigma \rangle$ is a traced monoidal category, with trace as in (22).

Proof. We have already shown that $\langle \mathbf{DP}, \otimes, \mathbf{1}, \sigma \rangle$ is a symmetric monoidal category (Lemma 26.5). We prove the trace axioms one by one, starting from vanishing ((5), (6)). Given any $\mathbf{P}, \mathbf{Q} \in \text{Ob}_{\mathbf{DP}}$ and $\mathbf{d} : \mathbf{P} \times \mathbf{1} \rightarrow \mathbf{Q} \times \mathbf{1}$ in \mathbf{DP} , we have

$$\begin{aligned} &= \text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{1}}(\mathbf{d})(p^*, q) \\ &= \bigvee_{r \in \mathbf{1}} \mathbf{d}(\langle p, r \rangle^*, \langle q, r \rangle) \\ &= \mathbf{d}(\langle p, \bullet \rangle^*, \langle q, \bullet \rangle) \\ &= \mathbf{d}(p^*, q). \end{aligned} \quad (23)$$

Furthermore, for any morphism $\mathbf{d} : \mathbf{P} \times \mathbf{X} \times \mathbf{Y} \rightarrow \mathbf{Q} \times \mathbf{X} \times \mathbf{Y}$ in \mathbf{DP} , we have

$$\begin{aligned} &\text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{X} \times \mathbf{Y}}(\mathbf{d})(p^*, q) \\ &= \bigvee_{\langle x, y \rangle \in \mathbf{X} \times \mathbf{Y}} \mathbf{d}(\langle p, x, y \rangle^*, \langle q, x, y \rangle) \\ &= \bigvee_{x \in \mathbf{X}} \left(\bigvee_{y \in \mathbf{Y}} \mathbf{d}(\langle p, x, y \rangle^*, \langle q, x, y \rangle) \right) \\ &= \text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{X}} \left(\text{Tr}_{\mathbf{P} \times \mathbf{X}, \mathbf{Q} \times \mathbf{X}}^{\mathbf{Y}}(\mathbf{d})(\langle p, x \rangle^*, \langle q, x \rangle) \right). \end{aligned} \quad (24)$$

For the superposing axiom ((7)), consider $\mathbf{d} : \mathbf{P} \times \mathbf{X} \rightarrow \mathbf{Q} \times \mathbf{X}$ in \mathbf{DP} . We

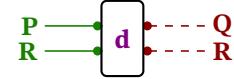


Figure 7.: Design problem with a resource and a functionality of the same type.

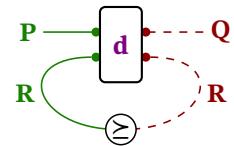


Figure 8.: Closing the loop in the design problem.

have

$$\begin{aligned}
 & \text{Tr}_{\mathbf{R} \times \mathbf{P}, \mathbf{R} \times \mathbf{Q}}^{\mathbf{X}}(\text{id}_R \otimes \mathbf{d})(\langle r_1, p \rangle^*, \langle r_2, q \rangle) \\
 &= \bigvee_{x \in X} \text{id}_R(r_1^*, r_2) \wedge \mathbf{d}(\langle p, x \rangle^*, \langle q, x \rangle) \\
 &= \text{id}_R(r_1^*, r_2) \wedge \bigvee_{x \in X} \mathbf{d}(\langle p, x \rangle^*, \langle q, x \rangle) \\
 &= (\text{id}_R \otimes \text{Tr}_{\mathbf{P}, \mathbf{Q}}^{\mathbf{X}}(\mathbf{d}))(\langle r_1, p \rangle^*, \langle r_2, q \rangle).
 \end{aligned} \tag{25}$$

Finally, for yanking (8) consider $\sigma_{X,X}$. We have

$$\begin{aligned}
 & \text{Tr}_{\mathbf{P}, \mathbf{P}}^{\mathbf{P}}(\sigma_{\mathbf{P}, \mathbf{P}})(p_1^*, p_2) \\
 &= \bigvee_{p \in P} \sigma_{\mathbf{P}, \mathbf{P}}(\langle p_1, p \rangle^*, \langle p, p_2 \rangle) \\
 &= \bigvee_{p \in P} p_1 \leq p_2 \wedge p \leq p \\
 &= \bigvee_{p \in P} p_1 \leq p_2 \\
 &= \text{id}_P(p_1^*, p_2).
 \end{aligned} \tag{26}$$

□

Graded exercise H.7 (DPSnakeTracePart2)

In this exercise we work again with the category **DP** of posets and design problems, equipped with the symmetric monoidal structure where the monoidal product is the cartesian product of posets. In the following we make the identification

$$(\mathbf{P} \times \mathbf{Q})^{\text{op}} = \mathbf{P}^{\text{op}} \times \mathbf{Q}^{\text{op}} \tag{27}$$

for any posets \mathbf{P}, \mathbf{Q} . Also, recall that $(\mathbf{P}^{\text{op}})^{\text{op}} = \mathbf{P}$.

In components, the associator for **DP** is

$$\text{as}_{\mathbf{P}, \mathbf{Q}, \mathbf{R}} : ((\mathbf{P} \times \mathbf{Q}) \times \mathbf{R})^{\text{op}} \times (\mathbf{P} \times (\mathbf{Q} \times \mathbf{R})) \rightarrow \mathbf{Bool} \tag{28}$$

with

$$\text{as}_{\mathbf{P}, \mathbf{Q}, \mathbf{R}}(\langle \langle p_1^*, q_1^* \rangle, r_1^* \rangle, \langle p_2, \langle q_2, r_2 \rangle \rangle) = p_1 \leq p_2 \wedge q_1 \leq q_2 \wedge r_1 \leq r_2, \tag{29}$$

and the left unit is

$$\text{lu}_{\mathbf{P}} : (\mathbf{1} \times \mathbf{P})^{\text{op}} \times \mathbf{P} \rightarrow \mathbf{Bool} \tag{30}$$

with

$$\text{lu}_{\mathbf{P}}(\langle \langle \bullet^*, p_1^* \rangle, p_2 \rangle) = p_1 \leq p_2. \tag{31}$$

The right unit is analogous.

The braiding

$$\text{br}_{\mathbf{P}, \mathbf{Q}} : \mathbf{P} \times \mathbf{Q} \leftrightarrow \mathbf{Q} \times \mathbf{P}$$

is

$$\text{br}_{\mathbf{P}, \mathbf{Q}}(\langle p_1^*, q_1^* \rangle, \langle q_2, p_2 \rangle) := (p_1 \leq_P p_2) \wedge (q_1 \leq_Q q_2). \tag{32}$$

We define the following duality data, with respect to which **DP** is compact closed:

- ▷ $\mathbf{P}^V := \mathbf{P}^{\text{op}}$
- ▷ $\text{ev}_{\mathbf{P}} : (\mathbf{P}^{\text{op}} \times \mathbf{P})^{\text{op}} \times \{\bullet\} \rightarrow \mathbf{Bool}, \quad \langle\langle x^*, y \rangle^*, \bullet \rangle \mapsto y \leq_{\mathbf{P}} x$
- ▷ $\text{coev}_{\mathbf{P}} : \{\bullet\}^{\text{op}} \times (\mathbf{P} \times \mathbf{P}^{\text{op}}) \rightarrow \mathbf{Bool}, \quad \langle \bullet, \langle x, y^* \rangle \rangle \mapsto y \leq_{\mathbf{P}} x$

Your task: given a morphism $\mathbf{d} : \mathbf{P} \times \mathbf{R} \rightarrow \mathbf{Q} \times \mathbf{R}$ in \mathbf{DP} , show that the design problem $\mathbf{P} \rightarrow \mathbf{Q}$ given by the following composition

$$\mathbf{ru}_{\mathbf{P}}^{-1} \circ (\text{id}_{\mathbf{P}} \otimes \text{coev}_{\mathbf{R}}) \circ \text{as}_{\mathbf{P}, \mathbf{R}, \mathbf{R}^{\text{op}}}^{-1} \circ (\mathbf{d} \otimes \text{id}_{\mathbf{R}^{\text{op}}}) \circ \text{as}_{\mathbf{Q}, \mathbf{R}, \mathbf{R}^{\text{op}}} \circ (\text{id}_{\mathbf{Q}} \otimes \text{br}_{\mathbf{R}, \mathbf{R}^{\text{op}}}) \circ (\text{id}_{\mathbf{Q}} \otimes \text{ev}_{\mathbf{R}}) \circ \mathbf{ru}_{\mathbf{Q}} \quad (33)$$

is equal to the design problem $\mathbf{P} \rightarrow \mathbf{Q}$ given by

$$\begin{aligned} & \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow \mathbf{Pos} \quad \mathbf{Bool}, \\ & \langle p^*, q \rangle \mapsto \bigvee_{r \in \mathbf{R}} \mathbf{d}(\langle p, r \rangle^*, \langle q, r \rangle). \end{aligned} \quad (34)$$

Trace of a linear transformation

Consider the category $\mathbf{FinVect}_{\mathbb{R}}$ of finite dimensional real vector spaces, which has as objects finite dimensional vector spaces and as morphisms linear maps between them. Using the tensor product \otimes of real vector spaces as monoidal product, we can show $\mathbf{FinVect}_{\mathbb{R}}$ is a monoidal category. Consider a linear transformation $f : B \otimes D \rightarrow C \otimes D$, with B, C, D vector spaces with bases $\{b_i\}, \{c_j\}$, and $\{d_k\}$ respectively. Here, the trace is a linear function $\text{Tr}_{B,C}^D(f) : B \rightarrow C$, given by

$$(\text{Tr}_{B,C}^D(f))_{i,j} = \sum_k f_{i \otimes k, j \otimes k} \quad (35)$$

Trace for symmetric strict monoidal categories

Definition 27.8 (Trace for $\langle \mathbf{Rel} \rangle$ with concatenation as monoidal product)
 Consider the symmetric strict monoidal category $\langle \mathbf{Rel} \rangle$, where the monoidal product is defined via the concatenation operation \circ for lists of sets. Given a relation $R : \langle A, C \rangle \rightarrow \langle B, C \rangle$, its *trace* is defined as

$$\text{Tr}_{\langle A \rangle, \langle B \rangle}^{\langle C \rangle}(R) = \{ \langle x, y \rangle \in \langle A \rangle \times \langle B \rangle \mid \exists z \in \langle C \rangle : \langle x \circ z, y \circ z \rangle \in R \} \quad (36)$$

27.2. Partial traces

Definition 27.9

Let X and Y be sets. A **partial function** f from X to Y , written $f : X \rightharpoonup Y$, is a function $f : U_f \rightarrow Y$ for some subset $U_f \subset X$.

For partial functions $f, g : X \rightharpoonup Y$ and $x \in X$ we define the following:

- ▷ We write $f(x) \downarrow$ if $x \in U_f$, i.e. $f(x)$ is defined.
- ▷ We write $f(x) \uparrow$ if $x \notin U_f$, i.e. $f(x)$ is undefined.
- ▷ We say **Kleene equality at x holds** and write $f(x) \asymp g(x)$, if either $f(x) = g(x)$ and $f(x)$ and $g(x)$ are defined, or $f(x)$ and $g(x)$ are undefined.
- ▷ We say **directed Kleene equality at x holds** and write $f(x) \gtrsim g(x)$, if either $f(x)$ is undefined or else $f(x)$ and $g(x)$ are both defined and $f(x) = g(x)$. Similarly, we write $f(x) \lesssim g(x)$ for the case when the roles of f and g are reversed.

Definition 27.10

Let $\langle C, \otimes, 1, \text{br} \rangle$ be a symmetric strict monoidal category. Then it is called **partially traced** if it is equipped with family of partial functions

$$\text{Tr}_{X,Y}^Z : \text{Hom}_C(X \otimes Z; Y \otimes Z) \rightarrow \text{Hom}_C(X; Y)$$

for all objects $X, Y, Z \in \text{Ob}_C$ such that the following axioms are satisfied:

1. **Tightening** (naturality in X, Y) For all objects X, Y, X', Y', Z and morphisms $g : X' \rightarrow X$, $f : X \otimes Z \rightarrow Y \otimes Z$ and $h : Y \rightarrow Y'$ it holds that

$$\text{Tr}_{X',Y'}^Z((g \otimes \text{id}_Z) ; f ; (h \otimes \text{id}_Z)) \gtrsim g ; \text{Tr}_{X,Y}^Z(f) ; h.$$

2. **Sliding** (naturality in Z) For all objects X, Y, Z, Z' and morphisms $f : X \otimes Z \rightarrow Y \otimes Z'$ and $g : Z' \rightarrow Z$ it holds that

$$\text{Tr}_{X,Y}^Z(f ; (\text{id}_Y \otimes g)) \asymp \text{Tr}_{X,Y}^{Z'}((\text{id}_X \otimes g) ; f).$$

3. **Vanishing** For all objects X, Y, Z, Z' and morphisms $g : X \rightarrow Y$ and $f : X \otimes Z \otimes Z' \rightarrow Y \otimes Z \otimes Z'$ it holds that

$$\text{Tr}_{X,Y}^1(g) \asymp g$$

and if $\text{Tr}_{X \otimes Z, Y \otimes Z}^{Z'}(f) \downarrow$, then

$$\text{Tr}_{X,Y}^{Z \otimes Z'}(f) \asymp \text{Tr}_{X,Y}^Z(\text{Tr}_{X \otimes Z, Y \otimes Z}^{Z'}(f)).$$

4. **Strength** For all objects X, Y, X', Y', Z and morphisms $f : X' \otimes Z \rightarrow Y' \otimes Z$ and $g : X \rightarrow Y$ it holds that

$$\text{Tr}_{X \otimes X', Y \otimes Y'}^Z(g \otimes f) \gtrsim g \otimes \text{Tr}_{X',Y'}^Z(f).$$

5. **Yanking** For all objects Z it holds that

$$\text{Tr}_{Z,Z}^Z(\text{br}_{Z,Z}) \asymp \text{id}_Z.$$

Definition 27.11

Consider the symmetric strict monoidal category $\text{Mat}_{\mathbb{R}}$ (objects are natural numbers and a morphism $m \rightarrow n$ is a $n \times m$ matrix with entries in \mathbb{R}), with

monoidal stacking operation

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix}. \quad (37)$$

A partial trace operation is defined as follows. Given

$$\underline{\mathbf{f}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} : m + k \rightarrow n + k \quad (38)$$

we set

$$\text{Tr}_{m,n}^k(\underline{\mathbf{f}}) = \mathbf{A} + \mathbf{C}(\mathbf{I} - \mathbf{D})^{-1}\mathbf{B} \quad (39)$$

if $\mathbf{I} - \mathbf{D}$ is invertible, and otherwise $\text{Tr}_{m,n}^k(\underline{\mathbf{f}})$ is undefined.

Remark 27.12. How might one arrive at the formula for the trace in Def. 27.11? Here are two informal derivations of why it is a plausible and suitable guess for a formula for a (partial) trace.

- One intuition is to treat this case similarly to the case of **Rel** when it is equipped with the cartesian product as monoidal product. If we think of a morphism

$$\underline{\mathbf{f}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} : m + k \rightarrow n + k \quad (40)$$

as a function (or: a relation)

$$\mathbb{R}^m \oplus \mathbb{R}^k \rightarrow \mathbb{R}^n \oplus \mathbb{R}^k \quad (41)$$

then a natural condition for feedback is to consider the equation

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} \quad (42)$$

and think of \mathbf{u} as input, \mathbf{y} as output, and \mathbf{x} as a state variable. From (42), the output \mathbf{y} is given then by the formula

$$\mathbf{y} = \mathbf{Au} + \mathbf{Cx} \quad (43)$$

where \mathbf{x} is required to satisfy the recursive equation

$$\mathbf{x} = \mathbf{Bu} + \mathbf{Dx} \quad (44)$$

which, if $\mathbf{I} - \mathbf{D}$ is invertible, may be solved thus:

$$\mathbf{x} = (\mathbf{I} - \mathbf{D})^{-1}\mathbf{Bu}. \quad (45)$$

Substituting this formula in (43), we obtain

$$\mathbf{y} = \mathbf{Au} + \mathbf{C}(\mathbf{I} - \mathbf{D})^{-1}\mathbf{Bu} \quad (46)$$

- Another way to think of the trace formula (39) is in analogy to the trace for **Rel** equipped with the sum of sets as monoidal product. There we noted that a relation of the type

$$\underline{R} : \underline{\mathbf{A}} + \underline{\mathbf{C}} \rightarrow \underline{\mathbf{B}} + \underline{\mathbf{C}} \quad (47)$$

gives rise to four relations, which we denoted R_{AB} , R_{CB} , R_{AC} , and R_{CC} . These are in analogy to the components \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} , respectively, of the matrix

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}. \quad (48)$$

The analogue here to the formula (17) for the trace of a relation is to set

$$\text{Tr}_{\mathbf{m}, \mathbf{n}}^{\mathbf{k}}(\mathbf{f}) = \mathbf{A} + \mathbf{C} \left(\sum_{i=0}^{\infty} \mathbf{D}^i \right) \mathbf{B}, \quad (49)$$

which, under suitable assumptions, is well-defined and equal to (39) because the geometric series $\sum_{i=0}^{\infty} \mathbf{D}^i$ then converges to $(\mathbf{I} - \mathbf{D})^{-1}$.

LTI systems

Let $\mathbf{f} = \langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$ be a LTI system from $\mathbf{l} \in \mathbb{N}$ to $\mathbf{m} \in \mathbb{N}$, and suppose we are given factorizations $\mathbf{l} = \mathbf{i} + \mathbf{k}$ and $\mathbf{m} = \mathbf{j} + \mathbf{k}$ of the dimension of the input and output spaces, \mathbf{U} and \mathbf{Y} , respectively. Then we can think of $\mathbf{U} = \mathbb{R}^l$ as an (internal) direct sum of the form $\mathbb{R}^l = \mathbb{R}^i \oplus \mathbb{R}^k$, and similarly so for $\mathbf{U} = \mathbb{R}^m = \mathbb{R}^j \oplus \mathbb{R}^k$. We will use the notation $\mathbf{U} = \mathbf{U}_1 \oplus \mathbf{U}_2$ and $\mathbf{Y} = \mathbf{U}_1 \oplus \mathbf{U}_2$, respectively for these factorizations. This induces corresponding factorizations of the matrices \mathbf{B} , \mathbf{C} , and \mathbf{D} as block matrices:

$$\mathbf{B} = [\mathbf{B}_1 \quad \mathbf{B}_2] \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}_{21} & \mathbf{D}_{22} \end{bmatrix}. \quad (50)$$

Definition 27.13

Let an LTI system $\mathbf{f} = \langle \mathbf{st}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \rangle$ and factorizations $\mathbf{l} = \mathbf{i} + \mathbf{k}$ and $\mathbf{m} = \mathbf{j} + \mathbf{k}$ be given, and let

$$\mathbf{B} = [\mathbf{B}_1 \quad \mathbf{B}_2] \quad \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} \\ \mathbf{D}_{21} & \mathbf{D}_{22} \end{bmatrix} \quad (51)$$

be the corresponding factorizations of \mathbf{B} , \mathbf{C} , and \mathbf{D} . If the matrix $\mathbf{I} - \mathbf{D}_{22}$ is invertible, we define the LTI system $\text{Tr}_{\mathbf{i}, \mathbf{j}}^{\mathbf{k}}(\mathbf{f})$ as

$$\begin{aligned} & (\mathbf{st}, \mathbf{A} + \mathbf{B}_2(\mathbf{I} - \mathbf{D}_{22})^{-1}\mathbf{C}_2, \mathbf{B}_1 + \mathbf{D}_2(\mathbf{I} - \mathbf{D}_{22})^{-1}\mathbf{D}_{21}, \\ & \quad \mathbf{C}_1 + \mathbf{D}_{12}(\mathbf{I} - \mathbf{D}_{22})^{-1}\mathbf{C}_2, \mathbf{D}_{11} + \mathbf{D}_{12}(\mathbf{I} - \mathbf{D}_{22})^{-1}\mathbf{D}_{21}). \end{aligned} \quad (52)$$

Example 27.14. Let's consider the simple signal-flow diagram reported in Fig. 9. Note that the represented signals are scalar. In basic engineering classes, you learn that you can find an expression of the output $y(t)$ as a function of the input $u(t)$, by following the diagram. In particular, one can write

$$\begin{aligned} K(u(t) - Cy(t)) &= y(t) \Leftrightarrow Ku(t) - KCy(t) = y(t) \\ &\Leftrightarrow Ku(t) = y(t) + KCy(t) \\ &\Leftrightarrow Ku(t) = y(t)(1 + KC) \\ &\Leftrightarrow y(t) = \frac{K}{1 + KC}u(t). \end{aligned} \quad (53)$$

Now, we want to get the same expression, but interpreting the presented system as a composition of LTI systems, and leveraging the newly introduced concept of trace.

This can be visualized as in Fig. 10. The systems are given by

$$\textcolor{blue}{f} = \langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 1}, \mathbf{0}^{1 \times 0}, [1 \ -1] \rangle$$

$$\textcolor{blue}{g} = \langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 1}, \mathbf{0}^{1 \times 0}, K \rangle$$

$$\textcolor{blue}{h} = \left\langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 1}, \mathbf{0}^{1 \times 0}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\rangle$$

$$\textcolor{blue}{i} = \langle \mathbf{0}^{0 \times 1}, \mathbf{0}^{0 \times 0}, \mathbf{0}^{0 \times 1}, \mathbf{0}^{1 \times 0}, \begin{bmatrix} 1 & 0 \\ 0 & C \end{bmatrix} \rangle$$

Intuitively, $\textcolor{blue}{f}$ is acting as the subtraction, $\textcolor{blue}{g}$ as the gain K , $\textcolor{blue}{h}$ is splitting the signal in two identical copies, one of which is used by the controller, expressed via $\textcolor{blue}{i}$. All of these LTI systems are described by their last component, and are therefore explicit input-output relationships. We can compose the LTI systems.

We just look at the last component of the composition, given by:

$$\begin{aligned} \mathbf{D} &= \mathbf{D}_i \mathbf{D}_h \mathbf{D}_g \mathbf{D}_f \\ &= \begin{bmatrix} 1 & 0 \\ 0 & C \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} K \begin{bmatrix} 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} K \\ KC \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} K & -K \\ CK & -CK \end{bmatrix} \end{aligned}$$

We can now apply the formula for the trace and we get:

$$\begin{aligned} \text{pr}_5(\text{Tr}_{1,1}^w) &= K - K(1 + CK)^{-1}CK \\ &= \frac{K}{1 + CK}. \end{aligned}$$

From this we get the LTI system from (53) (in other words, a direct input-output dependency).

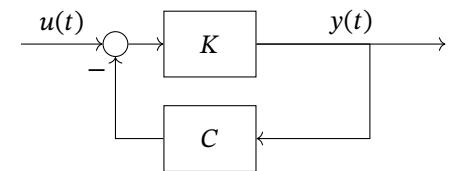


Figure 9.: Example with signal-flow diagram.

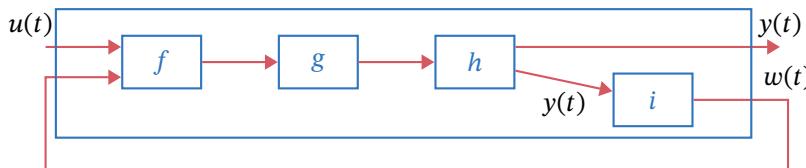


Figure 10.: Signal-flow diagram transformed into composition of LTI systems.

27.3. Feedback categories

The following definition of a feedback operator is identical with the definition of a trace operator, except for two important differences:

1. the dinaturality axiom for the feedback operator is only required to hold for isomorphisms;
2. the yanking axiom is omitted completely.

Definition 27.15 (Feedback category)

We call a symmetric monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ a *feedback category* if it is equipped with a *feedback operator*: a family of functions

$$\text{Fb}_{X,Y}^Z : \text{Hom}_{\mathbf{C}}(X \otimes Z; Y \otimes Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Y), \quad (54)$$

satisfying the following axioms:

1. *Naturality in X*: For any morphisms $f : X \otimes Z \rightarrow Y \otimes Z$ and $g : X' \rightarrow X$,

$$\text{Fb}_{X',Y}^Z((g \otimes \text{id}_Z) ; f) = g ; \text{Fb}_{X,Y}^Z(f) \quad (55)$$

2. *Naturality in Y*: For any morphisms $f : X \otimes Z \rightarrow Y \otimes Z$ and $g : Y \rightarrow Y'$,

$$\text{Fb}_{X,Y'}^Z(f ; (g \otimes \text{id}_Z)) = \text{Fb}_{X,Y}^Z(f) ; g \quad (56)$$

3. *Dinaturality in Z with respect to isomorphisms*: For any morphism $f : X \otimes Z \rightarrow Y \otimes Z'$ and any isomorphism $g : Z' \rightarrow Z$,

$$\text{Fb}_{X,Y}^Z(f ; (\text{id}_Y \otimes g)) = \text{Fb}_{X,Y'}^{Z'}((\text{id}_X \otimes g) ; f). \quad (57)$$

4. *Vanishing I*: For any morphisms $f : X \otimes \mathbf{1} \rightarrow Y \otimes \mathbf{1}$ in \mathbf{C} ,

$$\text{Fb}_{X,Y}^{\mathbf{1}}(f) = \text{ru}_X^{-1} ; f ; \text{ru}_Y. \quad (58)$$

5. *Vanishing II*: For any morphism $f : (X \otimes Z) \otimes U \rightarrow (Y \otimes Z) \otimes U$ in \mathbf{C} ,

$$\text{Fb}_{X,Y}^Z \left(\text{Fb}_{X \otimes Z, Y \otimes Z}^U(f) \right) = \text{Fb}_{X,Y}^{Z \otimes U}(\text{as}_{X,Z,U} ; f ; \text{as}_{Y,Z,U}^{-1}). \quad (59)$$

6. *Superposing*: For any morphism $f : X \otimes Z \rightarrow Y \otimes Z$ in \mathbf{C} ,

$$\text{Fb}_{V \otimes X, V \otimes Y}^Z(\text{as}_{V,X,Z} ; \text{id}_V \otimes f ; \text{as}_{V,Y,Z}^{-1}) = \text{id}_V \otimes \text{Fb}_{X,Y}^Z(f). \quad (60)$$

Example 27.16. Consider the symmetric monoidal category \mathbf{Mealy}/\sim of Mealy machines modulo congruence, where objects are sets and where a morphism $\mathbf{A} \rightarrow \mathbf{B}$ is an equivalence class represented by a pair $\langle \mathbf{S}, f \rangle$ consisting of a set \mathbf{S} (which we think of as a state space) and a function

$$f : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{B} \times \mathbf{S}. \quad (61)$$

A feedback operator for \mathbf{Mealy}/\sim is defined as follows. Given a morphism of the type

$$[g] : \mathbf{A} \times \mathbf{C} \rightarrow \mathbf{B} \times \mathbf{C}, \quad (62)$$

represented by a function

$$g : (\mathbf{A} \times \mathbf{C}) \times \mathbf{S} \rightarrow (\mathbf{B} \times \mathbf{C}) \times \mathbf{S}, \quad (63)$$

we let

$$\text{Fb}_{\mathbf{AB}}^{\mathbf{C}}([\mathbf{g}]) : \mathbf{A} \rightarrow \mathbf{B} \quad (64)$$

be the morphism in \mathbf{Mealy}/\sim represented by the function

$$\mathbf{A} \times (\mathbf{C} \times \mathbf{S}) \xrightarrow{\text{as}^{-1}} (\mathbf{A} \times \mathbf{C}) \times \mathbf{S} \xrightarrow{\mathbf{g}} (\mathbf{B} \times \mathbf{C}) \times \mathbf{S} \xrightarrow{\text{as}} \mathbf{B} \times (\mathbf{C} \times \mathbf{S}). \quad (65)$$

In other words, the feedback operator $\text{Fb}_{\mathbf{AB}}^{\mathbf{C}}$ simply shifts \mathbf{C} to being part of the state space, instead of as being part of the input and output spaces.

27.4. Dual objects and morphisms

There is a concept of “duality” for objects in a monoidal category which we will introduce with an illustrative example.

We have seen in Example 25.29 that the category $\mathbf{C} = \mathbf{Vect}_{\mathbb{R}}$ of real vector spaces is symmetric monoidal, with tensor product as the monoidal product. Given a vector space V , its *linear dual* is the real vector space

$$V^* := \{\text{linear maps } V \rightarrow \mathbb{R}\} = \text{Hom}_{\mathbf{C}}(V; \mathbb{R}). \quad (66)$$

Recall from linear algebra the following fact about any vector space V :

$$V \simeq (V^*)^* \text{ if and only if } \dim V < \infty. \quad (67)$$

One might say that the finite-dimensional real vector spaces are characterizable based on their behavior in this way with respect to the operation of taking the linear dual.

We will develop an alternative formulation of this fact, based on the notion of a *dualizable object*. This notion will make sense in the setting of any (symmetric) monoidal category, and we will see then, that (67) translates to the statement

$$V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}} \text{ is dualizable if and only if } \dim V < \infty. \quad (68)$$

Key protagonists in this reformulation are *evaluation* and *coevaluation* maps.

In the following, V denotes a *finite-dimensional* real vector space. The evaluation map ev_V associated to V is

$$\begin{aligned} \text{ev}_V : V^* \otimes V &\rightarrow \mathbb{R}, \\ \langle l, v \rangle &\mapsto l(v). \end{aligned} \quad (69)$$

In other words, given $\langle l, v \rangle$, the map ev_V evaluates l at v .

The coevaluation map coev_V associated to V is slightly trickier to describe. Let $\{e_1, \dots, e_n\}$ be a basis of V , and let $\{e_1^*, \dots, e_n^*\}$ be the corresponding dual basis of V^* . Then

$$\begin{aligned} \text{coev}_V : \mathbb{R} &\rightarrow V \otimes V^*, \\ \lambda &\mapsto \lambda \sum_{i=1}^n e_i \otimes e_i^*. \end{aligned} \quad (70)$$

It turns out that this map is independent of the choice of basis. One way to think of this coevaluation map is to recall that $V \otimes V^* \simeq \text{Hom}(V, V)$. Under this identification, coev_V maps the scalar λ to the linear endomorphism of V which is “multiplication by λ ”. (In terms of matrices, this is a diagonal matrix, with λ at every entry of the diagonal.)

Recall that as part of the monoidal structure on $\mathbf{Vect}_{\mathbb{R}}$ we have the left and right unitors

$$\text{lu}_V : \mathbf{1} \otimes V \xrightarrow{\cong} V \quad V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}} \quad (71)$$

$$\text{ru}_V : V \otimes \mathbf{1} \xrightarrow{\cong} V \quad V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}}. \quad (72)$$

The evaluation and coevaluation maps defined above satisfy the following equations:

$$\text{lu}_V^{-1} ; (\text{coev}_V \otimes \text{id}_V) ; \text{as}_{V, V^*, V} ; (\text{id}_V \otimes \text{ev}_V) ; \text{ru}_V = \text{id}_V \quad (73)$$

and

$$\text{ru}_{V^*}^{-1} ; (\text{id}_{V^*} \otimes \text{coev}_V) ; \text{as}_{V^*, V, V^*}^{-1} ; (\text{ev}_V \otimes \text{id}_{V^*}) ; \text{lu}_{V^*} = \text{id}_{V^*}. \quad (74)$$

Graded exercise H.8 (`VectSnakeEquations`)

Check (73) and (74) by direct calculation, assuming that $\textcolor{red}{V}$ is a finite-dimensional real vector space.

The equations (73) and (74) form the basis for the general notion of dualizability in a monoidal category.

Definition 27.17 (Dual object)

Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}} \rangle$ be a monoidal category, and let $X \in \text{Ob}_{\mathbf{C}}$. A *right dual object* of X is specified by:

Constituents

1. an object $X^{\vee} \in \text{Ob}_{\mathbf{C}}$;
2. a morphism $\text{ev}_X : X^{\vee} \otimes X \rightarrow \mathbf{1}$, called *evaluation*;
3. a morphism $\text{coev}_X : \mathbf{1} \rightarrow X \otimes X^{\vee}$, called *coevaluation*;

Conditions

1.

$$\text{lu}_X^{-1} \circ (\text{coev}_X \otimes \text{id}_X) \circ \text{as}_{X, X^{\vee}, X} \circ (\text{id}_X \otimes \text{ev}_X) \circ \text{ru}_X = \text{id}_X; \quad (75)$$

2.

$$\text{ru}_{X^{\vee}}^{-1} \circ (\text{id}_{X^{\vee}} \otimes \text{coev}_X) \circ \text{as}_{X^{\vee}, X, X^{\vee}}^{-1} \circ (\text{ev}_X \otimes \text{id}_{X^{\vee}}) \circ \text{lu}_{X^{\vee}} = \text{id}_{X^{\vee}}. \quad (76)$$

Definition 27.18

An object X in a monoidal category is called *right dualizable* if there exists, in the category, a right dual object of X .

Remark 27.19. There is an analogous definition of *left dual object* and *left dualizability*. One can show that when the monoidal category in question is symmetric, then left dual objects can be seen as right dual objects, and vice versa. In this case, we then speak simply of *dualizability*.

Definition 27.20 (Compact closed category)

A symmetric monoidal category is called *compact closed* if every object is dualizable.

Definition 27.21 (Dual morphism)

Let $f : X \rightarrow Y$ be a morphism in a monoidal category $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}} \rangle$ and suppose that X and Y have right duals X^{\vee} and Y^{\vee} , respectively. The *right dual* of f is the morphism $f^{\vee} : Y^{\vee} \rightarrow X^{\vee}$ given by the composition

$$\begin{array}{ccccccc}
 Y^{\vee} & \xrightarrow{\text{ru}_Y^{-1}} & Y^{\vee} \otimes \mathbf{1} & \xrightarrow{\text{id}_{Y^{\vee}} \otimes \text{coev}_X} & Y^{\vee} \otimes (X \otimes X^{\vee}) \\
 & \xrightarrow{\text{as}^{-1}} & (Y^{\vee} \otimes X) \otimes X^{\vee} & \xrightarrow{(\text{id}_{Y^{\vee}} \otimes f) \otimes \text{id}_{X^{\vee}}} & (Y^{\vee} \otimes Y) \otimes X^{\vee} \\
 & \xrightarrow{\text{ev}_Y \otimes \text{id}_{X^{\vee}}} & \mathbf{1} \otimes X^{\vee} & \xrightarrow{\text{lu}_{X^{\vee}}} & X^{\vee}
 \end{array} \quad (77)$$

Graded exercise H.9 (RelDualsTrace)

In this exercise we work with the category **Rel** of sets and relations, equipped with the symmetric monoidal structure where the monoidal product is the cartesian product of sets. The braiding is

$$\text{br}_{\mathbf{A}, \mathbf{B}} : \{\langle\langle x, y \rangle, \langle y', x' \rangle \rangle \in (\mathbf{A} \times \mathbf{B}) \times (\mathbf{B} \times \mathbf{A}) \mid x = x', y = y'\}. \quad (78)$$

This symmetric monoidal category is compact closed when we let the dual \mathbf{A}^\vee of any set \mathbf{A} be the set itself, $\mathbf{A}^\vee = \mathbf{A}$, and we define evaluation and co-evaluation by

$$\text{ev}_{\mathbf{A}} : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{1}, \quad \text{ev}_{\mathbf{A}} = \{\langle\langle x, y \rangle, \bullet \rangle \in (\mathbf{A} \times \mathbf{A}) \times \mathbf{1} \mid x = y\} \quad (79)$$

and

$$\text{coev}_{\mathbf{A}} : \mathbf{1} \rightarrow \mathbf{A} \times \mathbf{A}, \quad \text{coev}_{\mathbf{A}} = \{\langle \bullet, \langle x, y \rangle \rangle \in \mathbf{1} \times (\mathbf{A} \times \mathbf{A}) \mid x = y\} \quad (80)$$

respectively.

Your tasks:

1. Let $S : \mathbf{A} \rightarrow \mathbf{B}$ be a (generic) morphism in **Rel**. Compute the dual morphism $S^\vee : \mathbf{B}^\vee \rightarrow \mathbf{A}^\vee$.
2. Let $R : \mathbf{A} \times \mathbf{C} \rightarrow \mathbf{B} \times \mathbf{C}$ be a morphism in **Rel**. Show that the trace of R , given by the composition

$$\text{ru}_{\mathbf{A}}^{-1} \circ (\text{id}_{\mathbf{A}} \otimes \text{coev}_{\mathbf{C}}) \circ \text{as}_{\mathbf{A}, \mathbf{C}, \mathbf{C}}^{-1} \circ (R \otimes \text{id}_{\mathbf{C}}) \circ \text{as}_{\mathbf{B}, \mathbf{C}, \mathbf{C}} \circ (\text{id}_{\mathbf{B}} \otimes \text{br}_{\mathbf{A}, \mathbf{A}}) \circ (\text{id}_{\mathbf{B}} \otimes \text{ev}_{\mathbf{C}}) \circ \text{ru}_{\mathbf{B}} \quad (81)$$

is equal to the relation

$$\{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid \exists z \in \mathbf{C} : \langle\langle x, z \rangle, \langle y, z \rangle \rangle \in R\}. \quad (82)$$

27.5. Canonical trace

Definition 27.22 (Trace of an endomorphism)

Let $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ be a symmetric monoidal category. Let $X \in \text{Ob}_{\mathbf{C}}$ be dualizable and let

$$f : X \rightarrow X. \quad (83)$$

The *trace* of f is the morphism

$$\text{Tr}(f) : \mathbf{1} \rightarrow \mathbf{1} \quad (84)$$

defined by

$$\begin{array}{ccccccc} \mathbf{1} & \xrightarrow{\text{coev}_X} & X \otimes X^{\vee} & \xrightarrow{f \otimes \text{id}_{X^{\vee}}} & X \otimes X^{\vee} & \xrightarrow{\text{br}} & X^{\vee} \otimes X \xrightarrow{\text{ev}_X} \mathbf{1} \\ & & & & & & \\ & & & & & & \end{array} \quad (85)$$

Graded exercise H.10 (`LinearAlgebraTrace`)

Let \mathbf{C} be the category of finite-dimensional real vector spaces and \mathbb{R} -linear maps. We have seen that this category is symmetric monoidal when equipped with the usual tensor product as monoidal product. Furthermore, in Section 27.4 we saw that every object in this category is dualizable.

Fix a finite-dimensional real vector space V , and let $\{e_1, \dots, e_n\}$ be a basis of it. We saw that a choice of dual object for V is given by $V^* = \text{Hom}(V, \mathbb{R})$, together with the evaluation map

$$\begin{aligned} \text{ev}_V : V^* \otimes V &\rightarrow \mathbb{R}, \\ \langle l, v \rangle &\mapsto l(v). \end{aligned} \quad (86)$$

and the co-evaluation map

$$\begin{aligned} \text{coev}_V : \mathbb{R} &\rightarrow V \otimes V^*, \\ \lambda &\mapsto \lambda \sum_{i=1}^n e_i \otimes e_i^*. \end{aligned} \quad (87)$$

where $\{e_1^*, \dots, e_n^*\}$ is the basis dual to the one we chose for V .

Let $f : V \rightarrow V$ be a linear endomorphism – that is, $f \in \text{Hom}_{\mathbf{C}}(V, V)$. Compute the trace $\text{Tr}(f) \in \text{Hom}_{\mathbf{C}}(\mathbb{R}, \mathbb{R})$ of f according to Def. 27.22, and explain why it is the linear map “multiplication by the trace of f ”, where “trace” in this latter phrase is the usual notion that we know from linear algebra.

Graded exercise H.11 (`DPSnakeTrace`)

In this exercise we work with the category \mathbf{DP} of posets and design problems, equipped with the symmetric monoidal structure where the monoidal product is the cartesian product of posets. The braiding

$$\text{br}_{\mathbf{P}, \mathbf{Q}} : \mathbf{P} \times \mathbf{Q} \rightarrowtail \mathbf{Q} \times \mathbf{P}$$

is defined by

$$\text{br}_{\mathbf{P}, \mathbf{Q}}((p_1, q_1)^*, (q_2, p_2)) := (p_1 \leq_{\mathbf{P}} p_2) \wedge (q_1 \leq_{\mathbf{Q}} q_2). \quad (88)$$

In the following you are free to use the identification

$$(\mathbf{P} \times \mathbf{Q})^{\text{op}} = \mathbf{P}^{\text{op}} \times \mathbf{Q}^{\text{op}} \quad (89)$$

for any posets \mathbf{P}, \mathbf{Q} . Also, recall that $(\mathbf{P}^{\text{op}})^{\text{op}} = \mathbf{P}$.

Let us define the following duality data:

- ▷ $\mathbf{P}^{\vee} := \mathbf{P}^{\text{op}}$
- ▷ $\text{ev}_{\mathbf{P}} : (\mathbf{P}^{\text{op}} \times \mathbf{P})^{\text{op}} \times \{\bullet\} \rightarrow \mathbf{Bool}, \quad \langle \langle x^*, y \rangle^*, \bullet \rangle \mapsto y \leq_{\mathbf{P}} x$
- ▷ $\text{coev}_{\mathbf{P}} : \{\bullet\}^{\text{op}} \times (\mathbf{P} \times \mathbf{P}^{\text{op}}) \rightarrow \mathbf{Bool}, \quad \langle \bullet, \langle x, y^* \rangle \rangle \mapsto y \leq_{\mathbf{P}} x$

Your tasks:

1. Guess the definitions of the associator as and the unitors lu, ru for the monoidal category \mathbf{DP} , check that each has the correct type, and justify why each of them does indeed define a morphism in the category \mathbf{DP} .
2. Guess the definitions of lu^{-1} and ru^{-1} and check for one of them that it does indeed define the inverse morphism.
3. Check that $\text{ev}_{\mathbf{P}}$ and $\text{coev}_{\mathbf{P}}$, as defined above, are morphisms in \mathbf{DP} .
4. For an arbitrary poset \mathbf{P} and the duality data given above, prove that this snake equation

$$\text{lu}_{\mathbf{P}}^{-1} ; (\text{coev}_{\mathbf{P}} \otimes \text{id}_{\mathbf{P}}) ; \text{as}_{\mathbf{P}, \mathbf{P}^{\text{op}}, \mathbf{P}} ; (\text{id}_{\mathbf{P}} \otimes \text{ev}_{\mathbf{P}}) ; \text{ru}_{\mathbf{P}} = \text{id}_{\mathbf{P}} \quad (90)$$

holds.

Definition 27.23 (Trace of a generalized endomorphism)

Let $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru}, \text{br} \rangle$ be a symmetric monoidal category. Let $X \in \text{Ob}_{\mathbf{C}}$ be dualizable and let

$$f : (Y \otimes X) \rightarrow (Z \otimes X). \quad (91)$$

The *trace over X* of f is the morphism

$$\text{Tr}_{Y, Z}^X(f) : Y \rightarrow Z \quad (92)$$

defined by

$$Y \xrightarrow{\text{ru}_Y^{-1}} Y \otimes \mathbf{1} \xrightarrow{\text{id}_Y \otimes \text{coev}_Y} Y \otimes (X \otimes X^{\vee}) \xrightarrow{\text{as}^{-1}} (Y \otimes X) \otimes X^{\vee} \xrightarrow{f \otimes \text{id}_{X^{\vee}}} (Z \otimes X) \otimes X^{\vee} \quad (93)$$

$$\xrightarrow{\text{as}} Z \otimes (X \otimes X^{\vee}) \xrightarrow{\text{id}_Z \otimes \text{br}} Z \otimes (X^{\vee} \otimes X) \xrightarrow{\text{id}_Z \otimes \text{ev}_X} Z \otimes \mathbf{1} \xrightarrow{\text{ru}_Z} Z$$

Solutions to selected exercises

Solution of Exercise 50. To prove the statement we first check that the stacking operations satisfy Def. 25.41, we then show that they are compatible, and finally show associativity. The stacking operation on objects was already checked for $\langle \text{Set} \rangle$. The stacking operation on morphisms clearly returns a valid relation. Furthermore, compatibility is satisfied:

$$\frac{R : \langle A_1, \dots, A_m \rangle \rightarrow_{\langle \text{Rel} \rangle} \langle B_1, \dots, B_n \rangle \quad S : \langle C_1, \dots, C_o \rangle \rightarrow_{\langle \text{Rel} \rangle} \langle D_1, \dots, D_p \rangle}{R \otimes S : \langle A_1, \dots, A_m \rangle \circledast \langle C_1, \dots, C_o \rangle \rightarrow_{\langle \text{Rel} \rangle} \langle B_1, \dots, B_n \rangle \circledast \langle D_1, \dots, D_p \rangle} \quad (94)$$

Finally, associativity for the operation on objects was already shown for $\langle \text{Set} \rangle$.

Solution of Exercise 51.

PART I.CO-DESIGN



28. Design	405
29. Monotone Co-Design Theory	413
30. Feasibility	433
31. Lattices	439
32. Lattice structure of DPs	449
33. Constructing design problems	459

The *Sechseläuten* is a traditional spring holiday in the Zurich, Switzerland, usually happening on the 3rd monday of April. The old city guilds meet in the city center for a parade, climax of which is the burning of the “Böögg”, a snowman prepared with explosives, considered a weather oracle for the summer.



28. Design

This chapter introduces basic concepts of engineering design, and describes what are the design queries we want to answer.

28.1 What is “design”?	406
28.2 What is “co-design”?	407
28.3 Formal engineering design	409
28.4 Queries in design	411

Switzerland produces timepieces since the 14th century. Indeed, a “Swiss made” watch is very unique: according to official rules, the mechanics, casing, and inspection of a watch must be carried out in Switzerland to earn the hallmark. Today, Switzerland is the world’s largest watch exporter, counting over 60,000 employees and exporting over \$ 13.7 billion worth products.

28.1. What is “design”?

We take a broad view of what it means to “design”, that is not limited to engineering. Citing at length Hebert Simon’s* *The sciences of the artificial* ([25], Chapter 5):

Engineers are not the only professional designers. Everyone designs who devises courses of action aimed at changing existing situations into preferred ones. The intellectual activity that produces material artifacts is no different fundamentally from the one that prescribes remedies for a sick patient or the one that devises a new sales plan for a company or a social welfare policy for a state. Design, so construed, is the core of all professional training; it is the principal mark that distinguishes the professions from the sciences. Schools of engineering, as well as schools of architecture, business, education, law, and medicine, are all centrally concerned with the process of design.

The metaphors used in the book are biased towards engineering. It is easy for everybody to imagine creating a physical machine out of simple components, and what choices and trade-offs we must deal with. Furthermore, it is easy to imagine what is the boundary between the machine and the world, that is, to delimit the design space.

Yet the theory to be discussed is applicable to other disciplines, if one takes a more abstract view of what is a system and a component. For example, in urban planning, the components of a city are roads, sewers, residential areas, *etc.* In other disciplines, “components” can be logical instead of physical. For example, an economist might ask how to design an incentive scheme such that such scheme (a “component”) will move the system to a more desirable set of states.

* Hebert A. Simon (1916-2001). Winner of the 1978 Nobel Prize in Economics.

28.2. What is “co-design”?

The word “co-design” is not a new one. In this book, we will use a meaning that incorporates and extends the existing meaning.

We take the “Co” in “co-design” to have four meanings:

1. “co” for “compositional”;
2. “co” for “collaborative”;
3. “co” for “computational”;
4. “co” for “continuous”.

These meanings together describe the aspects of modern engineering design.

“Co” for “compositional”

The first meaning has to do with composition:

co-design = design everything together

We use the word “co-design” to refer to any decision procedure that has to do with making simultaneous choices about the components of a system to achieve system-level goals. This includes the choice of components, the interconnection of components, and the configuration of components. We will see that in most cases, choices that are made at the level of components without looking at the entire system are doomed to be suboptimal. Slightly modifying Haiken’s quote in Section 1.2, we choose this as our slogan:

A system is composed of components;
a component is something you understand **how to design**.

“Co” for “collaborative”

In a second broad meaning, “co-” stands for “collaborative”:

co-design = design everything, together

There are two types of collaborations. First, there is the collaboration between human and machine, in the definition and solution of design problems. Second, and most importantly, is the collaboration among different experts or teams of experts in the design process.

The typical situation is that the system design is suboptimal because every expert only knows one component and there are rigid interfaces/contracts designed early on. The problem here is sharing of knowledge across teams, specifically, knowledge about the design of systems.

In this case, this is the slogan:

«A system is composed of components;
a component is something that **somebody** understands **how to design**.»

There is a tight link between the “composition” and “collaboration” aspects.

As Conway[†] first observed for software systems:

«Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.»

[†] John Horton Conway (1937–2020) was a mathematician. Probably the most popular idea of his was the invention of the Game of Life, which inspired countless works on cellular automata. We remember him for the discovery of the *surreal numbers*, which should be just called *numbers*, as they contain all other ordered fields.

This “mirroring” hypothesis between system and organization was explored formally and found to hold [18]. The ultimate reason is that “the organization’s governance structures, problem-solving routines and communication patterns constrain the space in which it searches for new solutions”. This appears to be true for generic systems in addition to software.

In the end, civilization is about dividing up the work, and so we must choose where one’s work ends and the other’s work begins. But we need to keep talking if we want that everything works together.

“Co” for “computational”

The third meaning of “co-” in “co-design” will be **computational**. It is the age of machines, and we need machines to understand what we are doing.

Therefore, we strive to create not only a qualitative modeling for co-design, but also a formal and quantitative description that will be suitable for setting up an optimization problem that can be solved to obtain an optimal design.

Our slogan becomes:

«A system is composed of components;
a component is something that **somebody** understands **how to design well enough to teach a computer.** »

“Co” for “continuous”

The fourth meaning of “co-” is **continuous**. We look at designs not as something that exists as a single decision in time, but rather as something that continuous to exist and evolve, independently on the designer.

28.3. Basic concepts of formal engineering design

Later, all these concepts will find a formal definition in the language of category theory.

Functionality and functional requirements You are an engineer in front of an empty whiteboard, ready to start designing the next product. The first question to ask is: What is the *purpose* of the product to be designed? The purpose of the product is expressed by the *functional requirements*, sometimes called *functional specifications, desired behavior, objectives, or simply function*.

Unfortunately, the word “function” conflicts with the mathematical concept. Therefore, we will talk about *functionality*. Moreover, we will never use the word “function”, and instead use *map* to denote the mathematical concept.

Example 28.1. These are a few examples of functional requirements:

- ▷ A car must be able to transport at least $n \geq 4$ passengers.
- ▷ A battery must store at least 100 kJ of energy.
- ▷ An autonomous vehicle should reach at least 20 mph while guaranteeing safety.

Resources and resource constraints We call *resources* what we need to pay to realize the given functionality. In some contexts, these are better called *costs, or dependencies*.

Example 28.2. These are a few examples of resource constraints:

- ▷ A car should not cost more than 15,000 USD.
- ▷ A battery should not weigh more than 1 kg.
- ▷ A process should not take more than 10 s.

Duality of functionality and resources There is an interesting duality between functionality and resources. When designing systems, one is given functional requirements, as a *lower bound* on the functionality to provide, and one is given resource constraints, which are an *upper bound* on the resources to use.

As far as design objectives go, most can be understood as either *minimize resource usage* or *maximize functionality provided*.

This duality between functionality and resources will be at the center of our formalization.

Non-functional requirements Functionality and resources do not cover all the requirements—there is, for example, a large class of *non-functional requirements* [deweck11] such as the extensibility and the maintainability of the system. Nevertheless, functionality and resources can express most of the requirements which can be quantitatively evaluated, at least prior to designing, assembling, and testing the entire system.

Implementation space The *implementation space* or *design space* is the set of all possible design choices that could be chosen; by *implementation*, or the word “design”, used as a noun, we mean one particular set of choices. The implementation space \mathbf{I} is the set over which we are optimizing; an implementation $i \in \mathbf{I}$ is a particular point in that set (Fig. 1).

The interconnection between functionality, resources, and implementation spaces is as follows. We will assume that, given one implementation, we can evaluate it to know the functionality and the resources spaces (Fig. 2).

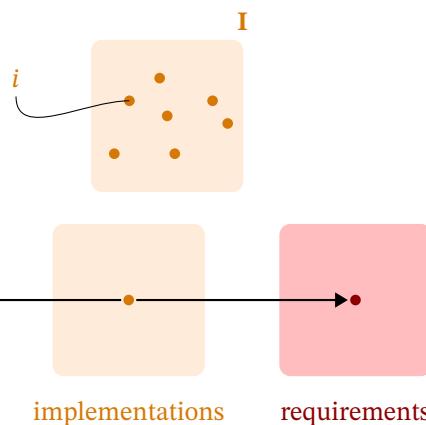


Figure 1.: An *implementation* *i* is a particular point in the implementation space **I**.

Figure 2.: Evaluation of specific implementations to get functionality and resources spaces.

Functional Interfaces and interconnection Components are *interconnected* to create a system. This implies that we have defined the *interfaces* of components, which have the dual function of delimiting when one component ends and another begins, and also to describe exactly what is the nature of their interaction.

We will develop a formalism in which the functionality and resources are the interfaces used for interconnection: two components are connected if the resources required by the first correspond to the functionality provided by the second.

Abstraction By *abstraction*, we mean that it is possible to “zoom out”, in the sense that a system of components can be seen as a component itself, which can be part of larger systems.

Compositionality A *compositional* property is a property that is preserved by interconnection and abstraction; assuming each component in a system satisfies that property, also the system as a whole satisfies the property.

Example 28.3. One can compose two electronic circuits by joining their terminals to obtain another electronic circuit. We would say that the property of being an electronic circuit is compositional.

28.4. Queries in design

Suppose that we have a model with a functionality space \mathbf{F} , a requirements space \mathbf{R} , and an implementation space \mathbf{I} .

There are several queries we can ask of a model. They all look at the same phenomenon from different angles, so they look similar; however the computational cost of answering each one might be very different.

The first kind of query is one that asks if the design is feasible when fixed all variables.

Problem (Feasibility problem). Given a triplet of implementation $i \in \mathbf{I}$, functionality $f \in \mathbf{F}$, requirements $r \in \mathbf{R}$, determine if the design is feasible.

The second type of query is that which fixes the boundary conditions of functionality and requirements, and asks to find a solution.

Problem (Find implementation). Given a pair of minimal requested functionality $f \in \mathbf{F}$ and maximum allowed requirements $r \in \mathbf{R}$, determine if there is an implementation $i \in \mathbf{I}$ that is feasible.

A different type of query is the one in which the design objective (the functionality) is fixed, and we ask what are the least resources necessary.

Problem (FixFunMinRes). Given a certain functionality $f \in \mathbf{F}$, find the set of “minimal” resources in \mathbf{R} that are needed to realize it (along with the implementations), or provide a proof that there are none (a certificate of infeasibility).

Dually, we can ask, fixed the resources available, what are the functionalities that can be provided.

Problem (FixResMaxFun). Given a certain requirement $r \in \mathbf{R}$, find the set of “maximal” functionalities in that can realize it (along with the implementations), or provide a proof that there are none (a certificate of infeasibility).

It is very natural to talk about the “minimal” requirements and “maximal” functionalities; after all, we always want to minimize costs and maximize performance. In the next chapter we start to put more mathematical scaffolding in place, starting from defining functionality and requirements as posets.



29. Monotone Co-Design Theory

This chapter introduces *Monotone Co-Design Theory*, a formalization for computational design theory. It is a compositional theory of which the primitive elements are *design problems* (DPI), formalized as relations among functionality, resources, and implementations.

We show that DPIs can capture design problems across diverse fields.

29.1	DPIs	414
29.2	Examples	417
29.3	Queries	423
29.4	Co-design problems	425
29.5	The semicategory DPI	429
29.6	Sum and intersection of DPIs	431

29.1. Design Problems with Implementation

We start by defining a “design problem with implementation”, which is a tuple of “functionality space”, “implementation space”, and “resources space”, together with two maps that describe the feasibility relations between these three spaces (Fig. 1).

Definition 29.1 (Design problem with implementation)

A *design problem with implementation* (DPI) is a tuple

$$\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle, \quad (1)$$

where:

- ▷ **F** is a poset, called *functionality* space;
 - ▷ **R** is a poset, called *requirements* space;
 - ▷ **I** is a set, called *implementation* space;
 - ▷ the map **prov** : **I** → **F** maps an implementation to the functionality it provides;
 - ▷ the map **req** : **I** → **R** maps an implementation to the resources it requires.

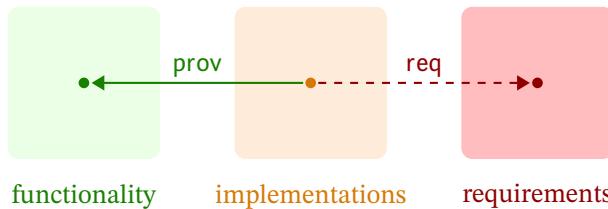


Figure 1.

Example 29.2 (Motor design). Suppose we need to choose a motor for a robot from a given set. The **functionality** of a motor could be parametrized by **torque** and **speed**. The **resources** to consider could include the **cost [USD]**, the **mass [g]**, the **input voltage [V]**, and the **input current [A]**. The map $\text{prov} : \mathbf{I} \rightarrow \mathbf{F}$ assigns to each motor its functionality, and the map $\text{req} : \mathbf{I} \rightarrow \mathbf{R}$ assigns to each motor the resources it needs (Fig. 2).

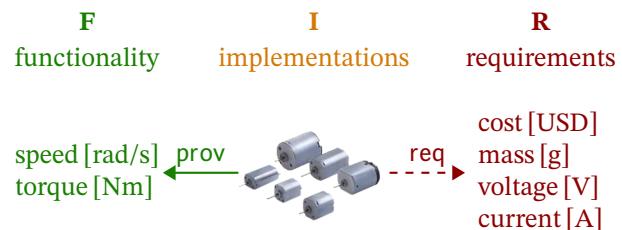


Figure 2.

Example 29.3 (Chassis design). Suppose we need to choose a chassis for a robot. The implementation space **I** (Fig. 3) could be the set of all chassis that could ever be designed (in case of a theoretical analysis), or just the set of chassis available in the catalogue at hand (in case of a practical design decision). The functionality of a chassis could be formalized as “the ability to transport a certain **payload [g]**” and “at a given **speed [m/s]**”. More refined functional requirements would include maneuverability, the cargo volume, *etc.* The resources to consider could be the **cost [USD]** of the chassis; the total mass; and, for each motor to be placed in the chassis, the required **speed [rad/s]** and **torque [Nm]**.



Figure 3.

Example 29.4. We revisit the leading example of Section 34.1 with the newly introduced co-design perspective. Consider a list of electrical motors as in Table 29.1.

Table 29.1.: A simplified catalogue of motors.

Motor ID	Company	Torque [kg · cm]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	0.18	60.0	2.34	19.95
1206	SOYO	0.95	140.0	3.00	19.95
1207	SOYO	0.65	130.0	2.07	12.95
2267	SOYO	3.7	285.0	4.76	16.95
2279	Sanyo Denki	1.9	165.0	5.40	164.95
1478	SOYO	19.0	1,000	8.96	49.95
2299	Sanyo Denki	2.2	150.0	5.90	59.95

We can think of this as a catalogue of electric motors $\langle I_{EM}, \text{prov}_{EM}, \text{req}_{EM} \rangle$. In particular, the set of implementations collects all the motor models, which we can specify using the motor IDs:

$$I_{EM} = \{1204, 1206, 1207, 2267, 2279, 1478, 2299\}. \quad (2)$$

We now have to think about **resources** and **functionalities**. Each motor **requires** some **weight** (in g), **power** (in W), and has some **cost** (in USD), and **provides** some **torque** (in kg · cm). Thus, we can identify

$$F = \mathbb{R}^{\text{kg}\cdot\text{cm}}, \quad R = \mathbb{R}^g \times \mathbb{R}^W \times \mathbb{R}^{\text{USD}}, \quad (3)$$

by considering the units as discussed in Section 15.3. The correspondences are given by the details in Table 29.1. For instance, we have

$$\text{prov}_{EM}(1204) = 0.18 \text{ kg} \cdot \text{cm}, \quad (4)$$

$$\text{req}_{EM}(1204) = \langle 60 \text{ g}, 2.34 \text{ W}, 19.95 \text{ USD} \rangle. \quad (5)$$

Graphical notation

A graphical notation will help reasoning about composition. A DPI is represented as a box with n_f green edges and n_r red edges (Fig. 4).

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

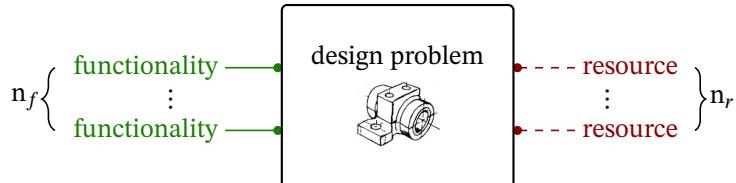


Figure 4.

This means that the functionality and resources spaces can be factorized in n_f and n_r components:

$$\mathbf{F} = \prod_{i=1}^{n_f} \text{pr}_i \mathbf{F}_i, \quad \mathbf{R} = \prod_{j=1}^{n_r} \text{pr}_j \mathbf{R}_j, \quad (6)$$

where “ pr_i ” represents the projection to the i -th component. If there are no green (respectively, red) edges, then n_f (respectively, n_r) is zero, and \mathbf{F} (respectively, \mathbf{R}) is equal to $\mathbf{1} = \{\langle \rangle\}$, the set containing one element, the empty tuple $\langle \rangle$.

These *co-design diagrams* are not to be confused with signal flow diagrams, in which the boxes represent oriented systems and the edges represent signals.

29.2. Examples

We now present a list of design problems for different disciplines, to showcase the universality of the approach.

Mechatronics

Many mechanisms can be readily modeled as relations between a provided functionality and required resources.

Example 29.5. A gearbox (Fig. 5) provides a certain **output torque** τ_o and **speed** ω_o , given a certain **input torque** τ_i and **speed** ω_i . For an ideal gearbox with a reduction ratio $r \in \mathbb{Q}_+$ and efficiency ratio γ , $0 < \gamma < 1$, the constraints among those quantities are $\omega_i \geq r \omega_o$ and $\tau_i \omega_i \geq \gamma \tau_o \omega_o$. With this simple model, the set of implementations are given by the possible values of reduction and efficiency ratio.

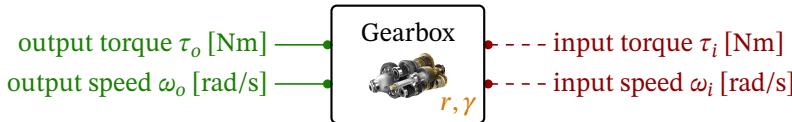


Figure 5.

Example 29.6. Propellers (Fig. 6) generate **thrust** given a certain **torque** and **speed**.

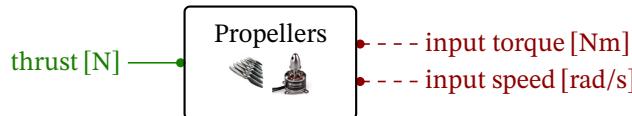


Figure 6.

Example 29.7. A four-bar *crank-rocker* (Fig. 7) converts **rotational motion** into a **rocking motion**. The parametrization depends on the length of the four linkages.

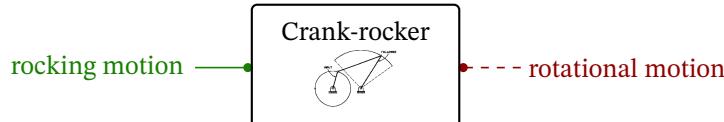


Figure 7.

Geometrical constraints

Geometrical constraints are examples of constraints that are easily recognized as monotone, but possibly hard to write down in closed form.

Example 29.8 (Bin packing). Suppose that each internal component occupies a volume bounded by a parallelepiped, and that we must choose the minimal enclosure in which to place all components (Fig. 8). What is the minimal size of the enclosure? This is a variation of the *bin packing* problem, which is in NP for both 2D and 3D [17]. It is easy to see that the problem is monotone, by noticing that, if one the components shapes increases, then the size of the enclosure cannot shrink. The implementations, in this case, are the configurations which one can choose to place all components in the container (one of the possible configurations is shown in the picture).

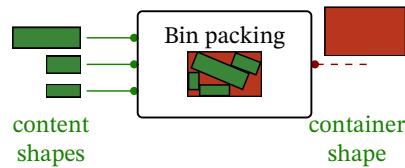


Figure 8.

Inference

Many inference problems have a monotone formalization, taking the **accuracy** or **robustness** as functionality, and **computation** or **sensing** as resources. Typically, these bounds are known in a closed form only for restricted classes of systems, such as the linear/Gaussian setting.

Example 29.9 (SLAM). One issue with particle-filter-based estimation procedures, such as the ones used in the popular GMapping [9] suite, is that the filter might diverge if there aren't enough particles. Although the relation might be hard to characterize, there is a monotone relation between the **robustness** (1 - probability of failure), the **accuracy**, and the **number of particles** (Fig. 9). Here, the implementation space contains the other choices of parameters for the filter: fixed the number of particles, by changing the tuning of the filter, we can explore the trade-off of accuracy and robustness.

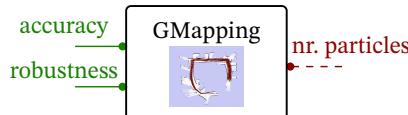


Figure 9.

Example 29.10 (Stereo reconstruction). Progressive reconstruction systems ([16]), which start with a coarse approximation of the solution that is progressively refined, are described by a smooth relation between the **resolution** and the **latency** to obtain the answer (Fig. 10). A similar relation characterizes any anytime algorithms in other domains, such as robot motion planning.

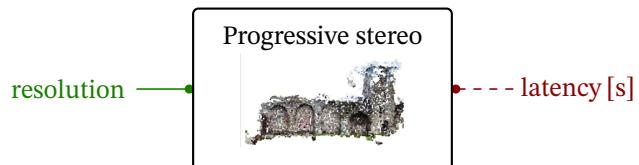


Figure 10.

Example 29.11. The empirical characterization of the monotone relation between the **accuracy of a visual SLAM solution** and the **power consumption** is the goal of recent work by Davison and colleagues [21, 32].

Communication

Example 29.12 (Transducers). Any type of “transducer” that bridges between different mediums can be modeled as a DP. For example, an access point (Fig. 11) provides the **wireless access** functionality, and requires that the infrastructure provides the **Ethernet access** resource.

Example 29.13 (Wireless link). The basic functionality of a wireless link is to provide a certain **bandwidth** (Fig. 12). Further refinements could include



Figure 11.

bounds on the latency or the probability that a packet drop is dropped. Given the established convention about the preference relations for functionality, in which a *lower* functionality is “easier” to achieve, one needs to choose “*minus the latency*” and “*minus the packet drop probability*” for them to count as functionality. As for the resources, apart from the **transmission power [W]**, one should consider at least **the spectrum occupation**, which could be described as an interval $[f_0, f_1]$ of the frequency axis $\mathbb{R}^{[\text{Hz}]}$. Thus, the resources space is $\mathbf{R} = \mathbb{R}^{[\text{W}]} \times \text{intervals}(\mathbb{R}^{[\text{Hz}]})$.

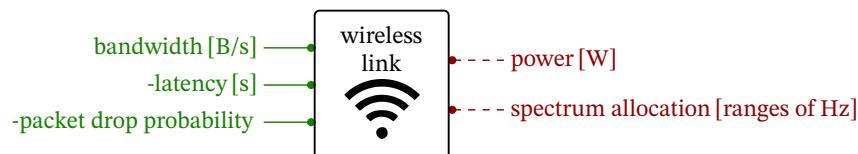


Figure 12.

Multi-robot systems

In a multi-robot system there is always a trade-off between the number of robots and the capabilities of the single robot.

Example 29.14. Suppose we need to create a swarm of agents whose functionality is **to sweep an area**. If the functionality is fixed, one expects a three-way trade-off between the three resources: number of agents, the speed of a single agent, and the execution time. For example, if the time available decreases, we have to increase either the speed of an agent or the number of agents (Fig. 13b).

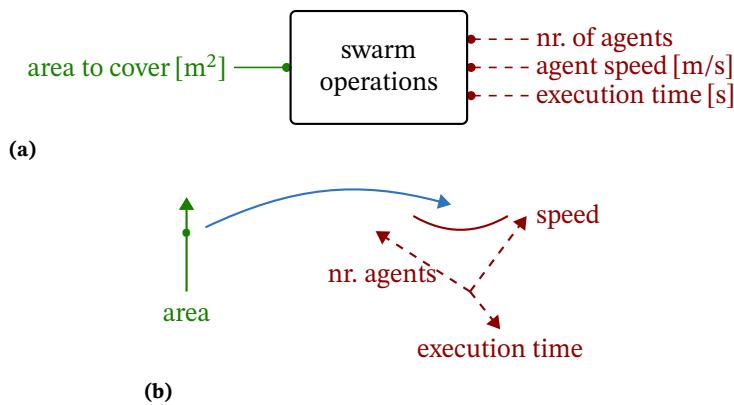


Figure 13.

Computation graphs

The trivial model of a CPU is as a device that provides **computation**, measured in **flops**, and requires **power [W]**. Clearly there is a monotone relation between the two.

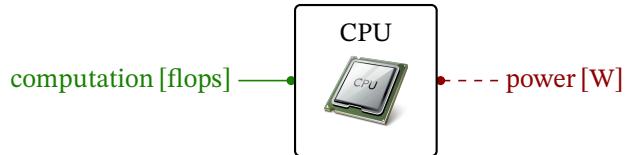


Figure 14.

A similar monotone relation between application requirements and computation resources holds in a much more general setting, where both application and computation resources are represented by graphs. This will be an example of a monotone relation between nontrivial partial orders.

In the Static Data Flow (SDF) model of computation [29, 14, Chapter 3], the application is represented as a graph of procedures that need to be allocated on a network of processors.

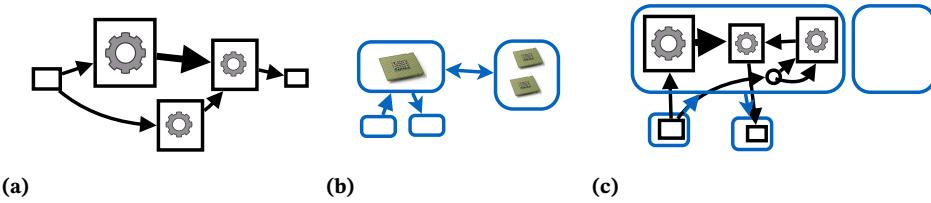


Figure 15.

Define the *application graph* (sometimes called “computation graph”) as a graph where each node is a procedure (or “actor”) and each edge is a message that needs to be passed between procedures. Each node is labeled by the number of ops necessary to run the procedure. Each edge is labeled by the size of the message. There is a partial order \leq on application graphs. In this order, it holds that $A_1 \leq A_2$ if the application graph A_2 needs more computation or bandwidth for its execution than A_1 . Formally, it holds that $A_1 \leq A_2$ if there is a homomorphism $\varphi : A_1 \Rightarrow A_2$; and, for each node $n \in A_1$, the node $\varphi(n)$ has equal or larger computational requirements than n ; and for each edge (n_1, n_2) in A_2 , the edge $(\varphi(n_1), \varphi(n_2))$ has equal or larger message size.

Define a *resource graph* as a graph where each node represents a processor, and each edge represents a network link. Each node is labeled by the processor capacity [flops]. Each edge is labeled by latency [s] and bandwidth [B/s]. There is a partial order on resources graph as well: it holds that $R_1 \leq R_2$ if the resource graph R_2 has more computation or network available than R_1 . The definition is similar to the case of the application graph: there must exist a graph homomorphism $\varphi : R_1 \Rightarrow R_2$ and the corresponding nodes (edges) of R_2 must have larger or equal computation (bandwidth) than those of R_1 .

Given an application graph A and a resource graph R , a typical resource allocation problem consists in choosing in which processor each actor must be scheduled to maximize the throughput T [Hz]. This is equivalent to the problem of finding a graph homomorphism $\Psi : A \Rightarrow R$. Let T^* be the optimal throughput, and write it as a function of the two graphs:

$$T^* = T^*(A, R). \quad (7)$$

Then the optimal throughput T^* is decreasing in A (a more computationally demanding application graph decreases the throughput) and increasing in R (more available computation/bandwidth increase the throughput).

Therefore, we can formalize this as a design problem where the two functionalities are the throughput T [Hz] and the application graph A , and the resource

graph R is the resource.

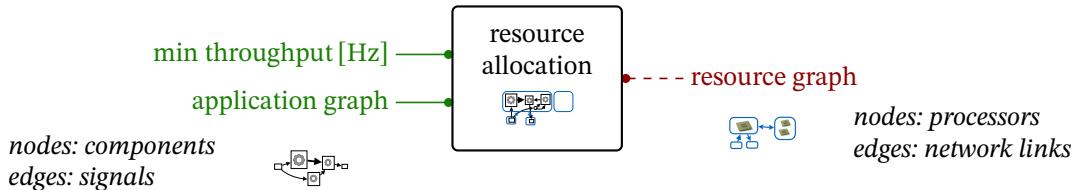


Figure 16.

Example 29.15. Svorenova *et al.* [30] consider a joint sensor scheduling and control synthesis problem, in which a robot can decide to not perform sensing to save power, given performance objectives on the probability of reaching the target and the probability of collision. The method outputs a Pareto frontier of all possible operating points. This can be cast as a design problem with functionality equal to the **probability of reaching the target** and (the inverse of) **the collision probability**, and with resources equal to the **actuation power**, **sensing power**, and **sensor accuracy**.

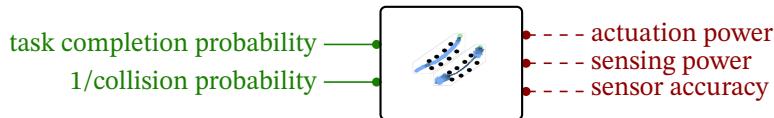


Figure 17.

Example 29.16. Nardi *et al.* [32] describe a benchmarking system for visual SLAM that provides the empirical characterization of the monotone relation between **the accuracy** of the visual SLAM solution, the **throughput** [**frames/s**] and **the energy for computation** [**J/frame**]. The implementation space is the product of algorithmic parameters, compiler flags, and architecture choices, such as the number of GPU cores active. This is an example of a design problem whose functionality-resources map needs to be experimentally evaluated.



Figure 18.

Other examples in minimal robotics

Many works have sought to find “minimal” designs for robots, and can be understood as characterizing the relation between the poset of **tasks** and the poset of physical resources, which is the product of **sensing**, **actuation**, and **computation** resources, plus other non-physical resources, such as **prior knowledge** (Fig. 19). Given a task, there is a minimal antichain in the resources poset that describes the possible trade-offs (for instance, compensating lousier sensors with more computation).

The poset structure arises naturally: for example, in the *sensor lattice* [13], a sensor dominates another if it induces a finer partition of the state space. Similar dominance relations can be defined for actuation and computation. O’Kane and LaValle [22] define a robot as a union of “robotic primitives”, where each primitive is an abstraction for a set of sensors, actuators, and control strategies that can be used together (for instance, a compass plus a contact sensor allow to “drive North until a wall is hit”). The effect of each primitive is modeled as an operator on the



Figure 19.

robot's information space. It is possible to work out what are the minimal combinations of robotic primitives (minimal antichain) that are sufficient to perform a task (for instance, global localization), and describe a dominance relation (partial order) of primitives. Other works have focused on minimizing the complexity of the controller. Egerstedt [4] studies the relation between the **complexity of the environment** and a notion of **minimum description length of control strategies**, which can be taken as a proxy for the computation necessary to perform the task. Soatto [26] studies the relation between the **performance of a visual task**, and the **minimal representation** that is needed to perform that task.

29.3. Queries

A DPI is a model to which we can associate a family of optimization problems. While in previous examples we covered the problem “feasibility”, we still miss `FixFunMinRes`, `FixResMaxFun`, and `FeasibleImp`.

The first can be translated to “Given a lower bound on the functionality f , what are the implementations that have minimal resource usage?” (Fig. 20).

Problem (FixFunMinRes). Given $f \in \mathbf{F}$, find the implementations in \mathbf{I} that realize the functionality f (or higher) with minimal resources, or provide a proof that there are none:

$$\left\{ \begin{array}{ll} \text{using} & i \in \mathbf{I}, \\ \text{Min}_{\leq_R} & r, \\ \text{s.t.} & r = \text{req}(i), \\ & f \leq_F \text{prov}(i). \end{array} \right. \quad (8)$$

Remark 29.17 (Minimal vs least solutions). Note the use of Min_{\leq_R} in (8), which indicates the set of minimal (non-dominated) elements according to \leq_R , rather than \min_{\leq_R} , which would presume the existence of the least element. In all problems in this paper, the goal is to find the optimal trade-off of resources (“Pareto front”). So, for each f , we expect to find an antichain $R \in \text{Anti } \mathbf{R}$. We will see that this formalization allows an elegant way to treat multi-objective optimization problems. The algorithm to be developed will directly solve for the set R , without resorting to techniques such as *scalarization*, and therefore is able to work with arbitrary posets, possibly discrete.

In an entirely symmetric fashion, we could fix an upper bound on the resource usage, and then maximize the functionality provided (Fig. 21). The formulation is entirely dual, in the sense that it is obtained from (8) by swapping Min with Max, \mathbf{F} with \mathbf{R} , and prov with req .

Problem (FixResMaxFun). Given $r \in \mathbf{R}$, find the implementations in \mathbf{I} that require r (or lower) and provide the maximal functionality, or provide a proof that there are none:

$$\left\{ \begin{array}{ll} \text{using} & i \in \mathbf{I}, \\ \text{Max}_{\leq_F} & f, \\ \text{s.t.} & f = \text{prov}(i), \\ & r \geq_R \text{req}(i). \end{array} \right. \quad (9)$$

Another type of query is: “Given a lower bound on the functionality f and an upper bound on the costs r , what are the feasible implementations?

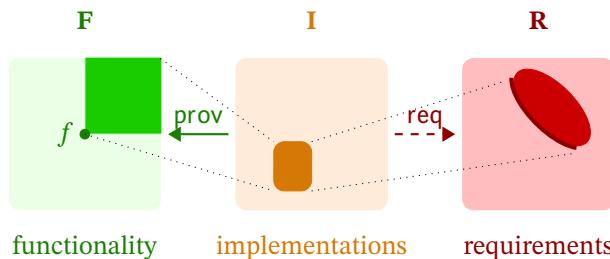


Figure 20.

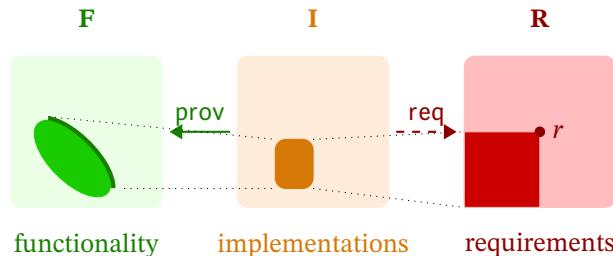


Figure 21.

Problem (FeasibleImp). Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find the implementations in \mathbf{I} that requires r (or lower) and provide f (or higher)

$$\begin{cases} \text{using } i \in \mathbf{I}, \\ \text{s.t. } f \leq_{\mathbf{F}} \text{prov}(i), \\ \text{s.t. } \text{prov}(i) \leq_{\mathbf{R}} \text{req}(i), \end{cases} \quad (10)$$

Another variation is to find only whether there are feasible solutions or not.

Problem (Feasibility). Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find if (10) is feasible.

29.4. Co-design problems

A “co-design problem” will be defined as a *multigraph* of design problems.

Definition 29.18 (Co-design problem with implementation)

A *co-design problem with implementation* (CDPI) is a tuple $\langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, where \mathbf{F} and \mathbf{R} are two posets, and $\langle \mathcal{V}, \mathcal{E} \rangle$ is a multigraph of DPIs. Each node $\mathbf{d} \in \mathcal{V}$ is a DPI $\mathbf{d} = \langle \mathbf{F}_d, \mathbf{R}_d, \mathbf{I}_d, \text{prov}_{\mathbf{d}}, \text{req}_{\mathbf{d}} \rangle$. An edge $e \in \mathcal{E}$ is a tuple $e = \langle \langle \mathbf{d}_1, i_1 \rangle, \langle \mathbf{d}_2, j_2 \rangle \rangle$, where $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{V}$ are two nodes and i_1 and j_2 are the indices of the components of the functionality and resources to be connected, and it holds that $\pi_{i_1} \mathbf{R}_{\mathbf{d}_1} = \pi_{j_2} \mathbf{F}_{\mathbf{d}_2}$ (Fig. 22).

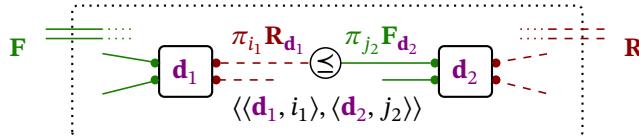


Figure 22.

A CDPI is equivalent to a DPI with an implementation space \mathbf{I} that is a subset of the product $\prod_{\mathbf{d} \in \mathcal{V}} \mathbf{I}_{\mathbf{d}}$, and contains only the tuples that satisfy the co-design constraints. An implementation tuple $i \in \prod_{\mathbf{d} \in \mathcal{V}} \mathbf{I}_{\mathbf{d}}$ belongs to \mathbf{I} iff it respects all functionality–resources constraints on the edges, in the sense that, for all edges $\langle \langle \mathbf{d}_1, i_1 \rangle, \langle \mathbf{d}_2, j_2 \rangle \rangle$ in \mathcal{E} , it holds that

$$\pi_{i_1} \text{req}_{\mathbf{d}_1}(\pi_{\mathbf{d}_1} i) \leq \pi_{j_2} \text{prov}_{\mathbf{d}_2}(\pi_{\mathbf{d}_2} i). \quad (11)$$

The posets \mathbf{F}, \mathbf{R} for the entire CDPI are the products of the functionality and resources of the nodes that remain *unconnected*. For a node \mathbf{d} , let $\text{UF}_{\mathbf{d}}$ and $\text{UR}_{\mathbf{d}}$ be the set of unconnected functionalities and resources. Then \mathbf{F} and \mathbf{R} for the CDPI are defined as the product of the unconnected functionality and resources of all DPIs: $\mathbf{F} = \prod_{\mathbf{d} \in \mathcal{V}} \prod_{j \in \text{UF}_{\mathbf{d}}} \pi_j \mathbf{F}_{\mathbf{d}}$ and $\mathbf{R} = \prod_{\mathbf{d} \in \mathcal{V}} \prod_{i \in \text{UR}_{\mathbf{d}}} \pi_i \mathbf{R}_{\mathbf{d}}$. The maps prov and req return the values of the unconnected functionality and resources:

$$\begin{aligned} \text{prov} : i &\mapsto \prod_{\mathbf{d} \in \mathcal{V}} \prod_{j \in \text{UF}_{\mathbf{d}}} \pi_j \text{prov}_{\mathbf{d}}(\pi_{\mathbf{d}} i), \\ \text{req} : i &\mapsto \prod_{\mathbf{d} \in \mathcal{V}} \prod_{i \in \text{UR}_{\mathbf{d}}} \pi_i \text{req}_{\mathbf{d}}(\pi_{\mathbf{d}} i). \end{aligned} \quad (12)$$

Example 29.19. The CDPI in Fig. 23 is the interconnection of 3 DPs $\mathbf{d}, \mathbf{e}, \mathbf{g}$. The implementation space is a subset of the product

$$\mathbf{I}_{\mathbf{d}} \times \mathbf{I}_{\mathbf{e}} \times \mathbf{I}_{\mathbf{g}}. \quad (13)$$

The elements $\langle i_{\mathbf{d}}, i_{\mathbf{e}}, i_{\mathbf{g}} \rangle$ that are feasible are the ones that respect the following constraints:

1. Functionality and resources of each DPI are given by their implementation:

$$r_{\mathbf{d}} = \text{req}(i_{\mathbf{d}}), \quad (14)$$

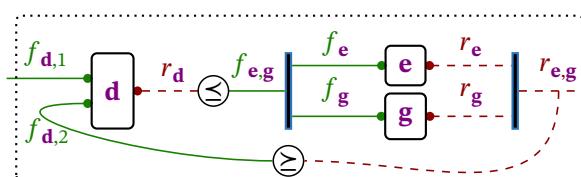


Figure 23.: Example of interconnection of 3 DPs

$$r_e = \text{req}(i_e), \quad (15)$$

$$r_g = \text{req}(i_g), \quad (16)$$

$$f_d = \text{prov}(i_d), \quad (17)$$

$$f_e = \text{prov}(i_e), \quad (18)$$

$$f_g = \text{prov}(i_g). \quad (19)$$

2. Wiring constraints:

$$\langle f_{d_1}, f_{d_2} \rangle = f_d, \quad (20)$$

$$r_{e,g} = \langle r_e, r_g \rangle, \quad (21)$$

$$f_{e,g} = \langle f_e, f_g \rangle. \quad (22)$$

3. Co-design constraints:

$$r_{e,g} \leq f_{d_2}, \quad (23)$$

$$r_d \leq f_{e,g}. \quad (24)$$

Recursive constraints

Example 29.20. Consider the co-design of chassis (Example 29.3) plus motor (Example 29.2). The design problem for a motor has **speed** and **torque** as the provided functionality (what the motor must provide), and **cost**, **mass**, **voltage**, and **current** as the required resources (Fig. 24).

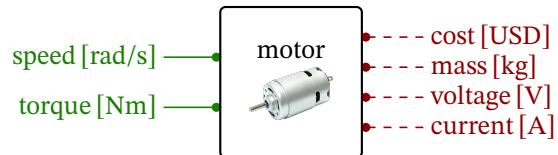


Figure 24.

For the chassis (Fig. 25), the provided functionality is parameterized by the **mass** of the payload and the **cost**, **total mass**, and what the chassis needs from its motor(s), such as **speed** and **torque**.

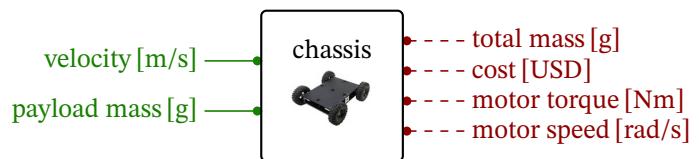


Figure 25.

The two design problems can be connected at the edges for torque and speed, as in Fig. 26. The semantics is that the motor needs to have *at least* the given torque and speed.

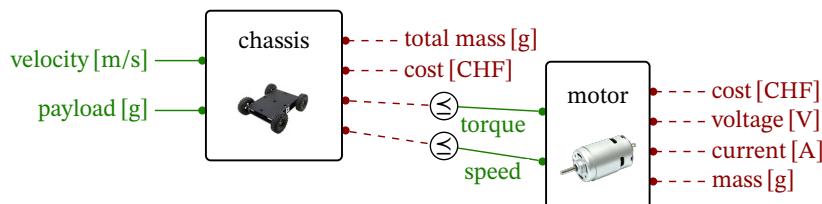


Figure 26.

Resources can be summed together using a trivial DP corresponding to the rela-

tion:

$$f_1 + f_2 \leq r. \quad (25)$$

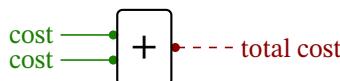


Figure 27.

A co-design problem might contain recursive co-design constraints. For example, if we set the payload to be transported to be the sum of the motor mass plus some extra payload, a cycle appears in the graph (Fig. 28).

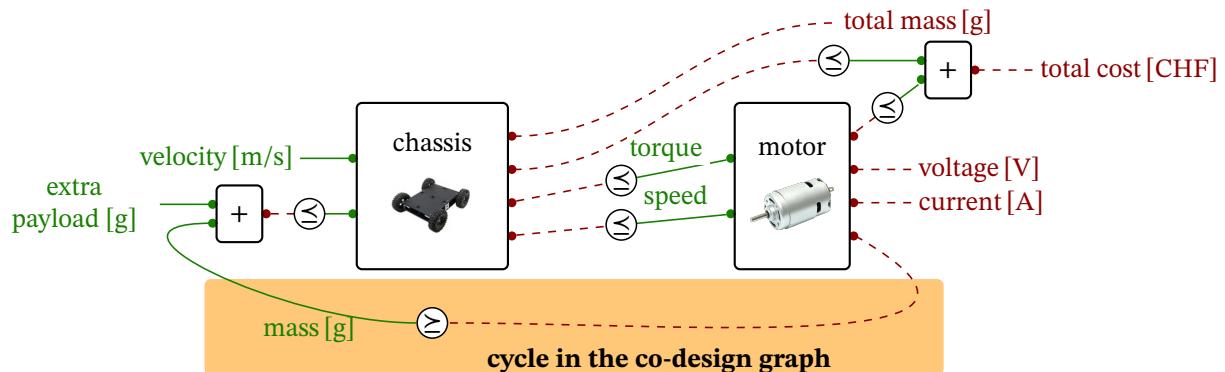


Figure 28.

Abstraction

This formalism makes it easy to abstract away the details in which we are not interested. Once a diagram like Fig. 28 is obtained, we can draw a box around it and consider the abstracted problem (Fig. 29).

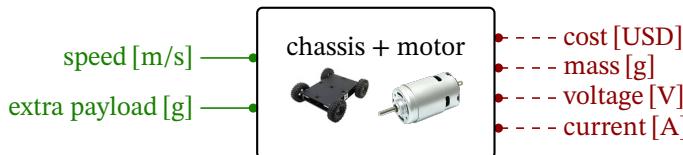


Figure 29.

Let us finish assembling our robot. A motor needs a motor control board. The functional requirements are the (peak) **output current** and the **output voltage range** (Fig. 30).

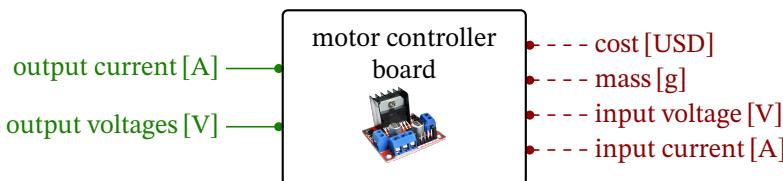


Figure 30.

The functionality for a power supply could be parameterized by the **output current**, the **output voltages**, and the **capacity**. The resources could include **cost** and **mass** (Fig. 31).

Relations such as **current \times voltage \leq power required** and **power \times endurance \leq energy required** can be modeled by a trivial “multiplication” DPI (Fig. 32).

We can connect these DPs to obtain a co-design problem with functionality **voltage**, **current**, **endurance**, and resources **mass** and **cost** (Fig. 33).

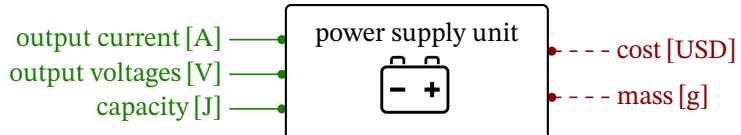


Figure 31.

Figure 32.

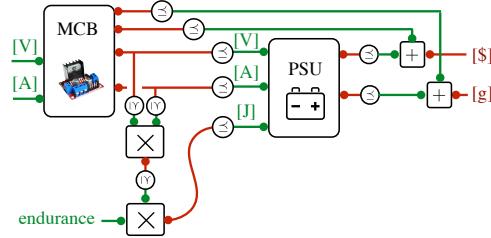
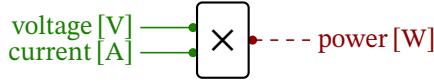


Figure 33.

Draw a box around the diagram, and call it “MCB+PSU”; then interconnect it with the “chassis+motor” diagram in Fig. 34.

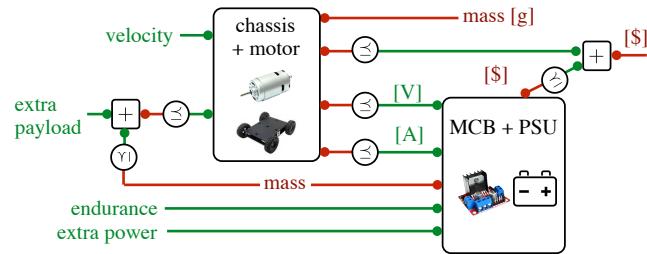


Figure 34.

We can further abstract away the diagram in Fig. 34 as a “mobility+power” CDPI, as in Fig. 35. The formalism allows considering **mass** and **cost** as independent resources, meaning that we wish to obtain the Pareto frontier for the minimal resources. Of course, we can always reduce everything to a scalar objective. For example, a conversion from mass to cost exists, and it is called “shipping”. Depending on the destination, the conversion factor is between \$0.5/lbs, using USPS, to \$10k/lbs for sending your robot to low Earth orbit.

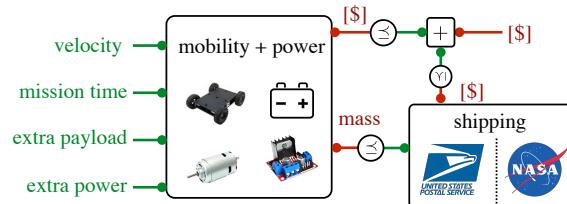


Figure 35.

29.5. The semicategory **DPI**

Definition 29.21 (Semicategory **DPI**)

There is a semicategory **DPI** where

- ▷ The objects are posets, objects of $\langle \text{Pos} \rangle$.
- ▷ The morphisms from **F** to **R** are DPIs $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle$, with **I** an object of $\langle \text{Set} \rangle$.
- ▷ Given two morphisms $\mathbf{d}_1 : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{d}_2 : \mathbf{B} \rightarrow \mathbf{C}$, described by

$$\mathbf{d}_1 = \langle \mathbf{A}, \mathbf{B}, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle, \quad (26)$$

$$\mathbf{d}_2 = \langle \mathbf{B}, \mathbf{C}, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle, \quad (27)$$

their composition $\mathbf{d}_1 ; \mathbf{d}_2$ is given by

$$\mathbf{d}_1 ; \mathbf{d}_2 = \langle \mathbf{A}, \mathbf{C}, \mathbf{I}, \text{prov}, \text{req} \rangle, \quad (28)$$

where

$$\mathbf{I} = \{i_1 \nparallel i_2 \in \mathbf{I}_1 \nparallel \mathbf{I}_2 \mid \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_2(i_2)\}, \quad (29)$$

$$\text{prov} : i_1 \nparallel i_2 \mapsto \text{prov}_1(i_1), \quad (30)$$

$$\text{req} : i_1 \nparallel i_2 \mapsto \text{req}_2(i_2). \quad (31)$$

The semantics of the interconnection is that the second DPI provides the resources required by the first DPI. This is a partial order inequality constraint of the type $r_1 \leq f_2$.

Lemma 29.22. Series composition is associative.

Proof. Consider

$$\mathbf{d}_1 = \langle \mathbf{A}, \mathbf{B}, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle,$$

$$\mathbf{d}_2 = \langle \mathbf{B}, \mathbf{C}, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle, \quad (32)$$

$$\mathbf{d}_3 = \langle \mathbf{C}, \mathbf{D}, \mathbf{I}_3, \text{prov}_3, \text{req}_3 \rangle.$$

We want to show that

$$(\mathbf{d}_1 ; \mathbf{d}_2) ; \mathbf{d}_3 = \mathbf{d}_1 ; (\mathbf{d}_2 ; \mathbf{d}_3). \quad (33)$$

We know that the first part of the left term of (33) gives

$$(\mathbf{d}_1 ; \mathbf{d}_2) = \langle \mathbf{A}, \mathbf{C}, \mathbf{I}_{1,2}, \text{prov}_{1,2}, \text{req}_{1,2} \rangle, \quad (34)$$

where

$$\mathbf{I}_{1,2} = \{i_1 \nparallel i_2 \in \mathbf{I}_1 \nparallel \mathbf{I}_2 \mid \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_2(i_2)\} \quad (35)$$

$$\text{prov}_{1,2} : i_1 \nparallel i_2 \mapsto \text{prov}_1(i_1) \quad (36)$$

$$\text{req}_{1,2} : i_1 \nparallel i_2 \mapsto \text{req}_2(i_2) \quad (37)$$

Therefore, the full left term of (33) reads

$$(\mathbf{d}_1 ; \mathbf{d}_2) ; \mathbf{d}_3 = \langle \mathbf{A}, \mathbf{D}, \mathbf{I}_{1,3}, \text{prov}_{1,3}, \text{req}_{1,3} \rangle, \quad (38)$$

where

$$\mathbf{I}_{1,3} = \{i_1 \nparallel i_2 \nparallel i_3 \in \mathbf{I} \mid \text{req}_{1,2}(i_1 \nparallel i_2) \leq_{\mathbf{C}} \text{prov}_3(i_3) \wedge \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_2(i_2)\}, \quad (39)$$

$$= \{i_1 \circ i_2 \circ i_3 \in \mathbf{I} \mid \text{req}_2(i_2) \leq_{\mathbf{C}} \text{prov}_3(i_3) \wedge \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_2(i_2)\}, \quad (40)$$

$$\text{prov}_{1,3} : i_1 \circ i_2 \circ i_3 \mapsto \text{prov}_{1,2}(i_1 \circ i_2) = \text{prov}_1(i_1), \quad (41)$$

$$\text{req}_{1,3} : i_1 \circ i_2 \circ i_3 \mapsto \text{req}_3(i_3), \quad (42)$$

where $\mathbf{I} = \mathbf{I}_1 \circ \mathbf{I}_2 \circ \mathbf{I}_3$.

By expanding the second part of the second term of (33), we have:

$$(\mathbf{d}_2 \circ \mathbf{d}_3) = \langle \mathbf{B}, \mathbf{D}, \mathbf{I}_{2,3}, \text{prov}_{2,3}, \text{req}_{2,3} \rangle, \quad (43)$$

where

$$\mathbf{I}_{2,3} = \{i_2 \circ i_3 \in \mathbf{I}_2 \circ \mathbf{I}_3 \mid \text{req}_2(i_2) \leq_{\mathbf{C}} \text{prov}_3(i_3)\} \quad (44)$$

$$\text{prov}_{2,3} : i_2 \circ i_3 \mapsto \text{prov}_2(i_2) \quad (45)$$

$$\text{req}_{2,3} : i_2 \circ i_3 \mapsto \text{req}_3(i_3) \quad (46)$$

Therefore, the full right term of (33) reads

$$\mathbf{d}_1 \circ (\mathbf{d}_2 \circ \mathbf{d}_3) = \langle \mathbf{A}, \mathbf{D}, \mathbf{I}_{1,3}', \text{prov}_{1,3}', \text{req}_{1,3}' \rangle, \quad (47)$$

where

$$\mathbf{I}_{1,3}' = \{i_1 \circ i_2 \circ i_3 \in \mathbf{I} \mid \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_{2,3}(i_2 \circ i_3) \wedge \text{req}_2(i_2) \leq_{\mathbf{C}} \text{prov}_3(i_3)\}, \quad (48)$$

$$= \{i_1 \circ i_2 \circ i_3 \in \mathbf{I} \mid \text{req}_1(i_1) \leq_{\mathbf{B}} \text{prov}_2(i_2) \wedge \text{req}_2(i_2) \leq_{\mathbf{C}} \text{prov}_3(i_3)\}, \quad (49)$$

$$\text{prov}_{1,3}' : i_1 \circ i_2 \circ i_3 \mapsto \text{prov}_1(i_1), \quad (50)$$

$$\text{req}_{1,3}' : i_1 \circ i_2 \circ i_3 \mapsto \text{req}_{2,3}(i_2 \circ i_3) = \text{req}_3(i_3). \quad (51)$$

It is clear that $\mathbf{I}_{1,3} = \mathbf{I}_{1,3}'$, $\text{prov}_{1,3} = \text{prov}_{1,3}'$, and $\text{req}_{1,3} = \text{req}_{1,3}'$. This, together with (38) and (47) shows associativity. \square

These two properties are sufficient to conclude that there exists a semicategory of design problems.

Lemma 29.23. **DPI** is not a category, because one cannot find identities.

Proof. We prove this by contradiction. Suppose we can find a DPI that works as an identity for interconnection for any other DPI with implementation space \mathbf{I}_{id} . Therefore, when postponed to a DPI with implementation space \mathbf{I}_1 we should have that

$$\mathbf{I}_1 \circ \mathbf{I}_{\text{id}} = \mathbf{I}_1. \quad (52)$$

This implies that \mathbf{I}_{id} is an empty list of sets, that is inhabited by only one element, the empty tuple. Therefore, the map req_{id} of the identity is necessarily a constant, because there is only one element in the domain. Therefore, it is impossible to preserve req_1 . \square

29.6. Sum and intersection of DPIS

Sum of DPIS

The sum of two design problems with implementation is a design problem with the implementation space $I = I_1 + I_2$, and it represents the exclusive choice between two possible alternative families of designs.

Definition 29.24 (Sum of DPIS)

Given two DPIS with same functionality and resources $d = \langle F, R, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $e = \langle F, R, I_2, \text{prov}_2, \text{req}_2 \rangle$, define their sum as

$$d \vee e := \langle F, R, I_1 + I_2, \text{prov}, \text{req} \rangle, \quad (53)$$

where

$$\begin{aligned} \text{prov} &= \text{prov}_1 + \text{prov}_2, \\ \text{req} &= \text{req}_1 + \text{req}_2. \end{aligned} \quad (54)$$

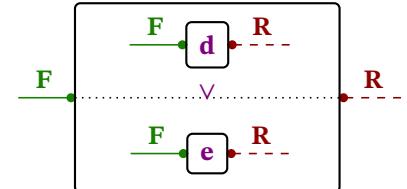


Figure 36.

Intersection of DPIS

Definition 29.25 (Intersection of DPIS)

Given two DPIS with same functionality and resources $d = \langle F, R, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $e = \langle F, R, I_2, \text{prov}_2, \text{req}_2 \rangle$, define their intersection as

$$d \wedge e := \langle F, R, I, \text{prov}, \text{req} \rangle, \quad (55)$$

where

$$\begin{aligned} I &= \{(i_1 \circ_i i_2 \circ_i f \circ_i r) \in (I_1 \circ_i I_2 \circ_i F \circ_i R) \mid \\ &(f \leq_F \text{prov}_1(i_1)) \wedge (f \leq_F \text{prov}_2(i_2)) \wedge \\ &(\text{req}_1(i_1) \leq_R r) \wedge (\text{req}_2(i_2) \leq_R r)\} \end{aligned} \quad (56)$$

and the maps prov and req are the projections of the third and fourth component.

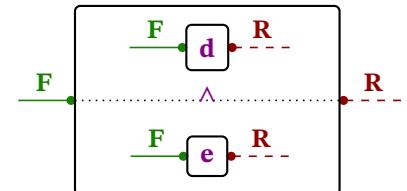


Figure 37.



30. Feasibility

In the previous chapter we have introduced the *design problems with implementations*. Those describe relations among 3 sets: functionality, resources, and implementations.

If we are not interested in the implementations, we can simplify the model and obtain a relation directly between functionality and resources. We obtain in this way a category **DP**.

30.1 From DP_Is to DP_s 434

30.1. From DPPIs to DPs

A DPPI (Def. 29.1) describes a relation between three sets: \mathbf{F} , \mathbf{R} , \mathbf{I} . If we are not interested in the implementations, but just in the relation between \mathbf{F} and \mathbf{R} , then we can describe a DPPI more compactly as a “DP”.

Remark 30.1. Given a DPPI $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle$ it is always possible to obtain from it the following DP

$$\begin{aligned} \mathbf{d}: \mathbf{F}^{\text{op}} \times \mathbf{R} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle f^*, r \rangle &\mapsto \exists i \in \mathbf{I}: (f \leq_{\mathbf{F}} \text{prov}(i)) \wedge (\text{req}(i) \leq_{\mathbf{R}} r). \end{aligned} \quad (1)$$

Evaluating this DP is the same as asking whether the set

$$\{i \in \mathbf{I}: (f \leq_{\mathbf{F}} \text{prov}(i)) \wedge (\text{req}(i) \leq_{\mathbf{R}} r)\} \quad (2)$$

is empty or not.

Example 30.2. Recall Example 29.4, with the catalogue of electric motors in Table 30.1.

Table 30.1.: A simplified catalogue of motors.

Torque [kg · cm]	Motor ID	Weight [g]	Max Power [W]	Cost [USD]
0.18	1204	60.0	2.34	19.95
0.95	1206	140.0	3.00	19.95
0.65	1207	130.0	2.07	12.95
3.7	2267	285.0	4.76	16.95
1.9	2279	165.0	5.40	164.95
19.0	1478	1,000	8.96	49.95
2.2	2299	150.0	5.90	59.95

Table 30.2.: Feasibility relations for the design problem of motors.

Torque [kg · cm]	Weight [g]	Max Power [W]	Cost [USD]
0.18	60.0	2.34	19.95
0.95	140.0	3.00	19.95
0.65	130.0	2.07	12.95
3.7	285.0	4.76	16.95
1.9	165.0	5.40	164.95
19.0	1,000	8.96	49.95
2.2	150.0	5.90	59.95

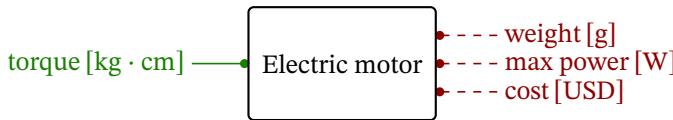


Figure 1.: Electric motor design problem.

The catalogue induces a design problem \mathbf{d}_{EM} , where each feasibility relation between functionality and resources is reported in Table 30.2. with diagrammatic form as in Fig. 1. In particular, we can query the design problem for combinations of **functionalities** and **resources**. For instance:

$$\mathbf{d}_{\text{EM}}(0.2 \text{ kg} \cdot \text{cm}, \langle 50.0 \text{ g} \rangle, 2.0 \text{ W}, 15.0 \text{ USD}) = \perp, \quad (3)$$

since no listed model can provide $0.2 \text{ kg} \cdot \text{cm}$ torque by requiring the set of resources $\langle 50.0 \text{ g}, 2.0 \text{ W}, 15.0 \text{ USD} \rangle$ or less.

We have already seen in Remark 30.1 that we can obtain a DP from a DPI. We can make this more formal and say that there exists a forgetful semifunctor from **DPI** to **DP**.

Definition 30.3

The forgetful semifunctor $F : \mathbf{DPI} \rightarrow \mathbf{DP}$ is given by:

1. Identity on the objects: $F_*(P) = P$.
2. Given $\mathbf{d} = \langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \mathbf{prov}, \mathbf{req} \rangle$, the action on morphisms is given by

$$F_*(\mathbf{d}) : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}, \quad (f^*, r) \mapsto \exists i \in \mathbf{I} : (f \leq_{\mathbf{F}} \mathbf{prov}(i)) \wedge (\mathbf{req}(i) \leq_{\mathbf{R}} r). \quad (4)$$

Lemma 30.4. Def. 30.3 indeed defines a semifunctor.

Proof. Consider

$$\begin{aligned}\mathbf{d} &= \langle \mathbf{P}, \mathbf{Q}, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle, \\ \mathbf{e} &= \langle \mathbf{Q}, \mathbf{R}, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle,\end{aligned}\quad (5)$$

We need to show

$$F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e}) = F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}). \quad (6)$$

Let's start with the left term. One has

$$F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e})(p^*, r) = \exists i \in \mathbf{I}: (p \leq_{\mathbf{P}} \text{prov}_1(i_1)) \wedge (\text{req}_2(i_2) \leq_{\mathbf{R}} r), \quad (7)$$

where $\mathbf{I} = \{i_1 \mid i_2 \in \mathbf{I}_1 \wedge i_2 \in \mathbf{I}_2 \mid \text{req}_1(i_1) \leq_{\mathbf{Q}} \text{prov}_2(i_2)\}$.

On the other hand,

$$\begin{aligned}&(F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}))(p^*, r) \\&= \bigvee_{q \in \mathbf{Q}} F_{\rightarrow}(\mathbf{d})(p^*, q) \wedge F_{\rightarrow}(\mathbf{e})(q^*, r) \\&= \bigvee_{q \in \mathbf{Q}} (\exists i_1 \in \mathbf{I}_1 : (p \leq_{\mathbf{P}} \text{prov}_1(i_1)) \wedge (\text{req}_1(i_1) \leq_{\mathbf{Q}} q)) \\&\quad \wedge (\exists i_2 \in \mathbf{I}_2 : (q \leq_{\mathbf{Q}} \text{prov}_2(i_2)) \wedge (\text{req}_2(i_2) \leq_{\mathbf{R}} r))\end{aligned}\quad (8)$$

Consider the following cases:

- ▷ If $F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e})(p^*, r) = \top$, there exist $i_1 \in \mathbf{I}_1, i_2 \in \mathbf{I}_2$ for which

$$\begin{aligned}p &\leq_{\mathbf{P}} \text{prov}_1(i_1), \\ \text{req}_2(i_2) &\leq_{\mathbf{R}} r, \\ \text{req}_1(i_1) &\leq_{\mathbf{Q}} \text{prov}_2(i_2).\end{aligned}\quad (9)$$

The first two terms are clear, and the last term implies that there exists a $q \in \mathbf{Q}$ such that

$$(q \leq_{\mathbf{Q}} \text{prov}_2(i_2)) \wedge (\text{req}_1(i_1) \leq_{\mathbf{Q}} q), \quad (10)$$

implying $(F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}))(p^*, r) = \top$.

- ▷ The case

$$\frac{F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e})(p^*, r) = \perp}{(F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}))(p^*, r) = \perp} \quad (11)$$

follows analogously.

- ▷ The other direction is easier to show, since clearly

$$\frac{(F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}))(p^*, r) = \top}{F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e})(p^*, r) = \top} \quad (12)$$

and

$$\frac{(F_{\rightarrow}(\mathbf{d}) ;_{\text{DP}} F_{\rightarrow}(\mathbf{e}))(p^*, r) = \perp}{F_{\rightarrow}(\mathbf{d} ;_{\text{DPI}} \mathbf{e})(p^*, r) = \perp} \quad (13)$$

by inspecting (7) and (8).

□

In the other direction, we can take a DP and find a corresponding DPI.

Given a DP $\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow \mathbf{Pos}$ \mathbf{Bool} , we can define a DPI $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle$ by

setting

$$\begin{aligned}\mathbf{I} &= \{\langle f, r \rangle \in \mathbf{F} \times \mathbf{R} : \mathbf{d}(f^*, r)\}, \\ \mathbf{prov} : \langle f, r \rangle &\mapsto f, \\ \mathbf{req} : \langle f, r \rangle &\mapsto r,\end{aligned}\tag{14}$$

However, this operation is not a semifunctor, since it does not preserve composition.



31. Lattices

31.1 Monoidal posets	440
31.2 Monoidal-time procedures	442
31.3 Lattices	444
31.4 Lattice homomorphisms	447
31.5 Categories Lat and BoundedLat	448

31.1. Monoidal posets

A monoidal poset is a poset that is also a monoid, and in which the monoidal product is a monotone map that is compatible with the order.

Definition 31.1 (Monoidal poset)

A *monoidal structure* on a poset $P = \langle P, \leq_P \rangle$ is specified by:

Constituents

1. A monotone map $\otimes : P \times P \rightarrow_{\text{Pos}} P$, called the *monoidal product*.
2. An element $1 \in P$, called the *monoidal unit*.

Conditions

1. Associativity: for all $x, y, z \in P$:

$$(x \otimes y) \otimes z = x \otimes (y \otimes z). \quad (1)$$

2. Left and right unitality: for all $x \in P$:

$$1 \otimes x = x \quad \text{and} \quad x \otimes 1 = x. \quad (2)$$

A poset equipped with a monoidal structure is called a *monoidal poset*.

Note that here we are implicitly assuming $P \times P$ as having the product order (Def. 6.1). In detail, monotonicity means that, for all $x_1, x_2, y_1, y_2 \in P$:

$$\frac{x_1 \leq_P y_1 \quad x_2 \leq_P y_2}{(x_1 \otimes x_2) \leq_P (y_1 \otimes y_2)}. \quad (3)$$

Definition 31.2 (Symmetric monoidal poset)

A *symmetric monoidal poset* is a monoidal poset $P = \langle P, \leq_P, \otimes, 1 \rangle$ such that, for all $x, y \in P$,

$$x \otimes y = y \otimes x. \quad (4)$$

Example 31.3 (Reals with addition). Consider the real numbers \mathbb{R} with the poset structure given the usual ordering. Consider 0 as the monoidal unit and the operation $+ : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ as monoidal product. It is easy to see that the conditions of Def. 31.1 are satisfied:

- (a) Given $p_1, p_2, q_1, q_2 \in \mathbb{R}$, we know:

$$\frac{p_1 \leq p_2 \quad q_1 \leq q_2}{(p_1 + p_2) \leq (q_1 + q_2)}. \quad (5)$$

- (b) $0 + p = p + 0 = 0, \forall p \in \mathbb{R}$.

- (c) $(p + q) + r = p + (q + r), \forall p, q, r \in \mathbb{R}$.

Counter-example 31.4. Someone proposes now to substitute the monoidal unit in Example 31.3 with 1 and the monoidal product with multiplication “ \cdot ”. This does not form a monoidal poset anymore. To see a simple counterexample, consider the fact that $-5 \leq 0$ and $-4 \leq 3$. However, $(-5) \cdot (-4) \not\leq 0 \cdot 3$.

\wedge	\perp	T
\perp	\perp	\perp
T	\perp	T

Example 31.5 (Boolean monoid). The booleans form a monoidal poset $\langle \text{Bool}, \leq_{\text{Bool}}, T, \wedge \rangle$ with the unit being T and the product being \wedge . The action of the monoidal product “ \wedge ” can be summarized in the table on the side. From this table,

it is clear that given $x_1 \leq_{\text{Bool}} y_1$ and $x_2 \leq_{\text{Bool}} y_2$, we have $x_1 \wedge x_2 \leq_{\text{Bool}} y_1 \wedge y_2$ (if you do not believe it, try all possible combinations). Furthermore, $x \wedge T = x = T \wedge x$.

Graded exercise I.1 (`HwkMonoidalPosets`)

Prove or disprove that the following are monoidal posets:

1. The set \mathbb{R} equipped with the usual ordering, addition as monoidal product, and $0 \in \mathbb{R}$ as monoidal unit.
2. The set \mathbb{R} equipped with the usual ordering, multiplication as monoidal product, and $1 \in \mathbb{R}$ as monoidal unit.

Graded exercise I.2 (`HwkInternalHomCancelling`)

Let $\mathbf{P} = \langle \mathbf{P}, \leq, \otimes, \mathbf{1}, \setminus \rangle$ be a closed monoidal poset. Prove that for any $x, y, z \in \mathbf{P}$, we have

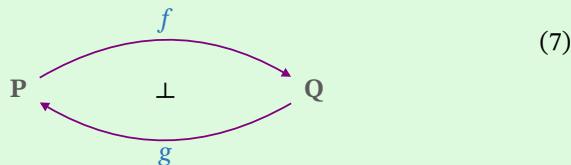
$$(x \setminus y) \otimes (y \setminus z) \leq x \setminus z. \quad (6)$$

Graded exercise I.3 (`HwkInternalLeastUpperBounds`) 1. Let $\mathbf{P} = \langle \mathbf{P}, \leq \rangle$ be a poset, and consider a subset $\mathbf{S} \subseteq \mathbf{P}$. A *least upper bound*, or *join*, for \mathbf{S} is an element $x \in \mathbf{P}$ which satisfies the conditions

- a) $y \leq x \forall y \in \mathbf{S}$;
- b) if $x' \in \mathbf{P}$ is such that $y \leq x' \forall y \in \mathbf{S}$, then $x \leq x'$ must hold.

Prove that, if a least upper bound of a subset \mathbf{S} exists, then it is unique. In this case we use the notation $\bigvee \mathbf{S}$ to denote it.

2. Suppose that we have a Galois connection



between posets \mathbf{P} and \mathbf{Q} . Furthermore, assume that \mathbf{P} has all joins, meaning that for any subset $\mathbf{S} \subseteq \mathbf{P}$, the least upper bound $\bigvee \mathbf{S}$ exists and is an element of \mathbf{P} .

Prove that for any subset $\mathbf{S} \subseteq \mathbf{P}$ it holds that

$$f(\bigvee \mathbf{S}) = \bigvee \{f(x) \mid x \in \mathbf{S}\}. \quad (8)$$

31.2. Monoidal-time procedures

Before, we thought of sized sets (Def. 15.14) as a datatype that can be measured with integer sizes. However, this does not capture some important cases. For example, if we are dealing with *trees*, from the point of view of computation it could be important to think about multidimensional sizes: for example, we might want to account for number of nodes, number of edges, maximum branching factor, and so on. Still, we want to know when an instance is bigger than another: this is a perfect job for posets.

Definition 31.6 (Poset-sized sets)

A *poset-sized set* is a tuple $\langle \mathbf{A}, \Sigma_{\mathbf{A}}, \text{size} \rangle$, where \mathbf{A} is a set, $\Sigma_{\mathbf{A}}$ is a poset, and $\text{size} : \mathbf{A} \rightarrow \Sigma_{\mathbf{A}}$ is the size function.

In **ProcSizeTime**, we assumed that time was measured using real numbers. We can generalize this to an arbitrary poset, for example allowing counting “number of operations”. We need an additional structure: in (55) we needed a “+” to sum the time of successive procedures. Therefore, we assume that we have a time monoidal poset \mathbf{T} .

Definition 31.7 (Semicategory $\mathbf{ProcSizeTime}_{\mathbf{T}}$)

For a given monoidal poset $\langle \mathbf{T}, \otimes_{\mathbf{T}} \rangle$, the semicategory $\mathbf{ProcSizeTime}_{\mathbf{T}}$ consists of the following constituents:

1. *Objects*: The objects are poset-sized sets.
2. *Morphisms*: A morphism

$$\mathfrak{f} : \mathbf{X} \rightarrow_{\mathbf{ProcSize}} \mathbf{Y} \quad (9)$$

between the two objects

$$\mathbf{X} = \langle \mathbf{A}, \Sigma_{\mathbf{A}}, \text{size}_{\mathbf{A}} \rangle \quad \text{and} \quad \mathbf{Y} = \langle \mathbf{B}, \Sigma_{\mathbf{B}}, \text{size}_{\mathbf{B}} \rangle \quad (10)$$

is a tuple

$$\langle \mathfrak{f}_e, \sigma, \text{dur} \rangle, \quad (11)$$

where:

- a) $\mathfrak{f}_e : \mathbf{A} \rightarrow \mathbf{B}$ is the function computed;
- b) $\sigma : \Sigma_{\mathbf{A}} \rightarrow \mathbf{Pos} \Sigma_{\mathbf{B}}$ is a monotone function that keeps track of how the size changes.
- c) $\text{dur} : \Sigma_{\mathbf{A}} \rightarrow \mathbf{Pos} \mathbf{T}$ is a monotone function that gives computation time as a function of instance size;

3. *Composition*: The composition of

$$\langle \mathfrak{f}_1, \sigma_1, \text{dur}_1 \rangle \quad \text{and} \quad \langle \mathfrak{g}_2, \sigma_2, \text{dur}_2 \rangle \quad (12)$$

is given by

$$\langle \mathfrak{f}_{1;2}, \sigma_{1;2}, \text{dur}_{1;2} \rangle, \quad (13)$$

where

$$\mathfrak{f}_{1;2} = \mathfrak{f}_1 ; \mathfrak{g}_2, \quad (14)$$

$$\sigma_{1;2} = \sigma_1 ; \sigma_2, \quad (15)$$

and $\text{dur}_{1,2}$ is defined as

$$\begin{aligned}\text{dur}_{1,2} : \Sigma_{\text{A}} &\rightarrow \mathbf{T}, \\ \sigma_{\text{A}} &\mapsto \text{dur}_1(\sigma_{\text{A}}) \otimes_{\mathbf{T}} \text{dur}_2(\sigma_1(\sigma_{\text{A}})).\end{aligned}\tag{16}$$

31.3. Lattices

Definition 31.8 (Lattice)

A lattice is a poset $P = \langle P, \leq_P \rangle$ with the additional property that, for any two-element subset $\{x, y\} \subseteq P$, both the join $\vee\{x, y\}$ and the meet $\wedge\{x, y\}$ exist. Usually these are written using infix notation as $x \vee y$ and $x \wedge y$, respectively.

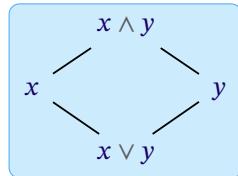
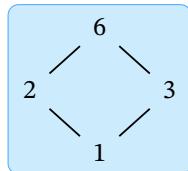


Figure 1.



(a) A lattice



(b) Not a lattice

Figure 2.: Examples of a lattice and a non-lattice.

a	b	$a \leq b$	$a \wedge b$	$a \vee b$
T	T	T	T	T
T	⊥	⊥	⊥	T
⊥	T	T	⊥	T
⊥	⊥	T	⊥	⊥

Table 31.1.: Properties of the **Bool** poset. Note that $\leq \equiv \Rightarrow$.

Example 31.10. In Def. 5.12 we presented the poset arising from the power set $\text{Pow } \mathbf{A}$ of a set \mathbf{A} and ordered via subset inclusion. This is a lattice, bounded by \mathbf{A} and by the empty set \emptyset . Note that this lattice possesses two (dual) monoidal structures $\langle \text{Pow } \mathbf{A}, \subseteq, \emptyset, \cup \rangle$ and $\langle \text{Pow } \mathbf{A}, \subseteq, \mathbf{A}, \cap \rangle$.

Example 31.11. Consider the poset **Bool**, in which $b_1 \leq_{\text{Bool}} b_2$ iff $b_1 \Rightarrow b_2$, that is, in addition to the operation

$$\Rightarrow : \text{Bool} \times \text{Bool} \rightarrow \text{Bool}, \quad (17)$$

called *implication*, there are also the familiar *and* (\wedge) and *or* (\vee) operations. Note that \wedge and \vee are commutative ($b \wedge c = c \wedge b$, $b \vee c = c \vee b$), whereas \Rightarrow is not. Furthermore, \wedge and \vee correspond to the meet and the join, respectively.

Example 31.12. Consider the set $\{1, 2, 3, 6\}$ ordered by divisibility. For instance, since 2 divides 6, we have $2 \leq 6$. This is a lattice. However, the set $\{1, 2, 3\}$ ordered by divisibility is not, since 2 and 3 lack a meet (Fig. 2).

Lemma 31.13. UP is a bounded lattice (Def. 31.8) with

$$\begin{aligned} \leq_{\text{UP}} &:= \sqsubseteq, \\ \perp_{\text{UP}} &:= \mathbf{P}, \\ \top_{\text{UP}} &:= \emptyset, \\ \wedge_{\text{UP}} &:= \cap, \\ \vee_{\text{UP}} &:= \cup. \end{aligned} \quad (18)$$

Proof. Consider the poset $\text{UP} = \langle \text{USets } \mathbf{P}, \leq_{\text{UP}} \rangle$ and $\mathbf{A}, \mathbf{B} \in \text{USets } \mathbf{P}$.

First, we need to show that $\mathbf{A} \cap \mathbf{B} \in \text{USets } \mathbf{P}$. To this extent, we need to show that, for all $a \in \mathbf{A} \cap \mathbf{B}$ and for all $a \leq_{\text{P}} b$, it holds $b \in \mathbf{A} \cap \mathbf{B}$. We have $\mathbf{A} \in \text{USets } \mathbf{P}$ and $\mathbf{B} \in \text{USets } \mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cap \mathbf{B}$, we have $a \in \mathbf{A} \wedge a \in \mathbf{B}$. It follows that $b \in \text{USets } \mathbf{P}$ and $b \in \text{USets } \mathbf{Q}$. Therefore, $b \in \text{USets } \mathbf{P} \cap \mathbf{B}$ and, thus, $\mathbf{A} \cap \mathbf{B} \in \text{USets } \mathbf{P}$. Furthermore, we need to show that $\mathbf{A} \cap \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} . Let $\mathbf{C} \in \text{USets } \mathbf{P}$ such that

$$\mathbf{A} \leq_{\text{UP}} \mathbf{C} \leq_{\text{UP}} (\mathbf{A} \cap \mathbf{B}) \iff \mathbf{A} \supseteq \mathbf{C} \supseteq (\mathbf{A} \cap \mathbf{B})$$

and

$$\mathbf{B} \leq_{\text{UP}} \mathbf{C} \leq_{\text{UP}} (\mathbf{A} \cap \mathbf{B}) \iff \mathbf{B} \supseteq \mathbf{C} \supseteq (\mathbf{A} \cap \mathbf{B}).$$

Using the fact that intersection preserves inclusions, we have

$$\begin{aligned} & (\mathbf{A} \cap \mathbf{B}) \supseteq (\mathbf{C} \cap \mathbf{C}) \supseteq (\mathbf{A} \cap \mathbf{B}) \\ \iff & (\mathbf{A} \cap \mathbf{B}) \supseteq \mathbf{C} \supseteq (\mathbf{A} \cap \mathbf{B}) \\ \iff & \mathbf{C} = (\mathbf{A} \cap \mathbf{B}). \end{aligned}$$

Therefore, $\mathbf{A} \cap \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} .

Second, we need to show that $\mathbf{A} \cup \mathbf{B} \in \text{USets } \mathbf{P}$, meaning that for all $a \in \mathbf{A} \cup \mathbf{B}$, $a \leq_{\mathbf{P}} b$ implies $b \in \mathbf{A} \cup \mathbf{B}$. We have $\mathbf{A} \in \text{USets } \mathbf{P}$ and $\mathbf{B} \in \text{USets } \mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cup \mathbf{B}$, we have either $a \in \mathbf{A}$ or $a \in \mathbf{B}$. If $a \in \mathbf{A}$, then $b \in \text{USets } \mathbf{A}$. If $a \in \mathbf{B}$, then $b \in \text{USets } \mathbf{B}$. Either way, $b \in \mathbf{A} \cup \mathbf{B}$ and, thus, $\mathbf{A} \cup \mathbf{B} \in \text{USets } \mathbf{P}$. Furthermore, we need to show that $\mathbf{A} \cup \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} . Let $\mathbf{C} \in \text{USets } \mathbf{P}$ such that

$$\mathbf{A} \cup \mathbf{B} \leq_{\text{UP}} \mathbf{C} \leq_{\text{UP}} \mathbf{A} \iff \mathbf{A} \cup \mathbf{B} \supseteq \mathbf{C} \supseteq \mathbf{A}$$

and

$$\mathbf{A} \cup \mathbf{B} \leq_{\text{UP}} \mathbf{C} \leq_{\text{UP}} \mathbf{B} \iff \mathbf{A} \cup \mathbf{B} \supseteq \mathbf{C} \supseteq \mathbf{B}.$$

Using the fact that union preserves inclusions, we have

$$\begin{aligned} & (\mathbf{A} \cup \mathbf{B}) \cup (\mathbf{A} \cup \mathbf{B}) \supseteq (\mathbf{C} \cup \mathbf{C}) \supseteq (\mathbf{A} \cup \mathbf{B}) \\ \iff & \mathbf{A} \cup \mathbf{B} \supseteq \mathbf{C} \supseteq (\mathbf{A} \cup \mathbf{B}) \\ \iff & \mathbf{C} = (\mathbf{A} \cup \mathbf{B}). \end{aligned}$$

Therefore, $\mathbf{A} \cup \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} .

We have therefore proved that $\text{UP} = \langle \text{USets } \mathbf{P}, \leq_{\text{UP}} \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq \mathbf{C}$ for any $\mathbf{C} \in \text{USets } \mathbf{P}$, meaning that \emptyset is the top. Furthermore, we notice that $\mathbf{C} \subseteq \mathbf{P}$ for any $\mathbf{C} \in \text{USets } \mathbf{P}$, meaning that \mathbf{P} is a bottom. Therefore, the lattice is bounded. \square

Lemma 31.14. LP is a bounded lattice (Def. 31.8) with:

$$\begin{aligned} \leq_{\text{LP}} &:= \subseteq, \\ \perp_{\text{LP}} &:= \emptyset, \\ \top_{\text{LP}} &:= \mathbf{P}, \\ \wedge_{\text{LP}} &:= \cap, \\ \vee_{\text{LP}} &:= \cup. \end{aligned}$$

Proof. Consider the poset $\text{LP} = \langle \text{LSets } \mathbf{P}, \leq_{\text{LP}} \rangle$ and $\mathbf{A}, \mathbf{B} \in \text{LSets } \mathbf{P}$.

First, we need to show that $\mathbf{A} \cup \mathbf{B} \in \text{LSets } \mathbf{P}$. That is, $b \leq_{\text{P}} a$ implies $b \in \mathbf{A} \cup \mathbf{B}$. We have $\mathbf{A} \in \text{LSets } \mathbf{P}$ and $\mathbf{B} \in \text{LSets } \mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cup \mathbf{B}$, either $a \in \mathbf{A}$ or $a \in \mathbf{B}$. If $a \in \mathbf{A}$, then $b \in \mathbf{A}$. If $a \in \mathbf{B}$, then $b \in \mathbf{B}$. It follows that $b \in \mathbf{A} \cup \mathbf{B}$ and, thus, $\mathbf{A} \cup \mathbf{B} \in \text{LSets } \mathbf{P}$. Furthermore, we need to show that $\mathbf{A} \cup \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} . Consider $\mathbf{C} \in \text{LSets } \mathbf{P}$ such that

$$\mathbf{A} \leq_{\text{LP}} \mathbf{C} \leq_{\text{LP}} \mathbf{A} \cup \mathbf{B} \iff \mathbf{A} \subseteq \mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B}$$

and

$$\mathbf{B} \leq_{\text{LP}} \mathbf{C} \leq_{\text{LP}} \mathbf{A} \cup \mathbf{B} \iff \mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B}.$$

Using the fact that union preserves inclusions, we have

$$\begin{aligned} & (\mathbf{A} \cup \mathbf{B}) \subseteq (\mathbf{C} \cup \mathbf{C}) \subseteq (\mathbf{A} \cup \mathbf{B}) \\ \iff & (\mathbf{A} \cup \mathbf{B}) \subseteq \mathbf{C} \subseteq (\mathbf{A} \cup \mathbf{B}) \\ \iff & \mathbf{C} = (\mathbf{A} \cup \mathbf{B}). \end{aligned}$$

Therefore, $\mathbf{A} \cup \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} .

Second, we need to show that $\mathbf{A} \cap \mathbf{B} \in \text{LSets } \mathbf{P}$. That is, $b \leq_{\mathbf{P}} a$ implies $b \in \mathbf{A} \cap \mathbf{B}$. We have $\mathbf{A} \in \text{LSets } \mathbf{P}$ and $\mathbf{B} \in \text{LSets } \mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cap \mathbf{B}$, we have $a \in \mathbf{A} \wedge a \in \mathbf{B}$. Since $\mathbf{A}, \mathbf{B} \in \text{LSets } \mathbf{P}$, this implies $b \in \mathbf{A} \wedge b \in \mathbf{B}$ and, thus, $b \in \mathbf{A} \cap \mathbf{B}$. Consider $\mathbf{C} \in \text{LSets } \mathbf{P}$ such that

$$\mathbf{A} \cap \mathbf{B} \leq_{\text{LP}} \mathbf{C} \leq_{\text{LP}} \mathbf{A} \iff \mathbf{A} \cap \mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{A}$$

and

$$\mathbf{A} \cap \mathbf{B} \leq_{\text{LP}} \mathbf{C} \leq_{\text{LP}} \mathbf{B} \iff \mathbf{A} \cap \mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{B}.$$

Using the fact that intersection preserves inclusions, we have

$$\begin{aligned} & (\mathbf{A} \cap \mathbf{B}) \cap (\mathbf{A} \cap \mathbf{B}) \subseteq (\mathbf{C} \cap \mathbf{C}) \subseteq (\mathbf{A} \cap \mathbf{B}) \\ \iff & \mathbf{A} \cap \mathbf{B} \subseteq \mathbf{C} \subseteq (\mathbf{A} \cap \mathbf{B}) \\ \iff & \mathbf{C} = (\mathbf{A} \cap \mathbf{B}). \end{aligned} \tag{19}$$

Therefore, $\mathbf{A} \cap \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} .

We have therefore proved that $\text{LP} = \langle \text{LSets } \mathbf{P}, \leq_{\text{LP}} \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq \mathbf{C}$ for any $\mathbf{C} \in \text{LSets } \mathbf{P}$, meaning that \emptyset is the bottom. Furthermore, we notice that $\mathbf{C} \subseteq \mathbf{P}$ for any $\mathbf{C} \in \text{LSets } \mathbf{P}$, meaning that \mathbf{P} is a top. Therefore, the lattice is bounded. \square

Graded exercise I.4 (UpperLowerBounds)

Let $\mathbf{A} = \{a, b, c, d, e\}$. Give examples of the following situations using Hasse diagrams. In each case, provide a poset structure on \mathbf{A} and a subset $\mathbf{B} \subseteq \mathbf{A}$ such that:

1. \mathbf{B} has a least upper bound;
2. \mathbf{B} has a greatest lower bound;
3. \mathbf{B} has no least upper bound;
4. \mathbf{B} has no greatest lower bound.

31.4. Lattice homomorphisms

In this section, we want to abstract the concept of lattice and describe a category in which the objects are lattices themselves, and the morphisms are lattice homomorphisms. We call this category **Lat**.

Definition 31.15 (Lattice homomorphism)

Given two lattices P, Q , a *lattice homomorphism* is a map $f : P \rightarrow Q$ which preserves meets and joins:

$$\begin{aligned} f(p \wedge_P q) &= f(p) \wedge_Q f(q), \\ f(p \vee_P q) &= f(p) \vee_Q f(q). \end{aligned} \quad (20)$$

Example 31.16. We consider the lattices $P = \langle \text{Pow}\{\text{beer}, \text{wine}\}, \cap, \cup \rangle$, and $Q = \langle \{\text{beer}, \text{wine}\}, \max, \min \rangle$, where min (max) refer to the minimum (maximum) alcoholic content of the beverage (assuming Swiss beers, which have alcohol content lower than wine). Furthermore, consider

$$\begin{array}{c} f : \text{Pow}\{\text{beer}, \text{wine}\} \rightarrow \{\text{beer}, \text{wine}\}, \\ \mathbf{A} \mapsto \begin{cases} \text{beer}, & \text{if } \text{beer} \in \mathbf{A}, \\ \text{wine}, & \text{otherwise} \end{cases} \end{array} \quad (21)$$

The explicit evaluations of f are reported in Table 31.2.

Is f a lattice homomorphism? Yes. We can check it explicitly. Consider $\mathbf{A}, \mathbf{B} \subseteq \{\text{beer}, \text{wine}\}$. We need to show that

$$f(\mathbf{A} \cap \mathbf{B}) = \max\{f(\mathbf{A}), f(\mathbf{B})\} \quad (22)$$

and

$$f(\mathbf{A} \cup \mathbf{B}) = \min\{f(\mathbf{A}), f(\mathbf{B})\}. \quad (23)$$

Technically, we can check every possible pair of \mathbf{A}, \mathbf{B} (only 16 for this case), but that's not efficient. First, consider $f(\mathbf{A} \cap \mathbf{B}) = \text{wine}$. Following (21), this means $\text{beer} \notin \mathbf{A} \cap \mathbf{B}$ (in other words, either $\text{beer} \notin \mathbf{A}$, $\text{beer} \notin \mathbf{B}$, or both). At least one of $f(\mathbf{A})$ and $f(\mathbf{B})$ is wine , because

$$\frac{\text{beer} \notin \mathbf{A}}{f(\mathbf{A}) = \text{wine}} \quad \text{and} \quad \frac{\text{beer} \notin \mathbf{B}}{f(\mathbf{B}) = \text{wine}}. \quad (24)$$

This implies $\max\{f(\mathbf{A}), f(\mathbf{B})\} = \text{wine}$, which verifies (22).

If instead, we have $f(\mathbf{A} \cap \mathbf{B}) = \text{beer}$, then $\text{beer} \in \mathbf{A} \cap \mathbf{B}$, meaning that $\text{beer} \in \mathbf{A}$ and $\text{beer} \in \mathbf{B}$. Therefore, $\max\{f(\mathbf{A}), f(\mathbf{B})\} = \text{beer}$, which verifies (22).

Condition (23) can be verified analogously.

The notion of lattice homomorphism can be extended to bounded lattices.

Definition 31.17 (Bounded lattice homomorphism)

Given two bounded lattices P, Q , a *bounded lattice homomorphism* is a lattice homomorphism $f : P \rightarrow Q$ which also preserves top and bottom:

$$\begin{aligned} f(\perp_P) &= \perp_Q, \\ f(\top_P) &= \top_Q. \end{aligned} \quad (25)$$

Note that (bounded) lattice homomorphisms are necessarily monotone.

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

A	$f(A)$
\emptyset	
$\{\text{beer}\}$	
$\{\text{wine}\}$	
$\{\text{beer}, \text{wine}\}$	

Table 31.2.

31.5. Categories **Lat** and **BoundedLat**

Definition 31.18 (Category **Lat**)

The category **Lat** is defined by:

1. *Objects*: The objects of this category are all lattices.
2. *Morphisms*: The morphisms from a lattice X to a lattice Y are the lattice homomorphisms from X to Y .
3. *Identity morphism*: The identity morphism for the lattice X is the identity map id_X .
4. *Composition operation*: The composition operation is composition of maps.

Definition 31.19 (Category **BoundedLat**)

The category **BoundedLat** is defined by:

1. *Objects*: The objects of this category are all bounded lattices.
2. *Morphisms*: The morphisms from a lattice X to a lattice Y are the lattice homomorphisms from X to Y .
3. *Identity morphism*: The identity morphism for the bounded lattice X is the identity map id_X .
4. *Composition operation*: The composition operation is composition of maps.

Exercise52. Show that **Lat** is a category.

See solution on page 465.

Exercise53. Show that **BoundedLat** is a category.

See solution on page 465.



32. Lattice structure of DPs

In the previous chapter we have talked about the posetal structure of hom-sets. In **DP** hom-sets are also posets: morphisms can be ordered, and this order is preserved by composition. Moreover, it also has a lattice structure that is preserved by composition.

32.1 Ordering DPs	450
32.2 Interaction with series composition	451
32.3 Union of Design Problems	452
32.4 Intersection of Design Problems	453
32.5 Lattice structure of DP hom-sets	454
32.6 Interaction with composition	457

32.1. Ordering DPs

Definition 32.1 (Order on DP)

Suppose that \mathbf{P} and \mathbf{Q} are posets, and that $\mathbf{d}, \mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$ are design problems. We define the order as follows:

$$\frac{\mathbf{d} \leq_{\text{DP}} \mathbf{e}}{\mathbf{d}(p^*, q) \leq_{\text{Bool}} \mathbf{e}(p^*, q) \text{ for all } p \in \mathbf{P}, q \in \mathbf{Q}}, \quad (1)$$

Remark 32.2. Recall that design problems are monotone functions, and note that the order defined in Def. 32.1 is just the usual order on monotone functions.

We diagrammatically represent the relation $\mathbf{d} \leq_{\text{DP}} \mathbf{e}$ as in Fig. 1.



Figure 1.: The design problem \mathbf{d} implies the design problem \mathbf{e} .

32.2. Interaction with series composition

In the previous section, we introduced the concept of order in \mathbf{DP} , and proved that the hom-sets of \mathbf{DP} form a bounded lattice. In this section, we show that composition (Def. 15.6) of design problems is an order-preserving operation.

Lemma 32.3. Given $\mathbf{d}, \mathbf{e} \in \text{Hom}_{\mathbf{DP}}(\mathbf{P}, \mathbf{Q})$ and $\mathbf{g}, \mathbf{h} \in \text{Hom}_{\mathbf{DP}}(\mathbf{Q}; \mathbf{R})$ we have:

$$\frac{\mathbf{d} \leq_{\mathbf{DP}} \mathbf{e} \quad \mathbf{g} \leq_{\mathbf{DP}} \mathbf{h}}{(\mathbf{d} ; \mathbf{g}) \leq_{\mathbf{DP}} (\mathbf{e} ; \mathbf{h})}. \quad (2)$$

In other words, series composition is order-preserving on \mathbf{DP} .

Proof. We have

$$\begin{aligned} & (\mathbf{d} ; \mathbf{g})(\mathbf{p}^*, \mathbf{r}) \\ &= \bigvee_{q \in \mathbf{Q}} \mathbf{d}(\mathbf{p}^*, q) \wedge \mathbf{g}(q^*, \mathbf{r}) \\ &\leq_{\mathbf{DP}} \bigvee_{q \in \mathbf{Q}} \mathbf{e}(\mathbf{p}^*, q) \wedge \mathbf{h}(q^*, \mathbf{r}) \\ &= (\mathbf{e} ; \mathbf{h})(\mathbf{p}^*, \mathbf{r}). \end{aligned} \quad (3)$$

Therefore, \circ is order-preserving on \mathbf{DP} . \square

32.3. Union of Design Problems

Let $\mathbf{d} : \mathbf{P} \rightarrow \mathbf{Q}$ and $\mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$ be design problems. We define the *union* $\mathbf{d} \vee \mathbf{e}$ to be the design problem which is feasible whenever *either* \mathbf{d} or \mathbf{e} is feasible. This models \mathbf{d} and \mathbf{e} as interchangeable technologies: either we can replace the other.

Definition 32.4 (Union of design problems)

Given two design problems $\mathbf{d} : \mathbf{P} \rightarrow \mathbf{Q}$ and $\mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$, their *union* $\mathbf{d} \vee \mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$ is defined by

$$\begin{aligned} (\mathbf{d} \vee \mathbf{e}) : \mathbf{P}^{\text{op}} \times \mathbf{Q} &\rightarrow \text{Pos Bool}, \\ \langle p^*, q \rangle &\mapsto \mathbf{d}(p^*, q) \vee \mathbf{e}(p^*, q). \end{aligned} \quad (4)$$

The union of design problems is represented as in Fig. 2.

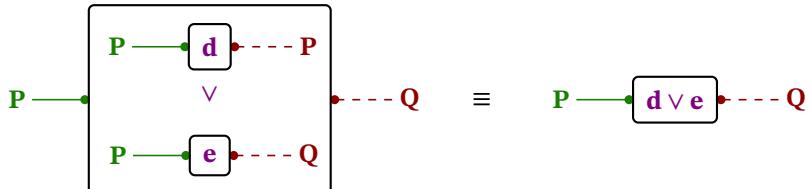


Figure 2.: Diagrammatic representation of the union of design problems.

32.4. Intersection of Design Problems

Given two design problems $\mathbf{d}, \mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$, we can define a design problem $\mathbf{d} \wedge \mathbf{e}$ that is feasible if and only if \mathbf{d} and \mathbf{e} are both feasible. We call $\mathbf{d} \wedge \mathbf{e}$ the *intersection* of \mathbf{d} and \mathbf{e} . One interpretation of $\mathbf{d} \wedge \mathbf{e}$ is that \mathbf{d} and \mathbf{e} are two slightly different models of the same process, and we want to make sure that the design is conservatively feasible for both models.

Definition 32.5 (Intersection of design problems)

Given design problems $\mathbf{d} : \mathbf{P} \rightarrow \mathbf{Q}$ and $\mathbf{e} : \mathbf{P} \rightarrow \mathbf{Q}$, their *intersection* is denoted $(\mathbf{d} \wedge \mathbf{e}) : \mathbf{P} \rightarrow \mathbf{Q}$, defined by:

$$\begin{aligned} (\mathbf{d} \wedge \mathbf{e}) : \mathbf{P}^{\text{op}} \times \mathbf{Q} &\rightarrow \text{Pos Bool}, \\ \langle p^*, q \rangle &\mapsto \mathbf{d}(p^*, q) \wedge \mathbf{e}(p^*, q). \end{aligned} \quad (5)$$

The intersection of design problems is represented as in Fig. 3.

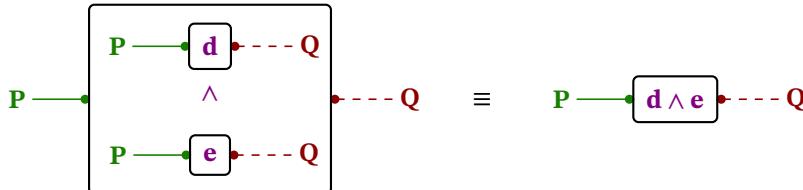


Figure 3.: Diagrammatic representation of the intersection of design problems.

We can directly generalize the intersection $\mathbf{d} \wedge \mathbf{e}$ by allowing \mathbf{d} and \mathbf{e} to have different domain and codomains, $\mathbf{d} : \mathbf{P} \rightarrow \mathbf{Q}$ and $\mathbf{e} : \mathbf{R} \rightarrow \mathbf{S}$. We call this putting two design problems “in parallel”.

32.5. Lattice structure of DP hom-sets

Given the definitions of \wedge and \vee in the previous sections, we can prove that every **DP** hom-sets have a lattice structure.

This lattice is bounded by a “true” and a “false” DP.

Definition 32.6 (False and true DP)

Given any two partial orders P, Q , we can define a *false* DP as

$$\begin{aligned}\perp_{P,Q} : P^{\text{op}} \times Q &\rightarrow \text{Pos Bool}, \\ \langle p^*, q \rangle &\mapsto \perp.\end{aligned}$$

We can define a *true* DP as

$$\begin{aligned}\top_{P,Q} : P^{\text{op}} \times Q &\rightarrow \text{Pos Bool}, \\ \langle p^*, q \rangle &\mapsto \top.\end{aligned}$$

For any functionality-resource pair P, Q , these represent the design problem which is never (respectively always) feasible.

Lemma 32.7. $\text{Hom}_{\text{DP}}(P; Q)$ is a bounded lattice with union \vee as join, intersection \wedge as meet, top $\top_{P,Q}$ and bottom $\perp_{P,Q}$.

Proof. First, we need to prove that $\text{Hom}_{\text{DP}}(P; Q)$ is a poset. To prove this, we check the following:

▷ *Reflexivity*: Given $d \in \text{Hom}_{\text{DP}}(P; Q)$:

$$\frac{}{\overline{\begin{array}{c} \top \\ d \leq_{\text{DP}} d \end{array}}}; \quad (6)$$

▷ *Antisymmetry*: Given $d, e \in \text{Hom}_{\text{DP}}(P; Q)$:

$$\frac{d \leq_{\text{DP}} e \quad e \leq_{\text{DP}} d}{d = e}; \quad (7)$$

▷ *Transitivity*: Given $d, e, g \in \text{Hom}_{\text{DP}}(P; Q)$:

$$\frac{d \leq_{\text{DP}} e \quad e \leq_{\text{DP}} g}{d \leq_{\text{DP}} g}. \quad (8)$$

Therefore, $\text{Hom}_{\text{DP}}(P; Q)$ is a poset. Furthermore, consider two design problems $d, e \in \text{Hom}_{\text{DP}}(P; Q)$. Their greatest lower bound (meet) is $d \wedge e$, since it is the greatest design problem implying both d and e . Their least upper bound (join), instead, is $d \vee e$, since it is the least design problem implied by both d and e . This proves that Hom_{DP} is a lattice. To prove that it is bounded, we identify the top element as $\top_{P,Q}$ (it is implied by all other design problems) and the bottom element as $\perp_{P,Q}$ (it implies by all the other design problems). \square

We show that a **DP** hom-set is a *complete lattice*.

Definition 32.8 (Complete Lattice)

A poset $P = \langle P, \leq_P \rangle$ is a *complete lattice* if every subset S of P has both a *greatest lower bound* (often referred to as the *infimum, meet*) and a *least upper*

bound (often referred to as the *supremum, join*).

Example 32.9. Consider the power set of any given set, ordered by inclusion. The supremum of any two subsets is given by their union. The infimum of any two subsets is given by their intersection.

Lemma 32.10 (DP hom-sets are complete lattices). Hom-sets of **DP** are complete lattices.

Proof. Consider any $\mathbf{P}, \mathbf{Q} \in \text{Ob}_{\mathbf{DP}}$ and $\text{Hom}_{\mathbf{DP}}(\mathbf{P}; \mathbf{Q})$. We have already shown that $\mathbf{A} = \text{Hom}_{\mathbf{DP}}(\mathbf{P}; \mathbf{Q})$ is a bounded lattice (Lemma 32.7). Now, take any subset \mathbf{B} of \mathbf{A} . We define the following two design problems:

$$\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}: \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow_{\text{Pos}} \text{Bool}, \quad (9)$$

$$\langle p, q \rangle \mapsto \exists \mathbf{d} \in \mathbf{B}: \mathbf{d}(p^*, q),$$

and

$$\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}: \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow_{\text{Pos}} \text{Bool}, \quad (10)$$

$$\langle p^*, q \rangle \mapsto \forall \mathbf{d} \in \mathbf{B}: \mathbf{d}(p^*, q).$$

These are clearly design problems (given that \mathbf{d} is a design problem) and given their signature they belong to \mathbf{A} . We will now argue that $\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the supremum of \mathbf{B} and $\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the infimum of \mathbf{B} .

$\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the supremum of \mathbf{B} : First, for any $\mathbf{d} \in \mathbf{B}$, we know that $\mathbf{d} \leq_{\mathbf{DP}} \bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$, proving that $\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is an upper bound of \mathbf{B} . We now want to show that $\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the least upper bound of \mathbf{B} : for any upper bound \mathbf{e} of \mathbf{B} , we need to show $\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d} \leq_{\mathbf{DP}} \mathbf{e}$. In other words, for any pair $\langle p^*, q \rangle \in \mathbf{P}^{\text{op}} \times \mathbf{Q}$, we need to show $(\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q) \leq_{\text{Bool}} \mathbf{e}(p^*, q)$. Fix any $\langle p^*, q \rangle$. If $(\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q) = \perp$, the condition is trivially satisfied.

If, instead, $(\bigvee_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q) = \top$, there exists a $\mathbf{d} \in \mathbf{B}$ such that $\mathbf{d}(p^*, q) = \top$. Given that \mathbf{e} is an upper bound of \mathbf{B} , this implies $\top = \mathbf{d}(p^*, q) \leq_{\text{Bool}} \mathbf{e}(p^*, q) = \top$, proving the condition.

$\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the infimum of \mathbf{B} : First, for any $\mathbf{d} \in \mathbf{B}$, we know that $\mathbf{d} \wedge \bigwedge_{\mathbf{d} \in \mathbf{B} \setminus \mathbf{d}} \mathbf{d} = \bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d} \leq_{\mathbf{DP}} \mathbf{d}$, proving that $\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is a lower bound of \mathbf{B} . We now want to show that $\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$ is the greatest lower bound of \mathbf{B} : for any lower bound \mathbf{e} of \mathbf{B} , we need to show $\mathbf{e} \leq_{\mathbf{DP}} \bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d}$. In other words, for any pair $\langle p^*, q \rangle \in \mathbf{P}^{\text{op}} \times \mathbf{Q}$, we need to show $\mathbf{e}(p^*, q) \leq_{\text{Bool}} (\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q)$. Fix any $\langle p^*, q \rangle$. If $(\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q) = \perp$, the condition is trivially satisfied. If, instead, $(\bigwedge_{\mathbf{d} \in \mathbf{B}} \mathbf{d})(p^*, q) = \top$, there is at least one $\mathbf{d} \in \mathbf{B}$ for which $\mathbf{d}(p^*, q) = \top$. Given that \mathbf{e} is a lower bound of \mathbf{B} , this implies $\perp = \mathbf{e}(p^*, q) \leq_{\text{Bool}} \mathbf{d}(p^*, q) = \top$, proving the condition. \square

Definition 32.11 (Distributive Lattice)

A lattice $\mathbf{P} = \langle \mathbf{P}, \wedge, \vee \rangle$ is a *distributive lattice* if for all $x, y, z \in \mathbf{P}$:

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z). \quad (11)$$

Remark 32.12. Note that condition (11) is equivalent to its dual:

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z), \quad (12)$$

for all $x, y, z \in P$.

Lemma 32.13. Consider $\mathbf{d}, \mathbf{e}, \mathbf{g} \in \text{Hom}_{\mathbf{DP}}(\mathbf{P}; \mathbf{Q})$. We have

$$(\mathbf{d} \wedge \mathbf{e}) \vee \mathbf{g} = (\mathbf{d} \vee \mathbf{g}) \wedge (\mathbf{e} \vee \mathbf{g}). \quad (13)$$

Proof. We have:

$$\begin{aligned} & ((\mathbf{d} \wedge \mathbf{e}) \vee \mathbf{g})(p^*, q) \\ &= (\mathbf{d} \wedge \mathbf{e})(p^*, q) \vee \mathbf{g}(p^*, q) \\ &= (\mathbf{d}(p^*, q) \wedge \mathbf{e}(p^*, q)) \vee \mathbf{g}(p^*, q) \\ &= (\mathbf{d}(p^*, q) \vee \mathbf{g}(p^*, q)) \wedge (\mathbf{e}(p^*, q) \vee \mathbf{g}(p^*, q)) \\ &= ((\mathbf{d} \vee \mathbf{g}) \wedge (\mathbf{e} \vee \mathbf{g}))(p^*, q). \end{aligned} \quad (14)$$

□

Lemma 32.14. Consider $\mathbf{d}, \mathbf{e}, \mathbf{g} \in \text{Hom}_{\mathbf{DP}}(\mathbf{P}; \mathbf{Q})$. We have

$$(\mathbf{d} \vee \mathbf{e}) \wedge \mathbf{g} = (\mathbf{d} \wedge \mathbf{g}) \vee (\mathbf{e} \wedge \mathbf{g}). \quad (15)$$

Proof. We have:

$$\begin{aligned} & ((\mathbf{d} \vee \mathbf{e}) \wedge \mathbf{g})(p^*, q) \\ &= (\mathbf{d} \vee \mathbf{e})(p^*, q) \wedge \mathbf{g}(p^*, q) \\ &= (\mathbf{d}(p^*, q) \vee \mathbf{e}(p^*, q)) \wedge \mathbf{g}(p^*, q) \\ &= (\mathbf{d}(p^*, q) \wedge \mathbf{g}(p^*, q)) \vee (\mathbf{e}(p^*, q) \wedge \mathbf{g}(p^*, q)) \\ &= ((\mathbf{d} \wedge \mathbf{g}) \vee (\mathbf{e} \wedge \mathbf{g}))(p^*, q). \end{aligned} \quad (16)$$

□

Lemma 32.15 (DP hom-sets are distributive lattices). Hom-sets of \mathbf{DP} are distributive lattices.

Proof. Either Lemma 32.13 or Lemma 32.14 prove the statement. □

32.6. Interaction with composition

Furthermore, we show that all composition operations preserve joins, and all composition operations except trace preserve meets.

Series composition

Lemma 32.16. Consider $\mathbf{d}, \mathbf{e} \in \text{Hom}_{\text{DP}}(\mathbf{P}; \mathbf{Q})$ and $\mathbf{g} \in \text{Hom}_{\text{DP}}(\mathbf{Q}; \mathbf{R})$. We have

$$(\mathbf{d} \vee \mathbf{e}) ; \mathbf{g} = (\mathbf{d} ; \mathbf{g}) \vee (\mathbf{e} ; \mathbf{g}). \quad (17)$$

This is diagrammatically represented in Fig. 4.



Figure 4.

Proof. We have:

$$\begin{aligned} & ((\mathbf{d} \vee \mathbf{e}) ; \mathbf{g})(p^*, r) \\ &= \bigvee_{q \in Q} (\mathbf{d} \vee \mathbf{e})(p^*, q) \wedge \mathbf{g}(q^*, r) \\ &= \bigvee_{q \in Q} (\mathbf{d}(p^*, q) \vee \mathbf{e}(p^*, q)) \wedge \mathbf{g}(q^*, r) \\ &= \bigvee_{q \in Q} (\mathbf{d}(p^*, q) \wedge \mathbf{g}(q^*, r)) \vee (\mathbf{e}(p^*, q) \wedge \mathbf{g}(q^*, r)) \\ &= ((\mathbf{d} ; \mathbf{g}) \vee (\mathbf{e} ; \mathbf{g}))(p^*, r). \end{aligned} \quad (18)$$

□

Remark 32.17. Consider $\mathbf{d}, \mathbf{e} \in \text{Hom}_{\text{DP}}(\mathbf{P}; \mathbf{Q})$ and $\mathbf{g}, \mathbf{h} \in \text{Hom}_{\text{DP}}(\mathbf{Q}; \mathbf{R})$. In general, we have:

$$(\mathbf{d} \vee \mathbf{e}) ; (\mathbf{g} \vee \mathbf{h}) \neq (\mathbf{d} ; \mathbf{g}) \vee (\mathbf{e} ; \mathbf{h}). \quad (19)$$

Indeed, consider $\mathbf{d} = \top_{\mathbf{P}, \mathbf{Q}}$, $\mathbf{e} = \perp_{\mathbf{P}, \mathbf{Q}}$, $\mathbf{g} = \perp_{\mathbf{Q}, \mathbf{R}}$, and $\mathbf{h} = \top_{\mathbf{Q}, \mathbf{R}}$. Clearly:

$$\begin{aligned} ((\mathbf{d} \vee \mathbf{e}) ; (\mathbf{g} \vee \mathbf{h}))(p^*, r) &= \bigvee_{q \in Q} (\mathbf{d} \vee \mathbf{e})(p^*, q) \wedge (\mathbf{g} \vee \mathbf{h})(q^*, r) \\ &= \top, \end{aligned} \quad (20)$$

but

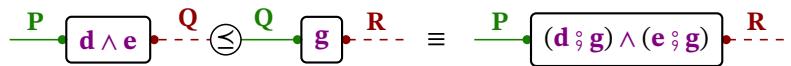
$$\begin{aligned} & ((\mathbf{d} ; \mathbf{g}) \vee (\mathbf{e} ; \mathbf{h}))(p^*, r) \\ &= \left(\bigvee_{q \in Q} \mathbf{d}(p^*, q) \wedge \mathbf{g}(q^*, r) \right) \vee \left(\bigvee_{q \in Q} \mathbf{e}(p^*, q) \wedge \mathbf{h}(q^*, r) \right) \\ &= \perp \vee \perp \\ &= \perp. \end{aligned} \quad (21)$$

Lemma 32.18. Consider $\mathbf{d}, \mathbf{e} \in \text{Hom}_{\text{DP}}(\mathbf{P}; \mathbf{Q})$ and $\mathbf{g} \in \text{Hom}_{\text{DP}}(\mathbf{Q}; \mathbf{R})$. We have

$$(\mathbf{d} \wedge \mathbf{e}) ; \mathbf{g} = (\mathbf{d} ; \mathbf{g}) \wedge (\mathbf{e} ; \mathbf{g}). \quad (22)$$

This is diagrammatically represented in Fig. 5.

Figure 5.



Proof. We have:

$$\begin{aligned}
 & ((\mathbf{d} \wedge \mathbf{e}) \circ \mathbf{g})(p^*, r) \\
 &= \bigvee_{q \in Q} (\mathbf{d} \wedge \mathbf{e})(p^*, q) \wedge \mathbf{g}(q^*, r) \\
 &= \bigvee_{q \in Q} (\mathbf{d}(p^*, q) \wedge \mathbf{e}(p^*, q)) \wedge \mathbf{g}(q^*, r) \\
 &= \bigvee_{q \in Q} (\mathbf{d}(p^*, q) \wedge \mathbf{g}(q^*, r)) \wedge (\mathbf{e}(p^*, q) \wedge \mathbf{g}(q^*, r)) \\
 &= ((\mathbf{d} \circ \mathbf{g}) \wedge (\mathbf{e} \circ \mathbf{g}))(p^*, r).
 \end{aligned} \tag{23}$$

□



33. Constructing design problems

We have previously seen ways to construct posets (Chapter 6) and categories (Chapter 16) from simpler pieces. Analogously, this chapter describes some ways to construct design problems.

33.1 Companion and conjoint use	460
33.2 Companions and conjoint	462
33.3 Monoidal DPs	464

33.1. Union and intersection with companions and conjoints

We can also re-define the sum \vee and intersection \wedge using companions and conjoints, which allows us to introduce some useful constructions.

Definition 33.1 (Diagonal function)

Define the *diagonal function* $\text{diag}_P : P \rightarrow P \times P$:

$$\begin{aligned} \text{diag}_P : P &\rightarrow P \times P, \\ p &\mapsto \langle p, p \rangle. \end{aligned} \tag{1}$$

Definition 33.2 (Codiagonal function)

Define the *codiagonal function* $\text{codiag}_P : P + P \rightarrow P$:

$$\begin{aligned} \text{codiag}_P : P + P &\rightarrow P, \\ \langle 1, p \rangle &\mapsto p, \\ \langle 2, p \rangle &\mapsto p. \end{aligned} \tag{2}$$

Using the diagonal function, (4) can be rewritten as the following lemma.

Lemma 33.3. Given $\mathbf{d}, \mathbf{e} : P \leftrightarrow Q$, we have:

$$\mathbf{d} \vee \mathbf{e} = \text{conj}(\text{diag}_P) ; (\mathbf{d} + \mathbf{e}) ; \text{comp}(\text{diag}_Q). \tag{3}$$

Proof. First, note that

$$\begin{aligned} \text{conj}(\text{diag}_P) : P &\leftrightarrow P + P \\ \langle p_1^*, \langle 1, p_2 \rangle \rangle &\mapsto p_1 \leq p_2 \\ \langle p_1^*, \langle 1, p_3 \rangle \rangle &\mapsto p_1 \leq p_3 \end{aligned} \tag{4}$$

and

$$\begin{aligned} \text{comp}(\text{diag}_Q) : Q + Q &\leftrightarrow Q \\ \langle \langle 1, q_1^*, q_3 \rangle \rangle &\mapsto q_1 \leq q_3 \\ \langle \langle 2, q_2^*, q_3 \rangle \rangle &\mapsto q_2 \leq q_3. \end{aligned} \tag{5}$$

We start by looking at $\underbrace{\text{conj}(\text{diag}_P) ; (\mathbf{d} + \mathbf{e})}_{\star} : P \leftrightarrow Q + Q$.

$$\begin{aligned} \star &(\langle p^*, q \rangle) \\ &= \bigvee_{p' \in P+P} \text{conj}(\text{diag}_P)(\langle p^*, p' \rangle) \wedge (\mathbf{d} + \mathbf{e})(\langle p'^*, q \rangle) \\ &= \left(\bigvee_{\langle 1, p' \rangle \in P+P} (p \leq p') \wedge \mathbf{d}(p'^*, q) \right) \vee \left(\bigvee_{\langle 2, p' \rangle \in P+P} (p \leq p') \wedge \mathbf{e}(p'^*, q) \right) \\ &= \mathbf{d}(p^*, q) \vee \mathbf{e}(p^*, q). \end{aligned} \tag{6}$$

We now look at $\star \circ \text{comp}(\text{diag}_Q) : P \rightarrow Q$:

$$\begin{aligned}
 & (\star \circ \text{comp}(\text{diag}_Q))(p^*, q') \\
 &= \bigvee_{q \in Q+Q} \star(p^*, q) \wedge \text{comp}(\text{diag}_Q)(q^*, q') \\
 &= \left(\bigvee_{\langle 1, q \rangle \in Q+Q} \mathbf{d}(p^*, q) \wedge (q \leq q') \right) \vee \left(\bigvee_{\langle 2, q \rangle \in Q+Q} \mathbf{e}(p^*, q) \wedge (q \leq q') \right) \quad (7) \\
 &= \mathbf{d}(p^*, q') \vee \mathbf{e}(p^*, q').
 \end{aligned}$$

□

33.2. Companions and conjoint

We round out our discussion of **DP** by introducing two formulas for transforming monotone maps in **Pos** into design problems in **DP**. Each monotone map f can be transformed into two design problems, called its *companion* $\text{comp}(f)$ and *conjoint* $\text{conj}(f)$. Many of the design problems that we have introduced can be realized as companions and conjoints of appropriate monotone maps.

Definition 33.4 (Companion and conjoint)

Let P and Q be posets, and suppose that $f : P \rightarrow_{\text{Pos}} Q$ is a monotone map. We define its *companion* in **DP**, denoted $\text{comp}(f) : P \rightarrow_{\text{DP}} Q$, and its *conjoint*, denoted $\text{conj}(f) : Q \rightarrow_{\text{DP}} P$ as

$$\text{comp}(f)(p^*, q) := f(p) \leq_Q q \quad \text{and} \quad \text{conj}(f)(q^*, p) := q \leq_P f(p). \quad (8)$$

Lemma 33.5. Both the companion and conjoint constructions from Def. 33.4 are functorial from **Pos** to **DP**: they preserve identities and composition.

Proof. We will show that the companion and conjoint are functors of the following forms:

$$\text{comp} : \text{Pos} \rightarrow \text{DP} \quad \text{and} \quad \text{conj} : \text{Pos} \rightarrow \text{DP}^{\text{op}}. \quad (9)$$

First, we see that they send the identity monotone maps id_P to the identity design problem id_P for any poset P , because

$$\text{comp}(\text{id}_P)(p_1^*, p_2) = (p_1 \leq_P p_2) = \text{conj}(\text{id}_P)(p_1^*, p_2). \quad (10)$$

Now suppose that $f : P \rightarrow_{\text{Pos}} Q$ and $g : Q \rightarrow_{\text{Pos}} R$ are given. We first show that $\text{conj}(g) ; \text{conj}(f) = \text{conj}(f ; g)$. For any $p \in P$ and $r \in R$, we have

$$\begin{aligned} & (\text{conj}(g) ; \text{conj}(f))(p^*, r) \\ &= \bigvee_{q \in Q} \text{conj}(g)(r^*, q) \wedge \text{conj}(f)(q^*, p) \\ &= \bigvee_{q \in Q} (r \leq_R g(q)) \wedge (q \leq_Q f(p)) \\ &= r \leq_R g(f(p)) \\ &= (\text{conj}(f ; g))(r^*, p). \end{aligned} \quad (11)$$

Similarly, we can prove that $\text{comp}(f) ; \text{comp}(g) = \text{comp}(f ; g)$:

$$\begin{aligned} & (\text{comp}(f) ; \text{comp}(g))(p^*, r) \\ &= \bigvee_{q \in Q} \text{comp}(f)(p^*, q) \wedge \text{comp}(g)(q^*, r) \\ &= \bigvee_{q \in Q} (g(p) \leq_Q q) \wedge (g(q) \leq_R r) \\ &= g(f(p)) \leq_R r \\ &= (\text{comp}(f ; g))(p^*, r). \end{aligned} \quad (12)$$

□

Example 33.6. The identity design problem $\text{id}_P : P \rightarrow_{\text{DP}} P$ is the companion

(and the conjoint) of the identity map $\text{id}_P^{\text{Pos}} : P \rightarrow_{\text{Pos}} P$. This is easy to check, as

$$\begin{aligned} \text{comp}(\text{id}_P)(p_1^*, p_2) \\ = \text{id}_P(p_1) \leq p_2 \\ = p_1 \leq p_2 \\ = \text{id}_P(p_1^*, p_2). \end{aligned} \tag{13}$$

Example 33.7. The coproduct injections ι_P, ι_Q for design problems are the companions of the coproduct injections for the disjoint union.

Example 33.8. The product projections π_P, π_Q for design problems are the conjoints of the coproduct injections for the disjoint union.

Deriving terminators

Using companion and conjoint we can obtain the equivalent of “terminators” representing constant functionality/resources. Consider an element x of a poset P . We can represent this constant element as a map f_x from the singleton to the poset:

$$\begin{aligned} f_x : 1 &\rightarrow_{\text{Pos}} P, \\ \bullet &\mapsto x. \end{aligned} \tag{14}$$

By taking the companion of f_x we get

$$\begin{aligned} \text{comp}(f_x) : 1 &\dashrightarrow P, \\ \langle \bullet, p \rangle &\mapsto (x \leq_P p). \end{aligned} \tag{15}$$

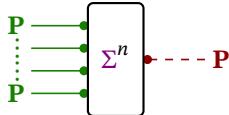
By taking the conjoint, we get

$$\begin{aligned} \text{conj}(f_x) : P &\dashrightarrow 1, \\ \langle p^*, \bullet \rangle &\mapsto (p \leq_P x). \end{aligned} \tag{16}$$

These two cases represent design problems with either *constant* resources or constant, functionalities, respectively.

33.3. Monoidal DPs

If the underlying posets of functionality and resources are monoidal (Def. 31.1), then we can define some canonical DPs.

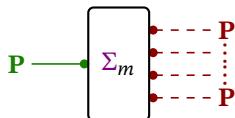


Definition 33.9 (Sum of resources for monoidal posets)

If the poset \mathbf{P} is monoidal with monoidal product \otimes , then the “sum” of n copies of \mathbf{P} is a design problem given by

$$\begin{aligned} \Sigma^n : (\mathbf{P}^n)^{\text{op}} \times \mathbf{P} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle \langle p_1, \dots, p_n \rangle^*, q \rangle &\mapsto (p_1 \otimes \dots \otimes p_n \leq_{\mathbf{P}} q). \end{aligned} \quad (17)$$

We can do the symmetric construction.



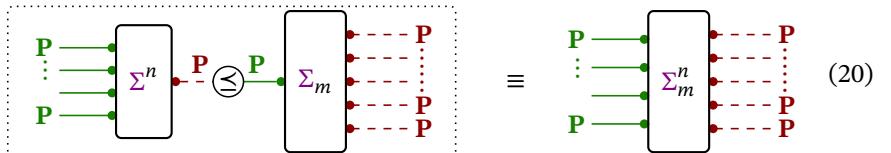
Definition 33.10 (Sum of functionalities for monoidal posets)

If the poset \mathbf{P} is monoidal with monoidal product \otimes , then the “sum” of m copies of \mathbf{P} is a design problem given by

$$\begin{aligned} \Sigma_m : \mathbf{P}^{\text{op}} \times (\mathbf{P}^m) &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle p^*, \langle q_1, \dots, q_m \rangle \rangle &\mapsto p \leq_{\mathbf{P}} (q_1 \otimes \dots \otimes q_m). \end{aligned} \quad (18)$$

We can now put these in series to obtain the generic DP Σ_m^n with n functionalities and m resources:

$$\Sigma_m^n = \Sigma^n ; \Sigma_m. \quad (19)$$



Note that this works with addition, but also with other associative operations, such as multiplication, max, etc.

Solutions to selected exercises

Solution of Exercise 52. Clearly, given any lattice X , the identity map id_X is a lattice homomorphism, since

$$\begin{aligned}\text{id}_X(x_1 \wedge_X x_2) &= x_1 \wedge_X x_2 \\ \text{id}_X(x_1 \vee_X x_2) &= x_1 \vee_X x_2.\end{aligned}\tag{21}$$

This said, the identity map satisfies unitality. Now, given lattice homomorphisms

$$f : X \rightarrow Y \quad \text{and} \quad g : Y \rightarrow Z,\tag{22}$$

their composition is a lattice homomorphism, since

$$\begin{aligned}(f \circ g)(x_1 \wedge_X x_2) &= f(x_1 \wedge_X x_2) \circ g \\ &= g(f(x_1) \wedge_Y f(x_2)) \\ &= (f \circ g)(x_1) \wedge_Z (f \circ g)(x_2),\end{aligned}\tag{23}$$

and

$$\begin{aligned}(f \circ g)(x_1 \vee_X x_2) &= f(x_1 \vee_X x_2) \circ g \\ &= g(f(x_1) \vee_Y f(x_2)) \\ &= (f \circ g)(x_1) \vee_Z (f \circ g)(x_2),\end{aligned}\tag{24}$$

We have already checked in the past the map composition is associative (e.g., when checking that **Set** and **Pos** are categories).

Solution of Exercise 53. Consider the solution of Exercise 52 as a starting point. Clearly, given any lattice X , the identity map id_X is also a bounded lattice homomorphism, since

$$\begin{aligned}\text{id}_X(\perp_X) &= \perp_X \\ \text{id}_X(\top_X) &= \top_X.\end{aligned}\tag{25}$$

This said, the identity map satisfies unitality. Now, given lattice homomorphisms $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, their composition is also a bounded lattice homomorphism, since

$$\begin{aligned}(f \circ g)(\perp_X) &= g(\perp_Y) \\ &= \perp_Z,\end{aligned}\tag{26}$$

and

$$\begin{aligned}(f \circ g)(\top_X) &= g(\top_Y) \\ &= \top_Z.\end{aligned}\tag{27}$$

We have already checked in the past the map composition is associative (e.g., when checking that **Set** and **Pos** are categories).

PART J. UNIVERSAL PROPERTIES



34. Sameness

469

The Laendler is a folk dance in 3/4 time, popular in Switzerland, Austria, Bavaria, and Slovenia. It is a partner dance, in which one hops and stamps a lot. It can purely instrumental, but also include a vocal part, featuring yodeling.



34. Sameness

34.1 Sameness in category theory . . . 470

34.2 Isomorphism is not identity . . . 473

Schaumkuesse are confections consisting of a biscuit, topped with creamy filling, and then coated in a hard shell of pure milk chocolate.

34.1. Sameness in category theory

One nice thing about the category of sets is that we are all used to working with sets and functions. And many concepts that are familiar in the setting of sets and functions can actually be reformulated in a way which makes sense for lots of other categories, if not for all categories. It can be fun, and insightful, to see known definitions transformed into “category theory language”. For example: the notion of a bijective function is a familiar concept. There are least two ways of saying what it means for a function $f : \mathbf{A} \rightarrow \mathbf{B}$ of sets to be bijective:

Definition 1: “ $f : \mathbf{A} \rightarrow \mathbf{B}$ is bijective if, for every $y \in \mathbf{B}$ there exists precisely one $x \in \mathbf{A}$ such that $f(x) = y$;

Definition 2: “ $f : \mathbf{A} \rightarrow \mathbf{B}$ is bijective if there exists a function $g : \mathbf{B} \rightarrow \mathbf{A}$ such that $f \circ g = \text{id}_{\mathbf{A}}$ and $g \circ f = \text{id}_{\mathbf{B}}$ ”.

It is a short proof to show that the above two definitions are equivalent. The first definition, however, does not lend itself well to generalization in category theory, because it is formulated using something that is very specific to sets: namely, it refers to *elements* of the sets \mathbf{A} and \mathbf{B} . And we have seen that the objects of a category need not be sets, and so in general we cannot speak of “elements” in the usual sense. Definition 2, on the other hand, can easily be generalized to work in any category. To formulate this version, all we need are morphisms, their composition, the notion of identity morphisms, and the notion of equality of morphisms (for equations such as “ $f \circ g = \text{id}_x$ ”). The generalization we obtain is the fundamental notion of an “isomorphism”.

Definition 34.1 (Isomorphism)

Let \mathbf{C} be a category, let $X, Y \in \text{Ob}_{\mathbf{C}}$ be objects, and let $f : X \rightarrow Y$ be a morphism. We say that f is an *isomorphism* if there exists a morphism $g : Y \rightarrow X$ such that $f \circ g = \text{id}_X$ and $g \circ f = \text{id}_Y$.

Remark 34.2. The morphism g in the above definition is called the **inverse** of f . Because of the symmetry in how the definition is formulated, it is easy to see that g is necessarily also an isomorphism, and its inverse is f .

Exercise 54. In the previous remark we wrote *the* inverse. We do this because inverses are in fact unique. Can you prove this? That is, show that if $f : X \rightarrow Y$ is an isomorphism, and if $g_1 : Y \rightarrow X$ and $g_2 : Y \rightarrow X$ are morphisms such that $f \circ g_1 = \text{id}_X$ and $g_1 \circ f = \text{id}_Y$, and $f \circ g_2 = \text{id}_X$ and $g_2 \circ f = \text{id}_Y$, then necessarily $g_1 = g_2$.

See solution on page 475.

Definition 34.3 (Isomorphic objects)

Let X, Y be two objects in a category. We say that X and Y are *isomorphic* if there exists an isomorphism $X \rightarrow Y$ or $Y \rightarrow X$.

For the formulation of the definition of “isomorphic”, mathematicians might often only require the existence of an isomorphism $X \rightarrow Y$, say, since by remark above we know there is then necessarily also an isomorphism in the opposing direction, namely the inverse. We choose here the longer, perhaps more cumbersome formulation just to emphasize the symmetry of the term “isomorphic”. Also note that the definition leaves unspecified whether there might be just one or perhaps many isomorphisms $X \rightarrow Y$.

When two objects are isomorphic, in some contexts we will want to think of them as “the same”, and in some contexts we will want to keep track of more information. In fact, in category theory, it is typical to think in terms of different

kinds of “sameness”. To give a sense of this, we look at some examples using sets.

Example 34.4 (Semantic coherence). Suppose Francesca and Gabriel want to share a dish at a restaurant. Francesca only speaks Italian, and Gabriel only speaks German. Let M denote the set of dishes on the menu. For each dish, Francesca can say if she is willing to eat it, or not. This can be modeled by a function $f : M \rightarrow \{\text{Sì, No}\}$ which maps a given dish $m \in M$ to the statement “Sì” (yes, I’d eat it) or “No” (no, I wouldn’t eat it). Gabriel can do similarly, and this can be modeled as a function $g : M \rightarrow \{\text{Ja, Nein}\}$. Then, the subset of dishes of M that both Francesca and Gabriel are willing to eat (and thus able to share) is

$$\{m \in M \mid f(m) = \text{Sì} \quad \text{and} \quad g(m) = \text{Ja}\}. \quad (1)$$

Suppose the server at the restaurant knows no Italian and no German. To help with the situation, he introduces a new two-element set: $\{\heartsuit, \clubsuit\}$. Then Francesca and Gabriel can each map their respective positive answers (“Sì” and “Ja”) to “ \heartsuit ”, and their respective negative answers to “ \clubsuit ”. This defines isomorphisms

$$\{\text{Sì, No}\} \longleftrightarrow \{\heartsuit, \clubsuit\} \longleftrightarrow \{\text{Ja, Nein}\} \quad (2)$$

whose compositions provide a translation between the Italian and German two-element sets. Using these isomorphisms, we obtain, by composition, new functions

$$\tilde{f} : M \longrightarrow \{\heartsuit, \clubsuit\}, \quad \tilde{g} : M \longrightarrow \{\heartsuit, \clubsuit\}, \quad (3)$$

and the set of dishes that Francesca and Gabriel would be willing to share can be written as

$$\{m \in M \mid \tilde{f}(m) = \heartsuit \quad \text{and} \quad \tilde{g}(m) = \heartsuit\}. \quad (4)$$

This may all seem unnecessarily complicated. The main point of this example is the following. There are infinitely many two-element sets; commonly used ones might be, for example

$$\{0, 1\}, \{\text{true, false}\}, \{\perp, \top\}, \{\text{left, right}\}, \{-, +\}, \text{etc.} \quad (5)$$

They are all isomorphic (for any two such sets, there are precisely two possible isomorphisms between them) and we can in principle use any one in place of another. However, in most cases, we should keep precise track of the semantics of what each of the two elements mean in a given context, such as how they are being used in interaction with other mathematical constructs.

Example 34.5 (Relabelling). Suppose we want to buy an electric stepper motor for a robot that we are building, and for this we consult a catalogue of electric stepper motors*.

The catalogue might be organized as a large table, where on the left-hand side there is a column listing all available motors (identified with a model ID), and the remaining columns correspond to different attributes that each of the models of motor might have, such as the name of the company that manufactures the motor, the size dimensions, the weight, the maximum power, the price, *etc.* A simple illustration is provided in Table 34.1.

Suppose that your old way of listing models of motors has become outdated, and you need to change to a new system, where each model is identified, say, by a unique numerical 10-digit code. Relabelling each of the models with its numerical code corresponds to an isomorphism, say relabel, from the new set N of numerical codes to the old set M of model names. In contrast to the previous example,

* See pololu.com for a standard catalogue of electric stepper motors.

Table 34.1.: A simplified catalogue of motors.

Motor ID	Company	Size [mm ³]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	20 × 20 × 30	60.0	2.34	19.95
1206	SOYO	28 × 28 × 45	140.0	3.00	19.95
1207	SOYO	35 × 35 × 26	130.0	2.07	12.95
2267	SOYO	42 × 42 × 38	285.0	4.76	16.95
2279	Sanyo Denki	42 × 42 × 31.5	165.0	5.40	164.95
1478	SOYO	56.4 × 56.4 × 76	1,000	8.96	49.95
2299	Sanyo Denki	50 × 50 × 16	150.0	5.90	59.95

however, it is of course absolutely necessary to keep track of the isomorphism relabel that defines the relabelling. This is what holds the information of which code denotes which model.

Note also that all the other labelling functionalities in our example database may be updated by precomposing with relabel. For example, the old “Company” label was described by a function

$$\text{Company} : M \rightarrow C. \quad (6)$$

The updated version of the “Company” label, using the new set N of model IDs, is obtained by the composition

$$N \xrightarrow{\text{relabel}} M \xrightarrow{\text{Company}} C. \quad (7)$$

Example 34.6. Going back to currency exchangers, recall that any currency exchanger $\langle a, d \rangle$, given by

$$\begin{aligned} f_{\langle a, d \rangle} : \mathbb{R} &\rightarrow \mathbb{R}, \\ x &\mapsto ax - d, \end{aligned} \quad (8)$$

is an isomorphism, since we can define a currency exchanger $\langle a', b' \rangle$ such that

$$\langle a, b \rangle ; \langle a', b' \rangle = \langle a', b' \rangle ; \langle a, b \rangle = \langle 1, 0 \rangle. \quad (9)$$

Example 34.7. In **FinSet**, isomorphisms from a set to itself are automorphisms, and correspond to *permutations* of the set. Assuming a cardinality of n for the set (for instance, the set has n elements), the number of isomorphisms is given by the number of ways in which we can “rearrange” n elements of the set, which is $n!$.

Example 34.8. In **Set**, isomorphisms between $\mathbb{R} \rightarrow \mathbb{R}$ correspond to invertible functions.

34.2. Isomorphism is not identity

Example 34.9. Consider the two currencies ₩ and satoshi. These are both objects of the category **Curr** and are isomorphic. Being isomorphic does not mean to be strictly “the same”. Indeed, even if the amounts correspond, 1 ₩ and 1,000,000 satoshi are different elements of different sets, but there exists an isomorphism between the two. For one direction, the isomorphism transforms ₩ into satoshi (multiplying the real number by 1,000,000); the other direction transforms satoshi into ₩ (dividing the real number by 1,000,000).

Invertible functions are isomorphisms

Definition 34.10 (Strict monotone functions)

A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is *strictly monotone* if for all $a, b \in \mathbb{R}$:

$$\frac{a < b}{f(a) < f(b)}. \quad (10)$$

Exercise 55. Show that strictly monotone maps $f : \mathbb{R} \rightarrow \mathbb{R}$ are invertible.

See solution on page 475.

Solutions to selected exercises

Solution of Exercise 54.

Solution is missing.



Solution of Exercise 55. Because of Def. 34.10, given $a, b \in \mathbb{R}$, if $f(a) = f(b)$, then we must have $a = b$. Suppose that f is strictly monotone, but there exist elements $a, b \in \mathbb{R}$ such that $f(a) = f(b)$ but $a \neq b$. In other words, we have $f(a) - f(b) = 0$ but $a - b \neq 0$, which implies

$$(f(a) - f(b))(a - b) = 0. \quad (11)$$

However, from the definition of monotonicity, one should have

$$(f(a) - f(b))(a - b) > 0. \quad (12)$$

Therefore, we have a contradiction, implying that if f is strictly monotone, then $f(a) = f(b)$ implies $a = b$.

PART K. COMPOSITIONAL COMPUTATION



35. DP queries as functors	479
36. Solving finite co-design problems	499
37. Monads	525

The *St. Bernard* is a pretty large working dog from the Western Alps in Italy and Switzerland. Originally, such dogs were bred for rescue work, by the hospice of the Great St. Bernard Pass on Italian-Swiss border.



35. DP queries as functors

In this chapter we show how the two types of queries `FixFunMinRes` and `FixResMaxFun` can be seen as *functors* from the category `DP` to two new categories to be defined that represent the types of “solutions”. The specification of these functors represents a complete solution for `DP` optimization at the “mathematical level”, without taking into account questions of computability or resource consumption, which will be explored in the successive chapters.

35.1 Queries are functors from problem statements to solutions	480
35.2 The <code>Pos_U</code> and <code>Pos_L</code> categories	482
35.3 Queries as functors	494

35.1. Queries are functors from problem statements to solutions

In this and the following chapters we are going to build towards the solution of co-design problems. We will consider an arbitrary graph of design problems, in which nodes are design problems and edges are arbitrary interconnections between functionality and resources, obtained through the operations of a traced monoidal category (series, parallel, feedback) plus the lattice structure (and, or) of design problems. On this structure we want to solve the query **FixFunMinRes** (Section 29.3) or, symmetrically, **FixResMaxFun** (Section 29.3).

We look at this from a compositional point of view. We will assume that we know the solution to **FixFunMinRes** for each of the components. We think of the components as primitive blocks, because they are given in a catalogue format as a DPI, or they are special cases (+, \otimes , etc.) which we will solve as special cases. Given the solution for the primitive blocks, we want to know what is the solution for **FixFunMinRes** for the entire diagram.

What is the form of the solution that we expect? Given a DP $\mathbf{d} : \mathbf{F} \rightarrowtail \mathbf{R}$ we expect the solution to **FixFunMinRes** to be a function that, given a fixed functionality $f \in \mathbf{F}$, returns the minimal resources, which form an upper set. We call this function $H_{\mathbf{d}}$.

Definition 35.1

Given a DP $\mathbf{d} : \mathbf{F} \rightarrowtail \mathbf{R}$ we denote by $H_{\mathbf{d}} : \mathbf{F} \rightarrow_{\text{Pos}} \mathbf{UR}$ the map that associates to each functionality f the set of minimal resources sufficient to realize f :

$$\begin{aligned} H_{\mathbf{d}} : \mathbf{F} &\rightarrow_{\text{Pos}} \mathbf{UR}, \\ f &\mapsto \{r \in \mathbf{R} : \mathbf{d}(f^*, r)\}. \end{aligned} \tag{1}$$

If a certain functionality f is infeasible, then $H(f) = \emptyset$.

Remark 35.2 (Monotonicity). Consider a DP $\mathbf{d} : \mathbf{F} \rightarrowtail \mathbf{R}$ and $f \leq f'$. We know

$$\begin{aligned} H_{\mathbf{d}}(f) &= \{r \in \mathbf{R} : \mathbf{d}(f^*, r)\} \\ &\supseteq \{r \in \mathbf{R} : \mathbf{d}(f'^*, r)\} \\ &= H_{\mathbf{d}}(f'), \end{aligned} \tag{2}$$

showing monotonicity.

Symmetrically, the solution to **FixResMaxFun** is given by a function that we call $K_{\mathbf{d}}$.

Definition 35.3

Given a DP $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle$, define the map $K_{\mathbf{d}} : \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{LF}$ that associates to each resource r the set of functionalities which can be realized with r :

$$\begin{aligned} K_{\mathbf{d}} : \mathbf{R} &\rightarrow_{\text{Pos}} \mathbf{LF}, \\ r &\mapsto \{f \in \mathbf{F} : \mathbf{d}(f^*, r)\}. \end{aligned} \tag{3}$$

If a certain resource r only leads to infeasible functionalities, then $K(r) = \emptyset$.

Remark 35.4 (Monotonicity). Consider a DP $\mathbf{d} : \mathbf{F} \rightarrowtail \mathbf{R}$ and $r \leq r'$. We know

$$\begin{aligned} K_{\mathbf{d}}(r) &= \{f \in \mathbf{F} : \mathbf{d}(f^*, r)\} \\ &\subseteq \{f \in \mathbf{F} : \mathbf{d}(f^*, r')\} \\ &= K_{\mathbf{d}}(r'), \end{aligned} \tag{4}$$

showing monotonicity.

A question that arises naturally is whether the map H_d is sufficient to reconstruct the original DP. The answer is yes. We will prove that H_d defines a morphism in a category called \mathbf{Pos}_U , and that this category is equivalent (Def. 24.19) to \mathbf{DP} , therefore being traced monoidal, with a lattice structure. In fact, FixFunMinRes can be seen as a functor from \mathbf{DP} to \mathbf{Pos}_U . Symmetrically, K_d is a morphism in a category \mathbf{Pos}_L equivalent to \mathbf{DP} and FixResMaxFun can be seen as the functor from \mathbf{DP} to \mathbf{Pos}_L .

This situation is represented in Fig. 1.

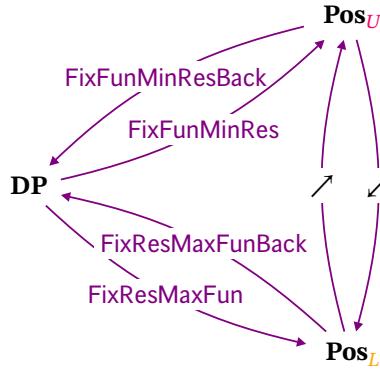


Figure 1.: In this chapter, we show that the queries FixResMaxFun and FixFunMinRes can be seen as functors from \mathbf{DP} to two new categories, \mathbf{Pos}_U and \mathbf{Pos}_L . We show that \mathbf{DP} is equivalent to these categories: a \mathbf{DP} is univocally defined by the answers to the two queries.

In the course of this chapter, by defining the two functors FixFunMinRes and FixResMaxFun , we effectively have solved the problem of optimization for DPs in the “mathematical” way. However, this is only the first step, because it does not say anything about whether the functor is actually computable. In the next chapter (Chapter 36) we will look at finite approximations of DPs and the computational complexity of the solution. Then, we will introduce the theory of monads, and based on that, we will be able to show how to construct bounded finite approximations of any DPs.

35.2. The \mathbf{Pos}_U and \mathbf{Pos}_L categories

Definition 35.5 (Category \mathbf{Pos}_U)

The category \mathbf{Pos}_U consists of:

1. *Objects*: objects are posets;
2. *Morphisms*: given objects $X, Y \in \text{Ob}_{\mathbf{Pos}_U}$, morphisms from $f : X \rightarrow Y$ are monotone maps of the form $f^* : X \rightarrow \text{Pos } UY$.
3. *Composition of morphisms*: Given morphisms $f : X \rightarrow Y, g : Y \rightarrow Z$, their composition $f ; g : X \rightarrow Z$ is given by

$$(f ; g)^* : X \rightarrow \text{Pos } UZ \\ x \mapsto \bigcup_{y \in f^*(x)} g^*(y); \quad (5)$$

4. *Identity morphism*: given an object $X \in \text{Ob}_{\mathbf{Pos}_U}$, the identity morphism $\text{id}_X : X \rightarrow X$ is given by the application of the upper closure operator:

$$\text{id}_X^*(x) := \uparrow\{x\}. \quad (6)$$

Remark 35.6. Note that the composition of morphisms in this category corresponds to the generalization of the series operator for boolean profunctors.

Analogously, we can define the \mathbf{Pos}_L category.

Definition 35.7 (Category \mathbf{Pos}_L)

The category \mathbf{Pos}_L consists of:

1. *Objects*: objects are posets;
2. *Morphisms*: given objects $X, Y \in \text{Ob}_{\mathbf{Pos}_L}$, morphisms $f : X \rightarrow Y$ are monotone maps of the form $f^* : X \rightarrow \text{Pos } LY$.
3. *Composition of morphisms*: Given morphisms $f : X \rightarrow Y, g : Y \rightarrow Z$, their composition $f ; g : X \rightarrow Z$ is given by

$$(f ; g)^* : X \rightarrow \text{Pos } LZ \\ x \mapsto \bigcup_{y \in f^*(x)} g^*(y); \quad (7)$$

4. *Identity morphism*: given an object $X \in \text{Ob}_{\mathbf{Pos}_L}$, the identity morphism $\text{id}_X : X \rightarrow X$ is given by the application of the lower closure operator:

$$\text{id}_X^*(x) := \downarrow\{x\}. \quad (8)$$

We now show that \mathbf{Pos}_U and \mathbf{Pos}_L are indeed categories.

Lemma 35.8. \mathbf{Pos}_U and \mathbf{Pos}_L are categories.

Proof. We prove that \mathbf{Pos}_U is a category. The proof for \mathbf{Pos}_L is analogous. In the following, we show unitality and associativity.

Unitality: Given $f : X \rightarrow Y$, we have:

$$(f ; \text{id}_Y)^*(x) = \bigcup_{y \in f^*(x)} \text{id}_Y^*(y) \\ = \bigcup_{y \in f^*(x)} \uparrow\{y\} \\ = \bigcup_{y \in f^*(x)} \{y' \in Y : y \leq_Y y'\}. \quad (9)$$

We know that $f^*(x)$ is an upper set:

$$\begin{aligned} f^*(x) &= \bigcup_{y \in f^*(x)} \{y\} \\ &= \bigcup_{y \in f^*(x)} \{y' \in Y : y \leq_Y y'\}. \end{aligned} \tag{10}$$

Therefore, $(f ; \text{id}_Y)^*(x) = f^*(x)$ for all $x \in X$. Similarly, we have:

$$\begin{aligned} (\text{id}_X ; f)^*(x) &= \bigcup_{x' \in \text{id}_X^*(x)} f^*(x') \\ &= \bigcup_{x' \in \uparrow\{x\}} f^*(x') \\ &= f^*(x), \end{aligned} \tag{11}$$

where the last equality holds since f^* is a monotone function and $f^*(x') \subseteq f^*(x)$ for all $x' \in \uparrow\{x\}$.

Associativity: Consider three morphisms $f : X \rightarrow Y$, $g : Y \rightarrow Z$, and $h : Z \rightarrow U$. We have:

$$\begin{aligned} ((f ; g) ; h)^*(x) &= \bigcup_{z \in \left(\bigcup_{y \in f^*(x)} g^*(y) \right)} h^*(z) \\ &= \bigcup_{y \in f^*(x)} \bigcup_{z \in g^*(y)} h^*(z) \\ &= (f ; (g ; h))^*(x). \end{aligned} \tag{12}$$

Therefore, \mathbf{Pos}_U is a category. \square

We can show that \mathbf{Pos}_U and \mathbf{Pos}_L are equivalent categories (Def. 24.19).

Lemma 35.9. \mathbf{Pos}_U and \mathbf{Pos}_L are isomorphic: there exists a pair of functors

$$\begin{aligned} \swarrow : \mathbf{Pos}_U &\rightarrow \mathbf{Pos}_L, \\ \nearrow : \mathbf{Pos}_L &\rightarrow \mathbf{Pos}_U, \end{aligned} \tag{13}$$

such that $\swarrow ; \nearrow = \text{id}_{\mathbf{Pos}_U}$ and $\nearrow ; \swarrow = \text{id}_{\mathbf{Pos}_L}$, where $\text{id}_{\mathbf{Pos}_U}$ and $\text{id}_{\mathbf{Pos}_L}$ are the identity functors on \mathbf{Pos}_U and \mathbf{Pos}_L , respectively.

Proof. To prove this, we need to define the needed functors and to show that they satisfy the listed properties. We choose the functors to be the ones that map a poset P in a category to its opposite version P^{op} in another category. Given a morphism $f : X \rightarrow Y$ in \mathbf{Pos}_U , we have:

$$\begin{aligned} (\swarrow(f))^* : X^{\text{op}} &\rightarrow \mathbf{Pos}_L Y^{\text{op}} \\ x &\mapsto f^*(x). \end{aligned} \tag{14}$$

Given a morphism $g : X \rightarrow Y$ in \mathbf{Pos}_L , we have:

$$\begin{aligned} (\nearrow(g))^* : X^{\text{op}} &\rightarrow \mathbf{Pos}_U Y^{\text{op}} \\ x &\mapsto g^*(x). \end{aligned} \tag{15}$$

\swarrow and \nearrow are functors:

▷ *Preservation of identities:* Given $X \in \text{Ob}_{\text{Pos}_U}$, we have:

$$\begin{aligned} (\swarrow(\text{id}_X))^\star &= \uparrow_X \{x\} \\ &= \downarrow_{X^{\text{op}}} \{x\} \\ &= \text{id}_{X^{\text{op}}}^\star, \end{aligned} \tag{16}$$

where id_X is an identity morphism in Pos_U , and $\text{id}_{X^{\text{op}}}$ is an identity morphism in Pos_L . Similarly, given $X \in \text{Ob}_{\text{Pos}_L}$ we have:

$$\begin{aligned} (\nearrow(\text{id}_X))^\star &= \downarrow_X \{x\} \\ &= \uparrow_{X^{\text{op}}} \{x\} \\ &= \text{id}_{X^{\text{op}}}^\star. \end{aligned} \tag{17}$$

▷ *Preservation of composition:* This can be easily seen as follows. Given any $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$, $g \in \text{Hom}_{\text{Pos}_U}(Y; Z)$:

$$\begin{aligned} (\swarrow(f \circ g))^\star &= (f \circ g)^\star \\ &= (\swarrow(f) \circ \swarrow(g))^\star. \end{aligned} \tag{18}$$

Similarly, given any $f \in \text{Hom}_{\text{Pos}_L}(X; Y)$, $g \in \text{Hom}_{\text{Pos}_L}(Y; Z)$:

$$\begin{aligned} (\nearrow(f \circ g))^\star &= (f \circ g)^\star \\ &= (\nearrow(f) \circ \nearrow(g))^\star. \end{aligned} \tag{19}$$

Compositions return identity functors: We want to show that by composing the two functors we obtain the identity functors in Pos_U and Pos_L , respectively. Clearly, composing the two functors returns the identity on the objects, since for any poset P , we have $(P^{\text{op}})^{\text{op}} = P$. The functors act on morphisms by “flipping the context”, and “flipping” twice is the “same” as not flipping. □

We can show that Pos_U and Pos_L are monoidal categories.

Lemma 35.10. Pos_U is a monoidal category with the following additional structure:

1. *Tensor product \otimes :* On objects, the tensor product corresponds to the product of posets. Given two morphisms $f : X \rightarrow Y$ and $g : Z \rightarrow U$, we have $f \otimes g : X \times Z \rightarrow Y \times U$, with

$$\begin{aligned} (f \otimes g)^\star : X \times Z &\rightarrow \text{Pos } U(Y \times U) \\ \langle x, z \rangle &\mapsto f^\star(x) \times g^\star(z). \end{aligned} \tag{20}$$

Note that the cartesian product of upper sets is an upper set.

2. *Unit:* The unit is the identity poset: the poset with a singleton carrier set and only the identity relation. We denote this by 1 .
3. *Left unit:* The left unit is given by the pair of morphisms $\text{lu}_X : \{*\} \times X \rightarrow X$ and $\text{lu}_X^{-1} : X \rightarrow \{*\} \times X$, with

$$\begin{aligned} \text{lu}_X^\star : \{*\} \times X &\rightarrow \text{Pos } UX \\ \langle *, x \rangle &\mapsto \uparrow \{x\}, \end{aligned} \tag{21}$$

and

$$\begin{aligned} \text{lu}_X^{-1}^{\star} : X &\rightarrow \mathbf{Pos} \quad U(\{\bullet\} \times X) \\ x &\mapsto \{\bullet\} \times \uparrow\{x\}, \end{aligned} \tag{22}$$

respectively.

4. *Right unit*: The right unit is given by the pair of morphisms $\text{ru}_X : X \times \{\bullet\} \rightarrow X$ and $\text{ru}_X^{-1} : X \rightarrow X \times \{\bullet\}$, with

$$\begin{aligned} \text{ru}_X^{\star} : X \times \{\bullet\} &\rightarrow \mathbf{Pos} \quad UX \\ \langle x, \bullet \rangle &\mapsto \uparrow\{x\}, \end{aligned} \tag{23}$$

and

$$\begin{aligned} \text{ru}_X^{-1}^{\star} : X &\rightarrow \mathbf{Pos} \quad U(X \times \{\bullet\}) \\ x &\mapsto \uparrow\{x\} \times \{\bullet\}, \end{aligned} \tag{24}$$

respectively.

5. *Associator*: The associator is given by the pair of morphisms $\text{as}_{XY,Z} : (X \times Y) \times Z \rightarrow X \times (Y \times Z)$ and $\text{as}_{X,YZ} : X \times (Y \times Z) \rightarrow (X \times Y) \times Z$, given by

$$\begin{aligned} \text{as}_{XY,Z}^{\star} : (X \times Y) \times Z &\rightarrow \mathbf{Pos} \quad UX \times (UY \times UZ) \\ \langle \langle x, y \rangle, z \rangle &\mapsto \uparrow\{x\} \times (\uparrow\{y\} \times \uparrow\{z\}), \end{aligned} \tag{25}$$

and

$$\begin{aligned} \text{as}_{X,YZ}^{\star} : X \times (Y \times Z) &\rightarrow \mathbf{Pos} \quad (UX \times UY) \times UZ \\ \langle x, \langle y, z \rangle \rangle &\mapsto (\uparrow\{x\} \times \uparrow\{y\}) \times \uparrow\{z\}. \end{aligned} \tag{26}$$

We now want to show that \mathbf{Pos}_U can be equipped to become a symmetric monoidal category. To do so, we first need the following two facts.

Lemma 35.11. Given posets P, Q , a monotone maps $f : P \rightarrow Q$, and a family of singleton sets $\{S_i\}_{i \in I}$, with $S_i = \{s_i\}$, $s_i \in P$, the following equality holds:

$$\uparrow\left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\}\right) = \uparrow\left(\bigcup_{i \in I} \{f(s_i)\}\right). \tag{27}$$

Proof. We first want to show that:

$$\underbrace{\uparrow\left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\}\right)}_{\star} \subseteq \underbrace{\uparrow\left(\bigcup_{i \in I} \{f(s_i)\}\right)}_{\diamond}. \tag{28}$$

Take a

$$q \in \uparrow\left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\}\right). \tag{29}$$

If we have such a q , it means that there exists a

$$q' \in \bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \tag{30}$$

such that $q' \leq_Q q$, and hence there is a $p' \in \uparrow \bigcup_{i \in I} S_i$ such that $q' = f(p')$. Consequently, there must exist an $i' \in I$ such that $s_{i'} \leq_P p'$. The monotonicity of f implies:

$$f(s_{i'}) \leq_P f(p') = q' \leq_Q q. \tag{31}$$

We know that $s_{i'} \in \diamond$ and any $q^* \in Q$ satisfying $f(s_{i'}) \leq_Q q^*$ belongs to $\uparrow \diamond$.

Therefore, $\star \subseteq \uparrow \diamond$, which proves the validity of (28).

We now want to show that:

$$\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right) \supseteq \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (32)$$

By now taking a

$$q \in \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right), \quad (33)$$

we know that there is an $i' \in I$ such that $f(s_{i'}) \leq_Q q$. Furthermore, we know that $f(s_{i'}) \in \diamond$. Therefore, any $q^* \leq_Q f(s_{i'})$ must be in $\uparrow \diamond$, meaning that $q \in \star$, and proving the validity of (32).

The validity of (28) and (32) implies (27). \square

Remark 35.12. Given posets P, Q and a monotone maps $f : P \rightarrow Q$, we have:

$$\uparrow \left(\bigcup_{p' \in \uparrow \{p\}} \{f(p')\} \right) = \uparrow \{f(p)\}. \quad (34)$$

This follows from Lemma 35.11, by considering a family of singleton sets consisting solely of the set $\{p\}$.

Lemma 35.13. The cartesian product of upper sets is an upper set. The cartesian product of lower sets is a lower set.

Proof. Consider two posets P, Q and two respective upper sets A, B . We have

$$\frac{a \in A \quad a \leq_P a'}{a' \in A}, \quad (35)$$

and

$$\frac{b \in B \quad b \leq_Q b'}{b' \in B}. \quad (36)$$

Therefore:

$$\frac{\langle a, b \rangle \in A \times B \quad \langle a, b \rangle \leq_{P \times Q} \langle a', b' \rangle}{\langle a', b' \rangle \in A \times B}, \quad (37)$$

which proves that $A \times B$ is an upper set. The proof for the product of lower sets is analogous. \square

Lemma 35.14. $\langle \text{Pos}_U, \otimes, \mathbf{1} \rangle$ from Lemma 35.10 equipped with the braiding isomorphism $\text{br}_{X,Y} : X \times Y \xrightarrow{\cong} Y \times X$, given by

$$\begin{aligned} \text{br}_{X,Y}^\star : X \times Y &\rightarrow \text{Pos } U(Y \times X) \\ \langle x, y \rangle &\mapsto \uparrow \{y\} \times \uparrow \{x\}, \end{aligned} \quad (38)$$

defined for all $X, Y \in \text{Ob } \text{Pos}_U$, forms a symmetric monoidal category.

Proof. We first show that the braiding defines an isomorphism. In other words, we want to show

$$(\text{br}_{X,Y} \circ \text{br}_{Y,X})^\star = \text{id}_{X \times Y}^\star. \quad (39)$$

We have

$$\begin{aligned}
 & (\mathbf{br}_{X,Y} \circ \mathbf{br}_{Y,X})^*(x, y) \\
 &= \bigcup_{\langle x', y' \rangle \in \mathbf{br}_{X,Y}^*(x, y)} \mathbf{br}_{Y,X}^*(x', y') \\
 &= \bigcup_{\langle y', x' \rangle \in \uparrow\{y\} \times \uparrow\{x\}} \uparrow\{x'\} \times \uparrow\{y'\} \\
 &= \uparrow\{x\} \times \uparrow\{y\} \\
 &= \mathbf{id}_{X \times Y}^*(x, y).
 \end{aligned} \tag{40}$$

Note that this comes from the fact that \mathbf{br} is an involution. We now show naturality. Consider $f : X \rightarrow Y, g : Z \rightarrow U$. We have

$$\begin{aligned}
 & ((f \otimes g) \circ \mathbf{br}_{Y,U})^*(x, z) \\
 &= \langle f^*(x), g^*(z) \rangle \circ \mathbf{br}_{Y,U} \\
 &= \left\langle \bigcup_{z' \in g^*(z)} \uparrow z', \bigcup_{x' \in f^*(x)} \uparrow x' \right\rangle.
 \end{aligned} \tag{41}$$

On the other hand:

$$\begin{aligned}
 & (\mathbf{br}_{U,Y} \circ (f \otimes g))^*(x, z) \\
 &= \langle \uparrow\{z\}, \uparrow\{x\} \rangle \circ (f \otimes g)^* \\
 &= \left\langle \bigcup_{z' \in \uparrow\{z\}} g^*(z'), \bigcup_{x' \in \uparrow\{x\}} f^*(x') \right\rangle.
 \end{aligned} \tag{42}$$

Clearly, from Lemma 35.11 and Remark 35.12 we know that (41) and (42) are equivalent, proving naturality. We now just need to show hexagon identities. First, we want to show that

$$(\mathbf{br}_{X,Y} \otimes \mathbf{id}_Z) \circ \mathbf{as}_{Y,X,Z} \circ (\mathbf{id}_Y \otimes \mathbf{br}_{X,Z}) = \mathbf{as}_{X,Y,Z} \circ \mathbf{br}_{X,Y \otimes Z} \circ \mathbf{as}_{Y,Z,X} \tag{43}$$

To do so, we first look at the left-hand side of (43). We have

$$\begin{aligned}
 & ((\mathbf{br}_{X,Y} \otimes \mathbf{id}_Z) \circ \mathbf{as}_{Y,X,Z})^*(\langle x, y \rangle, z) \\
 &= \bigcup_{\langle \langle y', x' \rangle, z' \rangle \in (\mathbf{br}_{X,Y} \otimes \mathbf{id}_Z)^*(\langle x, y \rangle, z)} \mathbf{as}_{Y,X,Z}^*(y', x', z') \\
 &= \bigcup_{\langle \langle y', x' \rangle, z' \rangle \in (\uparrow\{y\} \times \uparrow\{x\}) \times \uparrow\{z\}} \uparrow\{y'\} \times (\uparrow\{x'\} \times \uparrow\{z'\}) \\
 &= \uparrow\{y\} \times (\uparrow\{x\} \times \uparrow\{z\}).
 \end{aligned} \tag{44}$$

Furthermore, we have

$$\begin{aligned}
 & ((\mathbf{br}_{X,Y} \otimes \mathbf{id}_Z) \circ \mathbf{as}_{Y,X,Z} \circ (\mathbf{id}_Y \otimes \mathbf{br}_{X,Z}))^*(\langle x, y \rangle, z) \\
 &= \bigcup_{\langle y', \langle x', z' \rangle \rangle \in \uparrow\{y\} \times (\uparrow\{x\} \times \uparrow\{z\})} (\mathbf{id}_Y \otimes \mathbf{br}_{X,Z})^*(y', \langle x', z' \rangle) \\
 &= \bigcup_{\langle y', \langle x', z' \rangle \rangle \in \uparrow\{y\} \times (\uparrow\{x\} \times \uparrow\{z\})} \uparrow\{y'\} \times (\uparrow\{z'\} \times \uparrow\{x'\}) \\
 &= \uparrow\{y\} \times (\uparrow\{z\} \times \uparrow\{x\}).
 \end{aligned} \tag{45}$$

We now look at the right-hand side of (43). We have

$$\begin{aligned}
 & \text{as}_{X,Y,Z} ; \text{br}_{X,Y \otimes Z}^{\star}(\langle x, y \rangle, z) \\
 &= \bigcup_{\langle x', \langle y', z' \rangle \rangle \in \text{as}_{X,Y,Z}^{\star}(\langle x, y \rangle, z)} \text{br}_{X,Y \otimes Z}^{\star}(x', \langle y', z' \rangle) \\
 &= \bigcup_{\langle x', \langle y', z' \rangle \rangle \in \uparrow \{x\} \times \uparrow \{y\} \times \uparrow \{z\}} (\uparrow \{y'\} \times \uparrow \{z'\}) \times \uparrow \{x'\} \\
 &= (\uparrow \{y\} \times \uparrow \{z\}) \times \uparrow \{x\}.
 \end{aligned} \tag{46}$$

Furthermore, we have

$$\begin{aligned}
 & (\text{as}_{X,Y,Z} ; \text{br}_{X,Y \otimes Z} ; \text{as}_{Y,Z,X})^{\star}(\langle x, y \rangle, z) \\
 &= \bigcup_{\langle \langle y', z' \rangle, x' \rangle \in (\uparrow \{y\} \times \uparrow \{z\}) \times \uparrow \{x\}} \text{as}_{Y,Z,X}^{\star}(\langle y', z' \rangle, x') \\
 &= \bigcup_{\langle \langle y', z' \rangle, x' \rangle \in (\uparrow \{y\} \times \uparrow \{z\}) \times \uparrow \{x\}} \uparrow \{y'\} \times (\uparrow \{z'\} \times \uparrow \{x'\}) \\
 &= \uparrow \{y\} \times (\uparrow \{z\} \times \uparrow \{x\}).
 \end{aligned} \tag{47}$$

Clearly, since (45) and (47) are equal, the first hexagon identity is checked. The second hexagon identity can be checked analogously. \square

Definition 35.15 (Trace in \mathbf{Pos}_U)

Given a morphism $f : X \times Z \rightarrow Y \times Z$ in \mathbf{Pos}_U , its trace in is defined as a morphism $\text{Tr}_{X,Y}^Z(f) : X \rightarrow Y$, given by

$$\begin{aligned}
 \text{Tr}_{X,Y}^Z(f)^{\star} : X &\rightarrow UY \\
 x &\mapsto \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^{\star}(x, z) \right\}.
 \end{aligned} \tag{48}$$

Lemma 35.16. $\langle \mathbf{Pos}_U, \otimes, \mathbf{1}, \text{br} \rangle$ equipped with the trace operation defined in Def. 35.15 is a traced monoidal category.

Proof. We have already checked that $\langle \mathbf{Pos}_U, \otimes, \mathbf{1}, \text{br} \rangle$ forms a symmetric monoidal category. First, we check that the trace indeed returns a valid morphism in \mathbf{Pos}_U . Given any $X, Y, Z \in \text{Ob}_{\mathbf{Pos}_U}$ and $f : X \times Z \rightarrow Y \times Z$, and any $x \leq x' \in X$, we need to prove that

$$\begin{aligned}
 & \overline{\text{Tr}_{X,Y}^Z(f)(x) \leq_{\mathbf{Pos}_U} \text{Tr}_{X,Y}^Z(f)(x')} \\
 & \overline{\text{Tr}_{X,Y}^Z(f)^{\star}(x) \supseteq \text{Tr}_{X,Y}^Z(f)^{\star}(x')}
 \end{aligned} \tag{49}$$

We know that f^{\star} is a monotone map, meaning that

$$\frac{\langle y, z \rangle \in f^{\star}(x', z)}{\langle y, z \rangle \in f^{\star}(x, z)} \tag{50}$$

Therefore:

$$\begin{aligned}
 & \overline{y \in \text{Tr}_{X,Y}^Z(f)^{\star}(x')} \\
 & \overline{y \in \text{Tr}_{X,Y}^Z(f)(x)}
 \end{aligned} \tag{51}$$

proving that $\text{Tr}_{X,Y}^Z(f)^{\star}$ is a monotone function. Furthermore, due to the

monotonicity of f^* , for any $y \leq y' \in Y, x \in X, z \in Z$, we have:

$$\begin{array}{c} \langle y, z \rangle \in f^*(x, z) \\ \hline \langle y', z \rangle \in f^*(x, z) \end{array} \quad (52)$$

proving that $\text{Tr}_{X,Y}^Z(f)^*(x)$ is an upper set for all $x \in X$. We now check the trace axioms one by one.

Naturality I: Given any object $X, X', Y, Z \in \text{Ob}_{\mathbf{Pos}_U}$, a morphism $f : X \times Z \rightarrow Y \times Z$, and a morphism $g : X' \rightarrow X$, we have:

$$\begin{aligned} & \text{Tr}_{X',Y}^Z((g \otimes \text{id}_Z) \circ f)^*(x') \\ &= \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in ((g \otimes \text{id}_Z) \circ f)^*(x', z) \right\} \\ &= \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \bigcup_{(x, z') \in g^*(x') \times \uparrow \{z\}} f^*(x, z') \right\} \\ &= \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \bigcup_{x \in g^*(x')} f^*(x, z) \right\}. \end{aligned} \quad (53)$$

On the other hand, we have

$$\begin{aligned} (g \circ \text{Tr}_{X,Z}^Z(f))^*(x') &= \bigcup_{x \in g^*(x')} \text{Tr}_{X,Z}^Z(f)^*(x) \\ &= \bigcup_{x \in g^*(x')} \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^*(x, z) \right\} \\ &= \left\{ y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \bigcup_{x \in g^*(x')} f^*(x, z) \right\}. \end{aligned} \quad (54)$$

Clearly (53) and (54) are equivalent, proving the first naturality condition.

Naturality II: Given any $X, Y, Y', Z \in \text{Ob}_{\mathbf{Pos}_U}$, $f : X \times Z \rightarrow Y \times Z$, and $g : Y \rightarrow Y'$, we have:

$$\begin{aligned} & \text{Tr}_{X,Y'}^Z(f \circ (g \otimes \text{id}_Z))^*(x) \\ &= \left\{ y' \in Y' \mid \bigvee_{z \in Z} \langle y', z \rangle \in (f \circ (g \otimes \text{id}_Z))^*(x, z) \right\} \\ &= \left\{ y' \in Y' \mid \bigvee_{z \in Z} \langle y', z \rangle \in \bigcup_{(y, z) \in f^*(x, z)} g^*(y) \times \uparrow \{z\} \right\} \end{aligned} \quad (55)$$

On the other hand

$$(\text{Tr}_{X,Y}^Z(f) \circ g)^*(x) = \bigcup_{y \in \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^*(x, z)\}} g^*(y) \quad (56)$$

Vanishing: Given any $X, Y \in \text{Ob}_{\mathbf{Pos}_U}$ and $f : X \rightarrow Y$ in \mathbf{Pos}_U , we have

$$\begin{aligned} & \text{Tr}_{X,Y}^{\mathbf{1}}(f)^*(x) \\ &= \{y \in Y \mid \langle y, \bullet \rangle \in (f \otimes \text{id}_{\mathbf{1}})^*(x, \bullet)\} \\ &= f^*(x). \end{aligned} \quad (57)$$

Furthermore, given any $X, Y, Z, U \in \text{Ob}_{\mathbf{Pos}_U}$ and $f : X \times Z \times U \rightarrow Y \times Z \times U$, we have

$$\begin{aligned} & \text{Tr}_{X,Y}^{Z \times U}(f)^*(x) \\ &= \left\{ y \in Y \mid \bigvee_{(z, u) \in Z \times U} \langle y, z, u \rangle \in f^*(x, z, u) \right\} \end{aligned} \quad (58)$$

To check the second vanishing axiom, we also write:

$$\begin{aligned} & \text{Tr}_{X \times Z, Y \times Z}^U(f)^\star(x, z) \\ &= \{(y, z) \in Y \times Z \mid \bigvee_{u \in U} \langle y, z, u \rangle \in f^\star(x, z, u)\}. \end{aligned} \quad (59)$$

Therefore, we can write:

$$\begin{aligned} & \left(\text{Tr}_{X, Y}^Z \left(\text{Tr}_{X \times Z, Y \times Z}^U(f) \right) \right)^\star(x) \\ &= \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \text{Tr}_{X \times Z, Y \times Z}^U(f)^\star(x, z)\} \\ &= \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \{\langle y', z' \rangle \in Y \times Z \mid \bigvee_{u \in U} \langle y', z', u \rangle \in f^\star(x, z', u)\}\} \\ &= \{y \in Y \mid \bigvee_z \in Z \left(\bigvee_{u \in U} \langle y, z, u \rangle \in f^\star(x, z, u) \right)\} \\ &= \{y \in Y \mid \bigvee_{\langle z, u \rangle \in Z \times U} \langle y, z, u \rangle \in f^\star(x, z, u)\}. \end{aligned} \quad (60)$$

Clearly, (58) and (60) are equivalent, proving the second vanishing axiom.

Superposing: Given any $X, Y, Z \in \text{Ob}_{\text{Pos}_U}$ and $f : X \times Z \rightarrow Y \times Z$, we have:

$$\begin{aligned} & \text{Tr}_{U \times X, U \times Y}^Z(\text{id}_U \otimes f)^\star(u, x) \\ &= \{\langle u, y \rangle \in U \times Y \mid \bigvee_{z \in Z} \langle u, y, z \rangle \in (\text{id}_U \otimes f)^\star(u, x, z)\} \\ &= \{\langle u, y \rangle \in U \times Y \mid \bigvee_{z \in Z} (u \in \text{id}_U^\star(u)) \wedge (\langle y, z \rangle \in f^\star(x, z))\} \\ &= \{\langle u, y \rangle \in U \times Y \mid \bigvee_{z \in Z} (u \in \uparrow\{u\}) \wedge (\langle y, z \rangle \in f^\star(x, z))\} \\ &= \{\langle u, y \rangle \in \uparrow\{u\} \times Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^\star(x, z)\} \\ &= \uparrow\{u\} \times \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^\star(x, z)\} \end{aligned} \quad (61)$$

On the other hand, we have:

$$(\text{id}_U \otimes \text{Tr}_{X, Y}^Z(f))^\star(u, x) = \uparrow\{u\} \times \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in f^\star(x, z)\}. \quad (62)$$

Clearly, (61) and (62) are equivalent, proving the superposing axiom.

Yanking: Consider $X \in \text{Ob}_{\text{Pos}_U}$. We have

$$\begin{aligned} & \text{Tr}_{X, X}^X(\text{br}_{X, X})^\star(x) \\ &= \{x' \in X \mid \bigvee_{x'' \in X} \langle x', x'' \rangle \in \text{br}_{X, X}^\star(x, x'')\} \\ &= \{x' \in X \mid \bigvee_{x'' \in X} \langle x', x'' \rangle \in \uparrow\{x''\} \times \uparrow\{x\}\} \\ &= \{x' \in X \mid \bigvee_{x'' \in X} (x' \in \uparrow\{x''\}) \wedge (x'' \in \uparrow\{x\})\} \\ &= \{x' \in X \mid x' \in \uparrow\{x\}\} \\ &= \uparrow\{x\} \\ &= \text{id}_X^\star(x), \end{aligned} \quad (63)$$

proving the yanking axiom. \square

Definition 35.17 (Order on morphisms in \mathbf{Pos}_U)

Given any two morphisms $f, g : X \rightarrow Y$ in \mathbf{Pos}_U , we define an order between them as

$$\frac{f \leq_{\mathbf{Pos}_U} g}{f^*(x) \leq_{UY} g^*(x), \quad \forall x \in X}. \quad (64)$$

Definition 35.18 (Order on morphisms in \mathbf{Pos}_L)

Given any two morphisms $f, g : X \rightarrow Y$ in \mathbf{Pos}_L , we define an order between them as

$$\frac{f \leq_{\mathbf{Pos}_L} g}{f^*(x) \leq_{LY} g^*(x), \quad \forall x \in X}. \quad (65)$$

Definition 35.19 (Intersection of morphisms in \mathbf{Pos}_L)

Given two morphisms $f, g : X \rightarrow Y$ in \mathbf{Pos}_L , their *intersection* (meet) is a morphism $f \wedge g : X \rightarrow Y$, given by

$$\begin{aligned} (f \wedge g)^* &: X \rightarrow LY \\ x &\mapsto f^*(x) \cap g^*(x). \end{aligned} \quad (66)$$

Definition 35.20 (Union of morphisms in \mathbf{Pos}_U)

Given two morphisms $f, g : X \rightarrow Y$ in \mathbf{Pos}_U , their *union* (join) is a morphism $f \vee g : X \rightarrow Y$, given by

$$\begin{aligned} (f \vee g)^* &: X \rightarrow UY \\ x &\mapsto f^*(x) \cup g^*(x). \end{aligned} \quad (67)$$

Definition 35.21 (Union of morphisms in \mathbf{Pos}_L)

Given two morphisms $f, g : X \rightarrow Y$ in \mathbf{Pos}_L , their *union* (join) is a morphism $f \vee g : X \rightarrow Y$, given by

$$\begin{aligned} (f \vee g)^* &: X \rightarrow LY \\ x &\mapsto f^*(x) \cup g^*(x). \end{aligned} \quad (68)$$

Lemma 35.22. Given any $X, Y \in \text{Ob}_{\mathbf{Pos}_U}$, $\text{Hom}_{\mathbf{Pos}_U}(X; Y)$ is a bounded lattice with union \vee of morphisms in \mathbf{Pos}_U as join, intersection \wedge of morphisms in \mathbf{Pos}_U as meet, least upper bound $\top_{\text{Hom}_{\mathbf{Pos}_U}(X; Y)} : X \rightarrow Y$ given by

$$\begin{aligned} (\top_{\text{Hom}_{\mathbf{Pos}_U}(X; Y)})^* &: X \rightarrow UY \\ x &\mapsto \emptyset, \end{aligned} \quad (69)$$

and greatest lower bound $\perp_{\text{Hom}_{\mathbf{Pos}_U}(X; Y)} : X \rightarrow Y$ given by

$$\begin{aligned} (\perp_{\text{Hom}_{\mathbf{Pos}_U}(X; Y)})^* &: X \rightarrow UY \\ x &\mapsto Y. \end{aligned} \quad (70)$$

Proof. First, we need to prove that $\text{Hom}_{\text{Pos}_U}(X; Y)$ forms a poset. To prove this, we check the following, using the order defined in Def. 35.17

▷ *Reflexivity:* Given $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$, we can write

$$f^*(x) \sqsupseteq f^*(x), \quad \forall x \in X, \quad (71)$$

which implies $f \leq_{\text{Pos}_U} f$.

▷ *Antisymmetry:* Consider

$$f, g \in \text{Hom}_{\text{Pos}_U}(X; Y) \quad (72)$$

with $f \leq_{\text{Pos}_U} g$ and $g \leq_{\text{Pos}_U} f$. We know

$$(f \leq_{\text{Pos}_U} g) \Rightarrow f^*(x) \sqsupseteq g^*(x), \quad \forall x \in X, \quad (73)$$

but also

$$(g \leq_{\text{Pos}_U} f) \Rightarrow g^*(x) \sqsupseteq f^*(x), \quad \forall x \in X, \quad (74)$$

implying $f = g$.

▷ *Transitivity:* Consider

$$f, g, h \in \text{Hom}_{\text{Pos}_U}(X; Y) \quad (75)$$

with $f \leq_{\text{Pos}_U} g$ and $g \leq_{\text{Pos}_U} h$. We have, for all $x \in X$,

$$\begin{aligned} (f^*(x) \sqsupseteq g^*(x)) \wedge (g^*(x) \sqsupseteq h^*(x)) &\Rightarrow f^*(x) \sqsupseteq h^*(x) \\ &\Rightarrow f \leq_{\text{Pos}_U} h. \end{aligned} \quad (76)$$

Consider now $f, g \in \text{Hom}_{\text{Pos}_U}(X; Y)$. Their least upper bound (join) is $f \wedge g$, since it is the least morphism such that $f \leq_{\text{Pos}_U} (f \wedge g)$ and $g \leq_{\text{Pos}_U} (f \wedge g)$. Their greatest lower bound (meet) is $f \vee g$, since it is the greatest morphism such that $(f \vee g) \leq_{\text{Pos}_U} f$ and $(f \vee g) \leq_{\text{Pos}_U} g$. Furthermore, for any $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$, one will have, for all $x \in X$

$$f^*(x) \sqsupseteq \emptyset = \top_{\text{Hom}_{\text{Pos}_U}(X; Y)}(x), \quad (77)$$

implying that for all $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$ we have $f \leq_{\text{Pos}_U} \top_{\text{Hom}_{\text{Pos}_U}(X; Y)}$. Finally, for any $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$, one will have, for all $x \in X$

$$\perp_{\text{Hom}_{\text{Pos}_U}(X; Y)}(x) = Y \sqsupseteq f^*(x) \quad (78)$$

implying that for all $f \in \text{Hom}_{\text{Pos}_U}(X; Y)$ we have $\perp_{\text{Hom}_{\text{Pos}_U}(X; Y)} \leq_{\text{Pos}_U} f$. \square

Lemma 35.23. Given any $X, Y \in \text{Ob}_{\text{Pos}_U}$, $\text{Hom}_{\text{Pos}_L}(X; Y)$ is a bounded lattice with intersection \wedge of morphisms in Pos_L as meet, union \vee of morphisms in Pos_L as join, least upper bound $\top_{\text{Hom}_{\text{Pos}_L}(X; Y)}: X \rightarrow Y$ given by

$$\begin{aligned} \top_{\text{Hom}_{\text{Pos}_L}(X; Y)}(x) &= X \rightarrow LY \\ x &\mapsto Y, \end{aligned} \quad (79)$$

and greatest lower bound $\perp_{\mathbf{Hom} \mathbf{Pos}_U(X;Y)} : X \rightarrow Y$ given by

$$\begin{aligned} \perp_{\mathbf{Hom} \mathbf{Pos}_U(X;Y)}^* &: X \rightarrow LY \\ x &\mapsto \emptyset. \end{aligned} \tag{80}$$

Proof. The proof is analogous to the one of Lemma 35.22. Note that meets/joins and top/bottom are switched in meaning, because of the difference in order between UX and LX . \square

35.3. DP queries are functors from problem statements to solutions

Lemma 35.24. There is a functor

$$\text{FixFunMinRes} : \mathbf{DP} \rightarrow \mathbf{Pos}_{\mathbf{U}} \quad (81)$$

that maps:

1. An object (poset) in \mathbf{DP} to the same object (poset) in $\mathbf{Pos}_{\mathbf{U}}$.
2. A morphism $e \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$ to the morphism $H_e \in \text{Hom}_{\mathbf{Pos}_{\mathbf{U}}}(\mathbf{F}; \mathbf{R})$, where:

$$\begin{aligned} H_e^* : \mathbf{F} &\rightarrow \mathbf{Pos}_{\mathbf{U}} \mathbf{R} \\ f &\mapsto \{r \in \mathbf{R} \mid e(f^*, r)\}. \end{aligned} \quad (82)$$

Proof. We prove the two conditions.

Preservation of identities: We have

$$\begin{aligned} \text{FixFunMinRes}(\text{id}_X^{\mathbf{DP}})^*(x) &= \{y \in Y \mid \text{id}_X^{\mathbf{DP}}(x^*, y)\} \\ &= \{y \in Y \mid x \leq y\} \\ &= \uparrow\{x\} \\ &= \text{id}_X^{\mathbf{Pos}_{\mathbf{U}}}{}^*(x). \end{aligned} \quad (83)$$

Preservation of composition: On one hand, we have

$$\begin{aligned} \text{FixFunMinRes}(\mathbf{d} \circ_{\mathbf{DP}} e)^*(x) &= \{z \in Z \mid (\mathbf{d} \circ e)(x^*, z)\} \\ &= \{z \in Z \mid \bigvee_{y \in Y} \mathbf{d}(x^*, y) \wedge e(y^*, z)\}. \end{aligned} \quad (84)$$

On the other hand:

$$\begin{aligned} (\text{FixFunMinRes}(\mathbf{d}) \circ_{\mathbf{Pos}_{\mathbf{U}}} \text{FixFunMinRes}(e))^*(x) &= \bigcup_{y \in \text{FixFunMinRes}(\mathbf{d})^*(x)} \text{FixFunMinRes}(e)^*(y) \\ &= \bigcup_{y \in \{y \in Y \mid \mathbf{d}(x^*, y)\}} \{z \in Z \mid e(y^*, z)\} \\ &= \{z \in Z \mid (y \in Y) \wedge \mathbf{d}(x^*, y) \wedge e(y^*, z)\} \\ &= \{z \in Z \mid \bigvee_{y \in Y} \mathbf{d}(x^*, y) \wedge e(y^*, z)\}. \end{aligned} \quad (85)$$

Clearly, (84) and (85) coincide. \square

Lemma 35.25. There is a functor

$$\text{FixResMaxFun} : \mathbf{DP} \rightarrow \mathbf{Pos}_{\mathbf{L}} \quad (86)$$

which maps:

1. An object (poset) of \mathbf{DP} to the same object (poset) in $\mathbf{Pos}_{\mathbf{L}}$.
2. A morphism $e \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$ to the morphism $K_e \in \text{Hom}_{\mathbf{Pos}_{\mathbf{L}}}(\mathbf{R}; \mathbf{F})$, where:

$$\begin{aligned} K_e^* : \mathbf{R} &\rightarrow \mathbf{Pos}_{\mathbf{L}} \mathbf{F} \\ r &\mapsto \{f \in \mathbf{F} \mid e(f^*, r)\}. \end{aligned} \quad (87)$$

Proof. The proof is analogous to the one of Lemma 35.24. \square

Lemma 35.26. There is a functor $\text{FixFunMinResBack} : \mathbf{Pos}_{\textcolor{red}{U}} \rightarrow \mathbf{DP}$ which maps:

1. An object (poset) in $\mathbf{Pos}_{\textcolor{red}{U}}$ to the same object (poset) in \mathbf{DP} .
2. A morphism $g \in \text{Hom}_{\mathbf{Pos}_{\textcolor{red}{U}}}(\mathbf{F}; \mathbf{R})$ to the morphism $\mathbf{d}_g \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$, where:

$$\begin{aligned} \mathbf{d}_g : \mathbf{F}^{\text{op}} \times \mathbf{R} &\rightarrow \mathbf{Pos} \text{ Bool} \\ \langle f^*, r \rangle &\mapsto r \in g^*(f). \end{aligned} \tag{88}$$

Proof. We prove the two conditions.

Preservation of identities: We have

$$\begin{aligned} \text{FixFunMinResBack}(\text{id}_{\textcolor{red}{X}}^{\mathbf{Pos}_{\textcolor{red}{U}}})(x^*, y) &= y \in \text{id}_{\textcolor{brown}{Y}}^{\mathbf{Pos}_{\textcolor{red}{U}}}{}^* \\ &= y \in \uparrow\{x\} \\ &= \text{id}_{\textcolor{red}{X}}^{\mathbf{DP}}(x^*, y). \end{aligned} \tag{89}$$

Preservation of composition: On one hand, we have

$$\begin{aligned} \text{FixFunMinResBack}(f \circ_{\mathbf{Pos}_{\textcolor{red}{U}}} g)(x^*, z) &= z \in (f \circ_{\mathbf{Pos}_{\textcolor{red}{U}}} g)^*(x) \\ &= z \in \bigcup_{y \in f^*(x)} g^*(y). \end{aligned} \tag{90}$$

On the other hand:

$$\begin{aligned} &(\text{FixFunMinResBack}(f) \circ_{\mathbf{DP}} \text{FixFunMinResBack}(g))(x^*, z) \\ &= \bigvee_{y \in Y} (y \in f^*(x)) \wedge (z \in g^*(y)) \\ &= z \in \bigcup_{y \in f^*(x)} g^*(y). \end{aligned} \tag{91}$$

Clearly, (90) and (91) coincide. \square

Lemma 35.27. There is a functor $\text{FixResMaxFunBack} : \mathbf{Pos}_{\textcolor{orange}{L}} \rightarrow \mathbf{DP}$ which maps:

1. An object (poset) in $\mathbf{Pos}_{\textcolor{red}{U}}$ to the same object (poset) in \mathbf{DP} .
2. A morphism $g \in \text{Hom}_{\mathbf{Pos}_{\textcolor{orange}{L}}}(\mathbf{F}; \mathbf{R})$ to the morphism $\mathbf{d}_g \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$, where:

$$\begin{aligned} \mathbf{d}_g : \mathbf{F}^{\text{op}} \times \mathbf{R} &\rightarrow \mathbf{Pos} \text{ Bool} \\ \langle f^*, r \rangle &\mapsto f \in g^*(r). \end{aligned} \tag{92}$$

Proof. The proof is analogous to the one of Lemma 35.26. \square

Lemma 35.28. The pair of functors FixFunMinRes and FixFunMinResBack together with the natural isomorphisms

$$\text{FixFunMinRes} ; \text{FixFunMinResBack} \cong \text{id}_{\mathbf{DP}}, \tag{93}$$

and

$$\text{FixFunMinResBack} ; \text{FixFunMinRes} \cong \text{id}_{\mathbf{Pos}_{\textcolor{red}{U}}}, \tag{94}$$

form an equivalence for \mathbf{DP} and $\mathbf{Pos}_{\textcolor{red}{U}}$.

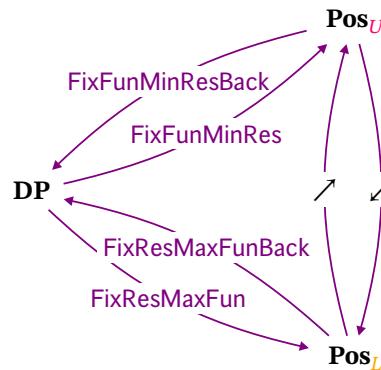


Figure 2.: From \mathbf{DP} to \mathbf{Pos}_U and \mathbf{Pos}_L , and back.

Proof. First, consider any morphism in $\text{Hom}_{\mathbf{DP}}(X; Y)$. We have

$$\begin{aligned}
 & (\text{FixFunMinRes} ; \text{FixFunMinResBack})(\mathbf{d})(x^*, y) \\
 &= y \in \text{FixFunMinRes}(\mathbf{d})^*(x) \\
 &= y \in \{y' \in Y \mid \mathbf{d}(x^*, y)\} \\
 &= \mathbf{d}(x^*, y) \\
 &= \text{id}_{\mathbf{DP}}(\mathbf{d})(x^*, y).
 \end{aligned} \tag{95}$$

Now consider any morphism $H_d \in \text{Hom}_{\mathbf{Pos}_U}(X; Y)$. We have

$$\begin{aligned}
 & (\text{FixFunMinResBack} ; \text{FixFunMinRes})(H_d)(x) \\
 &= \{y \in Y \mid y \in H_d^*(x)\} \\
 &= \{y \in Y \mid \mathbf{d}(x^*, y)\} \\
 &= \text{id}_{\mathbf{Pos}_U}(H_d)(x).
 \end{aligned} \tag{96}$$

□

Lemma 35.29. FixFunMinRes preserves the bounded lattice structure.

Proof. Given $X, Y \in \text{Ob}_{\mathbf{DP}}$ and $\mathbf{d}, \mathbf{e} \in \text{Hom}_{\mathbf{DP}}(X; Y)$, we want to check the following properties.

Order reversing: We want to check

$$\frac{\mathbf{d} \leq_{\mathbf{DP}} \mathbf{e}}{\text{FixFunMinRes}(\mathbf{d}) \geq_{\mathbf{Pos}_U} \text{FixFunMinRes}(\mathbf{e})}. \tag{97}$$

We have:

$$\begin{aligned}
 \text{FixFunMinRes}(\mathbf{d})^*(x) &= \{y \in Y \mid \mathbf{d}(x^*, y)\} \\
 &\subseteq \{y \in Y \mid \mathbf{e}(x^*, y)\} \\
 &= \text{FixFunMinRes}(\mathbf{e})^*(x),
 \end{aligned} \tag{98}$$

implying $\text{FixFunMinRes}(\mathbf{d}) \geq_{\mathbf{Pos}_U} \text{FixFunMinRes}(\mathbf{e})$.

Meet and join preservation: We want to check

$$\text{FixFunMinRes}(\mathbf{d} \wedge \mathbf{e}) = \text{FixFunMinRes}(\mathbf{d}) \wedge_{\mathbf{Pos}_U} \text{FixFunMinRes}(\mathbf{e}), \tag{99}$$

and

$$\text{FixFunMinRes}(\mathbf{d} \vee \mathbf{e}) = \text{FixFunMinRes}(\mathbf{d}) \vee_{\text{Pos}_U} \text{FixFunMinRes}(\mathbf{e}). \quad (100)$$

We have

$$\begin{aligned} & \text{FixFunMinRes}(\mathbf{d} \wedge \mathbf{e})^*(x) \\ &= \{y \in Y \mid (\mathbf{d} \wedge \mathbf{e})(x^*, y)\} \\ &= \{y \in Y \mid (\mathbf{d}(x^*, y) \wedge_{\mathbf{DP}} \mathbf{e}(x^*, y))\} \\ &= \{y \in Y \mid \mathbf{d}(x^*, y)\} \cap \{y \in Y \mid \mathbf{e}(x^*, y)\} \\ &= \text{FixFunMinRes}(\mathbf{d})^*(x) \wedge_{\text{Pos}_U} \text{FixFunMinRes}(\mathbf{e})^*(x). \end{aligned} \quad (101)$$

Similarly:

$$\begin{aligned} & \text{FixFunMinRes}(\mathbf{d} \vee \mathbf{DPe})^*(x) \\ &= \{y \in Y \mid (\mathbf{d} \vee \mathbf{e})(x^*, y)\} \\ &= \{y \in Y \mid (\mathbf{d}(x^*, y) \vee_{\mathbf{DP}} \mathbf{e}(x^*, y))\} \\ &= \{y \in Y \mid \mathbf{d}(x^*, y)\} \cup \{y \in Y \mid \mathbf{e}(x^*, y)\} \\ &= \text{FixFunMinRes}(\mathbf{d})^*(x) \vee_{\text{Pos}_U} \text{FixFunMinRes}(\mathbf{e})^*(x). \end{aligned} \quad (102)$$

Top and bottom preservation: We want to check

$$\text{FixFunMinRes}(\perp_{\text{Hom}_{\mathbf{DP}}(X; Y)}) = \perp_{\text{Hom}_{\text{Pos}_U}(X; Y)}, \quad (103)$$

and

$$\text{FixFunMinRes}(\top_{\text{Hom}_{\mathbf{DP}}(X; Y)}) = \top_{\text{Hom}_{\text{Pos}_U}(X; Y)}. \quad (104)$$

We have

$$\begin{aligned} & \text{FixFunMinRes}(\perp_{\text{Hom}_{\mathbf{DP}}(X; Y)})^*(x) = \emptyset \\ &= \perp_{\text{Hom}_{\text{Pos}_U}(X; Y)}^*(x) \end{aligned} \quad (105)$$

Similarly

$$\begin{aligned} & \text{FixFunMinRes}(\top_{\text{Hom}_{\mathbf{DP}}(X; Y)})^*(x) = Y \\ &= \top_{\text{Hom}_{\text{Pos}_U}(X; Y)}^*(x). \end{aligned} \quad (106)$$

□

Lemma 35.30. FixFunMinRes preserves traces. In other words:

Proof. We want to show that

$$\text{FixFunMinRes}(\text{Tr}_{X, Y}^Z(\mathbf{d})) = \text{Tr}_{X, Y}^Z(\text{FixFunMinRes}(\mathbf{d})), \quad (107)$$

for all $\mathbf{d} \in \text{Hom}_{\mathbf{DP}}(X \times Z; Y \times Z)$, and $X, Y, Z \in \text{Ob}_{\mathbf{DP}}$. On one hand, we have

$$\begin{aligned} & \text{FixFunMinRes}(\text{Tr}_{X, Y}^Z(\mathbf{d}))^*(x) = \{y \in Y \mid \text{Tr}_{X, Y}^Z(\mathbf{d})(x^*, y)\} \\ &= \{y \in Y \mid \bigvee_{z \in Z} \mathbf{d}(\langle x, z \rangle^*, \langle y, z \rangle)\} \end{aligned} \quad (108)$$

On the other hand, we have

$$\begin{aligned}
 & \text{Tr}_{X,Y}^Z(\text{FixFunMinRes}(\mathbf{d}))^*(x) \\
 &= \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \text{FixFunMinRes}(\mathbf{d})^*(x, z)\} \\
 &= \{y \in Y \mid \bigvee_{z \in Z} \langle y, z \rangle \in \{\langle y, z \rangle \in Y \times Z \mid \mathbf{d}(\langle x, z \rangle^*, \langle y, z \rangle)\}\} \\
 &= \{y \in Y \mid \bigvee_{z \in Z} \mathbf{d}(\langle x, z \rangle^*, \langle y, z \rangle)\}.
 \end{aligned} \tag{109}$$

□



36. Solving finite co-design problems

In this chapter we discuss the solution of *finite* design problem. A finite co-design problem is one in which the upper set of the solutions can be described as the upper closure of a finite antichain. Therefore, the solution can be represented with finite memory.

36.1 Domain theory and fixed points	500
36.2 Finite co-design problems	504
36.3 Handling loops	505
36.4 Example: Optimizing over the natural numbers	508
36.5 Extended Numerical Examples	511
36.6 Complexity of the solution	520
36.7 Decomposition of CDPs	522

36.1. Domain theory and fixed points

In this section we recall some fundamentals of domain theory. It is used in computer science for defining denotational semantics (see e.g., [19]). It is used in embedded systems for defining the semantics of models of computation (see, e.g., [14]). What we need from domain theory is the least necessary to define *least fixed points* and to use Kleene's theorem.

Domain theory builds on order theory by defining “directed” and “complete” partial orders. These attributes play the same role as compactness in analysis: they will be used to make sure that certain sequences can converge to a fixed point.

Directed and complete partial orders

Definition 36.1 (Directed set)

In a poset $P = \langle P, \leq_P \rangle$, we say that a set $S \subseteq P$ is *directed* if each pair of elements in S has an upper bound: for all $x, y \in S$, there exists $z \in S$ such that $x \leq z$ and $y \leq z$.

Definition 36.2 (Completeness)

A poset is a *directed complete partial order* (DCPO) if each of its directed subsets has a supremum (least of upper bounds). It is a *complete partial order* (CPO) if it also has a bottom.

Example 36.3 (Completion of $\mathbb{R}_{\geq 0}$ to $\overline{\mathbb{R}}_{\geq 0}$). The poset $\langle \mathbb{R}, \leq \rangle$ is not a CPO, because it lacks a bottom.

The non-negative reals $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$ have a bottom $\perp = 0$, however, they are not a DCPO because some of their directed subsets do not have an upper bound. For example, take $\mathbb{R}_{\geq 0}$, which is a subset of $\mathbb{R}_{\geq 0}$. Then $\mathbb{R}_{\geq 0}$ is directed, because for each $a, b \in \mathbb{R}_{\geq 0}$, there exists $c = \max\{a, b\} \in \mathbb{R}_{\geq 0}$ for which $a \leq c$ and $b \leq c$.

One way to make $\langle \mathbb{R}_{\geq 0}, \leq \rangle$ a CPO is by adding an artificial top element \top that we think as “a point at infinitely”. We can define then the completion

$$\overline{\mathbb{R}}_{\geq 0} := \mathbb{R}_{\geq 0} \cup \{\top\}, \quad (1)$$

and extending the partial order \leq so that $a \leq \top$ for all $a \in \mathbb{R}_{\geq 0}$.

Example 36.4. Any lattice is a DCPO.

Example 36.5. For any poset P , UP is a CPO, because it is a bounded lattice.

Scott continuity

Scott continuity is a property of maps on DCPOs that is slightly stronger than monotonicity.

Definition 36.6 (Scott continuity)

A map $f : P \rightarrow \text{Pos}$ between DCPOs is *Scott continuous* iff for each directed subset $S \subseteq P$, the image $f(S)$ is directed, and

$$f(\text{Sup } S) = \text{Sup } f(S). \quad (2)$$

Lemma 36.7. Scott continuity implies monotonicity.

Proof. Consider a map $f : P \rightarrow \text{Pos}$ that is Scott continuous. Take two elements $x, y \in P$ such that $x \leq y$. The set $S = \{x, y\}$ is directed. From (2), we know that

$$f(\text{Sup } S) = f(y) = \text{Sup } \{f(x), f(y)\}, \quad (3)$$

which implies that $f(x) \leq f(y)$. Therefore, f is monotone. \square

Remark 36.8. Scott continuity is not the same as the notion of continuity as used in analysis you might be familiar with. A map from the CPO $\langle \overline{\mathbb{R}}_{\geq 0}, \leq \rangle$ to itself is Scott continuous iff it is nondecreasing and left-continuous. For example, the ceiling function $x \mapsto \lceil x \rceil$ is not continuous in the usual sense, but it is Scott continuous (Fig. 1).

However, the name “continuity” for this property is aptly chosen. In analysis, a function is continuous if it preserves limits, in the sense that

$$\lim_{n \rightarrow \infty} f(a_n) = f(\lim_{n \rightarrow \infty} a_n), \quad (4)$$

which is, in spirit, the same as (2).

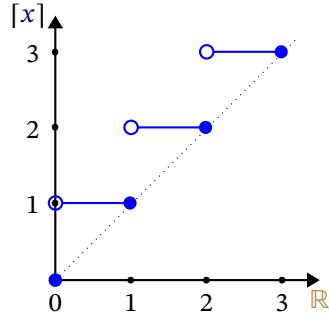


Figure 1.: The ceiling function is Scott continuous.

Least fixed points

Definition 36.9 (Fixed points)

A *fixed point* of $f : P \rightarrow \text{Pos}$ is a point x such that $f(x) = x$.

Definition 36.10 (Least fixed points)

A *least fixed point* of $f : P \rightarrow \text{Pos}$ is the minimum (if it exists) of the set of fixed points of f :

$$\text{lfp}(f) := \min_{\leq} \{x \in P : f(x) = x\}. \quad (5)$$

In general, a function need not have a fixed point. It also might have multiple fixed points; and also in that case there are might not be a *least* fixed point.

However, the conditions for a least fixed point to exist are quite weak. Monotonicity of the map f plus completeness is sufficient to ensure existence.

Lemma 36.11. If P is a CPO and $f : P \rightarrow \text{Pos}$ is monotone, then $\text{lfp}(f)$ exists and is unique.

This is given as CPO Fixpoint Theorem II, 8.22 in [3].

With the additional assumption of Scott continuity, Kleene’s algorithm is a systematic procedure to find the least fixed point.

Lemma 36.12 (Kleene’s fixed-point theorem). Assume P is a CPO, and $f : P \rightarrow \text{Pos}$ is Scott continuous. Then the least fixed point of f is the supremum of the Kleene ascent chain

$$\perp \leq f(\perp) \leq f(f(\perp)) \leq \dots \leq f^{(n)}(\perp) \leq \dots. \quad (6)$$

This is given as CPO fixpoint theorem I, 8.15 in [3].

Example: party invite

Consider again the party scenario of Example 4.29. Consider the case where a subset $S \subseteq A$ of people decide to throw a party. They then proceed to call all their

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

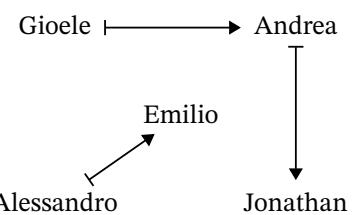


Figure 2.: Party invite relation.

friends, who accept, and, if they were not invited already, enthusiastically call *their* friends to extend the invite. We want to find out what is the final group of people that will show up at the party. We call this map $\phi : \text{Pow } \mathbf{A} \rightarrow \text{Pow } \mathbf{A}$, so that if \mathbf{S} is the initial group, $\phi(\mathbf{S})$ is the complete set of invites.

Note that this is related to the transitive closure operation, but we are only interested in the transitive closure from a certain initial set \mathbf{S} .

For example, consider the case in which the relation is as in Fig. 2. In this case, we would have

$$\phi(\emptyset) = \emptyset, \quad (7)$$

which means that, if nobody starts a party, no party takes place. Jonathan does not invite anybody, so we would have

$$\phi(\{\text{Jonathan}\}) = \{\text{Jonathan}\} \quad (8)$$

If Gioele and Alessandro start the party, everybody will get invited:

$$\phi(\{\text{Alessandro, Gioele}\}) = \text{everybody}. \quad (9)$$

We can show that

1. The function ϕ can be computed as a fixed point.
2. The recursive invite strategy corresponds to Kleene's iteration.

We summarize the properties that we want the function ϕ to have. Given an initial subset \mathbf{S} , we would like to find the set of people $\mathbf{T} = \phi(\mathbf{S})$ such that:

1. \mathbf{T} contains the initial set \mathbf{S} :

$$\mathbf{S} \subseteq \mathbf{T} \quad (10)$$

2. \mathbf{T} is closed with respect to a certain invite relation $R : \mathbf{A} \rightarrow \mathbf{A}$. If $x R y$, then x invites y to the party. Define the function

$$\begin{aligned} m : \text{Pow } \mathbf{A} &\rightarrow \text{Pow } \mathbf{A}, \\ \mathbf{T} &\mapsto \mathbf{T} \cup \bigcup_{x \in \mathbf{T}} \{y \in \mathbf{A} : x R y\}. \end{aligned} \quad (11)$$

This represents one iteration of the invite process: given a set \mathbf{T} , we add to \mathbf{T} all invitees of each of the elements of \mathbf{T} .

We are looking for a set \mathbf{T} such that it is a fixed point of the invite function:

$$\mathbf{T} = m(\mathbf{T}). \quad (12)$$

3. $\phi(\mathbf{S})$ is the smallest among all such sets that satisfy the two conditions above.

Let \mathbf{P} be the upper principal set of \mathbf{S} : given (10), we know that we want sets that contain at least \mathbf{S} :

$$\mathbf{P} = \uparrow \mathbf{S} = \{\mathbf{T} \in \text{Pow } \mathbf{A} : \mathbf{S} \subseteq \mathbf{T}\}. \quad (13)$$

The poset \mathbf{P} is a sublattice of $\text{Pow } \mathbf{A}$. Note also that the bottom of \mathbf{P} is \emptyset .

In summary, we are looking for the smallest point of \mathbf{P} that is closed to m :

$$\phi(\mathbf{S}) = \min_{\subseteq} \{\mathbf{T} \in \mathbf{P} : \mathbf{T} = m(\mathbf{T})\} \quad (14)$$

Comparing this with (5), we see that $\phi(\mathbf{S})$ is the least fixed point of m :

$$\phi(\mathbf{S}) = \text{lfp}(m). \quad (15)$$

Take Kleene's iteration in (6):

$$\perp \leq f(\perp) \leq f(f(\perp)) \leq \dots \leq f^{(n)}(\perp) \leq \dots . \quad (16)$$

Because the bottom of $P = \uparrow S$ is S , we can rewrite it as:

$$S \subseteq m(S) \subseteq m(m(S)) \subseteq m(m(m(S))) \dots . \quad (17)$$

Each element of the sequence corresponds to one iteration of the invite algorithm.

36.2. Finite co-design problems

If we want a computable algorithm for solving co-design queries, it is necessary that the solution can be finitely representable. One way to do this is to zero-in on those design problems that are guaranteed, by construction, to have a finite solution. This is what we do in this section. In the next chapters, we will see how we can construct bounded finite solutions to non-finitely-representable DPs.

In the `FixFunMinRes` queries, the solution lives in an upper set of resources. We now look at upper sets that can be represented as the upper closure of a finite antichains.

Definition 36.13 (Finitely-supported upper sets)

Given a poset P , we call an upper set $S \in UP$ *finitely supported* if it can be written as the upper closure of a finite antichain:

$$S = (\uparrow \alpha), \text{ for } \alpha \in \text{Anti } P, \text{card}(\alpha) < \infty. \quad (18)$$

We call $\overline{U}_f P$ the set of finitely-supported upper sets of a poset P .

We call $\text{Anti}_f P$ the set of finite antichains.

Definition 36.14 (Finite design problems)

We call a design problem finite if, in its representation $H : F \rightarrow UR$, $h(f) \in \overline{U}_f R$ for all $f \in F$.

We show that finite co-design problems form a subcategory of DP that is also monoidal and locally posetal. (Note that we are leaving out “traced” for now.) To show this, we just need to check that all the ways to compose finite DPs result in finite DPs. The formulas that we derive work also describe an algorithm to compute the solution to the queries.

Definition 36.15 (Category of finite design problems $\text{Pos}_{\overline{U}_f}$)

The category of *finite design problems* $\text{Pos}_{\overline{U}_f}$ consists of the following constituents:

1. *Objects*: The objects are posets.
2. *Morphisms*: The morphisms are *finite* design problems (Def. 36.14).
3. *Identity morphism*: The identity morphism $\text{id}_P : P \rightarrow P$ is as in DP .
4. *Composition operation*: Given morphisms $d : P \rightarrow Q$ and $e : Q \rightarrow R$, their composition $d ; e : P \rightarrow R$ is as in DP .

36.3. Handling loops

We are close to having a complete solution. The only part that is missing is dealing with loops (trace).

First, we will work with a particular form of loops, called “the Conway form”, shown in Fig. 3. This corresponds to working with design problems of the type

$$\mathbf{d} : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{B}, \quad (19)$$

The new feedback operator has signature

$$\text{loop} : (\mathbf{A} \times \mathbf{B} \rightarrow \mathbf{B}) \rightarrow (\mathbf{A} \rightarrow \mathbf{B}) \quad (20)$$

We can do this without loss of generality, because we can re-write the trace with this new operator. We leave Fig. 4 as a graphical proof that this is possible. Hagesawa [10] discusses the equivalence in detail.

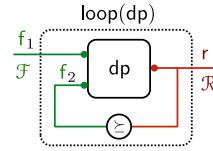


Figure 3.

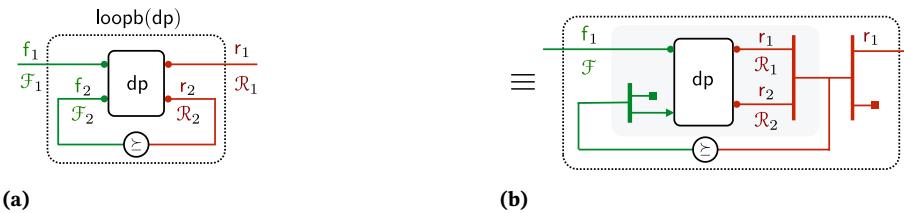


Figure 4.: We can rewrite the trace in Conway's form.

The following theorem establishes a closed form for $h_{\text{loop}(\mathbf{d})}$ as a least fixed point. Here on we consider $\text{Anti}_f \mathbf{R}$ as a poset with the order given by

$$\frac{\alpha_1 \leq_{\text{Anti}_f \mathbf{R}} \alpha_2}{\uparrow \alpha_1 \leq_{U_f \mathbf{R}} \uparrow \alpha_2}. \quad (21)$$

Theorem 36.16. For any DP \mathbf{d} of the right shape, we can compute $h_{\text{loop}(\mathbf{d})}$ as follows:

$$h_{\text{loop}(\mathbf{d})} : f_1 \mapsto \uparrow \text{lfp}(\Phi_{f_1}), \quad (22)$$

that is, as the *least fixed point* of a map Φ_{f_1} defined as

$$\begin{aligned} \Phi_{f_1} : \text{Anti}_f \mathbf{R} &\rightarrow \text{Anti}_f \mathbf{R}, \\ \alpha &\mapsto \text{Min}_{\leq_R} \bigcup_{r \in \alpha} h_{\mathbf{d}}(f_1, r) \cap \uparrow r. \end{aligned} \quad (23)$$

Proof. The diagram in Fig. 3 implies that the map $h_{\text{loop}(\mathbf{d})}$ can be described as:

$$h_{\text{loop}(\mathbf{d})} : \mathbf{F}_1 \rightarrow \text{Anti}_f \mathbf{R}, \quad (24)$$

$$f_1 \mapsto \begin{cases} \text{using } r, f_2 \in \mathbf{R}, \\ \text{Min}_{\leq_R} r, \\ \text{s.t. } r \in h_{\mathbf{d}}(f_1, f_2), \\ r \leq_R f_2. \end{cases} \quad (25)$$

Denote by h_{f_1} the map $h_{\mathbf{d}}$ with the first element fixed:

$$h_{f_1} : f_2 \mapsto h_{\mathbf{d}}(f_1, f_2). \quad (26)$$

Rewrite $r \in h_d(f_1, f_2)$ in (24) as

$$r \in h_{f_1}(f_2). \quad (27)$$

Let r be a feasible solution, but not necessarily minimal. Lemma 36.17 implies that the constraint (27) can be rewritten as

$$\{r\} = h_{f_1}(f_2) \cap \uparrow r. \quad (28)$$

Because $f_2 \geq r$, and h_{f_1} is Scott continuous, it follows that $h_{f_1}(f_2) \geq_{\text{Anti}_f \mathbf{R}} h_{f_1}(r)$. Therefore, by Lemma 36.18, we have

$$\{r\} \geq_{\text{Anti}_f \mathbf{R}} h_{f_1}(r) \cap \uparrow r. \quad (29)$$

This is a recursive condition that all feasible r must satisfy.

Let $\alpha \in \text{Anti}_f \mathbf{R}$ be an antichain of feasible resources, and let r be a generic element of \mathbf{R} . Tautologically, rewrite α as the minimal elements of the union of the singletons containing its elements:

$$\alpha = \text{Min}_{\leq_R} \bigcup_{r \in \alpha} \{r\}. \quad (30)$$

Substituting (29) in (30) we obtain (cf Lemma 36.19)

$$\alpha \geq_{\text{Anti}_f \mathbf{R}} \text{Min}_{\leq_R} \bigcup_{r \in \alpha} h_{f_1}(r) \cap \uparrow r. \quad (31)$$

Converse: It is also true that if an antichain α satisfies (31) then all $r \in \alpha$ are feasible. The constraint (31) means that for any $r_0 \in \alpha$ on the left side, we can find a r_1 on the right side so that $r_0 \geq_R r_1$. The point r_1 needs to belong to one of the sets of which we take the union; say that it comes from $r_2 \in \alpha$, so that $r_1 \in h_{f_1}(r_2) \cap \uparrow r_2$. Summarizing:

$$\forall r_0 \in \alpha : \exists r_1 : (r_0 \geq_R r_1) \wedge (\exists r_2 \in \alpha : r_1 \in h_{f_1}(r_2) \cap \uparrow r_2). \quad (32)$$

Because $r_1 \in h_{f_1}(r_2) \cap \uparrow r_2$, we can conclude that $r_1 \in \uparrow r_2$, and therefore $r_1 \geq_R r_2$, which together with $r_0 \geq_R r_1$, implies $r_0 \geq_R r_2$. We have concluded that there exist two points r_0, r_2 in the antichain α such that $r_0 \geq_R r_2$; therefore, they are the same point: $r_0 = r_2$. Because $r_0 \geq_R r_1 \geq_R r_2$, we also conclude that r_1 is the same point as well. We can rewrite (32) by using r_0 in place of r_1 and r_2 to obtain $\forall r_0 \in \alpha : r_0 \in h_{f_1}(r_0)$, which means that r_0 is a feasible resource.

We have concluded that all antichains of feasible resources α satisfy (31), and conversely, if an antichain α satisfies (31), then it is an antichain of feasible resources.

Equation (31) is a recursive constraint for α , of the kind

$$\Phi_{f_1}(\alpha) \leq_{\text{Anti}_f \mathbf{R}} \alpha, \quad (33)$$

with the map Φ_{f_1} defined by

$$\begin{aligned} \Phi_{f_1} : \text{Anti}_f \mathbf{R} &\rightarrow \text{Anti}_f \mathbf{R}, \\ \alpha &\mapsto \text{Min}_{\leq_R} \bigcup_{r \in \alpha} h_{f_1}(r) \cap \uparrow r. \end{aligned} \quad (34)$$

If we want the *minimal* resources, we are looking for the *least* antichain:

$$\min_{\leq_{\text{Anti}_f \mathbf{R}}} \{ \alpha \in \text{Anti}_f \mathbf{R} : \Phi_{f_1}(\alpha) \leq_{\text{Anti}_f \mathbf{R}} \alpha \}, \quad (35)$$

which is equal to the *least fixed point* of Φ_{f_1} . Therefore, the map $h_{\text{loop}(d)}$ can be written as

$$h_{\text{loop}(d)} : f_1 \mapsto \text{lfp}(\Phi_{f_1}). \quad (36)$$

Lemma 36.20 shows that $\text{lfp}(\Phi_{f_1})$ is Scott continuous in f_1 . \square

Lemma 36.17. Let S be an antichain in P . Then

$$\overline{\overline{x \in S}} = \{x\} = S \cap \uparrow x. \quad (37)$$

Lemma 36.18. For $S, T \in \text{Anti}_f P$, and $A \subseteq B, S \leq_{\text{Anti}_f \mathbf{R}} T$ implies $S \cap A \leq_{\text{Anti}_f \mathbf{R}} T \cap A$.

Lemma 36.19. For $S, T, U, V \in \text{Anti}_f P, S \leq_{\text{Anti}_f \mathbf{R}} U$ and $T \leq_{\text{Anti}_f \mathbf{R}} V$ implies $S \cup T \leq_{\text{Anti}_f \mathbf{R}} U \cup V$.

Lemma 36.20. Let $f : P \times Q \rightarrow \text{Pos}$ be Scott continuous. For each $x \in P$, define the map

$$f_x : y \mapsto f(x, y) \quad (38)$$

Then the map

$$f^\dagger : x \mapsto \text{lfp}(f_x) \quad (39)$$

is Scott continuous.

Proof. Davey and Priestly [3] leave this as Exercise 8.26. A proof is found in Gierz *et al.* [6, Exercise II-2.29]. \square

36.4. Example: Optimizing over the natural numbers

This is a simple example that can show two interesting properties of CDPIs:

1. the ability to work with discrete posets; and
2. the ability to treat multi-objective optimization problems.

Consider the family of optimization problems indexed by $c \in \mathbb{N}$:

$$\begin{cases} \text{Min}_{\leq_{\mathbb{N} \times \mathbb{N}}} & \langle x, y \rangle, \\ \text{s.t.} & x + y \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c. \end{cases} \quad (40)$$

One can show that this optimization problem is a CDP by producing a co-design diagram with an equivalent semantics, such as the one in Fig. 5.

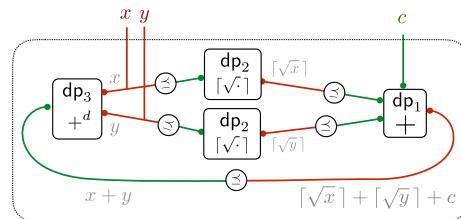


Figure 5.: Co-design diagram equivalent to (40)

The diagram contains three primitive DPs: \mathbf{d}_1 , \mathbf{d}_2 (used twice), and \mathbf{d}_3 . Their h maps are:

$$\begin{aligned} h_1 : \overline{\mathbb{N}} \times \overline{\mathbb{N}} \times \overline{\mathbb{N}} &\rightarrow \text{Anti}_{\mathbb{f}} \overline{\mathbb{N}}, \\ \langle f_1, f_2, f_3 \rangle &\mapsto \{f_1 + f_2 + f_3\}, \\ h_2 : \overline{\mathbb{N}} &\rightarrow \text{Anti}_{\mathbb{f}} \overline{\mathbb{N}}, \\ f &\mapsto \{\lceil \sqrt{f} \rceil\}, \\ h_3 : \overline{\mathbb{N}} &\rightarrow \text{Anti}_{\mathbb{f}} (\overline{\mathbb{N}} \times \overline{\mathbb{N}}), \\ f &\mapsto \{(a, b) \in \overline{\mathbb{N}} \times \overline{\mathbb{N}} : a + b = f\}. \end{aligned}$$

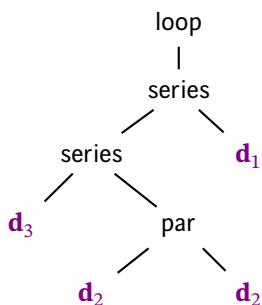


Figure 6.

The tree decomposition (Fig. 6) corresponds to the expression

$$\mathbf{d} = \text{loop}(\text{series}(\text{par}(\mathbf{d}_2, \mathbf{d}_2), \text{series}(\mathbf{d}_1, \mathbf{d}_3))). \quad (41)$$

From (41) we obtain an expression for h :

$$h = ((h_2 \otimes h_2) \circledcirc h_1 \circledcirc h_3)^\dagger. \quad (42)$$

This problem is small enough that we can write down an explicit expression for h . By substituting in (42) the definitions for \otimes , \dagger , \circledcirc , we obtain that evaluating $h(c)$ means finding the least fixed point of a map Ψ_c :

$$h : c \mapsto \text{lfp}(\Psi_c). \quad (43)$$

The map $\Psi_c : \text{Anti}_{\mathbb{f}} (\overline{\mathbb{N}} \times \overline{\mathbb{N}}) \rightarrow \text{Anti}_{\mathbb{f}} (\overline{\mathbb{N}} \times \overline{\mathbb{N}})$ can be obtained from Theorem 36.16 as follows:

$$\Psi_c : \alpha \mapsto \text{Min} \bigcup_{(x, y) \in \alpha} \uparrow(x, y) \cap \quad (44)$$

$$\cap \{(a, b) \in \mathbb{N}^2 : (a + b \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c)\}. \quad (45)$$

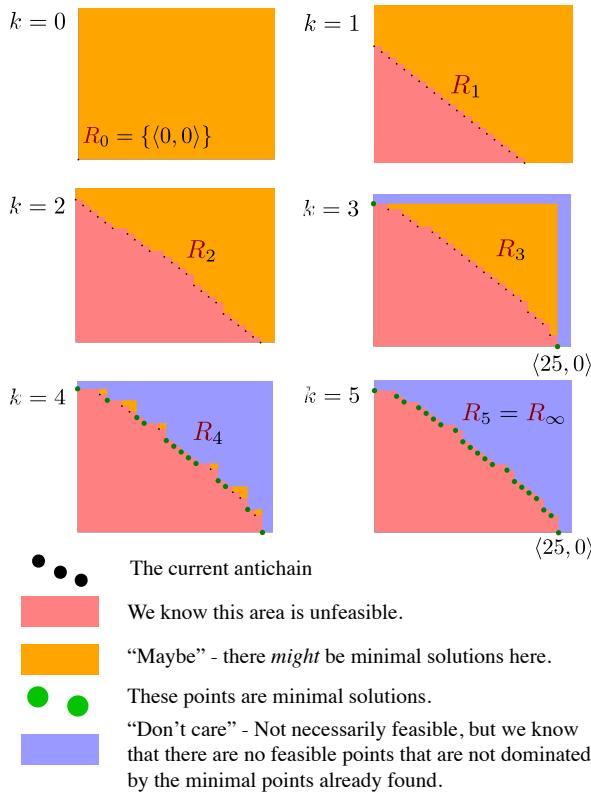


Figure 7.1: Kleene ascent to solve the problem (40) for $c = 20$. The sequence converges in five steps to $\alpha_5 = \alpha_\infty$.

Kleene's algorithm is the iteration $\alpha_{k+1} = \Psi_c(\alpha_k)$ starting from

$$\alpha_0 = \perp_{\text{Antif } (\overline{\mathbb{N}} \times \overline{\mathbb{N}})} = \{(0,0)\}. \quad (46)$$

For $c = 0$, the sequence converges immediately:

$$\alpha_0 = \{(0,0)\} = \textcolor{blue}{h}(0). \quad (47)$$

For $c = 1$, the sequence converges at the sixth step; however, some solutions (in bold) converge sooner:

$$\alpha_0 = \{(0,0)\}, \quad (48)$$

$$\alpha_1 = \{(0,1), \langle 1,0 \rangle\}, \quad (49)$$

$$\alpha_2 = \{(0,2), \langle 1,1 \rangle, \langle 2,0 \rangle\}, \quad (50)$$

$$\alpha_3 = \{\langle 0,3 \rangle, \langle 1,2 \rangle, \langle 2,1 \rangle, \langle 3,0 \rangle\}, \quad (51)$$

$$\alpha_4 = \{\langle 0,3 \rangle, \langle 2,2 \rangle, \langle 3,0 \rangle\}, \quad (52)$$

$$\alpha_5 = \{\langle 0,3 \rangle, \langle 3,0 \rangle\} = \textcolor{blue}{h}(1). \quad (53)$$

For $c = 2$, the sequence converges at the fifth step; however, some solutions (in bold) converge sooner:

$$\alpha_0 = \{(0,0)\}, \quad (54)$$

$$\alpha_1 = \{(0,2), \langle 1,1 \rangle, \langle 2,0 \rangle\}, \quad (55)$$

$$\alpha_2 = \{\langle 0,4 \rangle, \langle 1,3 \rangle, \langle 2,2 \rangle, \langle 3,1 \rangle, \langle 4,0 \rangle\}, \quad (56)$$

$$\alpha_3 = \{\langle 0,4 \rangle, \langle 3,2 \rangle, \langle 2,3 \rangle, \langle 4,0 \rangle\} \quad (57)$$

$$\alpha_4 = \{\langle 0,4 \rangle, \langle 3,3 \rangle, \langle 4,0 \rangle\} = \textcolor{blue}{h}(2). \quad (58)$$

The next values in the sequence are:

$$h(3) = \{\langle 0, 6 \rangle, \langle 3, 4 \rangle, \langle 4, 3 \rangle, \langle 6, 0 \rangle\}, \quad (59)$$

$$h(4) = \{\langle 0, 7 \rangle, \langle 3, 6 \rangle, \langle 4, 4 \rangle, \langle 6, 3 \rangle, \langle 7, 0 \rangle\}. \quad (60)$$

Figure 7 shows the sequence for $c = 20$.

Guarantees of Kleene ascent

Solving an CDP with cycles reduces to computing a Kleene ascent sequence α_k . At each instant k we have some additional guarantees.

For any finite k , the resources “below” α_k (the set $\mathbf{R} \setminus \uparrow \alpha_k$) are infeasible. (In Fig. 7, those are colored in red.)

If the iteration converges to a non-empty antichain α_∞ , the antichain α_∞ divides \mathbf{R} in two. Below the antichain, all resources are infeasible. However, above the antichain (purple area), it is not necessarily true that all points are feasible, because there might be holes in the feasible set. Note that this method does not compute the entire feasible set, but rather only the *minimal elements* of the feasible set, which might be much easier to compute.

Finally, if the sequence converges to the empty set, it means that there are no solutions. The sequence α_k can be considered a certificate of infeasibility.

36.5. Extended Numerical Examples

This example considers the choice of different battery technologies for a robot. The goals of this example are: 1) to show how design problems can be composed; 2) to show how to define hard constraints and precedence between resources to be minimized; 3) to show how even relatively simple models can give very complex trade-offs surfaces; and 4) to introduce MCDPL, a formal language for the description of co-design problems.

Language and interpreter/solver

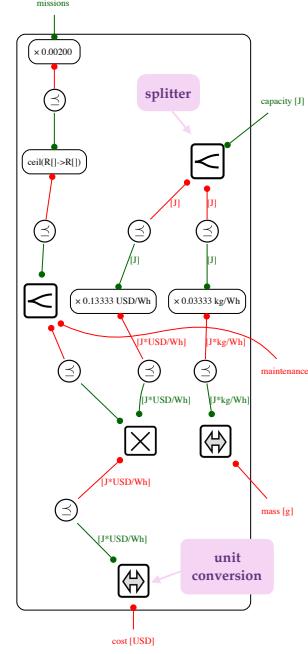
MCDPL is a modeling language to describe CDPs and their compositions. It is inspired by CVX and “disciplined convex programming” [8]. MCDPL is even more disciplined than CVX; for example, multiplying by a negative number is a *syntax* error. The figures are generated by PyMCDP, an interpreter and solver for CDPs, which implements the techniques described in these sections. An in-depth description of MCDPL is available in the next volume of this series.

Model of a battery

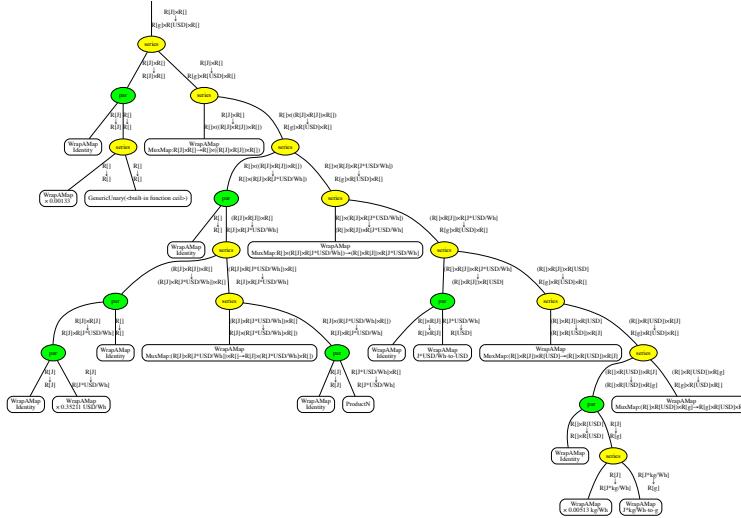
```

1 mcdp {
2   provides capacity [J]
3   provides missions [R]
4
5   requires mass [g]
6   requires cost [$]
7
8   # Number of replacements
9   requires maintenance [R]
10
11  # Battery properties
12  specific_energy = 30 Wh/kg
13  specific_cost = 7.50 Wh/$
14  cycles = 500 []
15
16  # Constraint between mass and capacity
17  required mass >= provided capacity / specific_energy
18
19  # How many times should it be replaced?
20  num_replacements = ceil(provided missions / cycles)
21  required maintenance >= num_replacements
22
23  # Cost is proportional to number of replacements
24  unit_cost = provided capacity / specific_cost
25  required cost >= unit_cost * num_replacements
26 }

```



(a) MCDP code equivalent to equations Eqs. (64) to (66). (b) Co-design diagram generated by PyMCDP from code in panel (b).



(c) Tree representation using par/series of diagram in panel (c).

Figure 8.: Panel (c) shows the co-design diagram generated from the code in (b). Panel (d) shows a tree representation (series, parallel) for the diagram. The edges show the types of functionality and resources. The leaves are labeled with the Python class used internally by the interpreter PyMCDP.

The choice of a battery can be modeled as a DPI (Fig. 9) with functionalities **capacity [J]** and **number of missions** and with resources **mass [kg]**, **cost [\$]** and “**maintenance**”, defined as the number of times that the battery needs to be replaced over the lifetime of the robot.

Each battery technology is described by the three parameters specific energy, specific cost, and lifetime (number of cycles):

$$\rho := \text{specific energy [Wh/kg]}, \quad (61)$$

$$\alpha := \text{specific cost [Wh/$]}, \quad (62)$$

$$c := \text{battery lifetime [\# of cycles]}. \quad (63)$$

The relation between functionality and resources is described by three nonlinear monotone constraints:

$$\text{mass} \geq \text{capacity}/\rho, \quad (64)$$

$$\text{maintenance} \geq \lceil \text{missions}/c \rceil, \quad (65)$$

$$\text{cost} \geq \lceil \text{missions}/c \rceil (\text{capacity}/\alpha). \quad (66)$$

Figure 8a shows the MCDPL code that describes the model corresponding to Eqs. (64) to (66). The diagram in Fig. 8b is automatically generated from the code. Fig. 8c shows a tree representation of the diagram using the series/par operators.

Competing battery technologies

The parameters for the battery technologies used in this example are shown in Table 36.1.

technology	energy density [Wh/kg]	specific cost [Wh/\$]	operating life # cycles
NiMH	100	3.41	500
NiH2	45	10.50	20000
LCO	195	2.84	750
LMO	150	2.84	500
NiCad	30	7.50	500
SLA	30	7.00	500
LiPo	250	2.50	600
LFP	90	1.50	1500

Table 36.1.: Specifications of common batteries technologies

Each row of the table is used to describe a model as in Fig. 8a by plugging in the specific values in lines 12–14.

Given the different models, we can define their coproduct (Fig. 10a) using the MCDPL code in Fig. 10b.

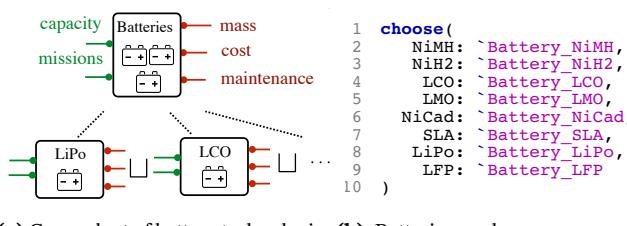
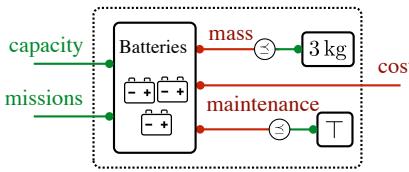


Figure 10.: The coproduct of design problems describes the choices among different technologies. The MCDPL keyword for the coproduct is “choose”.



Figure 9.: Interface of battery design problem.

(a) Co-design diagram that expresses hard constraints for **mass**.

```

1 mcdp {
2   provides capacity [J]
3   provides missions [R]
4
5   requires cost [$]
6
7   battery = instance `Batteries
8
9   provided capacity <= capacity provided by battery
10  provided missions <= missions provided by battery
11
12  mass required by battery <= 3 kg
13
14  ignore maintenance required by battery
15
16  required cost >= cost required by battery
17 }
```

(b) MCDPL code equivalent to diagram in (a).

Figure 11.: Composition of DPs can express hard constraints and precedence of objectives. In this case, there is a hard constraint on the **mass**. Because there is only one outgoing edge for **mass**, and the **cost** and **maintenance** are terminated by a dummy constraint ($x \leq \top$), the semantics of the diagram is that the objective is to minimize the **mass** as primary objective.

Introducing other variations or objectives

The design problem for the battery has two functionalities (**capacity** and **number of missions**) and three resources (**cost**, **mass**, and **maintenance**). Thus, it describes a family of multi-objective optimization problems, of the type “Given **capacity** and **missions**, minimize $\langle \text{cost}, \text{mass}, \text{maintenance} \rangle$ ”. We can further extend the class of optimization problems by introducing other hard constraints and by choosing which resource to prioritize. This can be done by composition of design problems; that is, by creating a larger DP that contains the original DP as a subproblem, and contains some additional degenerate DPs that realize the desired semantics.

For example, suppose that we would like to find the optimal solution(s) such that: 1) The mass does not exceed 3 kg; 2) The mass is minimized as a primary objective, while cost/maintenance are secondary objectives.

This semantics can be described by the co-design diagram in Fig. 11a, which contains two new symbols. The DP labeled “3 kg” implements the semantics of hard constraints. It has one functionality ($F = \mathbb{R}_{\geq 0}^{-\text{kg}}$) and zero resources ($R = \mathbf{1}$). The poset $\mathbf{1} = \{\langle \rangle\}$ has exactly two antichains: \emptyset and $\{\langle \rangle\}$. These represent “infeasible” and “feasible”, respectively. The DP is described by the map

$$h : \mathbb{R}_{\geq 0}^{-\text{kg}} \rightarrow \text{Anti } \mathbf{1}, \quad (67) \quad \text{---} \bullet [3 \text{ kg}]$$

$$f \mapsto \begin{cases} \{\langle \rangle\}, & \text{if } f \leq 3 \text{ kg}, \\ \emptyset, & \text{if } f > 3 \text{ kg}. \end{cases} \quad (68)$$

The block labeled “ \top ” is similarly defined and always returns “feasible”, so it has the effect of ignoring **cost** and **maintenance** as objectives. The only resource edge is the one for **mass**, which is then the only objective.

The MCDPL code is shown in Fig. 11b. Note the intuitive interface: the user can directly write “mass required by battery $\leq 3 \text{ kg}$ ” and “ignore maintenance required by battery”, which is compiled to “maintenance required by battery $\leq \top$ ”.

This relatively simple model for energetics already shows the complexity of CDPs. Figure 14 shows the optimal choice of the battery technology as a function of capacity and number of missions, for several slight variations of the problem that differ in constraints and objectives. For each battery technology, the figures show whether at each operating point the technology is the optimal choice, and

how many optimal choices there are. Some results are intuitive. For example, Fig. 14f shows that if the only objective is minimizing **mass**, then the optimal choice is simply the technology with the largest specific energy (LiPo). The decision boundaries become complex when considering nonlinear objectives. For example, Fig. 14d shows the case where the objective is to minimize the **cost**, which, defined by (66), is nonlinearly related to both **capacity** and **number of missions**. When considering multi-objective problems, such as minimizing jointly **(mass, cost)** (Fig. 14h) or **(mass, cost, maintenance)** (Fig. 14h), there are multiple non-dominated solutions.

From component to system co-design

The rest of the section reuses the battery DP into a larger co-design problem that considers the co-design of actuation together with energetics for a drone (Fig. 12a). We will see that the decision boundaries change dramatically, which shows that the optimal choices for a component cannot be made in isolation from the system.

The functionality of the drone's subsystem considered (Fig. 12a) are parametrized by **endurance**, **number of missions**, **extra power** to be supplied, and **payload**. We model "actuation" as a design problem with functionality **lift [N]** and resources **cost**, **mass** and **power**, and we assume that power is a quadratic function of lift (Fig. 13). Any other monotone map could be used.



The co-design constraints that combine energetics and actuation are

$$\text{battery capacity} \geq \text{total power} \times \text{endurance}, \quad (69)$$

$$\text{total power} = \text{actuation power} + \text{extra power},$$

$$\text{weight} = \text{total mass} \times \text{gravity},$$

$$\text{actuation lift} \geq \text{weight},$$

$$\text{labor cost} = \text{cost per replacement} \times \text{battery maintenance},$$

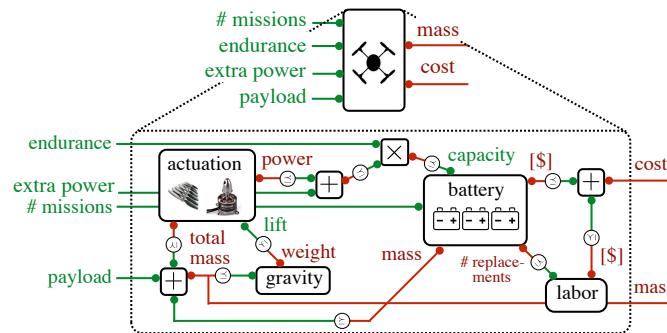
$$\text{total cost} = \text{battery cost} + \text{actuation cost} + \text{labor cost},$$

$$\text{total mass} = \text{battery mass} + \text{actuation mass} + \text{payload}. \quad (70)$$

The co-design graph contains recursive constraints: the power for actuation depends on the total weight, which depends on the mass of the battery, which depends on the capacity to be provided, which depends on the power for actuation. The MCDPL code for this model is shown in Fig. 12b; it refers to the previously defined models for "batteries" and "actuation".

The co-design problem is now complex enough that we can appreciate the compositional properties of CDPs to perform a qualitative analysis. Looking at Fig. 12a, we know that there is a monotone relation between any pair of functionality and resources, such as **payload** and **cost**, or **endurance** and **mass**, even without knowing exactly what are the models for battery and actuation.

When fully expanded, the co-design graph (too large to display) contains 110 nodes and 110 edges. It is possible to remove all cycles by removing only one edge (e.g., the **energy** \leq **capacity** constraint), so the design complexity (Def. 36.25) is equal to $\text{width}(\mathbb{R}_{\geq 0}) = 1$. The tree representation is shown in Fig. 12c. Because the co-design diagram contains cycles, there is a loop operator at the root of the tree, which implies we need to solve a least fixed point problem. Because of the scale of the problem, it is not possible to show the map **h** explicitly, like we did in (41) for the previous example. The least fixed point sequence converges to 64



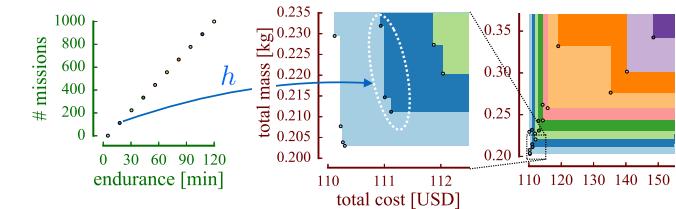
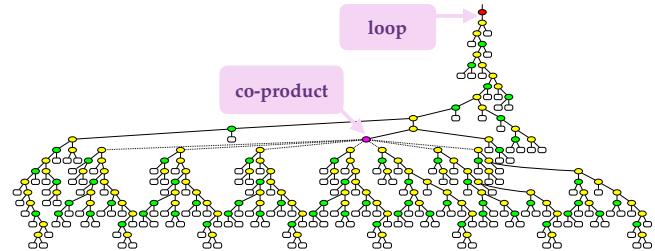
(a) Co-design diagram corresponding to Eqs. (69) to (70).

```

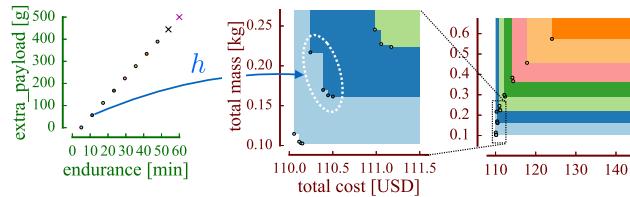
1 mcdp {
2   provides endurance [s]
3   provides extra_payload [kg]
4   provides extra_power [W]
5   provides num_missions [R]
6   provides velocity [m/s]
7
8   requires total_cost [$]
9   requires total_mass [g]
10
11   battery = instance `Batteries
12   actuation = instance `Actuation
13
14   total_power = extra_power +
15     power required by actuation
16
17   missions provided by battery >= num_missions
18
19   energy = provided endurance * total_power
20   capacity provided by battery >= energy
21
22   total_mass =
23     mass required by battery +
24     actuator mass required by actuation +
25     extra_payload)
26
27   required total_mass >= total_mass
28
29   gravity = 9.81 m/s^2
30   weight = total_mass * gravity
31
32   lift provided by actuation >= weight
33   velocity provided by actuation >= velocity
34
35   replacements = maintenance required by battery
36   cost_per_replacement = 1 $/s
37   labor_cost = cost_per_replacement * replacements
38
39   required total_cost >= (
40     cost required by actuation +
41     cost required by battery +
42     labor_cost)
43 }

```

(b) MCDPL code for Eqs. (69) to (70). The “instance” statements refer to (c) Tree representation for the CDP. Yellow/green rounded ovals are previously defined models for batteries (Fig. 10b) and actuation (not shown). series/par junctions. There is one coproduct junction, signifying the choice between different battery technologies, and one loop junction, at the root of the tree.



(d) Relation between endurance and number of missions and cost and mass.



(e) Relation between endurance and payload and cost and mass.

Figure 12.: In panel (c), the payload is fixed to 100 g and extra power is set to 1 W. In panel (d), the number of missions is fixed to 400 and extra power is set to 1 W. The last two values, marked with “X”, are not feasible.

bits machine precision in 50–100 iterations.

To visualize the multidimensional relation

$$h : \overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0}^s \times \overline{\mathbb{R}}_{\geq 0}^w \times \overline{\mathbb{R}}_{\geq 0}^g \rightarrow \text{Anti}(\overline{\mathbb{R}}_{\geq 0}^{\text{kg}} \times \overline{\mathbb{R}}_{\geq 0}^{\text{USD}}), \quad (71)$$

we need to project across 2D slices. Fig. 12d shows the relation when the functionality varies in a chain in the space `endurance/missions`, and Fig. 12e shows the results for a chain in the space `endurance/payload`.

Finally, Fig. 15 shows the optimal choices of battery technologies in the `endurance/missions` space, when one wants to minimize `mass`, `cost`, or $\langle \text{mass}, \text{cost} \rangle$. The decision boundaries are completely different from those in Fig. 14. This shows that it is not possible to optimize a component separately from the rest of the system, if there are cycles in the co-design diagram.



Figure 14.: This figure shows the optimal decision boundaries for the different battery technologies for the design problem “batteries”, defined as the coproduct of all battery technologies (Fig. 10). Each row shows a different variation of the problem. The first row (panels *a–b*) shows the case where the objective function is the product of $\langle \text{mass}, \text{cost}, \text{maintenance} \rangle$. The shape of the symbols shows how many minimal solutions exists for a particular value of the functionality $\langle \text{capacity}, \text{missions} \rangle$. In this case, there are always three or more minimal solutions. The second row (panels *c–d*) shows the decision boundaries when minimizing only the scalar objective cost , with a hard constraint on mass . The hard constraints make some combinations of the functionality infeasible. Note how the decision boundaries are nonconvex, and how the formalism allows defining slight variations of the problem.

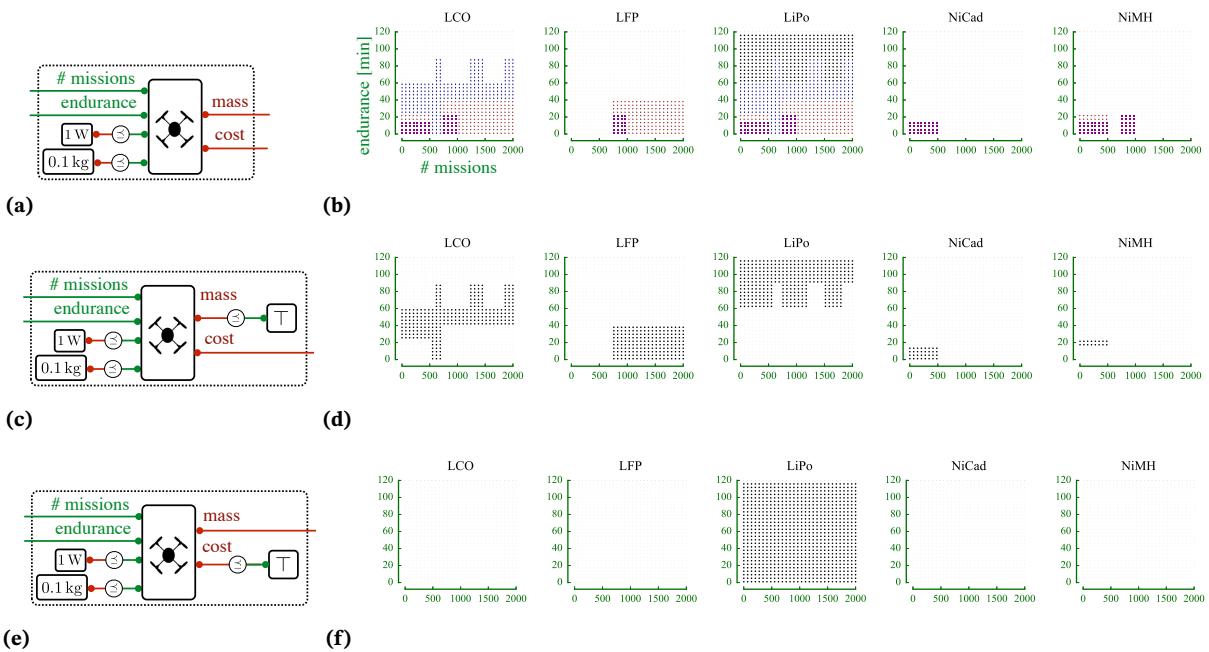


Figure 15.: This figure shows the decision boundaries for the different values of battery technologies for the integrated actuation-energetics model described in Fig. 12. Please see the caption of Fig. 14 for an explanation of the symbols. Notice how in most cases the decision boundaries are different from those in Fig. 14: this is an example in which one component cannot be optimized by itself without taking into account the rest of the system.

36.6. Complexity of the solution

Complexity of fixed point iteration

Consider first the case of an DP that can be described as $\mathbf{d} = \text{loop}(\mathbf{d}_0)$, where \mathbf{d}_0 is an DP that is described only using the series and par operators. Suppose that \mathbf{d}_0 has resource space \mathbf{R} . Then evaluating h for \mathbf{d} is equivalent to computing a least fixed point iteration on the space of antichains $\text{Anti } \mathbf{R}$. This allows to give worst-case bounds on the number of iterations.

Proposition 36.21. Suppose that $\mathbf{d} = \text{loop}(\mathbf{d}_0)$ and \mathbf{d}_0 has resource space \mathbf{R}_0 and evaluating h_0 takes at most c computation. Then we can obtain the following bounds for the algorithm's resources usage:

memory	$O(\text{width}(\mathbf{R}_0))$
number of steps	$O(\text{height}(\text{Anti } \mathbf{R}_0))$
total computation	$O(\text{width}(\mathbf{R}_0) \cdot \text{height}(\text{Anti } \mathbf{R}_0) \cdot c)$

Proof. The memory utilization is bounded by $\text{width}(\mathbf{R}_0)$, because the state is an antichain, and $\text{width}(\mathbf{R}_0)$ is the size of the largest antichain. The iteration happens in the space $\text{Anti } \mathbf{R}_0$, and we are constructing an ascending chain, so it can take at most $\text{height}(\text{Anti } \mathbf{R}_0)$ steps to converge. Finally, in the worst case the map h_0 needs to be evaluated once for each element of the antichain for each step. \square

These worst case bounds are strict.

Example 36.22. Consider solving $\mathbf{d} = \text{loop}(\mathbf{d}_0)$ with \mathbf{d}_0 defined by $h_0 : \langle \langle \rangle, x \mapsto x + 1$ with $x \in \overline{\mathbb{N}}$. Then the least fixed point equation is equivalent to solving $\min \{x : \Psi(x) \leq x\}$ with $\Psi : x \mapsto x + 1$. The iteration $R_{k+1} = \Psi(R_k)$ converges to \top in $\text{height}(\overline{\mathbb{N}}) = \aleph_0$ steps.

Remark 36.23. Making more precise claims requires additional more restrictive assumptions on the spaces involved. For example, without adding a metric on \mathbf{R} , it is not possible to obtain properties such as linear or quadratic convergence.

Remark 36.24 (Invariance to re-parameterization). All the results given in this paper are invariant to any order-preserving re-parameterization of all the variables involved.

Relating complexity to the graph properties

Prop. 36.21 above assumes that the DP is already in the form $\mathbf{d} = \text{loop}(\mathbf{d}_0)$, and relates the complexity to the poset \mathbf{R}_0 . Here we relate the results to the graph structure of an DP.

Take a DP $\mathbf{d} = \langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$. To put \mathbf{d} in the form $\mathbf{d} = \text{loop}(\mathbf{d}_0)$ according to the procedure in Section 36.7, we need to find an arc feedback set (AFS) of the graph $\langle \mathcal{V}, \mathcal{E} \rangle$. Given an AFS $F \subset \mathcal{E}$, then the resource space \mathbf{R} for a \mathbf{d}_0 such that $\mathbf{d} = \text{loop}(\mathbf{d}_0)$ is the product of the resources spaces along the edges: $\mathbf{R}_0 = \prod_{e \in F} \mathbf{R}_e$.

Now that we have a relation between the AFS and the complexity of the iteration, it is natural to ask what is the optimal choice of AFS—which, so far, was left as an arbitrary choice. The AFS should be chosen as to minimize one of the performance measures in Prop. 36.21.

Of the three performance measures in Prop. 36.21, the most fundamental appears to be $\text{width}(\mathbf{R}_0)$, because that is also an upper bound on the number of distinct minimal solutions. Hence, we can call it “design complexity” of the DP.

Definition 36.25 (Design complexity)

Given a graph $\langle \mathcal{V}, \mathcal{E} \rangle$ and a labeling of each edge $e \in \mathcal{E}$ with a poset \mathbf{R}_e , the *design complexity* $\text{DC}(\langle \mathcal{V}, \mathcal{E} \rangle)$ is defined as

$$\text{DC}(\langle \mathcal{V}, \mathcal{E} \rangle) = \min_{F \text{ is an AFS}} \text{width}\left(\prod_{e \in F} \mathbf{R}_e\right). \quad (72)$$

In general, the width and height of posets are not additive with respect to products; therefore, this problem does not reduce to any of the known variants of the minimum arc feedback set problem, in which each edge has a weight and the goal is to minimize the sum of the weights.

Considering relations with infinite cardinality

This analysis shows the limitations of the simple solution presented so far: it is easy to produce examples for which $\text{width}(\mathbf{R}_0)$ is infinite, so that one needs to represent a continuum of solutions.

Example 36.26. Suppose that the platform to be designed must travel a distance d [m], and we need to choose the endurance T [s] and the velocity v [m/s]. The relation among the quantities is $d \leq T v$. This is a design problem described by the map

$$\begin{aligned} h : \overline{\mathbb{R}}_{\geq 0} &\rightarrow \text{Anti } \overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0}, \\ d &\mapsto \{ \langle T, v \rangle \in \overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0} : d = T v \}. \end{aligned}$$

For each value of d , there is a continuum of solutions.

One approach to solving this problem would be to discretize the functionality \mathbf{F} and the resources \mathbf{R} by sampling and/or coarsening. However, sampling and coarsening makes it hard to maintain completeness and consistency.

One effective approach that we will develop later is to *approximate the design problem itself*, rather than the spaces \mathbf{F}, \mathbf{R} , which are left as possibly infinite. The basic idea is that an infinite antichain can be bounded from above and above by two antichains that have a finite number of points. This idea leads to an algorithm that, given a prescribed computation budget, can compute an inner and outer approximation to the solution antichain.

36.7. Decomposition of CDPs

This section shows how to describe an arbitrary interconnection of design problems using only three composition operators. More precisely, for each CDPI with a set of atoms \mathcal{V} , there is an equivalent one that is built from series/par/loop applied to the set of atoms \mathcal{V} plus some extra “plumbing” (identities, multiplexers).

Proposition 36.27. Given a CDPI $\langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, we can find an equivalent CDPI obtained by applying the operators par/series/loop to a set of atoms \mathcal{V}' that contains \mathcal{V} plus a set of trivial DPIs. Furthermore, one instance of loop is sufficient.

Proof. We show this constructively. We will temporarily remove all cycles from the graph, to be reattached later. To do this, find an *arc feedback set* (AFS) $F \subseteq \mathcal{E}$. An AFS is a set of edges that, when removed, remove all cycles from the graph (see [7]). For example, the CDPI represented in Fig. 18a has a minimal AFS that contains the edge $c \rightarrow a$ (Fig. 18b).

Remove the AFS F from \mathcal{E} to obtain the reduced edge set $\mathcal{E}' = \mathcal{E} \setminus F$. The resulting graph $\langle \mathcal{V}, \mathcal{E}' \rangle$ does not have cycles, and can be written as a series-parallel graph, by applying the operators par and series from a set of nodes \mathcal{V}' . The nodes \mathcal{V}' will contain \mathcal{V} , plus some extra “connectors” that are trivial DPIs.

Find a weak topological ordering of \mathcal{V} . Then the graph $\langle \mathcal{V}, \mathcal{E}' \rangle$ can be written as the series of $|\mathcal{V}|$ subgraphs, each containing one node of \mathcal{V} . In the example, the weak topological ordering is $\langle a, b, c \rangle$ and there are three subgraphs (Fig. 16).

Each subgraph can be described as the parallel interconnection of a node $v \in \mathcal{V}$ and some extra connectors. For example, the second subgraph in the graph can be written as the parallel interconnection of node b and the identity $\text{Triv}(\text{id})$ (Fig. 17).

After this is done, we just need to “close the loop” around the edges in the AFS F to obtain a CDPI that is equivalent to the original one. Suppose the AFS F contains only one edge. Then one instance of the loopb operator is sufficient (Fig. 19a). In this example, the tree representation (Fig. 19b) is

$$\text{loopb(series(series(a, par(id, b)), c)).} \quad (73)$$

If the AFS contains multiple edges, then, instead of closing one loop at a time, we can always rewrite multiple nested loops as only one loop by taking the product of the edges. For example, a diagram like the one in Fig. 20a can be rewritten as Fig. 20b. This construction is analogous to the construction used for the analysis of process networks [14] (and any other construct involving a traced monoidal category). Therefore, it is possible to describe an arbitrary graph of design problems using only one instance of the loop operator. \square

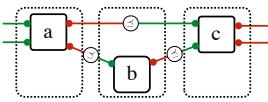


Figure 16.

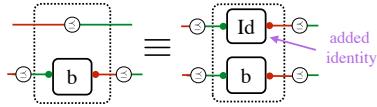


Figure 17.

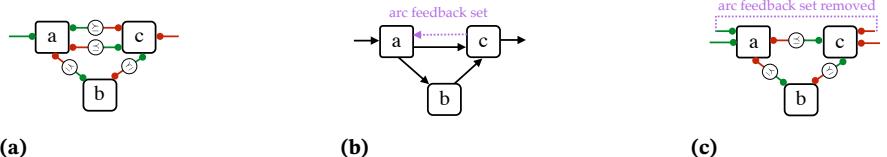


Figure 18.: An example co-design diagram with three nodes $\mathcal{V} = \{a, b, c\}$, in which a minimal arc feedback set is $\{c \rightarrow a\}$.

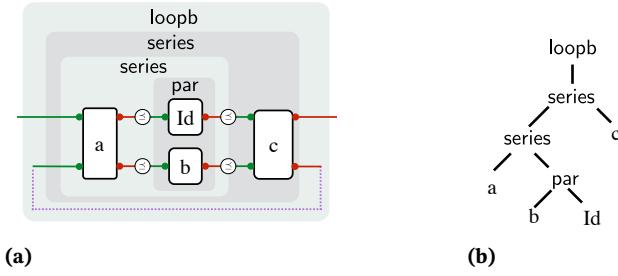


Figure 19.: Tree representation for the co-design diagram in Fig. 18a.

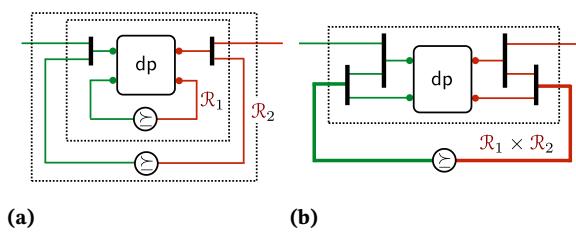


Figure 20.: If there are nested loops in a co-design diagram, they can be rewritten as one loop, by taking the product of the edges.



37. Monads

In this chapter, we introduce the concept of monad. We introduce monads using the computer science perspective, in which they are interpreted as “generalized computation”.

37.1 Generalized objects and operations	526
37.2 Monads	531
37.3 The Kleisli construction	535
37.4 Algebras of a monad	537
37.5 Monads from Adjunctions	541

When one thinks about Switzerland and the alps, one usually thinks about snow. Starting from 1,500 m height, most areas are covered in snow during winter (from December to March). Some areas (above 3,000 m), are *always* covered in snow.

37.1. Generalized objects and operations

The definition of monads is very powerful because it is very abstract and can fit many possible scenarios. Before getting to the formal definition, it is useful to build up some intuition using several examples.

Monads are a type of algebraic structure that is well-suited to represent generalized objects and operations. We begin by giving several examples.

Modeling nondeterministic uncertainty

For the engineer, one intuitive scenario where generalization is necessary is handling uncertainty. We have seen the category **Set** of sets and functions between sets. A function

$$f : X \rightarrow Y \quad (1)$$

between sets is “deterministic”, in the sense that, given as input an element of X , it always produces the same output in Y .

Suppose now we want to deal with nondeterministic functions: functions that return, for each input, a *set* of possible values. We can model nondeterministic functions from X to Y as functions of the type

$$f : X \rightarrow \text{Pow } Y. \quad (2)$$

Note that this is a generalization, in the sense that any deterministic function is a special nondeterministic function. For example, the function

$$\begin{aligned} \alpha : \mathbb{N} &\rightarrow \mathbb{N}, \\ x &\mapsto x^2, \end{aligned} \quad (3)$$

can be rewritten as the function

$$\begin{aligned} \alpha' : \mathbb{N} &\rightarrow \text{Pow } \mathbb{N}, \\ x &\mapsto \{x^2\}, \end{aligned} \quad (4)$$

which maps each element to a singleton set.

Once we have these generalized functions, we really want them to form a category, so that they can compose. To do this, we need extra information: additional structure. So far we know the rules of composition for functions:

$$\begin{array}{c} f : X \rightarrow Y \qquad g : Y \rightarrow Z \\ \hline \hline (f ; g) : X \rightarrow Z, \\ x \mapsto g(f(x)), \end{array} \quad (5)$$

but this does not help for the generalized functions. How can we compose nondeterministic functions? What to fill in the space below?

$$\begin{array}{c} f : X \rightarrow \text{Pow } Y \qquad g : Y \rightarrow \text{Pow } Z \\ \hline \hline (f ; g) : X \rightarrow \text{Pow } Z, \\ x \mapsto ?. \end{array} \quad (6)$$

There is one natural way to define this operation. Did you notice the following?

Lemma 37.1. Functions from X to $\text{Pow } Y$ are in one-to-one correspondence with relations from X to Y .

Therefore, we expect that nondeterministic functions can be composed like rela-

tions. If we have $f : X \rightarrow \text{Pow } Y$ and $g : Y \rightarrow \text{Pow } Z$ then we define the composite non-deterministic function to be

$$(f \circ g) : X \rightarrow \text{Pow } Z, \quad x \mapsto \bigcup_{y \in f(x)} g(y). \quad (7)$$

Figure 1.

We should also check that this composition is associative; however, this comes automatically from the fact that we already know that **Rel** is a category.

To summarize:

- ▷ We wanted to extend **Set** from functions $X \rightarrow Y$ to nondeterministic functions of type $X \rightarrow \text{Pow } Y$.
- ▷ To do this, we needed three things:
 1. The particular choice of **Pow** as what maps a set to another set.
 2. A way to lift a function of type $X \rightarrow Y$ to a function of type $X \rightarrow \text{Pow } Y$, so that we can say that nondeterministic functions are a generalization of deterministic functions. This lifting operation is a family of functions of type

$$\text{lift}_{X,Y} : (X \rightarrow Y) \rightarrow (X \rightarrow \text{Pow } Y). \quad (8)$$

3. A way to define composition, through a map, traditionally called “fish”, of type

$$\text{fish}_{X,Y,Z} : (X \rightarrow \text{Pow } Y) \times (Y \rightarrow \text{Pow } Z) \rightarrow (X \rightarrow \text{Pow } Z) \quad (9)$$

- ▷ And we needed these pieces to satisfy the conditions:

1. **fish** is associative. This ensures the generalized functions form a semicategory.
2. The composition of the lifted functions are the lifting of the composition:

$$\frac{f : X \rightarrow \text{Pow } Y \quad g : Y \rightarrow \text{Pow } Z}{\text{lift}_{X,Y}(f) \circ \text{lift}_{Y,Z}(g) = \text{lift}_{X,Z}(f \circ g)}. \quad (10)$$

This ensures that inside the generalized functions, the composition of regular functions continues to work as it should.

Modeling interval uncertainty

We continue to build intuition considering another type of uncertainty.

We have seen the category **Pos** of posets and monotone functions between posets (Def. 14.8).

In this category it is easy to propagate uncertainty if the uncertain sets are represented by intervals.

Recall that for a poset **P**, we can define the poset of intervals **Arr P** (see Def. 6.10) and **Tw P** (see Def. 6.9).

Analogously to the previous case, here we want to generalize from monotone functions to nondeterministic monotone functions, where the uncertainty is represented by intervals.

We recall the notation for intervals:

- ▷ The interval

$$\{x : a \leq x \leq b\} \quad (11)$$

is denoted

$$[a, b]. \quad (12)$$

▷ $\text{L}\cdot$ and $\text{U}\cdot$ extract the lower and upper bound from the interval, so that we have

$$\text{L}[a, b] = a \quad (13)$$

$$\text{U}[a, b] = b \quad (14)$$

Because an interval is defined by two values, a function that returns an interval is a pair of functions, whose results are constrained to be ordered. For example, one such function is

$$\begin{aligned} f : \mathbb{R} &\rightarrow \text{Arr}(\mathbb{R}, \leq), \\ x &\mapsto [x - 1, x + 1]. \end{aligned} \quad (15)$$

In the example above, we always have $x - 1 \leq x + 1$.

We now retrace the steps of the previous example.

First, we need to ensure that we can see regular monotone functions as special cases of interval functions. For example, the function

$$\begin{aligned} \alpha : \langle \mathbb{N}, \leq \rangle &\rightarrow \langle \mathbb{N}, \leq \rangle, \\ x &\mapsto x^2, \end{aligned} \quad (16)$$

can be rewritten as the function

$$\begin{aligned} \alpha' : \langle \mathbb{N}, \leq \rangle &\rightarrow \text{Arr}(\mathbb{N}, \leq), \\ x &\mapsto [x^2, x^2], \end{aligned} \quad (17)$$

Generically, this is the definition of the `lift` function

$$\begin{aligned} \text{lift}_{X,Y} : (X \rightarrow \text{Pos } Y) &\rightarrow (X \rightarrow \text{Pos } \text{Arr } Y), \\ f &\mapsto \left\{ \begin{array}{l} \text{lift}_{X,Y} f : X \rightarrow \text{Arr } Y \\ x \mapsto [f(x), f(x)] \end{array} \right.. \end{aligned} \quad (18)$$

What is the “fish” function? Note that an interval-valued monotone map is also a special relation. Therefore, we want that composition continues to work in the same manner.

Therefore, we obtain for the `fish` operation:

$$\begin{array}{c} f : X \rightarrow \text{Arr } Y \quad g : Y \rightarrow \text{Arr } Z \\ \hline (f ; g) : X \rightarrow \text{Arr } Z \\ x \mapsto [\text{L}_g(\text{L}_f(x)), \text{U}_g(\text{U}_f(x))] \end{array} \quad (19)$$

Upper sets

We have seen that to solve DPs we encountered functions of type

$$f : \text{F} \rightarrow \text{Pos } \text{UR} \quad (20)$$

We can see this as another example of generalization from a function

$$f : X \rightarrow \text{Pos } Y \quad (21)$$

to functions

$$f : X \rightarrow \text{Pos } \text{UY} \quad (22)$$

Once again we see these as particular types of relations.

The **lift** operation is

$$\begin{array}{lcl} \text{lift}_{X,Y} : (X \rightarrow \text{pos } Y) & \rightarrow & (X \rightarrow \text{Pos } UY) \\ f & \mapsto & \left\{ \begin{array}{l} \text{lift}_{X,Y} f : X \rightarrow UY \\ x \mapsto \uparrow f(x) \end{array} \right. \end{array} \quad (23)$$

The **fish** operation is

$$\begin{array}{c} f : X \rightarrow UY \quad g : Y \rightarrow UZ \\ \hline (f \circ g) : X \rightarrow UZ \\ x \mapsto \bigcup_{y \in f(x)} g(y) \end{array} \quad (24)$$

Note that the expression is the same as in (7) - only we are guaranteed to obtain upper sets.

Keeping track of resource usage

As another example, we consider the case where we want to attach additional information to a category.

Suppose we want to consider not functions, but *procedures* which have resource consumption. A function is a mathematical entity - a procedure is a program that *implements* a function. Suppose we want to model execution time - and, that execution time might depend on the input of the procedure.

In this case, we would like to extend a function

$$f : X \rightarrow Y \quad (25)$$

into a procedure

$$f : X \rightarrow (Y \times \mathbb{R}_{\geq 0}) \quad (26)$$

which gives both result and the execution time.

We consider the ideal functions to be procedures that have zero execution time.

We can define the **lift** map as follows:

$$\begin{array}{lcl} \text{lift}_{X,Y} : (X \rightarrow Y) & \rightarrow & (X \rightarrow Y \times \mathbb{R}_{\geq 0}) \\ f & \mapsto & \left\{ \begin{array}{l} \text{lift}_{X,Y} f : X \rightarrow Y \times \mathbb{R}_{\geq 0} \\ x \mapsto \langle x, 0 \rangle \end{array} \right. \end{array} \quad (27)$$

As for the fish function, we have

$$\begin{array}{c} f : X \rightarrow (Y \times \mathbb{R}_{\geq 0}) \quad g : Y \rightarrow (Z \times \mathbb{R}_{\geq 0}) \\ \hline (f \circ g) : X \rightarrow (Z \times \mathbb{R}_{\geq 0}) \\ x \mapsto \langle g_1(f_1(x)), g_2(f_1(x)) + f_2(x) \rangle \end{array} \quad (28)$$

Note that we can recover the naked function by taking the first component of the tuple.

Generalization with monoid

This can be further generalized from the real numbers to an arbitrary monoid structure

$$\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle. \quad (29)$$

For the `lift` function we have

$$\begin{aligned} \text{lift}_{X,Y} : (X \rightarrow Y) &\rightarrow (X \rightarrow Y \times M), \\ f &\mapsto \begin{cases} \text{lift}_{X,Y} f : X \rightarrow Y \times M \\ x \mapsto \langle x, \text{id}_M \rangle \end{cases} . \end{aligned} \quad (30)$$

As for the fish function we have

$$\begin{array}{c} f : X \rightarrow (Y \times M) \qquad g : Y \rightarrow (Z \times M) \\ \hline (f ; g) : X \rightarrow (Z \times M) \\ x \mapsto \langle g_1(f_1(x)), g_2(f_1(x)) \circ_M f_2(x) \rangle . \end{array} \quad (31)$$

37.2. Monads

Definition 37.2 (Monad)

Let \mathbf{C} be a category. A *monad* on \mathbf{C} is specified by:

Constituents

1. A functor $M : \mathbf{C} \rightarrow \mathbf{C}$;
2. A natural transformation $\text{mu} : M \circ M \Rightarrow M$, called the *composition* or *multiplication*;
3. A natural transformation $\text{un} : \text{id}_{\mathbf{C}} \Rightarrow M$, called the *unit*.

Conditions

1. *Associativity*: the following diagram must commute:

$$\begin{array}{ccc} M \circ M \circ M & \xrightarrow{M \text{mu}} & M \circ M \\ \Downarrow \text{mu}M & & \Downarrow \text{mu} \\ M \circ M & \xrightarrow{\text{mu}} & M \end{array} \quad (32)$$

2. *Left and right unitality*: the following diagrams must commute:

$$\begin{array}{ccc} M & \xrightarrow{\text{un}M} & M \circ M \\ \text{id}_M \swarrow & & \downarrow \text{mu} \\ M & & M \end{array} \quad \begin{array}{ccc} M & \xrightarrow{M \text{un}} & M \circ M \\ \text{id}_M \swarrow & & \downarrow \text{mu} \\ M & & M \end{array} \quad (33)$$

Remark 37.3. In terms of components, the unitality conditions state that for every object $X \in \text{Ob}_{\mathbf{C}}$, the following diagram commutes:

$$\begin{array}{ccc} M(X) & \xrightarrow{\text{un}MX} & (M \circ M)(X) \\ \text{id}_M \searrow & & \downarrow \text{mu}_X \\ M(X) & & M(X) \end{array} \quad \begin{array}{ccc} M(X) & \xrightarrow{M \text{un}_X} & (M \circ M)(X) \\ \text{id}_M \searrow & & \downarrow \text{mu}_X \\ M(X) & & M(X) \end{array} \quad (34)$$

The associativity condition states that for every object $X \in \text{Ob}_{\mathbf{C}}$, the following diagram commutes:

$$\begin{array}{ccc} (M \circ M \circ M)(X) & \xrightarrow{M \text{mu}_X} & (M \circ M)(X) \\ \downarrow \text{mu}_{M(X)} & & \downarrow \text{mu}_X \\ (M \circ M)(X) & \xrightarrow{\text{mu}_X} & M(X) \end{array} \quad (35)$$

Graded exercise K.1 (PowersetMonad)

The aim of this exercise is to prove in full detail that the powerset functor $\text{Pow} : \mathbf{Set} \rightarrow \mathbf{Set}$ is a monad, when equipped with the following *unit* and *multiplication*. We define $\text{un} : \text{id}_{\mathbf{Set}} \Rightarrow \text{Pow}$ and $\text{mu} : \text{Pow} \circ \text{Pow} \Rightarrow \text{Pow}$ in terms of components: given an object $A \in \mathbf{Set}$, let

$$\begin{aligned} \text{mu}_A : (\text{Pow} \circ \text{Pow})(A) &\rightarrow \text{Pow}(A) \\ D &\mapsto \bigcup_{S \in D} S \end{aligned} \quad (36)$$

and

$$\begin{aligned} \text{un}_{\mathbf{A}} : \mathbf{A} &\rightarrow \text{Pow}(\mathbf{A}), \\ x &\mapsto \{x\}. \end{aligned} \quad (37)$$

To show that $\langle \text{Pow}, \text{mu}, \text{un} \rangle$ is a monad,

1. prove that un , as defined in components in (37), is a natural transformation;
2. prove that mu , as defined in components in (36), is a natural transformation;
3. prove that un and mu satisfy the associativity condition and the left and right unitality conditions given in Def. 37.2. For this, work in components, as in Remark 37.3.

Graded exercise K.2 (`FinProbMonad`)

Let \mathbf{Set} denote the category of sets, let $\text{Id}_{\mathbf{Set}} : \mathbf{Set} \rightarrow \mathbf{Set}$ be the identity functor, and let $\mathcal{P} : \mathbf{Set} \rightarrow \mathbf{Set}$ be the functor defined as follows. Given a set \mathbf{A} , the set $\mathcal{P}(\mathbf{A})$ is the set of finitely supported probability measures on \mathbf{A} . We recall that these are functions $p : \mathbf{A} \rightarrow [0, 1]$ with only finitely-many non-zero values and such that these values sum to 1:

$$\sum_{x \in \mathbf{A}} p(x) = 1. \quad (38)$$

On morphisms, \mathcal{P} is defined thus: for any function $f : \mathbf{A} \rightarrow \mathbf{B}$, the function $\mathcal{P}(f) : \mathcal{P}(\mathbf{A}) \rightarrow \mathcal{P}(\mathbf{B})$ is

$$\begin{aligned} \mathcal{P}(f)(p) : \mathbf{B} &\rightarrow [0, 1], \\ y &\mapsto \sum_{x \in f^{-1}(\{y\})} p(x). \end{aligned} \quad (39)$$

Your tasks:

1. Given a set \mathbf{A} , let $\delta_{\mathbf{A}} : \mathbf{A} \rightarrow \mathcal{P}\mathbf{A}$ be the function

$$\delta_{\mathbf{A}}(x)(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{else.} \end{cases} \quad x, y \in \mathbf{A}. \quad (40)$$

Check that the collection of functions $\{\delta_{\mathbf{A}}\}_{\mathbf{A} \in \text{Ob}_{\mathbf{Set}}}$ defines a natural transformation $\delta : \text{Id}_{\mathbf{Set}} \Rightarrow \mathcal{P}$.

2. Given a set \mathbf{A} , let $\mu_{\mathbf{A}} : \mathcal{P}\mathcal{P}\mathbf{A} \rightarrow \mathcal{P}\mathbf{A}$ be the function defined by

$$\mu_{\mathbf{A}}(f)(x) = \sum_{p \in \mathcal{P}\mathbf{A}} f(p)p(x) \quad f \in \mathcal{P}\mathcal{P}\mathbf{A}, x \in \mathbf{A}. \quad (41)$$

Check that the collection of functions $\{\mu_{\mathbf{A}}\}_{\mathbf{A} \in \text{Ob}_{\mathbf{Set}}}$ defines a natural transformation $\mu : \mathcal{P}\mathcal{P} \Rightarrow \mathcal{P}$.

3. Show that $\langle \mathcal{P}, \mu, \delta \rangle$ defines a monad on the category \mathbf{Set} .

Example: interval construction In the following, we analyze the interval construction and show that it gives rise to a monad. Given \mathbf{Pos} , we define the action of the \mathbf{Arr} functor on objects (posets) as

$$\begin{aligned} \mathbf{Arr} : \text{Ob}_{\mathbf{Pos}} &\rightarrow \text{Ob}_{\mathbf{Pos}} \\ P &\mapsto \mathbf{Arr} P \end{aligned} \quad (42)$$

and on morphisms as

$$\begin{aligned} \mathbf{Arr} f : \mathbf{Arr} P &\rightarrow \mathbf{Arr} Q \\ [a, b] &\mapsto [f(a), f(b)]. \end{aligned} \tag{43}$$

Furthermore, we define the multiplication as

$$\begin{aligned} \mathbf{mu}_X : \mathbf{Arr}(\mathbf{Arr} X) &\rightarrow \mathbf{Arr} X \\ [[\alpha, \beta], [\gamma, \delta]] &\mapsto [\min(\alpha, \gamma), \max(\beta, \delta)] = [\alpha, \delta], \end{aligned} \tag{44}$$

and the unit as

$$\begin{aligned} \mathbf{un}_X : X &\rightarrow \mathbf{Arr} X \\ x &\mapsto [x, x]. \end{aligned} \tag{45}$$

First, we need to show that both the multiplication and the unit are indeed natural transformations. We start with multiplication. On the one hand, we have:

$$\begin{aligned} ((\mathbf{Arr} ; \mathbf{Arr})(f) ; \mathbf{mu}_Y)([[a, b], [c, d]]) &= \mathbf{mu}_Y([[f(a), f(b)], [f(c), f(d)]]]) \\ &= [f(a), f(d)]. \end{aligned} \tag{46}$$

On the other hand we have:

$$\begin{aligned} (\mathbf{mu}_X ; \mathbf{Arr})([[a, b], [c, d]]) &= (\mathbf{Arr} f)([a, d]) \\ &= [f(a), f(d)]. \end{aligned} \tag{47}$$

The equivalence of (46) and (47) proves naturality. We now check that the unit defines a natural transformation. We have

$$\begin{aligned} (f ; \mathbf{un}_Y)(x) &= f(x) ; \mathbf{un}_Y \\ &= [f(x), f(x)] \end{aligned} \tag{48}$$

and

$$\begin{aligned} (\mathbf{un}_X ; \mathbf{Arr} f)(x) &= (\mathbf{Arr} f)([x, x]) \\ &= [f(x), f(x)] \end{aligned} \tag{49}$$

Again, equivalence of (48) and (49) proves naturality.

We now need to check associativity and left and right unitality. We start with associativity. To clearly see the property holding, we need to define intermediate quantities. We have

$$\begin{aligned} \mathbf{Arr} \mathbf{mu}_X : \mathbf{Arr}(\mathbf{Arr}(\mathbf{Arr}(X))) &\rightarrow \mathbf{Arr}(\mathbf{Arr} X) \\ [[[[\alpha_1, \beta_1], [\gamma_1, \delta_1]], [[\alpha_2, \beta_2], [\gamma_2, \delta_2]]]] &\mapsto [[[\alpha_1, \delta_1], [\alpha_2, \delta_2]]] \end{aligned} \tag{50}$$

and

$$\begin{aligned} \mathbf{mu}_{\mathbf{Arr} X} : \mathbf{Arr}(\mathbf{Arr}(\mathbf{Arr} X)) &\rightarrow \mathbf{Arr}(\mathbf{Arr} X), \\ [[[[\alpha_1, \beta_1], [\gamma_1, \delta_1]], [[\alpha_2, \beta_2], [\gamma_2, \delta_2]]]] &\mapsto [[[[\alpha_1, \beta_1], [\gamma_2, \delta_2]]]] \end{aligned} \tag{51}$$

With this in mind, it is easy to see that the following diagram commutes.

For left and right unitality, we need to work out two additional quantities:

$$\begin{aligned} \mathbf{un}_{\mathbf{Arr} X} : \mathbf{Arr} X &\rightarrow \mathbf{Arr} \mathbf{Arr} X \\ [a, b] &\mapsto [[a, b], [a, b]], \end{aligned} \tag{52}$$

Something incorrect or unclear or missing? Report issues on GitHub by clicking here.

and

$$\begin{aligned}\mathbf{Arr}(\mathbf{un}_X) : \mathbf{Arr} X &\rightarrow \mathbf{Arr}(\mathbf{Arr} X) \\ [a, b] &\mapsto [[a, a], [b, b]].\end{aligned}\tag{53}$$

Monads, computer science definition

For reference, we give the definition of a monad in functional programming, as a set of operations with particular types.

Definition 37.4 (Monad in functional programming)

A monad

$$\langle \mathbf{return}, \mathbf{join}, \mathbf{fmap}, \mathbf{bind}, \mathbf{fish}, \mathbf{lift} \rangle\tag{54}$$

is a set of operations with the following signature:

$$\mathbf{return} : X \rightarrow MX\tag{55}$$

$$\mathbf{lift} : (X \rightarrow Y) \rightarrow (X \rightarrow MY)\tag{56}$$

$$\mathbf{fish} : (X \rightarrow MY) \rightarrow (Y \rightarrow MZ) \rightarrow (X \rightarrow MZ)\tag{57}$$

$$\mathbf{join} : MMX \rightarrow MX\tag{58}$$

$$\mathbf{fmap} : (X \rightarrow Y) \rightarrow (MX \rightarrow MY)\tag{59}$$

$$\mathbf{bind} : MX \rightarrow (X \rightarrow MY) \rightarrow MY\tag{60}$$

These maps satisfy the equivalent axioms of unitality and associativity:

- ▷ `return` is a left identity for `bind`;
- ▷ `return` is a right identity for `bind`;
- ▷ `bind` is associative.

37.3. The Kleisli construction

We return now to the discussion from the opening section of this chapter, in order to spell out further the relationship to monads.

Definition 37.5 (Kleisli morphisms)

Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} , and let $X, Y \in \text{Ob}_{\mathbf{C}}$. A *Kleisli morphism* $X \rightarrow Y$ is a morphism of \mathbf{C} of the form $X \rightarrow MY$.

Definition 37.6 (Kleisli composition)

Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} , let $X, Y, Z \in \text{Ob}_{\mathbf{C}}$, and let $f : X \rightarrow MY$ and $g : Y \rightarrow MZ$ be morphisms in \mathbf{C} (so, they are Kleisli morphisms). Their *Kleisli composition* is the morphism in \mathbf{C} given by the composition

$$X \xrightarrow{f} M(Y) \xrightarrow{Mg} (M \circ M)(Z) \xrightarrow{\text{mu}_Z} M(Z). \quad (61)$$

Definition 37.7 (Kleisli category)

Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} . The *Kleisli category* \mathbf{C}_M of the monad M is specified by:

1. *Objects*: $\text{Ob}(\mathbf{C}_M) := \text{Ob}(\mathbf{C})$;
2. *Morphisms*: $\text{Hom}_{\mathbf{C}_M}(X, Y) := \text{Hom}_{\mathbf{C}}(X, M(Y))$;
3. *Identities*: $\text{id}_X := \text{un}_X$;
4. *Composition*: Kleisli composition.

Graded exercise K.3 (HwkRelKleisli)

Let \mathbf{Set} denote the category of sets and $\text{Pow} : \mathbf{Set} \rightarrow \mathbf{Set}$ the powerset functor which assigns, to any given set \mathbf{A} , the set of subsets of \mathbf{A} . The endofunctor Pow may be equipped with the structure of a monad $\langle \text{Pow}, \text{mu}, \text{un} \rangle$, where the components of the natural transformations mu and un are given, respectively, by the functions

$$\begin{aligned} \text{mu}_{\mathbf{A}} : \text{Pow Pow } \mathbf{A} &\rightarrow \text{Pow } \mathbf{A} \\ \mathbf{C} &\mapsto \bigcup_{B \in \mathbf{C}} B \end{aligned} \quad (62)$$

and

$$\begin{aligned} \text{un}_{\mathbf{A}} : \mathbf{A} &\rightarrow \text{Pow } \mathbf{A}, \\ x &\mapsto \{x\}. \end{aligned} \quad (63)$$

Let $\mathbf{Set}_{\text{Pow}}$ denote the Kleisli category of the monad $\langle \text{Pow}, \text{mu}, \text{un} \rangle$. The aim of this exercise is to show that the category $\mathbf{Set}_{\text{Pow}}$ and the category \mathbf{Rel} of sets and relations are isomorphic as categories.

We define functors $F : \mathbf{Set}_{\text{Pow}} \rightarrow \mathbf{Rel}$ and $G : \mathbf{Rel} \rightarrow \mathbf{Set}_{\text{Pow}}$ as follows. On objects F is the identity function, and given a function $f : \mathbf{A} \rightarrow \mathbf{Set}_{\text{Pow}} \mathbf{B}$ (in other words, a Kleisli morphism $f : \mathbf{A} \rightarrow \mathbf{Set}_{\text{Pow}} \mathbf{B}$) we let $F(f)$ be the relation

$$F(f) = \{(x, y) \in \mathbf{A} \times \mathbf{B} \mid y \in f(x)\}. \quad (64)$$

The functor G is also defined to be the identity function on objects, and given a relation $R : \mathbf{A} \rightarrow \mathbf{B}$, we let $G(R)$ be the Kleisli morphisms represented by the following function:

$$G(R) : \mathbf{A} \rightarrow \mathbf{Set}_{\text{Pow}} \mathbf{B}, x \mapsto \{y \in \mathbf{B} \mid (x, y) \in R\}. \quad (65)$$

Your tasks:

1. Prove that $F : \mathbf{Set}_{\mathbf{Pow}} \rightarrow \mathbf{Rel}$ and $G : \mathbf{Rel} \rightarrow \mathbf{Set}_{\mathbf{Pow}}$ are in fact functors.
2. Prove that F is an isomorphism of categories, with inverse G .

37.4. Algebras of a monad

In the context of Kleisli morphisms, we developed the intuition that monads can be used to encode “generalized objects” and “generalized morphisms”. In this section we will introduce a different intuition for monads: that they can be used to provide coherent way to encode “formal expressions”, together with a way to “evaluate” or “compute” such expressions.

Let us explain what we mean using an example. Given a set \mathbf{A} , say

$$\mathbf{A} = \{\text{apple}, \text{banana}, \text{carrot}\}, \quad (66)$$

let’s define a certain type of “formal expressions”, using elements of \mathbf{A} and using a “formal composition symbol”, which we choose to be “*”. Now, the “formal expressions” we consider are all finite expressions which have a form such as

$$\text{banana} * \text{banana} * \text{apple}, \quad (67)$$

or

$$\text{apple} * \text{banana} * \text{apple} * \text{apple} * \text{banana} * \text{apple}, \quad (68)$$

and so on. These expressions are “formal” (or “purely symbolic”) in the sense that, a priori, the symbol “*” does not have any *meaning* beyond simply being a symbol, a “marking”. After all, so far, \mathbf{A} is just a set, and we did not assume that it comes equipped with any sort of “multiplication operation”, for instance. In the following we will discuss a way of giving such formal expressions a “computational meaning” by specifying a way to evaluate them.

Before we come to this however, let us introduce a notation to explicitly distinguish when we are thinking about apple as an element of \mathbf{A} , or apple as a “formal expression”. For the latter situation we write “[apple]”. In other words, the square brackets indicate that [apple] is a formal expression. And we’ll say that formal expressions can be combined, using *, to larger formal expressions. So, following this convention, [apple] * [banana] is also a formal expression. And in particular, instead of (68), we’ll write

$$[\text{apple}] * [\text{banana}] * [\text{apple}] * [\text{apple}] * [\text{banana}] * [\text{apple}]. \quad (69)$$

In an expression such as (69) we’ll think of the components [apple] and [banana] as if they are “letters” (but we won’t count “*” as a letter) and we’ll think of the whole expression (69) like a “word”. For any such word, we’ll say its *length* is the number of letters it is built from. So, for instance, the word in (69) has a length of 6. In our notion of formal expression, we’ll choose to include a unique special word of length 0, which we call the “empty formal expression” and denote by “[]”.

Using square brackets we can also build “formal expressions of formal expressions” or “second-order formal expressions”. For example, given the formal expression [apple] * [banana], we can turn it into a second-order formal expression by putting brackets around it:

$$[[\text{apple}] * [\text{banana}]]. \quad (70)$$

And given another second-order formal expression, say [[banana] * [banana] * [apple]], we can “compose” it with the one in (70) like so:

$$[[\text{apple}] * [\text{banana}]] * [[\text{banana}] * [\text{banana}] * [\text{apple}]]. \quad (71)$$

The notion of length will also apply to second-order expressions. For instance, the second-order expression (71) has length 2, and is composed of one first-order expression of length 2 and one first-order expression of length 3.

This whole game can continue ad infinitum: we define third-order formal ex-

pressions to be those with three-layers of square brackets, fourth-order formal expressions have four layers of brackets, and so on. In the following, we'll probably only ever consider up to three layers.

We started our story just with the set $\mathbf{A} = \{\text{apple}, \text{banana}, \text{carrot}\}$. However, we can do the same construction – using “ $*$ ” and building formal expressions of any order – with any set. In fact, we can define a functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ which maps any set \mathbf{A} to the set $F\mathbf{A}$ whose elements are all finite first-order formal expressions built from elements of \mathbf{A} . We also include this to mean the empty formal expression “[]”. What might this functor do on the level of morphisms? For concreteness, let $\mathbf{A} = \{\text{apple}, \text{banana}, \text{carrot}\}$ and $\mathbf{B} = \{\text{jelly}, \text{beer}\}$. Given a function $f : \mathbf{A} \rightarrow \mathbf{B}$, we define

$$F(f) : F(\mathbf{A}) \rightarrow F(\mathbf{B}) \quad (72)$$

to act on (first-order) expressions like so

$$F(f)([\text{apple}] * [\text{banana}]) = [f(\text{apple})] * [f(\text{banana})]. \quad (73)$$

It turns out that this functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ can be made into a monad!

Let us explain how the unit and multiplication for this monad are defined. For any set \mathbf{A} , the component of the unit at \mathbf{A} is

$$\begin{aligned} \mathbf{un}_{\mathbf{A}} : \mathbf{A} &\rightarrow F\mathbf{A}, \\ x &\mapsto [x]. \end{aligned} \quad (74)$$

The multiplication is a bit more of a mouthful. Its component at \mathbf{A} ,

$$\mathbf{mu}_{\mathbf{A}} : (F ; F)\mathbf{A} \rightarrow F\mathbf{A}, \quad (75)$$

is the function which takes a second-order formal expression

$$[[x_{11}] * [x_{12}] * \dots * [x_{1k_1}]] * \dots * [[x_{n1}] * [x_{n2}] * \dots * [x_{nk_n}]] \quad (76)$$

and “collapses” it to the first-order expression

$$[x_{11}] * [x_{12}] * \dots * [x_{1k_1}] * \dots * [x_{n1}] * [x_{n2}] * \dots * [x_{nk_n}]. \quad (77)$$

In other words, this operation simply “removes the outer brackets” from a second-order formal expression.

Graded exercise K.4 (ListMonad)

Let $F : \mathbf{Set} \rightarrow \mathbf{Set}$ be the functor above that sends any set \mathbf{A} to the set of first-order formal expressions of the form

$$[x_1] * [x_2] * \dots * [x_n] \quad x_i \in \mathbf{A}, n \in \mathbb{Z}_{\geq 0}, \quad (78)$$

and let $\mathbf{mu}_{\mathbf{A}}$ and $\mathbf{un}_{\mathbf{A}}$ be defined as above.

Show that:

1. the components $\mathbf{mu}_{\mathbf{A}}$ define a natural transformation $\mathbf{mu} : F ; F \Rightarrow F$;
2. the components $\mathbf{un}_{\mathbf{A}}$ define a natural transformation $\mathbf{un} : \mathbf{Id}_{\mathbf{Set}} \rightarrow F$;
3. \mathbf{mu} and \mathbf{un} satisfy the conditions for $\langle F, \mathbf{mu}, \mathbf{un} \rangle$ to be a monad.

Now let's finally talk about giving formal expressions a computational meaning: a way to *evaluate* them. The way we will do this is to define a notion of “evaluation map” $a : F\mathbf{A} \rightarrow \mathbf{A}$ which we will interpret as a way of specifying how any formal expression – an element of $F\mathbf{A}$ – should be evaluated (or: computed) to an element of \mathbf{A} . We will require such evaluation maps to additionally satisfy two

coherence conditions, and the resulting mathematical structure will be what is called an *algebra* of the monad \mathbf{F} .

Definition 37.8 (Algebra of a monad)

Let $\langle \mathbf{M}, \mathbf{mu}, \mathbf{un} \rangle$ be a monad on a category \mathbf{C} . An algebra of \mathbf{M} (also called an \mathbf{M} -algebra) is specified by:

Constituents

1. an object X of \mathbf{C} ;
2. a morphism $a : M(X) \rightarrow X$ of \mathbf{C} .

Conditions

1. *Composition*: the following diagram commutes:

$$\begin{array}{ccc} (M ; M)(X) & \xrightarrow{Ma} & M(X) \\ \mathbf{mu}_X \downarrow & & \downarrow a \\ M(X) & \xrightarrow{a} & X \end{array} \quad (79)$$

2. *Unit*: the following diagram commutes:

$$\begin{array}{ccc} X & \xrightarrow{\mathbf{un}_X} & M(X) \\ & \searrow \mathbf{id} & \downarrow a \\ & & X \end{array} \quad (80)$$

Definition 37.9 (\mathbf{M} -algebra morphism)

Let $\langle \mathbf{M}, \mathbf{un}, \mathbf{mu} \rangle$ be a monad on a category \mathbf{C} , and let $\langle X_1, a_1 \rangle$ and $\langle X_2, a_2 \rangle$ be algebras of \mathbf{M} . A morphism $\langle X_1, a_1 \rangle \rightarrow \langle X_2, a_2 \rangle$ of \mathbf{M} -algebras is specified by:

Constituents

1. A morphism $f : X_1 \rightarrow X_2$ in \mathbf{C} .

Conditions

1. The following diagram commutes:

$$\begin{array}{ccc} M(X_1) & \xrightarrow{Mf} & M(X_2) \\ a_1 \downarrow & & \downarrow a_2 \\ X_1 & \xrightarrow{f} & X_2 \end{array} \quad (81)$$

Definition 37.10 (Category of \mathbf{M} -algebras)

Let $\langle \mathbf{M}, \mathbf{un}, \mathbf{mu} \rangle$ be a monad on a category \mathbf{C} . The *category of \mathbf{M} -algebras* $\mathbf{C}^{\mathbf{M}}$ of the monad \mathbf{M} is specified by:

1. *Objects*: \mathbf{M} -algebras;
2. *Morphisms*: \mathbf{M} -algebra morphisms;
3. *Identities*: given an \mathbf{M} -algebra $\langle X, a \rangle$, its identity morphism is \mathbf{id}_X ;
4. *Composition*: is induced by the composition of morphisms in \mathbf{C} .

Proposition 37.11. There is an \mathbf{M} -algebra structure on X if and only if there is a monoidal structure on X . The correspondence is as follows:

Something incorrect or unclear or missing? Report issues on GitHub by clicking [here](#).

- ▷ The neutral element of $\textcolor{red}{X}$ corresponds to $\textcolor{blue}{a}([])$.
- ▷ The composition in $\textcolor{red}{X}$ corresponds to the action of $\textcolor{blue}{a}$:

$$[x_1] * [x_2] \xrightarrow{\textcolor{blue}{a}} x_1 \circ x_2. \quad (82)$$

- ▷ The associativity and unitality of monoid composition are encoded by the $\textcolor{violet}{M}$ -algebra axioms.

$$\textcolor{red}{X} = \{0, 1, 2, 3, \dots\} \quad (83)$$

$$\begin{aligned} \textcolor{blue}{a} : \textcolor{violet}{M}\textcolor{red}{X} &\rightarrow \textcolor{red}{X} \\ [] &\mapsto 0 \\ [x] &\mapsto x \\ [x] * [y] &\mapsto x + y \end{aligned} \quad (84)$$

Graded exercise K.5 (HwkFreeAlgebras)

Let $\langle \textcolor{violet}{M}, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} , and let $\textcolor{red}{X}$ be an object of \mathbf{C} . Your task is to prove that the object $\textcolor{violet}{M}\textcolor{red}{X}$, together with the morphism

$$\text{mu}_{\textcolor{red}{X}} : \textcolor{violet}{M}\textcolor{violet}{M}\textcolor{red}{X} \rightarrow \textcolor{violet}{M}\textcolor{red}{X} \quad (85)$$

defines an algebra for the monad $\textcolor{violet}{M}$. Hint: use the axioms/conditions in the definition of a monad.

37.5. Monads from Adjunctions

Solutions to selected exercises



Example exams**555**

When one thinks about Switzerland and the alps, one usually thinks about snow. Starting from 1,500 m height, most areas are covered in snow during winter (from December to March). Some areas (above 3,000 m), are *always* covered in snow.

Nomenclature

symbol	meaning	defined in
Generic symbols		
<i>Booleans</i>		
\top	True	
\perp	False	
\wedge	Boolean and	
\vee	Boolean or	
\Rightarrow	Implies	
\Leftrightarrow	Equivalence	
<i>Categories</i>		
<i>Adjunctions</i>		
L	Left adjunct functor	→Def. 24.18 on p.334
R	Right adjunct functor	→Def. 24.18 on p.334
$L \dashv R$	L and R are adjoint functors.	→Def. 24.18 on p.334
τ	Adjunction isomorphism	→Def. 24.18 on p.334
un	Unit	→Def. 24.20 on p.335
co	Co-unit	→Def. 24.20 on p.335
<i>Arrows</i>		
\rightarrow	Set arrow	
\rightarrowtail	Morphism arrow	
\rightarrowtail	Morphism arrow	
\leftarrowtail	Morphism arrow	
\rightarrowtail	Functors arrow	
\Rightarrowtail	Natural transformation arrow	
$\cong\rightarrowtail$		
\rightarrowtail	profunctor arrow	
<i>Companion/conjoints</i>		
<i>Composition</i>		
$f ; g$	Composition of morphisms	
\circ	Composition for functors	
\circ	Composition for natural transformations	
$F ; G$	Composition of functors	
Ob_A	Objects of the category A .	→Def. 13.9 on p.192
Mor_A	Collection of all morphisms of the category A .	→Def. 13.9 on p.192
id_X	Identity morphism for the object X	→Def. 13.9 on p.192
$\text{Hom}_A(X; Y)$	Hom-set between X and Y .	→Def. 13.9 on p.192
<i>Constructors</i>		
Free	free construction	→Def. 13.18 on p.198
$\text{Arr } A$	Arrow construction on category A .	
$\text{Tw } A$	Twisted arrow construction on category A .	→Def. 16.10 on p.240
<i>Generic names</i>		
f, g, h, i	Generic morphisms	→Def. 13.9 on p.192
F, G, H, K	Generic functors	→Def. 19.2 on p.280
id_A	Identity functor for category A	

symbol	meaning	defined in
$\alpha, \beta, \gamma, \delta$	Generic natural transformations	→Def. 23.1 on p.310
id_F	Identity natural transformation for functor F	
<i>Monoidal categories</i>		
\otimes	Stacking category objects semigroup operation	
$\otimes_{\mathbf{A}}$	Stacking category morphisms semigroup operation	
\otimes_A	Monoidal product for category A .	→Def. 25.26 on p.359
1	Identity object for monoidal operation	→Def. 25.26 on p.359
lu	Left unitor	→Def. 25.26 on p.359
ru	Right unitor	→Def. 25.26 on p.359
as	Associator	→Def. 25.26 on p.359
br	Braiding	→Def. 25.26 on p.359
μ	Isomorphism for strong monoidal functor	→Def. 25.31 on p.364
u	Natural isomorphism for strong monoidal functor	→Def. 25.31 on p.364
ev	evaluation map for dualizable objects	
$coev$	coevaluation map for dualizable objects	
<i>Well-known categories</i>		
\mathbf{DP}	Category of design problems	→Def. 15.12 on p.226
\mathbf{Feas}	Synonym of \mathbf{DP}	→Def. 15.12 on p.226
\mathbf{UDP}		
\mathbf{DPI}	Semi-category \mathbf{DPI}	→Def. 29.1 on p.414
\mathbf{Bool}	Posets/category of Booleans	
\mathbf{Cat}	Category of small categories	→Def. 22.3 on p.301
$\mathbf{FinVect}$	Category of finite-dimensional vector spaces	
$\mathbf{Vect}_{\mathbb{R}}$	Category of real vector spaces	→Def. 25.13 on p.352
\mathbf{Rel}	Category of sets and relations	→Def. 14.1 on p.203
\mathbf{FinSet}	Category of finite sets and functions	→Example 20.2 on p.290
\mathbf{Set}	Category of sets and functions	→Def. 13.11 on p.193
\mathbf{InjSet}	Category of sets and injective functions	→Example 20.5 on p.292
\mathbf{Prof}	Category of profunctors	
\mathbf{Pos}	Category of posets and monotone maps	→Def. 14.8 on p.206
\mathbf{Lat}	Category of lattices and lattice homomorphisms	→Def. 31.18 on p.448
$\mathbf{BoundedLat}$	Category of lattices and lattice homomorphisms	→Def. 31.19 on p.448
\mathbf{Grph}	Category of directed graphs	→Def. 14.14 on p.208
\mathbf{Eff}	The effects category	→Def. 25.13 on p.352
\mathbf{EndSet}	Category of endofunctions	
\mathbf{Set}_*	Category of pointed sets	
$\mathbf{EquivRel}$	Category of equivalence relations	
\mathbf{Euc}_*	Category of pointed Euclidean spaces	→Def. 12.2 on p.180
\mathbf{Grph}	Category of graphs	→Def. 18.22 on p.271
\mathbf{LTI}		
\mathbf{Beh}		
\mathbf{LBeh}		
$\langle \mathbf{Rel} \rangle$	category of tuple-sets and relations	→Def. 25.35 on p.367
$\langle \mathbf{Set} \rangle$		→Def. 18.3 on p.258
$\langle \mathbf{Pos} \rangle$	category of tuple-posets and monotone maps	→Def. 25.33 on p.367
$\langle \mathbf{DP} \rangle$	Category of lists of DPs	→Def. 25.37 on p.368
<i>Operations</i>		
\times	Product in a category	
$+$	Product in a category	
$+$	Disjoint union of posets	
$+$	Co-product in a category	

symbol	meaning	defined in
\times	Product in a category	
$+$	Product in a category	
	<i>Traced monoidal categories</i>	
Tr	Trace operator	→Def. 27.1 on p.384
Fb	Feedback operator	→Def. 27.1 on p.384
Conw	Conway operator	
	<i>DP</i>	
	<i>Computational representation</i>	
k		
K		
h		
H		
	<i>Formalization</i>	
f	A generic functionality in \mathbf{F} .	→Def. 29.1 on p.414
r	A generic cost in \mathbf{R} .	
i	A generic implementation in in \mathbf{I} .	→Def. 29.1 on p.414
\mathbf{F}	Functionality space	
\mathbf{R}	Requirements space	→Def. 29.1 on p.414
\mathbf{I}	Implementation space	→Def. 29.1 on p.414
$\text{prov} : \mathbf{I} \rightarrow \mathbf{F}$	functionality of an implementation	→Def. 29.1 on p.414
$\text{req} : \mathbf{I} \rightarrow \mathbf{R}$	requirements of an implementation	→Def. 29.1 on p.414
	<i>Queries in DP</i>	
Feasibility		→Section 29.3 on p.424
FeasibleImp		→Section 29.3 on p.423
FixFunMinRes		→Section 29.3 on p.423
FixFunFeasRes		
FixResMaxFun		→Section 29.3 on p.423
FixResFeasFun		
	<i>DP</i>	
$\mathbf{d}, \mathbf{e}, \mathbf{g}$	Generic design problems as profunctors	
vdc	Van Der Corput sequence	
	<i>Groups</i>	
id	identity for group	→Def. 9.34 on p.144
\mathbf{G}, \mathbf{H}	Generic group names	→Def. 9.34 on p.144
\mathbf{G}, \mathbf{H}	Underlying set of groups.	→Def. 9.16 on p.141
inv	Group inverse	→Def. 9.34 on p.144
x, y, z	Generic group elements	
\circ	Group composition operation	→Def. 9.34 on p.144
	<i>Linear Algebra</i>	
$\mathbf{0}$	Zero matrix	
$\mathbf{1}$	Identity matrix	
$\det A$	matrix determinant (as functor)	
\mathcal{P}^+	Positive-definite matrices	
	<i>Monoids</i>	
id	identity for monoid	→Def. 9.16 on p.141
$\mathbf{M}, \mathbf{N}, \dots$	Generic monoid names	→Def. 9.16 on p.141
$\mathbf{M}, \mathbf{N}, \dots$	Underlying sets for the monoids.	→Def. 9.16 on p.141
x, y, z	Generic monoid elements	→Def. 9.16 on p.141
m, n	Generic monoid elements	→Def. 9.16 on p.141

symbol	meaning	defined in
\circ	Semigroup/monoid/group operation	→Def. 9.16 on p.141
<i>Posets</i>		
<i>Attributes</i>		
$\text{width}(\mathbf{P})$	Width of the poset \mathbf{P} .	→Def. 5.19 on p.99
$\text{height}(\mathbf{P})$	Height of the poset \mathbf{P} .	→Def. 5.20 on p.99
<i>Poset constructors</i>		
\times	Product of posets	
$\text{Arr } \mathbf{P}$	Arrow construction on poset \mathbf{P} .	→Def. 6.10 on p.108
$\text{Tw } \mathbf{P}$	Twisted arrow construction on poset \mathbf{P} .	→Def. 6.9 on p.106
\mathbb{E}	expectedvalue	
$\mathbf{1}$	singleton poset	
\perp	global dualizing object	
\mathbf{P}	a polycategory	
\mathbf{Q}	a polycategory	
$\mathbf{\Gamma}$	a polycategory	
$\mathbf{\Gamma}$	a polycategory	
Δ	a polycategory	
Σ	a polycategory	
<i>Constructors</i>		
$\text{Anti } \mathbf{P}$	The set of antichains of a poset \mathbf{P} .	→Def. 5.14 on p.97
$\text{Anti}_f \mathbf{P}$	The set of finite antichains of a poset \mathbf{P} .	
$\text{L}\mathbf{P}$	$= \langle \text{UpSets } \mathbf{P}, \subseteq \rangle$ Poset of upward-closed lowerset of \mathbf{P} ordered by \subseteq .	→Def. 8.11 on p.127
$\text{U}\mathbf{P}$	$= \langle \overline{\text{UpSets}} \mathbf{P}, \supseteq \rangle$ Poset of downward-closed uppersets of \mathbf{P} ordered by \supseteq .	
$\overline{\text{U}}_f \mathbf{P}$	$\subseteq \text{U}\mathbf{P}$ Poset of finitely-supported upper sets of a poset \mathbf{P} .	→Def. 36.13 on p.504
<i>Domain theory</i>		
lfp	Least fixed point	
CPO	Complete partial order	→Def. 36.2 on p.500
DCPO	Directed-complete partial order	→Def. 36.2 on p.500
<i>Generic poset names</i>		
$\mathbf{P}, \mathbf{Q}, \mathbf{R}$	Generic posets	→Def. 5.3 on p.91
$\mathbf{P}, \mathbf{Q}, \mathbf{R}$	Underlying set for the posets	→Def. 5.3 on p.91
$\preceq_{\mathbf{P}}$	Order on poset \mathbf{P}	
$\text{Pow } \mathbf{A}$	Power poset of \mathbf{A} .	→Def. 5.12 on p.96
<i>Monoidal posets</i>		
\otimes	Monoidal poset operation	
<i>Operations on elements</i>		
$x \vee y$	Join of two elements x, y	→Def. 31.8 on p.444
$x \wedge y$	Meet of two elements x, y	→Def. 31.8 on p.444
<i>Operations on sets</i>		
$\text{Min}_{\leq_p} \mathbf{S}$	Minimal elements of the subset \mathbf{S} .	→Def. 8.1 on p.124
$\text{Max}_{\leq_p} \mathbf{S}$	Maximal elements of the subset \mathbf{S} .	→Def. 8.2 on p.124
$\text{Inf}_{\leq_p} \mathbf{S}$	Infimum element of the subset \mathbf{S} .	→Def. 8.7 on p.125
$\text{Sup}_{\leq_p} \mathbf{S}$	Supremum element of the subset \mathbf{S} .	→Def. 8.3 on p.125
$\uparrow \mathbf{S}$	Upper closure of \mathbf{S} .	→Def. 8.12 on p.128
$\downarrow \mathbf{S}$	Lower closure of \mathbf{S} .	→Def. 8.16 on p.128

symbol	meaning	defined in
<i>Symbols</i>		
\top_P	Top of poset P	→ Def. 8.9 on p.126
\perp_P	Bottom of poset P	→ Def. 8.9 on p.126
<i>Relations</i>		
-		
<i>Rings</i>		
1	identity for ring	→ Def. 9.58 on p.149
0	identity for ring	→ Def. 9.58 on p.149
R	Generic ring names	
R, S	Underlying set of rings.	
S	Underlying set of S .	
K	Generic ring names	
R, S	Underlying set of rings.	
L	Underlying set of L .	
<i>Semigroups</i>		
<i>Semigroups</i>		
S, T, U	Generic semigroup names.	
S, T, U	Generic names for underlying carrier set of a semigroup.	
\log	Matrix logarithm	
\exp	Matrix exponential	
x, y, z	Generic semigroup elements	
F, G	Generic semigroup morphisms.	
<i>Sets</i>		
<i>Constructors</i>		
$\text{Pow } A$	Powerset of A	
$\text{Pow } A$	Power set of A .	→ Def. 3.6 on p.37
<i>Generic sets and elements</i>		
$ x $	norm of vector x	
codf	Codomain of function f	→ Section 3.4 on p.42
domf	Domain of function f	→ Section 3.4 on p.42
\cap	Set intersection	
\cup	Set union	
$x \in A$	The element x belongs to the set A .	
$x \notin A$	The element x does not belong to the set A .	
$A \subseteq B$	The set A is a subset of B .	
C	subset	
$A \supseteq B$	The set A is a superset of B .	
<i>Well-known sets</i>		
\emptyset	Empty set	
\mathbb{C}	Complex numbers	
\mathbb{R}	Real numbers	
\mathbb{N}	Natural numbers: $0, 1, 2, \dots$	
\mathbb{Z}	Integers: $0, 1, -1, 2, -2, \dots$	
\mathbb{Q}	Rational numbers	
$\mathbb{R}_{>0}$	Positive real numbers	
$\mathbb{R}_{\geq 0}$	Non-negative real numbers	
$\overline{\mathbb{R}_{\geq 0}}$	Completion of non-negative real numbers.	
1	Singleton set, containing the element \bullet .	

symbol	meaning	defined in
$\mathbb{R}^{[m]}$	A copy of \mathbb{R} with units of meters	
<i>Operations</i>		
$A \times B$	Cartesian product of two sets.	→Def. 3.7 on p.39
$f \times g$	Product of two functions	
$f + g$	Direct sum of two functions	
$A + B$	Disjoint union of two sets.	
$\langle 1, a \rangle, \langle 1, b \rangle$	Decorated elements of disjoint union	
in_1, in_2	Injections into $A + B$.	
<i>Well-known functions</i>		
id_A	Identity map on A	
ceil	ceiling function	→Example 7.17 on p.121
floor	floor function	→Example 7.17 on p.121
rtntt	Round to nearest, ties to even	→Example 7.17 on p.121
<i>Tuples</i>		
$\langle \rangle$	zero-size tuple	
$[]$	zero-size list	

Specific use in Volume 1

Part D - Algebra

Chapter 10 - Morphisms

ASCII example

emchar	includes spaces
<i>Morse code</i>	
\cdot	Morse dot
$-$	Morse dash
s_1	Silence between dots and dashes
s_3	Silence between letters
s_7	Silence between words
	Beep of ℓ
	Beep of 3ℓ
	Silence of ℓ
	Silence of 3ℓ
	Silence of 7ℓ

Chapter 11 - Actions

Moo	Category of Moore machines	→Def. 18.6 on p.261
Mor	Category of More machines	→Def. 18.12 on p.268
P_{effort}	effort	
P_{track}	tracking	
<i>Matrix groups</i>		
$O(n, \mathbb{R})$	Orthogonal group	→Def. 9.49 on p.146
$SO(n, \mathbb{R})$	Special orthogonal group	→Def. 9.51 on p.146
$GL(n, \mathbb{R})$	General linear group	→Def. 9.48 on p.146
$SL(n, \mathbb{R})$	Special linear group	→Def. 9.50 on p.146
$E(n, \mathbb{R})$	Euclidean group	→Def. 11.14 on p.171
$SE(n, \mathbb{R}^n)$	Special euclidean group	→Def. 11.15 on p.171

Part E - Categories

Curr	Currency category	→Def. 15.3 on p.217
Temp	Temperature category	→Exercise 36 on p.218

symbol	meaning	defined in
Berg	The category of Swiss mountains	→Def. 15.2 on p.214
BergAma		→Section 20.4 on p.293
BergLazy		→Section 20.4 on p.293
<i>Chapter 18 - (Semi)Category actions</i>		
<i>Procedures</i>		
ProcTime	Procedures with execution time	→Def. 15.13 on p.230
size	Size of datatype	→Def. 15.13 on p.230
ProcSize	Procedures with sized sets	→Def. 15.15 on p.231
σ		→Def. 15.15 on p.231
ProcSizeTime	Procedures with size-dependent durations	→Def. 15.16 on p.231
dur		→Def. 15.16 on p.231
<i>Part F - Functors</i>		
<i>Chapter 19 - Translation</i>		
Plans		
<i>Chapter 20 - Specialization</i>		
<i>drawings</i>		
Draw	Category of drawings	→Def. 20.4 on p.291
<i>Part I - Co-Design</i>		
<i>Construction</i>		
diag_p	Diagonal function	→Def. 33.1 on p.460
codiag_p	Co-diagonal function	→Def. 33.2 on p.460
<i>Part K - Compositional computation</i>		
<i>Chapter 37 - Monads</i>		
M, N	Generic monads.	→Def. 37.2 on p.531
un	Monad unit	→Def. 37.2 on p.531
mu	Monad identity	→Def. 37.2 on p.531
fish		→Def. 37.4 on p.534
lift		→Def. 37.4 on p.534
join		→Def. 37.4 on p.534
bind		→Def. 37.4 on p.534
fmap		→Def. 37.4 on p.534
return		→Def. 37.4 on p.534
U	upper-set endofunctor	
U	upper-set monad	
L	lower-set endofunctor	
L	lower-set monad	
To categorize		
:=	“defined as”	



Example exams

This chapter contains some examples of the style of the ETH class exam. Both sample exams are thought to last 90 minutes and to be open book (all notes allowed).

1 Exam 1 556

1. Exam 1

Example 1: Uncertain Machines

Consider the category of Moore machines acting on signals as described in the course. Assuming the input/output sets are also ordered sets (posets), construct the Kleisli category corresponding to the interval monad \mathcal{U} . That is, the signals are closed intervals of values. Call this category **UMoore**.

1. (30%) Is **UMoore** a monoidal category?
2. (20%) Is **UMoore** a traced monoidal category?
3. (50%) Would the answers be the same if asked about the More category?

Example 2: Machines with resources consumption

Consider the category of Moore machines. We want to be more precise about resource consumption and want to define an extension of Moore machines in which each machine also has associated a certain time $T_1 \geq 0$ to run the dynamic function `dyn` and a certain time $T_2 \geq 0$ for running the readout function `ro`. We call these resource-Moore machines (**RMoore**).

1. (30%) Formalize the **RMoore** category giving formulas for identities, compositions, and proof of associativity.
2. (70%) Suppose that you want to design the software for a robot. You are given the wiring diagram of the architecture, in which you have to plug in specific **RMoore** machines to implement the algorithmic functionality. Assume that the wiring diagram does not contain any loop - only series and parallel composition, and that there is only one input (robot observations) and one output (robot commands). For each hole in the diagram, you are given a set of 1 or more **RMoore** machines that can implement the functionality. Assume that the computer on which to run everything has $N \geq 1$ processors. Think of each processor as a “lane” in which the operations of each machine are cars that must run sequentially.

Formalize the design problem in the category **DPI** that corresponds to choosing the best combination of machines and the best assignment to processors. Include as a resource the number of processors and as functionality the throughput of the system (how many commands are generated per second).

Bibliography

- [1] Sergei L Bezrukov and Ian T Roberts. «On Antichains in Product Posets». In: () .
- [2] P. Cuffe and A. Keane. «Visualizing the Electrical Structure of Power Systems». In: *IEEE Systems Journal* 11.3 (2017), pp. 1810–1821. DOI: [10.1109/JYST.2015.2427994](https://doi.org/10.1109/JYST.2015.2427994).
- [3] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002. ISBN: 9780521784511. DOI: [10.1017/cbo9780511809088](https://doi.org/10.1017/cbo9780511809088).
- [4] Magnus Egerstedt. «Motion Description Languages for Multi-Modal Control in Robotics». In: *Control Problems in Robotics*. Springer Science + Business Media, 2003, pp. 75–89. DOI: [10.1007/3-540-36224-x_5](https://doi.org/10.1007/3-540-36224-x_5).
- [5] Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press, 2019.
- [6] G. Gierz et al. *Continuous Lattices and Domains*. Cambridge University Press, 2003. DOI: [10.1017/cbo9780511542725](https://doi.org/10.1017/cbo9780511542725).
- [7] Petr A Golovach et al. «An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets». In: *Algorithmica* 72.3 (June 2015), pp. 836–859.
- [8] Michael Grant and Stephen Boyd. «Graph implementations for nonsmooth convex programs». In: *Recent Advances in Learning and Control*. Ed. by V. Blondel, S. Boyd, and H. Kimura. Lecture Notes in Control and Information Sciences. http://stanford.edu/~boyd/graph_dcp.html. Springer-Verlag Limited, 2008, pp. 95–110.
- [9] G. Grisetti, C. Stachniss, and W. Burgard. «Improved Techniques for Grid Mapping With Rao Blackwellized Particle Filters». In: *IEEE Transactions on Robotics* 23.1 (Feb. 2007), pp. 34–46. ISSN: 1552-3098. DOI: [10.1109/TRO.2006.889486](https://doi.org/10.1109/TRO.2006.889486).
- [10] Masahito Hasegawa. «The Uniformity Principle on Traced Monoidal Categories». In: *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002*. Ed. by Richard Blute and Peter Selinger. Vol. 69. Electronic Notes in Theoretical Computer Science. Elsevier, 2002, pp. 137–155. DOI: [10.1016/S1571-0661\(04\)80563-2](https://doi.org/10.1016/S1571-0661(04)80563-2). URL: [https://doi.org/10.1016/S1571-0661\(04\)80563-2](https://doi.org/10.1016/S1571-0661(04)80563-2).
- [11] IEEE Task P754. *IEEE 754-2008, Standard for Floating-Point Arithmetic*. Aug. 2008, p. 58. ISBN: 0-7381-5753-8 (paper), 0-7381-5752-X (electronic). DOI: <https://doi.org/10.1109/IEEEESTD.2008.4610935>. URL: http://en.wikipedia.org/wiki/IEEE_754-2008 ; <http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>.

- [12] ISO. *ISO 80000-2: Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology*. Dec. 2009, p. 40. URL: <https://www.iso.org/standard/31887.html>.
- [13] S. M. LaValle. *Sensing and Filtering: A Fresh Perspective Based on Preimages and Information Spaces*. Foundations and Trends in Robotics Series. Delft, The Netherlands: Now Publishers, 2012. DOI: [10.1561/2300000004](https://doi.org/10.1561/2300000004).
- [14] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. 1st ed. Available for download on the authors' website <http://leeseshia.org/>. 2010. ISBN: 978-0-557-70857-4.
- [15] Nancy. Leveson. *Engineering a safer world systems thinking applied to safety*. eng. Engineering systems. Cambridge, Mass: MIT Press.
- [16] Alex Locher, Michal Perdoch, and Luc Van Gool. «Progressive Prioritized Multi-view Stereo». In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).
- [17] A Lodi, S Martello, and M Monaci. «Two-dimensional packing problems: A survey». In: *European Journal of Operational Research* 141.2 (2002), pp. 241–252.
- [18] Alan MacCormack, Carliss Baldwin, and John Rusnak. «Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis». In: *Research Policy* 41.8 (Oct. 2012), pp. 1309–1324. ISSN: 00487333. DOI: [10.1016/j.respol.2012.04.011](https://doi.org/10.1016/j.respol.2012.04.011). URL: <http://dx.doi.org/10.1016/j.respol.2012.04.011>.
- [19] E.G. Manes and M.A. Arbib. *Algebraic approaches to program semantics*. Springer-Verlag, 1986. ISBN: 9780387963242. DOI: [10.1007/978-1-4612-4962-7](https://doi.org/10.1007/978-1-4612-4962-7).
- [20] S.E. Morison. *Admiral of the Ocean Sea: A Life of Christopher Columbus*. Read Books, 2007. ISBN: 9781406750270. URL: <https://books.google.ch/books?id=T5x5xjsJt1wC>.
- [21] Luigi Nardi et al. «Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM». In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2015.
- [22] Jason M. O’Kane and Steven M. LaValle. «On comparing the power of robots». In: *International Journal of Robotics Research* 27.1 (Jan. 2008), pp. 5–23.
- [23] E. Riehl. *Category Theory in Context*. Aurora: Dover Modern Math Originals. Dover Publications, 2017. ISBN: 9780486820804. URL: <https://books.google.ch/books?id=6B9MDgAAQBAJ>.
- [24] S. Roman. *Lattices and Ordered Sets*. Springer, 2008. ISBN: 9780387789019. DOI: [10.1007/978-0-387-78901-9](https://doi.org/10.1007/978-0-387-78901-9).
- [25] Herbert A. Simon. *The Sciences of the Artificial* (3rd Ed.) Cambridge, MA, USA: MIT Press, 1996. ISBN: 0262691914.
- [26] Stefano Soatto. «Steps Towards a Theory of Visual Information: Active Perception, Signal-to-Symbol Conversion and the Interplay Between Sensing and Control». In: *CoRR* abs/1110.2053 (2011). URL: <http://arxiv.org/abs/1110.2053>.
- [27] David I Spivak. «Categorical databases». In: *Presented at Kensho* (2019).
- [28] David I Spivak and Robert E Kent. «Ologs: a categorical framework for knowledge representation». In: *PloS one* 7.1 (2012), e24274.
- [29] Sundararajan Sriram and Shuvra S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. 1st. New York, NY, USA: Marcel Dekker, Inc., 2000. ISBN: 0824793188. DOI: [10.1201/9781420048025](https://doi.org/10.1201/9781420048025).
- [30] Maria Svorenova et al. *Resource-Performance Trade-off Analysis for Mobile Robots*. 2016. eprint: [arXiv:1609.04888](https://arxiv.org/abs/1609.04888).
- [31] Philip Wadler. «Linear Types Can Change the World!» In: *PROGRAMMING CONCEPTS AND METHODS*. North, 1990.
- [32] M. Zeeshan Zia et al. «Comparative Design Space Exploration of Dense and Semi-Dense SLAM». In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

Index

- adjoint equivalence, **336**
- adjunction, **334**, 334, **335**, 335–338, 340
- algebra of a monad, **539**
- antichain in a poset, **97**, 97–99, 117, 129, 421–423, 499, 504, 506, 507, 510, 514, 520, 521
- antisymmetric relation, **73**, 91
- antitone Galois connection, **332**
- antitone map, **114**, 114, 128, 332
- arrow category, **239**
- arrow poset of intervals, **108**
- associative law, 138–141, 163, 175, 191
- associative stacking semicategory, 354, 366–368, 374
- associativity, 137, 138, 224, 226, 256, 257, 259, 261, 346, 368, 429, 464, 465, 527, 534
- asymmetric relation, **73**
- automorphism, 168

- bijective function, **44**, 44, **45**, 46, 49, 80, 145, 153, 158, 470
- Bool** (poset/category of booleans), **93**
- Bool** set of booleans, **35**
- bottom element of a poset \perp , 454

- bounded lattice, **444**, 444, **445**, 447, 448, 451, 454, 455, 491, 492, 496, 500
- bounded lattice homomorphism, **447**
- BoundedLat** (category of bounded lattices and lattice homomorphisms), **448**
- braided monoidal category, 379

- cardinality of a set **card**, **31**
- cartesian product, **39**, 39–41, 51, 65, 102, 235, 256, 257, 346, 360, 484, 486
- Cat** (category of small categories), **301**
- categorical coproduct, 41
- categorical product, 41, 301
- category, **192**, 272, 346, 347, 350, 351, 353
- category of currencies, **217**
- category of mountain trips, **214**
- chain in a poset, **97**, 97–99, 103, 501, 517, 520
- co-design problem with implementation, **425**
- codiagonal function, **460**
- commutative diagram, **195**, 239, 240
- commutative group, **144**

- commutative magma, 136
 commutative semigroup, 137, 144
 companion construction, 460, 462, 463
 complete lattice, 454
 composition (of functors), 300, 301
 composition of functions, 45
 composition of relations, 66
 concatenation of tuples 3, 258
 conjoint construction, 462, 463
 coslice category, 242
- design complexity, 521
 design problem (with implementation), 414, 431
 design problem (without implementation), 115, 116–119, 223,
 224, 226, 226, 356, 368, 380, 387, 407, 413, 417, 420,
 421, 425, 426, 430, 431, 435, 450–455, 462–464,
 480, 504, 505, 511, 513–515, 518, 521, 522, 556
- diagonal function, 460
 directed complete partial order, 500
 directed graph, 180
 directed multigraph, 180
 directed subset, 500, 500
 disjoint union (of categories) +, 237, 237
 disjoint union (of posets) +, 104
 disjoint union (of sets) +, 40, 41, 51, 104
 distributive lattice, 455
 downward-closed, 129
DP (category of design problems), 226
DPI (semicategory of design problems with implementation),
 429
 $\langle \text{DP} \rangle$ (category of tuple-posets and design problems), 368
Draw category of drawings, 291
- endofunctor, 281, 281, 283
 endomorphism, 168, 396, 399
 endorelation, 73, 73, 74, 91
 equivalence of categories, 335
 equivalence of LTI systems, 271
 equivalence relation, 76, 76, 120
 everywhere-defined, 71, 71, 72
- finite design problem, 504
 finitely supported upper set, 504
FinSet (category of finite sets and functions), 290
 fixed point, 501
 forgetful functor, 435
 free semicategory on a graph, 198
 free semigroup, 163
 function, 44
 function isomorphism, 46
 functor, 280, 280–284, 286, 287, 289, 299–301, 303, 305, 306,
 309, 310, 313, 319, 321, 322, 334, 335, 337–340,
 346, 350, 360, 371, 462, 479, 481, 483, 484, 495,
 531, 532, 538
 functorial stacking semicategory, 351, 353, 368, 369, 371, 371,
 373
- graph homomorphism, 182, 182, 183, 208, 420
 greatest lower bound, 126
 group, 135, 136, 144, 144–146, 160–162, 165, 168, 171, 186,
 204
- group morphism, 159, 159, 160, 168, 204
 groups of symmetries, 144
Grp (category of groups and morphisms), 204
Grph (category of graphs and graphs homomorphisms),
 208
- Hasse diagram, 92, 93
 hom-sets, 189, 195, 449, 451, 454
- identity design problem, 225, 462
 identity function, 46, 152, 160, 192, 204, 249
 identity functors, 281, 282, 301, 321, 322, 360, 483, 484
 identity matrix, 147, 205
 identity morphism, 192, 192, 193, 195, 198, 202–204, 209, 214,
 216, 217, 222, 226, 236, 244, 249, 282, 284, 290,
 292, 293, 301, 304, 305, 322, 339, 361, 374, 448,
 470, 482, 484, 504, 539
- identity morphism of groups, 160
 identity morphism of monoids, 157
 identity morphism of semigroups, 152
 infimum, 126
 injective function, 44, 45, 80, 202, 249, 292
 injective relation, 71, 71, 72
InjSet (category of sets and injective functions), 292
 intersection of sets \cap , 36
 interval, 106
 inverse, 470
 irreflexive relation, 73
 isomorphism, 470
 isomorphism (of categories), 335
 isomorphism (of groups), 160
 isomorphism (of monoids), 157, 158
 isomorphism (of semigroups), 152, 153, 154
 isomorphism (of sets), 46
 isomorphism in a category, 470
- join, 125, 125, 444, 454, 491, 492, 496
- Kleisli category, 535, 556
 Kleisli composition, 535, 535
 Kleisli morphism, 535
- Lat** (category of lattices and lattice homomorphisms), 448
 lattice, 126, 144, 444, 444–449, 454–456, 465, 480, 481, 500
 lattice homomorphism, 447, 447, 448, 465
 least fixed point, 501, 501, 502, 505, 508, 515, 520
 least upper bound, 125
 left adjoint, 334, 335
 lower bounds, 125
 lower closure, 128, 129, 482
 lower set, 119, 123, 127, 127, 128, 130, 486
LTI (category of finite-dimensional linear time-invariant
 systems), 271
- magma, 136, 136–138
Mat $_{\mathbb{R}}$ (category of real matrices), 205
 matrix groups, 144, 160
 matrix multiplication, 205
 meet, 126, 126, 444, 454, 491, 492
Mon (category of monoids and morphisms), 204, 281
 monad, 481, 525, 526, 531, 535, 537

monoid, 135, 136, **141**, 141–145, 147, 151, 157, 159–162, 165, 166, 168, 185, 186, 192, 204, 221, 222, 281, 346, 440, 529
 monoid morphism, **157**, 157–160, 204
 monoidal category, 7, 346, **359**, 360, 364, 378, 389, 396, 397, 484
 monoidal poset, **440**, 440, 442
 monotone Galois connection, **332**
 monotone map, **110**, 110, 113–116, 119, 121, 129, 206, 282, 284, 332, 440, 442, 462, 473, 482, 483, 485, 486, 488, 515, 527, 528
Moo (semicategory of Moore machines), **261**
 Moore machine, 254–257, **260**, 260, 261, 263, 264, 266–268, 296, 346, 556
Mor (semicategory of More machines), **268**
 More machine, 268, 269

 natural isomorphism, **316**, 334–336, 340, 359, 360, 364
 natural transformation, 7, 309, **310**, 310, 311, 313, 316, 320–322, 335, 340, 360, 531–533, 538
 neutral element of a monoid, 18, 142–145, 147, 148, 157, 159, 160, 162, 192, 222

 opposite category, **238**, 238
 opposite poset, **105**, 238
 order isomorphism, **113**
 order on monotone maps, **121**

 partition, **76**, 421
 path in a graph, **180**
 permutation, **472**
Pos (category of posets and monotone maps), **206**
 poset, **91**, 91, 93, 95, 97, 99, 101–108, 110, 111, 113, 114, 116–121, 124, 125, 128, 131, 206, 209, 223, 224, 226, 238–240, 282, 332, 366–369, 411, 421, 423, 425, 429, 440, 442, 444, 446, 450, 462, 464, 484–486, 500, 504, 505, 521, 527
 positive definite matrices, **94**
 $\langle \text{Pos} \rangle$ (category of tuple-posets and monotone maps), **367**
 power poset, **96**
 powerset, **37**, 37, 38, 119, 313, 531
 pre-ordered set, **90**, 90, 91, 120
 product (of posets) \times , **102**, 440
 product of categories, **236**
 product of functions \times , **47**
 profunctor, 482

 real vector space, **170**, 205
 reflexive relation, **73**, 74, 76, 90, 92, 209
 $\langle \text{Rel} \rangle$ (category of tuple-sets and relations), **367**
Rel (category of sets and relations), **203**
 relation, **65**, 68
 relation on a semigroup, **162**
 right adjoint, 334, **335**
 right dual object, **397**

 Scott continuity, **500**, 500, 501, 506, 507

 semicategory, v, 7, 185–188, **189**, 189–192, 195, 197, 230, 231, 256, 257, 261, 266, 267, 279, 351, 354, 371, 373, 374, 413, 429, 430, 442, 527
 semicategory action, **267**, **282**, 282
 semifunctor, 7, **279**, 279, 280, 282, 435
 semigroup, 6, 136, **137**, 137–143, 151–154, 157, 160–163, 165, 169, 185, 186, 192, 204, 266
 semigroup action, 172, 266
 semigroup morphism, **152**, 152–154, 157, 159, 204, 266, 282
 $\langle \text{Set} \rangle$ (category of tuple-sets and functions), **258**
Set (category of sets and functions), **193**, 281
SGrp (category of semigroups and morphisms), **204**
 single-valued, **71**, 71, 72
 skeletal category, 209
 slice category, **242**
 stacking semicategory, **371**, 374
 standard action of Moore machines, **267**
 strict monoidal category, 7, **357**, 378
 strictly monotone, **473**
 strong monoidal functor, **364**
 subcategory, 7, **290**, 290–293, 367, 368, 504
 subgroup, **148**
 sublattice, 502
 submonoid, **143**, 143
 subsemigroup, **139**
 sum of functions $+$, **47**
 supremum, **125**, 455, 500, 501
 surjective function, **44**, 44, 45, 80, 202
 surjective relation, **71**, 71, 72
 symmetric monoidal category, **378**, 379, 380, 384, 387, 394, 399, 400, 485, 488
 symmetric monoidal poset, **440**
 symmetric relation, **73**, 74, 76

 terminal object, **397**
 top element of a poset \top , 454, 500
 total relation, **74**, 91
 totally ordered set, **91**
 traced symmetric monoidal category, 385, 387, 480, 488, 522
 transitive closure of a relation, **74**, 75, 502
 transitive relation, **74**, 74–76, 90, 92, 93, 209
 transpose of a relation, **72**
 twisted arrow category, **240**
 twisted poset of intervals, **106**

 union of sets \cup , **36**
 unitality property, 226
 upper bounds, **125**
 upper closure, **128**, 128, 482, 499, 504
 upper set, 116, 119, **127**, 127–129, 284, 322, 480, 483, 484, 486, 489, 499, 504, 529
 upward-closed, **130**

Vect _{\mathbb{R}} (category of real vector spaces), **205**
 vector space, 174, 270, 283, 321, 322, 334, 338, 361, 389, 396, 397, 399
 width of a poset, **99**