

Applied Compositional Thinking

for Engineers



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.

Contents

1. Introduction	9
1.1. Thesis	10
1.2. Systems and components	10
1.3. What ACT can do for you	10
1.4. What ACT cannot do	10
1.5. Book organization	11
1.6. Contributors	11
A. A first look at Applied Category Theory	13
2. Overview	15
2.1. Everything is the same	16
2.2. Guidebook	16
3. Transmutation	17
3.1. Interfaces and transformations	18
3.2. Formal definition of category	20
3.3. Currency categories	22
4. Culture	27
4.1. Definitional impetus vs. computational goals	28
4.2. Things that don't matter	29
5. Connection	33
5.1. Mobility	34
5.2. Trekking in the Swiss Mountains	36
5.3. Generating categories from graphs	37
6. Relation	39
6.1. Distribution networks	40
6.2. Relations	42
6.3. Relational Databases	48
7. Computing	51
7.1. Databases, sets, functions	52
7.2. The Category Set	54
7.3. Propositions	54
8. Specialization	55
8.1. Notion of subcategory	56

8.2. Drawings	56
8.3. Other examples of subcategories in engineering	57
8.4. Subcategories of Berg	59
9. Sameness	61
9.1. Sameness in category theory	62
9.2. Isomorphism is not identity	64
10. Universal properties	67
10.1. Universal properties	68
11. Trade-offs	69
11.1. Trade-offs	70
11.2. Ordered sets	70
11.3. Chains and Antichains	74
11.4. Upper and lower sets	75
11.5. From antichains to uppersets, and viceversa	77
11.6. Lattices	80
12. Poset constructions	83
12.1. Opposite of a poset	85
12.2. Poset of intervals	86
12.3. A different poset of intervals	86
13. Life is hard	87
13.1. Monotone maps	88
13.2. Compositionality of monotonicity	90
13.3. The category Pos of posets and monotone maps	90
13.4. Why Pos is not sufficient for design theory	91
14. Functors	93
14.1. Functors	94
14.2. A poset as a category	95
14.3. Other examples of functors	96
15. Up the ladder	99
15.1. Functor composition	100
15.2. A category of categories	100
15.3. Full and faithful functors	101
15.4. Forgetful functor	101
16. Duality	103
16.1. Galois connections	104
17. Naturality	109
17.1. Natural transformations	110
18. Adjunctions	113
18.1. An example	114

18.2. Adjunctions: hom-set definition	114
18.3. Adjunctions: (co)unit definition	114
18.4. Example of a “Product-Hom” adjunction	116
18.5. Example of a “Free-Forgetful” adjunction	117
18.6. Relating the two definitions	118
19. Combination	121
19.1. Products	122
19.2. Coproduct	127
19.3. Other examples	133
20. Counter-examples	135
20.1. Not quite categories	136
20.2. Not quite functors	136
B. Monotone Co-Design	137
21. Design	139
21.1. What is “design”?	140
21.2. What is “co-design”?	140
21.3. Basic concepts of formal engineering design	142
21.4. Queries in design	144
22. Design problems	147
22.1. Design Problems	148
22.2. Querying a DPI	156
22.3. Co-design problems	157
22.4. Discussion of related work	160
23. Feasibility	165
23.1. Design problems as monotone maps	166
23.2. Series composition of design problems	168
23.3. The category of design problems	172
23.4. DP Isomorphisms	173
24. Profunctors	175
24.1. Profunctors	176
24.2. Hom Profunctor	176
24.3. Other examples of profunctors	176
24.4. The bicategory of profunctors	176
24.5. DPI as profunctors	176
25. Parallelism	177
25.1. Monoids	178
25.2. Monoids homomorphism	179
25.3. Dynamical systems and monoids	179
25.4. Monoidal posets	180
25.5. Monoidal categories	181

25.6. DP is a monoidal category	185
26. Feedback	191
26.1. Trace of a linear transformation	192
26.2. Continuous LTI	192
26.3. Feedback in category theory	193
26.4. Feedback in design problems	193
27. Ordering design problems	199
27.1. Ordering DPs	200
27.2. Restrictions and alternatives	201
27.3. Interaction with composition	202
28. Constructing design problems	205
28.1. Creating design problems from catalogues	206
28.2. Companions and conjoint	208
28.3. Sum and intersection with companion and conjoints	210
28.4. Monoidal DPs	212
C. Computation	213
29. From math to implementation	215
30. Solving	217
30.1. Types of queries	218
30.2. Computational representation of DPs	218
30.3. Problem statement and summary of results	219
30.4. Composition operators for design problems	220
30.5. Decomposition of MCDPs	222
30.6. Monotonicity as compositional property	224
30.7. Solution of MCDPs	228
30.8. Example: Optimizing over the natural numbers	229
30.9. Complexity of the solution	232
30.10. Extended Numerical Examples	234
30.11. Monotonicity and fixed points	243
31. Uncertainty	245
31.1. Monads	246
31.2. Using monads to understand uncertainty	247
31.3. L and U monads	247
D. Higher-order theory	253
32. Enrichments	255
32.1. Enrichments	256
32.2. Enriched categories	256
32.3. Set-enriched DPs (DPIs)	258

33. Negative designs	265
E. Operadic structures	267
34. Wiring diagrams	269
35. Operads	271
36. Recursion	273
36.1. Recursive Design Problems	273
37. Higher-order design	275
37.1. Diagrams	276
37.2. Diagram as a monotone function	276
37.3. Representation Results	276
37.4. Compact closed structure	276
37.5. A locally-posetal pro-arrow equipment	280
F. Networks and systems	283
38. Decorated co-spans	285
G. Extensions of co-design theory	287
39. Linear logic	289
40. Linear DPs	291
41. Temporal DPs	293
H. Control theory in category theory	295
I. Case study: co-design of AV fleets	299
42. Co-design of control systems	301
43. Co-design of autonomous systems	303
44. Co-design of mobility systems	305

J. To move	313
45. Paper to dismember and move	315
45.1. Introduction	315
45.2. Conclusions	318
46. Other paper	319
46.1. Introduction	319
46.2. Design Problems	322
46.3. Monotone Co-Design Problems	323
46.4. Semantics of MCDPs	324
46.5. Solution of MCDPs	326
46.6. Uncertain Design Problems	326
46.7. Partial order \leq_{DP}	327
46.8. Uncertain DPs (UDPs)	327
46.9. Order on UDP	328
46.10. DPs as degenerate UDPs	328
46.11. Interconnection of Uncertain Design Problems	328
46.12. Approximation results	329
46.13. Applications	331
46.14. Application: Dealing with Parametric Uncertainty	331
46.15. Application: Introducing Tolerances	332
46.16. Application: Relaxation for relations with infinite cardinality	334
46.17. Conclusions and future work	337
46.18. Proofs	338
46.19. Software	343
46.20. Source code	343
46.21. Virtual machine	343
47. Relationship between products	357
47.1. Biproduct, Product, and Coproduct of Design Problems	357
47.2. Relationship between intersection and monoidal product	363
48. Computation	367

1. Introduction

to write

Contents

1.1.	Thesis	10
1.2.	Systems and components	10
1.3.	What ACT can do for you	10
1.4.	What ACT cannot do	10
1.5.	Book organization	11
1.6.	Contributors	11



1.1. Thesis

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.

1.2. Systems and components

What is a “system”?

Here is a great quote¹:

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

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.

Haiken 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” [29].

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 (Fig. 1.1). 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.

Make better figure

1.3. What ACT can do for you

describe how ACT can help

1.4. What ACT cannot do

describe limitations of ACT

¹This 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 in [34], but ultimately sourceless and probably apocryphal.

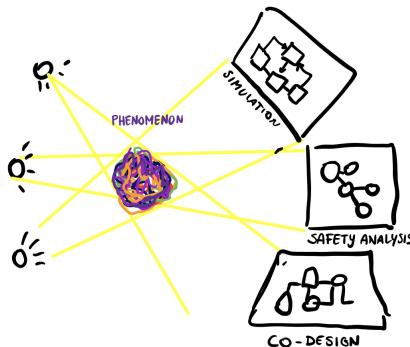


Figure 1.1.: Engineers live in a Plato's cave with multiple light sources. A certain phenomenon can be illuminated from different angles and look very different, very simple or complex.

1.5. Book organization

To write.

1.6. Contributors

To write.

Part A.

A first look at Applied Category Theory

2. Overview

to write

Contents

2.1. Everything is the same	16
2.2. Guidebook	16



2.1. Everything is the same

Reproduce here the discussion from the first lecture, about many different things that are actually the same.

2.2. Guidebook

Broader overview of the abstractions that we will develop in this book, including posets, lattices, etc. up to operad.

3. Transmutation

to write

Contents

3.1. Interfaces and transformations	18
3.2. Formal definition of category	20
3.3. Currency categories	22



3.1. Interfaces and transformations

Gives more examples before getting to the definition. Give more examples with different conventions for the arrow direction.

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 chain 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 “transmuters”: the **motor** and **wheels**.
2. The “transmuted”: the **electricity**, the **rotation**, the **translation**.

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. 3.1).



Figure 3.1.: 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

$$\text{X} \xrightarrow{\text{transmuter}} \text{Y} \quad (3.1)$$

means that “using transmuter, having **Y** is sufficient to produce **X**”.

Remark 3.1 (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 Eq. (3.1) would be as follows:

$$\text{transmuter} : \text{X} \rightarrow \text{Y}. \quad (3.2)$$

This is now to you something syntactically familiar; when we study the categories of sets and functions between sets we will see that in that context the familiar meaning is also the correct meaning.

With these conventions, we can describe the two transmutes as these arrows:

$$\text{motor} : \text{rotation} \rightarrow \text{electricity}, \quad (3.3)$$

$$\text{wheels} : \text{translation} \rightarrow \text{rotation}. \quad (3.4)$$

We can put these arrows in the diagrams, and obtain the following (Fig. 3.2).

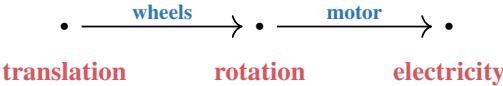


Figure 3.2.: Transmutes are arrows between resources.

In this representation, the arrows are the components of the system. We will learn how to compose these arrows according to the rules of category theory. The basic rule will be *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. 3.3).

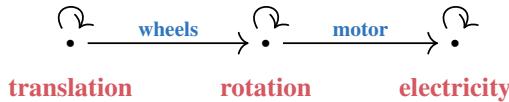


Figure 3.3.: System components and identities.

Furthermore, we might consider the idea of composition of arrows. Suppose that we know that

$$\mathbf{X} \xrightarrow{\mathbf{a}} \mathbf{Y} \quad \text{and} \quad \mathbf{Y} \xrightarrow{\mathbf{b}} \mathbf{Z},$$

that is, using a b we can get a \mathbf{Y} from a \mathbf{Z} , and using an a we can get a \mathbf{X} from a \mathbf{Y} , then we conclude that using and a and a b we can get an \mathbf{X} from a \mathbf{Z} .

In our example, if the arrows **wheels** and **motor** exist, then also the arrow “**wheels** then **motor**” exists (Fig. 3.4).

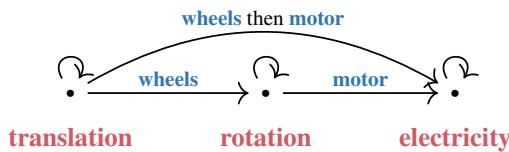


Figure 3.4.: Composition of system components.

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. 3.5 shows two models of motors (**motor A**, and **motor B**) and two models of wheels (**wheels U** and **wheels V**).

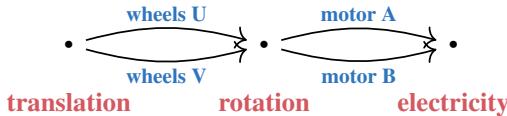


Figure 3.5.: Multiple models for wheels and motors.

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**;

A “category” is an abstract mathematical structure that captures the properties of these systems of points and arrows and the logic of composition.

The basic terminology is that the points are called **objects**, and the arrows are called **morphisms**.

In our example, the **motor** and the **wheels** are the morphisms, and **electricity**, **rotation**, **translation** are the objects.

Many things can be defined as categories and we will see many examples in this book.

We are now just biding our time before introducing the formal definition of category. At first sight it will be intimidating: there are four parts to the definition, two axioms to define. Moreover, it is quite a bit technical and it takes half a page to write.

3.2. Formal definition of category

The following is the formal definition.

Definition 3.2 (Category). A *category* **C** is specified by four components:

1. **Objects**: a collection¹ Ob_C , whose elements are called *objects*.
2. **Morphisms**: for every pair of objects $X, Y \in \text{Ob}_C$, there is a set $\text{Hom}_C(X, Y)$, elements of which are called *morphisms* from X to Y . The set is called the “hom-set from X to Y ”.
3. **Identity morphisms**: for each object X , there is an element $\text{id}_X \in \text{Hom}_C(X, X)$ which is called *the identity morphism of X*.
4. **Composition rules**: given any morphism $f \in \text{Hom}_C(X, Y)$ and any morphism $g \in \text{Hom}_C(Y, Z)$, there exists a morphism $f \circ g$ in $\text{Hom}_C(X, Z)$ which is the *composition of f and g*.

Furthermore, the constituents are required to satisfy the following conditions:

1. *Unitality*: for any morphism $f \in \text{Hom}_C(X, Y)$,

$$\text{id}_X \circ f = f = f \circ \text{id}_Y. \quad (3.5)$$

2. *Associativity*: for $f \in \text{Hom}_C(X, Y)$, $g \in \text{Hom}_C(Y, Z)$, and $h \in \text{Hom}_C(Z, W)$,

$$(f \circ g) \circ h = f \circ (g \circ h). \quad (3.6)$$

Remark 3.3 (Are we sure we are not going in the wrong direction?). We denote composition of morphisms in a somewhat unusual way—sometimes preferred by category-theorists and computer scientists—namely in *diagrammatic order*.

That is, given $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, we denote their composite by $(f ; g) : X \rightarrow Z$, pronounced “ f then g ”. This is in contrast to the more typical notation for composition, namely $g \circ f$, or simply gf , which reads as “ g after f ”. The notation $f ; g$ is sometimes called *infix notation*.

We promise, at some point it will be clear what are the advantages of seemingly doing everything in the wrong direction.

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 Definition 3.2, they always exist.
- Given arrows $X \rightarrow Y$ and $Y \rightarrow Z$, we do not need to draw their composition because, by Definition 3.2, 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. 3.6).

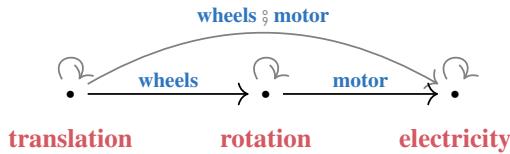


Figure 3.6.: Electric car example. The grey arrows are implied by the properties of a category.

In particular, the electric car example corresponds to the category **C** specified by

- *Objects*: $\text{Ob}_C = \{\text{electricity}, \text{rotation}, \text{translation}\}$.
- *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. 3.7, Fig. 3.8).

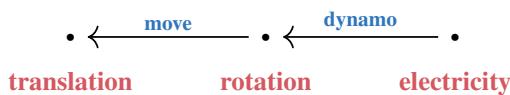


Figure 3.7.: Electric power can be produced from motion.

¹A “collection” is something which may be thought of as a set, but may be “too large” to technically be a set in the formal sense. This distinction is necessary in order to avoid such issues as Russel’s paradox.

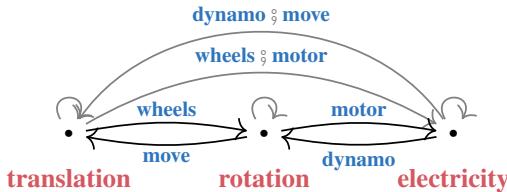


Figure 3.8.: Electric car example: forward and backward transformations.

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 **wheels** ; **motor** ; **dynamo** ; **move**, 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. 3.9). 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.

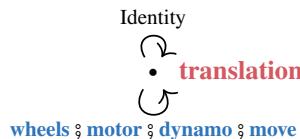


Figure 3.9.: There can be multiple morphisms from an object to itself.

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. Fig. 3.10 shows an example of a diagram that describes a process which is definitely not invertible.



Figure 3.10.: An example of a process which is not invertible.

3.3. 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, e.g., as offered by a currency exchange service.

We start with a set **C** of labels for all the currencies we wish to consider, i.e.:

$$\mathbf{C} = \{\text{EUR}, \text{USD}, \text{CHF}, \text{SGD}, \dots\}.$$

For each currency $\mathbf{c} \in \mathbf{C}$ we define an object $\mathbb{R} \times \{\mathbf{c}\}$ which represents possible amounts of the given currency \mathbf{c} (we will ignore the issue of rounding currencies to an accuracy of

two decimal places, and we allow negative amounts). The currency label keeps track of which “units” we are using.

Now consider two such objects, say $\mathbb{R} \times \{\text{USD}\}$ and $\mathbb{R} \times \{\text{EUR}\}$. 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 a and a commission b for the transaction. Given an amount $x \in \mathbb{R}$ of USD, we define a morphism (a currency exchanger) as:

$$E_{a,b} : \mathbb{R} \times \{\text{USD}\} \rightarrow \mathbb{R} \times \{\text{EUR}\},$$

by the formula

$$\langle x, \text{USD} \rangle \mapsto \langle ax - b, \text{EUR} \rangle.$$

Note that the commission is given in the units of 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 $\mathbb{R} \times \{\text{USD}\}$ to $\mathbb{R} \times \{\text{EUR}\}$.

To build our category, we also need to specify how currency exchangers compose. Given currencies c_1, c_2, c_3 , and given currency exchangers

$$E_{a,b} : \mathbb{R} \times \{c_1\} \rightarrow \mathbb{R} \times \{c_2\} \quad \text{and} \quad E_{a',b'} : \mathbb{R} \times \{c_2\} \rightarrow \mathbb{R} \times \{c_3\}$$

we define the composition $E_{a,b} ; E_{a',b'}$ to be the currency exchanger

$$E_{aa',a'b+b'} : \mathbb{R} \times \{c_1\} \rightarrow \mathbb{R} \times \{c_3\}. \quad (3.7)$$

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 unit morphisms for our category. These are currency exchangers which “do nothing”. For any object $\mathbb{R} \times \{c\}$, its identity morphism is

$$E_{1,0} : \mathbb{R} \times \{c\} \rightarrow \mathbb{R} \times \{c\},$$

the currency exchanger with exchange rate “1” and commission “0”.

It is straightforward to check that the composition of currency exchangers as defined above obeys the associative law, and that the identity morphisms act neutrally for composition. Thus we indeed have a category!

Remark 3.4. In the above specification of our category of currency exchangers, we can actually just work with the set of currency labels C as our objects, instead of using “amounts” of the form $\mathbb{R} \times \{c\}$ as our objects. Indeed, on a mathematical level, the definition of currency exchangers and their composition law Eq. (3.7) do not depend on using amounts! Namely, a currency exchanger $E_{a,b}$ is specified by the pair of numbers $\langle a, b \rangle$, and the composition law Eq. (3.7) may then, in this notation, be written as

$$\langle a, b \rangle ; \langle a', b' \rangle = \langle a'a, a'b + b' \rangle. \quad (3.8)$$

The interpretation is still that currency exchangers change amounts of one currency to amounts in another currency, but for this we do not need to carry around copies

of \mathbb{R} in our notation.

Following the above remark:

Definition 3.5 (Category \mathbf{Curr}). The *category of currencies \mathbf{Curr}* is specified by:

1. *Objects:* a collection of currencies \mathbf{C} .
2. *Morphisms:* given two currencies $c_1, c_2 \in \mathbf{C}$, morphisms between them are currency exchangers $\langle a, b \rangle$ from c_1 to c_2 .
3. *Identity morphism:* given an object $c \in \mathbf{C}$, 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 Eq. (3.8).

As an illustration, consider three currency exchange companies **ExchATM**, **MoneyLah**, and **Frankurrencies**, which operate on several currencies (Table 3.1).

Company name	Exchanger label	Direction	a (exchange rate)	b (fixed commission)
ExchATM	A	USD to CHF	0.95 CHF/USD	2.0 CHF
ExchATM	B	CHF to USD	1.05 USD/CHF	1.5 USD
ExchATM	C	USD to SGD	1.40 SGD/USD	1.0 SGD
MoneyLah	D	USD to CHF	1.00 CHF/USD	1.0 CHF
MoneyLah	E	SGD to USD	0.72 USD/SGD	3.0 USD
Frankurrencies	F	EUR to CHF	1.20 CHF/EUR	0.0 CHF
Frankurrencies	G	CHF to EUR	1.00 EUR/CHF	1.0 EUR

Table 3.1.: Three currency exchange companies operating different currencies.

We can represent this information as a graph, where the nodes are the currencies and the edges are particular exchange operations (Fig. 3.11).

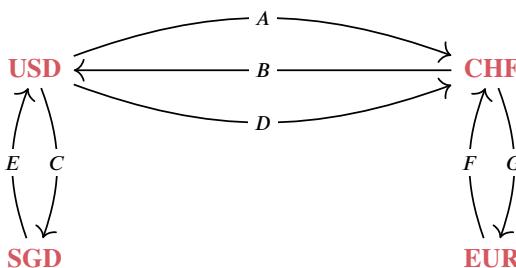


Figure 3.11.: Three currency exchange companies operating different currencies as a graph.

There is a currency category built from the information in Table 3.1 and the graph in Fig. 3.11. Its collection of objects is the set $\{\text{EUR, USD, CHF, SGD}\}$, 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 3.1,

- all possible compositions of the currency exchangers listed in Table 3.1.

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. 3.11, and obtaining from it an associated category, called the *free category on G* . We introduce this concept in the next section.

Exercise 3.6 (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.

What relation is there with the currency category?

4. Culture

to write

Contents

4.1. Definitional impetus vs. computational goals	28
4.2. Things that don't matter	29



4.1. 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:

$$\langle \underbrace{eca}_g, \underbrace{ecb + ed + f}_h \rangle.$$

Assuming $e = a^{-1}$ (i.e., an exchange rate direction is not more profitable than the other), and $h \neq 0$, because of the commissions one 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.

4.2. 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.
- 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*.¹ 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**.

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 in an intuitive way.

4.2.1. 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 viceversa. For example, the formula

$$(f ; g) ; h = f ; (g ; h)$$

would be transformed in

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

(A bit more advanced category theory can describe this transformation more precisely.)

The same considerations apply for the convention regarding the arrow directions. If we have a category with morphisms such as

$$\text{motor} : \text{rotation} \rightarrow \text{electricity}$$

with the semantics of "the **motor** can produce **rotation** given **electricity**", we could define a *different* category, where the conventions are inverted. In this other category, for which

¹IEEE Spectrum

we use arrows of different color, we would write

motor : **electricity** → **rotation**

and the semantics would be “the **motor** consumes **electricity** to produce **rotation**” (Fig. 4.1).

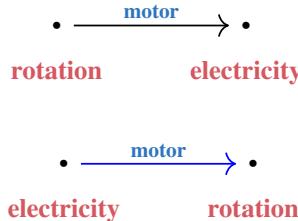


Figure 4.1.: Opposite convention for arrows direction.

These two categories would have the same objects, and the same number of arrows; it's just that the arrows change direction when moving from one category to the other. In particular, there exists a transformation which maps every black arrow to a blue arrow, reverting the direction (Fig. 4.2). This transformation is invertible. Intuitively, we would not expect anything substantial to change, because we are just changing a convention. We will see that there is a concept called *opposite category* that formalizes this idea of reversing the direction of the arrows.

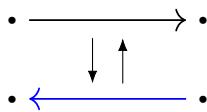


Figure 4.2.

Diagrams conventions don't matter

Now that we are flexed our isomorphism muscles, we can also talk about the isomorphisms of the visual language.

In engineering, “boxes and wires” diagrams are commonly used to talk about materials transformations and signal flows. In those diagrams one would use boxes to describe the processes and the wires to describe the materials or information that is being transformed. Boxes have “inputs” and “outputs”, and arrows have directions representing the causality. From left to right, what is to the left causes what is to the right. The left-to-right directionality seems an utterly obvious choice for most of you who learned languages that are written left-to-right, top-to-bottom as in this book².

²Note that this paragraph cannot be translated literally to Japanese. It breaks the assumption that we made before, about the fact that we can have a 1:1 literal translation of this book without changing the meaning. You might think that our future hypothetical Japanese translator can make an outstanding job and translate also our figures to go right-to-left, then saying that right-to-left is natural to people that write right-to-left. However, that does not work, because in fact Japanese engineers also use left-to-right diagrams.

Fig. 4.3 shows how we would have visually described the first example using the boxes-and-wires conventions. Again, we say that this is just a different convention, because we have a procedure to transform one diagram into the other. This is not as simple as changing the direction of the arrows as in the case of an opposite category. Rather, to go from points-and-arrows to boxes-and-wires:

- Arrows that describe transmutes become boxes that describe processes;
- The points that describe the resources become wires between the boxes.

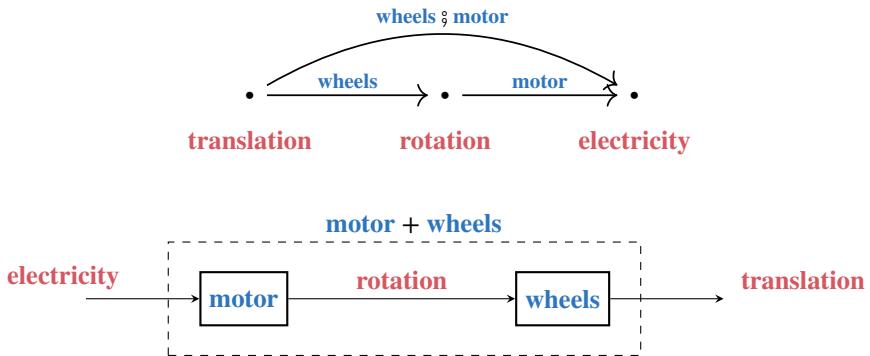


Figure 4.3.: Isomorphisms of resource diagrams.

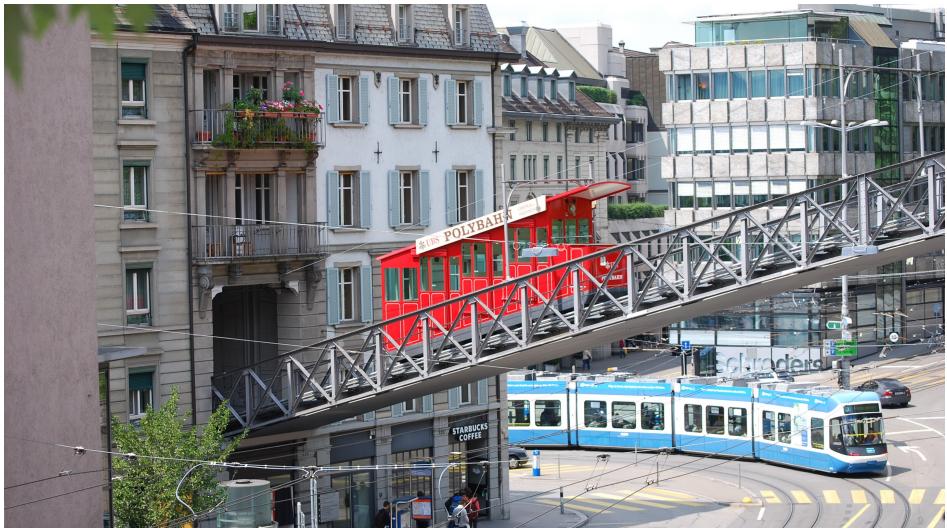
Expand example as in the slides

5. Connection

Currency categories illustrated how one can use category theory to think about things transforming into each other. In this chapter, we want to think about how things connect to each other.

Contents

5.1. Mobility	34
5.2. Trekking in the Swiss Mountains	36
5.3. Generating categories from graphs	37



5.1. Mobility

For a specific mode of transportation, say a car, we can define a graph

$$G_c = \langle V_c, A_c, s_c, t_c \rangle,$$

where V_c represents geographical locations which the car can reach and A_c represents the paths it can take (e.g. roads). Similarly, we consider a graph $G_s = \langle V_s, A_s, s_s, t_s \rangle$, representing the subway system of a city, with stations V_s and subway lines going through paths A_s , and a graph $G_b = \langle V_b, A_b, s_b, t_b \rangle$, representing onboarding and offboarding 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 $G = (V, A, s, t)$ with $V = V_c \cup V_s \cup V_b$ and $A = A_c \cup A_s \cup A_b$, we obtain the desired intermodality graph. Graph G can be seen as a new category, with objects V and morphisms A .

Example 5.1. Consider the **car** category, describing your road trip in California, with

$$V_c = \{SFO_c, S. Mateo, Half Moon Bay, SBP_c, Lake Balboa, LAX_c\},$$

and arrows as in Fig. 5.1. The nodes represent typical touristic road-trip checkpoints in California and the arrows represent famous highways connecting them.

SFO_c — US101 → S. Mateo — CA92 → H. M. Bay — CA1 → SBP_c - US101S → Lake Balboa — I405 →

Figure 5.1.: The **car** category.

Furthermore, consider the **flight** category with $V_f = \{SFO_f, SJC, SBP_f, LAX_f\}$ and arrows as in Fig. 5.2. The nodes represent airports in California and the arrows represent connections, offered by specific flight companies.

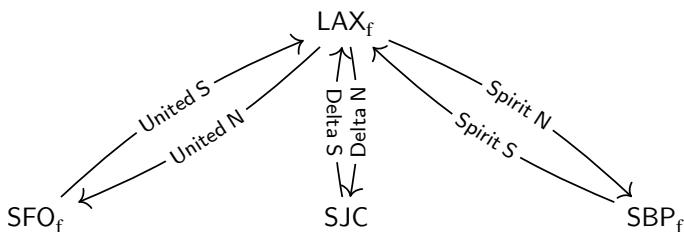
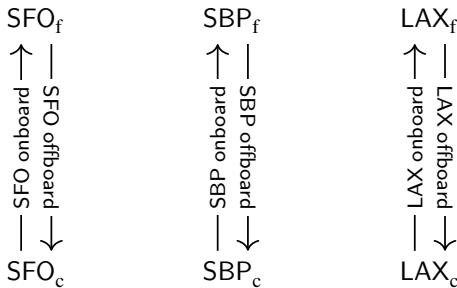


Figure 5.2.: The **flight** category.

We then consider the **board** category, with nodes

$$V_b = \{SFO_f, SFO_c, SBP_f, SBP_c, LAX_f, LAX_c\}$$

and arrows as in Fig. 5.3. Nodes represent airports and airport parkings, and arrows represent the onboarding and offboarding paths one has to walk from the parkings to the airport and vice-versa.

Figure 5.3.: The **board** category.

The combination of the three, which we call the *intermodal graph*, can be represented as a graph, with red arrows for the car network, blue arrows for the flight network, green arrows for the boarding network, and black dashed arrows for intermodal morphisms, arising from composition of morphisms involving multiple modes (Fig. 5.4). Imagine that you are in the parking lot of LAX airport and you want to reach S. Mateo. From there, you will e.g. onboard to a United flight to SFO_f , will then offboard reaching the parking lot SFO_c , and drive on highway US-101 reaching S. Mateo. This is intermodality.

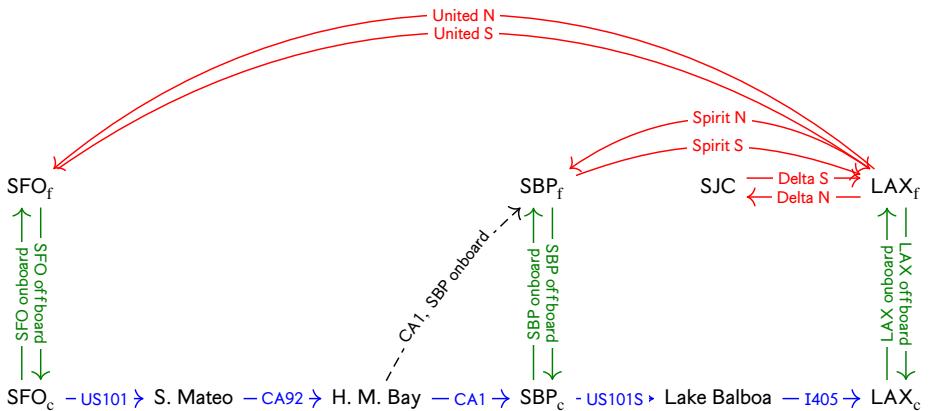


Figure 5.4.: Intermodal graph. The dashed arrows represent intermodal morphisms, and we depict just one of them for simplicity.

The intermodal network category **intermodal** is the free category on the graph illustrated in Fig. 5.4.

5.2. Trekking in the Swiss Mountains

In the section we'll 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 , i.e. continuously differentiable. This means that our map of the landscape forms a C^1 -manifold; let's call it 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 (steapness) of a path in any given point, as given by its derivative.

Definitely we need a picture of Swiss mountains

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 5.2 (Berg). Let **Berg** be the category defined as follows:

- 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).
- 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).
- For any object $\langle p, v \rangle$, we define its identity morphism $1_{\langle p, v \rangle} = \langle \gamma, 0 \rangle$ formally: its path γ is defined on the closed interval $[0, 0]$, i.e. $T = 0$ and $\gamma(0) = p$. We declare this path to be C^1 by convention, and declare its derivative at 0 to be v .
- Given morphisms $\langle \gamma_1, T_1 \rangle : \langle p_1, v_1 \rangle \rightarrow \langle p_2, v_2 \rangle$ and $\langle \gamma_2, T_2 \rangle : \langle p_2, v_2 \rangle \rightarrow \langle p_3, v_3 \rangle$, their composition is $\langle \gamma, T \rangle$ with $T = T_1 + T_2$ and

$$\gamma(t) = \begin{cases} \gamma_1(t) & 0 \leq t \leq T_1 \\ \gamma_2(t - T_1) & T_1 \leq t \leq T_1 + T_2. \end{cases} \quad (5.1)$$

Make a technical sketch of the manifold showing what are the velocities, how do paths look like, etc.

Since we are only amateurs, we don't feel comfortable with 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 homset $\text{Hom}(\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle)$, a function

$$\text{MaxSteepness} : \text{Hom}(\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle) \longrightarrow [0, 1].$$

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*. Let's 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.

5.3. Generating categories from graphs

Put this section after all concrete examples

To begin, we recall some formal definitions related to (directed) graphs.

Need nice pictures of graphs and various quantities.

Definition 5.3 (Graph). A (directed) *graph* $G = \langle V, A, s, t \rangle$ consists of a set of vertices V , a set of arrows A , and two functions $s, t : A \rightarrow V$, called the *source* and *target* functions, respectively. Given $a \in A$ with $s(a) = v$ and $t(a) = w$, we say that a is an *arrow* from v to w .

Remark 5.4. Both directed graphs and undirected graphs play a prominent role in many kinds of mathematics. In this text, we work primarily with directed graphs and so, from now on, we will drop the “directed”: unless indicated otherwise, the word “graph” will mean “directed graph”.

Definition 5.5 (Path). Let G be a graph. A *path* in G is a sequence of arrows such that the target of one arrow is the source of the next. The *length* of a path is the number of arrows in the sequence. We also formally allow for sequences made up of “zero-many” arrows (such paths therefore have length zero). We call such paths *trivial* or *empty*. If paths describe a journey, then trivial paths correspond to “not going anywhere”. The notions of source and target for arrows extend, in an obvious manner, to paths. For trivial paths, the source and target always coincide.

The following definition provides a way of turning any graph into a category.

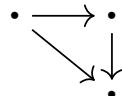
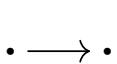
Definition 5.6 (Free category on a graph). Let $G = \langle V, A, s, t \rangle$ be a graph. The *free category on G* , denoted **Free**(G), has as objects the vertices V of G , and given vertices $x \in V$ and $y \in V$, the morphisms **Free**(G)(x, y) are the paths from x to y . The composition of morphisms is given by concatenation of paths, and for any object $x \in V$, the associated identity morphism id_x is the trivial path which starts and ends at x .

Show a picture of a graph and its induced category.

We leave it to the reader to check that the above definition does indeed define a category.

Let's do it ourselves

Exercise 5.7. Consider the following five graphs. For each graph G , how many morphisms in total are there in the associated category $\mathbf{Free}(G)$?



6. Relation

to write

Contents

6.1. Distribution networks	40
6.2. Relations	42
6.3. Relational Databases	48



6.1. 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, transistor stations, low voltage transmission lines and nodes, and consumers (e.g. homes and businesses). The situation is depicted in Fig. 6.1.

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

Figure 6.1.: Components of electrical power grids.

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. 6.2.

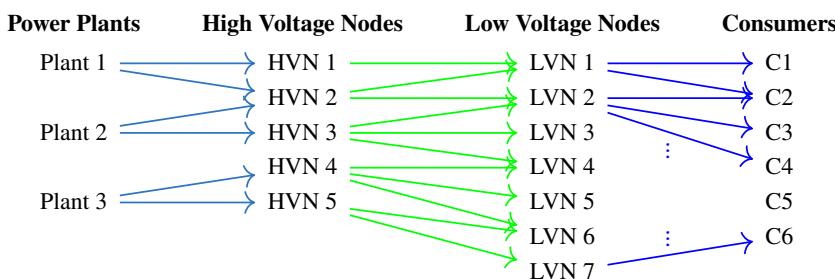


Figure 6.2.: Connectivity between components in electric power grids.

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 paths traced by sequences of arrows, as in Fig. 6.3. 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. 6.2, we can also include, for example, information about the connectivity of high voltage nodes among themselves, as

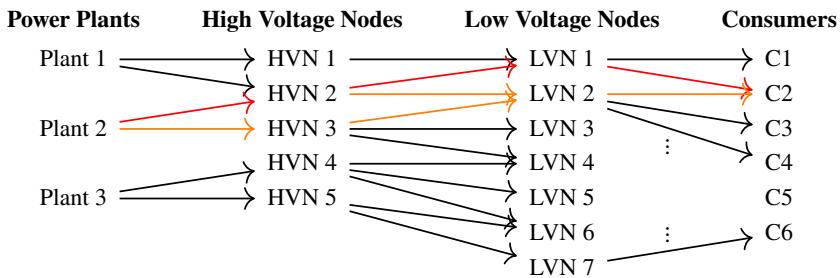


Figure 6.3.: Connection between consumers and power plants.

in Fig. 6.4.

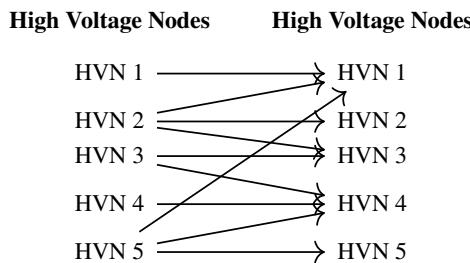


Figure 6.4.: Connectivity between high voltage nodes.

The information encoded in Fig. 6.4 and Fig. 6.2 can also be displayed as a single graph, see Fig. 6.5, Fig. 6.4. If we ignore the directionality of the arrows, this is analogous to a

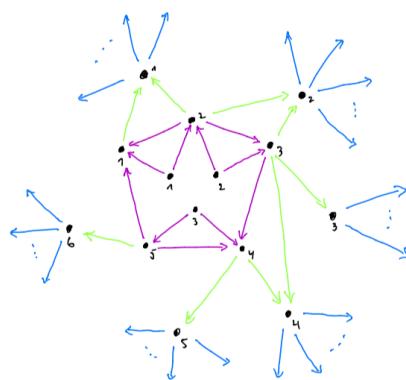


Figure 6.5.: Alternative visualization for connectivity.

depiction of type shown in Fig. 6.6, which is a schema of a power grid.¹

¹See https://en.wikipedia.org/wiki/Electrical_grid and <https://doi.org/10.1109/JSYST.2015.2427994>

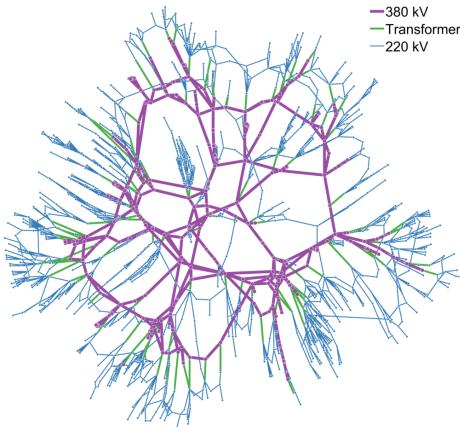


Figure 6.6.: A schematic view of a power grid.

6.2. Relations

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

Definition 6.1 (Binary relation). A *binary relation* from a set X to a set Y is a subset of the Cartesian product $X \times Y$.

Remark 6.2. We will often drop the word “binary” and simply use the name “relation”.

If X and Y are finite sets, we can depict a relation $R \subseteq X \times Y$ graphically as in Fig. 6.7. For each element $\langle x, y \rangle \in X \times Y$, we draw an arrow from x to y if and only if $\langle x, y \rangle \in R \subseteq X \times Y$.

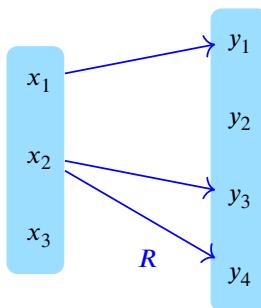


Figure 6.7.

We can also depict this relation graphically as a subset of $X \times Y$ in a “coordinate system way”, as in Fig. 6.8. The shaded grey area is the subset R defining the relation.

Exercise 6.3. Let $X = Y = \{1, 2, 3, 4\}$ and consider the relation $R \subseteq X \times Y$ defined by

$$R = \{\langle x, y \rangle \in X \times Y \mid x \leq y\}. \quad (6.1)$$

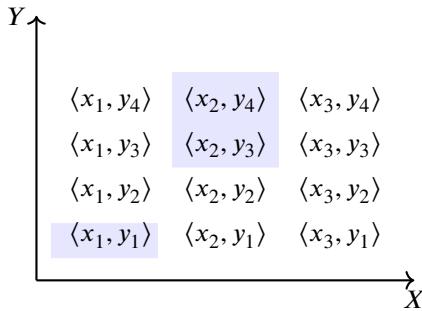


Figure 6.8.: Relations visualized in “coordinate systems”.

Visualize the relation R via the method in Fig. 6.7 and Fig. 6.8 each.

The visualization in Fig. 6.7 hints at the fact that we can think of a relation $R \subseteq X \times Y$ as a *morphism* from X to Y .

Definition 6.4 (Category **Rel**). The category **Rel** of relations **Rel** is given by:

1. *Objects*: The objects of this category are all sets.
2. *Morphisms*: Given sets X, Y , the homset $\text{Hom}_{\text{Rel}}(X, Y)$ consists of all relations $R \subseteq X \times Y$.
3. *Identity morphisms*: Given a set X , its identity morphism is

$$1_X := \{\langle x, y \rangle \mid x = y\}. \quad (6.2)$$

4. *Composition*: Given relations $R : X \rightarrow Y, S : Y \rightarrow Z$, their composition is given by

$$R ; S := \{\langle x, z \rangle \mid \exists y \in Y : (\langle x, y \rangle \in R) \wedge (\langle y, z \rangle \in S)\}. \quad (6.3)$$

To illustrate the composition rule in Eq. (6.3) for relations, let’s consider a simple example, involving sets X, Y , and Z , and relations $R : X \rightarrow Y$ and $S : Y \rightarrow Z$, as depicted graphically below in Fig. 6.9. Now, according to the rule in Eq. (6.3), the composition $R ; S \subseteq X \times Z$ will be such that $\langle x, z \rangle \in R ; S$ if and only if there exists some $y \in Y$ such that $\langle x, y \rangle \in R$ and $\langle y, z \rangle \in S$, which, graphically, means that for $\langle x, z \rangle$ to be an element of the relation $R ; S$, x and y 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. 6.9, there is an arrow from x_2 to y_3 , and from there on to z_1 , and therefore, in the composition $R ; S$ depicted in Fig. 6.10, there is an arrow from x_2 to z_1 .

Remark 6.5. Relations with the same source and target can be *compared* via inclusion. Given $R \subseteq X \times Y$ and $R' \subseteq X \times Y$, we can ask whether $R \subseteq R'$ or $R' \subseteq R$.

A question on your mind at this point might be: what is the relationship between relations and functions? One point of view is that functions are special kinds of relations.

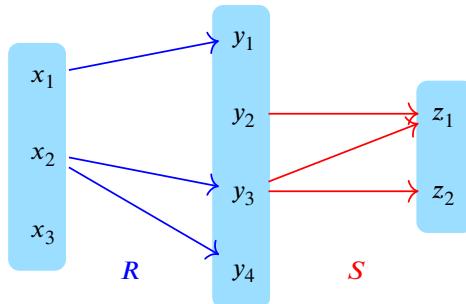


Figure 6.9.: Relations compatible for composition.

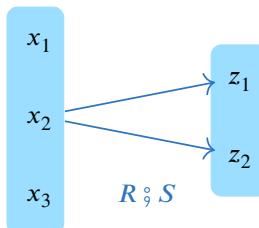


Figure 6.10.: Composition of relations.

Definition 6.6 (Functions as relations). Let X and Y be sets. A relation $R \subseteq X \times Y$ is a **function** if it satisfies the following two conditions:

1. $\forall x \in X \quad \exists y \in Y : \langle x, y \rangle \in R$
2. $\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R \text{ holds: } x_1 = x_2 \Rightarrow y_1 = y_2$.

What does this definition have to do with the “usual” way that we think about functions?

Let start with a relation $R \subseteq X \times Y$ satisfying the conditions of Definition 6.6. We’ll build from it a function $f_R : X \rightarrow Y$. Choose an arbitrary $x \in X$. According to point 1. in Definition 6.6, there exists a $y \in Y$ such that $\langle x, y \rangle \in R$. So let’s choose such a y , and call it $f_R(x)$. This gives us recipe to get from any x to a y . But maybe you are worried: given a specific $x \in X$, 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 $x \in X$ 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 $x_1 \neq x_2$.

Given a function $f : X \rightarrow Y$, we can turn it into a relation in a simple way: we consider its graph

$$R_f := \text{graph}(f) = \{\langle x, y \rangle \in X \times Y \mid y = f(x)\}.$$

The relation R_f encodes the same information that f encodes – simply in a different form. In this text, we take Definition 6.6 as our rigorous definition of what a function is. Nevertheless, we’ll often use functions “in the usual way”, e.g. we’ll write things like $y = f(x)$.

Another question you may be wondering about is this: if we define functions as special kinds of relations, how then do we define the composition of functions? The answer is that we compose functions simply by the rule for composing relations.

Lemma 6.7. Let $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ be relations which are functions. Then their composition $R ; S \subseteq X \times Z$ is again a function.

Proof. We check that $R ; S$ satisfies the two conditions stated in Definition 6.6.

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

$$\langle x_1, y_1 \rangle \in R \text{ and } \langle y_1, z_1 \rangle \in S,$$

$$\langle x_2, y_2 \rangle \in R \text{ and } \langle y_2, z_2 \rangle \in S.$$

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.

□

Example 6.8. Can we have a function (or relation) whose source is the empty set \emptyset ? Given any set Y , such a relation would be of the form $R \subseteq \emptyset \times Y := \emptyset$. This implies that $R = \emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $\emptyset \rightarrow Y$:

- For all $x \in \emptyset = \emptyset$, $\exists y \in Y$ such that $\langle x, y \rangle \in R$ (trivially satisfied).
- Clearly, given $\langle x, y \rangle, \langle x', y' \rangle \in R = \emptyset$, having $x = x'$ implies $y = y'$.

Therefore, the answer to the original question is yes.

Example 6.9. Can we have a function (or relation) whose target is the empty set \emptyset ? Again, given any set X , such a relation would be of the form $R \subseteq X \times \emptyset := \emptyset$. This, again, implies $R\emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $X \rightarrow \emptyset$:

- For all $x \in X, \exists y \in \emptyset = \emptyset$ such that $\langle x, y \rangle \in R$? Unless $X = \emptyset$, this is not satisfied. Therefore, given $X \neq \emptyset$, there is no function (or relation) $X \rightarrow \emptyset$.

Definition 6.10 (Properties of a relation). Let $R \subseteq X \times Y$ be a relation. R is:

1. *Surjective* if $\forall y \in Y \exists x \in X : \langle x, y \rangle \in R$;
2. *Injective* if $\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R$ it holds: $y_2 = y_2 \Rightarrow x_1 = x_2$;
3. *Defined-everywhere* if $\forall x \in X \exists y \in Y : \langle x, y \rangle \in R$;
4. *Single-valued* if $\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R$ it holds: $x_1 = x_2 \Rightarrow y_1 = y_2$.

Example 6.11. The relation depicted in Fig. 6.7 is injective but not surjective, i.e. if $\langle x, y \rangle, \langle x', y' \rangle \in R$ and $y = y'$, then $x = x'$.

One can notice a certain duality in the properties listed in Definition 6.10. This is made more precise through the following definition.

Definition 6.12 (Transpose of a relation). Let $R \subseteq X \times Y$ be a relation. The *transpose* (or *opposite, reverse*) of R is the relation given by:

$$R^T := \{\langle y, x \rangle \in Y \times X \mid \langle x, y \rangle \in R\}.$$

note that $R^T : Y \rightarrow X$, while $R : X \rightarrow Y$.

Remark 6.13. In the following, we list some properties which refer to relations and their opposites. It is a good exercise to prove them:

- $(R^T)^T = R$;
- If R is everywhere-defined, then R^T is surjective;
- If R is single-valued, then R^T is injective.
- If R is everywhere defined, then $\text{id}_X \subseteq R \circ R^T$;
- If R is single-valued, then $R^T \circ R \subseteq \text{id}_Y$.

Remark 6.14. The aforementioned duality can be seen by “reading the relations (arrows) backwards” (Fig. 6.11).

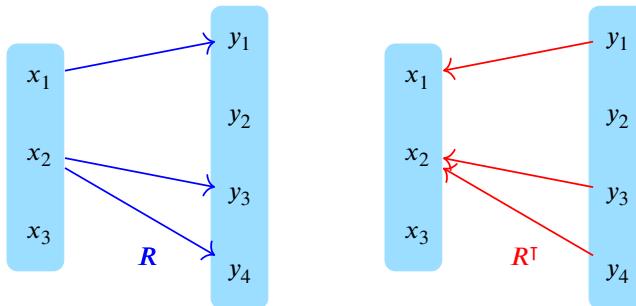


Figure 6.11.

Definition 6.15 (Endorelation). An *endorelation* on a set X is a relation $R \subseteq X \times X$.

Example 6.16. “Equality” is an endorelation of the form

$$\{\langle x_1, x_2 \rangle \in X \times X \mid x_1 = x_2\}.$$

Example 6.17. Take $X = \mathbb{N}$. The relation “less than or equal” is an endorelation of the form

$$\{\langle m, n \rangle \in \mathbb{N} \times \mathbb{N} \mid m \leq n\}.$$

Example 6.18. The relation depicted in Fig. 6.4 is an endorelation between the set of high voltage nodes.

Definition 6.19 (Properties of endorelations). Let $R \subseteq X \times X$ be an endorelation. R is:

- *Symmetric* if $\forall x, x' \in X : \langle x, x' \rangle \in R \Leftrightarrow \langle x', x \rangle \in R$;
- *Reflexive* if $\forall x \in X : \langle x, x \rangle \in R$;
- *Transitive* if $\forall \langle x, x' \rangle \in R$ and $\langle x', x'' \rangle \in R$, we have $\langle x, x'' \rangle \in R$.

Example 6.20. 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 6.21. The relation depicted in Fig. 6.4 is reflexive (each node is connected with itself).

Example 6.22. The endorelation reported in Fig. 6.12 is a symmetric relation on $X = \{x_1, x_2\}$.

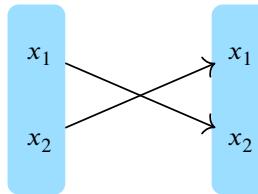


Figure 6.12.: Example of symmetric endorelation.

Definition 6.23 (Equivalence relation). An endorelation $R \subseteq X \times X$ is an *equivalence relation* if it is symmetric, reflexive, and transitive. We write $x \sim x'$ if $\langle x, x' \rangle \in R$.

Example 6.24. The relation “equals” on \mathbb{N} is an equivalence relation. The relation “less than or equal” on \mathbb{N} is not.

Example 6.25. 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 everyone has the same birthday as itself. It 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.

Example 6.26. Let $f : X \rightarrow Y$ be a function between sets. The following defines an equivalence relation:

$$x \sim x' \Leftrightarrow f(x) = f(x').$$

Definition 6.27 (Partition). A *partition* of a set X is a collection $\{X_i\}_{i \in I}$ of subsets $X_i \subseteq X$ such that

1. $X_i \cap X_j = \emptyset \quad \forall i \neq j$;
2. $\bigcup_{i \in I} X_i = X$.

Remark 6.28. Equivalence relations are a way to group together elements of a set which we think of as “the same” in some respect. There is a one-to-one correspondence between equivalence relations on a set X and partitions on X .

1

Example 6.29. An example of partitions can be shown through information networks. An exemplary network is reported in Fig. 6.13. Here, nodes represent data centers, and the arrows represent information flows. We say that data centers x and y are equivalent (i.e., $x \sim y$) if and only if there is a path from x to y and a path from y to x . In this case, we have $a \sim b$, $e \sim d$, and all centers equivalent with themselves.

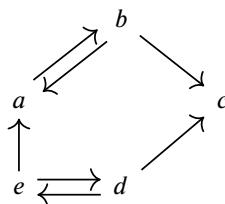


Figure 6.13.

6.3. Relational Databases

A *relational database* like PostgreSQL, MySQL, etc. presents the data to the user as relations. This does not necessarily mean that the data is stored as tuples, as in the mathematical model, but rather that what the user can do is query and manipulate relations. This conceptual model is now 50 years old.

Can we use the category **Rel** to represent databases [11]?

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 motors².

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 6.1.

Motor ID	Company	Size [mm ³]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	20 x 20 x 30	60.0	2.34	19.95
1206	SOYO	28 x 28 x 45	140.0	3.00	19.95
1207	SOYO	35 x 35 x 26	130.0	2.07	12.95
2267	SOYO	42 x 42 x 38	285.0	4.76	16.95
2279	Sanyo Denki	42 x 42 x 31.5	165.0	5.40	164.95
1478	SOYO	56.4 x 56.4 x 76	1,000	8.96	49.95
2299	Sanyo Denki	50 x 50 x 16	150.0	5.90	59.95

Table 6.1.: A simplified catalogue of motors.

²See pololu.com for a standard catalogue of electric stepper motors.

Such database table can be seen as representing an n -ary relation with $n = 7$, as we are expressing a relation over the sets

$$M \times C \times S \times W \times J \times P,$$

where M represents the set of motor IDs, C the set of companies producing motors, S the set of motor sizes, W the set of motor weights, J the set of possible maximal powers, and P the set of possible prices. An n -ary relation is a relation over n sets, just like a binary relation is a relation over 2 sets.

Definition 6.30 (n -ary relation). An n -ary relation on n sets $\langle X_1, X_2, \dots, X_n \rangle$ is a subset of the product set

$$X_1 \times X_2 \times \dots \times X_n.$$

Rel only allows binary relations. Morphisms in **Rel** have 1 source and 1 target. There is no immediate and natural way to represent n -ary relations using **Rel**.

To represent relational databases categorically, there are at least 3 options.

Option 1: Hack it We will introduce the notion of *products* and *isomorphisms*. This will allow us to say that because

$$X_1 \times X_2 \times X_3 \times \dots \times X_n,$$

is isomorphic to

$$X_1 \times (X_2 \times X_3 \dots \times X_n)$$

we can talk about n -ary relations in terms of binary relations. This is not really a natural way to do it.

Option 2: Mutant Morphisms What if morphisms could have more than “two legs”? There are indeed theories that work with more complicated arrows. For example: [multicategories](#), [polycategories](#), [operads](#).

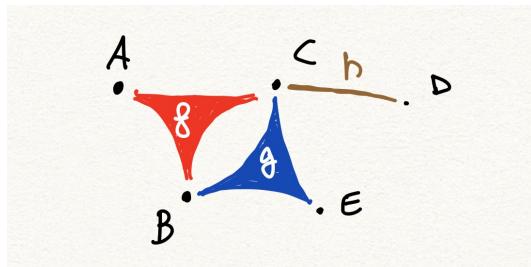


Figure 6.14.: Can you imagine how to define composition with mutant morphisms with more than two legs?

Option 3: Categorical databases A different perspective is that of *categorical databases* [42]. In this modeling framework one does not model the data tables as relations directly. Rather the data is described as a functor from a category representing the schema to **Set**.

7. Computing

to write

Contents

7.1. Databases, sets, functions	52
7.2. The Category Set	54
7.3. Propositions	54



7.1. Databases, sets, functions

In this section, transmuted and transmuter are switched

AC: I'm not sure about this example. I feel that if one is happy with relational databases as a trivial example, they already know Intuitively functions as morphisms (maybe because they know types). If you don't know databases, then that topic cannot be used to introduce a simple concept such as databases.

JL: In the second sentence above, is “that topic” referring to “functions as morphisms”? I’m also not sure if I understand the overall comment. In any case, if it’s helpful: the original idea behind this database example was to do things in a way that is compatible with how databases are treated in Seven Sketches. (I’m not necessarily attached to keeping this example, this is just to explain the idea/intention.) I don’t know anything about databases anyway :) I just thought it probably works well if it is in line with the FQL way of seeing things.

AC: I propose to have here the **Spreadsheet** category: objects are cells, morphisms are the formulas. There is an object 1 that can be the domain for the constant value of cells.

We continue the discussion of Section 6.3.

In the particular case of tables with primary keys, things are easier. In relational databases, a table column is a primary key if the values of that column are guaranteed to be unique. If the values in the column are unique, the column serves as the name of the row. In the table above the motor ID serves as the primary key.

Consider a table with columns $P \times X_1 \times X_2 \times \dots \times X_n$, where P is the primary key column. Then, given a key $p \in P$, we can obtain the value in the other columns. We first find the unique row with the key p , and then we read out the values.

Therefore, a table with columns $P \times X_1 \times X_2 \times \dots \times X_n$ can be seen as a tuple $\langle S, \{f_i\}_i \rangle$:

- A subset $A \subseteq P$ that gives us the available keys.
- n read-out functions $f_i : P \rightarrow X_i$, each giving the corresponding value of the i -th attribute.

JL: I don’t see yet how the set A comes into play.

GZ:here maybe we don’t want color, maybe let’s not use transmuted thing?

In this example, we can consider the primary key to be the set

$$M := \{1204, 1206, 1207, 2267, 2279, 1478, 2299\},$$

of models of motors. The other columns are given by the set

$$C := \{\text{SOYO, Sanyo Denki}\}$$

of manufacturing companies, the set S of possible motor sizes, the set W of possible weights, the set J of possible maximal powers, and the set P of possible prices. Each attribute of a motor may be thought of as a function from the set M to set of possible values for the given attribute. For example, there is a function **Company** : $M \rightarrow C$ which maps each model to the corresponding company that manufactures it. So, according to Table 6.1, we have e.g. **Company(1204) = SOYO**, and **Company(2279) = Sanyo Denki**, etc.

Note that in “real life”, the catalogue of motors might not have seven entries, as in Table 6.1, but has in fact hundreds of entries, and is implemented digitally as a database, i.e. a collection of interrelated tables. In this case, we will want to be able to search and filter the data based on various criteria. Many natural operations on tables and databases may be described simply in terms of operations with functions. We will use this setting as a way to introduce compositional aspects of working with sets and functions, and a preview of how this might be useful for thinking, in particular, about databases.

Sticking with Table 6.1, suppose, for instance, that we want to consider only motors from Company **Sanyo Denki**. In terms of the function

$$\text{Company} : M \rightarrow C$$

this corresponds to the preimage $\text{Company}^{-1}(\{\text{Sanyo Denki}\}) = \{2279, 2299\}$, which is a subset of the set M . Or, we may want to consider only motors which cost between 40 and 200 USD. In terms of the obvious function

$$\text{Price} : M \rightarrow P,$$

this means we wish to restrict ourselves to the preimage

$$\text{Price}^{-1}(\{49.95, 59.95, 164.95\}) = \{1478, 2299, 2279\} \subseteq M.$$

Now suppose we wish to add a column to our table for “volume”, because we may want to only consider motors that have, at most, a certain volume. For this we define a set V of possible volumes (let’s take $V = \mathbb{R}_{\geq 0}$, the non-negative real numbers), and define a function

$$\begin{aligned} \text{Multiply} : S &\rightarrow V \\ \langle l, w, h \rangle &\mapsto l \cdot w \cdot h, \end{aligned}$$

which maps any size of motor to its corresponding volume by multiplying together the given numbers for length, width, and height. Now we can compose this function with the function

$$\text{Size} : M \rightarrow S$$

to obtain a function

$$\text{Volume} : M \rightarrow V,$$

which defines a new column in our table. The composition of functions is usually written as $\text{Volume} = \text{Multiply} \circ \text{Size}$, however we stick to our convention of writing $\text{Volume} = \text{Size} ; \text{Multiply}$. Schematically, we can represent what we did as a diagram (Fig. 7.1).

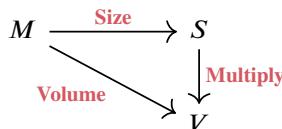


Figure 7.1.: A diagram of functions.

We can interpret arrows in this diagram as being part of a category, where M , S , and V are among the objects, and where the functions **Size**, **Multiply** and **Volume** are morphisms.

We probably want to consider the other sets associated with our database as also part of this category, and the other functions which we defined so far, too. One idea might be to just include all the sets and functions that we've defined so far, as well as all possible compositions of those functions, and obtain a category, which we call **Database**, in a way that is similar to how one can build a category from a graph (Section 5.3). This would be an option. However, we may want soon to add new sets and functions to our database framework, or think about new kinds of functions between them that we had not considered before. And we might not want to re-think each time precisely which category we are working with.

7.2. The Category Set

A helpful concept here is to think of our specific sets and functions as living in a very (very) large category which contains all possible sets as its objects and all possible functions as its morphisms. This category is known as the category of sets, and it is an important protagonist in category theory. We will denote it by **Set**. It is a short exercise to check that the following does indeed define a category.

Definition 7.1 (Category of sets). The category of sets **Set** is defined by:

1. *Objects*: all sets.
2. *Morphisms*: given sets X and Y , the homset $\text{Hom}_{\text{Set}}(X, Y)$ is the set of all functions from X to Y .
3. *Identity morphism*: given a set X , its identity morphism id_X is the identity function $X \rightarrow X$, $\text{id}_X(x) = x$.
4. *Composition operation*: the composition operation is the usual composition of functions.

We did say above, however, that we could build a category **Database** which only involves the sets that we are using for our database, and the functions between them that we are working with. What we would need for **Database** to be a category is that if any function is in **Database**, then also its sources and target sets are, and we would need that any composition of functions in **Database** is again in **Database**. (Also, we define the identity morphism for any set in **Database** to be the identity function on that set.) If these conditions are met, **Database** is what is called a *subcategory* of **Set**.

7.3. Propositions

Define the category of propositions where objects are propositions and morphisms are proofs. Also use to introduce the sequent notations

8. Specialization

to write

Contents

8.1. Notion of subcategory	56
8.2. Drawings	56
8.3. Other examples of subcategories in engineering	57
8.4. Subcategories of Berg	59



8.1. Notion of subcategory

Definition 8.1 (Subcategory). A *subcategory* **D** of a category **C** is a category for which:

1. All the objects in $\text{Ob}_{\mathbf{D}}$ are in $\text{Ob}_{\mathbf{C}}$;
2. For any objects $X, Y \in \text{Ob}_{\mathbf{D}}$, $\text{Hom}_{\mathbf{D}}(X, Y) \subseteq \text{Hom}_{\mathbf{C}}(X, Y)$;
3. If $X \in \text{Ob}_{\mathbf{D}}$, then $\text{id}_X \in \text{Hom}_{\mathbf{C}}(X, X)$ is in $\text{Hom}_{\mathbf{D}}(X, X)$ and acts as its identity morphism;
4. If $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ in **D**, then the composite $f \circ g$ in **C** is in **D** and represents the composite in **D**.

Two important examples of subcategory are the following.

Example 8.2 (Finite Sets). **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 $X \in \text{Ob}_{\mathbf{Set}}$ is a set with arbitrary cardinality, $\text{Ob}_{\mathbf{FinSet}}$ only includes sets which have finitely many elements. Objects of **FinSet** are in **Set**, but the converse is not true. Furthermore, given $X, Y \in \text{Ob}_{\mathbf{FinSet}}$, we take $\text{Hom}_{\mathbf{FinSet}}(X, Y) = \text{Hom}_{\mathbf{Set}}(X, Y)$.

Example 8.3 (**Set** and **Rel**). The category **Set** is a subcategory of **Rel**. To show this, we need to prove the conditions presented in Definition 8.1.

1. In both **Rel** and **Set**, the collection of objects is all sets.
2. Given $X, Y \in \text{Ob}_{\mathbf{Set}}$, we know that $\text{Hom}_{\mathbf{Set}}(X, Y) \subseteq \text{Hom}_{\mathbf{Rel}}(X, Y)$, i.e., that all functions between sets X, Y are a particular subset of all relations between X, Y .
3. For each $X \in \text{Ob}_{\mathbf{Set}}$, 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**.
4. 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 $R \circ S \subseteq X \times Z$, is again a function. This was proven in Lemma 6.7.

8.2. Drawings

Definition 8.4 (Drawings). There exists a category **Draw** in which:

1. An object in $\alpha \in \text{Ob}_{\mathbf{Draw}}$ is a black-and-white drawing, that is a function $\alpha : \mathbb{R}^2 \rightarrow \mathbf{Bool}$.
2. A morphism in $\text{Hom}_{\mathbf{Draw}}(\alpha, \beta)$ 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 8.5. Check whether just considering

- affine invertible transformations, or
- rototranslations, or
- scalings, or

- translations, or
- rotations,

as morphisms forms a subcategory of **Draw**.

Add a few figures here.

We can now think about the different types of transformations.

- **Scalings.** Let $s, t \in \mathbb{R}$. Scalings can be represented as functions of the form

$$\begin{aligned} f_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle sx, ty \rangle, \end{aligned}$$

- **Translations.** Let $s, t \in \mathbb{R}$. Translations are functions of the form

$$\begin{aligned} f_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle x + s, y + t \rangle. \end{aligned}$$

- **Rotations.** Let $\theta \in [0, 2\pi)$. Rotations are functions of the form

$$\begin{aligned} f_\theta : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle x \cos(\theta) + y \sin(\theta), y \cos(\theta) - x \sin(\theta) \rangle. \end{aligned}$$

- ...

Finish the above, show which ones are subcategories, etc.

8.3. 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 forms a subcategory of Set

Definition 8.6 (Injective function). Let $f : X \rightarrow Y$ be a function. The function f is *injective* if, for all $x, x' \in X$ holds: $f(x) = f(x') \implies x = x'$.

Example 8.7. We can define a category **InjSet** which has the same objects as **Set** but restricts the morphisms to be *injective functions*. 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 Definition 8.1 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 $\text{Hom}_{\text{InjSet}}(X, X)$, i.e., the identity function is injective. This proves the third condition. To check the fourth 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 $x, x' \in X$, $f(x) = f(x') \Leftrightarrow x = x'$ and $y, y' \in$

$X, g(y) = g(y') \Leftrightarrow y = y'$. Furthermore, we have:

$$\begin{aligned}(f \circ g)(x) &= (f \circ g)(x') \implies f(x) = f(x') \\ &\implies x = x',\end{aligned}$$

which proves the fourth condition of Definition 8.1, i.e. that the composition of injective functions is injective.

Definition 8.8 (Continuous functions). Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a function. We call f *continuous* at $c \in \mathbb{R}$ if $\lim_{x \rightarrow c} f(x) = f(c)$; f is continuous over \mathbb{R} if the condition is satisfied for all $c \in \mathbb{R}$.

Example 8.9. We can define a category **Cont** which $\text{Ob}_{\text{Cont}} = \mathbb{R}$ and in which the morphisms are given by continuous functions. Composition and identity are as in **Set**. We want to show that **Cont** is a subcategory of **Set**.

write down formally and use that composition of continuous is continuous

Continuous functions (topologies)

Differentiable functions: Set to Manifolds

Definition 8.10 (Differentiable functions). A function $f : U \subset \mathbb{R} \rightarrow \mathbb{R}$, defined on an open set U , is *differentiable* at $a \in U$ if the derivative

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} \quad (8.1)$$

exists; f is differentiable on U if it is differentiable at every point of U .

Example 8.11. the composition of differentiable functions is differentiable

Lipschitz bounded

Definition 8.12 (Lipschitz continuous function). A real valued function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called *Lipschitz* continuous if there exists a positive real constant κ such that, for all $x_1, x_2 \in \mathbb{R}$:

$$|f(x_1) - f(x_2)| \leq \kappa|x_1 - x_2|. \quad (8.2)$$

Example 8.13. the composition of differentiable functions is differentiable

smooth

cont diff, composition is compdiff

Generalization outside of R

Generalization to more general spaces. We used the fact that R is:

- For defining continuous functions, we used the fact that R is topological space (minimum needed for defining a continuous function.). In fact, the real definitoin of continuous function is:

ADD: continuous function

- To define Lipschitz we needed the fact that R is a metric space
- For differentiable, smooth, you define this on Manifolds. Exists tangent space.

Hence in general, the objects of these are different, so it's not really a relation of subcategory, that requires the objects to be the same. However we will see later that you can generalize this notion using functors.

functor $F:C \rightarrow D$ that is both injective on objects and a faithful functor.

8.4. Subcategories of Berg

Recall the category **Berg** presented in Section 5.2. In the following, we want to give both a positive and a negative example of subcategories related to **Berg**.

We first start our discussion by introducing an *amateur* version of **Berg**, called **BergAma**, which only considers paths (morphisms) in **Berg**, whose steepness does not exceed a critical value, say 1/2. Is **BergAma** a subcategory of **Berg**? Let's check the different conditions:

1. The constraint on the maximum steepness restricts the objects which are acceptable in **BergAma** 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, **BergAma** will only contain objects whose identity morphisms do not exceed the steepness constraint, i.e. $\text{Ob}_{\text{BergAma}} \subseteq \text{Ob}_{\text{Berg}}$.
2. For $A, B \in \text{Ob}_{\text{BergAma}}$, we know that paths satisfying the steepness constraint are specific paths in **Berg**, i.e. $\text{Hom}_{\text{BergAma}} \subseteq \text{Hom}_{\text{Berg}}$.
3. The identity morphisms in **Berg** which satisfy the steepness constraint are, by definition, in **BergAma** and they act as identities there.
4. Given two morphisms f, g which can be composed in **BergAma**, the maximum steepness of their composition $f ; g$ is given by:

$$\text{MaxSteepness}(f ; g) = \max \{ \text{MaxSteepness}(f), \text{MaxSteepness}(g) \} < 1/2.$$

This shows that **BergAma** is a subcategory of **Berg**. What would an example of non-subcategory of **Berg** be? Let's define 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 fourth property of Definition 8.1.

9. Sameness

to write

Contents

9.1. Sameness in category theory	62
9.2. Isomorphism is not identity	64



9.1. 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 $f : X \rightarrow Y$ of sets to be bijective:

Definition 1: “ $f : X \rightarrow Y$ is bijective if, for every $y \in Y$ there exists precisely one $x \in X$ such that $f(x) = y$;

Definition 2: “ $f : X \rightarrow Y$ is bijective if there exists a function $g : Y \rightarrow X$ such that $f \circ g = \text{id}_X$ and $g \circ f = \text{id}_Y$ ”.

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 X and Y . 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 9.1 (Isomorphism). Let \mathbf{C} be a category, let $X, Y \in \mathbf{C}$ 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 9.2. 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 9.3. In Remark 9.2 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 $g_1 = g_2$.

Definition 9.4 (Isomorphic). Let \mathbf{C} be a category, and let $X, Y \in \mathbf{C}$ be objects. 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 9.2 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, let’s look at some examples using sets.

Example 9.5 (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 $f : M \rightarrow \{\text{Si, No}\}$ which maps a given dish $m \in M$ to the statement “Si” (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

$$\{m \in M \mid f(m) = \text{Si} \quad \text{and} \quad g(m) = \text{Ja}\}.$$

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 (“Si” and “Ja”) to “ \heartsuit ”, and their respective negative answers to “ \clubsuit ”. This defines isomorphisms

$$\{\text{Si, No}\} \longleftrightarrow \{\heartsuit, \clubsuit\} \longleftrightarrow \{\text{Ja, Nein}\}$$

whose compositions provide a translation between the Italian and German two-element sets. Using these isomorphisms, we obtain, by composition, new functions

$$\tilde{f} : M \longrightarrow \{\heartsuit, \clubsuit\}, \quad \tilde{g} : M \longrightarrow \{\heartsuit, \clubsuit\},$$

and the set of dishes that Francesca and Gabriel would be willing to share can be written as

$$\{m \in M \mid \tilde{f}(m) = \heartsuit \quad \text{and} \quad \tilde{g}(m) = \heartsuit\}.$$

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

$$\{0, 1\}, \{\text{true, false}\}, \{\perp, \top\}, \{\text{left, right}\}, \{-, +\}, \text{etc.}$$

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, i.e. how they are being used in interaction with other mathematical constructs.

Example 9.6 (Sizes). Suppose we are a manufacturer and we are counting how many wheels are in a certain warehouse. If W denotes the set of wheels that we have, then counting can be modelled as a function $f : W \rightarrow \mathbb{N}$ to the natural numbers. If we find

that there are, say, 273 wheels, then our counting procedure gives us a bijective function from W to the set $\{1, 2, 3, \dots, 272, 273\}$. In this case, we don't care which specific wheel we counted first, second, or last. We could just as well have counted in a different order, which would amount to a different function $f' : W \rightarrow \mathbb{N}$. The only thing we care about is the fact that the sets W and $\{1, 2, 3, \dots, 272, 273\}$ are *isomorphic*; we don't need to keep track of which counting isomorphism exhibits this fact.

Example 9.7 (Relabelling). Consider the little catalogue in Table 6.1. 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, 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

$$\text{Company} : M \rightarrow C.$$

The updated version of the “Company” label, using the new set N of model IDs, is obtained by the composition

$$N \xrightarrow{\text{relabel}} M \xrightarrow{\text{Company}} C.$$

Example 9.8. Going back to currency exchangers, recall that any currency exchanger $E_{a,b}$, given by

$$\begin{aligned} E_{a,b} : \mathbb{R} \times \{\text{USD}\} &\rightarrow \mathbb{R} \times \{\text{EUR}\} \\ \langle x, \text{USD} \rangle &\mapsto \langle ax - b, \text{EUR} \rangle \end{aligned}$$

is an isomorphism, since one can define a currency exchanger $E_{a',b'}$ such that

$$E_{a,b} \circ E_{a',b'} = E_{a',b'} \circ E_{a,b} = E_{1,0}.$$

Example 9.9. 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 (i.e., the set has n elements), the number of isomorphisms is given by the number of ways in which one can “rearrange” n elements of the set, which is $n!$.

Example 9.10. In **Set**, isomorphisms between $\mathbb{R} \rightarrow \mathbb{R}$ correspond to invertible functions.

9.2. Isomorphism is not identity

Example 9.11. Let's consider currencies, and in particular the sets $\mathbb{R} \times \{\text{USD}\}$ and $\mathbb{R} \times \{\text{USD cents}\}$. 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, 10 **USD** and 1,000 **USD cents** are different elements of different sets, but there exists an isomorphism between the two. For one direction, the isomorphism transforms **USD** into **USD cents** (multiplying the real number by 100); the other direction transforms **USD cents** into **USD** (dividing the real number by 100).

Invertible functions are isomorphisms

Strictly monotone functions are invertible from \mathbb{R} to \mathbb{R} ?

isomorphisms in 2D

isomorphisms are permutations

Examples from before: Dynamical systems (open, $d=0$). Objects are sets, morphisms given by state update and readout. Can go back and forth. (category of processes in computer science)

10. Universal properties

to write

Contents

10.1. Universal properties	68
--------------------------------------	----



move universal properties somewhere else

10.1. Universal properties

JL: I think this might be a good place to introduce universal properties via initial and terminal objects (and in particular using the examples of Set and Rel)

Definition 10.1 (Initial and terminal object). Let \mathbf{C} be a category and let $A \in \mathbf{C}$ be an object. We say that A is an *initial object* if, for all $B \in \mathbf{C}$, the hom-set $\text{Hom}_{\mathbf{C}}(A, B)$ has exactly one element. We say that A is a *terminal object* if, for all $D \in \mathbf{C}$, the hom-set $\text{Hom}_{\mathbf{C}}(D, A)$ has exactly one element.

11. Trade-offs

to write

Contents

11.1. Trade-offs	70
11.2. Ordered sets	70
11.3. Chains and Antichains	74
11.4. Upper and lower sets	75
11.5. From antichains to uppersets, and viceversa	77
11.6. Lattices	80



11.1. Trade-offs

Add the examples from session 6:

Trade-offs characterize all engineering disciplines, and can be literally found everywhere. A typical trade-off is the one reported in Fig. 11.1. When designing a product, you want it to be *good*, *fast*, and *cheap*, and typically you can just choose between two of these qualities.

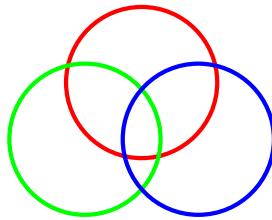


Figure 11.1.

finish example

Functionality, requirements, 3 types of achievable accuracy plots

example: Trade-offs for the human body

example: Masks

example: Hats and headphones

11.2. Ordered sets

So far, the discussion has been purely qualitative. While we discussed how categories can describe the way in which one resource can be turned into another, this kind of modelling did not allow for quantitative statements. For example, it is good to know that we can obtain motion from electric power, but, how fast can we go with a certain amount of power?

To achieve a quantitative theory, we need to specify various degrees of resources and functionality. One way of doing this, is through the idea of partial orders.

Such orderings arise naturally in engineering as criteria for judging whether one design is better or worse than another. As an example, suppose you need to prepare some pizza, i.e., you have to buy specific ingredients and cook them, using a recipe you decide to follow. In this simple example, you can think of having two resources: time and money. A quicker recipe might include more expensive ingredients, and a slower recipe could feature more affordable ones. How to choose among recipes, if you do not prefer one resource over the other? How to model this? In this section, we will assume that functionality and resources are *partially-ordered sets (posets)*.

Davey and Priestley [15] and Roman [39] are possible reference texts.

Introduce at the same time pre-order, poset, total order, like in slides.

Remark: add remark about more general preference functions (Semiorder etc.)

Definition 11.1 (Partially ordered set). A *partially-ordered set (poset)* is a tuple $\langle P, \leq \rangle$, where P is a set (also called the *carrier set*), together with a relation \leq on P that is

1. *Reflexive*: For all $x \in P$, $x \leq x$.
2. *Antisymmetric*: For all $x, y \in P$, if $x \leq y$ and $y \leq x$, then $x = y$.
3. *Transitive*: For all $x, y, z \in P$, if $x \leq y$ and $y \leq z$, then $x \leq z$.

A *Hasse diagram* is an economical (in terms of arrows) way to visualize a poset. In a Hasse diagram elements are points, and if $x \leq y$ then x is drawn lower than y and with an edge connected to it, if no other point is in between. Hasse diagrams are directed graphs.

In the example of the pizza recipes, both time and money can be thought of as partially ordered sets $\langle \mathbb{R}_{\geq 0}, \leq \rangle$. Imagine that you have recipes costing 1 **USD**, 2 **USD**, and 3 **USD**. This can be represented as in Fig. 11.2.



Figure 11.2.: The cost of pizza ingredients can be represented as a poset.

Example 11.2. Consider a poset $P = \{a, b, c, d, e\}$ with $a \leq b$, $a \leq c$, $d \leq c$, and $d \leq e$. This can be represented with a Hasse diagram as in Fig. 11.3.

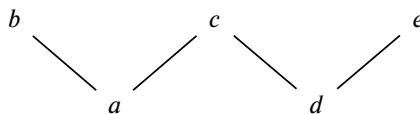


Figure 11.3.: Example of Hasse diagram of P .

Example 11.3 (Singleton poset). If a set has only one element, say $\{1\}$, then there is a unique order relation on it (Fig. 11.4). We denote the resulting poset again by $\{1\}$.



Figure 11.4.: The singleton poset.



Figure 11.5.: All posets on 2-elements sets, up to isomorphisms.

Example 11.4. In this example, we represent all posets up to isomorphisms on up to 4 elements. For one element, one has only the singleton poset (Fig. 11.4). On 2-elements sets, one has the posets reported in Fig. 11.5. On 3-elements sets, one has the posets reported in Fig. 11.6. On 4-elements sets, one has the posets reported in Fig. 11.7.

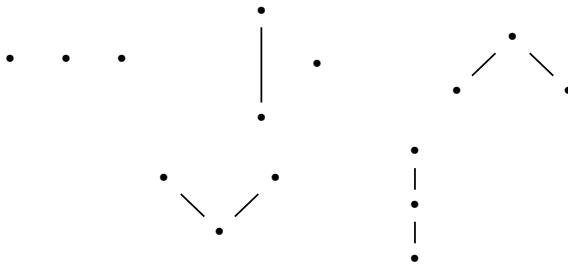


Figure 11.6.: All posets on 3-elements sets, up to isomorphisms.

Example 11.5 (Booleans). The booleans is a poset with carrier set $\{\top, \perp\}$ and the order relation given by $b_1 \leq_{\text{Bool}} b_2$ iff $b_1 \Rightarrow b_2$, that is, $\perp \leq_{\text{Bool}} \top$ (Fig. 11.8).

This relation should be familiar from Table 11.1.

a	b	$a \leq_{\text{Bool}} b$	$a \wedge b$	$a \vee b$
\top	\top	\top	\top	\top
\top	\perp	\perp	\perp	\top
\perp	\top	\top	\perp	\top
\perp	\perp	\top	\perp	\perp

Table 11.1.: Properties of the **Bool** poset.

In addition to the operation

$$\Rightarrow : \text{Bool} \times \text{Bool} \rightarrow \text{Bool},$$

called *implies*, 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.

Example 11.6 (Reals). The real numbers \mathbb{R} form a poset with carrier \mathbb{R} and order relation given by the usual ordering $r_1 \leq r_2$.

Example 11.7 (Discrete partially ordered sets). Every set X can be considered as a *discrete poset* $\langle X, = \rangle$. Discrete posets are represented as collection of points (Fig. 11.9).

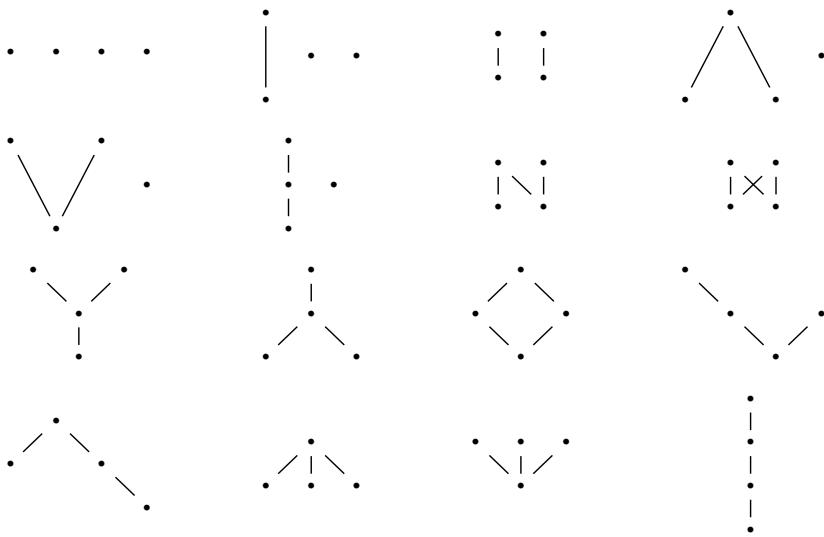


Figure 11.7.: All posets on 4-elements sets, up to isomorphisms.

$$\begin{array}{c} \top \\ | \leq \\ \perp \end{array}$$

Figure 11.8.

Example 11.8. Given a set $X = \{a, b, c\}$, consider its power set $\mathcal{P}(X)$. Define sets as the objects of this new category and define the morphisms to be inclusions (Fig. 11.10).

The identity morphism of each set is the inclusion with itself (every set is a subset of itself). Composition is given by composition of inclusions, i.e., if $X \subseteq Y \subseteq Z$, then $X \subseteq Z$.

A note on preorders

Definition 11.9 (Preorder). A *preorder* is a tuple $\langle P, \leq \rangle$, where P is a set (also called the *carrier set*), together with a relation \leq on P that is

1. *Reflexive*: For all $x \in P$, $x \leq x$.
2. *Transitive*: For all $x, y, z \in P$, if $x \leq y$ and $y \leq z$, then $x \leq z$.

The theory of design problems can be easily generalized to preorders. This means that

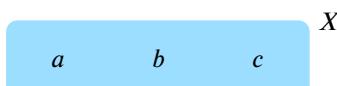


Figure 11.9.: Example of a discrete poset.

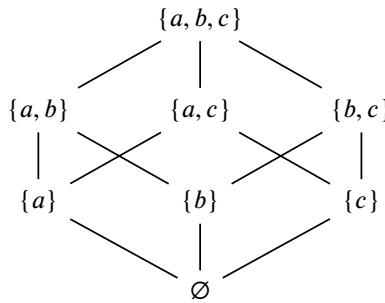
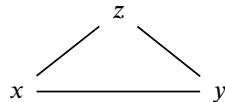


Figure 11.10.: Power set as a category.

there could be two elements x and y such that $x \leq y$ and $x \geq y$ but $x \neq y$ (Fig. 11.11).



Let's not use Hasse for preorders, it's not a thing.

Figure 11.11.: Example of a preorder.

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 preorder, computation will always involve passing to the poset representation.

This means that, given a preorder, we can consider the poset of its isomorphism classes, by means of the following equivalence relation:

$$x \simeq y \equiv (x \leq y) \wedge (y \leq x). \quad (11.1)$$

11.3. Chains and Antichains

This part is partially repeated from above. Do chains, antichains, upper/lower sets before "poset as a category".

Definition 11.10 (Chain in a poset). Given a poset S , a *chain* is a sequence of elements s_i in S where two successive elements are comparable, i.e.:

$$i \leq j \Rightarrow s_i \leq s_j. \quad (11.2)$$

Definition 11.11 (Antichain in a poset). An *antichain* is a subset S of a poset where no elements are comparable. If $a, b \in S$, then $a \leq b$ implies $a = b$.

Remark 11.12. We denote the set of antichains of a poset P by \mathcal{AP} .

Remark 11.13. Note that given a poset $\langle P, \leq \rangle$, $\emptyset \in \mathcal{AP}$ since \emptyset contains no elements (and hence no comparable elements).

In the context of pizza recipes, consider the diagram reported in Fig. 11.12. The blue points represent an antichain of recipes $\{\langle 1 \text{ USD}, 2 \text{ h} \rangle, \langle 2 \text{ USD}, 1 \text{ h} \rangle\}$, i.e. recipes which do not dominate each other (one is cheaper but takes longer and the other is more expensive but quicker). The red point represents a recipe which cannot be part of the antichain, since it is dominated by $\langle 2 \text{ USD}, 1 \text{ h} \rangle$.

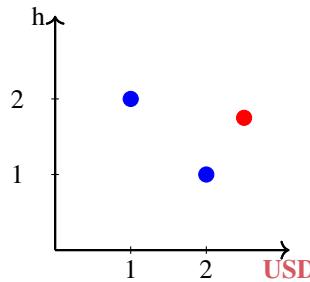


Figure 11.12.: Example of an antichain of pizza recipes.

Example 11.14. Let's consider the poset $\langle P, \leq \rangle$ where $a \leq b$ if a is a divisor of b and $P = \{1, 5, 10, 11, 13, 15\}$. A chain of P is $\{1, 5, 10, 15\}$. An antichain of P is $\{10, 11, 13\}$.

Example 11.15. Consider Example 11.8. Examples of chains are

$$\{\emptyset, \{a\}, \{a, b\}, \{a, b, c\}\}, \quad \{\emptyset, \{b\}, \{b, c\}, \{a, b, c\}\}. \quad (11.3)$$

Examples of antichains are

$$\{\{a\}, \{b\}, \{c\}\}, \quad \{\{a, b\}, \{a, c\}, \{b, c\}\}. \quad (11.4)$$

Example 11.16. Suppose you have to choose a battery model based on its cost and its weight, both to be minimized. There may be models which dominate others. For instance, a model $\langle 10 \text{ USD}, 1 \text{ kg} \rangle$ is better than a model $\langle 11 \text{ USD}, 1.1 \text{ kg} \rangle$. Also, there may be models which are incomparable, i.e. which form an antichain. For example, you cannot say whether $\langle 10 \text{ USD}, 1 \text{ kg} \rangle$ is better than $\langle 5 \text{ USD}, 2 \text{ kg} \rangle$. The incomparable models form an antichain.

11.4. Upper and lower sets

Definition 11.17 (Upper set). An *upper set* is a subset U of a poset P such that, if an element is inside, all elements above it are inside as well. In formulas:

$$U \text{ is an upperset} \equiv \forall x \in U, \forall y \in P : x \leq y \Rightarrow y \in U. \quad (11.5)$$

Remark 11.18. We call UP the set of upper sets of P .

Definition 11.19 (Lower set). A *lower set* is a subset L of a poset P if, if a point is inside, all points below it are inside as well. In formulas:

$$L \text{ is a lower set} \equiv \forall x \in L, \forall y \in P : y \leq x \Rightarrow y \in L. \quad (11.6)$$

Remark 11.20. We call LP the set of lower sets of P .

Remark 11.21. Note that if A is an antichain of a poset P , then the set

$$I(A) = \{x : x \leq y, y \in A\} \quad (11.7)$$

is a lower set of P .

Consider the blue poset of pizza recipes from before. The upper and lower sets of this poset can be represented as in Fig. 11.13. The upper set can be interpreted as all the potential pizza recipes for which we can find better alternatives in the poset. Similarly, the lower set can be interpreted as all the potential pizza recipes which would be better than the ones in the poset.

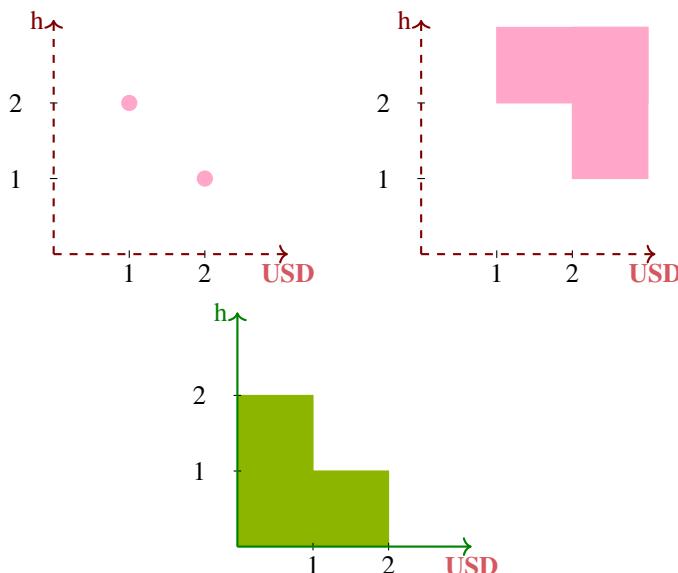


Figure 11.13.: Example of upper and lower sets of a poset of pizza recipes.

Example 11.22 (Upper and lower sets in **Bool**). The booleans $\{\perp, \top\}$ form a poset with $\perp \leq \top : (\text{Bool}, \leq)$. The subset $\{\perp\} \subseteq \text{Bool}$ is not an upper set, since $\perp \leq \top$ and $\top \notin \{\perp\}$.

11.5. From antichains to uppersets, and viceversa

Definition 11.23 (Upper closure operator). The *upper closure operator* \uparrow maps a subset to the smallest upper set that includes it, i.e.:

$$\begin{aligned}\uparrow : \mathcal{P}(P) &\rightarrow \text{UP} \\ S &\mapsto \{y \in P \mid \exists x \in S : x \leq y\}.\end{aligned}\tag{11.8}$$

Remark 11.24. Note that, by definition, an upper set is closed to upper closure.

Remark 11.25. For any $S \in \mathcal{P}(P)$, $\uparrow S$ is in fact an upper set.

Proof. Suppose $y \in \uparrow S$ and $z \in P$, and suppose $y \leq z$. By definition $\exists x$ s.t. $x \leq y$, meaning that $x \leq z$. Thus, $z \in \uparrow S$, as was to be shown. \square

Lemma 11.26. The upper closure operator \uparrow is a monotone map.

Proof. Consider the posets $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle \text{UP}, \supseteq \rangle$, and $S_1, S_2 \in \mathcal{P}(P)$. It is clear that given $S_1 \subseteq S_2$, one has

$$\{y \in P \mid \exists x \in S_1 : x \leq y\} \supseteq \{y \in P \mid \exists x \in S_2 : x \leq y\}.$$

Therefore, $\uparrow S_1 \supseteq \uparrow S_2$, satisfying the monotonicity property for \uparrow . \square

Lemma 11.27. Let A and B be subsets of P that are antichains. Then

$$\uparrow A = \uparrow B \quad \Rightarrow \quad A = B.$$

Proof. First, let's fix an $a \in A$. From $\uparrow A = \uparrow B$ we know that in particular $A \subseteq \uparrow B$. This means that for our fixed $a \in A$ there exists $b \in B$ such that $b \leq a$. From $\uparrow A = \uparrow B$ it also follows that $B \subseteq \uparrow A$, so to the $b \in B$ given above, there exists $a' \in A$ such that $a' \leq b$. In total, we have $a' \leq b \leq a$, and since A is an antichain, we must have $a' = a$. This implies that $a' = b = a$. In particular, we have $a \in B$.

The above shows that $A \subseteq B$. To show $B \subseteq A$, we can fix any $b \in B$ and repeat the above argumentation, now with the roles of A and B exchanged. \square

In the example of the pizza recipes, first, consider the upper set of a single element of the poset, e.g. $p_1 = \langle 1 \text{ USD}, 2 \text{ h} \rangle$ (Fig. 11.14). Then, consider the case of two elements, with $p_2 = \langle 2 \text{ USD}, 1 \text{ h} \rangle$ (Fig. 11.15).

Note that the upper set of the subset formed by the two elements is the union of the upper sets of the single elements.

Definition 11.28 (Lower closure operator). The *lower closure operator* \downarrow maps a subset to the smallest lower set that includes it, i.e.

$$\begin{aligned}\downarrow : \mathcal{P}(P) &\rightarrow \text{LP} \\ S &\mapsto \{y \in P \mid \exists x \in S : y \leq x\}.\end{aligned}$$

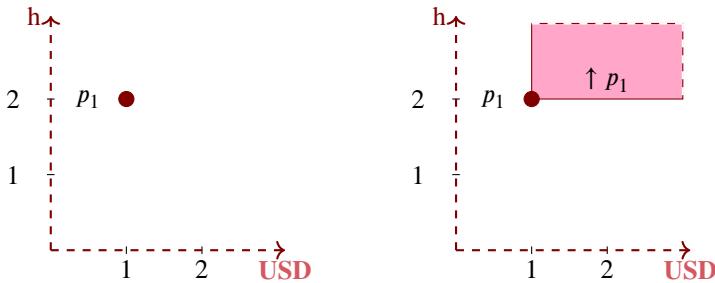


Figure 11.14.: The upper closure of a singleton set of pizza recipes.

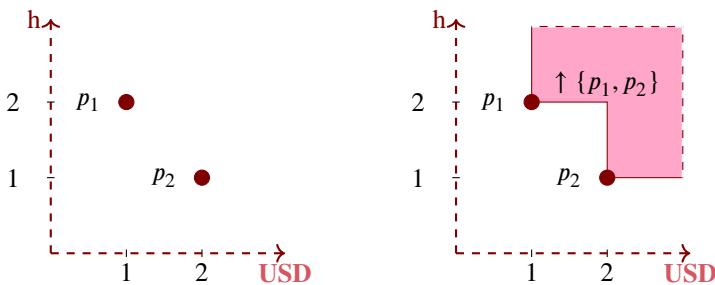


Figure 11.15.: The upper closure of a set of pizza recipes.

Lemma 11.29. The lower closure operator \downarrow is a monotone map.

JL: The following proof is a bit redundant... we can say “analogous to the case of the upper closure operation” and/or invoke the principle of duality for posets.

Proof. Consider the posets $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle \mathcal{L}P, \subseteq \rangle$, and let $S_1, S_2 \in \mathcal{P}(P)$. It is clear that given $S_1 \subseteq S_2$, one has

$$\{y \in P \mid \exists x \in S_1 : y \leq x\} \subseteq \{y \in P \mid \exists x \in S_2 : y \leq x\}. \quad (11.9)$$

Therefore, $\uparrow S_1 \subseteq \uparrow S_2$, satisfying the monotonicity property for \downarrow . \square

Example 11.30. Consider the battery example of Example 11.16, and the antichain given by the battery models $a = \langle 10 \text{ USD}, 1 \text{ kg} \rangle$, $b = \langle 20 \text{ USD}, 0.5 \text{ kg} \rangle$, and $c = \langle 30 \text{ USD}, 0.25 \text{ kg} \rangle$ (Fig. 11.16). The lower closure operator $\downarrow \{a, b, c\}$ represents all the battery models which, if existing, would dominate $\{a, b, c\}$.

Definition 11.31 (Min). $\text{Min} : \mathcal{P}(P) \rightarrow \mathcal{AP}$ is the map that sends a subset S of a poset to the minimal elements of that subset, i.e., those elements $a \in S$ such that $a \leq b$ for all $b \in S$. In formulas:

$$\text{Min} : \mathcal{P}(P) \rightarrow \mathcal{AP}$$

$$S \mapsto \{x \in S : (y \in S) \wedge (y \leq x) \Rightarrow (x = y)\}.$$

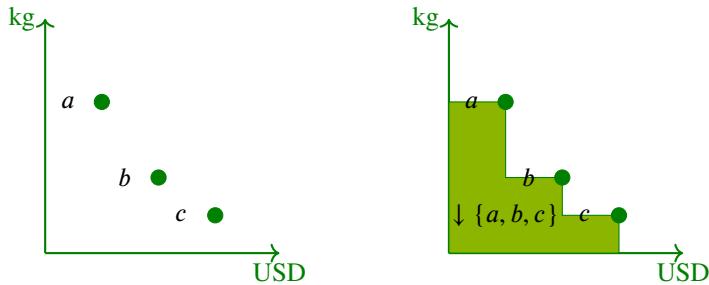


Figure 11.16.: Battery example. From the left: antichain, upper closure, and lower closure.

Note that $\text{Min}(S)$ could be empty.

Definition 11.32 (Max). $\text{Max} : \mathcal{P}(P) \rightarrow \mathcal{AP}$ is the map that sends a subset S of a poset to the maximal elements of that subset, i.e., those elements $a \in S$ such that $a \succeq b$ for all $b \in S$. In formulas:

$$\text{Max} : \mathcal{P}(P) \rightarrow \mathcal{AP}$$

$$S \mapsto \{x \in S : (y \in S) \wedge (y \succeq x) \Rightarrow (x = y)\}.$$

Note that $\text{Max}(S)$ could be empty.

This is a remnant of older times. To remove.

Lemma 11.33. Given a poset $\langle P, \leq \rangle$, $\langle \mathcal{AP}, \leq_{\mathcal{AP}} \rangle$ is a poset with

$$A \leq_{\mathcal{AP}} B \text{ if and only if } \uparrow A \supseteq \uparrow B. \quad (11.10)$$

Furthermore, it is bounded by the top $\top_{\mathcal{AP}} = \emptyset$ and the bottom $\perp_{\mathcal{AP}} = \{\perp_P\}$.

Proof. We need to show the poset properties (Definition 11.1). We can prove the following:

- *Reflexivity:* From $\langle P, \leq \rangle$ being a poset we know that

$$\begin{aligned} \{y \in P \mid \exists x \in A : x \leq y\} &\supseteq \{y \in P \mid \exists x \in A : x \leq y\}, \\ \uparrow A &= \uparrow A \end{aligned} \quad (11.11)$$

and hence $A \leq_{\mathcal{AP}} A$.

- *Antisymmetry:* One has

$$\begin{aligned} (A \leq_{\mathcal{AP}} B) \wedge (B \leq_{\mathcal{AP}} A) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow A) \\ &\Leftrightarrow \uparrow A = \uparrow B \\ &\Rightarrow A = B. \end{aligned} \quad (11.12)$$

The last implication is by ??.

- *Transitivity:* One has

$$\begin{aligned} (A \leq_{AP} B) \wedge (B \leq_{AP} C) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow C) \\ &\Rightarrow \uparrow A \supseteq \uparrow C \\ &\Rightarrow A \leq_{AP} C. \end{aligned} \tag{11.13}$$

In order to find the top, we need to find the smallest set T_{AP} such that $A \leq_{AP} T_{AP}$ for all $A \in AP$. In other words, such that $\uparrow A \supseteq \uparrow T_{AP}$ for all $A \in AP$. This is clearly \emptyset , since $\uparrow \emptyset = \emptyset$. Similarly, in order to find the bottom, we need to find the set \perp_{AP} such that $\perp_{AP} \leq_{AP} A$ for all $A \in AP$. In other words, such that $\uparrow \perp_{AP} \supseteq \uparrow A$ for all $A \in AP$. We obtain a bottom if we set $\perp_{AP} := T_P$, since $T_P \supseteq A$ for all $A \subseteq P$, and hence, by monotonicity of \uparrow , we have in particular $\uparrow T_P \supseteq \uparrow A$ for all antichains A .

□

Definition 11.34 (Downward closed set). An upper set S is *downward-closed* in a poset P if

$$S = \uparrow \text{Min } S. \tag{11.14}$$

Remark 11.35. The set of downward-closed upper sets of P is denoted \underline{UP} .

11.6. Lattices

Definition 11.36 (Lattice). A *lattice* is a poset $\langle P, \leq \rangle$ with some additional properties:

1. Given two points $p, q \in P$, it is always possible to define their least upper bound, called *join*, and indicated as $p \vee q$.
2. Given two points $p, q \in P$, it is always possible to define their greatest lower bound, called *meet*, and indicated as $p \wedge q$.

Add the wooden lattice figure here.

These can be defined for preorders as well... move before.

Remark 11.37 (Bounded lattices). If there is a least upper bound for the entire lattice A , it is called the *top* (\top). If a greatest lower bound exists it is called the *bottom* (\perp). If both a top and a bottom exist, we call the lattice *bounded*, and denote it by $\langle A, \leq, \vee, \wedge, \perp, \top \rangle$.

Example 11.38. In Example 11.8 we presented the poset arising from the power set $\mathcal{P}(A)$ of a set A and ordered via subset inclusion. This is a lattice, bounded by A and by the empty set \emptyset . Note that this lattice possesses two (dual) monoidal structures $\langle \mathcal{P}(A), \subseteq, \emptyset, \cup \rangle$ and $\langle \mathcal{P}(A), \subseteq, A, \cap \rangle$.

Example 11.39. Consider the set $\{1, 2, 3, 6\}$ ordered by divisibility. For instance, since 2 divides by 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. 11.17).

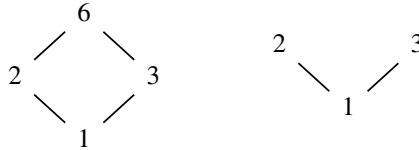


Figure 11.17.: Examples of a lattice and a non-lattice.

Lemma 11.40. UR is a bounded lattice (Definition 11.36) with

$$\langle UR, \leq_{UR}, \perp_{UR}, \top_{UR}, \vee_{UR}, \wedge_{UR} \rangle = \langle UR, \supseteq, R, \emptyset, \cap, \cup \rangle. \quad (11.15)$$

Proof. Consider the poset $\langle UR, \supseteq \rangle$ and $P, Q \in UR$.

- First, we need to show that $P \cap Q \in UR$. One has $P \subseteq UR$ and $Q \subseteq UR$, meaning that by definition, if $x \in P \cap Q$, we have $x \in P \wedge x \in Q$. It follows that $x \in UR$ for all $x \in P \cap Q$. Furthermore, we need to show that $P \cap Q$ is the least upper bound of P, Q . Assume this is not true, i.e. there exists a $T \in UR$, $T \neq P \cap Q$, such that $P \supseteq T \supseteq P \cap Q$ and $Q \supseteq T \supseteq P \cap Q$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} P \cap Q &\supseteq T \cap T \supseteq P \cap Q \\ P \cap Q &\supseteq T \supseteq P \cap Q \\ T &= P \cap Q, \end{aligned} \quad (11.16)$$

which contradicts the assumption. Therefore, $P \cap Q$ is the least upper bound of P, Q .

- Second, we need to show that $P \cup Q \in LF$. One has $P \subseteq UR$ and $Q \subseteq UR$, meaning that by definition, if $x \in P \cup Q$, we have either $x \in P$ or $x \in Q$. If $x \in P$, then $x \in UR$. If $x \in Q$, then $x \in UR$. It follows that $x \in UR$ for all $x \in P \cup Q$. Furthermore, we need to show that $P \cup Q$ is the greatest lower bound of P, Q . Assume this is not true, i.e. there exists a $T \in UR$, $T \neq P \cup Q$, such that $P \cup Q \supseteq T \supseteq P$ and $P \cup Q \supseteq T \supseteq Q$. Using the fact that union preserves inclusions, one has

$$\begin{aligned} (P \cup Q) \cup (P \cup Q) &\supseteq T \cup T \supseteq P \cup Q \\ P \cup Q &\supseteq T \supseteq P \cup Q \\ T &= P \cup Q, \end{aligned} \quad (11.17)$$

which contradicts the assumption. Therefore, $P \cup Q$ is the greatest lower bound of P, Q .

We have therefore proved that $\langle \text{UR}, \supseteq \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq T$ for any $T \in \text{UR}$, meaning that \emptyset is the top. Furthermore, we notice that $T \subseteq R$ for any $T \in \text{UR}$, meaning that R is a bottom. Therefore, the lattice is bounded. \square

Lemma 11.41. LF is a bounded lattice (Definition 11.36) with

$$\langle \text{LF}, \leq_{\text{LF}}, \perp_{\text{LF}}, \top_{\text{LF}}, \vee_{\text{LF}}, \wedge_{\text{LF}} \rangle = \langle \text{LF}, \subseteq, \emptyset, F, \cup, \cap \rangle. \quad (11.18)$$

Proof. Consider the poset $\langle \text{LF}, \subseteq \rangle$ and $P, Q \in \text{LF}$.

- First, we need to show that $P \cup Q \in \text{LF}$. One has $P \subseteq \text{LF}$ and $Q \subseteq \text{LF}$, meaning that by definition, if $x \in P \cup Q$, either $x \in P$ or $x \in Q$. If $x \in P$, then $x \in \text{LF}$. If $x \in Q$, then $x \in \text{LF}$. It follows that $x \in \text{LF}$ for all $x \in P \cup Q$. Furthermore, we need to show that $P \cup Q$ is the least upper bound of P, Q . Assume this is not true, i.e. there exists a $T \in \text{LF}$, $T \neq P \cup Q$, such that $P \subseteq T \subseteq P \cup Q$ and $Q \subseteq T \subseteq P \cup Q$. Using the fact that union preserves inclusions, one has

$$\begin{aligned} P \cup Q &\subseteq T \cup T \subseteq P \cup Q \\ P \cup Q &\subseteq T \subseteq P \cup Q \\ T &= P \cup Q, \end{aligned} \quad (11.19)$$

which contradicts the assumption. Therefore, $P \cup Q$ is the least upper bound of P, Q .

- Second, we need to show that $P \cap Q \in \text{LF}$. One has $P \subseteq \text{LF}$ and $Q \subseteq \text{LF}$, meaning that by definition, if $x \in P \cap Q$, we have $x \in P \wedge x \in Q$, i.e. $x \in \text{LF}$, for all $x \in P \cap Q$. Furthermore, we need to show that $P \cap Q$ is the greatest lower bound of P, Q . Assume this is not true, i.e. there exists a $T \in \text{LF}$, $T \neq P \cap Q$, such that $P \cap Q \subseteq T \subseteq P$ and $P \cap Q \subseteq T \subseteq Q$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} (P \cap Q) \cap (P \cap Q) &\subseteq T \cap T \subseteq P \cap Q \\ P \cap Q &\subseteq T \subseteq P \cap Q \\ T &= P \cap Q, \end{aligned} \quad (11.20)$$

which contradicts the assumption. Therefore, $P \cap Q$ is the greatest lower bound of P, Q .

We have therefore proved that $\langle \text{LF}, \subseteq \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq T$ for any $T \in \text{LF}$, meaning that \emptyset is the bottom. Furthermore, we notice that $T \subseteq F$ for any $T \in \text{LF}$, meaning that F is a top. Therefore, the lattice is bounded. \square

12. Poset constructions

to write

Contents

12.1. Opposite of a poset	85
12.2. Poset of intervals	86
12.3. A different poset of intervals	86



12.0.1. Product of posets

We can think of the product of posets.

Definition 12.1 (Product of posets). Given two posets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, the *product poset* is $\langle X \times Y, \leq_{X \times Y} \rangle$, where $X \times Y$ is the Cartesian product of two sets (Definition 19.1) and the order $\leq_{X \times Y}$ is given by:

$$\langle x_1, y_1 \rangle \leq_{A \times B} \langle x_2, y_2 \rangle \Leftrightarrow (x_1 \leq_X x_2) \wedge (y_1 \leq_Y y_2). \quad (12.1)$$

Recalling the pizza recipes example, we have the two posets representing time and money. Given that we want to minimize both time and costs, by considering the money poset containing elements 1 USD, 2 USD, and 3 USD, and the time poset containing elements 1 h, and 2 h, one can represent the product as in Fig. 12.1.

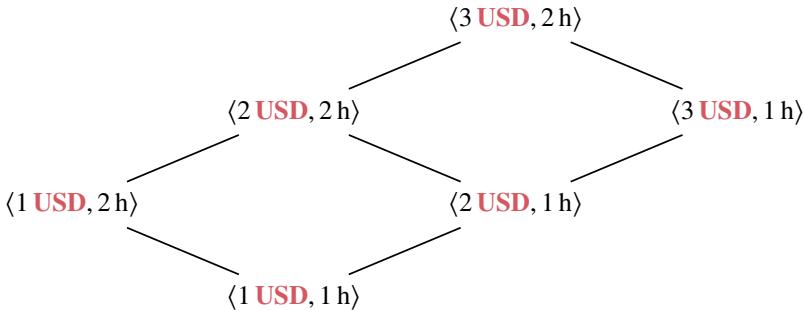


Figure 12.1.: Product poset of time and cost for pizza recipes.

Example 12.2. Consider now the two posets given in Fig. 12.2. Their product is depicted

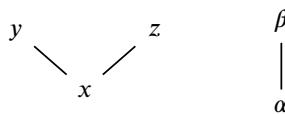


Figure 12.2.: Two posets.

in Fig. 12.3.

12.0.2. Disjoint union of posets

Similarly to what we have done for sets in Section 19.2, we can think of alternatives in the poset case through their disjoint union.

Definition 12.3 (Disjoint union of posets). Given posets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, we can define their *disjoint union* $\langle X + Y, \leq_{X+Y} \rangle$, where $X + Y$ is the disjoint union of the sets X

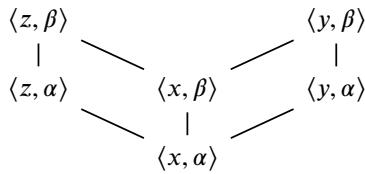


Figure 12.3.: Product of two posets.

and Y (Definition 19.14), and the order \leq_{X+Y} is given by:

$$x \leq_{X+Y} y \equiv \begin{cases} x \leq_A y, & x, y \in X, \\ x \leq_B y, & x, y \in Y. \end{cases} \quad (12.2)$$

$$\begin{aligned} \leq_{X+Y} : (X + Y) \times (X + Y) &\rightarrow \text{Bool} \\ \langle 1, x_1 \rangle, \langle 1, x_2 \rangle &\mapsto (x_1 \leq_X x_2) \\ \langle 2, y_1 \rangle, \langle 1, x \rangle &\mapsto \perp \\ \langle 1, x \rangle, \langle 2, y_1 \rangle &\mapsto \perp \\ \langle 2, y_1 \rangle, \langle 2, y_2 \rangle &\mapsto (y_1 \leq_Y y_2). \end{aligned} \quad (12.3)$$

Example 12.4. Consider the posets $X = \langle \diamond, \star \rangle$ with $\diamond \leq_X \star$, and $Y = \langle \dagger, * \rangle$, with $* \leq_Y \dagger$. Their disjoint union can be represented as in Fig. 12.4.

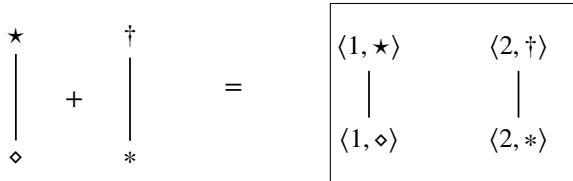


Figure 12.4.: Disjoint union of posets.

12.1. Opposite of a poset

Definition 12.5. The *opposite* of a poset $\langle A, \leq \rangle$ is the poset denoted as $\langle A^{\text{op}}, \leq^{\text{op}} \rangle$ that has the same elements as A and the reverse ordering (Fig. 12.5). For a given $x \in A$, we use x^* to represent its corresponding copy in A^{op} ; note that x and x^* belong to distinct posets. Reversing the order means that, for all $x, y \in A$,

$$x \leq y \Leftrightarrow y^* \leq^{\text{op}} x^*. \quad (12.4)$$

Example 12.6 (Credit and debt). Let us define the set

$$\text{USD} = \{\$0.00, \$0.01, \$0.02, \dots\}$$

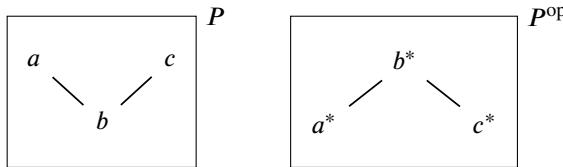


Figure 12.5.: Opposite of a poset.

of all US dollars monetary quantities approximated to the cent. From this set we can define two posets: $\text{USD}^+ = \langle \text{USD}, \leq \rangle$ and $\text{USD}^- = \langle \text{USD}, \geq \rangle$, that are the opposite of each other. If the context is that, given two quantities \$1 and \$2, we prefer \$1 to \$2 (for example because it is a cost to pay to acquire a component), then we are working in USD^+ , otherwise we are working in USD^- (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”.

12.2. Poset of intervals

Definition 12.7 (Poset of intervals). An interval is an ordered pair of elements $\langle l, u \rangle$ of P , such that $l \leq_P u$. Given a poset P , one can define a *poset of intervals* on P . Intervals can be ordered by inclusion, e.g.:

$$\langle l_1, u_1 \rangle \preceq_{\text{Int } P} \langle l_2, u_2 \rangle \Leftrightarrow (l_1 \leq_P l_2) \wedge (u_2 \leq_P u_1).$$

12.3. A different poset of intervals

to write

13. Life is hard

to write

Contents

13.1. Monotone maps	88
13.2. Compositionality of monotonicity	90
13.3. The category Pos of posets and monotone maps	90
13.4. Why Pos is not sufficient for design theory	91



13.1. Monotone maps

AC: Add some motivation from design. E.g. generalization of "non-decreasing" and "non-increasing" from real numbers to partial orders

Definition 13.1 (Monotone map). A *monotone map* between two posets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$ is a map that preserves the ordering, in the sense that

$$a \leq_A b \Rightarrow f(a) \leq_B f(b).$$

Remark 13.2. Given a poset A , the map id_A is monotone, since for $a_1, a_2 \in A$, one has:

$$\begin{aligned} a_1 \leq_A a_2 &\Rightarrow a_1 = a_1 \circ \text{id}_A \\ &\leq_A a_2 \circ \text{id}_A \\ &= a_2. \end{aligned}$$

Definition 13.3 (Antitone map). An *antitone map* between two posets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$ is a map that reverses the ordering, in the sense that

$$a \leq_A b \Rightarrow f(a) \geq_B f(b).$$

Example 13.4 (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 $t : \mathbb{N} \rightarrow \mathbb{R}_+$ is a map between the ordered sets \mathbb{N} and \mathbb{R}_+ , and maps each quantity of widgets to a total manufacturing cost (Fig. 13.2). Clearly, t is a monotone function. Conversely, the unit cost function $u : \mathbb{N} \rightarrow \mathbb{R}_+$ is antitone (Fig. 13.1).

Figure 13.1.:

add picture

Figure 13.2.:

add picture

Example 13.5 (Rounding functions).

to write

Example 13.6 (Cardinality map). In Example 11.8 we presented the poset arising from the power set of a set $A = \{a, b, c\}$ and ordered via subset inclusion.

The map $|\cdot| : \mathcal{P}(A) \rightarrow \mathbb{N}$ (cardinality), is a monotone map (Fig. 13.3).

Lemma 13.7. Consider a discrete poset A and a poset B . Any map $f : A \rightarrow B$ is monotone.

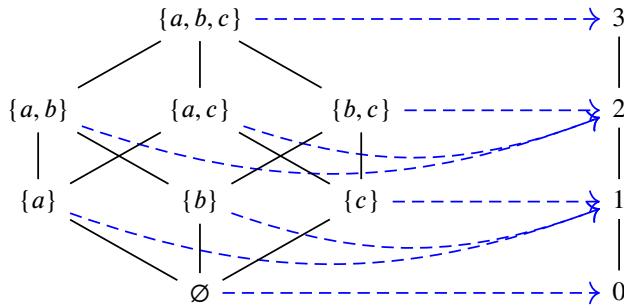


Figure 13.3.: The cardinality map is a monotone map.

Proof. Since A is a discrete poset, one has

$$a_1 \leq_A a_2 \iff a_1 = a_2. \quad (13.1)$$

Therefore, one has

$$\begin{aligned} a_1 \leq_A a_2 &\Rightarrow a_1 = a_2 \\ &\Rightarrow f(a_1) = f(a_2) \\ &\Rightarrow f(a_1) \leq_B f(a_2). \end{aligned} \quad (13.2)$$

□

Unless indicated otherwise, in this paper all maps between posets are assumed to be monotone or will turn out to be monotone. In a similar way, one can define antitone maps.

Example 13.8. We now look at an example of **set-based filtering**, where filtering refers to online inference. Suppose that we want to track the value of a quantity $x \in [0, 100]$, without having a priory information about x . We are equipped with sensors, which periodically measure the quantity x with some variable precision, i.e., 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

$$x \in ([0, 100] \cap [21, 24]) \equiv x \in [21, 24].$$

Let’s now 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:

$$x \in ([20, 25] \cap [21, 24]) \equiv x \in [21, 24].$$

One of the structures which could sustain this kind of inference, is the of *posets of intervals* (Definition 12.7). The Hasse diagram representing a situation related to this diagram could be as reported in Fig. 13.4.

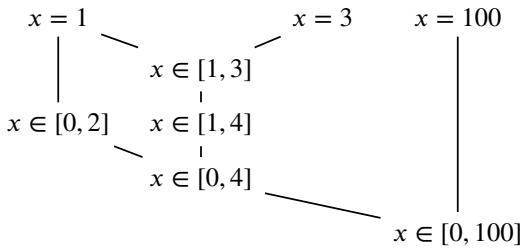


Figure 13.4.

13.2. Compositionality of monotonicity

Note that monotonicity is a compositional property.

Lemma 13.9. Given posets A, B, C and monotone maps $f : A \rightarrow B$ and $g : B \rightarrow C$, the composite map $f ; g : A \rightarrow C$ is monotone as well.

Proof. Consider $a_1, a_2 \in A$, $b_1, b_2 \in B$. We have, by definition,

$$\begin{aligned} a_1 \leq_A a_2 &\Rightarrow f(a_1) \leq_B f(a_2) \\ b_1 \leq_B b_2 &\Rightarrow g(b_1) \leq_C g(b_2). \end{aligned} \tag{13.3}$$

By substituting the above in the map composition formula, one has

$$a_1 \leq_A a_2 \Rightarrow (f ; g)(a_1) \leq_C (f ; g)(a_2), \tag{13.4}$$

which is the monotonicity condition for $(f ; g)$. \square

13.3. The category Pos of posets and monotone maps

In this section, we want to abstract the concept of poset and describe a category in which the objects are posets themselves, and the morphisms are monotone functions between them. This category is called **Pos**.

Definition 13.10 (Category Pos). The category **Pos** is defined by:

1. *Objects:* The objects of this category are all posets.
2. *Morphisms:* The morphisms from a poset X to a poset Y are the monotone maps from X to Y .
3. *Identity morphism:* The identity morphism for the poset X is the identity map id_X .
4. *Composition operation:* The composition operation is composition of maps.

Occasionally we will write $f : X \rightarrow_{\mathbf{Pos}} Y$ to emphasize that a monotone map between posets is a morphism in **Pos**.

13.4. Why **Pos** is not sufficient for design theory

Rewrite this without assuming we know what is a design problem.

The category **Pos** of posets and monotone maps that we have described can model many facts that are useful for design theory. However, there are also limitations which motivate us to describe a more general category. This section describes the usefulness and the limitations of **Pos**.

Example 13.11 (Battery). Consider a model of a battery where the capacity is the functionality and the mass of the battery is the resource. There is certainly a monotone map from capacity to mass. This map answers the question: “Given a value of the capacity, what is the minimum mass needed?”. Conversely, in the other direction, the map that answers the question: “Given a certain mass, what is the maximum capacity that can be provided?” is also a monotone map.

Therefore, at first sight it might seem that posets and monotone maps would be sufficient to describe a quantitative theory of design. However, there are more general relations to be modeled. It is easy enough to describe examples in which having a simple monotone map from functionality to resources is not sufficient.

Example 13.12 (Delivery drone). Consider the design of a delivery drone, in which the functional requirement is that the drone should be able to make a delivery at a distance d and we need to reason about how powerful to make the drone. In particular, we need to choose at what (average) *velocity* v the drone should travel and what is the optimal *mission duration*. The relation between distance d , velocity v , and mission duration T is given by $d = v \cdot T$. We can choose to have either a fast drone and short missions, or a slow drone and long missions. This is an interesting trade-off. Flying fast takes more energy, both for propulsion as well as for computation (more objects to be observed and processed). Flying too slow will also be excessively energy-consuming because of the long mission duration. If we consider v and T as given, then the map $\langle v, T \rangle \mapsto v \cdot T$ is clearly a monotone function that gives the distance which the drone can cover. However, in the other direction, we do not have a simple map, but rather a 1-to-many relation $\text{distance} \rightarrow \text{velocity} \times \text{time}$. For each fixed value of the distance, there is an entire continuum of values of v and T which we can choose, as it can be seen in Fig. 13.5.

In other words, using **Pos** it is not possible to make a theory of *trade-offs*. We will introduce a more general category, called the category **DP** of *design problems* (Section 23.1), which will allow to describe such a theory.

GZ: Check with JL what he means.

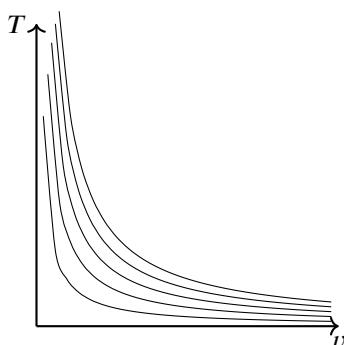


Figure 13.5.: Antichains in $\langle v, T \rangle$ for different values of d .

14. Functors

to write

Contents

14.1. Functors	94
14.2. A poset as a category	95
14.3. Other examples of functors	96



14.1. Functors

We can think of a given category **C** as a “compositional world”: inside of **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*.

Definition 14.1 (Functor). Given categories **C** and **D**, a *functor* $F : \mathbf{C} \rightarrow \mathbf{D}$ from **C** to **D** is defined by the following data, satisfying the following conditions.

Data:

- i) For every object $x \in \text{Ob}_{\mathbf{C}}$, an object $F(x) \in \text{Ob}_{\mathbf{D}}$;
- ii) For every morphism $f : x \rightarrow y$ in **C**, a morphism $F(f) : F(x) \rightarrow F(y)$ in **D**.

Conditions:

- i) For every object $x \in \text{Ob}_{\mathbf{C}}$, one has $F(\text{id}_x) = \text{id}_{F(x)}$;
- ii) For every three objects $x, y, z \in \text{Ob}_{\mathbf{C}}$ and two morphisms $f \in \text{Hom}_{\mathbf{C}}(x, y)$, $g \in \text{Hom}_{\mathbf{C}}(y, z)$, the equation

$$F(f \circ g) = F(f) \circ F(g) \quad (14.1)$$

holds in **D**.

This situation is graphically reported in Fig. 14.1.

Remark 14.2. A functor from a category to itself it is called an *endofunctor*.

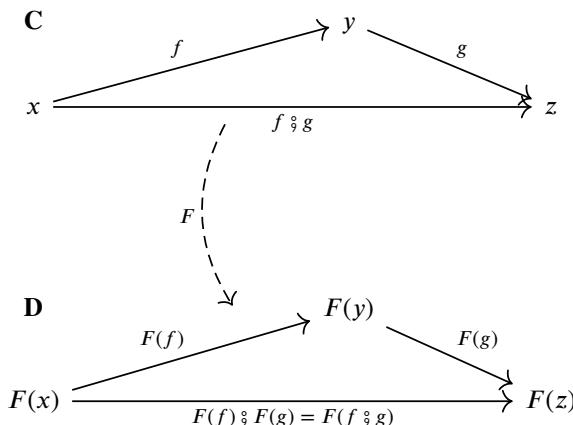


Figure 14.1.

14.2. A poset as a category

A single poset $\langle P, \leq \rangle$ can be described as a category, in which each point $x \in P$ is an object, and there is a morphism between x and y if and only if $x \leq y$. This is a “thin” category, which means that there is at most one morphism between two objects: For any $x, y \in P$, there exist only one relation $x \leq y$ in P (Definition 11.1). The identity morphism is given by the reflexivity property of posets, i.e. for any $x \in P$, we have $x \leq x$. Furthermore, composition is given by the transitivity property of posets, i.e. for $x, y, z \in P$, $x \leq y$ and $y \leq z$ implies $x \leq z$.

Example 14.3. Let's revisit Example 11.8, in which we had a poset $\mathcal{P}(\{a, b, c\})$ with order given by inclusion (Fig. 14.2).

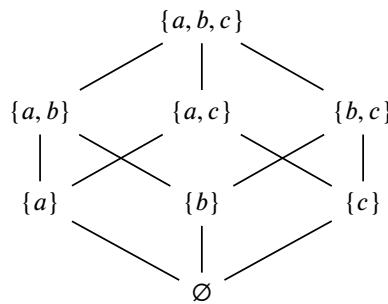


Figure 14.2.: Power set $\mathcal{P}\{a, b, c\}$ as a category.

This is a category **C**, with $\text{Ob}_C = \mathcal{P}(\{a, b, c\})$, and morphisms given by the inclusions. Note that we omit to draw self-arrows for the identity morphisms. Composition is given by the transitivity law of posets. For instance, since $\{a\} \subseteq \{a, b\}$ and $\{a, b\} \subseteq \{a, b, c\}$, we can say that $\{a\} \subseteq \{a, b, c\}$.

14.2.1. Monotone maps are functors

Note that morphisms in **Pos** are morphisms between posets, and we have just discovered that posets are examples of categories.

Lemma 14.4. A monotone map F between posets P, Q is a functor between the “posetal categories” P and Q .

Proof. We start by specifying the functor F and two posets P and Q . We first specify the action of F on objects (elements of a poset) and on morphisms (order relations). A monotone function maps each element of a poset $p \in P$ to $F(p) \in Q$, and it guarantees that for every $p_1, p_2 \in P$, if $p_1 \leq p_2$ then $F(p_1) \leq F(p_2)$. We now need to check the two conditions that a functor must satisfy. First, consider the identity morphism for $p \in P$, namely $p \leq p$. The application of the map F results in the condition $F(p) \leq F(p)$, which is the identity morphism on Q . Second, morphisms $p_1 \leq p_2$ and $p_2 \leq p_3$ in P , by applying the map F to the morphism composition $p_1 \leq p_3$ one obtains $F(p_1) \leq F(p_3)$,

which is the same as the composition of $F(p_1) \leq F(p_2)$ and $F(p_2) \leq F(p_3)$. \square

14.3. Other examples of functors

14.3.1. The list functor

In the following, we present a concept related to the monoidal structure introduced.

This is the free monoid functor and we need the concept of monoid (or at least it is elegant if we have it). This is now at section 21 though....

to write

14.3.2. Planning as the search of a functor

Example 14.5. Recall the category **Berg** introduced in Section 5.2 and define a category **Plans** where objects are specific areas of the mountain and morphisms describing visiting order constraints, illustrated in Fig. 14.3. For instance, there is a morphism from “mountain

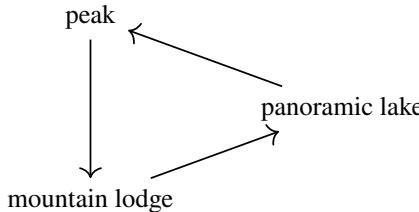


Figure 14.3.: Example of visiting order constraints on a mountain.

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 area” to “peak”, their composition is the plan of going from “mountain lodge” to the “peak”, passing through the “panoramic lake” (concatenation).

When we talk about **planning** in this context, we refer to the action of finding a functor from **Plans** to **Berg**. Let’s look at this in more detail. The objects of **Berg** are tuples (p, v) , 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. 14.3, we want to find a map P which maps each object in **Plans** (area of the mountain) to an object of **Berg** (specific location and velocity). Similarly, it must map each morphism in **Plans** (visiting order constraints) to a morphism in **Berg** (specific paths). This is illustrated in Fig. 14.4.

Figure 14.4.:

to create

15. Up the ladder

to write

Contents

15.1. Functor composition	100
15.2. A category of categories	100
15.3. Full and faithful functors	101
15.4. Forgetful functor	101



15.1. Functor composition

JL: I think this should be a definition, not a lemma.

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 $x \in \mathbf{A}$, we have:

$$\begin{aligned}(F \circ G)(\text{id}_x) &= G(F(\text{id}_x)) \\ &= G(\text{id}_{F(x)}) \\ &= \text{id}_{G(F(x))},\end{aligned}$$

where we used that F and G are functors (i.e., they preserve identities).

- Furthermore, given composable morphisms $f, g \in \mathbf{A}$, one has:

$$\begin{aligned}(F \circ G)(f \circ g) &= G(F(f) \circ F(g)) \\ &= G(F(f)) \circ G(F(g)),\end{aligned}$$

where again used that F, G are functors (i.e., they preserve composition).

We can define an identity functor. Given a category **C**, we define it as $\text{id}_{\mathbf{C}} : \mathbf{C} \rightarrow \mathbf{C}$, $\text{id}_{\mathbf{C}}(x) = x$ for every object and morphism in **C**. 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:

$$\begin{aligned}\text{id}_{\mathbf{C}}(\text{id}_x) &= \text{id}_x \\ &= \text{id}_{\text{id}_{\mathbf{C}}(x)}\end{aligned}$$

Furthermore, given composable morphisms f, g in **C**, we have:

$$\begin{aligned}\text{id}_{\mathbf{C}}(f \circ g) &= f \circ g \\ &= \text{id}_{\mathbf{C}}(f) \circ \text{id}_{\mathbf{C}}(g).\end{aligned}$$

15.2. 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**.

Definition 15.1. There is a category, called **Cat**, which is constituted of

- Objects: categories;
- Morphisms: functors;
- Identity morphisms: identity functors;
- Composition: composition of functors.

15.3. Full and faithful functors

Definition 15.2 (Full and faithful functors). A functor $F : \mathbf{C} \rightarrow \mathbf{D}$ is *full* (respectively *faithful*) if for each pair of objects $x, y \in \mathbf{C}$, the function

$$F : \text{Hom}_{\mathbf{C}}(x, y) \rightarrow \text{Hom}_{\mathbf{D}}(F(x), F(y)) \quad (15.1)$$

is surjective (respectively injective).

Example 15.3. Let \mathbf{C} be the category depicted in Fig. 15.1. Let $E : \mathbf{C} \rightarrow \mathbf{C}$ be the endofunctor which maps object $X \in \text{Ob}_{\mathbf{C}}$ to X and object $Y \in \text{Ob}_{\mathbf{C}}$ to X . This functor is full and faithful. Note that the map of objects and the map of morphisms are neither surjective nor injective.

$$X \begin{array}{c} \xleftarrow{\quad} \\[-1ex] \xrightarrow{\quad} \end{array} Y$$

Figure 15.1.

OK, so what? examples? why is it useful?

15.4. Forgetful functor

AC: What is the definition of a forgetful functor?

JL: There is not a formal definition of a “forgetful functor”; I think this lemma doesn’t make sense as stated. We could instead state/explain how “forgetting structure” is often functorial, and include, among other things, the $\text{Pos} \rightarrow \text{Set}$ example.

Example 15.4. The functor $\text{Pos} \rightarrow \text{Set}$ is a forgetful functor. This functor maps every poset to the set which has the same elements, but no notion of order. Furthermore, it maps each monotone map between posets to the corresponding function between sets. This is a forgetful functor in the sense that it forgets the notion of order and monotone maps.

Add more examples

16. Duality

to write

Contents

16.1. Galois connections	104
---------------------------------	-----



16.1. Galois connections

GZ: Define the category with object Posets and morphisms Galois connections.

AC: Explain what this category can and cannot represent. (It is a subcategory of DP. (preliminary exercise).)

In ref to next section, reorganize the flow like this:

- 1) What can Pos represent and what not.
- 2) What can Galois represent.
- 3) What's missing and justifies DP.

Definition 16.1 (Monotone Galois Connection). A *monotone Galois connection* between posets A and B is a pair of monotone maps $f : A \rightarrow B$ and $g : B \rightarrow A$ such that for all $a \in A, b \in B$:

$$f(a) \leq_B b \Leftrightarrow a \leq_A g(b). \quad (16.1)$$

This is equivalent to ask, for all $a \in A, b \in B$, that:

$$a \leq_A g(f(a)) \quad \text{and} \quad b \geq_B f(g(b)). \quad (16.2)$$

Definition 16.2 (Antitone Galois Connection). An *antitone Galois connection* between A and B is a pair of antitone maps $f : A \rightarrow B$ and $g : B \rightarrow A$ such that for all $a \in A, b \in B$:

$$b \leq_B f(a) \Leftrightarrow a \leq_A g(b). \quad (16.3)$$

This is equivalent to ask for all $a \in A, b \in B$:

$$a \leq_A g(f(a)) \quad \text{and} \quad b \leq_B f(g(b)). \quad (16.4)$$

Consider a boolean profunctor $d : F \nrightarrow R$. We can define the maps that work on single functionality and resources:

$$\begin{aligned} \theta : F &\rightarrow \mathsf{UR} \\ f &\mapsto \{r \in R : d(f, r)\}, \end{aligned} \quad (16.5)$$

$$\begin{aligned} \psi : R &\rightarrow \mathsf{LF} \\ r &\mapsto \{f \in F : d(f, r)\}. \end{aligned} \quad (16.6)$$

We can define the maps that work on multiple functionality and resources:

$$\begin{aligned} \alpha : \mathsf{LF} &\rightarrow \mathsf{UR} \\ S &\mapsto \{r \in R : \exists f \in S : d(f, r)\}, \end{aligned} \quad (16.7)$$

Alternatively, we can write

$$\begin{aligned}\alpha : \text{LF} &\rightarrow \text{UR} \\ S &\mapsto \bigcup_{f \in S} \theta(f).\end{aligned}\tag{16.8}$$

$$\begin{aligned}\beta : \text{UR} &\rightarrow \text{LF} \\ T &\mapsto \{f \in F : \exists r \in T : d(f, r)\},\end{aligned}\tag{16.9}$$

Alternatively, we can write

$$\begin{aligned}\beta : \text{UR} &\rightarrow \text{LF} \\ T &\mapsto \bigcup_{r \in T} \psi(r).\end{aligned}\tag{16.10}$$

$$\begin{aligned}\delta : \text{LF} &\rightarrow \text{UR} \\ S &\mapsto \{r \in R : \forall f \in S : d(f, r)\},\end{aligned}\tag{16.11}$$

Alternatively, we can write

$$\begin{aligned}\delta : \text{LF} &\rightarrow \text{UR} \\ S &\mapsto \bigcap_{f \in S} \theta(f).\end{aligned}\tag{16.12}$$

$$\begin{aligned}\gamma : \text{UR} &\rightarrow \text{LF} \\ T &\mapsto \{f \in F : \forall r \in T : d(f, r)\},\end{aligned}\tag{16.13}$$

Alternatively, we can write

$$\begin{aligned}\delta : \text{UR} &\rightarrow \text{LF} \\ T &\mapsto \bigcap_{r \in T} \psi(r).\end{aligned}\tag{16.14}$$

Properties of these maps are reported in Table 16.1.

\star	X	Y	$\star(\perp)$	$\star(\top)$	$A \leq_X B$	$\star(A \vee_X B)$	$\star(A \wedge_X B)$
α	LF	UR	$\alpha(\emptyset) = \emptyset$	$\alpha(F) \geq_{UR} \alpha(\cdot)$	$\alpha(A) \geq_{UR} \alpha(B)$	$\alpha(A) \vee_{LF} \alpha(B)$	$\alpha(A) \wedge_{LF} \alpha(B)$
β	UR	LF	$\beta(R) \geq_{LF} \beta(\cdot)$	$\beta(\emptyset) = \emptyset$	$\beta(A) \geq_{LF} \beta(B)$	$\beta(A) \vee_{LF} \beta(B)$	$\beta(A) \wedge_{LF} \beta(B)$
δ	LF	UR	$\delta(\emptyset) = R$	$\delta(F) \geq_{UR} \delta(\cdot)$	$\delta(A) \leq_{UR} \delta(B)$	$\delta(A) \wedge_{UR} \delta(B)$	$\delta(A) \vee_{UR} \delta(B)$
γ	UR	LF	$\gamma(R) \leq_{LF} \gamma(\cdot)$	$\gamma(\emptyset) = F$	$\gamma(A) \leq_{LF} \gamma(B)$	$\gamma(A) \wedge_{LF} \gamma(B)$	$\gamma(A) \vee_{LF} \gamma(B)$

Table 16.1.: Properties of $\alpha, \beta, \delta, \gamma$

Lemma 16.3. δ and γ are monotone maps.

Proof. We first prove that δ is a monotone map. Given $A, B \in \text{LF}$ with $A \subseteq B$, one has

$$\begin{aligned}\delta(A) &= \{r \in R : \forall f \in A : d(f, r)\} \\ &\supseteq \{r \in R : \forall f \in B : d(f, r)\} \\ &= \delta(B),\end{aligned}\tag{16.15}$$

meaning that $A \leq_{LF} B \Rightarrow \delta(A) \leq_{UR} \delta(B)$. We now prove that γ is a monotone map.

Given $C, D \in UR$, with $C \supseteq D$, one has

$$\begin{aligned}\gamma(C) &= \{f \in F : \forall r \in C : d(f, r)\} \\ &\subseteq \{f \in F : \forall r \in D : d(f, r)\} \\ &= \gamma(D),\end{aligned}\tag{16.16}$$

meaning that $C \leq_{UR} D \Rightarrow \gamma(C) \leq_{LF} \gamma(D)$. \square

Lemma 16.4. α and β are antitone maps.

Proof. We first prove that α is an antitone map. Given $A, B \in LF$, with $A \subseteq B$, one has

$$\begin{aligned}\alpha(A) &= \{r \in R : \exists f \in A : d(f, r)\} \\ &\subseteq \{r \in R : \exists f \in B : d(f, r)\} \\ &= \alpha(B),\end{aligned}\tag{16.17}$$

meaning that $A \leq_{LF} B \Rightarrow \alpha(A) \geq_{UR} \alpha(B)$. We now prove that β is an antitone map. Given $C, D \in UR$, with $C \supseteq D$, one has

$$\begin{aligned}\beta(C) &= \{f \in F : \exists r \in C : d(f, r)\} \\ &\supseteq \{f \in F : \exists r \in D : d(f, r)\} \\ &= \beta(D),\end{aligned}\tag{16.18}$$

meaning that $C \leq_{UR} D \Rightarrow \beta(C) \geq_{UR} \beta(D)$. \square

Lemma 16.5. (δ, γ) forms a **monotone** Galois connection between LF and UR .

Proof. In Lemma 16.3 we proved that δ and γ are monotone maps. We now need to show that for any lower set $L \subseteq F$ of functionalities and upper set $U \subseteq R$ of resources, we have

$$L \subseteq \gamma(U) \iff \delta(L) \supseteq U\tag{16.19}$$

The left-hand side says that if $f \in L$, then $\forall r \in U$ we have $d(f, r) = \top$. The right-hand side says that if $r \in U$ then $\forall f \in L$, $d(f, r) = \top$. Both are equivalent to $\forall f \in L, r \in U, d(f, r) = \top$, and hence to each other. In formulas:

$$\begin{aligned}L \subseteq \gamma(U) &\equiv L \subseteq \{f \in F : \forall r \in U : d(f, r)\} \\ &\equiv \forall f \in L, r \in U : d(f, r) = \top \\ &\equiv \forall r \in U, f \in L : d(f, r) = \top \\ &\equiv U \subseteq \{r \in R : \forall f \in L : d(f, r) = \top\} \\ &\equiv U \subseteq \delta(L).\end{aligned}\tag{16.20}$$

\square

Lemma 16.6. (α, β) does not form an **antitone** Galois connection between LF and UR .

Proof. In Lemma 16.4 we have proved that α and β are antitone maps. For $L \in F$, $U \in R$, we want to show that the following does not hold:

$$L \subseteq \beta(\alpha(L)) \quad (16.21)$$

and

$$U \supseteq \alpha(\beta(U)). \quad (16.22)$$

Example Consider d as the design problem which is always not feasible (the empty profunctor), i.e. $d(f, r) = \perp, \forall f \in F, r \in R$. Take any $L \in F$. We know that $\alpha(L) = \emptyset$, and $\beta(\alpha(L)) = \beta(\emptyset) = \emptyset$. But $L \subseteq \emptyset$ is not true. \square

16.1.1. Opposites

Definition 16.7 (Opposite category). Given a category \mathbf{C} , the *opposite category* \mathbf{C}^{op} has the same objects as \mathbf{C} , but a morphism $f : x \rightarrow y$ in \mathbf{C}^{op} is the same as a morphism $f : y \rightarrow x$ in \mathbf{C} . Furthermore, a composite of morphisms $f ; g$ in \mathbf{C}^{op} is the composite $g ; f$ in \mathbf{C} .

Example 16.8 (Opposite of a poset). We have defined the opposite of a poset in Section 12.1. Since a poset is a category (Section 14.2) this is an example of the above definition. Consider \mathbf{C} representing any poset. We can think of $(\cdot)^{\text{op}}$ as a functor of the form

$$(\cdot)^{\text{op}} : \mathbf{C} \rightarrow \mathbf{C}^{\text{op}}, \quad (16.23)$$

which preserves objects (i.e., every object $A \in \mathbf{C}$ is mapped to $A^* \in \mathbf{C}^{\text{op}}$), and maps each morphism $f : A \rightarrow B$ in \mathbf{C} to a morphism $f' : B \rightarrow A$ in \mathbf{C}^{op} . This is a functor:

- For any object A , the identity morphism $\text{id}_A : A \rightarrow A$ in \mathbf{C} is itself, i.e., it is mapped to $\text{id}'_A : A \rightarrow A$, meaning that $(\text{id}_A)^{\text{op}} = \text{id}_{(A)^{\text{op}}} = \text{id}_A$.
- Given two morphisms $a \leq b, b \leq c$ in \mathbf{C} , one has

$$\begin{aligned} (a \leq_{\mathbf{C}} c)^{\text{op}} &= c \leq_{\mathbf{C}^{\text{op}}} a \\ &= (b \leq_{\mathbf{C}^{\text{op}}} a) \wedge (c \leq_{\mathbf{C}^{\text{op}}} b) \\ &= (a \leq_{\mathbf{C}} b)^{\text{op}} \wedge (b \leq_{\mathbf{C}} c)^{\text{op}}. \end{aligned} \quad (16.24)$$

17. Naturality

to write

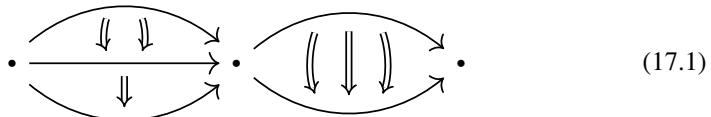
Contents

17.1. Natural transformations	110
---	-----



17.1. Natural transformations

We have seen that functors are “morphisms between categories”. Indeed, categories may be assembled into a category where the objects are categories and morphisms are functors. It turns out that there is an important third layer to this world of categories: there are also kinds of morphisms *between* functors, and these are known as “natural transformations”. To represent the three layers of structure involved in the world of categories, we will often draw diagrams like this:



where the points represent categories, the single arrows represent functors, and the double arrows represent natural transformations.

Definition 17.1 (Natural transformation). Let \mathbf{C} and \mathbf{D} be categories, and let $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors. A *natural transformation* $\alpha : F \rightarrow G$

$$\mathbf{C} \xrightarrow[\text{G}]{\text{F} \atop \Downarrow \alpha} \mathbf{D} \quad (17.2)$$

is defined by the following constituent data, satisfying the following condition. Data:

1. For each object $x \in \mathbf{C}$, a morphism $\alpha_x : F(x) \rightarrow G(x)$ in \mathbf{D} , called the x -component of α .

Condition:

1. For every morphism $f : x \rightarrow y$ in \mathbf{C} , the components of α must satisfy the *naturality condition*

$$F(f) \circ \alpha_y = \alpha_x \circ G(f), \quad (17.3)$$

i.e. the following diagram must commute:

$$\begin{array}{ccc} F(x) & \xrightarrow{F(f)} & F(y) \\ \alpha_x \downarrow & & \downarrow \alpha_y \\ G(x) & \xrightarrow{G(f)} & G(y) \end{array} \quad (17.4)$$

The situation is represented diagrammatically in Fig. 17.1.

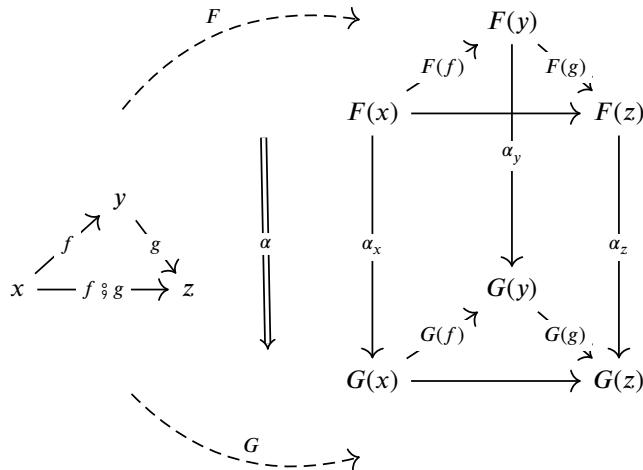


Figure 17.1.

Definition 17.2 (Natural isomorphism). A natural transformation $\alpha : F \rightarrow G$ is called a *natural isomorphism* if each component α_x is an isomorphism in \mathbf{D} .

Example 17.3. 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

$$V^* := \text{Hom}_{\mathbf{Vect}}(V, \mathbb{R}),$$

i.e., the space of all linear maps from V to \mathbb{R} . Also, recall that if $f : V \rightarrow W$ is a linear map, then its dual is a linear map $f^* : W^* \rightarrow V^*$.

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 Double dual : $\mathbf{Vect} \rightarrow \mathbf{Vect}$ 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 $\alpha_V : V \rightarrow V^{**}$ defined by

$$\alpha_V(v)(l) = l(v), \quad v \in V, l \in V^*.$$

These form the components of a natural transformation from the identity functor on \mathbf{Vect} to the double dual functor.

$$\begin{array}{ccc}
 & \text{Id} & \\
 \text{Vect}_{\mathbb{R}} & \Downarrow \alpha & \text{Vect}_{\mathbb{R}} \\
 & \text{Double dual} &
 \end{array} \tag{17.5}$$

18. Adjunctions

to write

Contents

18.1. An example	114
18.2. Adjunctions: hom-set definition	114
18.3. Adjunctions: (co)unit definition	114
18.4. Example of a “Product-Hom” adjunction	116
18.5. Example of a “Free-Forgetful” adjunction	117
18.6. Relating the two definitions	118



18.1. An example

JL: An example of a Galois connection between functionalities and resources could be treated here to motivate the adjunction discussion

18.2. 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

$$(Fv, w)_W = (v, F^*w)_V, \quad \forall v \in V, w \in W.$$

Definition 18.1 (Adjunction, Version 1). Let **C** and **D** be categories. An *adjunction* from **C** to **D** is given by the following data:

1. A functor $F : \mathbf{C} \rightarrow \mathbf{D}$ (the *left adjoint*);
2. A functor $G : \mathbf{D} \rightarrow \mathbf{C}$ (the *right adjoint*);
3. A natural isomorphism $\tau : \text{Hom}_{\mathbf{D}}(F-, -) \Rightarrow \text{Hom}_{\mathbf{C}}(-, G-)$

We use the notation $F \dashv G$ to indicate that F and G form an adjunction, with F the left adjoint and G the right adjoint.

Remark 18.2. Note that τ is a natural isomorphism between functors of the form

$$\mathbf{C}^{op} \times \mathbf{D} \longrightarrow \mathbf{Set} \tag{18.1}$$

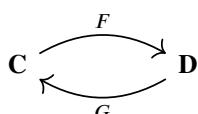
18.3. Adjunctions: (co)unit definition

Recall: in a category **C**, a morphism $f : x \rightarrow y$ is an isomorphism if there exists a morphism $g : y \rightarrow x$ such that

$$f \circ g = \text{id}_x \text{ and } g \circ f = \text{id}_y$$

Now let's think about this definition in the case where **C** is the category **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 18.6.

The idea of “weakening” the notion of isomorphism of categories is as follows. Given functors



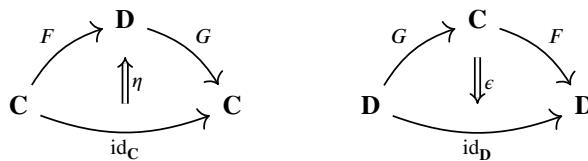
instead of requiring the equations

$$F \circ G = \text{id}_{\mathbf{C}} \text{ and } G \circ F = \text{id}_{\mathbf{D}}$$

we can replace the equality symbols with 2-morphisms! We'll do it like this:

$$F \circ G \xleftarrow{\eta} \text{id}_{\mathbf{C}}, \quad G \circ F \xrightarrow{\epsilon} \text{id}_{\mathbf{D}}$$

The last two relationships can also be depicted in the following more geometric manner:



Definition 18.3 (Equivalence of categories). Let \mathbf{C} and \mathbf{D} be categories. An *equivalence* between \mathbf{C} and \mathbf{D} is the following data:

1. A functor $F : \mathbf{C} \rightarrow \mathbf{D}$;
2. A functor $G : \mathbf{D} \rightarrow \mathbf{C}$;
3. Natural isomorphisms $\eta : \text{id}_{\mathbf{C}} \Rightarrow F \circ G$ and $\epsilon : G \circ F \Rightarrow \text{id}_{\mathbf{D}}$.

Definition 18.4 (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 $F : \mathbf{C} \rightarrow \mathbf{D}$ (the *left adjoint*);
2. A functor $G : \mathbf{D} \rightarrow \mathbf{C}$ (the *right adjoint*);
3. Natural transformations $\eta : \text{id}_{\mathbf{C}} \Rightarrow F \circ G$ and $\epsilon : G \circ F \Rightarrow \text{id}_{\mathbf{D}}$

Conditions:

1. For all objects x of \mathbf{C} , it holds that

$$F\eta_x \circ \epsilon_{Fx} = \text{id}_{Fx} \text{ and } \eta_{Gy} \circ G\epsilon_y = \text{id}_{Gy}$$

i.e. that the following diagrams commute:

$$\begin{array}{ccc} Fx & \xrightarrow{F\eta_x} & FGFx \\ & \searrow \text{id}_{Fx} & \downarrow \epsilon_{Fx} \\ & & Fx \end{array} \qquad \begin{array}{ccc} Gy & \xrightarrow{\eta_{Gy}} & GFGy \\ & \searrow \text{id}_{Gy} & \downarrow G\epsilon_y \\ & & Gy \end{array}$$

The 2-morphisms η and ϵ are called the *unit* and *counit* of the adjunction. An adjunction is called an *adjoint equivalence* if the unit and counit are natural isomorphisms.

Remark 18.5. The conditions (triangle identities) from Definition 18.4 are “hidden” in Definition 18.1 in the condition that τ be an natural isomorphism. In Section 18.6 we spell out how the two definitions are related.

18.4. 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 Y and consider the functors F and G which act as follows. Given a set X ,

$$F(X) = Y \times X$$

and

$$G(X) = \text{Hom}_{\text{Set}}(Y, X) =: X^Y.$$

Given a morphism $f : X \rightarrow X'$,

$$F(f) = f \times \text{id}_Y$$

and

$$\begin{aligned} G(f) : X^Y &\rightarrow X'^Y \\ g &\mapsto g \circ f. \end{aligned}$$

These functors are part of an adjunction

$$\begin{array}{ccc} & Y \times - & \\ \text{Set} & \begin{array}{c} \swarrow \\ \perp \\ \searrow \end{array} & \text{Set} \\ & (-)^Y & \end{array}$$

In terms of Definition 18.1, there is a natural isomorphism

$$\tau : \text{Hom}_{\text{Set}}(F(-), -) \Rightarrow \text{Hom}_{\text{Set}}(-, G(-))$$

whose component at $\langle X, Z \rangle$ is the isomorphism

$$\tau_{X,Z} : \text{Hom}_{\text{Set}}(Y \times X, Z) \rightarrow \text{Hom}_{\text{Set}}(X, Z^Y)$$

given by “partial evaluation”. Namely, given $f : Y \times X \rightarrow Z$, this is mapped by $\tau_{X,Z}$ to the function $\hat{f} : X \rightarrow Z^Y$, $x \mapsto f(-, x)$.

In terms of Definition 18.4, the component at X of the counit and unit, respectively, are

$$\begin{aligned} \eta_X : X &\rightarrow (Y \times X)^Y \\ x &\mapsto (y \mapsto \langle y, x \rangle) \end{aligned}$$

and

$$\begin{aligned} \epsilon_X : Y \times (X^Y) &\rightarrow X \\ \langle y, f \rangle &\mapsto f(y) \end{aligned}$$

18.5. Example of a “Free-Forgetful” adjunction

Another “type” of adjunction that appears frequently can be called “Free-Forgetful” adjunction. Such adjunctions are composed of a “free functor” and a “forgetful functor”. These terms are informal, but the idea is this. 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} .

For example: 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 from the category $\mathbf{Vect}_{\mathbb{R}}$ of real vector spaces to \mathbf{Set} which maps any vector space to its underlying set of vectors. On the other hand, there is a “free” construction going the other way: given a set X , we can build the “free real vector space generated by X ”. To do this, we think of the elements of X as basis vectors, and we build a vector space by taking formal finite \mathbb{R} -linear combinations of them.

In the following we will consider an example in detail where we “freely” generate a category from a directed graph.

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 5.3). There is also a functor $G : \mathbf{Cat} \rightarrow \mathbf{Grph}$ which turns a category \mathbf{C} into a graph where the set of vertices is $\text{Ob}_{\mathbf{C}}$ and there is a directed edge between vertices for every morphism in \mathbf{C} between the corresponding vertices.

Let’s first describe this adjunction via Definition 18.1. The natural isomorphism

$$\tau : \text{Hom}_{\mathbf{Cat}}(F(-), -) \rightarrow \text{Hom}_{\mathbf{Grph}}(-, G(-))$$

is the one whose component at $\langle D, \mathbf{C} \rangle$ is the isomorphism

$$\tau_{D, \mathbf{C}} : \text{Hom}_{\mathbf{Cat}}(F(D), \mathbf{C}) \rightarrow \text{Hom}_{\mathbf{Grph}}(D, G(\mathbf{C}))$$

which assigns to any functor $F : F(D) \rightarrow \mathbf{C}$ the morphism of graphs $D : G(\mathbf{C})$ given by restricting F to D and only keeping track of its action on vertices and edges (i.e., we ignore its compositional properties and think of it just as a graph morphism).

Now let’s consider this adjunction from the perspective of Definition 18.4. The component at D of the counit is the morphism of graphs

$$\eta_D : D \rightarrow G(F(D))$$

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? It’s component at \mathbf{C} is a functor

$$\epsilon_{\mathbf{C}} : F(G(\mathbf{C})) \rightarrow \mathbf{C}.$$

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 (i.e., 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 $\epsilon_{\mathbf{C}}$ 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} .

18.6. Relating the two definitions

Let’s 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 Definition 18.1 a natural isomorphism τ with components

$$\tau_{X,Y} : \text{Hom}_{\mathbf{D}}(F(X), Y) \rightarrow \text{Hom}_{\mathbf{C}}(X, G(Y)).$$

From this data we can construct the unit and counit of the adjunction as follows.

Given an object A of \mathbf{C} , we define

$$\eta_C : A \rightarrow G(F(A))$$

to be the image under $\tau_{A,F(A)}$ of $\text{id}_{F(A)} \in \text{Hom}_{\mathbf{D}}(F(A), F(A))$.

And given an object B of \mathbf{D} , we define

$$\epsilon_B : F(G(B)) \rightarrow B$$

to the the image under $\tau_{G(B),B}^{-1}$ of $\text{id}_{G(B)} \in \text{Hom}_{\mathbf{D}}(G(B), G(B))$.

Exercise 18.6. Show that if we define η and ϵ in terms of their components as above, then they do indeed define natural transformations

$$\eta : \text{id}_{\mathbf{C}} \Rightarrow G \circ F$$

and

$$\epsilon : G \circ F \Rightarrow \text{id}_{\mathbf{D}}$$

respectively. In other words, check the naturality conditions for η and ϵ .

Exercise 18.7. Show that η and ϵ , as defined above, satisfy the triangle identites stated in Definition 18.4.

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

$$\tau_{X,Y}(f) = \eta_X \circ G(f).$$

Similarly, given $g \in \text{Hom}_{\mathbf{C}}(X, G(Y))$, the inverse component is given by

$$\tau_{X,Y}^{-1}(g) = F(g) \circ \epsilon_Y.$$

Exercise 18.8. Show that $\tau_{X,Y}$ and $\tau_{X,Y}^{-1}$ are indeed functions which are inverses of each other.

Exercise 18.9. Show that the functions $\tau_{X,Y}$ do assemble to a natural transformation

$$\tau : \text{Hom}_{\mathbf{D}}(F(-), -) \rightarrow \text{Hom}_{\mathbf{C}}(-, G(-))$$

between functors $\mathbf{C}^{\text{op}} \times D \rightarrow \mathbf{Set}$.

19. Combination

to write

Contents

19.1. Products	122
19.2. Coproduct	127
19.3. Other examples	133



19.1. Products

We'll start off by recalling a familiar way of combining two sets, X and Y .

Definition 19.1 (Cartesian product of sets). Given two sets X, Y , their *cartesian product* is denoted $X \times Y$ and defined as

$$X \times Y = \{\langle x, y \rangle \mid x \in X \text{ and } y \in Y\}.$$

Example 19.2. Consider the sets $X = \{1, 2, 3, 4\}$ and $Y = \{*, †\}$. We have

$$X \times Y = \{\langle 1, * \rangle, \langle 2, * \rangle, \langle 3, * \rangle, \langle 4, * \rangle, \langle 1, † \rangle, \langle 2, † \rangle, \langle 3, † \rangle, \langle 4, † \rangle\}.$$

We can, however, also represent $X \times Y$ in a way which highlights its structure more (Fig. 19.1).

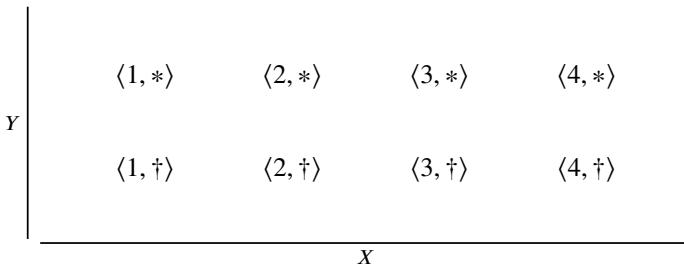


Figure 19.1.

In particular, the cartesian product comes naturally equipped with two projection maps π_1 and π_2 which map an element of $X \times Y$ to its first and second coordinate, respectively:

$$\pi_1(\langle x, y \rangle) = x \text{ and } \pi_2(\langle x, y \rangle) = y.$$

We will often depict the situation like this:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & X \times Y & \xrightarrow{\pi_2} & Y \\ & & x & \longleftarrow & y \end{array}$$

In this section, we will introduce the “categorical product”. This notion generalizes the definition of a cartesian product of sets. We will see that the cartesian product exemplifies the categorical product when we are working within the category **Set**, but that in other other categories, the categorical product can show itself quite differently!

To give an introduction to the ideas, we'll work through an example which involves the familiar cartesian product of sets, but we'll view this example through eyeglasses which will highlight the “categorical product” essence of the cartesian product.

Our introductory example is as follows. Suppose you are at an engineering conference in Switzerland, and there will be a hike as a group outing. The organizers have prepared

snacks to go. Each participant can choose a food from $X = \{a, b, c\}$ (think: apple, banana, carrot) and a drink from $Y = \{w, t\}$ (think: water, tea). Let T denote the set of participants. The choice of snacks could be organized as depicted in Fig. 19.2, i.e., each participant chooses a food, and chooses a drink. This can be described via functions $f : T \rightarrow X$ and $g : T \rightarrow Y$.

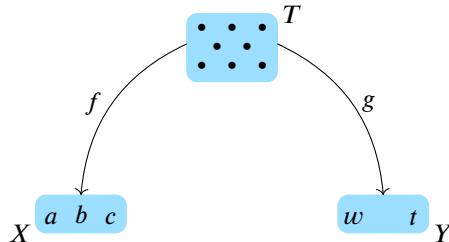


Figure 19.2.: Each participant chooses a food and a drink.

Alternatively, snacks could be pre-packaged in such a way as to allow all possible combinations of food and drink choices. This corresponds to $X \times Y$. Then the choice participants make of which lunch package they'd like is described by a single function $\phi : T \rightarrow X \times Y$, see Fig. 19.3).

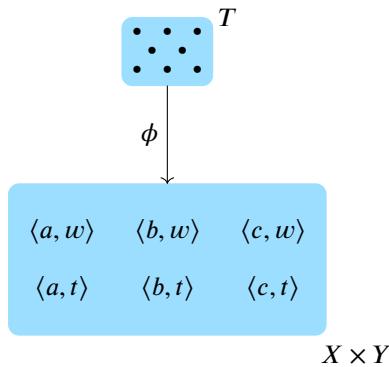


Figure 19.3.: Each participant chooses a combination of food and a drink.

Intuitively, the two situations (two choices separately, or one choice of a pre-packaged snack) are “the same” in a certain sense. In our models, we can make this precise. Specifically, if we start with the functions f and g , we can use them to build the following function:

$$\begin{aligned}\phi_{f,g} : T &\rightarrow X \times Y \\ s &\mapsto \langle f(s), g(s) \rangle.\end{aligned}$$

Furthermore, given $\phi_{f,g}$, one can recover f and g :

$$f = \phi_{f,g} \circ \pi_1 \quad \text{and} \quad g = \phi_{f,g} \circ \pi_2.$$

These two equations say that the diagram in Fig. 19.4 is commutative. The whole situation can be summarized thus: given a set T and functions $f : T \rightarrow X$ and $g : T \rightarrow Y$ as in Fig. 19.3, there is a unique function $\phi_{f,g} : T \rightarrow X \times Y$ such that the diagram Fig. 19.4 commutes. This is the general pattern for the definition of the categorical product, which we state now. It is probably helpful to read the definition together with the clarifying remarks that follow it.

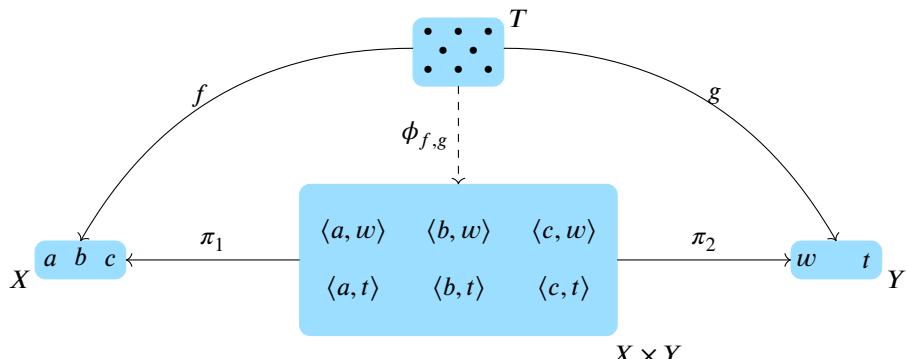


Figure 19.4.: Choosing food and drink separately is essentially the same as choosing a combination of the two.

Definition 19.3 (Categorical Product). Let \mathbf{C} be a category and let $X, Y \in \text{Ob}_{\mathbf{C}}$ be objects. The *product* of X and Y is defined by the following constituent data, satisfying the following condition.

Data:

1. an object $Z \in \text{Ob}_{\mathbf{C}}$ (this is “the product” of X and Y);
2. *projection morphisms* $\pi_1 : Z \rightarrow X$ and $\pi_2 : Z \rightarrow Y$,

Condition:

1. For any $T \in \text{Ob}_{\mathbf{C}}$ and any morphisms $f : T \rightarrow X, g : T \rightarrow Y$, there exists a *unique* morphism $\phi_{f,g} : T \rightarrow Z$ such that $f = (\phi_{f,g}) \circ \pi_1$ and $g = (\phi_{f,g}) \circ \pi_2$.

Remark 19.4. Diagrammatically, the condition above states that the diagrams of this form commute:

$$\begin{array}{ccccc}
 & & T & & \\
 & \swarrow f & \downarrow \phi_{f,g} & \searrow g & \\
 X & \xleftarrow{\pi_1} & X \times Y & \xrightarrow{\pi_2} & Y
 \end{array} \tag{19.1}$$

Remark 19.5. In the above definition, technically both Z and the projection morphisms constitute the data of “the product of X and Y ”. However, for simplicity, we usually refer only to Z as “the product”. Furthermore, we will usually use the notation $X \times Y$ to denote the product of X and Y , in place of Z . Similarly, we will usually write $f \times g$ in place of $\phi_{f,g}$. The reason we do not do this directly in the definition itself is the following. In general, for fixed X and Y , there may be several different objects Z (together with projection morphisms) that satisfy the definition of being “the product of X and Y ”. Thus, there is, technically, no such thing as “*the*” (unique) product of X and Y . However, one can prove that any two candidates which satisfy the definition of being “the product of X and Y ” will necessarily be isomorphic in a canonical manner. Thus, for simplicity, we will sometimes be slightly sloppy and speak of “the product of X and Y ” as if it were unique. In many categories there is also indeed a choice for “the product of X and Y ” that we are used to. For example, in the category **Set**, given sets X and Y , the familiar choice for “the product of X and Y ” is the cartesian product $X \times Y$. However, other representatives of the product of X and Y are possible! Example 19.7 illustrates this.

Remark 19.6. The condition in the definition of the categorical product is known as the “universal property of the product”. We will attempt to explain this naming. The stated condition involves the product Z of X and Y interacting with every possible choice of object T and every possible choice of morphisms $f : T \rightarrow X$ and $g : T \rightarrow Y$. We think of the ambient category **C** as “the universe” (or the “context”), and this condition states how the product must interact “with the whole universe”. We choose the letter “ T ” because we think of this as a “test object” (similar e.g. to how, in electrodynamics, a “test charge” is used to probe an electromagnetic field).

Example 19.7. Suppose that as a manufacturer, you want to label your products with

- A production date (8-digit code), and
- a model number (4-digit code).

Instead of two separate labels, you can make one

202101155900

where the first 8 digits represent a date, and the last 4 digits are a model number. We call this single label the *product code*. Let Z denote the set of all product codes, and consider the maps $\pi_1 : Z \rightarrow X$, and $\pi_2 : Z \rightarrow Y$ which, respectively, map a 12-digit product code to its first 8 digits and its last 4 digits. One may check that Z , together with the map π_1 and π_2 , will satisfy the definition of “the product of X and Y ”.

$$\begin{array}{ccccc} & & T & & \\ & \swarrow & \downarrow & \searrow & \\ X & \xleftarrow{\pi_1} & Z & \xrightarrow{\pi_2} & Y \\ & \text{first 8} & & \text{last 4} & \end{array}$$

However, Z is not precisely the cartesian product of X and Y (which we will call $X \times Y$). The elements of Z are 12-digit codes, while elements of $X \times Y$ are pairs $\langle x, y \rangle$ where x is

a 8-digit code and y is a 4-digit code. Since both Z and $X \times Y$ satisfy the definition of categorical product, they must, by Remark 19.5, be isomorphic. To see concretely what this isomorphism between them looks like, note that there is a unique map $\phi_{\text{first } 8, \text{last } 4}$ making the following diagram commute:

$$\begin{array}{ccccc} & & Z & & \\ & \swarrow \text{first 8} & \downarrow \phi_{\text{first } 8, \text{last } 4} & \searrow \text{last 4} & \\ X & \xleftarrow{\quad} & X \times Y & \xrightarrow{\quad} & Y \end{array}$$

Concretely, $\phi_{\text{first } 8, \text{last } 4} : Z \rightarrow X \times Y$ maps for instance

$$202101155900 \mapsto \langle 20210115, 5900 \rangle.$$

One can readily show that $\phi_{\text{first } 8, \text{last } 4}$ is an isomorphism.

Now we will take a small tour to see some examples of how the categorical product may look in categories different than the category **Set** of sets and functions.

Example 19.8. Let $m, n \in \mathbb{N}$, and draw an arrow $m \rightarrow n$ if m divides n . For instance, 6 divides 12 and hence there is an arrow $6 \rightarrow 12$. The product between any two $m, n \in \mathbb{N}$ in this category is given by the greatest common divisor.

Example 19.9. Let's consider the ordered set $\langle \mathbb{R}, \leq \rangle$, where given $x_1, x_2 \in \mathbb{R}$ we can draw an arrow $x_1 \rightarrow x_2$ if $x_1 \leq x_2$. By following the products's commutative diagram, we know that the product of x_1 and x_2 is a $z \in \mathbb{R}$ such that

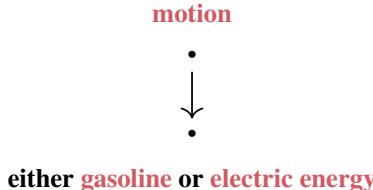
- $z \leq x_1$;
- $z \leq x_2$;
- For all $x \in \mathbb{R}$ with $x \leq x_1$ and $x \leq x_2$, we have $x \leq z$.

In other words, the product of $x_1, x_2 \in \mathbb{R}$ is given by $\min\{x_1, x_2\}$, and is also called *meet*.

Example 19.10. Let S be a set, and $X, Y \subseteq S$ subsets. We can draw an arrow $X \rightarrow Y$ if $X \subseteq Y$. By following the product's commutative diagram, it is easy to see that the product of X and Y is given by $X \cap Y$.

Example 19.11. Suppose that we are designing a vehicle, and we are thinking about choices of engine. Both electric engines and internal combustion engines can produce **motion**, but each from a different source of energy. The electric engine uses **electric energy**; the internal combustion engine uses **gasoline**. The situation is depicted in ??, using the interpretation of the arrows that we have introduced for engineering design components. Namely, the arrow from motion to gasoline represents the internal combustion engine, and its direction is to be read as follows: given the desired functionality **motion**, **internal combustion engine** provides a way of getting it using **gasoline**. The other arrow in the figure represents the component **electric engine**, and is interpreted in a similar way.

We could also consider building a hybrid vehicle, where we can obtain **motion** from **either gasoline or electric energy** (Fig. 19.6).

Figure 19.5.: Alternative ways to generate **motion**.Figure 19.6.: We can generate **motion** from either **gasoline** or **electric energy**.

Definition 19.12 (Product category). Given two categories **C** and **D**, one defines the *product category* **C × D** to be the category specified as follows:

1. *Objects*: Objects are pairs $\langle c, d \rangle$, with $c \in \mathbf{C}$ and $d \in \mathbf{D}$.
2. *Morphisms*: Morphisms are pairs of morphisms $\langle f, g \rangle : \langle c, d \rangle \rightarrow \langle c', d' \rangle$, with $f : c \rightarrow c'$, $g : d \rightarrow d'$.
3. *Composition of morphisms*: The composition of morphisms is given by composing each component of the pair separately, i.e. $\langle f, g \rangle ; \langle f', g' \rangle = \langle f ; f', g ; g' \rangle$.

Example 19.13. Consider two posets P, Q as categories. The product poset $P \times Q$ (Definition 12.1) is the product category of the two posetal categories.

19.2. Coproduct

There exists a “dual” notion to “product” that is called “coproduct”. Just like the notion of categorical product generalized the definition of the cartesian product of two sets, the categorical coproduct generalizes the definition of the *disjoint union* of two sets.

Recall that given sets X and Y , their disjoint union $X + Y$ is a set that contains a distinct copy of X and Y each. If an element is contained in both X and Y , then there will be two distinct copies of it in the disjoint union $X + Y$.

Definition 19.14 (Disjoint union of sets). The *disjoint union* (or sum) of sets X and Y is

$$X + Y = \{\langle 1, x \rangle \mid x \in X\} \cup \{\langle 2, y \rangle \mid y \in Y\}.$$

Example 19.15. Consider the sets $\{\star, \diamond\}$ and $\{\ast, \dagger\}$. Their disjoint union can be represented as in Fig. 19.7.

We can define the disjoint union of a set with itself; this corresponds to having two distinct copies of the set (Fig. 19.8).

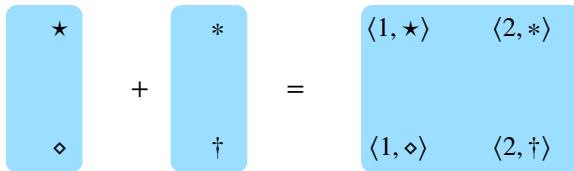


Figure 19.7.: Example of a disjoint union of sets.

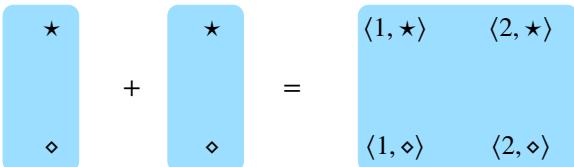


Figure 19.8.: Disjoint union of a set with itself .

As mentioned above, the disjoint union is a particular instance – in the category **Set** – of the notion of “coproduct”. We will now give the definition of a coproduct in an arbitrary category. Note that it is very similar to the definition that we gave, in the previous section, for the product – but with a few twists. Analogous remarks to those we gave following the definition of the product apply here!

Definition 19.16 (Coproduct). Let \mathbf{C} be a category and let $X, Y \in \text{Ob}_{\mathbf{C}}$ be objects. The *coproduct* of X and Y consists of the following constituent data, satisfying the following condition.

Data:

1. an object $Z \in \text{Ob}_{\mathbf{C}}$ (“the coproduct” of X and Y)
2. injection morphisms $\iota_1 : X \rightarrow Z$ and $\iota_2 : Y \rightarrow Z$

Condition:

1. For any $T \in \text{Ob}_{\mathbf{C}}$ and any morphisms $f : X \rightarrow T, g : Y \rightarrow T$, there exists a *unique* morphism $\psi_{f,g} : Z \rightarrow T$ such that $f = \iota_1 \circ \psi_{f,g}$ and $g = \iota_2 \circ \psi_{f,g}$.

Remark 19.17. Diagrammatically, the condition above states that diagrams of this form commute:

$$\begin{array}{ccccc}
 & & X & & \\
 & \nearrow f & \uparrow \psi_{f,g} & \searrow g & \\
 X & \xrightarrow{i_X} & X + Y & \xleftarrow{i_Y} & Y
 \end{array} \tag{19.2}$$

Remark 19.18. Similarly as was the case with the categorical product, “the coproduct” of X and Y is unique only “up to isomorphism”. Nevertheless, we will usually simply write $X + Y$ for “the” coproduct (in place of Z above), and we will usually write $f + g$ in place of $\psi_{f,g}$.

Example 19.19. Let’s consider two battery producers, each producing specific battery technologies. The first company produces a set $A = \{\text{LiPo}, \text{LCO}, \text{NiH2}\}$ of technologies, and the second one a set $B = \{\text{LFP}, \text{LMO}, \text{LiPo}\}$. Each technology has a specific price, belonging to a set of prices $P = \{50, 60, 70, 80\} \times \{\text{USD}\}$. We specify the price mappings for different technologies via the functions $f : A \rightarrow P$ and $g : B \rightarrow P$. A battery vendor wants to sell batteries from both producers and wants to create a battery catalogue, which needs to take into account which technology comes from which producer, to be able to distribute the earnings from the sales fairly. To this end, the disjoint union of the sets of technology is considered:

$$A + B = \{\langle 1, \text{LiPo} \rangle, \langle 1, \text{LCO} \rangle, \langle 1, \text{NiH2} \rangle, \langle 2, \text{LFP} \rangle, \langle 2, \text{LMO} \rangle, \langle 2, \text{LiPo} \rangle\}.$$

It is possible to map each technology in A, B to its own representative in $A + B$ via the so-called injection maps:

$$\begin{aligned} \iota_A &: A \rightarrow A + B \\ a &\mapsto \langle 1, a \rangle, \\ \iota_B &: B \rightarrow A + B \\ b &\mapsto \langle 2, b \rangle. \end{aligned}$$

This situation is graphically represented in Fig. 19.9, and mimics the coproduct diagram presented in Definition 19.16.

Here, the universal property says that there is a **unique** function $f + g : A + B \rightarrow P$ such that

$$\iota_A \circ (f + g) = f \text{ and } \iota_B \circ (f + g) = g.$$

If we take a $x \in A + B$ is either “from A or from B ”:

$$\text{either } \exists a \in A : x = \iota_A(a) \text{ or } \exists b \in B : x = \iota_B(b).$$

From this, we can deduce that the desired map $f + g$ is:

$$\begin{aligned} f + g &: A + B \rightarrow P \\ x &\mapsto \begin{cases} f(x), & \text{if } x = \iota_A(a), \quad a \in A, \\ g(x), & \text{if } x = \iota_B(b), \quad b \in B. \end{cases} \end{aligned}$$

This is a specific example of **Set/FinSet**, in which the coproduct is a generalization of the concept of disjoint union. Now, we could spontaneously ask ourselves: why does the union not “suffice” for the coproduct definition in **Set**? To see this, let’s consider the same situation as before, but now having the catalogue of technologies given by $A \cup B$ (Fig. 19.10). The interpretation of maps f, g does not change, and injections work as depicted. Note, however, that when asked for a map from the technology **LiPo** $\in A \cup B$, we have no notion of the company which produces it, and we are therefore unsure whether to assign it to $f(\text{LiPo}) = 50 \text{ USD}$ or $g(\text{LiPo}) = 80 \text{ USD}$. Indeed, the unique map $f + g$ required by the universal property of the coproduct cannot exist, since in case $A \cap B \neq \emptyset$, any element $x \in A \cap B$ should be simultaneously sent to $f(x)$ and $g(x)$.

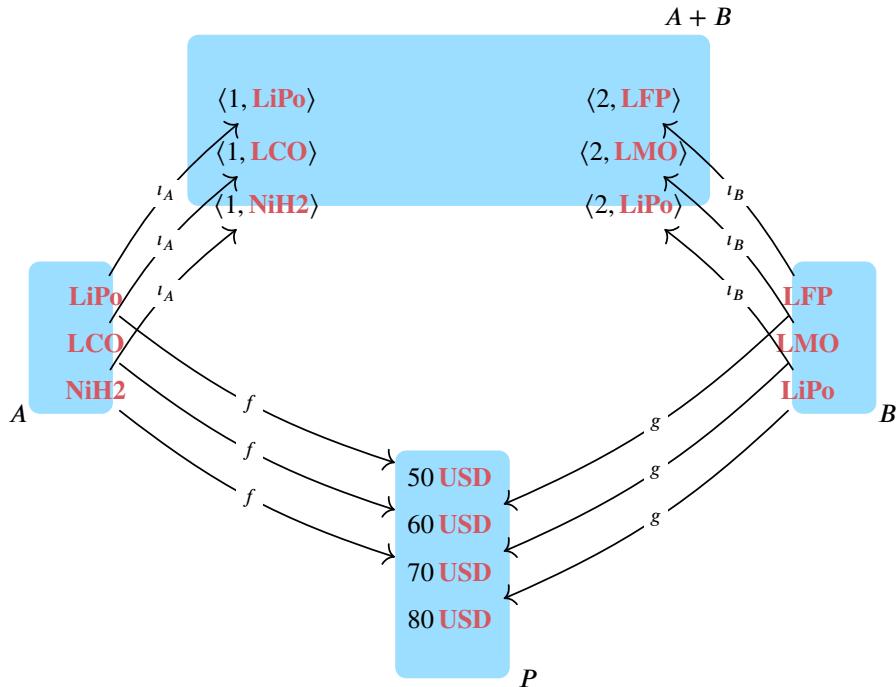


Figure 19.9.: Battery technologies, companies, prices, and a catalogue.

Example 19.20. Given $X, Y \in \text{Ob}_{\text{Rel}}$ (so X and Y are sets) their coproduct is the disjoint union $X + Y$. The disjoint union of sets comes equipped with inclusion functions $i_X : X \rightarrow X + Y$ and $i_Y : Y \rightarrow X + Y$. If we turn these functions into relations

$$\begin{aligned} R_{i_X} &\subseteq X \times (X + Y) \\ R_{i_Y} &\subseteq Y \times (X + Y). \end{aligned}$$

then these are the injection morphisms for the coproduct in **Rel**. As an aside, we note that in **Rel** products and coproducts are *both* given by the disjoint union of sets. We will see later why this is might be expected.

Example 19.21. Let $m, n \in \mathbb{N}$, and draw an arrow $m \rightarrow n$ if m divides n . For instance, 6 divides 12 and hence there is an arrow $6 \rightarrow 12$. The coproduct between any two $m, n \in \mathbb{N}$ in this category is given by the least common multiple.

Example 19.22. Let's consider the ordered set $\langle \mathbb{R}, \leq \rangle$, where given $x_1, x_2 \in \mathbb{R}$ we can draw an arrow $x_1 \rightarrow x_2$ if $x_1 \leq x_2$. By following the coproduct's commutative diagram, we know that the coproduct of x_1 and x_2 is a $z \in \mathbb{R}$ such that

- $x_1 \leq z$;
- $x_2 \leq z$;
- For all $x \in \mathbb{R}$ with $x_1 \leq x$ and $x_2 \leq x$, we have $z \leq x$.

In other words, the coproduct of $x_1, x_2 \in \mathbb{R}$ is given by $\max\{x_1, x_2\}$, and is also called *join*.

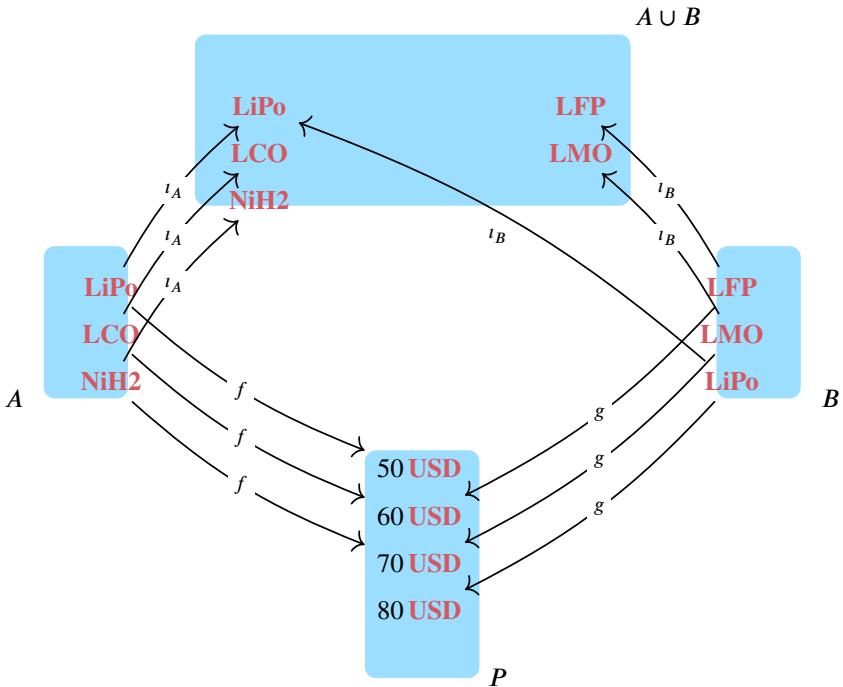


Figure 19.10.: Example of why the union is not the coproduct in **Set**.

Example 19.23. Let S be a set, and $X, Y \subseteq S$ subsets. We can draw an arrow $X \rightarrow Y$ if $X \subseteq Y$. By following the coproduct's commutative diagram, it is easy to see that the coproduct of X and Y is given by $X \cup Y$, i.e., the “smallest” set containing both X and Y .

Example 19.24. We can define a category **Vect**, composed of:

- *Objects*: vector spaces;
- *Morphisms*: linear maps;
- *Identity morphisms*: identity maps;
- *Composition*: composition of linear maps.

It is a good exercise to prove that **Vect** really forms a category. In the following, we want to look at the coproduct in **Vect**. It is given by the *direct sum* of vector spaces. Recall that given vector spaces V and W , their direct sum is the set

$$V \oplus W := \{\langle v, w \rangle \mid v \in V, w \in W\},$$

equipped with a notion of addition and scalar multiplication derived component-wise from V and W . For addition, this means that given $\langle v_1, w_1 \rangle, \langle v_2, w_2 \rangle \in V \oplus W$, their sum in $V \oplus W$ is

$$\langle v_1, w_1 \rangle + \langle v_2, w_2 \rangle := \langle v_1 + v_2, w_1 + w_2 \rangle.$$

The injection morphisms for the coproduct are given by:

$$\begin{aligned}\iota_V : V &\rightarrow V \oplus W \\ v &\mapsto \langle v, 0_W \rangle, \\ \iota_W : W &\rightarrow V \oplus W \\ w &\mapsto \langle 0_V, w \rangle,\end{aligned}$$

where 0_V and 0_W represent the zero vectors in V and W . Let's now look at the universal property in this case, by considering any vector space $U \in \text{Ob}_{\text{Vect}}$, and linear maps $S : V \rightarrow U$, $T : W \rightarrow U$. The universal property says that we need a unique linear map $S + T : V \oplus W \rightarrow U$ such that $S = \iota_V ; h$ and $T = \iota_W ; g$. By taking any $\langle v, w \rangle \in V \oplus W$, we can write:

$$\begin{aligned}h(\langle v, w \rangle) &= h(\langle v, 0_W \rangle + \langle 0_V, w \rangle) \\ &= h(\iota_V(v) + \iota_W(w)) \\ &= h(\iota_V(v)) + h(\iota_W(w)) \quad (h \text{ is linear}) \\ &= (\iota_V ; h)(v) + (\iota_W ; h)(w) \\ &\stackrel{!}{=} Sv + Tw.\end{aligned}$$

We can hence write the map $S + T$ as

$$\begin{aligned}S + T : V \oplus W &\rightarrow U \\ \langle v, w \rangle &\mapsto Sv + Tw.\end{aligned}$$

Example 19.25 (Adapted from [43]). We can define a category of graphs **Grph**. Objects of this category are graphs $G = \langle V, A, s, t \rangle$, composed of:

- A set of *vertices* V ;
- A set of *arrows* A ;
- A *source* function $s : A \rightarrow V$, mapping each arrow to its source vertex;
- A *target* function $t : A \rightarrow V$, mapping each arrow to its target vertex.

Morphisms are *graph homomorphisms*. Given graphs $G = \langle V, A, s, t \rangle$ and $G' = \langle V', A', s', t' \rangle$, a graph homomorphism $f : G \rightarrow G'$ is given by maps $f_0 : V \rightarrow V'$ and $f_1 : A \rightarrow A'$, such that the following diagrams commute:

$$\begin{array}{ccc}A & \xrightarrow{f_1} & A' \\ \downarrow s & & \downarrow s' \\ V & \xrightarrow{f_0} & V'\end{array} \qquad \begin{array}{ccc}A & \xrightarrow{f_1} & A' \\ \downarrow t & & \downarrow t' \\ V & \xrightarrow{f_0} & V'\end{array}$$

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 $f_0(v_1)$ and $f_0(v_2)$ have to be connected via an arrow $f_1(a)$. We are now ready to define the coproduct in **Grph**. Given two graphs $G = \langle V, A, s, t \rangle$ and $G' = \langle V', A', s', t' \rangle$, their coproduct is a graph

$$G + G' = \langle V + V', A + A', s + s', t + t' \rangle.$$

In $G + G'$, an arrow connects v_1 to v_2 if:

- $v_1, v_2 \in V$ or $v_1, v_2 \in V'$, i.e., if both vertices belong to the same graph, and
- an arrow between from v_1 to v_2 exists in G or G' .

Given $s : A \rightarrow V$ and $s' : A' \rightarrow V'$, we have:

$$s + s' : A + A' \rightarrow V + V'$$

$$x \mapsto \begin{cases} s(x) & \text{if } x \in A \\ s'(x) & \text{if } x \in A'. \end{cases}$$

and

$$t + t' : A + A' \rightarrow V + V'$$

$$x \mapsto \begin{cases} t(x) & \text{if } x \in A \\ t'(x) & \text{if } x \in A'. \end{cases}$$

This is nicely graphically representable. Consider two graphs as in Fig. 19.11.

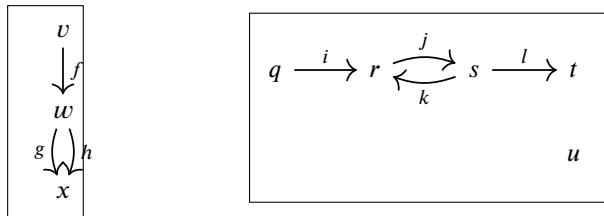


Figure 19.11.: Example of graphs for which we want to consider the coproduct.

Their coproduct is a graph including the “disjoint union” of both original graphs, without connecting them (Fig. 19.12).

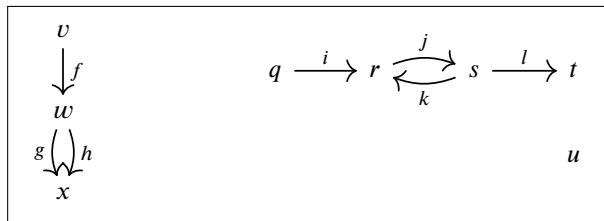


Figure 19.12.: Example of coproduct of graphs.

19.3. Other examples

19.3.1. Product and coproduct for power set

Example 19.10 and Example 19.23 are specific instances of the power set lattice. Given a set S , its power set $\mathcal{P}(S)$ (the set of all subsets) is a lattice where, given $A, B \in \mathcal{P}(S)$:

- Order is given by inclusion, i.e. $A \leq B \doteq A \subseteq B$;
- The meet is given by the union of sets, i.e. $A \vee B \doteq A \cup B$;

- The join is given by the intersection of sets, i.e. $A \wedge b \doteq A \cap B$;
- The top element is the set S itself, i.e. $\top = S$;
- The bottom element is the emptyset, i.e. $\perp = \emptyset$.

The Hasse diagram reported in Fig. 19.13 illustrates the structure of the power set lattice for three sets $A, B, C \in \mathcal{P}(S)$.

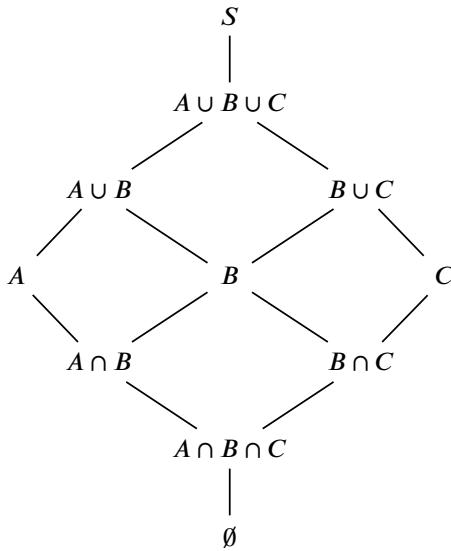


Figure 19.13.

As previously discovered, the lattice can be seen as a category. In this category, the meet \wedge is the product, and the join \vee is the coproduct. Specifically, for $A, B \subseteq S$ the product corresponds to $A \cap B$, and the projection maps $\pi_A : A \cap B \rightarrow A$ and $\pi_B : A \cap B \rightarrow B$ simply state the inclusions of $A \cap B$ in A and B . Similarly, the coproduct corresponds to $A \cup B$, and the injection maps $\iota_A : A \rightarrow A \cup B$ and $\iota_B : B \rightarrow A \cup B$ simply state the inclusion of A, B in $A \cup B$.

Product and coproduct for logical sequents

Add here content of Andrea's slides

20. Counter-examples

to write

Contents

20.1. Not quite categories	136
20.2. Not quite functors	136



20.1. Not quite categories

20.1.1. Failures because identities do not exist

Example of discrete time systems without pass through

20.1.2. Failures because composition is not associative

Add artificial example created by David

20.2. Not quite functors

Part B.

Monotone Co-Design

21. Design

to write

Contents

21.1. What is “design”?	140
21.2. What is “co-design”?	140
21.3. Basic concepts of formal engineering design	142
21.4. Queries in design	144



This chapter introduces basic concepts of engineering design, and describes what are the design queries we want to answer.

21.1. 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’s¹ *The sciences of the artificial* ([40], 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 to create 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.

21.2. 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.

21.2.1. “Co” for “compositional”

The first meaning has to do with composition:

¹Hebert A. Simon (1916-2001). Winner of the 1978 Nobel Prize in Economics.

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**.

21.2.2. “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 Conway² first observed for software systems:

«Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.»

This “mirroring” hypothesis between system and organization was explored formally and found to hold [32]. 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.

²John 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.

21.2.3. “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.»

21.2.4. “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.

21.3. Basic concepts of formal engineering design

We will informally introduce some of the basic nomenclature of engineering design [1, 49]. 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*, 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 21.1. 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 21.2. 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* [49] 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 I is the set over which we are optimizing; an implementation $i \in I$ is a particular point in that set (Fig. 21.1).

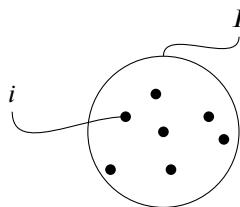


Figure 21.1.: An *implementation* i is a particular point in the implementation space I .

Make cute implementation set as in the definition of DPI

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. 21.2).

Make cute diagrams as in the definition of DPI



Figure 21.2.: 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*, one usually means 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.

JL: I think here we might want to be careful... people (and in particular mathematicians) maybe mean lots of different things by ‘abstraction’ and here it is being used in a very specific sense... perhaps we don’t say ‘one usually means’... or perhaps there is a different word which fits?

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 21.3. 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.

JL: Would be nice to also give an example of something that is *not* compositional

21.4. Queries in design

Suppose that we have a model with a functionality space \mathcal{F} , a requirements space \mathcal{R} , and an implementation space \mathcal{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 $i \in \mathcal{I}$, functionality $f \in \mathcal{F}$, requirements $r \in \mathcal{R}$, 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 $f \in \mathcal{F}$ and maximum allowed requirements $r \in \mathcal{R}$, determine if there is a an implementation $i \in \mathcal{I}$ 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 (FixFunMinReq). Given a certain functionality $f \in F$, find the set of “minimal” resources in R that are needed to realize it (along with the implementations), or provide a proof that there are none.

Dually, we can ask, fixed the resources available, what are the functionalities that can be required.

Problem (FixReqMinFun). Given a certain requirement $r \in R$, find the set of “maximal” functionalities in that can be realized it (along with the implementations), or provide a proof that there are none.

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.

22. Design problems

to write

Contents

22.1. Design Problems	148
22.2. Querying a DPI	156
22.3. Co-design problems	157
22.4. Discussion of related work	160



22.1. Design Problems

We start by defining a “design problem with implementation”, which is a tuple of “**functional**ity space”, “**implementation** space”, and “**resources** space”, together with two maps that describe the feasibility relations between these three spaces (Fig. 22.1).

Definition 22.1. A *design problem with implementation* (DPI) is a tuple

$$\langle \mathcal{F}, \mathcal{R}, \mathcal{I}, \text{prov}, \text{req} \rangle$$

where:

- \mathcal{F} is a poset, called *functional*ity space;
- \mathcal{R} is a poset, called *requirements* space;
- \mathcal{I} is a set, called *implementation* space;
- the map $\text{prov} : \mathcal{I} \rightarrow \mathcal{F}$ maps an implementation to the functionality it provides;
- the map $\text{req} : \mathcal{I} \rightarrow \mathcal{R}$ maps an implementation to the resources it requires.

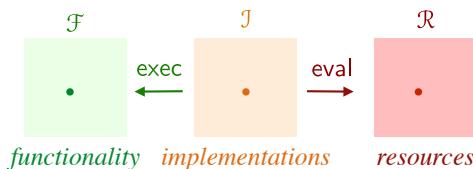


Figure 22.1.

Redo figure with consistent style, change resources to requirements, eval/exec to requires/provides.

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. 22.2).

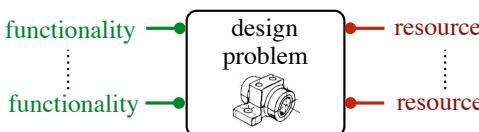


Figure 22.2.

Redo figure with consistent style.

This means that the functionality and resources spaces can be factorized in n_f and n_r components: $\mathcal{F} = \prod_{i=1}^{n_f} \pi_i \mathcal{F}_i$, $\mathcal{R} = \prod_{j=1}^{n_r} \pi_j \mathcal{R}_j$, where “ π_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 \mathcal{F} (respectively, \mathcal{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.

Example 22.2 (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 [\\$]**, the **mass [g]**, the input **voltage [V]**, and the input **current [A]**. The map $\text{prov} : \mathcal{I} \rightarrow \mathcal{F}$ assigns to each motor its functionality, and the map $\text{req} : \mathcal{I} \rightarrow \mathcal{R}$ assigns to each motor the resources it needs (Fig. 22.3).



Figure 22.3.

Redo figure with consistent style

Example 22.3 (Chassis design). Suppose we need to choose a chassis for a robot (Fig. 22.4). The implementation space \mathcal{I} 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 [\\$]** of the chassis; the total mass; and, for each motor to be placed in the chassis, the required **speed [rad/s]** and **torque [Nm]**.

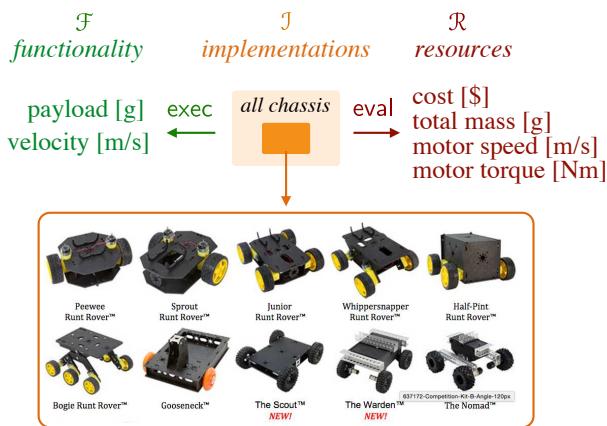


Figure 22.4.

Redo figure with consistent style

22.1.1. Mechatronics

Many mechanisms can be readily modeled as relations between a provided functionality and required resources.

Example 22.4. The **functionality** of a DC motor (Fig. 22.5) is to provide a certain **speed** and **torque**, and the **resources** are **current** and **voltage**.



Figure 22.5.

Example 22.5. A gearbox (Fig. 22.6) 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 $\tau_i \omega_i \geq \gamma \tau_o \omega_o$.



Figure 22.6.

Example 22.6. Propellers (Fig. 22.7) generate **thrust** given a certain **torque** and **speed**.

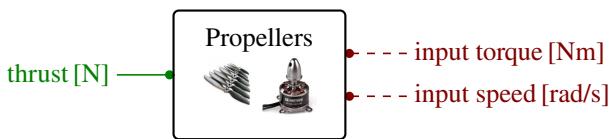


Figure 22.7.

Example 22.7. A *crank-rocker* (Fig. 22.8) converts **rotational motion** into a **rocking motion**.

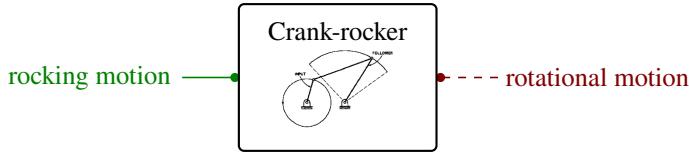


Figure 22.8.

22.1.2. Geometrical constraints

Geometrical constraints are examples of constraints that are easily recognized as monotone, but possibly hard to write down in closed form.

Example 22.8 (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. 22.9). 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 [31]. 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.



Figure 22.9.

Add pics as res/fun

22.1.3. 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 22.9. (SLAM) One issue with particle-filter-based estimation procedures, such as the ones used in the popular GMapping [22] method, 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** ($1 - \text{probability of failure}$), the **accuracy**, and the **number of particles** (Fig. 22.10).

Example 22.10. (Stereo reconstruction) Progressive reconstruction system (e.g., [30]), 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. 22.11). A similar relation characterizes any anytime algorithms in other domains, such as robot motion planning.

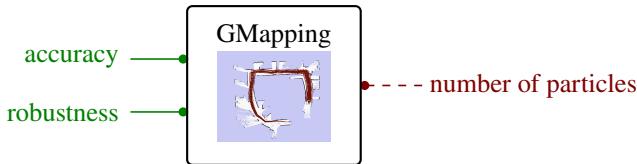


Figure 22.10.

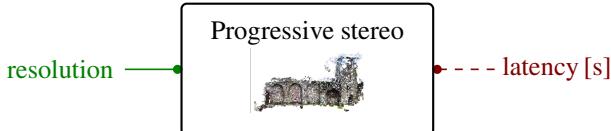


Figure 22.11.

Example 22.11. 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 [35, 50].

Reproduce plot from paper.

22.1.4. Communication

Example 22.12 (Transducers). Any type of "transducer" that bridges between different mediums can be modeled as a DP. For example, an access point (Fig. 22.12) provides the "wireless access" functionality, and requires that the infrastructure provides the "Ethernet access" resource.



Figure 22.12.

Example 22.13 (Wireless link). The basic functionality of a wireless link is to provide a certain bandwidth (Fig. 22.13). Further refinements could include 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 $\mathbb{R}^{[Hz]}$. Thus the resources space is $R = \mathbb{R}^{[W]} \times \text{intervals}(\mathbb{R}^{[Hz]})$.

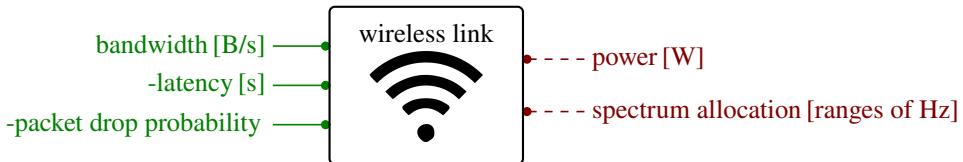


Figure 22.13.

22.1.5. 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 22.14. 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, one has to increase either the speed of an agent or the number of agents (Fig. 22.14b).

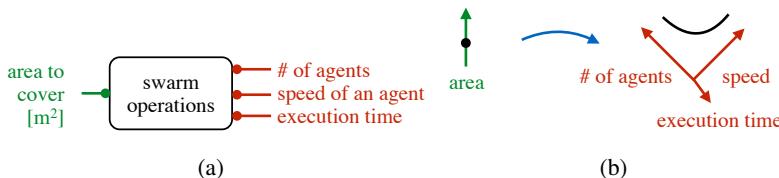


Figure 22.14.

22.1.6. LQG Control

Write short summary

22.1.7. Computation

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.

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 [45, 28, Chapter 3], the application is represented as a graph of procedures that need to be allocated on a network of processors.

Make the three small wrapped figures as 3 subfigures in the same figures

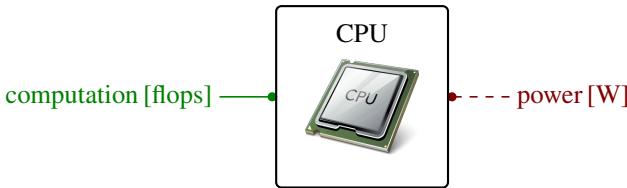
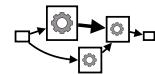
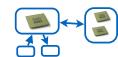


Figure 22.15.

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 $\langle n_1, n_2 \rangle$ in A_2 , the edge $\langle \varphi(n_1), \varphi(n_2) \rangle$ has equal or larger message size.

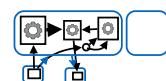


Define a *resource graph* as a graph where each node represents a processor, and each edge represents a network link. Each node is labeled by the processor capacity [flops]. Each edge is labeled by latency [s] and bandwidth [B/s]. There is a partial order on resources graph as well: it holds that $R_1 \leq R_2$ 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:

$$T^* = T^*(A, R).$$



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.

Example 22.15. Svorenova et al. [48] 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

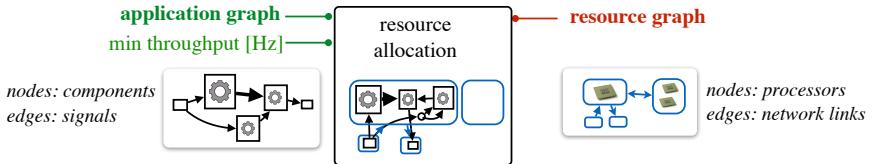


Figure 22.16.

power, sensing power, and sensor accuracy.

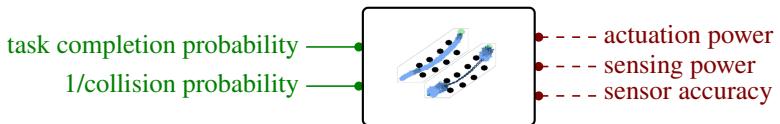


Figure 22.17.

Example 22.16. Nardi et al. [50] 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 22.18.

22.1.8. 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. 22.19). Given a task, there is a minimal antichain in the resources poset that describes the possible trade-offs (e.g., compensating lousier sensors with more computation).

The poset structure arises naturally: for example, in the *sensor lattice* [26], 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 [36] 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 (e.g., a compass plus a contact

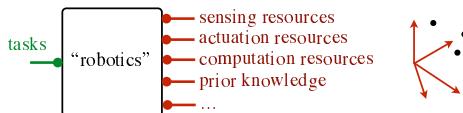


Figure 22.19.:

re do in tikz

sensor allow to “drive North until a wall is hit”). The effect of each primitive is modeled as an operator on the 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 (e.g., global localization), and describe a dominance relation (partial order) of primitives. Other works have focused on minimizing the complexity of the controller. Egerstedt [17] 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 [41] studies the relation between the **performance of a visual task**, and the **minimal representation** that is needed to perform that task.

Example 22.17 (Hoare logic).

to write

22.2. Querying a DPI

A DP is a model that induces a family of optimization problems.

Connect with discussion at beginning of the part.

One query is “Given a lower bound on the functionality f , what are the implementations that have minimal resources usage?” (Fig. 22.20).

Problem (FixFunMinReq). Given $f \in F$, find the implementations in I that realize the functionality f (or higher) with minimal resources, or provide a proof that there are none:

$$\left\{ \begin{array}{l} \text{using } i \in I, \\ \text{Min}_{\leq_R} r, \\ \text{s.t. } r = \text{req}(i), \\ f \leq_F \text{prov}(i). \end{array} \right. \quad (22.1)$$

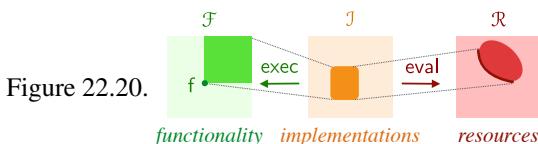


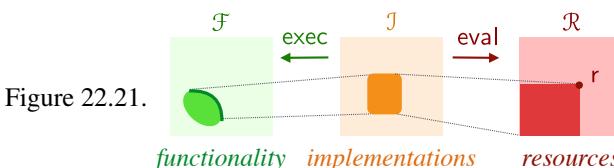
Figure 22.20.

Remark 22.18 (Minimal vs least solutions). Note the use of “ Min_{\leq_R} ” in Eq. (22.1), which indicates the set of minimal (non-dominated) elements according to \leq_R , rather than “ \min_{\leq_R} ”, which would presume the existence of a 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 $R \in \mathcal{A}R$. We will see that this formalization allows an elegant way to treat multi-objective optimization. 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 resources usage, and then maximize the functionality provided (Fig. 22.21). The formulation is entirely dual, in the sense that it is obtained from Eq. (22.1) by swapping Min with Max, F with R , and prov with req .

Problem (FixResMinFun). Given $r \in R$, find the implementations in I that requires r (or lower) and provide the maximal functionality, or provide a proof that there are none:

$$\left\{ \begin{array}{ll} \text{using} & i \in I, \\ \text{Max}_{\leq_F} & f, \\ \text{s.t.} & f = \text{prov}(i), \\ & r \geq_R \text{req}(i). \end{array} \right. \quad (22.2)$$



22.3. Co-design problems

A “co-design problem” will be defined as a multigraph of design problems. Graphically, one is allowed to connect only edges of different color, and of the same type. This interconnection is indicated with the symbol “ \preceq ” in a rounded box (Fig. 22.22).

$$\boxed{\textcolor{red}{r}_1 \textcolor{green}{\circlearrowleft} \textcolor{green}{f}_2} \equiv \textcolor{red}{r}_1 \preceq \textcolor{green}{f}_2$$

Figure 22.22.

The semantics of the interconnection is that the resources required by the first DPI are provided by the second DPI. This is a partial order inequality constraint of the type $r_1 \preceq f_2$.

Definition 22.19. A *Co-Design Problem with Implementation* (CDPI) is a tuple $\langle F, R, \langle V, E \rangle \rangle$, where F and R are two posets, and $\langle V, E \rangle$ is a multigraph of DPIs. Each node $v \in V$ is a DPI $v = \langle F_v, R_v, I_v, \text{prov}_v, \text{req}_v \rangle$. An edge $e \in E$ is a tuple $e = \langle \langle v_1, i_1 \rangle, \langle v_2, j_2 \rangle \rangle$,

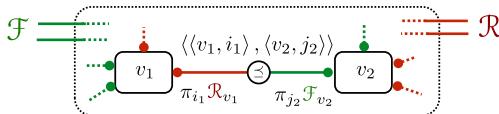


Figure 22.23.

where $v_1, v_2 \in \mathcal{V}$ 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 $\pi_{i_1} \mathbf{R}_{v_1} = \pi_{j_2} \mathbf{F}_{v_2}$ (Fig. 22.23).

A CDPI is equivalent to a DPI with an implementation space \mathcal{I} that is a subset of the product $\prod_{v \in \mathcal{V}} \mathcal{I}_v$, and contains only the tuples that satisfy the co-design constraints. An implementation tuple $i \in \prod_{v \in \mathcal{V}} \mathcal{I}_v$ belongs to \mathcal{I} iff it respects all functionality–resources constraints on the edges, in the sense that, for all edges $\langle(v_1, i_1), (v_2, j_2)\rangle$ in \mathcal{E} , it holds that

$$\pi_{i_1} \mathbf{req}_{v_1}(\pi_{v_1} i) \leq \pi_{j_2} \mathbf{prov}_{v_2}(\pi_{v_2} i).$$

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 v , let \mathbf{UF}_v and \mathbf{UR}_v 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: $\mathbf{F} = \prod_{v \in \mathcal{V}} \prod_{j \in \mathbf{UF}_v} \pi_j \mathbf{F}_v$ and $\mathbf{R} = \prod_{v \in \mathcal{V}} \prod_{i \in \mathbf{UR}_v} \pi_i \mathbf{R}_v$. The maps \mathbf{prov} , \mathbf{req} return the values of the unconnected functionality and resources:

$$\begin{aligned} \mathbf{prov} : i &\mapsto \prod_{v \in \mathcal{V}} \prod_{j \in \mathbf{UF}_v} \pi_j \mathbf{prov}_v(\pi_v i), \\ \mathbf{req} : i &\mapsto \prod_{v \in \mathcal{V}} \prod_{i \in \mathbf{UR}_v} \pi_i \mathbf{req}_v(\pi_v i). \end{aligned}$$

Example 22.20. Consider the co-design of chassis (Example 22.3) plus motor (Example 22.2). 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. 22.24).



Figure 22.24.

For the chassis (Fig. 22.25), the provided functionality is parameterized by the **mass** of the payload and the platform **velocity**. The required resources include the **cost**, **total mass**, and what the chassis needs from its motor(s), such as **speed** and **torque**.

The two design problem can be connected at the edges for torque and speed (Fig. 22.26). The semantics is that the motor needs to have *at least* the given torque and speed.

Resources can be summed together using a trivial DP corresponding to the map $h : \langle f_1, f_2 \rangle \mapsto \{f_1 + f_2\}$ (Fig. 22.27).



Figure 22.25.

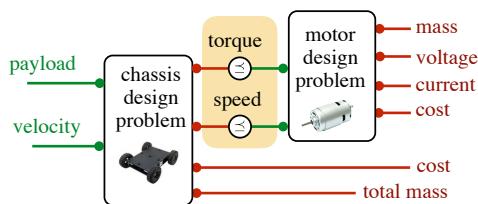


Figure 22.26.

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. 22.28).

This formalism makes it easy to abstract away the details in which we are not interested. Once a diagram like Fig. 22.28 is obtained, we can draw a box around it and consider the abstracted problem (Fig. 22.29).

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. 22.30).

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. 22.31).

Relations such as **current×voltage** \leq **power required** and **power×endurance** \leq **energy required** can be modeled by a trivial “multiplication” DPI (Fig. 22.32).

We can connect these DPs to obtain a co-design problem with functionality **voltage**, **current**, **endurance** and resources **mass** and **cost** (Fig. 22.33).

Draw a box around the diagram, and call it “MCB+PSU”; then interconnect it with the “chassis+motor” diagram in Fig. 22.34.

We can further abstract away the diagram in Fig. 22.34 as a “mobility+power” CDPI, as in Fig. 22.35. The formalism allows to consider **mass** and **cost** as independent resources, meaning that we wish to obtain the Pareto frontier for the minimal resources. Of course, one 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.

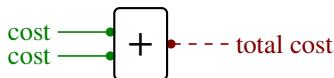


Figure 22.27.

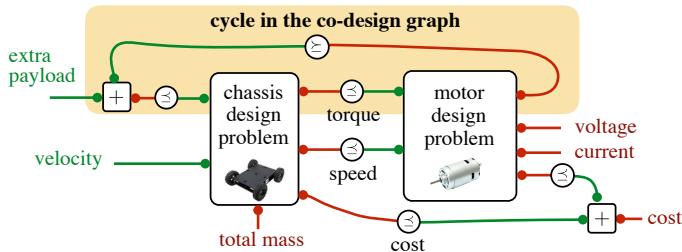


Figure 22.28.

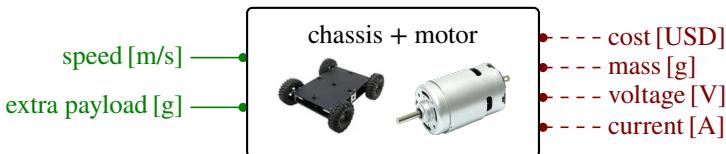


Figure 22.29.

22.4. Discussion of related work

22.4.1. Theory of design

Modern engineering has long since recognized the two ideas of modularity and hierarchical decomposition, yet there exists no general quantitative theory of design that is applicable to different domains. Most of the works in the theory of design literature study abstractions that are meant to be useful for a human designer, rather than an automated system. For example, a *function structure* diagram [37, p. 32] decomposes the function of a system in subsystems that exchange energy, materials, and signals, but it is not a formal representation. From the point of view of the theory of design, the contribution of this work is that the *design problem* abstraction developed, where one takes functionality and resources as the interfaces for the subsystems, is at the same time (1) mathematically precise; (2) intuitive to understand; and (3) leads to tractable optimization problems.

This work also provides a clear answer to one long-standing issue in the theory of design: the inter-dependence between subsystems, (i.e., cycles in the co-design graph). Consider, as an example, Suh's theory of *axiomatic design* [47], in which the first "axiom" is to keep the design requirements orthogonal (i.e., do not introduce cycles). This work shows that it is possible to deal elegantly with recursive constraints.



Figure 22.30.

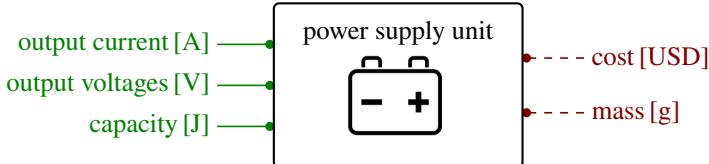


Figure 22.31.

22.4.2. Partial Order Programming

In “Partial Order Programming” [38] Parker studies a hierarchy of optimization problems that can be represented as a set of partial order constraints. The main interest is to study partial order constraints as the means to define the semantics of programming languages and for declarative approaches to knowledge representation.

In Parker’s hierarchy, MCDPs are most related to the class of problems called *continuous monotone partial order program* (CMPOP). CMPOPs are the least specific class of problems studied by Parker for which it is possible to obtain existence results and a systematic solution procedure. MCDPs subsume CMPOPs. A CMPOP is an MCDP where: 1) All functionality and resources belong to the same poset \mathcal{P} ($\mathcal{F}_v = \mathcal{R}_v = \mathcal{P}$); 2) Each functionality/resource relation is a simple map, rather than a multi-valued relation; 3) There are no dangling functionality edges in the co-design diagram ($F = 1$).

In a MCDP, each DP is described by a Scott continuous map $h : \mathcal{F} \rightarrow \mathcal{A} \mathcal{R}$ which maps one functionality to a minimal set of resources. By contrast, in a CMPOP an operator corresponds to a Scott continuous map $h : \mathcal{F} \rightarrow \mathcal{R}$. The consequence is that a CMPOP has a unique solution [38, Theorem 8], while an MCDP can have multiple minimal solutions (or none at all).

22.4.3. Abstract interpretation

The methods used from order theory are the same used in the field of *abstract interpretation* [14]. In that field, the least fixed point semantics arises from problems such as computing the sets of reachable states. Given a starting state, one is interested to find a subset of states that is closed under the dynamics (i.e., a fixed point), and that is the smallest that contains the given initial state (i.e., a *least* fixed point). Reachability and other properties lead to considering systems of equation of the form

$$x_i = \varphi_i(x_1, \dots, x_i, \dots, x_n), \quad i = 1, \dots, n, \quad (22.3)$$

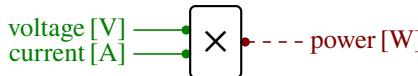


Figure 22.32.

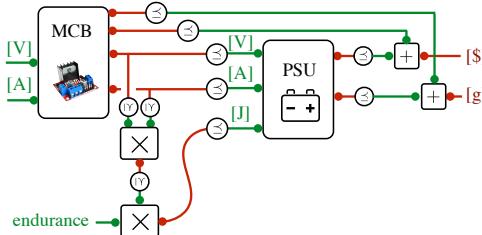


Figure 22.33.

where each value of the index i is for a control point of the program, and φ_i are Scott continuous functions on the abstract lattice that represents the properties of the program. In the simplest case, each x_i could represent intervals that a variable could assume at step i . By applying the iterations, one finds which properties can be inferred to be valid at each step.

We can repeat the same considerations we did for Parker's CMPOPs vs MCDPs. In particular, in MCDP we deal with multi-valued maps, and there is more than one solution.

In the field of abstract interpretation much work has been done towards optimizing the rate of convergence. The order of evaluation in Eq. (22.3) does not matter. Asynchronous and “chaotic” iterations were proposed early [13] and are still object of investigation [4]. To speed up convergence, the so called “widening” and “narrowing” operators are used [12]. The ideas of chaotic iteration, widening, narrowing, are not immediately applicable to MCDPs, but it is a promising research direction.

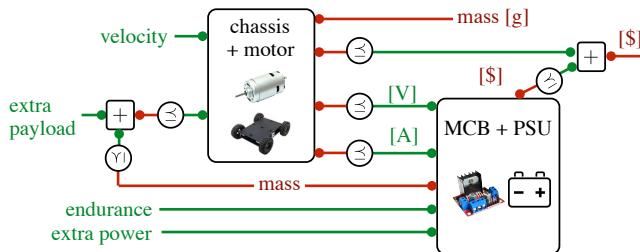


Figure 22.34.

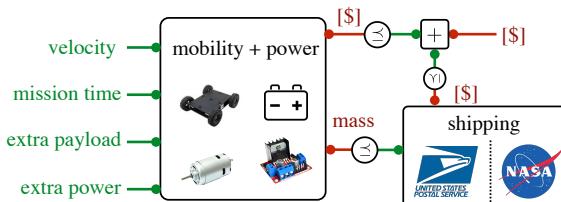


Figure 22.35.

23. Feasibility

to write

Contents

23.1. Design problems as monotone maps	166
23.2. Series composition of design problems	168
23.3. The category of design problems	172
23.4. DP Isomorphisms	173



23.1. Design problems as monotone maps

A DPI (Definition 22.1) 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 DPI more compactly as a DP.

Definition 23.1 (Design Problem). A design problem (DP) is a tuple $\langle \mathbf{F}, \mathbf{R}, d \rangle$, where \mathbf{F}, \mathbf{R} are posets and d is a monotone map of the form

$$d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}.$$

We represent it by an arrow $d : \mathbf{F} \leftrightarrow \mathbf{R}$.

Intended semantics When we consider a design problem $d : \mathbf{F} \leftrightarrow \mathbf{R}$, we imagine the poset \mathbf{F} to represent the functionality to be provided, while the poset \mathbf{R} represents the resources required. The object $d : \mathbf{F} \leftrightarrow \mathbf{R}$ is a relation that describes which combinations of functionality and resources are feasible: for each $f^* \in \mathbf{F}^{\text{op}}$ and $r \in \mathbf{R}$, $d(f^*, r)$ is a truth value, \top or \perp , which we call the *feasibility of f given r* . The value $d(f^*, r)$ is the answer to the question “is the functionality f feasible with resources r ?”.

This is the basic fact of life in engineering: there is a price to pay for everything, and there are trade-offs to be made.

The monotonicity of d represents the two following assumptions:

1. If f is feasible with r , then any $f' \leq_{\mathbf{F}} f$ is feasible with r .
2. If f is feasible with r , then f is feasible with any $r' \geq_{\mathbf{R}} r$.

Example 23.2. Imagine a truck to be driving at constant speed on a straight street. If it can cover 100 miles with 5 gallons of gasoline, it can also cover 80 miles with it. Furthermore, it will be able to cover the 100 miles also with 10 gallons of gasoline.

A design problem d will satisfy these conditions if and only if it is represented by a monotone function.

Definition 23.3 (Feasible set of a design problem). We define the *feasible set* K_d of a design problem $d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ as the subset of $\mathbf{F}^{\text{op}} \times \mathbf{R}$ for which d is the *indicator function*, that is

$$K_d = \{ \langle f^*, r \rangle \in \mathbf{F}^{\text{op}} \times \mathbf{R} \mid d(f^*, r) = \top \}.$$

Remark 23.4. The set K_d is always an upper set (Definition 11.17). In fact, another way to define a design problem is to declare it as “an upper set in $\mathbf{F}^{\text{op}} \times \mathbf{R}$ ”. This is simpler than declaring it as “a monotone map to \mathbf{Bool} ”. However, the definition as a monotone map will lend very easily to further generalization.

The Boolean-valued design problems we are considering here do not distinguish between particular implementations: they only tell us if *any* implementation or solution exists for given functionality and resources. We will define Set-enriched design problems in Section 32.1, which directly generalize Boolean-valued design problems and do distinguish between particular implementations.

Diagrammatic notation We represent design problems using a diagrammatic notation. One design problem $d : \mathbf{F} \leftrightarrow \mathbf{R}$ is represented as a box with functionality \mathbf{F} on the *left* and resources \mathbf{R} on the *right* (Fig. 23.1). We will connect these diagrams.



Figure 23.1.: Diagrammatic representation of a design problem.

Example 23.5. 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. 23.2.

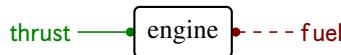


Figure 23.2.: Diagram of the engine design problem.

Concretely, “engine” is represented as a monotone map

$$\text{engine} : \mathbf{thrust}^{\text{op}} \times \mathbf{fuel} \rightarrow_{\text{Pos}} \mathbf{Bool}. \quad (23.1)$$

Assuming that the posets **fuel**, $\mathbf{thrust}^{\text{op}}$ 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 ?“:

$$\begin{array}{c} \text{Fuel} \\ \begin{matrix} r_1 = 0 & r_2 & r_3 & \dots & r_m \\ \left(\begin{array}{ccccc} 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & & 1 \\ 0 & 1 & 1 & & 1 \\ \ddots & & & \ddots & \\ 1 & 1 & 1 & & 1 \end{array} \right) \end{matrix} \end{array} \quad (23.2)$$

Thrust^{op}

Suppose we have tested or are given the performance data of a few different engines, i.e., 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. 23.3.

Note that the shaded regions cover the feasible solution set. This feasible solution set is always an *upper set* (Definition 11.17) in $\mathbf{thrust}^{\text{op}} \times \mathbf{fuel}$, 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 in Chapter 48, when discussing how to compute optimal solutions of design problems.

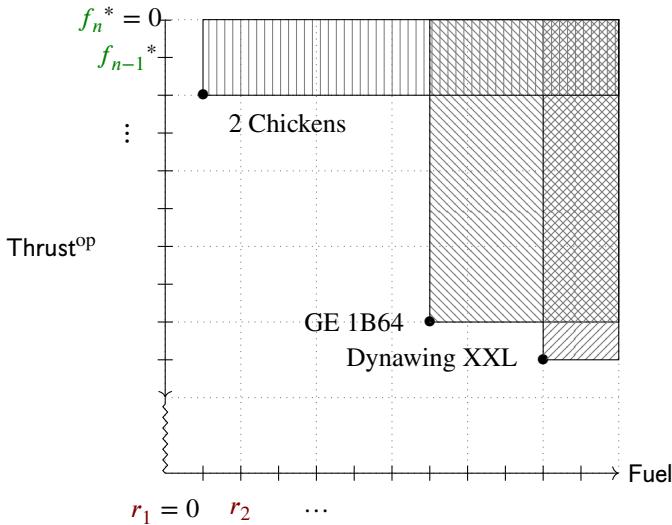


Figure 23.3.: Graphical representation of the possible solutions of the engine design problem.

23.2. Series composition of design problems

We will define several ways to connect and compose design problems. The first and most basic way is series composition, or just ‘composition’.

Definition 23.6 (Series composition). Let $f : \mathcal{A} \rightarrow \mathcal{B}$ and $g : \mathcal{B} \rightarrow \mathcal{C}$ be design problems. We define their *series composition* $(f ; g) : \mathcal{A} \rightarrow \mathcal{C}$ as:

$$(f ; g) : \mathcal{A}^{\text{op}} \times \mathcal{C} \rightarrow_{\text{Pos}} \text{Bool}, \\ \langle \mathbf{a}^*, \mathbf{c} \rangle \mapsto \bigvee_{b \in \mathcal{B}} f(\mathbf{a}^*, b) \wedge g(b^*, \mathbf{c}). \quad (23.3)$$

Alternatively:

$$(f ; g) : \mathcal{A}^{\text{op}} \times \mathcal{C} \rightarrow_{\text{Pos}} \text{Bool}, \\ \langle \mathbf{a}^*, \mathbf{c} \rangle \mapsto \bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in \mathcal{B}}} f(\mathbf{a}^*, b_1) \wedge g(b_2^*, \mathbf{c}). \quad (23.4)$$

We represent series in the diagrammatic notation reported in Fig. 23.4.

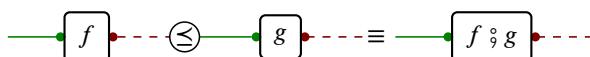


Figure 23.4.: Diagrammatic representation of the series composition of design problems.

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.

Remark 23.7. The series composition operations given in Eqs. (23.3) and (23.4) are equivalent.

First consider the direction Eq. (23.4) \implies Eq. (23.3). In order for

$$\bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in B}} f(\textcolor{violet}{a}^*, \textcolor{red}{b}_1) \wedge g(\textcolor{violet}{b}_2^*, \textcolor{red}{c})$$

to be true, there should exist some $b_1 \leq b_2$ for which $f(\textcolor{violet}{a}^*, \textcolor{red}{b}_1) \wedge g(\textcolor{violet}{b}_2^*, \textcolor{red}{c})$ is true. However, due to the monotonicity of f , $f(\textcolor{violet}{a}^*, b_2) \wedge g(\textcolor{violet}{b}_2^*, \textcolor{red}{c})$ and Eq. (23.3) must be true as well. On the other hand, if

$$\bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in B}} f(\textcolor{violet}{a}^*, \textcolor{red}{b}_1) \wedge g(\textcolor{violet}{b}_2^*, \textcolor{red}{c})$$

is false, then due to the equality, it is false for any $b_1 = b_2$, and therefore all inner terms of Eq. (23.3) must be false as well.

The other direction, Eq. (23.3) \implies Eq. (23.4), can be shown in a similar way. If Eq. (23.3) is true, then there must exist a b' such that $f(\textcolor{violet}{a}^*, \textcolor{red}{b}') = \top$ and $g(\textcolor{violet}{b}'^*, \textcolor{red}{c}) = \top$. Then, the inner term in Eq. (23.4) is true for $b_1 = b_2 = b'$. If Eq. (23.3) is false, then there is no such b' for which both $f(\textcolor{violet}{a}^*, \textcolor{red}{b}')$ and $g(\textcolor{violet}{b}'^*, \textcolor{red}{c})$ are true, but then due to the monotonicity of f and g they also cannot be true for any $b_1 \leq b_2 = b'$ or $b' = b_1 \leq b_2$. Hence, Eq. (23.4) must also be false.

Remark 23.8. At first sight, Eq. (23.4) might seem like a more verbose version of Eq. (23.3). However, assume that we have the means to obtain the minimal antichain of the feasible set of resources that provide $\textcolor{violet}{a}$ for the first term:

$$B_f = \text{Min}\{b_1 \in B \mid f(\textcolor{violet}{a}^*, \textcolor{red}{b}_1) = \top\} \in AB.$$

This represents the minimal resources with which f can provide $\textcolor{violet}{a}$. Assume further that we similarly have the means to obtain the maximal antichain of the feasible set of functionalities that $\textcolor{red}{c}$ provides for the second term

$$B_g = \text{Max}\{b_2 \in B \mid g(\textcolor{violet}{b}_2^*, \textcolor{red}{c}) = \top\} \in AB,$$

which represents the maximal functionality that g can provide given $\textcolor{red}{c}$. Then, Eq. (23.4) implies that it suffices to only evaluate

$$\bigvee_{\substack{b_1 \leq b_2 \\ b_1 \in B_f, b_2 \in B_g}} f(\textcolor{violet}{a}^*, \textcolor{red}{b}_1) \wedge g(\textcolor{violet}{b}_2^*, \textcolor{red}{c}),$$

which can be much more efficient than iterating over all $b \in B$.

Intended semantics The series composition $f ; g$ judges a pair $\langle \textcolor{green}{a}^*, \textcolor{red}{c} \rangle$ as feasible if and only if there exists a $b \in B$ such that $f(\textcolor{green}{a}^*, b)$ and $g(b^*, \textcolor{red}{c})$ are feasible.¹

Example 23.9. After the Bucket of Boom X100 blew upon re-entry, Jeb's Spaceship Parts is building the X101. This time, they are making sure to take into account other aspects of the rocket design, such as the choice of propellant and nozzle (Fig. 23.5).

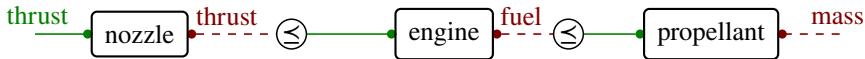


Figure 23.5.: Example of composition.

Remark 23.10. 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.

Let us check that, given design problems f and g , their series composition $f ; g$ is in fact a design problem.

Lemma 23.11. Series composition as in Eq. (23.3) is monotone in a and c .

Proof. We need to show that $[f ; g](\textcolor{green}{a}^*, \textcolor{red}{c})$ is monotone in $\textcolor{green}{a}^*$ and $\textcolor{red}{c}$. Because f represents a design problem, $f(\textcolor{green}{a}^*, b)$ is monotone in $\textcolor{green}{a}^*$, and similarly $g(\textcolor{blue}{b}^*, \textcolor{red}{c})$ is monotone in $\textcolor{red}{c}$. The conjunction “ \wedge ” is monotone in both variables, and likewise the “ \vee ” operation. \square

We can show two important properties for the “ \circledast ” operation: associativity and unitality.

Lemma 23.12. The series composition operation as in Eq. (23.3) is associative, i.e.

$$(f ; g) ; h = f ; (g ; h). \quad (23.5)$$

¹In Eq. (23.3) we could have written “ $\exists_{b \in B}$ ” instead of “ $\bigvee_{b \in B}$ ”; the latter form highlights the connection with operations on matrices. Given a set I and a map $s : I \rightarrow \mathbf{Bool}$, we can define the boolean $\bigvee_{i \in I} s(i)$ by

$$\bigvee_{i \in I} s(i) := \begin{cases} \top & \text{if there exists } i \in I \text{ for which } s(i) = \top, \\ \perp & \text{if there exists no } i \in I \text{ for which } s(i) = \top. \end{cases}$$

For any I , if we have $i_0 \in I$ then $s(i) \leq \bigvee_{i \in I} s(i)$. One can also check that for any $b \in \mathbf{Bool}$ or, more generally, any set of booleans $t : J \rightarrow \mathbf{Bool}$, we have

$$\bigvee_{i \in I} (b \wedge s(i)) = b \wedge \left(\bigvee_{i \in I} s(i) \right) \quad \text{and} \quad \bigvee_{(i,j) \in I \times J} (s(i) \wedge t(j)) = \left(\bigvee_{i \in I} s(i) \right) \wedge \left(\bigvee_{j \in J} t(j) \right).$$

Proof. To show that the operation is associative, we can use distributivity and commutativity in **Bool**:

$$\begin{aligned}
 ((f ; g) ; h)(\textcolor{blue}{a}^*, \textcolor{red}{d}) &= \bigvee_{c \in C} \left(\bigvee_{b \in B} f(\textcolor{blue}{a}^*, \textcolor{red}{b}) \wedge g(\textcolor{blue}{b}^*, \textcolor{blue}{c}) \right) \wedge h(\textcolor{blue}{c}^*, \textcolor{red}{d}) \\
 &= \bigvee_{c \in C} \left(\bigvee_{b \in B} f(\textcolor{blue}{a}^*, \textcolor{red}{b}) \wedge g(\textcolor{blue}{b}^*, \textcolor{blue}{c}) \wedge h(\textcolor{blue}{c}^*, \textcolor{red}{d}) \right) \\
 &= \bigvee_{b \in B} f(\textcolor{blue}{a}^*, \textcolor{red}{b}) \wedge \left(\bigvee_{c \in C} g(\textcolor{blue}{b}^*, \textcolor{blue}{c}) \wedge h(\textcolor{blue}{c}^*, \textcolor{red}{d}) \right) \\
 &= (f ; (g ; h))(\textcolor{blue}{a}^*, \textcolor{red}{d}).
 \end{aligned} \tag{23.6}$$

□

Because of associativity, we can write $f ; g ; h$ without ambiguity. Associativity of composition is represented as in Fig. 23.6.

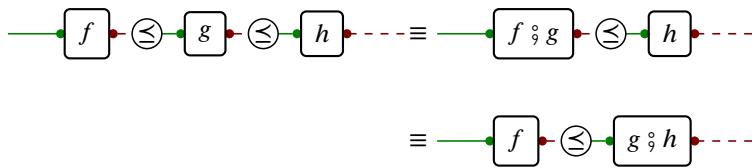


Figure 23.6.: Series composition is associative.

There exists an identity for the “ \circledast ” operation. We define the identity $\text{id}_A : \textcolor{blue}{A} \leftrightarrow \textcolor{red}{A}$ as follows.

Definition 23.13 (Identity design problem). For any poset A , the *identity design problem* $\text{id}_A : \textcolor{blue}{A} \leftrightarrow \textcolor{red}{A}$ is a monotone map

$$\begin{aligned}
 \text{id}_A : \textcolor{blue}{A}^{\text{op}} \times \textcolor{red}{A} &\rightarrow_{\text{Pos}} \textbf{Bool}, \\
 \langle \textcolor{blue}{a}_1^*, \textcolor{red}{a}_2 \rangle &\mapsto \textcolor{blue}{a}_1 \leq_A \textcolor{red}{a}_2.
 \end{aligned} \tag{23.7}$$

In the diagrammatic notation, we represent id_A as in Fig. 23.7.

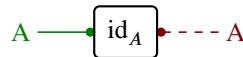


Figure 23.7.: Diagrammatic representation of the identity design problem.

Lemma 23.14. The series composition operation as in Eq. (23.3) satisfies the left and right unit laws (Fig. 23.8).

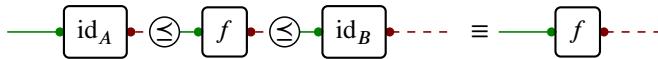


Figure 23.8.: Composition satisfies left and right unit laws.

Proof. Given $f : \mathbf{A} \leftrightarrow \mathbf{B}$, we need to show:

$$\text{id}_A ; f = f = f ; \text{id}_B .$$

In the following, we prove $\text{id}_A ; f = f$. Proving $f ; \text{id}_B = f$ is similar. Consider the poset **Bool**. Since for $x, y \in \mathbf{Bool}$, $x \cong y \Rightarrow x = y$ (also referred to as skeletal [18]), we just need to show that $f \leq \text{id}_A ; f$ and $\text{id}_A ; f \leq f$. We have

$$\begin{aligned}
f(\mathbf{a}^*, \mathbf{b}) &= \top \wedge f(\mathbf{a}^*, \mathbf{b}) \\
&\leq \text{id}_A(\mathbf{a}^*, \mathbf{a}) \wedge f(\mathbf{a}^*, \mathbf{b}) \\
&\leq \bigvee_{a' \in A} \text{id}_A(\mathbf{a}^*, \mathbf{a}') \wedge f(\mathbf{a}'^*, \mathbf{b}) \\
&= (\text{id}_A ; f)(\mathbf{a}^*, \mathbf{b}).
\end{aligned}$$

For the other direction, we need to show that $\text{id}_A ; f \leq f$, i.e.

$$\bigvee_{a' \in A} \text{id}_A(\mathbf{a}^*, \mathbf{a}') \wedge f(\mathbf{a}'^*, \mathbf{b}) \leq f(\mathbf{a}^*, \mathbf{b}).$$

This holds if and only if $\text{id}_A(\mathbf{a}^*, \mathbf{a}') \wedge f(\mathbf{a}'^*, \mathbf{b}) \leq f(\mathbf{a}^*, \mathbf{b})$ for some $a' \in A$. If there is no such a' , then the inequality holds ($\perp \leq \perp$ and $\perp \leq \top$). If there is such an a' , it means that $\text{id}_A(\mathbf{a}^*, \mathbf{a}') = \top$ and $f(\mathbf{a}'^*, \mathbf{b}) = \top$. We know that $\text{id}_A(\mathbf{a}^*, \mathbf{a}') = \top \Leftrightarrow \mathbf{a} \leq \mathbf{a}'$, and hence $f(\mathbf{a}^*, \mathbf{b}) = \top$. \square

23.3. The category of design problems

We will show that the class of all design problems forms a category, which we call **DP**.

Definition 23.15 (Category of design problems). 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 (Definition 23.1).
3. *Identity morphism*: The identity morphism $\text{id}_A : \mathbf{A} \leftrightarrow \mathbf{A}$ is given by Definition 23.13.
4. *Composition operation*: Given two morphisms $f : \mathbf{A} \leftrightarrow \mathbf{B}$ and $g : \mathbf{B} \leftrightarrow \mathbf{C}$, their composition $f ; g : \mathbf{A} \leftrightarrow \mathbf{C}$ is given by Definition 23.6.

We have already shown that the composition operator “ \circ ” is associative and unital, and that the composition of two design problems is a design problem (closure). Therefore, **DP** is a category.

Remark 23.16. **DP** is called **Feas** or **Prof_{Bool}** in [18].

23.4. DP Isomorphisms

Show here the various isomorphisms, also graphically.

24. Profunctors

to write

Contents

24.1. Profunctors	176
24.2. Hom Profunctor	176
24.3. Other examples of profunctors	176
24.4. The bicategory of profunctors	176
24.5. DPI as profunctors	176



24.1. Profunctors

Definition 24.1 (Profunctor). Given two categories **C** and **D**, a *profunctor* from **C** to **D** is a functor of the form

$$H : \mathbf{C}^{\text{op}} \times \mathbf{D} \rightarrow \mathbf{Set}. \quad (24.1)$$

Work out section about profunctors, maybe taking inspiration from spivak

24.2. Hom Profunctor

Discussion as in the lecture

24.3. Other examples of profunctors

To write

24.4. The bicategory of profunctors

To write

24.5. DPI as profunctors

To write

25. Parallelism

In this chapter we add some structure to the definition of a poset, by introducing *monoidal posets* and *monoidal categories*.

Contents

25.1. Monoids	178
25.2. Monoids homomorphism	179
25.3. Dynamical systems and monoids	179
25.4. Monoidal posets	180
25.5. Monoidal categories	181
25.6. DP is a monoidal category	185



25.1. Monoids

make a command for the identity. Also note later we use I

Definition 25.1 (Monoid). A is a set M with a binary operation $\otimes : M \times M \rightarrow M$, and a *neutral element* $1 \in M$, which satisfy:

1. Associative law: $(x \otimes y) \otimes z = x \otimes (y \otimes z)$;
2. Unit Laws: $1 \otimes x = x = x \otimes 1$.

Example 25.2. Consider $\langle \mathbb{R}, +, 0 \rangle$. This is a monoid, since, for all $x, y \in \mathbb{R}$, we have:

$$(x + y) + z = x + (y + z),$$

and

$$x + 0 = 0 + x.$$

Example 25.3. 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:

$$\max(\max(x, y), z) = \max(x, \max(y, z)),$$

and

$$\max(x, 0) = x = \max(0, x).$$

Example 25.4. In this example we look at sequences. A sequence is a function whose domain is a subset of \mathbb{N} , and are called *finite* if the domain of the function is finite. Often finite sequences are referred to as *lists*. Given a set S , we denote the set of all lists on S by S^* . This can be made into a monoid, by considering *concatenation* as the operation, and the empty list as the neutral element. Specifically, a list is an element $s \in S^*$ consists of a $n \in \mathbb{N}$ and a function $f : [n] \rightarrow S$, where $[n] = \{i : \mathbb{N} \mid i < n\} \subseteq \mathbb{N}$. The empty list, denoted $()$, is the unique list of length 0. Given $n > 0$, we can write the list which assigns $0, \dots, n-1$ to s_0, s_1, \dots, s_{n-1} as $(s_0, s_1, \dots, s_{n-1})$. Given a list $x \in S^*$ of length m and a list $y \in S^*$ of length n , we can define their *concatenation* $x * y$ as list of length $m+n$ with:

$$i \mapsto \begin{cases} x_i & \text{if } i < m \\ y_{i-m} & \text{if } i \geq m. \end{cases}$$

Clearly, this definition of concatenation satisfies associativity and unitality, making this construction a monoid. This is often referred to as the *free monoid on S*.

Example 25.5. Given any category \mathbf{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. 25.1 has three objects X, Y, Z and several morphisms. X has four endomorphisms, Y two, and Z three (including identity morphisms). Let's now take the binary operation \otimes 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.

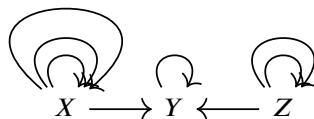


Figure 25.1.

25.2. Monoids homomorphism

Nice example: the map from sequence of characters to sequences of sounds is monoidal in certain languages (Korean, Japanese, almost in Italian.) and also invertible.

25.3. Dynamical systems and monoids

JL: inserting this here as an un-baked idea for a subsection. maybe it could be the first subsection of this chapter; that way identity laws and associative laws can be introduced before talking about categories What are the simplest kinds of mathematical models of a dynamical system that we can think of?

One possible answer is something like this: we can describe a dynamical system as a set S of possible states, together with a description of how states change over time. For the latter, consider time to be labeled by distinct “points in time”. Then, we can just think in terms of time-steps, e.g. seconds, or we can think of points in time where e.g. an action is triggered and the system passes to a new state.

One thing we want to describe is how the state of our system changes over time, and in particular from one moment in time to the next. For any time step, we will not assume that we know what specific state the system is in, but rather we will describe, at once, all possible evolutions during that time step, i.e. we consider all possible initial conditions at once. Given two consecutive moments in time, we might describe the possible changes in the system by a function $T : S \rightarrow S$, which maps each state $s \in S$ to a next state $T(s) \in S$. This is a deterministic change of state: given s , the function T determines the next state $T(s)$. The function T is like a rule. Let's call T an “evolution operator”, because it describes how the system states might evolve over a time step.

We might want to consider various possible evolution operators. We could consider functions T_a , T_b , T_c , etc. We can also compose these functions: given T_a and T_b , we might have, over the course of two time steps, the change described by $T_a \circ T_b$. For simplicity, let's suppose we work with three evolution operations T_a , T_b , and T_c .

-> introduce semigroups (implicitly or explicitly)

-> introduce monoids

AC: For me the basic example of monoid with dynamical systems is taking the transition

function.

Let $E_t^s : X \rightarrow X$ be the evolution function from s to t . Then states evolve like this: $x_t = E_t^s(x_s)$.

If you assume that the system is time invariant, then the evolution only depends on the difference $\delta = s - t$. You have now a commutative monoid of transition functions T_δ where $T_0 = \text{identity}$.

(No need to do semigroups.)

25.4. Monoidal posets

make a command for the identity. Also use 1 instead of I.

Move when we discuss monoidal categories as a basic example.

Definition 25.6 (Monoidal poset). A *monoidal structure* on a poset $\langle P, \leq \rangle$ consists of:

1. An element $I \in P$, called *monoidal unit*, and
2. a function $\otimes : P \times P \rightarrow P$, called the *monoidal product*. Note that we write

$$\otimes(p_1, p_2) = p_1 \otimes p_2, \quad p_1, p_2 \in P.$$

The constituents must satisfy the following properties:

- (a) *Monotonicity*: For all $p_1, p_2, q_1, q_2 \in P$, if $p_1 \leq q_1$ and $p_2 \leq q_2$, then

$$p_1 \otimes p_2 \leq q_1 \otimes q_2.$$

- (b) *Unitality*: For all $p \in P$, $I \otimes p = p$ and $p \otimes I = p$.

- (c) *Associativity*: For all $p, q, r \in P$, $(p \otimes q) \otimes r = p \otimes (q \otimes r)$.

A poset equipped with a monoidal structure $\langle P, \leq, I, \otimes \rangle$ is called a *monoidal poset*.

Example 25.7. Consider the real numbers \mathbb{R} with the poset structure given the usual ordering. Consider 0 as 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 Definition 25.6 are satisfied:

- (a) If $p_1 \leq p_2$ and $q_1 \leq q_2$, it is true that $p_1 + p_2 \leq q_1 + q_2$, $\forall p_1, p_2, q_1, q_2 \in \mathbb{R}$.
- (b) $0 + p = p + 0 = 0$, $\forall p \in \mathbb{R}$.
- (c) $(p + q) + r = p + (q + r)$, $\forall p, q, r \in \mathbb{R}$.

Example 25.8. Someone proposes now to substitute the monoidal unit in Example 25.7 with 1 and the monoidal product with “ $*$ ”. 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, $(-5) \cdot (-4) \not\leq 0 \cdot 3$.

Example 25.9. Consider now $\langle \text{Bool}, \leq_{\text{Bool}}, \top, \wedge \rangle$. The action of the monoidal product “ \wedge ” can be summarized in a table:

\wedge	\perp	\top
\perp	\perp	\perp
\top	\perp	\top

From this table, it is clear that given $x_1 \leq_{\text{Bool}} y_1$ and $x_2 \leq_{\text{Bool}} y_2$, one has $x_1 \wedge x_2 \leq_{\text{Bool}} y_1 \wedge y_2$ (if you do not believe it, try all possible combinations). Furthermore, $x \wedge \top = x = \top \wedge x$.

25.5. Monoidal categories

So far, we have described a single way to compose morphisms of a category: the \circ operation. However, category theory allows to define other ways of composing morphisms, adding structure to the basic category defined in Definition 3.2.

Definition 25.10 (Monoidal category). A *monoidal structure* on a category \mathbf{C} consists of:

1. An object $I \in \text{Ob}_{\mathbf{C}}$ called the *monoidal unit*.
2. A functor $\otimes : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$, called the *monoidal product*.

The two constituents are subject to the natural isomorphisms:

- a) $\lambda_c : I \otimes c \cong c$ for every $c \in \text{Ob}_{\mathbf{C}}$,
- b) $\rho_c : c \otimes I \cong c$ for every $c \in \text{Ob}_{\mathbf{C}}$,
- c) $\alpha_{c,d,e} : (c \otimes d) \otimes e \cong c \otimes (d \otimes e)$ for every $c, d, e \in \text{Ob}_{\mathbf{C}}$.

These isomorphisms are themselves required to satisfy the triangle identity

$$\begin{array}{ccc} (c \otimes I) \otimes d & \xrightarrow{\alpha_{c,I,d}} & c \otimes (I \otimes d) \\ & \searrow \rho_c \otimes I & \swarrow I \otimes \lambda_d \\ & c \otimes d & \end{array} \quad (25.1)$$

and the pentagon identity

$$\begin{array}{ccccc} & & (a \otimes b) \otimes (c \otimes d) & & \\ & \nearrow \alpha_{a \otimes b,c,d} & & \searrow \alpha_{a,b,c \otimes d} & \\ ((a \otimes b) \otimes c) \otimes d & & & & (a \otimes (b \otimes (c \otimes d))) \\ \downarrow \alpha_{a,b,c} \otimes \text{id}_d & & & & \uparrow \text{id}_a \otimes \alpha_{b,c,d} \\ (a \otimes (b \otimes c)) \otimes d & \xrightarrow{\alpha_{a,b \otimes c,d}} & & & a \otimes ((b \otimes c) \otimes d) \end{array} \quad (25.2)$$

for $a, b, c, d \in \text{Ob}_{\mathbf{C}}$. A category equipped with a monoidal structure is called a *monoidal category*. Note that in the case in which the isomorphisms in a), b), and c) are equivalences, one calls the category *strict* monoidal.

Example 25.11. Let's digest the definition of monoidal category with an explanatory example. We consider the structure $\langle \mathbf{Set}, \times, \{1\} \rangle$ and show that it indeed forms a monoidal category. First of all, we specify how the monoidal product (cartesian product here) acts on objects and morphisms in \mathbf{Set} (it is a functor). Given $A, B \in \text{Ob}_{\mathbf{Set}}$, $A \times B$ is the cartesian

product of sets, and given $f : A \rightarrow A'$, $g : B \rightarrow B'$, we have:

$$(f \times g) : A \times B \xrightarrow{\sim} A' \times B'$$

$$\langle a, b \rangle \mapsto \langle f(a), g(b) \rangle.$$

Furthermore, given any $A, B, C \in \text{Ob}_{\text{Set}}$, we specify the unitor $\alpha_{A,B,C}$:

$$\alpha_{A,B,C} : (A \times B) \times C \rightarrow A \times (B \times C)$$

$$\langle \langle a, b \rangle, c \rangle \mapsto \langle a, \langle b, c \rangle \rangle$$

This defines an isomorphism (I can “back and forth”, by switching the tuple separation). We now need to check whether α is natural. We check this graphically, using the commutative diagram in Fig. 25.2.

$$\begin{array}{ccccc} & \langle \langle a, b \rangle, c \rangle & \xrightarrow{\hspace{2cm}} & \langle a, \langle b, c \rangle \rangle & \\ & (A \times B) \times C & \xrightarrow{\alpha_{A,B,C}} & A \times (B \times C) & \\ (f \times g) \times h \downarrow & & & \downarrow f \times (g \times h) & \\ (A' \times B') \times C' & \xrightarrow{\alpha_{A',B',C'}} & A' \times (B' \times C') & & \\ & \langle \langle f(a), g(b) \rangle, h(c) \rangle & \longleftarrow & \langle f(a), \langle g(b), h(c) \rangle \rangle & \end{array}$$

Figure 25.2.

Given an object $A \in \text{Ob}_{\text{Set}}$, the unitor λ_A is given by:

$$\lambda_A : \{1\} \times A \xrightarrow{\sim} A$$

$$\langle *, a \rangle \mapsto a.$$

Again, this defines an isomorphism. Consider a morphism $f : A \rightarrow A'$. We now prove naturality graphically (Fig. 25.3).

$$\begin{array}{ccccc} & \langle *, a \rangle & \xrightarrow{\hspace{2cm}} & a & \\ & \{1\} \times A & \xrightarrow{\lambda_A} & A & \\ \text{id}_{\{1\}} \times f \downarrow & & & \downarrow f & \\ \{1\} \times A' & \xrightarrow{\lambda_{A'}} & A' & & \\ & \langle *, f(a) \rangle & \longleftarrow & f(a) & \end{array}$$

Figure 25.3.

Analogously, given an object $A \in \text{Ob}_{\text{Set}}$, the unitor isomorphism ρ_A is given by:

$$\rho_A : A \times \{1\} \xrightarrow{\sim} A$$

$$\langle a, * \rangle \mapsto a.$$

The proof for naturality is analogous to the one of λ_A . We now need to check whether the triangle and pentagon identites are satisfied. We start by the triangle. Given $A, B \in \text{Ob}_{\text{Set}}$, the proof is displayed in Fig. 25.4.

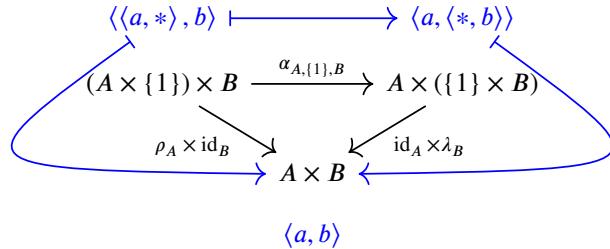


Figure 25.4.

We now prove the pentagon identity. Given $A, B, C, D \in \text{Ob}_{\text{Set}}$, the proof is reported in Fig. 25.5.

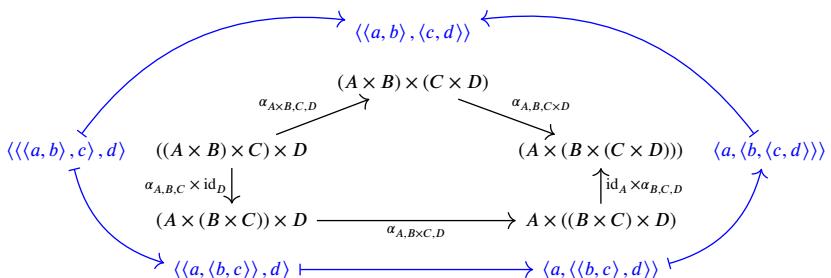


Figure 25.5.

Example 25.12. 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. 25.6). 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 ”. 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.

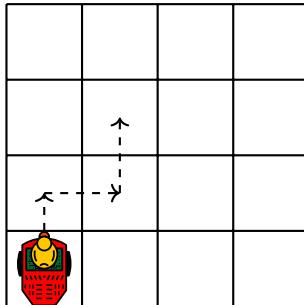


Figure 25.6.: Example of the robot category.

Definition 25.13 (Symmetric monoidal category). Let $\langle \mathbf{C}, \otimes, I \rangle$ be a monoidal category (Definition 25.10). A *symmetric structure* on it consists of one component: For any objects $c, d \in \text{Ob}_{\mathbf{C}}$ an isomorphism $\sigma_{c,d} : (c \otimes d) \xrightarrow{\cong} (d \otimes c)$, called the *braiding*. The braiding must satisfy:

1. *Naturality*: Given any morphisms $f_1 : c_1 \rightarrow d_1$ and $f_2 : c_2 \rightarrow d_2$, the following diagram must commute:

$$\begin{array}{ccc} c_1 \otimes c_2 & \xrightarrow{f_1 \otimes f_2} & d_1 \otimes d_2 \\ \downarrow \sigma & & \downarrow \sigma \\ c_2 \otimes c_1 & \xrightarrow{f_2 \otimes f_1} & d_2 \otimes d_1 \end{array} \quad (25.3)$$

2. *Triangle identities*: Given any objects $c, d \in \text{Ob}_{\mathbf{C}}$, the following diagrams must commute:

$$\begin{array}{ccc} I \otimes c & \xrightarrow{\sigma} & c \otimes I \\ \lambda \searrow & & \downarrow \rho \\ & c & \end{array} \quad \begin{array}{ccc} c \otimes d & \xrightarrow{\sigma} & d \otimes c \\ \swarrow & & \downarrow \sigma \\ c \otimes d & & \end{array} \quad (25.4)$$

3. *Hexagon identity*: Given any objects $c, d, e \in \text{Ob}_{\mathbf{C}}$, the following diagram must commute:

$$\begin{array}{ccccc} (c \otimes d) \otimes e & \xrightarrow{\sigma_{c,d} \otimes \text{id}_e} & (d \otimes c) \otimes e & \xrightarrow{\alpha_{d,c,e}} & d \otimes (c \otimes e) \\ \alpha_{c,d,e} \downarrow & & & & \downarrow \text{id}_d \otimes \sigma_{c,e} \\ c \otimes (d \otimes e) & \xrightarrow{\sigma_{c,d \otimes e}} & (d \otimes e) \otimes c & \xrightarrow{\alpha_{d,e,c}} & d \otimes (e \otimes c) \end{array} \quad (25.5)$$

25.6. DP is a monoidal category

Example 25.14. After the X101 spontaneously combusted in low Earth orbit, the astronauts at Jeb's Spaceship Parts go on strike. They demand that the engineers take into account safety and living conditions on the future X102. As long as the propulsion and life support systems of the X102 do not interact, we can simply tensor the two design problems representing these systems into one, big co-design problem (Fig. 25.7).

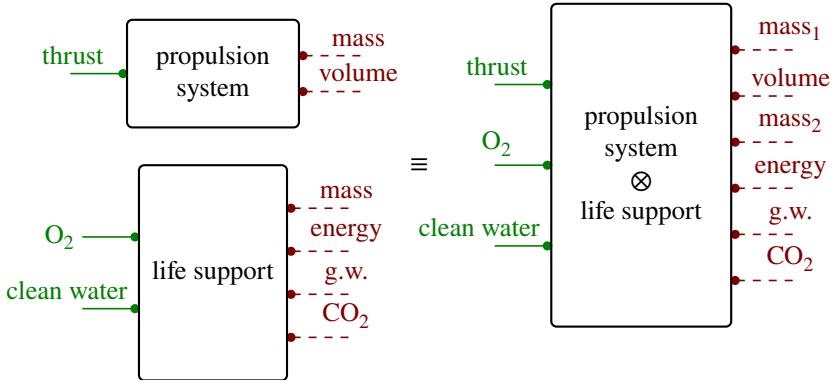


Figure 25.7.: Example of tensor of design problems.

In **DP**, putting two design problems in parallel corresponds to their *monoidal product*.

Definition 25.15 (Monoidal product in **DP**). Given two design problems $f : \textcolor{blue}{A} \leftrightarrow \textcolor{red}{B}$ and $g : \textcolor{blue}{C} \leftrightarrow \textcolor{red}{D}$, their *monoidal product* $f \otimes g : \textcolor{blue}{A} \times \textcolor{blue}{C} \leftrightarrow \textcolor{red}{B} \times \textcolor{red}{D}$ is their conjunction:

$$f \otimes g : (\textcolor{blue}{A} \times \textcolor{blue}{C})^{\text{op}} \times (\textcolor{red}{B} \times \textcolor{red}{D}) \rightarrow_{\text{Pos Bool}} \langle \langle \textcolor{blue}{a}, \textcolor{blue}{c} \rangle^*, \langle \textcolor{red}{b}, \textcolor{red}{d} \rangle \rangle \mapsto f(\textcolor{blue}{a}^*, \textcolor{red}{b}) \wedge g(\textcolor{blue}{c}^*, \textcolor{red}{d}). \quad (25.6)$$

The diagrammatic representation of the monoidal product is reported in Fig. 25.8.

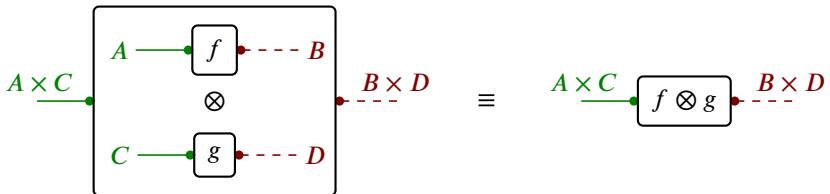


Figure 25.8.: Monoidal product of design problems.

Remark 25.16. For $f : \textcolor{blue}{A} \leftrightarrow \textcolor{red}{B}$ and $g : \textcolor{blue}{C} \leftrightarrow \textcolor{red}{D}$, the monoidal product

$$(f \otimes g)(\langle \textcolor{blue}{a}^*, \textcolor{blue}{c}^* \rangle, \langle \textcolor{red}{b}, \textcolor{red}{d} \rangle) \quad (25.7)$$

is true if *both* $f(\mathbf{a}^*, \mathbf{b})$ and $g(\mathbf{c}^*, \mathbf{d})$ are true, and false otherwise.

Lemma 25.17. The monoidal product \otimes is functorial (Definition 14.1) in **DP**.

Proof. First, consider posets $A, B \in \text{Ob}_{\mathbf{DP}}$. We show that $\text{id}_A \otimes \text{id}_B = \text{id}_{A \times B}$. It holds

$$\begin{aligned} (\text{id}_A \otimes \text{id}_B) ((\mathbf{a}_1, \mathbf{b}_1)^*, (\mathbf{a}_2, \mathbf{b}_2)) &= \text{id}_A(\mathbf{a}_1^*, \mathbf{a}_2) \wedge \text{id}_B(\mathbf{b}_1^*, \mathbf{b}_2) \\ &= (\mathbf{a}_1 \leq_A \mathbf{a}_2) \wedge (\mathbf{b}_1 \leq_B \mathbf{b}_2) \\ &= \langle \mathbf{a}_1, \mathbf{b}_1 \rangle \leq_{A \times B} \langle \mathbf{a}_2, \mathbf{b}_2 \rangle \\ &= \text{id}_{A \times B} ((\mathbf{a}_1, \mathbf{b}_1)^*, (\mathbf{a}_2, \mathbf{b}_2)). \end{aligned} \tag{25.8}$$

Furthermore, consider the design problems

$$f_1 : \mathbf{A}_1 \leftrightarrow \mathbf{B}_1, \quad f_2 : \mathbf{A}_2 \leftrightarrow \mathbf{B}_2, \quad g_1 : \mathbf{B}_1 \leftrightarrow \mathbf{C}_1, \quad g_2 : \mathbf{B}_2 \leftrightarrow \mathbf{C}_2.$$

We need to show that $\underbrace{((f_1 \circ g_1) \otimes (f_2 \circ g_2))}_{\star} = ((f_1 \otimes f_2) \circ (g_1 \otimes g_2))$. It holds

$$\begin{aligned} \star ((\mathbf{a}_1, \mathbf{a}_2)^*, (\mathbf{c}_1, \mathbf{c}_2)) &= (f_1 \circ g_1)(\mathbf{a}_1^*, \mathbf{c}_1) \wedge (f_2 \circ g_2)(\mathbf{a}_2^*, \mathbf{c}_2) \\ &= \left(\bigvee_{b_1 \in B_1} (f_1(\mathbf{a}_1^*, b_1) \wedge g_1(\mathbf{b}_1^*, \mathbf{c}_1)) \right) \wedge \left(\bigvee_{b_2 \in B_2} (f_2(\mathbf{a}_2^*, b_2) \wedge g_2(\mathbf{b}_2^*, \mathbf{c}_2)) \right) \\ &= \bigvee_{b_1 \in B_1} \bigvee_{b_2 \in B_2} (f_1(\mathbf{a}_1^*, b_1) \wedge g_1(\mathbf{b}_1^*, \mathbf{c}_1) \wedge f_1(\mathbf{a}_2^*, b_2) \wedge f_2(\mathbf{b}_2^*, \mathbf{c}_2)) \\ &= \bigvee_{(b_1, b_2) \in B_1 \times B_2} (f_1(\mathbf{a}_1^*, b_1) \wedge f_2(\mathbf{a}_2^*, b_2) \wedge g_1(\mathbf{b}_1^*, \mathbf{c}_1) \wedge g_2(\mathbf{b}_2^*, \mathbf{c}_2)) \\ &= ((f_1 \otimes f_2) \circ (g_1 \otimes g_2)) ((\mathbf{a}_1, \mathbf{a}_2)^*, (\mathbf{c}_1, \mathbf{c}_2)). \end{aligned} \tag{25.9}$$

Therefore, \otimes is functorial in **DP**. \square

Lemma 25.18. $\langle \mathbf{DP}, \otimes, \{1\} \rangle$ is a monoidal category.

Proof. To show that **DP** is a monoidal category, we have to first identify the constituents presented in Definition 25.10. First, recall $\{1\}$ to be singleton: this is the monoidal unit. In Lemma 25.17 we have shown that \otimes is a functor. Furthermore, we identify

- $\lambda_A : \{1\} \times \mathbf{A} \leftrightarrow \mathbf{A}$, for all $A \in \text{Ob}_{\mathbf{DP}}$, is the left unit. This is given by

$$\lambda_A ((1, \mathbf{a}_1)^*, \mathbf{a}_2) := \mathbf{a}_1 \leq_A \mathbf{a}_2. \tag{25.10}$$

To prove that this is an isomorphism, we define its inverse $\lambda_A^{-1} : \mathbf{A} \leftrightarrow \{1\} \times \mathbf{A}$

and show that $\lambda_A \circ \lambda_A^{-1} = \text{id}_{\{1\} \times A}$ and $\lambda_A^{-1} \circ \lambda_A = \text{id}_A$. One has

$$\begin{aligned} (\lambda_A^{-1} \circ \lambda_A) ((\mathbf{a}_1^*, \mathbf{a}_2)) &= \bigvee_{\langle 1, a \rangle \in \{1\} \times A} \lambda_A^{-1}(\mathbf{a}_1^*, \langle 1, \mathbf{a} \rangle) \wedge \lambda_A(\langle 1, \mathbf{a} \rangle^*, \mathbf{a}_2) \\ &= \bigvee_{\langle 1, a \rangle \in \{1\} \times A} (\mathbf{a}_1 \leq \mathbf{a}) \wedge \mathbf{a} \leq \mathbf{a}_2 \\ &= \mathbf{a}_1 \leq \mathbf{a}_2 \\ &= \text{id}_A((\mathbf{a}_1^*, \mathbf{a}_2)). \end{aligned} \tag{25.11}$$

Similarly, one can show that $\lambda_A \circ \lambda_A^{-1} = \text{id}_{\{1\} \times A}$.

- $\rho_A : \mathbf{A} \times \{1\} \leftrightarrow \mathbf{A}$, for all $A \in \text{Ob}_{\text{DP}}$, is the right unitor. This is given by

$$\rho((\mathbf{a}_1, 1)^*, \mathbf{a}_2) := \mathbf{a}_1 \leq_A \mathbf{a}_2. \tag{25.12}$$

The proof that ρ_A is an isomorphism is analogous to the one for λ_A .

- $\alpha_{A,B,C} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \leftrightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C})$ for all $A, B, C \in \text{Ob}_{\text{DP}}$, is the associator. It is given by

$$\alpha_{A,B,C}((\langle \mathbf{a}_1, \mathbf{b}_1 \rangle, \mathbf{c}_1)^*, \langle \mathbf{a}_2, \langle \mathbf{b}_2, \mathbf{c}_2 \rangle \rangle) := (\mathbf{a}_1 \leq_A \mathbf{a}_2) \wedge (\mathbf{b}_1 \leq_B \mathbf{b}_2) \wedge (\mathbf{c}_1 \leq_C \mathbf{c}_2). \tag{25.13}$$

To prove that $\alpha_{A,B,C}$ is an isomorphism, we define its inverse

$$\alpha_{A,B,C}^{-1} : \mathbf{A} \times (\mathbf{B} \times \mathbf{C}) \leftrightarrow (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \tag{25.14}$$

and show $\alpha_{A,B,C}^{-1} \circ \alpha_{A,B,C} = \alpha_{A,B,C} \circ \alpha_{A,B,C}^{-1} = \text{id}_{A \times B \times C}$. One has

$$\begin{aligned} &(\alpha_{A,B,C}^{-1} \circ \alpha_{A,B,C})(\langle \mathbf{a}_1, \langle \mathbf{b}_1, \mathbf{c}_1 \rangle \rangle^*, \langle \mathbf{a}_2, \langle \mathbf{b}_2, \mathbf{c}_2 \rangle \rangle) \\ &= \bigvee_{\langle \langle \mathbf{a}, \mathbf{b} \rangle, \mathbf{c} \rangle \in (A \times B) \times C} \alpha_{A,B,C}^{-1}((\mathbf{a}_1, \langle \mathbf{b}_1, \mathbf{c}_1 \rangle)^*, \langle \langle \mathbf{a}, \mathbf{b} \rangle, \mathbf{c} \rangle) \wedge \\ &\quad \alpha_{A,B,C}((\langle \mathbf{a}, \mathbf{b} \rangle, \mathbf{c})^*, \langle \mathbf{a}_2, \langle \mathbf{b}_2, \mathbf{c}_2 \rangle \rangle) \\ &= \bigvee_{\langle \langle \mathbf{a}, \mathbf{b} \rangle, \mathbf{c} \rangle \in (A \times B) \times C} ((\mathbf{a}_1 \leq \mathbf{a}) \wedge (\mathbf{b}_1 \leq \mathbf{b}) \wedge (\mathbf{c}_1 \leq \mathbf{c})) \wedge \\ &\quad ((\mathbf{a} \leq \mathbf{a}_2) \wedge (\mathbf{b} \leq \mathbf{b}_2) \wedge (\mathbf{c} \leq \mathbf{c}_2)) \\ &= (\mathbf{a}_1 \leq \mathbf{a}_2) \wedge (\mathbf{b}_1 \leq \mathbf{b}_2) \wedge (\mathbf{c}_1 \leq \mathbf{c}_2) \\ &= \text{id}_{A \times B \times C}((\mathbf{a}_1, \mathbf{b}_1, \mathbf{c}_1)^*, \langle \mathbf{a}_2, \mathbf{b}_2, \mathbf{c}_2 \rangle). \end{aligned} \tag{25.15}$$

The proof for $\alpha_{A,B,C} \circ \alpha_{A,B,C}^{-1}$ is analogous.

Therefore, $\langle \text{DP}, \otimes, \{1\} \rangle$ is a monoidal category. \square

25.6.1. DP is a symmetric monoidal category

Lemma 25.19. For any $A, B \in \text{Ob}_{\text{DP}}$, the design problem $\sigma_{A,B} : \mathbf{A} \times \mathbf{B} \leftrightarrow \mathbf{B} \times \mathbf{A}$ given by

$$\sigma_{A,B}((\mathbf{a}_1, \mathbf{b}_1)^*, \langle \mathbf{b}_2, \mathbf{a}_2 \rangle) := (\mathbf{a}_1 \leq_A \mathbf{a}_2) \wedge (\mathbf{b}_1 \leq_B \mathbf{b}_2) \tag{25.16}$$

constitutes the braiding operation for a symmetric monoidal structure on $\langle \mathbf{DP}, \otimes, \{1\} \rangle$. In other words, $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is a symmetric monoidal category.

Proof. In this proof, given two elements a_1, a_2 of a poset A , we denote for brevity $a_1 \leq_A a_2$ by $a_1 \leq a_2$. To prove that $\sigma_{A,B}$ is an isomorphism, we use Definition 25.10 and show $\sigma_{A,B} \circ \sigma_{B,A} = \text{id}_{A \times B}$. One has

$$\begin{aligned}
 & (\sigma_{A,B} \circ \sigma_{B,A}) (\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle) \\
 &= \bigvee_{(b,a) \in B \times A} \sigma_{A,B} (\langle a_1, b_1 \rangle^*, \langle b, a \rangle) \wedge \sigma_{B,A} (\langle b, a \rangle^*, \langle a_2, b_2 \rangle) \\
 &= ((a_1 \leq a) \wedge (b_1 \leq b)) \wedge ((a \leq a_2) \wedge (b \leq b_2)) \\
 &= (a_1 \leq a_2) \wedge (b_1 \leq b_2) \\
 &= \text{id}_{A \times B} (\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle).
 \end{aligned} \tag{25.17}$$

This also shows the second triangle identity, i.e. that $\sigma_{A,B}$ is its own identity. For naturality, let's consider two morphisms (design problems) $f_1 : \mathbf{A}_1 \rightarrow \mathbf{B}_1$, $f_2 : \mathbf{A}_2 \rightarrow \mathbf{B}_2$. For brevity, denote $\sigma_{B_1 \times B_2, B_2 \times B_1}$ by σ_B and $\sigma_{A_1 \times A_2, A_2 \times A_1}$ by σ_A . One has

$$\begin{aligned}
 & ((f_1 \otimes f_2) \circ \sigma_B) (\langle a_1, a_2 \rangle^*, \langle b_2, b_1 \rangle) \\
 &= \bigvee_{(b', b'') \in B_1 \times B_2} (f_1 \otimes f_2) (\langle a_1, a_2 \rangle^*, \langle b', b'' \rangle) \wedge \sigma_B (\langle b', b'' \rangle, \langle b_2, b_1 \rangle) \\
 &= \bigvee_{(b', b'') \in B_1 \times B_2} (f_1(a_1^*, b') \wedge f_2(a_2^*, b'')) \wedge ((b' \leq b_1) \wedge (b'' \leq b_2)) \\
 &= f_1(a_1^*, b_1) \wedge f_2(a_2^*, b_2),
 \end{aligned} \tag{25.18}$$

where the last step comes from the monotonicity of f_1 and f_2 . Similarly,

$$\begin{aligned}
 & (\sigma_A \circ (f_2 \otimes f_1)) (\langle a_1, a_2 \rangle^*, \langle b_2, b_1 \rangle) \\
 &= \bigvee_{(a'', a') \in A_2 \times A_1} \sigma_A (\langle a_1, a_2 \rangle^*, \langle a'', a' \rangle) \wedge (f_2 \otimes f_1) (\langle a'', a' \rangle^*, \langle b_2, b_1 \rangle) \\
 &= \bigvee_{(a'', a') \in A_2 \times A_1} ((a_1 \leq a') \wedge (a_2 \leq a'')) \wedge (f_2(a''^*, b_2) \wedge f_1(a'^*, b_1)) \\
 &= f_2(a_1^*, b_1) \wedge f_1(a_2^*, b_2).
 \end{aligned} \tag{25.19}$$

To show the first triangle identity, we write

$$\begin{aligned}
 & (\sigma_{\{1\} \times A} \circ \rho_A) (\langle 1, a_1 \rangle^*, a_2) \\
 &= \bigvee_{\langle a', 1 \rangle \in A \times \{1\}} \sigma_{\{1\} \times A} (\langle 1, a_1 \rangle^*, \langle a', 1 \rangle) \wedge \rho_A (\langle a', 1 \rangle^*, a_2) \\
 &= \bigvee_{\langle a', 1 \rangle \in A \times \{1\}} (1 \leq 1) \wedge (a_1 \leq a') \wedge (a' \leq a_2) \\
 &= a_1 \leq a_2 \\
 &= \lambda_A (\langle 1, a_1 \rangle^*, a_2).
 \end{aligned} \tag{25.20}$$

The hexagon identities are more verbose. Consider $A, B, C \in \text{Ob}_{\mathbf{DP}}$. For brevity, we denote $\alpha_{A, B, C}$ by α , $\sigma_{A, B} \otimes \text{id}_C$ by σ' , $\text{id}_B \otimes \sigma_{A, C}$ by σ'' , $(B \times A) \times C$ as \Diamond , and $B \times (A \times C)$ as Δ . Recall that

$$\begin{aligned}
 \sigma' (\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) &= ((a_1 \leq a_2) \wedge (b_1 \leq b_2)) \wedge (c_1 \leq c_2) \\
 &= (a_1 \leq a_2) \wedge (b_1 \leq b_2) \wedge (c_1 \leq c_2)
 \end{aligned} \tag{25.21}$$

One has

$$\begin{aligned}
 & (\sigma' \circ \alpha) (\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) \\
 &= \bigvee_{\langle \langle b, a \rangle, c \rangle \in \Diamond} \sigma' (\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle \langle b, a \rangle, c \rangle) \wedge \alpha (\langle \langle b, a \rangle, c \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) \\
 &= \bigvee_{\langle \langle b, a \rangle, c \rangle \in \Diamond} ((a_1 \leq a) \wedge (b_1 \leq b) \wedge (c_1 \leq c)) \wedge \\
 &\quad ((b \leq b_2) \wedge (a \leq a_2) \wedge (c \leq c_2)) \\
 &= (b_1 \leq b_2) \wedge (a_1 \leq a_2) \wedge (c_1 \leq c_2) \\
 &= \underbrace{\alpha (\langle \langle b_1, a_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle)}_{\star}.
 \end{aligned} \tag{25.22}$$

Furthermore, consider

$$\begin{aligned}
 & (\star \circ \sigma'') (\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_3, \langle c_3, a_3 \rangle \rangle) \\
 &= \bigvee_{\langle b_2, \langle a_2, c_2 \rangle \rangle \in \Delta} \star (\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) \wedge \\
 &\quad \sigma'' (\langle b_2, \langle a_2, c_2 \rangle \rangle^*, \langle b_3, \langle c_3, a_3 \rangle \rangle) \\
 &= \bigvee_{\langle b_2, \langle a_2, c_2 \rangle \rangle \in \Delta} ((b_1 \leq b_2) \wedge (a_1 \leq a_2) \wedge (c_1 \leq c_2)) \wedge \\
 &\quad ((b_2 \leq b_3) \wedge (a_2 \leq a_3) \wedge (c_2 \leq c_3)) \\
 &= (a_1 \leq a_3) \wedge (b_1 \leq b_3) \wedge (c_1 \leq c_3).
 \end{aligned} \tag{25.23}$$

It is easy to see that the other direction in the hexagon commutative diagram commutes. With this we have proved that σ is a valid braiding operation and hence that $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is a symmetric monoidal category. \square

26. Feedback

to write

Contents

26.1. Trace of a linear transformation	192
26.2. Continuous LTI	192
26.3. Feedback in category theory	193
26.4. Feedback in design problems	193



26.1. Trace of a linear transformation

Consider the category **FinVect** of finite dimensional vector spaces, which has as objects finite dimensional vector spaces and as morphisms linear maps between them. Using the tensor product \otimes of vector spaces as monoidal product, one can show **FinVect** 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 (also called “partial trace”) is a linear function $\text{Tr}_{B,C}^D(f) : B \rightarrow C$, given by

$$\left(\text{Tr}_{B,C}^D(f) \right)_{i,j} = \sum_k f_{i \otimes k, j \otimes k} \quad (26.1)$$

Explain notation and show the simple case of sum of diagonals.

Add figure using the graphical notations for tensor operations

26.2. Continuous LTI

rough here, to be polished and add trace thing with Gr

In the following, we present the category of continuous linear time-invariant dynamical systems. Let $T = \mathbb{R}_{\geq 0}$ represent time.

Objects The objects of the category are natural numbers $n \in \mathbb{N}_0$. These represent sequences $s : T \rightarrow \mathbb{R}^n$, defining tuples in $\mathbb{R}^n \times T$.

Morphisms A morphism in **LTI** is an arrow $l \rightarrow m$, for $l, m \in \mathbb{N}_0$. The arrow describes the transformation of an input sequence $u : T \rightarrow \mathbb{R}^l$ into an output sequence $y : T \rightarrow \mathbb{R}^m$. Such an arrow is given by a continuous time LTI system of the form

$$\begin{aligned} \dot{x}_t &= Ax_t + Bu_t \\ y_t &= Cx_t + Du_t, \end{aligned} \quad (26.2)$$

where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times l}$, $C \in \mathbb{R}^{m \times n}$, $D \in \mathbb{R}^{m \times l}$, $x_t \in \mathbb{R}^n$, $u_t \in \mathbb{R}^l$, and $y_t \in \mathbb{R}^m$. For brevity, we refer to the LTI as the tuple $\langle A, B, C, D \rangle$, leaving the dimensions of the input/output implicit.

Identity morphism The identity morphism for $l \in \text{Ob}_{\text{LTI}}$ is an arrow $l \rightarrow l$, parametrized by the system $\langle 0, 0^{1 \times l}, 0^{l \times 1}, \mathbb{I}^{l \times l} \rangle$.

Composition of morphisms Given two arrows $a \rightarrow b$ and $b \rightarrow c$, parametrized by the two LTI systems $\langle A_1, B_1, C_1, D_1 \rangle$ and $\langle A_2, B_2, C_2, D_2 \rangle$, their composition is an arrow $a \rightarrow c$, parametrized by the LTI system $\langle A, B, C, D \rangle$, where

$$A = \begin{bmatrix} A_1 & 0 \\ B_2 C_1 & A_2 \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ B_2 D_1 \end{bmatrix}, \quad C = [D_2 C_1 \quad C_2], \quad D = D_2 D_1. \quad (26.3)$$

26.3. Feedback in category theory

Definition 26.1 (Traced monoidal category). A symmetric monoidal category $\langle \mathbf{C}, \otimes, \{1\}, \sigma \rangle$ is said to be *traced* if equipped with a family of functions

$$\text{Tr}_{A,B}^X : \mathbf{C}(A \otimes X, B \otimes X) \rightarrow \mathbf{C}(A, B), \quad (26.4)$$

satisfying the following axioms:

1. *Vanishing*: For all morphisms $f : A \rightarrow B$ in \mathbf{C} ,

$$\text{Tr}_{A,B}^1(f) = f. \quad (26.5)$$

Furthermore, for all morphisms $f : A \otimes X \otimes Y \rightarrow B \otimes X \otimes Y$ in \mathbf{C} :

$$\text{Tr}_{A,B}^{X \otimes Y}(f) = \text{Tr}_{A,B}^X \left(\text{Tr}_{A \otimes X, B \otimes X}^Y(f) \right). \quad (26.6)$$

2. *Superposing*: For all morphisms $f : A \otimes X \rightarrow B \otimes X$ in \mathbf{C} :

$$\text{Tr}_{C \otimes A, C \otimes B}^X(\text{id}_C \otimes f) = \text{id}_C \otimes \text{Tr}_{A,B}^X(f). \quad (26.7)$$

3. *Yanking*:

$$\text{Tr}_{X,X}^X(\sigma_{X,X}) = \text{id}_X. \quad (26.8)$$

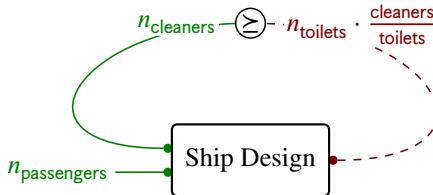
26.4. Feedback in design problems

Motivation

26.4.1. The rocket equation

26.4.2. The scalar case: Toilets in a ship

Consider the case in which you are designing the toilets of a cruise ship. You know that you need a toilet every 10 passengers (i.e., if you have 11 passengers, you need 2 toilets). Furthermore, you know that each toilet needs an employee for its service, i.e. an extra passenger. Now, the problem of maximizing the number of people you can put on the ship, by minimizing the number of toilets you need to install, is a design problem. The resource poset is the one describing the number of toilets needed n_{toilets} , and the functionality poset is the one describing the number of people you can accommodate on the ship $n_{\text{passengers}}$. This can also be written diagrammatically, as



Don't want to introduce sum and multiplication

Example 26.2. Consider a vehicle motor that weighs a certain amount but can also carry some weight (Fig. 26.1). In the diagram, we haven't added anything to the weight of the

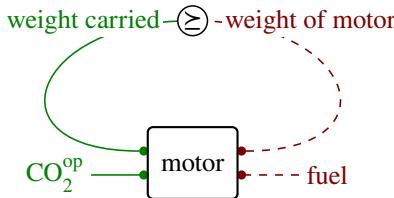


Figure 26.1.: Example of feedback in design problems.

close the loop below

motor, so currently the only thing the motor is carrying is itself. Also, note that we are considering CO_2^{op} since we want to optimize toward less CO_2 . Fix a given amount of CO_2 and fuel. In that case, closing the loop corresponds to drawing a line ($c^* = c$) in the graph of $\text{weight}^{\text{op}} \times \text{weight}$ and taking only solutions under the line in Fig. 26.2.

Note that the shaded area is *not* an upper set. This is admissible, since the actual feasible set of 'motor' is a subset of $\text{CO}_2 \times \text{fuel}$, and there, it is an upper set.

change this example

Suppose that we are given a design problem with a resource and a functionality of the same type C (Fig. 26.3).

Can we "close the loop", as in the diagram reported in Fig. 26.4?

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 26.3 (Trace of a design problem). Given a design problem $f : C \times A \leftrightarrow C \times B$, we can define its trace $\text{Tr}_{A,B}^C(f) : A \leftrightarrow B$ as follows:

$$\begin{aligned} \text{Tr}_{A,B}^C(f) : A^{\text{op}} \times B &\rightarrow_{\text{Pos}} \text{Bool} \\ \langle a^*, b \rangle &\mapsto \bigvee_{c \in C} f(\langle a^*, c \rangle, \langle b^*, c \rangle). \end{aligned} \tag{26.9}$$

Think of drawing a loop as a way of writing down the following requirement: Something that produces C should not use up more of C than it produces.

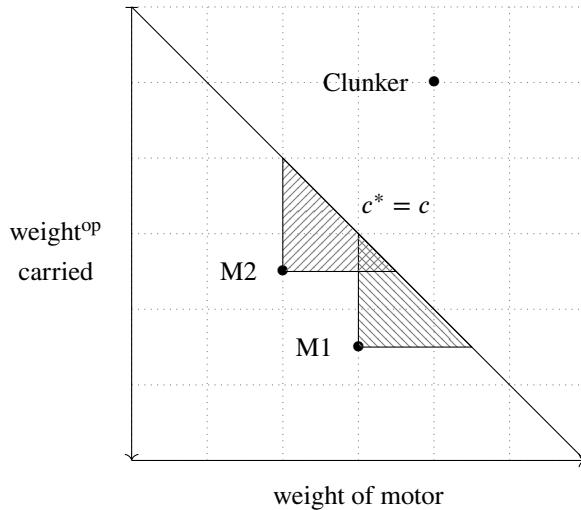


Figure 26.2.: The shaded area marks a portion of the feasibility set of a traced design problem, ‘motor’. Note that it is not an upper set in the subspace “weight \times weight” of ‘motor’.

close the loop below

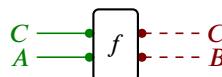


Figure 26.3.: Design problem with a resource and a functionality of the same type.

have C be below so that we can close the loop below

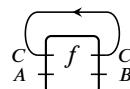


Figure 26.4.: Closing the loop in the design problem.

have C be below so that we can close the loop below

Trace axioms We will show that the loop operation $\text{Tr}_{A,B}^C$ corresponds to the *trace* in **DP**. Intuitively, forming a loop models the idea of feedback in a control-theoretic setting—the output of a process influences the choice of input—while the idea of “trace” of a monoidal category comes from the trace of a square matrix ($\text{Tr } A = \sum_i a_{ii}$), which defines the categorical trace in the (monoidal) category of vector spaces. The connection between the two is more apparent if one decomposes the trace of a square matrix as a set of properties that any linear map from a space to itself should satisfy. One can find the trace axioms in any standard text on category theory, e.g. [macLane]; these are equivalent to certain diagrammatic conditions [24], as in Table 26.1.

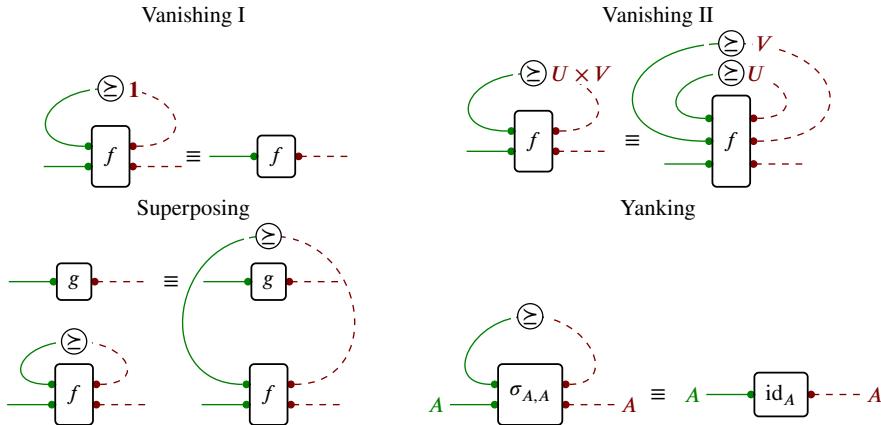


Table 26.1.: The trace axioms in diagrammatic form [24].

Lemma 26.4. Trace as in Definition 26.3 satisfies the trace axioms. In other words, $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is a traced monoidal category, with trace as in Eq. (26.9).

Proof. We have already shown that $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is a symmetric monoidal category (Lemma 25.19). We prove the trace axioms one by one, starting from vanishing (Eq. (26.5), Eq. (26.6)). Given any $A, B \in \text{Ob}_{\mathbf{DP}}$ and $f : A \times \{1\} \rightarrow B \times \{1\}$ in \mathbf{DP} , we have

$$\begin{aligned} \text{Tr}_{A,B}^{\{1\}}(f)(a^*, b) &= \bigvee_{c \in \{1\}} f(\langle a, c \rangle^*, \langle b, c \rangle) \\ &= f(\langle a, 1 \rangle^*, \langle b, 1 \rangle) \\ &= f(a^*, b). \end{aligned} \tag{26.10}$$

Furthermore, for any morphism $f : A \times X \times Y \rightarrow B \times X \times Y$ in \mathbf{DP} , one has

$$\begin{aligned} \text{Tr}_{A,B}^{X \times Y}(f)(a^*, b) &= \bigvee_{\langle x,y \rangle \in X \times Y} f(\langle a, x, y \rangle^*, \langle b, x, y \rangle) \\ &= \bigvee_{x \in X} \left(\bigvee_{y \in Y} f(\langle a, x, y \rangle^*, \langle b, x, y \rangle) \right) \\ &= \text{Tr}_{A,B}^X \left(\text{Tr}_{A \times X, B \times X}^Y(f)(\langle a, x \rangle^*, \langle b, x \rangle) \right). \end{aligned} \tag{26.11}$$

For the superposing axiom (Eq. (26.7)), consider $f : A \times X \rightarrow B \times X$ in \mathbf{DP} . One

has

$$\begin{aligned}
 \text{Tr}_{C \times A, C \times B}^X(\text{id}_C \otimes f)(\langle c_1, a \rangle^*, \langle c_2, b \rangle) &= \bigvee_{x \in X} \text{id}_C(c_1^*, c_2) \wedge f(\langle a, x \rangle^*, \langle b, x \rangle) \\
 &= \text{id}_C(c_1^*, c_2) \wedge \bigvee_{x \in X} f(\langle a, x \rangle^*, \langle b, x \rangle) \\
 &= (\text{id}_C \otimes \text{Tr}_{A, B}^X(f))(\langle c_1, a \rangle^*, \langle c_2, b \rangle).
 \end{aligned} \tag{26.12}$$

Finally, for yanking (Eq. (26.8)) consider $\sigma_{X, X}$. One has

$$\begin{aligned}
 \text{Tr}_{X, X}^X(\sigma_{X, X})(x_1^*, x_2) &= \bigvee_{x \in X} \sigma_{X, X}(\langle x_1, x \rangle^*, \langle x, x_2 \rangle) \\
 &= \bigvee_{x \in X} x_1 \leq x_2 \wedge x \leq x \\
 &= \bigvee_{x \in X} x_1 \leq x_2 \\
 &= \text{id}_X(x_1^*, x_2).
 \end{aligned} \tag{26.13}$$

□

26.4.3. to put back up

Definition 26.5 (Strong monoidal functor). Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, I_{\mathbf{C}} \rangle$ and $\langle \mathbf{D}, \otimes_{\mathbf{D}}, I_{\mathbf{D}} \rangle$ be two monoidal categories. A *strong monoidal functor* between \mathbf{C} and \mathbf{D} is given by:

1. A functor

$$F : \mathbf{C} \rightarrow \mathbf{D}; \tag{26.14}$$

2. An isomorphism

$$\epsilon : I_{\mathbf{D}} \rightarrow F(I_{\mathbf{C}}); \tag{26.15}$$

3. A natural isomorphism

$$\mu_{x, y} : F(x) \otimes_{\mathbf{D}} F(y) \rightarrow F(x \otimes_{\mathbf{C}} y), \quad \forall x, y \in \mathbf{C}, \tag{26.16}$$

satisfying the following conditions:

- a) *Associativity*: For all objects $x, y, z \in \mathbf{C}$, the following diagram commutes.

$$\begin{array}{ccc}
 (F(x) \otimes_{\mathbf{D}} F(y)) \otimes_{\mathbf{D}} F(z) & \xrightarrow{\alpha_{F(x), F(y), F(z)}^{\mathbf{D}}} & F(x) \otimes_{\mathbf{D}} (F(y) \otimes_{\mathbf{D}} F(z)) \\
 \downarrow & & \downarrow \\
 F(x \otimes_{\mathbf{C}} y) \otimes_{\mathbf{D}} F(z) & & F(x) \otimes_{\mathbf{D}} F(y \otimes_{\mathbf{C}} z) \\
 \downarrow & & \downarrow \\
 F((x \otimes_{\mathbf{C}} y) \otimes_{\mathbf{C}} z) & \xrightarrow[F(a_{x, y, z}^{\mathbf{C}})]{} & F(x \otimes_{\mathbf{C}} (y \otimes_{\mathbf{C}} z))
 \end{array} \tag{26.17}$$

where $a^{\mathbf{C}}$ and $a^{\mathbf{D}}$ are called *associators*.

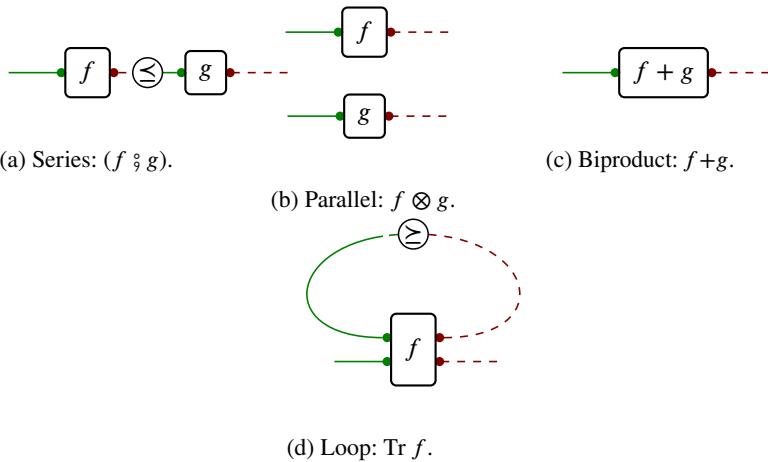
series	$f : A \leftrightarrow B$	$g : B \leftrightarrow C$	$f ; g :$	$A \leftrightarrow C$
sum	$f : A \leftrightarrow B$	$g : A \leftrightarrow B$	$f \vee g :$	$A \leftrightarrow B$
intersection	$f : A \leftrightarrow B$	$g : A \leftrightarrow B$	$f \wedge g :$	$A \leftrightarrow B$
monoidal product	$f : A \leftrightarrow C$	$g : B \leftrightarrow D$	$f \otimes g :$	$A \times B \leftrightarrow C \times D$
product	$f : A \leftrightarrow C$	$g : A \leftrightarrow D$	$f \times g :$	$A \leftrightarrow C + D$
coproduct	$f : A \leftrightarrow C$	$g : B \leftrightarrow C$	$f \sqcup g :$	$A + B \leftrightarrow C$
biproduct	$f : A \leftrightarrow B$	$g : A \leftrightarrow B$	$f + g :$	$A + A \leftrightarrow B + B$
trace	$f : C \times A \leftrightarrow C \times B$	-	$\text{Tr}_{A,B}^C(f) :$	$A \leftrightarrow B$

Table 26.2.: Various composition operations on design problems (i.e. morphisms) in \mathbf{DP} .

b) *Unitality*: For all $x \in \mathbf{C}$, the following diagrams commute:

$$\begin{array}{ccc}
 I_{\mathbf{D}} \otimes_{\mathbf{D}} F(x) & \xrightarrow{\epsilon \otimes \text{id}} & F(I_{\mathbf{C}}) \otimes_{\mathbf{D}} F(x) \\
 \downarrow & & \downarrow \\
 F(x) & \xleftarrow[F(l_x^{\mathbf{C}})]{} & F(I_{\mathbf{C}} \otimes_{\mathbf{C}} x)
 \end{array}
 \quad
 \begin{array}{ccc}
 F(x) \otimes_{\mathbf{D}} I_{\mathbf{D}} & \xrightarrow{\text{id} \otimes \epsilon} & F(x) \otimes_{\mathbf{D}} F(I_{\mathbf{C}}) \\
 \downarrow & & \downarrow \\
 F(x) & \xleftarrow[F(r_x^{\mathbf{C}})]{} & F(x \otimes_{\mathbf{C}} I_{\mathbf{C}})
 \end{array} \tag{26.18}$$

where $l^{\mathbf{C}}$ and $r^{\mathbf{C}}$ represent the left and right *unitors*.



27. Ordering design problems

to write

Contents

27.1. Ordering DPs	200
27.2. Restrictions and alternatives	201
27.3. Interaction with composition	202



27.1. Ordering DPs

We claimed that category theory is an efficient language for talking about *structure*, and showed how the category **DP** could accommodate all the basic operations required by a theory of formal engineering design. Here, we illustrate some of the applications and advantages of **DP** in reasoning about and solving design problems, starting with the fact that that **DP** is compact closed, which allows us to compose and reason about “design problems of design problems”.

Definition 27.1 (Order on **DP**). Suppose that A and B are posets, and that $f, g : A \leftrightarrow B$ are design problems. We say that f implies g , denoted $f \succeq_{\text{DP}} g$, if $f(a^*, b) \leq g(a^*, b)$ in **Bool**, for all $a \in A$ and $b \in B$. In other words, if the fact that f is feasible implies that g is feasible. We diagrammatically represent the relation $f \succeq_{\text{DP}} g$ as in Fig. 27.1.

AC: I don't think it is particularly useful here to show a figure like this. I would draw the homset as a bubble and put two points in there. Also note that this figure was from before when we used the implication arrow instead of \leq .

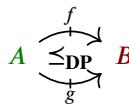


Figure 27.1.: The design problem f implies the design problem g .

Remark 27.2. For any functionality-resource pair A, B , we denote by $1_{A,B}$ the design problem which is always feasible. We denote by $0_{A,B}$ the design problem which is never feasible, for any functionality-resource pair A, B .

Lemma 27.3. $\text{Hom}_{\text{DP}}(A, B)$ is a bounded lattice with union \vee as meet, intersection \wedge as join, least upper bound $1_{A,B}$ and greatest lower bound $0_{A,B}$.

Proof. First of all, we need to prove that $\text{Hom}_{\text{DP}}(A, B)$ is a poset. To prove this, we check the following:

- *Reflexivity:* Given $f \in \text{Hom}_{\text{DP}}(A, B)$, $f \succeq_{\text{DP}} f$ is always true.
- *Antisymmetry:* Given $f, g \in \text{Hom}_{\text{DP}}(A, B)$, if $f \succeq_{\text{DP}} g$ and $g \succeq_{\text{DP}} f$, then $f = g$.
- *Transitivity:* Given $f, g, h \in \text{Hom}_{\text{DP}}(A, B)$, $f \succeq_{\text{DP}} g$ and $g \succeq_{\text{DP}} h$, then $f \succeq_{\text{DP}} h$.

Therefore, Hom_{DP} is a poset. Furthermore, consider two design problems $f, g \in \text{Hom}_{\text{DP}}(A, B)$. Their least upper bound (join) is $f \wedge g$, since it is the least design problem implying both f and g . Their greatest lower bound (meet), instead, is $f \vee g$, since it is the greatest design problem implied by both f and g . This proves that Hom_{DP} is a lattice. To prove that it is bounded, we identify the top element as $1_{A,B}$ (it implies

all other design problems) and the bottom element as $0_{A,B}$ (it is implied by all the other design problems). \square

27.2. Restrictions and alternatives

27.2.1. Union of Design Problems

Let $f : A \leftrightarrow B$ and $g : A \leftrightarrow B$ be design problems. We define the *union* $f \vee g$ to be the design problem which is feasible whenever *either* f or g is feasible. This models f and g as interchangeable technologies: either one can replace the other.

Definition 27.4 (Union of design problems). Given two design problems $f : A \leftrightarrow B$ and $g : A \leftrightarrow B$, their *union* $f \vee g : A \leftrightarrow B$ is defined by

$$(f \vee g) : A^{\text{op}} \times B \rightarrow_{\text{Pos}} \text{Bool} \\ (a^*, b) \mapsto f(a^*, b) \vee g(a^*, b), \quad (27.1)$$

and represented as in Fig. 27.2.

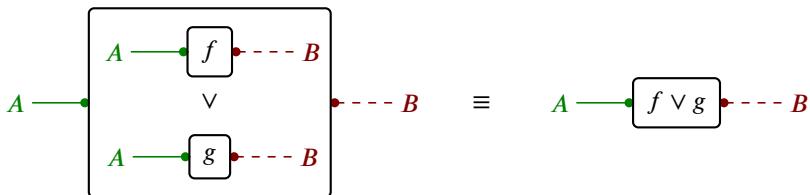


Figure 27.2.: Diagrammatic representation of the union of design problems.

Example 27.5. Jeb's Spaceship Parts is locked in a deadly rivalry with Starshow Bob to supply engines for the new X103 space orbiter. Neither knows the exact operational scenario that the X103 will encounter, but have provided a range of performance benchmarks for their engines (Fig. 27.3). Back at NASA headquarters, Beau has uploaded Jeb and Bob's

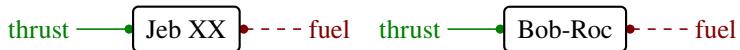


Figure 27.3.: Example of two engine producers.

data in order to construct the design problem reported in Fig. 27.4.



Figure 27.4.: Example of the union of the engine design problems.

27.2.2. Intersection of Design Problems

Given two design problems $f, g : \mathbf{A} \leftrightarrow \mathbf{B}$, we can define a design problem $f \wedge g$ that is feasible if only if f and g are both feasible. We call $f \wedge g$ the *intersection* of f and g . One interpretation of $f \wedge g$ is that f and g 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 27.6 (Intersection of design problems). Given design problems $f : \mathbf{A} \leftrightarrow \mathbf{B}$ and $g : \mathbf{A} \leftrightarrow \mathbf{B}$, their *intersection* is denoted $(f \wedge g) : \mathbf{A} \leftrightarrow \mathbf{B}$, defined by:

$$(f \wedge g) : \mathbf{A}^{\text{op}} \times \mathbf{B} \rightarrow_{\text{Pos}} \mathbf{Bool} \quad (27.2)$$

$$\langle a^*, b \rangle \mapsto f(a^*, b) \wedge g(a^*, b),$$

and represented as in Fig. 27.5.

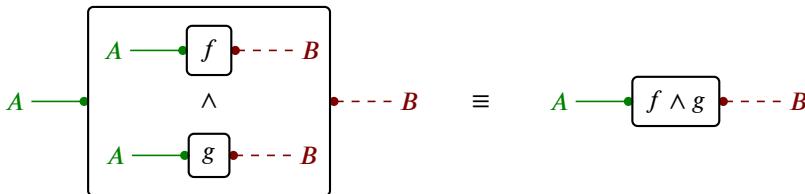


Figure 27.5.: Diagrammatic representation of the intersection of design problems.

We can directly generalize the intersection $f \wedge g$ by allowing f and g to have different domain and codomains, $f : \mathbf{A} \leftrightarrow \mathbf{B}$ and $g : \mathbf{C} \leftrightarrow \mathbf{D}$. We call this putting two design problems “in parallel”.

27.3. Interaction with composition

Show that all composition operations are monotone

In the previous section, we introduced the concept of order in **DP**, and proved that hom-sets of **DP** form a bounded lattice. In this section, we show that composition (Definition 23.6), monoidal product (Definition 25.15), and trace (Definition 26.3) of design problems are order-preserving operations.

Lemma 27.7. Given $f_1, f_2 \in \text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B})$ and $g_1, g_2 \in \text{Hom}_{\mathbf{DP}}(\mathbf{B}, \mathbf{C})$, with $f_1 \succeq_{\mathbf{DP}} f_2$ and $g_1 \succeq_{\mathbf{DP}} g_2$, one has $f_1 ; g_1 \succeq_{\mathbf{DP}} f_2 ; g_2$, i.e., series composition (Definition 23.6) is order-preserving on **DP**.

Proof. We have

$$\begin{aligned}
 (f_1 \circ g_1)(\textcolor{red}{a}^*, \textcolor{blue}{c}) &= \bigvee_{b \in B} f_1(\textcolor{red}{a}^*, b) \wedge g_1(\textcolor{blue}{b}^*, \textcolor{blue}{c}) \\
 &\geq_{\mathbf{DP}} \bigvee_{b \in B} f_2(\textcolor{red}{a}^*, b) \wedge g_2(\textcolor{blue}{b}^*, \textcolor{blue}{c}) \\
 &= (f_2 \circ g_2)(\textcolor{red}{a}^*, \textcolor{blue}{c}).
 \end{aligned} \tag{27.3}$$

Therefore \circ is order-preserving on \mathbf{DP} . \square

Lemma 27.8. Given $f_1, f_2 \in \text{Hom}_{\mathbf{DP}}(\textcolor{violet}{A}, \textcolor{red}{B})$ and $g_1, g_2 \in \text{Hom}_{\mathbf{DP}}(\textcolor{violet}{C}, \textcolor{red}{D})$, with $f_1 \geq_{\mathbf{DP}} f_2$ and $g_1 \geq_{\mathbf{DP}} g_2$, one has $f_1 \otimes g_1 \geq_{\mathbf{DP}} f_2 \otimes g_2$, i.e. monoidal product (Definition 25.15) preserves order on \mathbf{DP} .

Proof. We have

$$\begin{aligned}
 (f_1 \otimes g_1)(\langle \textcolor{red}{a}, \textcolor{blue}{c} \rangle^*, \langle \textcolor{red}{b}, \textcolor{blue}{d} \rangle) &= f_1(\textcolor{red}{a}^*, \textcolor{red}{b}) \wedge g_1(\textcolor{blue}{c}^*, \textcolor{blue}{d}) \\
 &\geq_{\mathbf{DP}} f_2(\textcolor{red}{a}^*, \textcolor{red}{b}) \wedge g_2(\textcolor{blue}{c}^*, \textcolor{blue}{d}) \\
 &= (f_2 \otimes g_2)(\langle \textcolor{red}{a}, \textcolor{blue}{c} \rangle^*, \langle \textcolor{red}{b}, \textcolor{blue}{d} \rangle)
 \end{aligned} \tag{27.4}$$

Therefore, \otimes is order-preserving on \mathbf{DP} . \square

Lemma 27.9. Given $f_1, f_2 \in \text{Hom}_{\mathbf{DP}}(\textcolor{violet}{C} \times \textcolor{violet}{A}, \textcolor{violet}{C} \times \textcolor{red}{B})$ with $f_1 \geq_{\mathbf{DP}} f_2$, one has

$$\text{Tr}_{\textcolor{violet}{A}, \textcolor{red}{B}}^{\textcolor{violet}{C}}(f_1) \geq_{\mathbf{DP}} \text{Tr}_{\textcolor{violet}{A}, \textcolor{red}{B}}^{\textcolor{violet}{C}}(f_2), \tag{27.5}$$

i.e., trace (Definition 26.3) preserves order on \mathbf{DP} .

Proof. One has

$$\begin{aligned}
 \text{Tr}_{\textcolor{violet}{A}, \textcolor{red}{B}}^{\textcolor{violet}{C}}(f_1)(\textcolor{red}{a}^*, \textcolor{red}{b}) &= \bigvee_{c \in C} f_1(\langle \textcolor{blue}{c}, \textcolor{red}{a} \rangle^*, \langle \textcolor{blue}{c}, \textcolor{red}{b} \rangle) \\
 &\geq_{\mathbf{DP}} \bigvee_{c \in C} f_2(\langle \textcolor{blue}{c}, \textcolor{red}{a} \rangle^*, \langle \textcolor{blue}{c}, \textcolor{red}{b} \rangle) \\
 &= \text{Tr}_{\textcolor{violet}{A}, \textcolor{red}{B}}^{\textcolor{violet}{C}}(f_2)(\textcolor{red}{a}^*, \textcolor{red}{b}).
 \end{aligned} \tag{27.6}$$

Therefore, Tr is order-preserving on \mathbf{DP} . \square

28. Constructing design problems

to write

Contents

28.1. Creating design problems from catalogues	206
28.2. Companions and conjoint	208
28.3. Sum and intersection with companion and conjoints	210
28.4. Monoidal DPs	212



28.1. Creating design problems from catalogues

Definition 28.1 (Span). Given a category \mathbf{C} , a *span* from an object x to an object y is a diagram of the form

$$\begin{array}{ccc} & z & \\ f \swarrow & & \searrow g \\ x & & y \end{array} \quad (28.1)$$

where z is some other object of \mathbf{C} .

Example 28.2. Consider the category **Berg**, introduced in Section 5.2. An example of span in this category is reported in Fig. 28.1. Recall that Matterhorn Peak, Jungfrau Peak,

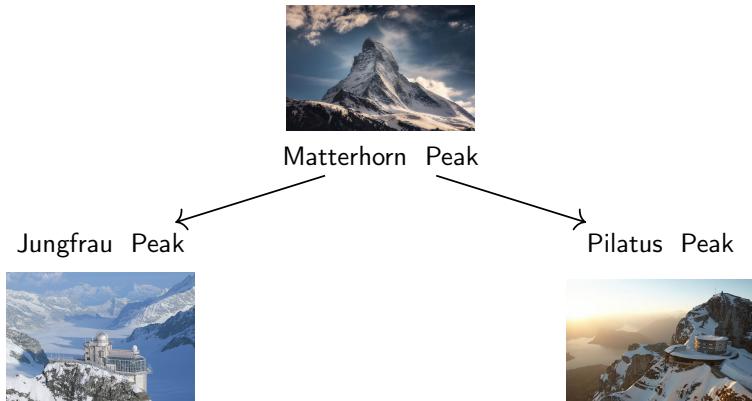


Figure 28.1.: Swiss peaks can be thought of as a span in **Trek**.

and Pilatus Peak are objects of **Trek**, and the arrows are morphisms in **Trek** (paths from one location to the other).

We have already defined a DPI like this.

Definition 28.3 (Catalogue). A *catalogue* is a span in **Pos**. It thus consists of 3 posets I , F , R . We call them implementation space, functionality space, and requirements space, respectively. We need to define two maps $\text{prov} : I \rightarrow F$ (an implementation **provides** a functionality) and $\text{req} : I \rightarrow R$ (an implementation **requires** resources):

$$\begin{array}{ccc} & I & \\ \text{prov} \swarrow & & \searrow \text{req} \\ F & & R \end{array}$$

Definition 28.4 (Design problem induced by a catalogue). Every catalogue $\langle \textcolor{brown}{I}, \text{prov}, \text{req} \rangle$ induces a design problem of the form $d : \textcolor{green}{F}^{\text{op}} \times \textcolor{red}{R} \rightarrow \text{Bool}$,

$$d : \textcolor{green}{F}^{\text{op}} \times \textcolor{red}{R} \rightarrow \text{Bool}$$

$$\langle \textcolor{green}{f}^*, \textcolor{red}{r} \rangle \mapsto \bigvee_{i \in \textcolor{brown}{I}} (\text{prov}(\textcolor{brown}{i}) \succeq_{\textcolor{green}{F}} \textcolor{green}{f}) \wedge (\text{req}(i) \leq_{\textcolor{red}{R}} \textcolor{red}{r})$$

Example 28.5. We now want to revisit the leading example of Chapter 9 with the newly introduced co-design perspective. Let's consider a list of electrical motors as in Table 28.1.

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

Table 28.1.: A simplified catalogue of motors.

We can think of this as a catalogue of electric motors $\langle \textcolor{brown}{I}_{\text{EM}}, \text{prov}_{\text{EM}}, \text{req}_{\text{EM}} \rangle$. In particular, the set of implementations collects all the motor models, which we can specify using the motor IDs:

$$\textcolor{brown}{I}_{\text{EM}} = \{1204, 1206, 1207, 2267, 2279, 1478, 2299\}. \quad (28.2)$$

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

$$\textcolor{green}{F} = \mathbb{R} \times \{\text{kg} \cdot \text{cm}\}, \quad \textcolor{red}{R} = (\mathbb{R} \times \{g\}) \times (\mathbb{R} \times \{W\}) \times (\mathbb{R} \times \{\text{USD}\}),$$

by considering the units as discussed in Section 3.3. The correspondences are given by the details in Table 28.1. For instance, we have

$$\text{prov}_{\text{EM}}(1204) = 0.18, \quad \text{req}_{\text{EM}}(1204) = \langle 60 \text{ g}, 2.34 \text{ W}, 19.95 \text{ USD} \rangle. \quad (28.3)$$

The catalogue induces a design problem d_{EM} with diagrammatic form as in Fig. 28.2. In particular, we can query the design problem for combinations of **functionalities** and **resources**. For instance:

$$d(0.2 \text{ kg} \cdot \text{cm}, \langle 50.0 \text{ g}, 2.0 \text{ W}, 15.0 \text{ USD} \rangle) = \perp, \quad (28.4)$$

since no listed model can provide 0.2 kg · cm **torque** by requiring the set of **resources** $\langle 50.0 \text{ g}, 2.0 \text{ W}, 15.0 \text{ USD} \rangle$ or less.

finish with design problem description

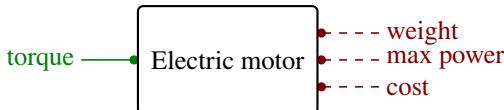


Figure 28.2.: Electric motor design problem.

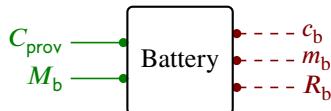


Figure 28.3.: The battery design problem.

28.2. Companions and conjoint

We round out our discussion of **DP** by introducing two formulae 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* \hat{f} and *conjoint* \check{f} . Many of the design problems that we have introduced can be realized as companions and conjoints of appropriate monotone maps.

Definition 28.6 (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 $\hat{f} : P \leftrightarrow Q$, and its *conjoint*, denoted $\check{f} : Q \leftrightarrow P$ as

$$\hat{f}(p^*, q) := f(p) \leq_Q q \quad \text{and} \quad \check{f}(q^*, p) := q \leq_Q f(p). \quad (28.5)$$

Lemma 28.7. Both the companion and conjoint constructions from Definition 28.6 are functorial from **Pos** to **DP**: they preserve identities and composition.

Technology	Specific energy [J/kg]	Specific cost [J/\$]	Life [# cycles]
NiMH	100.0	3.41	500
NiH2	45.0	10.5	20,000
LCO	195.0	2.84	750
LMO	150.0	2.84	500
NiCad	30.0	7.50	500
SLA	30.0	7.00	500
LiPo	250.0	2.50	600
LFP	90.0	1.50	1,500

Table 28.2.: Specifications of common battery technologies [8].

Proof. We will show that the companion and conjoint are functors of the following forms:

$$\hat{(\cdot)} : \mathbf{Pos} \rightarrow \mathbf{DP} \quad \text{and} \quad \check{(\cdot)} : \mathbf{Pos} \rightarrow \mathbf{DP}^{\text{op}}. \quad (28.6)$$

First, we see that they send the identity monotone map $f(p) = p$ to the unit id_P for any poset P , because

$$\begin{aligned} \hat{\text{id}}(\textcolor{violet}{p}_1^*, \textcolor{violet}{p}_2) &= (\textcolor{violet}{p}_1 \leq_P \textcolor{red}{p}_2) \\ &= \check{\text{id}}(\textcolor{violet}{p}_1^*, \textcolor{violet}{p}_2). \end{aligned} \quad (28.7)$$

Now suppose that $f : P \rightarrow_{\mathbf{Pos}} Q$ and $g : Q \rightarrow_{\mathbf{Pos}} R$ are given. We first show that $\check{g} ; \check{f} = \widehat{f ; g}$. For any $p \in P$ and $r \in R$, one has

$$\begin{aligned} (\check{g} ; \check{f})(\textcolor{violet}{r}^*, \textcolor{red}{p}) &= \bigvee_{q \in Q} \check{g}(\textcolor{violet}{r}^*, \textcolor{red}{q}) \wedge \check{f}(\textcolor{violet}{q}^*, \textcolor{red}{p}) \\ &= \bigvee_{q \in Q} (\textcolor{violet}{r} \leq_R g(\textcolor{red}{q})) \wedge (\textcolor{violet}{q} \leq_Q f(\textcolor{red}{p})) \\ &= \textcolor{violet}{r} \leq_R g(f(\textcolor{red}{p})) \\ &= (\widehat{f ; g})(\textcolor{violet}{r}^*, \textcolor{red}{p}). \end{aligned} \quad (28.8)$$

Similarly, we can prove that $\hat{f} ; \hat{g} = \widehat{f ; g}$:

$$\begin{aligned} (\hat{f} ; \hat{g})(\textcolor{violet}{p}^*, \textcolor{red}{r}) &= \bigvee_{q \in Q} \hat{f}(\textcolor{violet}{p}^*, \textcolor{red}{q}) \wedge \hat{g}(\textcolor{violet}{q}^*, \textcolor{red}{r}) \\ &= \bigvee_{q \in Q} (f(\textcolor{violet}{p}) \leq_Q \textcolor{red}{q}) \wedge (g(\textcolor{violet}{q}) \leq_R \textcolor{red}{r}) \\ &= g(f(\textcolor{violet}{p})) \leq_R \textcolor{red}{r} \\ &= (\widehat{f ; g})(\textcolor{violet}{p}^*, \textcolor{red}{r}). \end{aligned} \quad (28.9)$$

□

Example 28.8. The identity design problem $\text{id}_A : \textcolor{violet}{A} \leftrightarrow \textcolor{red}{A}$ is the companion (and the conjoint) of the identity map $\text{id}'_A : A \rightarrow_{\mathbf{Pos}} A$. This is easy to check, as

$$\begin{aligned} \hat{\text{id}}'_A(\textcolor{violet}{a}_1^*, \textcolor{red}{a}_2) &= \text{id}'_A(\textcolor{violet}{a}_1) \leq \textcolor{red}{a}_2 \\ &= \textcolor{violet}{a}_1 \leq \textcolor{red}{a}_2 \\ &= \text{id}_A(\textcolor{violet}{a}_1^*, \textcolor{red}{a}_2). \end{aligned} \quad (28.10)$$

Example 28.9. The coproduct injections ι_A, ι_B for design problems are the companions of the coproduct injections for the disjoint union.

Example 28.10. The product projections π_A, π_B for design problems are the conjoints of the coproduct injections for the disjoint union.

Interesting implications Consider a poset A , which can be thought of as a map $f : 1 \rightarrow A$. By taking the companion of f one gets

$$\begin{aligned}\hat{f} &: 1 \leftrightarrow A \\ \langle 1, a \rangle &\mapsto f(1) \leq a.\end{aligned}\tag{28.11}$$

By taking the conjoint, one gets

$$\begin{aligned}\check{f} &: A \leftrightarrow 1 \\ \langle a^*, 1 \rangle &\mapsto a \leq f(1).\end{aligned}\tag{28.12}$$

These two cases represent design problems with either *constant* resources or constant, functionalities, respectively.

28.3. Sum 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 28.11 (Diagonal function). Define the *diagonal function* $\Delta_P : P \rightarrow P \times P$:

$$\begin{aligned}\Delta_P &: P \rightarrow P \times P, \\ p &\mapsto \langle p, p \rangle.\end{aligned}\tag{28.13}$$

Definition 28.12 (Codiagonal function). Define the *codiagonal function* $\Diamond_P : P + P \rightarrow P$:

$$\begin{aligned}\Diamond_P &: P + P \rightarrow P, \\ \langle 1, p \rangle &\mapsto p, \\ \langle 2, p \rangle &\mapsto p.\end{aligned}\tag{28.14}$$

Using the diagonal function, Lemma 47.15 can be rewritten as the following lemma.

Lemma 28.13. Given $f, g : A \leftrightarrow B$, we have:

$$f \vee g = \Diamond_A \circ (f + g) \circ \hat{\Diamond}_B.\tag{28.15}$$

Proof. First of all, note that

$$\begin{aligned}\Diamond_A &: A \leftrightarrow A + A \\ \langle a_1^*, \langle 1, a_2 \rangle \rangle &\mapsto a_1 \leq a_2 \\ \langle a_1^*, \langle 1, a_3 \rangle \rangle &\mapsto a_1 \leq a_3\end{aligned}\tag{28.16}$$

and

$$\begin{aligned}\hat{\Diamond}_B &: B + B \leftrightarrow B \\ \langle \langle 1, b_1 \rangle^*, b_3 \rangle &\mapsto b_1 \leq b_3 \\ \langle \langle 2, b_2 \rangle^*, b_3 \rangle &\mapsto b_2 \leq b_3\end{aligned}\tag{28.17}$$

We start by looking at $\underbrace{\Diamond_A \circ (f + g)}_{\star} : \mathbf{A} \leftrightarrow \mathbf{B} + \mathbf{B}$.

$$\begin{aligned}\star(\langle \mathbf{a}^*, \mathbf{b} \rangle) &= \bigvee_{a' \in A+A} \Diamond_A(\langle \mathbf{a}^*, \mathbf{a}' \rangle) \wedge (f+g)(\langle \mathbf{a}'^*, \mathbf{b} \rangle) \\ &= \left(\bigvee_{\langle 1, a' \rangle \in A+A} (\mathbf{a} \leq \mathbf{a}') \wedge f(\mathbf{a}'^*, \mathbf{b}) \right) \vee \left(\bigvee_{\langle 2, a' \rangle \in A+A} (\mathbf{a} \leq \mathbf{a}') \wedge g(\mathbf{a}'^*, \mathbf{b}) \right) \\ &= f(\mathbf{a}^*, \mathbf{b}) \vee g(\mathbf{a}^*, \mathbf{b}).\end{aligned}\tag{28.18}$$

Let's now look at $\star \circ \widehat{\Diamond}_B : \mathbf{A} \leftrightarrow \mathbf{B}$:

$$\begin{aligned}(\star \circ \widehat{\Diamond}_B)(\mathbf{a}^*, \mathbf{b}') &= \bigvee_{b \in B+B} \star(\mathbf{a}^*, \mathbf{b}) \wedge \widehat{\Diamond}_B(\mathbf{b}^*, \mathbf{b}') \\ &= \left(\bigvee_{\langle 1, b \rangle \in B+B} f(\mathbf{a}^*, \mathbf{b}) \wedge (\mathbf{b} \leq \mathbf{b}') \right) \vee \left(\bigvee_{\langle 2, b \rangle \in B+B} g(\mathbf{a}^*, \mathbf{b}) \wedge (\mathbf{b} \leq \mathbf{b}') \right) \\ &= f(\mathbf{a}^*, \mathbf{b}') \vee g(\mathbf{a}^*, \mathbf{b}').\end{aligned}\tag{28.19}$$

□

Similarly, using the codiagonal function, one can prove the following.

Lemma 28.14. Given $f, g : \mathbf{A} \leftrightarrow \mathbf{B}$, we have:

$$f \wedge g = \widehat{\Delta}_A \circ (f + g) \circ \check{\Delta}_B.\tag{28.20}$$

Proof. First, note that

$$\begin{aligned}\widehat{\Delta}_A : \mathbf{A} &\leftrightarrow \mathbf{A} \times \mathbf{A} \\ \langle \mathbf{a}_1^*, \langle \mathbf{a}_2, \mathbf{a}_3 \rangle \rangle &\mapsto \Delta_A(\mathbf{a}_1) \leq \langle \mathbf{a}_2, \mathbf{a}_3 \rangle \\ &= \langle \mathbf{a}_1, \mathbf{a}_1 \rangle \leq \langle \mathbf{a}_2, \mathbf{a}_3 \rangle \\ &= (\mathbf{a}_1 \leq \mathbf{a}_2) \wedge (\mathbf{a}_1 \leq \mathbf{a}_3).\end{aligned}\tag{28.21}$$

and

$$\begin{aligned}\check{\Delta}_B : \mathbf{B} \times \mathbf{B} &\leftrightarrow \mathbf{B} \\ \langle \langle \mathbf{b}_1, \mathbf{b}_2 \rangle^*, \mathbf{b}_3 \rangle &\mapsto \langle \mathbf{b}_1, \mathbf{b}_2 \rangle \leq \Delta_B(\mathbf{b}_3) \\ &= (\mathbf{b}_1 \leq \mathbf{b}_3) \wedge (\mathbf{b}_2 \leq \mathbf{b}_3).\end{aligned}\tag{28.22}$$

We start by looking at $\widehat{\Delta}_A \circ (f + g) : \mathbf{A} \leftrightarrow \mathbf{B} + \mathbf{B}$:

$$\left(\widehat{\Delta}_A \circ (f + g) \right) (\langle \mathbf{a}^*, \mathbf{b} \rangle) = \bigvee\tag{28.23}$$

Adjust signatures, have to find a good way to write it down



Unlike $\check{\Diamond} = \text{split}$ and $\hat{\Diamond} = \text{fuse}$, $\check{\Delta}$ and $\hat{\Delta}$ do not have an intuitive diagrammatic representation.

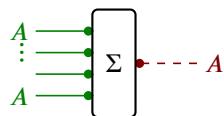
28.4. Monoidal DPs

Intro, reference "monoidal poset".

Definition 28.15 (Sum of resources). If the poset A is monoidal, then the *sum* of n copies of A is a design problem given by

$$\begin{aligned}\sum : (A^n)^{\text{op}} \times A &\rightarrow_{\text{Pos}} \mathbf{Bool} \\ \langle a_1^*, \dots, a_n^*, a_{\text{resource}} \rangle &\mapsto (a_1 + \dots + a_n \leq_A a_{\text{resource}}).\end{aligned}$$

Clearly \sum is monotone. Diagrammatically:



Now do sum of functionality, and then the mixed one

Now show that it is the composition of conjoint and companion of "+".

Part C.

Computation

29. From math to implementation

to write



Discuss the 4 phases

example: fast inverse square root

30. Solving

to write

Contents

30.1. Types of queries	218
30.2. Computational representation of DPIs	218
30.3. Problem statement and summary of results	219
30.4. Composition operators for design problems	220
30.5. Decomposition of MCDPs	222
30.6. Monotonicity as compositional property	224
30.7. Solution of MCDPs	228
30.8. Example: Optimizing over the natural numbers	229
30.9. Complexity of the solution	232
30.10 Extended Numerical Examples	234
30.11 Monotonicity and fixed points	243



30.1. Types of queries

30.2. Computational representation of DPis

It is useful to also describe a design problem as a map from functionality to resources or viceversa that abstracts over implementations.

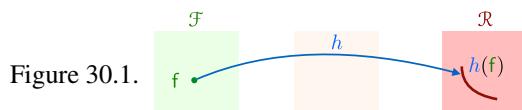
A useful analogy is the state space representation *vs* the transfer function representation of a linear time-invariant system: the state space representation is richer, but we only need the transfer function to characterize the input-output response.

define the map from functionality to upper sets

Definition 30.1. Given a DPI $\langle F, R, I, \text{prov}, \text{req} \rangle$, define the map $h : F \rightarrow \mathcal{AR}$ that associates to each functionality f the objective function of Section 22.2, which is the set of minimal resources necessary to realize f :

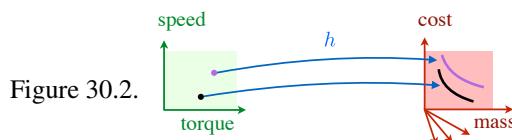
$$\begin{aligned} h : F &\rightarrow \mathcal{AR}, \\ f &\mapsto \underset{\leq_R}{\text{Min}} \{ \text{req}(i) \mid (i \in I) \wedge (f \leq \text{prov}(i)) \}. \end{aligned}$$

If a certain functionality f is infeasible, then $h(f) = \emptyset$.



add remark of when this might fail (antichains do not capture the uppersets)

Example 30.2. In the case of the motor design problem, the map h assigns to each pair of $\langle \text{speed}, \text{torque} \rangle$ the achievable trade-off of cost , mass , and other resources (Fig. 30.2). The antichains are depicted as continuous curves, but they could also be composed by a finite set of points.



By construction, h is monotone (Definition 13.1), which means that

$$f_1 \leq_F f_2 \Rightarrow h(f_1) \leq_{\mathcal{AR}} h(f_2),$$

where $\leq_{\mathcal{AR}}$ is the order on antichains defined in Lemma 45.3. Monotonicity of h means that if the functionality f is increased the antichain of resources will go “up” in the poset of antichains \mathcal{AR} , and at some point it might reach the top of \mathcal{AR} , which is the empty set, meaning that the problem is not feasible.

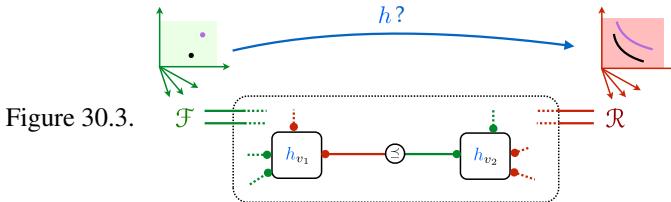
describe equally the map φ

30.3. Problem statement and summary of results

Given an arbitrary graph of design problems, and assuming we know how to solve each problem separately, we ask whether we can solve the *co-design* problem optimally.

Problem. Suppose that we are given a CDPI $\langle \mathcal{F}, \mathcal{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, and that we can evaluate the map h_v for all $v \in \mathcal{V}$. Given a required functionality $f \in \mathcal{F}$, we wish to find the *minimal* resources in \mathcal{R} for which there exists a feasible implementation vector that makes all sub-problems feasible at the same time and all co-design constraints satisfied; or, if none exist, provide a certificate of infeasibility.

In other words, given the maps $\{h_v, v \in \mathcal{V}\}$ for the subproblems, one needs to evaluate the map $h : \mathcal{F} \rightarrow \mathcal{AR}$ for the entire CDPI (Fig. 30.3).



The rest of the paper will provide a solution to Section 30.3, under the following assumptions:

1. The posets \mathcal{F}, \mathcal{R} are complete partial orders (Definition 30.31).
2. The maps h is Scott continuous (Definition 30.33).

30.3.1. Expressivity of MCDPs

The results are significant because MCDPs induce a rich family of optimization problems. We are not assuming, let alone strong properties like convexity, even weaker properties like differentiability or continuity of the constraints. In fact, we are not even assuming that functionality and resources are continuous spaces; they could be arbitrary discrete posets. (In that case, completeness and Scott continuity are trivially satisfied.)

Moreover, even assuming topological continuity of all spaces and maps considered, MCDPs are strongly not convex. What makes them nonconvex is the possibility of introducing feedback interconnections. To show this, I will give an example of a 1-dimensional problem with a continuous h for which the feasible set is disconnected.

Example 30.3. Consider the CDPI in Fig. 30.4a. The minimal resources $M \subseteq \mathcal{AR}$ are the objectives of this optimization problem:

$$M \doteq \begin{cases} \text{using } f, r \in \mathcal{F} = \mathcal{R}, \\ \text{Min}_{\leq} r, \\ r \in h(f), \\ r \leq f. \end{cases}$$



Figure 30.4.: One feedback connection and a topologically continuous h are sufficient to induce a disconnected feasible set.

The feasible set $\Phi \subseteq F \times R$ is the set of functionality and resources that satisfy the constraints $r \in h(f)$ and $r \leq f$:

$$\Phi = \{ \langle f, r \rangle \in F \times R : (r \in h(f)) \wedge (r \leq f) \}. \quad (30.1)$$

The projection P of Φ to the functionality space is:

$$P = \{ f \mid \langle f, r \rangle \in \Phi \}.$$

In the scalar case ($F = R = (\overline{\mathbb{R}}_+, \leq)$), the map $h : F \rightarrow \mathcal{A}R$ is simply a map $h : \overline{\mathbb{R}}_+ \rightarrow \overline{\mathbb{R}}_+$. The set P of feasible functionality is described by

$$P = \{ f \in \overline{\mathbb{R}}_+ : h(f) \leq f \}. \quad (30.2)$$

Fig. 30.4b shows an example of a continuous map h that gives a disconnected feasible set P . Moreover, P is disconnected under any order-preserving nonlinear re-parametrization.

30.3.2. Approach

The strategy to obtain these results consists in reducing an arbitrary interconnection of design problems to considering only a finite number of composition operators (series, parallel, and feedback). Section 30.4 defines these composition operators. Section 30.5 shows how to turn a graph into a tree, where each junction is one of the three operators. Given the tree representation of an MCDPs, we will be able to give inductive arguments to prove the results.

30.4. Composition operators for design problems

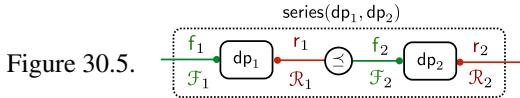
This section defines a handful of composition operators for design problems. Later, Section 30.5 will prove that any co-design problem can be described in terms of a subset of these operators.

Definition 30.4 (series). The series composition of two DPIs $dp_1 = \langle F_1, R_1, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $dp_2 = \langle F_2, R_2, I_2, \text{prov}_2, \text{req}_2 \rangle$, for which $F_2 = R_1$, is

$$\text{series}(dp_1, dp_2) \doteq \langle F_1, R_2, I, \text{prov}, \text{req} \rangle,$$

where:

$$\begin{aligned} \textcolor{brown}{I} &= \{\langle i_1, i_2 \rangle \in I_1 \times I_2 \mid \text{req}_1(i_1) \leq_{R_1} \text{prov}_2(i_2)\}, \\ \text{prov} &: \langle i_1, i_2 \rangle \mapsto \text{prov}_1(i_1), \\ \text{req} &: \langle i_1, i_2 \rangle \mapsto \text{req}_2(i_2). \end{aligned}$$

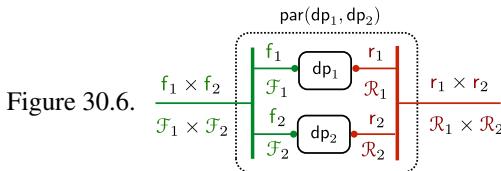


Definition 30.5 (par). The parallel composition of two DPIs $\text{dp}_1 = \langle F_1, R_1, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $\text{dp}_2 = \langle F_2, R_2, I_2, \text{prov}_2, \text{req}_2 \rangle$ is

$$\text{par}(\text{dp}_1, \text{dp}_2) \doteq \langle F_1 \times F_2, R_1 \times R_2, I_1 \times I_2, \text{prov}, \text{req} \rangle,$$

where:

$$\begin{aligned} \text{prov} &: \langle i_1, i_2 \rangle \mapsto \langle \text{prov}_1(i_1), \text{prov}_2(i_2) \rangle, \\ \text{req} &: \langle i_1, i_2 \rangle \mapsto \langle \text{req}_1(i_1), \text{req}_2(i_2) \rangle. \end{aligned} \quad (30.3)$$



Definition 30.6 (loop). Suppose dp is a DPI with factored functionality space $F_1 \times R$:

$$\text{dp} = \langle F_1 \times R, R, I, \langle \text{prov}_1, \text{prov}_2 \rangle, \text{req} \rangle.$$

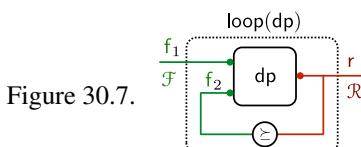
Then we can define the DPI $\text{loop}(\text{dp})$ as

$$\text{loop}(\text{dp}) \doteq \langle F_1, R, I', \text{prov}_1, \text{req} \rangle,$$

where $I' \subseteq I$ limits the implementations to those that respect the additional constraint $\text{req}(i) \leq \text{prov}_2(i)$:

$$I' = \{i \in I : \text{req}(i) \leq \text{prov}_2(i)\}.$$

This is equivalent to “closing a loop” around dp with the constraint $f_2 \geq r$ (Fig. 30.7).



The operator loop is asymmetric because it acts on a design problem with 2 functionalities and 1 resources. We can define a symmetric feedback operator loopb as in Fig. 30.8a, which can be rewritten in terms of loop, using the construction in Fig. 30.8b.

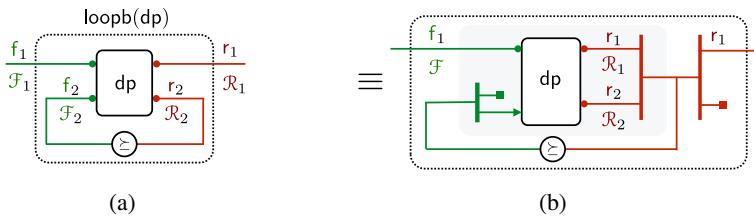


Figure 30.8.: A symmetric operator loopb can be defined in terms of loop .

A “co-product” (see, e.g., [44, Section 2.4]) of two design problems is a design problem with the implementation space $I = I_1 \sqcup I_2$, and it represents the exclusive choice between two possible alternative families of designs.

Definition 30.7 (Co-product). Given two DPIs with same functionality and resources $\text{dp}_1 = \langle F, R, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $\text{dp}_2 = \langle F, R, I_2, \text{prov}_2, \text{req}_2 \rangle$, define their co-product as

$$\text{dp}_1 \sqcup \text{dp}_2 \doteq \langle \textcolor{violet}{F}, \textcolor{red}{R}, \textcolor{brown}{I}_1 \sqcup \textcolor{brown}{I}_2, \text{prov}, \text{req} \rangle,$$

where

$$\begin{aligned} \text{prov} & : i \mapsto \begin{cases} \text{prov}_1(i), & \text{if } i \in I_1, \\ \text{prov}_2(i), & \text{if } i \in I_2, \end{cases} \\ \text{req} & : i \mapsto \begin{cases} \text{req}_1(i), & \text{if } i \in I_1, \\ \text{req}_2(i), & \text{if } i \in I_2. \end{cases} \end{aligned} \quad (30.4)$$

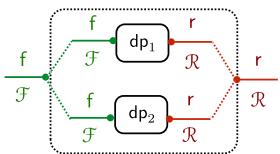


Figure 30.9.

30.5. Decomposition of MCDPs

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).

30.5.1. Equivalence

The definition of equivalence below ensures that two equivalent DPs have the same map from functionality to resources, while one of the DPs can have a slightly larger implementation space.

Definition 30.8. Two DPIs $\langle F, R, I_1, \text{prov}_1, \text{req}_1 \rangle$ and $\langle F, R, I_2, \text{prov}_2, \text{req}_2 \rangle$ are *equivalent* if there exists a map $\varphi : I_2 \rightarrow I_1$ such that $\text{prov}_2 = \text{prov}_1 \circ \varphi$ and $\text{req}_2 = \text{req}_1 \circ \varphi$.

30.5.2. Plumbing

We also need to define “trivial DPIs”, which serve as “plumbing”. These can be built by taking a map $f : \mathbf{F} \rightarrow \mathbf{R}$ and lifting it to the definition of a DPI. The implementation space of a trivial DPI is a copy of the functionality space and there is a 1-to-1 correspondence between functionality and implementation.

Definition 30.9 (Trivial DPIs). Given a map $f : \mathbf{F} \rightarrow \mathbf{R}$, we can lift it to define a trivial DPI $\text{Triv}(f) = \langle \mathbf{F}, \mathbf{R}, \mathbf{F}, \text{Id}_{\mathbf{F}}, f \rangle$, where $\text{Id}_{\mathbf{F}}$ is the identity on \mathbf{F} .

Proposition 30.10. 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 [20]). For example, the CDPI represented in Fig. 30.12a has a minimal AFS that contains the edge $c \rightarrow a$ (Fig. 30.12b).

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. 30.10).

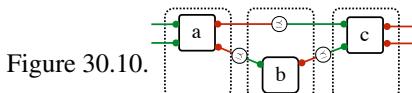


Figure 30.10.

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. 30.11).

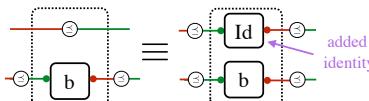


Figure 30.11.

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. 30.13a). In this example, the tree representation (Fig. 30.13b) is

$$\text{loopb}(\text{series}(\text{series}(a, \text{par}(\text{Id}, b)), c)).$$

If the AFS contains multiple edges, then, instead of closing one loop at a time, one 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. 30.14a can be rewritten as Fig. 30.14b. This construction is analogous to the construction used for the analysis of process networks [28] (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

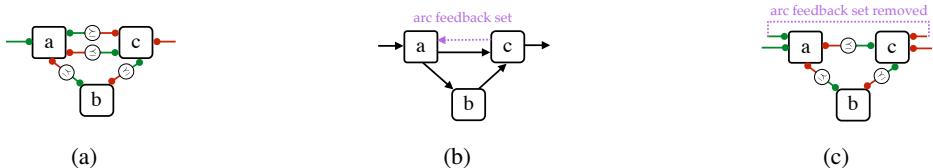


Figure 30.12.: An example co-design diagram with three nodes $\mathcal{V} = \{a, b, c\}$, in which a minimal arc feedback set is $\{c \rightarrow a\}$.

30.6. Monotonicity as compositional property

The first main result of this paper is an invariance result.

Theorem 30.11. The class of MCDPs is closed with respect to interconnection.

Proof. Proposition 30.10 has shown that any interconnection of design problems can be described using the three operators *par*, *series*, and *loop*. Therefore, we just need to check that monotonicity in the sense of ?? is preserved by each operator separately. This is done below in Propositions 30.12 and 30.14. \square

Proposition 30.12. If dp_1 and dp_2 are monotone (??), then also the composition $\text{par}(dp_1, dp_2)$ is monotone.

Proof. We need to refer to the definition of *par* in Definition 30.5 and check the conditions in ???. If F_1, F_2, R_1, R_2 are CPOs, then $F_1 \times F_2$ and $R_1 \times R_2$ are CPOs as well.

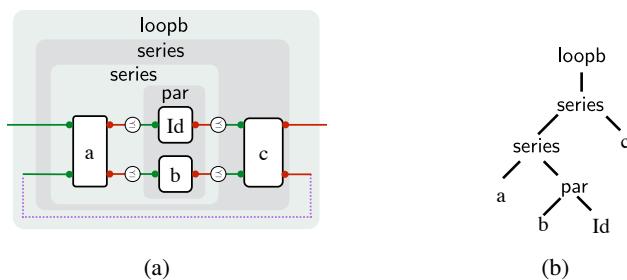


Figure 30.13.: Tree representation for the co-design diagram in Fig. 30.12a.

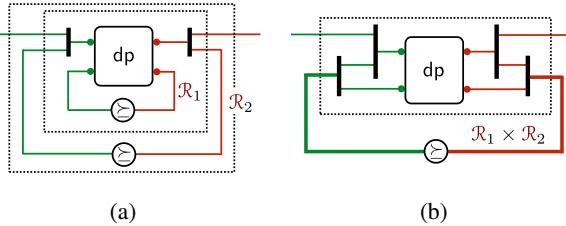


Figure 30.14.: If there are nested loops in a co-design diagram, they can be rewritten as one loop, by taking the product of the edges.

From Definition 30.1 and Definition 30.5 we know h can be written as

$$\begin{aligned} h : F_1 \times F_2 &\rightarrow \mathcal{A}(R_1 \times R_2) \\ \langle f_1, f_2 \rangle &\mapsto \underset{\leq_{R_1}}{\text{Min}}\{\langle \text{req}_1(i_1), \text{req}_2(i_2) \rangle \mid \\ &\quad (\langle i_1, i_2 \rangle \in I_1 \times I_2) \\ &\quad \wedge (\langle f_1, f_2 \rangle \leq \langle \text{prov}_1(i_1), \text{prov}_2(i_2) \rangle)\}. \end{aligned}$$

All terms factorize in the two components, giving:

$$\begin{aligned} h : \langle f_1, f_2 \rangle &\mapsto \underset{R_1}{\text{Min}}\{\langle \text{req}_1(i_1) \rangle \mid (i \in I_1) \wedge (f_1 \leq \text{prov}_1(i_1))\} \\ &\quad \times \underset{R_2}{\text{Min}}\{\langle \text{req}_2(i_2) \rangle \mid (i \in I_2) \wedge (f_2 \leq \text{prov}_2(i_2))\}, \end{aligned}$$

which reduces to

$$h : \langle f_1, f_2 \rangle \mapsto h_1(f_1) \times h_2(f_2). \quad (30.5)$$

The map h is Scott continuous iff h_1 and h_2 are [19, Lemma II.2.8]. \square

Proposition 30.13. If dp_1 and dp_2 are monotone (??), then also the composition $\text{series}(dp_1, dp_2)$ is monotone.

Proof. From the definition of series (Definition 30.4), the semantics of the interconnection is captured by this problem:

$$h : f_1 \mapsto \begin{cases} \text{using } & r_1, f_2 \in R_1, \quad r_2 \in R_2, \\ \underset{R_2}{\text{Min}} & r_2, \\ \text{s.t.} & r_1 \in h_1(f_1), \\ & r_1 \leq_{R_1} f_2, \\ & r_2 \in h_2(f_2). \end{cases} \quad (30.6)$$

The situation is described by Fig. 30.15. The point f_1 is fixed, and thus $h(f_1)$ is a fixed antichain in R_1 . For each point $r_1 \in h(f_1)$, we can choose a $f_2 \geq r_1$. For each f_2 , the antichain $h_2(f_2)$ traces the solution in R_2 , from which we can choose r_2 .

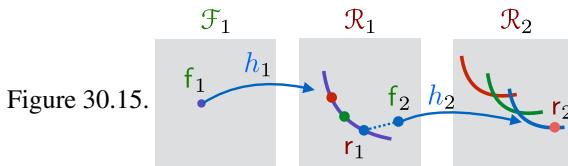


Figure 30.15.

Because h_2 is monotone, $h_2(f_2)$ is minimized when f_2 is minimized, hence we know that the constraint $r_1 \leq f_2$ will be tight. We can then conclude that the objective does not change introducing the constraint $r_1 = f_2$. The problem is reduced to:

$$h : f_1 \mapsto \begin{cases} \text{using } & f_2 \in R_1, \quad r_2 \in R_2, \\ \text{Min}_{\leq_{R_2}} & r_2, \\ \text{s.t.} & f_2 \in h_1(f_1), \\ & r_2 \in h_2(f_2). \end{cases} \quad (30.7)$$

Minimizing r_2 with the only constraint being $r_2 \in h_2(f_2)$, and with $h_2(f_2)$ being an antichain, the solutions are all and only $h_2(f_2)$. Hence the problem is reduced to

$$h : f_1 \mapsto \begin{cases} \text{using } & f_2 \in R_1, \\ \text{Min}_{\leq_{R_2}} & h_2(f_2), \\ \text{s.t.} & f_2 \in h_1(f_1). \end{cases} \quad (30.8)$$

The solution is simply

$$h : f_1 \mapsto \text{Min}_{\leq_{R_2}} \bigcup_{f_2 \in h_1(f_1)} h_2(f_2). \quad (30.9)$$

This map is Scott continuous because it is the composition of Scott continuous maps. \square

Proposition 30.14. If dp is monotone (??), so is $\text{loop}(dp)$.

Proof. The diagram in Fig. 30.7 implies that the map $h_{\text{loop}(dp)}$ can be described as:

$$h_{\text{loop}(dp)} : F_1 \rightarrow \mathcal{A}R, \quad (30.10)$$

$$f_1 \mapsto \begin{cases} \text{using } & r, f_2 \in R, \\ \text{Min}_{\leq_R} & r, \\ \text{s.t.} & r \in h_{dp}(f_1, f_2), \\ & r \leq_R f_2. \end{cases} \quad (30.11)$$

Denote by h_{f_1} the map h_{dp} with the first element fixed:

$$h_{f_1} : f_2 \mapsto h_{dp}(f_1, f_2).$$

Rewrite $r \in h_{dp}(f_1, f_2)$ in Eq. (30.10) as

$$r \in h_{f_1}(f_2). \quad (30.12)$$

Let r be a feasible solution, but not necessarily minimal. Because of Lemma 30.15, the constraint Eq. (30.12) can be rewritten as

$$\{r\} = h_{f_1}(f_2) \cap \uparrow r. \quad (30.13)$$

Because $f_2 \succeq r$, and h_{f_1} is Scott continuous, it follows that $h_{f_1}(f_2) \succeq_{\mathcal{A}R} h_{f_1}(r)$. Therefore, by Lemma 30.16, we have

$$\{r\} \succeq_{\mathcal{A}R} h_{f_1}(r) \cap \uparrow r. \quad (30.14)$$

This is a recursive condition that all feasible r must satisfy.

Let $R \in \mathcal{A}R$ be an antichain of feasible resources, and let r be a generic element of R . Tautologically, rewrite R as the minimal elements of the union of the singletons containing its elements:

$$R = \text{Min}_{\leq_R} \bigcup_{r \in R} \{r\}. \quad (30.15)$$

Substituting Eq. (30.14) in Eq. (30.15) we obtain (cf Lemma 30.17)

$$R \succeq_{\mathcal{A}R} \text{Min}_{\leq_R} \bigcup_{r \in R} h_{f_1}(r) \cap \uparrow r. \quad (30.16)$$

[Converse: It is also true that if an antichain R satisfies Eq. (30.16) then all $r \in R$ are feasible. The constraint Eq. (30.16) means that for any $r_0 \in R$ on the left side, we can find a r_1 in the right side so that $r_0 \succeq_R r_1$. The point r_1 needs to belong to one of the sets of which we take the union; say that it comes from $r_2 \in R$, so that $r_1 \in h_{f_1}(r_2) \cap \uparrow r_2$. Summarizing:

$$\forall r_0 \in R : \exists r_1 : (r_0 \succeq_R r_1) \wedge (\exists r_2 \in R : r_1 \in h_{f_1}(r_2) \cap \uparrow r_2). \quad (30.17)$$

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 $r_1 \succeq_R r_2$, which together with $r_0 \succeq_R r_1$, implies $r_0 \succeq_R r_2$. We have concluded that there exist two points r_0, r_2 in the antichain R such that $r_0 \succeq_R r_2$; therefore, they are the same point: $r_0 = r_2$. Because $r_0 \succeq_R r_1 \succeq_R r_2$, we also conclude that r_1 is the same point as well. We can rewrite Eq. (30.17) by using r_0 in place of r_1 and r_2 to obtain $\forall r_0 \in R : r_0 \in h_{f_1}(r_0)$, which means that r_0 is a feasible resource.]

We have concluded that all antichains of feasible resources R satisfy Eq. (30.16), and conversely, if an antichain R satisfies Eq. (30.16), then it is an antichain of feasible resources.

Equation!Eq. (30.16) is a recursive constraint for R , of the kind

$$\Phi_{f_1}(R) \leq_{\mathcal{A}R} R,$$

with the map Φ_{f_1} defined by

$$\begin{aligned}\Phi_{f_1} : \mathcal{A}\mathbf{R} &\rightarrow \mathcal{A}\mathbf{R}, \\ \mathbf{R} &\mapsto \text{Min}_{\leq_{\mathbf{R}}} \bigcup_{r \in \mathbf{R}} h_{f_1}(r) \cap \uparrow r.\end{aligned}\tag{30.18}$$

If we want the *minimal* resources, we are looking for the *least* antichain:

$$\min_{\leq_{\mathcal{A}\mathbf{R}}} \{ \mathbf{R} \in \mathcal{A}\mathbf{R} : \Phi_{f_1}(\mathbf{R}) \leq_{\mathcal{A}\mathbf{R}} \mathbf{R} \},$$

which is equal to the *least fixed point* of Φ_{f_1} . Therefore, the map $h_{\text{loop(dp)}}$ can be written as

$$h_{\text{loop(dp)}} : f_1 \mapsto \text{fp}(\Phi_{f_1}).\tag{30.19}$$

Lemma 30.18 shows that $\text{fp}(\Phi_{f_1})$ is Scott continuous in f_1 . \square

Lemma 30.15. Let A be an antichain in \mathcal{P} . Then

$$a \in A \quad \equiv \quad \{a\} = A \cap \uparrow a.$$

Lemma 30.16. For $A, B \in \mathcal{AP}$, and $S \subseteq P$, $A \leq_{\mathcal{A}\mathbf{R}} B$ implies $A \cap S \leq_{\mathcal{A}\mathbf{R}} B \cap S$.

Lemma 30.17. For $A, B, C, D \in \mathcal{AP}$, $A \leq_{\mathcal{A}\mathbf{R}} C$ and $B \leq_{\mathcal{A}\mathbf{R}} D$ implies $A \cup B \leq_{\mathcal{A}\mathbf{R}} C \cup D$.

Lemma 30.18. Let $f : \mathcal{P} \times \mathcal{Q} \rightarrow \mathcal{Q}$ be Scott continuous. For each $x \in \mathcal{P}$, define $f_x : y \mapsto f(x, y)$. Then $f^\dagger : x \mapsto \text{fp}(f_x)$ is Scott continuous.

Proof. Davey and Priestly [15] leave this as Exercise 8.26. A proof is found in Gierz et al. [19, Exercise II-2.29]. \square

30.7. Solution of MCDPs

The second main result is that the map h of a MCDP has an explicit expression in terms of the maps h of the subproblems.

Theorem 30.19. The map h for an MCDP has an explicit expression in terms of the maps h of its subproblems, defined recursively using the rules in Table 30.1.

Table 30.1.: Recursive expressions for h

series	$dp = \text{series}(dp_1, dp_2)$	$h = h_1 \odot h_2$
parallel	$dp = \text{par}(dp_1, dp_2)$	$h = h_1 \otimes h_2$
feedback	$dp = \text{loop}(dp_1)$	$h = h_1^\dagger$
co-product	$dp = dp_1 \sqcup dp_2$	$h = h_1 \oslash h_2$

Proof. These expressions were derived in the proofs of Propositions 30.12 and 30.14. The operators $\odot, \otimes, \dagger, \oslash$ are defined in Definitions 30.23 and 46.11. \square

Definition 30.20 (Series operator \odot). For two maps $h_1 : F_1 \rightarrow \mathcal{A}R_1$ and $h_2 : F_2 \rightarrow \mathcal{A}R_2$, if $R_1 = F_2$, define

$$h_1 \odot h_2 : F_1 \rightarrow \mathcal{A}R_2, \\ f_1 \mapsto \text{Min}_{\leq_{R_2}} \bigcup_{s \in h_1(f)} h_2(s).$$

Definition 30.21 (Parallel operator \otimes). For two maps $h_1 : F_1 \rightarrow \mathcal{A}R_1$ and $h_2 : F_2 \rightarrow \mathcal{A}R_2$, define

$$h_1 \otimes h_2 : (F_1 \times F_2) \rightarrow \mathcal{A}(R_1 \times R_2), \\ \langle f_1, f_2 \rangle \mapsto h_1(f_1) \times h_2(f_2), \quad (30.20)$$

where \times is the product of two antichains.

Definition 30.22 (Feedback operator \dagger). For $h : F_1 \times R \rightarrow \mathcal{A}R$, define

$$h^\dagger : F_1 \rightarrow \mathcal{A}R, \\ f_1 \mapsto \text{Ifp}(\Psi_{f_1}^h), \quad (30.21)$$

where $\Psi_{f_1}^h$ is defined as

$$\Psi_{f_1}^h : \mathcal{A}R \rightarrow \mathcal{A}R, \\ R \mapsto \text{Min}_{\leq_R} \bigcup_{r \in R} h(f_1, r) \cap \uparrow r. \quad (30.22)$$

Definition 30.23 (Coproduct operator \oslash). For $h_1, h_2 : F \rightarrow \mathcal{A}R$, define

$$h_1 \oslash h_2 : F \rightarrow \mathcal{A}R, \\ f \mapsto \text{Min}_{\leq_R} (h_1(f) \cup h_2(f)).$$

30.8. Example: Optimizing over the natural numbers

This is the simplest example that can show two interesting properties of MCDPs:

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 $c \in \mathbb{N}$:

$$\begin{cases} \text{Min}_{\leq_{\mathbb{N} \times \mathbb{N}}} & \langle x, y \rangle, \\ \text{s.t.} & x + y \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c. \end{cases} \quad (30.23)$$

One can show that this optimization problem is an MCDP by producing a co-design diagram with an equivalent semantics, such as the one in Fig. 30.16a.

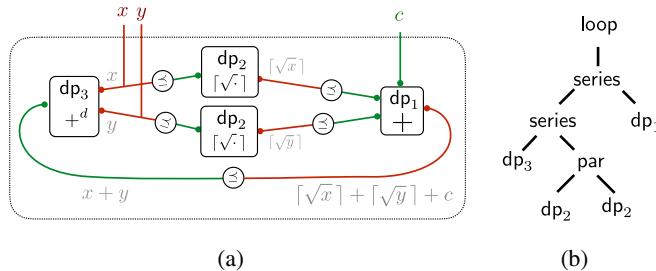


Figure 30.16.: Co-design diagram equivalent to Eq. (30.23) and its tree representation.

The diagram contains three primitive DPs: dp_1 , dp_2 (used twice), and dp_3 . Their h maps are:

$$\begin{aligned}\text{h}_1 : \bar{\mathbb{N}} \times \bar{\mathbb{N}} \times \bar{\mathbb{N}} &\rightarrow \mathcal{A}\bar{\mathbb{N}}, \\ \langle f_1, f_2, f_3 \rangle &\mapsto \{f_1 + f_2 + f_3\}, \\ \text{h}_2 : \bar{\mathbb{N}} &\rightarrow \mathcal{A}\bar{\mathbb{N}}, \\ f &\mapsto \{\lceil \sqrt{f} \rceil\}, \\ \text{h}_3 : \bar{\mathbb{N}} &\rightarrow \mathcal{A}(\bar{\mathbb{N}} \times \bar{\mathbb{N}}), \\ f &\mapsto \{(a, b) \in \bar{\mathbb{N}} \times \bar{\mathbb{N}} : a + b = f\}.\end{aligned}$$

The tree decomposition (Fig. 30.16b) corresponds to the expression

$$\text{dp} = \text{loop}(\text{series}(\text{par}(\text{dp}_2, \text{dp}_2), \text{series}(\text{dp}_1, \text{dp}_3))). \quad (30.24)$$

Consulting Table 30.1, from Eq. (30.24) one obtains an expression for h :

$$\text{h} = ((\text{h}_2 \otimes \text{h}_2) \odot \text{h}_1 \odot \text{h}_3)^\dagger. \quad (30.25)$$

This problem is small enough that we can write down an explicit expression for h . By substituting in Eq. (30.25) the definitions given in Definitions 46.11 and 46.12, we obtain that evaluating $\text{h}(c)$ means finding the least fixed point of a map Ψ_c :

$$\text{h} : c \mapsto \text{fp}(\Psi_c).$$

The map $\Psi_c : \mathcal{A}(\bar{\mathbb{N}} \times \bar{\mathbb{N}}) \rightarrow \mathcal{A}(\bar{\mathbb{N}} \times \bar{\mathbb{N}})$ can be obtained from Eq. (30.22) as follows:

$$\Psi_c : R \mapsto \text{Min} \bigcup_{(x,y) \in R} \uparrow \langle x, y \rangle \cap \quad (30.26)$$

$$\cap \left\{ \langle a, b \rangle \in \mathbb{N}^2 : (a + b \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c) \right\}. \quad (30.27)$$

Kleene's algorithm is the iteration $R_{k+1} = \Psi_c(R_k)$ starting from $R_0 = \perp_{\mathcal{A}(\bar{\mathbb{N}} \times \bar{\mathbb{N}})} = \{\langle 0, 0 \rangle\}$.

For $c = 0$, the sequence converges immediately:

$$R_0 = \{\langle 0, 0 \rangle\} = \text{h}(0).$$

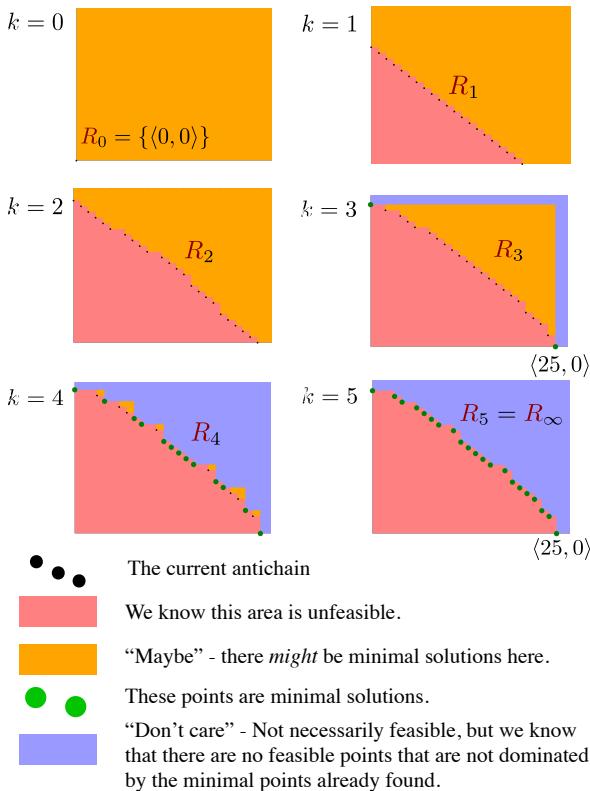


Figure 30.17.: Kleene ascent to solve the problem Eq. (30.23) for $c = 20$. The sequence converges in five steps to $R_5 = R_\infty$.

For $c = 1$, the sequence converges at the second step:

$$\begin{aligned}\mathbf{R}_0 &= \{\langle 0, 0 \rangle\}, \\ \mathbf{R}_1 &= \{\langle \mathbf{0}, \mathbf{1} \rangle, \langle \mathbf{1}, \mathbf{0} \rangle\} = \textcolor{blue}{h}(1).\end{aligned}$$

For $c = 2$, the sequence converges at the fourth step; however, some solutions (in bold) converge sooner:

$$\begin{aligned}\mathbf{R}_0 &= \{\langle 0, 0 \rangle\}, \\ \mathbf{R}_1 &= \{\langle 0, 2 \rangle, \langle 1, 1 \rangle, \langle 2, 0 \rangle\}, \\ \mathbf{R}_2 &= \{\langle \mathbf{0}, \mathbf{4} \rangle, \langle 2, 2 \rangle, \langle \mathbf{4}, \mathbf{0} \rangle\}, \\ \mathbf{R}_3 &= \{\langle \mathbf{0}, \mathbf{4} \rangle, \langle 3, 3 \rangle, \langle 4, 0 \rangle\} = \textcolor{blue}{h}(2).\end{aligned}$$

The next values in the sequence are:

$$\begin{aligned}\textcolor{blue}{h}(3) &= \{\langle \mathbf{0}, \mathbf{6} \rangle, \langle 3, 4 \rangle, \langle 4, 3 \rangle, \langle 6, 0 \rangle\}, \\ \textcolor{blue}{h}(4) &= \{\langle \mathbf{0}, \mathbf{7} \rangle, \langle 3, \mathbf{6} \rangle, \langle 4, 4 \rangle, \langle 6, 3 \rangle, \langle 7, 0 \rangle\}.\end{aligned}$$

Fig. 30.17 shows the sequence for $c = 20$.

30.8.1. Guarantees of Kleene ascent

Solving an MCDP with cycles reduces to computing a Kleene ascent sequence \mathbf{R}_k . At each instant k we have some additional guarantees.

For any finite k , the resources “below” \mathbf{R}_k (the set $\mathbf{R} \setminus \mathbf{R}_k$) are infeasible. (In Fig. 30.17, those are colored in red.)

If the iteration converges to a non-empty antichain \mathbf{R}_∞ , the antichain \mathbf{R}_∞ 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, as in Example 30.3. 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 \mathbf{R}_k can be considered a certificate of infeasibility.

30.9. Complexity of the solution

30.9.1. Complexity of fixed point iteration

Consider first the case of an MCDP that can be described as $\text{dp} = \text{loop}(\text{dp}_0)$, where dp_0 is an MCDP that is described only using the series and par operators. Suppose that dp_0 has resource space \mathbf{R} . Then evaluating h for dp is equivalent to computing a least fixed point iteration on the space of antichains $\mathcal{A}\mathbf{R}$. This allows to give worst-case bounds on the number of iterations.

Proposition 30.24. Suppose that $\text{dp} = \text{loop}(\text{dp}_0)$ and dp_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	$O(\text{height}(\mathcal{A}\mathbf{R}_0))$
total computation	$O(\text{width}(\mathbf{R}_0) \times \text{height}(\mathcal{A}\mathbf{R}_0) \times c)$

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 $\mathcal{A}\mathbf{R}_0$, and we are constructing an ascending chain, so it can take at most $\text{height}(\mathcal{A}\mathbf{R}_0)$ 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 30.25. Consider solving $\text{dp} = \text{loop}(\text{dp}_0)$ with dp_0 defined by $h_0 : \langle \langle \rangle, x \rangle \mapsto x + 1$ 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 T in $\text{height}(\overline{\mathbb{N}}) = \aleph_0$ steps.

Remark 30.26. 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 30.27 (Invariance to re-parameterization). All the results given in this paper are invariant to any order-preserving re-parameterization of all the variables involved.

30.9.2. Relating complexity to the graph properties

Proposition 30.24 above assumes that the MCDP is already in the form $\text{dp} = \text{loop}(\text{dp}_0)$, and relates the complexity to the poset \mathbf{R}_0 . Here we relate the results to the graph structure of an MCDP.

Take an MCDP $\text{dp} = \langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$. To put dp in the form $\text{dp} = \text{loop}(\text{dp}_0)$ according to the procedure in Section 30.5, we need to find an arc feedback set (AFS) of the graph $(\mathcal{V}, \mathcal{E})$. Given a AFS $F \subset \mathcal{E}$, then the resource space \mathbf{R} for a dp_0 such that $\text{dp} = \text{loop}(\text{dp}_0)$ 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 Proposition 30.24.

Of the three performance measures in Proposition 30.24, 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 MCDP.

Definition 30.28. 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

$$\text{DC}(\langle \mathcal{V}, \mathcal{E} \rangle) = \min_{F \text{ is an AFS}} \text{width}\left(\prod_{e \in F} \mathbf{R}_e\right). \quad (30.28)$$

In general, 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.

30.9.3. 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 30.29. 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

$$\begin{aligned} h : \overline{\mathbb{R}}_+ &\rightarrow \mathcal{A}\overline{\mathbb{R}}_+ \times \overline{\mathbb{R}}_+, \\ d &\mapsto \{ \langle T, v \rangle \in \overline{\mathbb{R}}_+ \times \overline{\mathbb{R}}_+ : d = T v \}. \end{aligned}$$

For each value of d , there is a continuum of solutions.

One approach to solving this problem would be to discretize the functionality F and the resources R by sampling and/or coarsening. However, sampling and coarsening makes it hard to maintain completeness and consistency.

One effective approach, outside of the scope of this paper, that allows to use finite computation is to *approximate the design problem itself*, rather than the spaces F, 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 [9].

30.10. 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 MCDPs.

30.10.1. Language and interpreter/solver

MCDPL is a modeling language to describe MCDPs and their compositions. It is inspired by CVX and “disciplined convex programming” [21]. MCDPL is even more disciplined

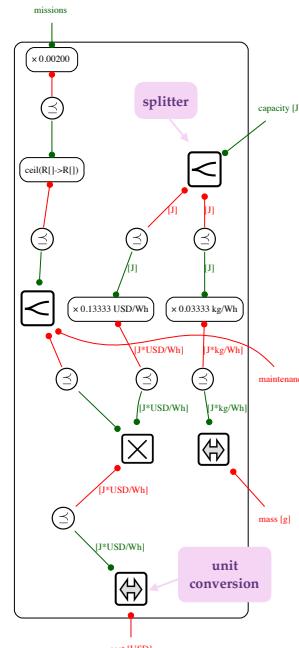


(a) Interface of battery design problem. (b) MCDPL code equivalent to equations Eqs. (30.29) to (30.31).

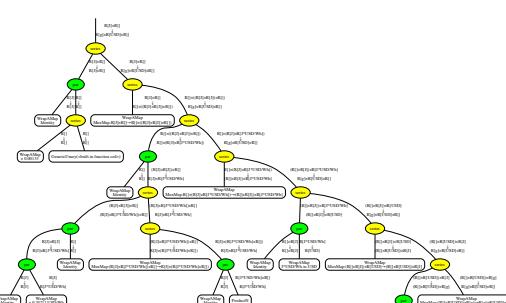
```

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 }

```



(c) Co-design diagram generated by PyMCDP from code in panel (b).



than CVX; for example, multiplying by a negative number is a *syntax* error. The figures are generated by PyMCDP, an interpreter and solver for MCDPs, which implements the techniques described in this paper. The software and a manual are available at <http://mcdp.mit.edu>.

30.10.2. Model of a battery

The choice of a battery can be modeled as a DPI (Fig. 30.18a) 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):

$$\rho \doteq \text{specific energy [Wh/kg]},$$

$$\alpha \doteq \text{specific cost [Wh/$]},$$

$$c \doteq \text{battery lifetime [\# of cycles]}.$$

The relation between functionality and resources is described by three nonlinear monotone constraints:

$$\text{mass} \geq \text{capacity}/\rho, \quad (30.29)$$

$$\text{maintenance} \geq \lceil \text{missions}/c \rceil, \quad (30.30)$$

$$\text{cost} \geq \lceil \text{missions}/c \rceil (\text{capacity}/\alpha). \quad (30.31)$$

Fig. 30.18b shows the MCDPL code that describes the model corresponding to Eqs. (30.29) to (30.31). The diagram in Fig. 30.18c is automatically generated from the code. Fig. 30.18d shows a tree representation of the diagram using the series/par operators.

30.10.3. Competing battery technologies

The parameters for the battery technologies used in this example are shown in Table 30.2.

Table 30.2.: Specifications of common batteries technologies

<i>technology</i>	<i>energy density</i> [Wh/kg]	<i>specific cost</i> [Wh/\$]	<i>operating life</i> # 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

Each row of the table is used to describe a model as in Fig. 30.18b by plugging in the specific values in lines 12–14.

Given the different models, we can define their co-product (Fig. 30.19a) using the MCDPL code in Fig. 30.19b.

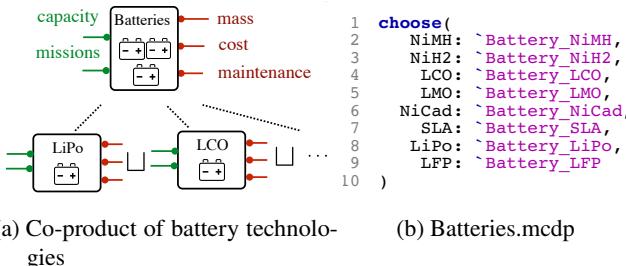


Figure 30.19.: The co-product of design problems describes the choices among different technologies. The MCDPL keyword for the co-product is “choose”.

30.10.4. 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 **<cost, mass, maintenance>**”. 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.

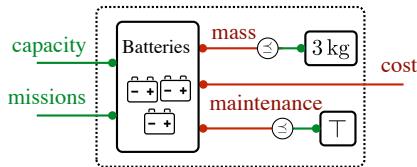
This semantics can be described by the co-design diagram in Fig. 30.20a, which contains two new symbols. The DP labeled “3 kg” implements the semantics of hard constraints. It has one functionality ($F = \overline{\mathbb{R}}_+^{\text{kg}}$) and zero resources ($R = \mathbf{1}$). The poset $\mathbf{1} = \{\langle \rangle\}$ has exactly two antichains: \emptyset and $\{\langle \rangle\}$. These represent “infeasible” and “feasible”, respectively. The DP is described by the map

$$h : \overline{\mathbb{R}}_+^{\text{kg}} \rightarrow \mathbf{A1}, \quad f \mapsto \begin{cases} \{\langle \rangle\}, & \text{if } f \leq 3 \text{ kg,} \\ \emptyset, & \text{if } f > 3 \text{ kg.} \end{cases}$$

$\longrightarrow 3 \text{ kg}$

The block labeled “T” 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. 30.20b. Note the intuitive interface: the user can directly



(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 30.20.: Composition of MCDPs 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 ($x \leq T$), the semantics of the diagram is that the objective is to minimize the **mass** as primary objective.

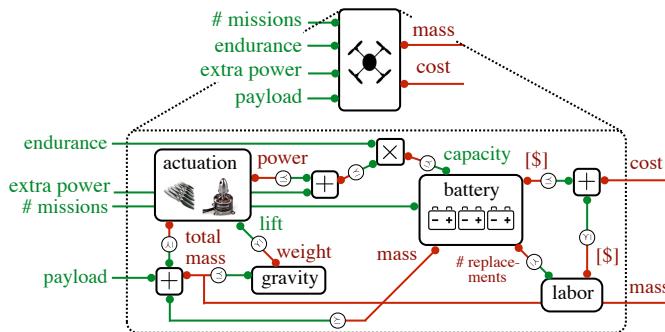
write “mass required by battery $\leq 3 \text{ kg}$ ” and “ignore maintenance required by battery”, which is compiled to “maintenance required by battery $\leq T$ ”.

This relatively simple model for energetics already shows the complexity of MCDPs. Fig. 30.23 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 of the results are intuitive. For example, Fig. 30.23f shows that if the only objective is minimizing **mass**, then the optimal choice is simply the technology with largest specific energy (LiPo). The decision boundaries become complex when considering nonlinear objectives. For example, Fig. 30.23d shows the case where the objective is to minimize the **cost**, which, defined by Eq. (30.31), is nonlinearly related to both **capacity** and **number of missions**. When considering multi-objective problems, such as minimizing jointly $\langle \text{mass}, \text{cost} \rangle$ (Fig. 30.23h) or $\langle \text{mass}, \text{cost}, \text{maintenance} \rangle$ (Fig. 30.23h), there are multiple non-dominated solutions.

30.10.5. 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. 30.21a). 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. 30.21a) are parametrized

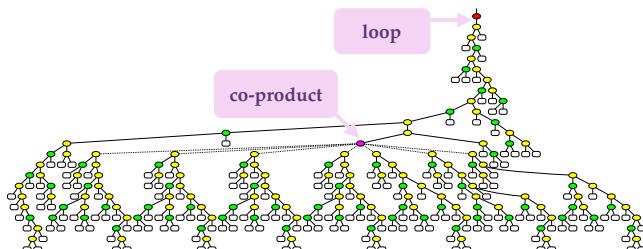


(a) Co-design diagram corresponding to Eqs. (30.32) to (30.33).

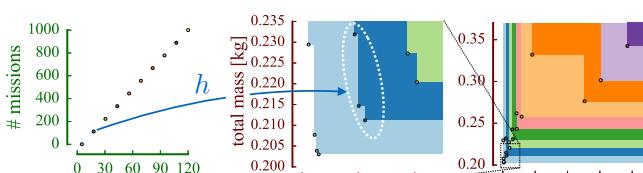
```

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 = 10 $
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. (30.32) to (30.33). The “instance” statements refer to previously defined models for batteries (Fig. 30.19b) and actuation (not shown).



(c) Tree representation for the MCDP. Yellow/green rounded ovals are series/par junctions. There is one co-product junction, signifying the choice between different battery technologies, and one loop junction, at the root of the tree.



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. 30.22). Any other monotone map could be used.



Figure 30.22.

The co-design constraints that combine energetics and actuation are

$$\text{battery capacity} \geq \text{total power} \times \text{endurance}, \quad (30.32)$$

$$\text{total power} = \text{actuation power} + \text{extra power},$$

$$\text{weight} = \text{total mass} \times \text{gravity},$$

$$\text{actuation lift} \geq \text{weight},$$

$$\text{labor cost} = \text{cost per replacement} \times \text{battery maintenance},$$

$$\text{total cost} = \text{battery cost} + \text{actuation cost} + \text{labor cost},$$

$$\text{total mass} = \text{battery mass} + \text{actuation mass} + \text{payload}. \quad (30.33)$$

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. 30.21b; 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 MCDPs to perform a qualitative analysis. Looking at Fig. 30.21a, 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 (Definition 30.28) is equal to $\text{width}(\bar{\mathbb{R}}_+) = 1$. The tree representation is shown in Fig. 30.21c. 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 Eq. (30.24) for the previous example. The least fixed point sequence converges to 64 bits machine precision in 50–100 iterations.

To visualize the multidimensional relation

$$h : \bar{\mathbb{R}}_+ \times \bar{\mathbb{R}}_+^s \times \bar{\mathbb{R}}_+^W \times \bar{\mathbb{R}}_+^g \rightarrow \mathcal{A}(\bar{\mathbb{R}}_+^{kg} \times \bar{\mathbb{R}}_+^{USD}),$$

we need to project across 2D slices. Fig. 30.21d shows the relation when the functionality varies in a chain in the space **endurance/missions**, and Fig. 30.21e shows the results for a chain in the space **endurance/payload**.

Finally, Fig. 30.24 shows the optimal choices of battery technologies in the **endurance/missions** space, when one wants to minimize **mass**, **cost**, or **(mass, cost)**. The decision boundaries are completely different from those in Fig. 30.23. 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 30.23.: This figure shows the optimal decision boundaries for the different battery technologies for the design problem “batteries”, defined as the co-product of all battery technologies (Fig. 30.19). 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 makes some combinations of the functionality infeasible. Note how the decision boundaries are nonconvex, and how the formalisms allows to define slight variations of the problem.

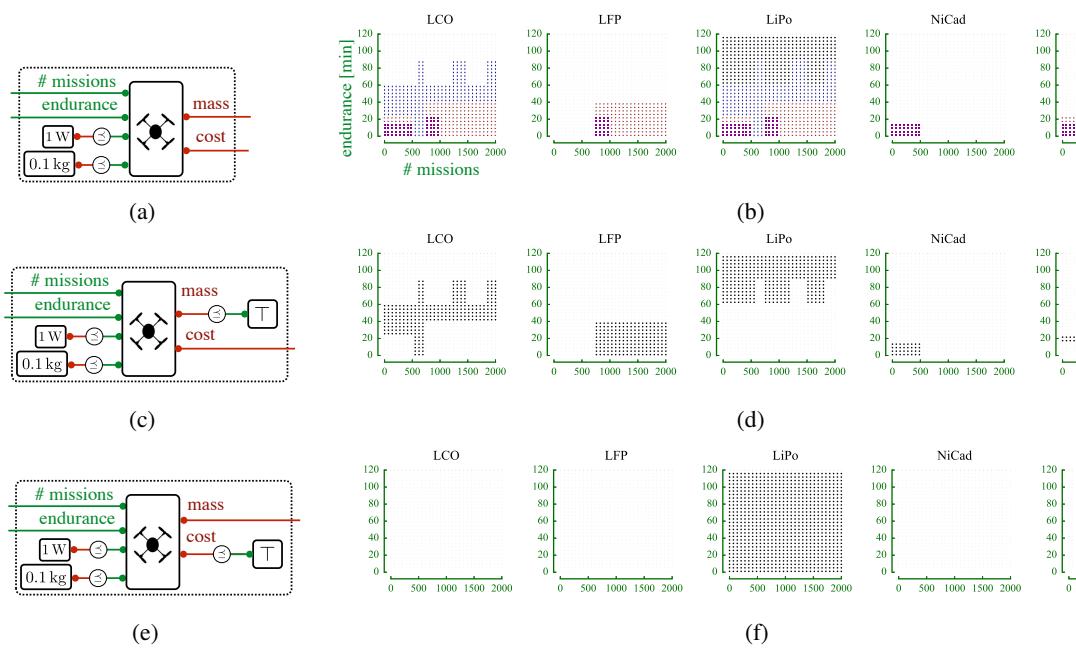


Figure 30.24.: This figure shows the decision boundaries for the different values of battery technologies for the integrated actuation-energetics model described in Fig. 30.21. Please see the caption of Fig. 30.23 for an explanation of the symbols. Notice how in most cases the decision boundaries are different than those in Fig. 30.23: this is an example in which one component cannot be optimized by itself without taking into account the rest of the system.

30.11. Monotonicity and fixed points

We will use Kleene's theorem, a celebrated result that is used in disparate fields. It is used in computer science for defining denotational semantics (see e.g., [33]). It is used in embedded systems for defining the semantics of models of computation (see, e.g., [28]).

Definition 30.30 (Directed set). A set $S \subseteq \mathcal{P}$ is *directed* if each pair of elements in S has an upper bound: for all $a, b \in S$, there exists $c \in S$ such that $a \leq c$ and $b \leq c$.

Definition 30.31 (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.

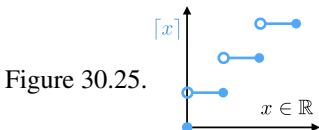
Example 30.32 (Completion of \mathbb{R}_+ to $\overline{\mathbb{R}}_+$). The set of real numbers \mathbb{R} is not a CPO, because it lacks a bottom. The nonnegative reals $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$ have a bottom $\perp = 0$, however, they are not a DCPO because some of their directed subsets do not have an upper bound. For example, take \mathbb{R}_+ , which is a subset of $\overline{\mathbb{R}}_+$. Then \mathbb{R}_+ is directed, because for each $a, b \in \mathbb{R}_+$, there exists $c = \max\{a, b\} \in \mathbb{R}_+$ for which $a \leq c$ and $b \leq c$. One way to make $\langle \mathbb{R}_+, \leq \rangle$ a CPO is by adding an artificial top element \top , by defining $\overline{\mathbb{R}}_+ \triangleq \mathbb{R}_+ \cup \{\top\}$, and extending the partial order \leq so that $a \leq \top$ for all $a \in \mathbb{R}_+$.

A property of maps that will be important are monotonicity and the stronger property of Scott continuity.

Definition 30.33 (Scott continuity). A map $f : \mathcal{P} \rightarrow \mathcal{Q}$ between DCPOs is *Scott continuous* iff for each directed subset $D \subseteq \mathcal{P}$, the image $f(D)$ is directed, and $f(\sup D) = \sup f(D)$.

Remark 30.34. Scott continuity implies monotonicity.

Remark 30.35. Scott continuity does not imply topological continuity. A map from the CPO $\text{tup}\overline{\mathbb{R}}_+, \leq$ to itself is Scott continuous iff it is nondecreasing and left-continuous. For example, the ceiling function $x \mapsto \lceil x \rceil$ is Scott continuous (Fig. 30.25).



A *fixed point* of $f : \mathcal{P} \rightarrow \mathcal{P}$ is a point x such that $f(x) = x$.

Definition 30.36. A *least fixed point* of $f : \mathcal{P} \rightarrow \mathcal{P}$ is the minimum (if it exists) of the set of fixed points of f :

$$\text{lfp}(f) \triangleq \min_{\leq} \{x \in \mathcal{P} : f(x) = x\}. \quad (30.34)$$

The equality in Eq. (30.34) can be relaxed to “ \leq ”.

The least fixed point need not exist. Monotonicity of the map f plus completeness is sufficient to ensure existence.

Lemma 30.37 ([15, CPO Fixpoint Theorem II, 8.22]). If \mathcal{P} is a CPO and $f : \mathcal{P} \rightarrow \mathcal{P}$ is monotone, then $\text{lfp}(f)$ exists.

With the additional assumption of Scott continuity, Kleene's algorithm is a systematic procedure to find the least fixed point.

Lemma 30.38 (Kleene's fixed-point theorem [15, CPO fixpoint theorem I, 8.15]). Assume \mathcal{P} is a CPO, and $f : \mathcal{P} \rightarrow \mathcal{P}$ is Scott continuous. Then the least fixed point of f is the supremum of the Kleene ascent chain

$$\perp \leq f(\perp) \leq f(f(\perp)) \leq \cdots \leq f^{(n)}(\perp) \leq \cdots .$$

31. Uncertainty

to write

Contents

31.1. Monads	246
31.2. Using monads to understand uncertainty	247
31.3. L and U monads	247



Add introduction on uncertainty

Give two different forms of interval orders

Give interpretation as categorical constructs

Definition 31.1 (Twisted arrow category). Given a category \mathbf{C} , we denote its *twisted arrow category* by $\text{Tw}(\mathbf{C})$. This is a category which is composed of:

1. *Objects*: Arrows (morphisms) in \mathbf{C} .
2. *Morphisms*: A morphism between two arrows $f : A \rightarrow B, g : C \rightarrow D$ is given by the pair of arrows $\langle p, q \rangle$ such that the following diagram commutes:

$$\begin{array}{ccc} A & \xleftarrow{p} & C \\ f \downarrow & & \downarrow g \\ B & \xrightarrow{q} & D \end{array} \quad (31.1)$$

Example 31.2 (Intervals). Consider a poset P . The twisted arrow category $\text{Tw}(P)$ is isomorphic to the set of nonempty *intervals* $[a, b] = \{p \in P \mid a \leq_P p \leq_P b\}$. Note that $\text{Tw}(P)$ is a poset as well, ordered by inclusion.

Remark 31.3. Recall Section 14.2 and note that the map which sends a poset (a category) to its twisted arrow category is a functor, which sends objects of the poset

31.1. Monads

Definition 31.4 (Monad). Let \mathbf{C} be a category. A *monad* on \mathbf{C} consists of:

1. A functor $T : \mathbf{C} \rightarrow \mathbf{C}$.
2. A natural transformation $\eta : \text{id}_{\mathbf{C}} \Rightarrow T$ called *unit*.
3. A natural transformation $\mu : TT \Rightarrow T$ called *composition* or *multiplication*.

The constituents must satisfy *left and right unitality*

$$\begin{array}{ccc} T & \xrightarrow{\eta T} & TT & T & \xrightarrow{T\eta} & TT \\ & \searrow \text{id} & \Downarrow \mu & & \searrow \text{id} & \Downarrow \mu \\ & & T & & & T \end{array} \quad (31.2)$$

and *associativity*

$$\begin{array}{ccc} TTT & \xrightarrow{T\mu} & TT \\ \Downarrow \mu T & & \Downarrow \mu \\ TT & \xrightarrow{\mu} & T \end{array} \quad (31.3)$$

31.2. Using monads to understand uncertainty

Take the Unc functor $\text{Unc} : \mathbf{DP} \rightarrow \mathbf{DP}$ which

1. Maps an object P in \mathbf{DP} (poset) to its twisted arrow category $\text{Tw}(P)$, representing a poset interval.
2. Maps a morphism in \mathbf{DP} $d : \mathbf{F} \leftrightarrow \mathbf{R}$ to $\langle \text{Low}d, \text{Upp}d \rangle$, where

$$\begin{aligned}\text{Low}d &: \mathbf{F}_{\text{Low}} \leftrightarrow \mathbf{R}_{\text{Low}}, \\ \text{Upp}d &: \mathbf{F}_{\text{Upp}} \leftrightarrow \mathbf{R}_{\text{Upp}},\end{aligned}\tag{31.4}$$

and $\langle \text{Low}d, \text{Upp}d \rangle$ is a boolean profunctor (i.e., a morphism in \mathbf{DP}) of the form

$$\begin{aligned}\langle \text{Low}d, \text{Upp}d \rangle &: (\mathbf{F}_{\text{Low}} \times \mathbf{F}_{\text{Upp}})^{\text{op}} \times (\mathbf{R}_{\text{Low}} \times \mathbf{R}_{\text{Upp}}) \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \left\langle \langle \underline{f}_{\text{Low}}, \underline{f}_{\text{Upp}} \rangle^*, \langle r_{\text{Low}}, r_{\text{Upp}} \rangle \right\rangle &\mapsto \text{Low}d(\underline{f}_{\text{Low}}^*, r_{\text{Low}}) \wedge \text{Upp}d(\underline{f}_{\text{Upp}}^*, r_{\text{Upp}})\end{aligned}\tag{31.5}$$

write down better

Is this a functor?

Proof. Consider two design problems $f : \mathbf{A} \leftrightarrow \mathbf{B}$ and $g : \mathbf{B} \leftrightarrow \mathbf{C}$.

We know that

$$\begin{aligned}\text{Unc}(f) &: (\mathbf{A}_{\text{Low}} \times \mathbf{A}_{\text{Upp}})^{\text{op}} \times (\mathbf{B}_{\text{Low}} \times \mathbf{B}_{\text{Upp}}) \rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \text{Unc}(g) &: (\mathbf{B}_{\text{Low}} \times \mathbf{B}_{\text{Upp}})^{\text{op}} \times (\mathbf{C}_{\text{Low}} \times \mathbf{C}_{\text{Upp}}) \rightarrow_{\mathbf{Pos}} \mathbf{Bool}.\end{aligned}\tag{31.6}$$

We have

$$\begin{aligned}&(\text{Unc}(f) ; \text{Unc}(g))(\langle \underline{a}, \bar{a} \rangle^*, \langle \underline{c}, \bar{c} \rangle) \\ &= \bigvee_{\langle \underline{b}, \bar{b} \rangle \in \mathbf{B}_{\text{Low}} \times \mathbf{B}_{\text{Upp}}} \text{Unc}(f)(\langle \underline{a}, \bar{a} \rangle^*, \langle \underline{b}, \bar{b} \rangle) \wedge \text{Unc}(g)(\langle \underline{b}, \bar{b} \rangle^*, \langle \underline{c}, \bar{c} \rangle) \\ &= \bigvee_{\langle \underline{b}, \bar{b} \rangle \in \mathbf{B}_{\text{Low}} \times \mathbf{B}_{\text{Upp}}} \text{Low}f(\underline{a}^*, \underline{b}) \wedge \text{Upp}f(\bar{a}^*, \bar{b}) \wedge \text{Low}g(\underline{b}^*, \underline{c}) \wedge \text{Upp}g(\bar{b}^*, \bar{c}) \\ &= (\text{Low}f ; \text{Low}g)(\underline{a}^*, \underline{c}) \wedge (\text{Upp}f ; \text{Upp}g)(\bar{a}^*, \bar{c}) \\ &= \text{Low}f ; g(\underline{a}^*, \underline{c}) \wedge \text{Upp}f ; g(\bar{a}^*, \bar{c}) \\ &= \text{Unc}(f ; g)(\langle \underline{a}, \bar{a} \rangle^*, \langle \underline{c}, \bar{c} \rangle).\end{aligned}\tag{31.7}$$

□

31.3. L and U monads

In this section we propose another example of monads related to posets and design problems. We start by defining the U endofunctor.

Definition 31.5 (U endofunctor). The U endofunctor has the form $U : \mathbf{Pos} \rightarrow \mathbf{Pos}$ and acts on objects and morphisms as follows:

1. *On objects*: Given a poset $P \in \mathbf{Ob}_{\mathbf{Pos}}$, U maps P to its upper set¹.
2. *On morphisms*: Given posets P, Q , and a monotone map $f : P \rightarrow Q$, the U endofunctor acts as:

$$\begin{aligned} U(f) : UP &\rightarrow UQ \\ P' &\mapsto \uparrow \left(\bigcup_{p \in P'} \{f(p)\} \right). \end{aligned} \quad (31.8)$$

We now want to prove that the U endofunctor is an endofunctor, and the proof requires the following two facts.

Lemma 31.6. Given posets P, Q , a monotone map $f : P \rightarrow Q$, and a family of singleton sets $\{S_i\}_{i \in I}$, with $S_i = \{s_i\}$, $s_i \in P$, the following equality holds:

$$\uparrow \underbrace{\left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right)}_{\star} = \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (31.9)$$

Proof. We first want to show that:

$$\underbrace{\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right)}_{\star} \subseteq \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (31.10)$$

Let's take a

$$q \in \uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right). \quad (31.11)$$

If we have such a q , it means that there exists a

$$q' \in \bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \quad (31.12)$$

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:

$$f(s_{i'}) \leq_Q f(p') = q' \leq_Q q. \quad (31.13)$$

We know that $s_{i'} \in \diamond$ and any $q^* \in Q$ satisfying $f(s_{i'}) \leq_Q q^*$ belongs to $\uparrow \diamond$. Therefore, $\star \subseteq \diamond$, which proves the validity of Eq. (31.10).

¹Recall that in Lemma 11.40 we proved that the upper set is itself an object of \mathbf{Pos} .

We now want to show that:

$$\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} s_i} \{f(p)\} \right) \supseteq \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (31.14)$$

By now taking a

$$q \in \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right), \quad (31.15)$$

we know that there is a $i' \in I$ such that $f(s_{i'}) \leq_Q q$. Furthermore, we know that $f(s_{i'}) \in \diamond$. Therefore, any $q^* \succeq_Q f(s_{i'})$ must be in $\uparrow \diamond$, meaning that $q \in \star$, and proving the validity of Eq. (31.14).

The validity of Eq. (31.10) and Eq. (31.14) implies Eq. (31.9). \square

Lemma 31.7. Given posets P, Q and a monotone map $f : P \rightarrow Q$, we have:

$$\uparrow \left(\bigcup_{p' \in \uparrow \{p\}} \{f(p')\} \right) = \uparrow \{f(p)\}. \quad (31.16)$$

Proof. The proof follows from Lemma 31.6, by considering a family of singleton sets consisting solely of the set $\{p\}$. \square

We can now show that the U endofunctor is indeed a functor.

Lemma 31.8. The U endofunctor is indeed a functor.

Proof. U has a valid form and given a poset P , maps id_P to id_{UP} . We now need to show that U fulfills morphism composition. Consider maps $f : P \rightarrow Q$ and $g : Q \rightarrow R$. We have:

$$\begin{aligned} U(f \circ g) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ P' &\mapsto \uparrow \left(\bigcup_{p \in P'} \{g(f(p))\} \right). \end{aligned} \quad (31.17)$$

On the other hand, we have:

$$\begin{aligned} U(f) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ P' &\mapsto \uparrow \left(\bigcup_{p \in P'} \{f(p)\} \right), \end{aligned} \quad (31.18)$$

and

$$U(g) : \mathbf{Pos} \rightarrow \mathbf{Pos} \quad Q' \mapsto \uparrow \left(\bigcup_{q \in Q'} \{g(q)\} \right), \quad (31.19)$$

leading to

$$\begin{aligned} U(f) \circ U(g) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ P' &\mapsto \uparrow \left(\bigcup_{q \in \uparrow \bigcup_{p \in P'} \{f(p)\}} \{g(q)\} \right) \\ &\mapsto \uparrow \left(\bigcup_{p \in P'} \{g(f(p))\} \right). \end{aligned} \quad (\text{Lemma 31.6}) \quad (31.20)$$

Since Eq. (31.17) and Eq. (31.20) are equivalent, U is a functor. \square

Having proven that U is a valid functor, we are now ready to define the U monad.

Definition 31.9 (U monad). The U monad on \mathbf{Pos} consists of:

1. The U endofunctor (Definition 31.5).
2. The unit natural transformation $\eta_U : \text{id}_{\mathbf{Pos}} \Rightarrow U$, which associates to every object $P \in \text{Ob}_{\mathbf{Pos}}$ a morphisms in \mathbf{Pos} given by:

$$\begin{aligned} \eta_U^P : P &\rightarrow UP \\ p &\mapsto \uparrow \{p\}. \end{aligned} \quad (31.21)$$

3. The compositiona natural transformation $\mu_U : U \circ U \Rightarrow U$, which associates to every $P \in \text{Ob}_{\mathbf{Pos}}$ the morphism in \mathbf{Pos} given by:

$$\begin{aligned} \mu_U^P : U(UP) &\rightarrow UP \\ P'' &\mapsto \bigcup_{P' \in P''} P'. \end{aligned} \quad (31.22)$$

Lemma 31.10. The U monad is indeed a monad.

Proof. To show that U is indeed a monad, we need to show the following:

1. η_U is a natural transformation;
2. μ_U is a natural transformation;
3. left unitality holds;
4. right unitality holds;
5. associativity holds;

We prove them in order.

1) η_U is a natural transformation: We need to show that for any $f \in \text{hom}_{\mathbf{Pos}}(P, Q)$, we have:

$$\text{id}_{\mathbf{Pos}}(f) \circ \eta_U^Q = \eta_U^P \circ U(f). \quad (31.23)$$

By expanding the left-hand side, we obtain:

$$\left[\text{id}_{\mathbf{Pos}}(f) \circ \eta_U^Q \right](p) = \uparrow \{f(p)\}. \quad (31.24)$$

By expanding the right-hand side, we get:

$$\begin{aligned}\eta_U^P : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ p &\mapsto \uparrow \{p\}.\end{aligned}\tag{31.25}$$

and

$$\begin{aligned}U(f) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ P' &\mapsto \uparrow \bigcup_{p' \in P'} \{f(p')\},\end{aligned}\tag{31.26}$$

and hence

$$\begin{aligned}[\eta_U^P \circ U(f)](p) &= \uparrow \left(\bigcup_{p' \in \uparrow \{p\}} \{f(p')\} \right) \\ &= \uparrow \{f(p)\}.\end{aligned}\tag{31.27}$$

2) μ_U is a natural transformation:

finish/continue

□

Part D.

Higher-order theory

32. Enrichments

to write

Contents

32.1. Enrichments	256
32.2. Enriched categories	256
32.3. Set-enriched DPs (DPIs)	258



32.1. Enrichments

A design problem “enriched in **Bool**” answers the question, “is it feasible to provide a given functionality f with resources r ?”. A design problem “enriched in **Set**” answers the question, “which implementations provide f with r ?”. A design problem “enriched in **DP**” answers the question, “which design problems provide f with r ?”. If compact closure allows us to zoom out by studying design problems of design problems, then enrichment allows us to zoom in by studying the ‘subatomic’ composition of design problems.

Example 32.1. Beau, the NASA engineer, stares hard at his screen. His boss, Elly May, only wants to know whether the rocket will fly, but he’s the one that has to deal with all potential suppliers and technologies. Since NASA started letting any podunk junkyard company bid on rocket contracts, the number of potential parts had skyrocketed. Intuitively, he wants to keep track of this extra parts information, so he defines, for every design problem $d : F \leftrightarrow R$, an extra set of implementations I_{adp} and two functions $\text{prov} : I_{\text{adp}} \rightarrow F$ and $\text{req} : I_{\text{adp}} \rightarrow R$; we say that i provides at most $f \in F$ and i requires at least $r \in R$. Now the familiar $\text{engine} : \text{Thrust} \leftrightarrow \text{Fuel}$ design problem looks like this:

$$\begin{aligned} \text{engine} : \text{Thrust}^{\text{op}} \times \text{Fuel} &\rightarrow_{\text{Pos}} \mathcal{P}(I_{\text{engine}}) \\ \langle t^*, f \rangle &\mapsto \{i \in I_{\text{engine}} \mid \text{exec}(i) \leq_{\text{Thrust}} t, \text{eval}(i) \geq_{\text{Fuel}} f\} \end{aligned}$$

Beau is feeling pretty proud of himself. Now any time NASA wants a design to be built, he can just press a button and the computer will spit out all the parts needed!

In order to sharpen the intuition of the above example (implementation spaces of the sort above were actually implemented in [7]), we will need to learn more about the theory of enriched categories.

Given that **DP** is compact closed and the external hom is a poset, it is a short, visual step to seeing that **DP** is also what we call a *2-category*: a category where the (external) hom-object between any two objects is itself a category (remember that a poset is a kind of category, from Section 14.2). In other words, a 2-category is a category endowed with the extra structure of “morphisms between morphisms”.

Another way of defining a 2-category is to say that it is “enriched in **Cat**”, the category of categories, whose objects are (small) categories and morphisms are functors. More generally, to say that a category **C** is *enriched in D*, where **D** is a monoidal category, is to say that $\text{Hom}_{\mathbf{C}}(A, B) \in \mathbf{D}$ for all $A, B \in \mathbf{C}$, and that composition of morphisms in $\text{Hom}_{\mathbf{C}}(A, B)$ respects the rules of composition in **D**.

32.2. Enriched categories

Definition 32.2 (Enriched category). We say that a category **C** is enriched in **D** if:

1. For all objects x, y of **C**, the set $\text{Hom}_{\mathbf{C}}(x, y)$ can be considered an object of **D**;
2. **D** is a monoidal category (Definition 25.10), with monoidal product $\otimes_{\mathbf{D}}$;

3. For all objects x, y, z of \mathbf{C} , there exists a certain morphism $m_{x,y,z}$ in \mathbf{D} , which goes from the object $\text{Hom}_{\mathbf{C}}(x, y) \otimes_{\mathbf{D}} \text{Hom}_{\mathbf{C}}(y, z)$ to the object $\text{Hom}_{\mathbf{C}}(x, z)$:

$$m_{x,y,z} : \text{Hom}_{\mathbf{C}}(x, y) \otimes_{\mathbf{D}} \text{Hom}_{\mathbf{C}}(y, z) \rightarrow \text{Hom}_{\mathbf{C}}(x, z). \quad (32.1)$$

The following is obvious, but we record it anyway.

Lemma 32.3. If \mathbf{C} is locally posetal, then so is \mathbf{C}^{op} .

Example 32.4. Every poset (as a category) is enriched in **Bool**, since between any two elements a, b of the poset, either the morphism $a \leq b$ exists ($\text{Hom}_A(a, b) = \top$) or it does not ($\text{Hom}_A(a, b) = \perp$).

Example 32.5. The poset **Bool** is enriched in **Bool**.

Example 32.6. The category **Pos** of posets is enriched in **Pos**, where the partial order on monotone maps is given by $f \Rightarrow g$ (i.e. for $f, g : A \rightarrow B$, $f(a) \leq g(a) \forall a \in A$).

Example 32.7. **Cat** is enriched in **Cat**.

Example 32.8. Every category is enriched in **Set**.

Example 32.9. A poset $\langle P, \leq \rangle$ can be considered a category enriched in the category **Bool**. First, recall the construction that makes each poset into a category (Section 14.2). The poset P as a category is a category with the objects being the elements of P , and with a morphism $f : x \rightarrow y$ existing if and only if $x \leq y$.

Bool as a category contains two elements, \top and \perp , with the three morphism $\perp \rightarrow \top$, $\top \rightarrow \top$, and $\perp \rightarrow \perp$. This is equivalent to say that there is a morphism between $a, b \in \mathbf{Bool}$ if and only if $a \Rightarrow b$. So we can set $\Rightarrow \equiv \rightarrow_{\mathbf{Bool}}$.

Bool can be also considered a monoidal category, by letting \otimes be the *and* operation, so that

$$a \otimes b = a \wedge b. \quad (32.2)$$

Looking at P again, we can show how it can be considered a category enriched in **Bool**. For any two points a, b of P , either $a \leq b$, or not: There are two choices. The hom-set $\text{Hom}(a, b)$ is either non-empty (if $a \leq b$) or empty (if $a \not\leq b$). We can make the correspondence that an empty hom-set corresponds to \perp and a non-empty hom-set corresponds to \top .

Now we can verify that the condition Eq. (32.1) holds. We know that, in a poset, $x \leq y$ and $y \leq z$ implies $x \leq z$. Rewritten in the language of categories, this is:

$$(\text{Hom}(x, y) \text{ non-empty}) \wedge (\text{Hom}(y, z) \text{ non-empty}) \Rightarrow \text{Hom}(x, z) \text{ non-empty}.$$

By making the identification $\Rightarrow \equiv \rightarrow_{\mathbf{Bool}}$ and $\wedge \equiv \otimes_{\mathbf{Bool}}$, we can rewrite the above as

$$(\text{Hom}(x, y) \text{ non-empty}) \otimes_{\mathbf{Bool}} (\text{Hom}(y, z) \text{ non-empty}) \rightarrow_{\mathbf{Bool}} \text{Hom}(x, z) \text{ non-empty},$$

This is the specialization of Eq. (32.1) for \mathbf{C} a poset and \mathbf{D} equal to **Bool**.

Example 32.10. A category “enriched in **Set**” is just a regular category, as we have defined it.

Recall that **Set** is a monoidal category with \otimes_{Set} equal to the Cartesian product \times .

Take an arbitrary category **C**. For all x, y in **C**, we know by definition that $\text{Hom}(x, y)$ is a set.

Consider three objects x, y, z in **C**. We know from the definition of a category that if there exists a morphism $f : x \rightarrow y$ and another $g : y \rightarrow z$, then there also exists $(f ; g) : x \rightarrow z$.

Consider now the set of all morphisms between x, y , given by $\text{Hom}(x, y)$, and all morphisms between y and z , given by $\text{Hom}(y, z)$. What is the relation between those hom-sets and $\text{Hom}(x, z)$?

It is not quite true that $\text{Hom}(x, z)$ is the Cartesian product $\text{Hom}(x, y) \times \text{Hom}(y, z)$. If there are m morphisms between x and y , and n morphisms between y and z , there are not necessarily $m \cdot n$ morphisms from x to z , because we are not guaranteed that all the compositions $(f ; g)$ will be distinct morphisms.

What we are guaranteed is that all of the compositions will be mapped to something in $\text{Hom}(x, z)$; or, in other words, we are guaranteed that there is a map ϕ of the type

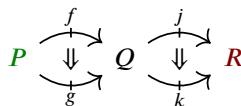
$$\phi : \text{Hom}(x, y) \times \text{Hom}(y, z) \rightarrow \text{Hom}(x, z).$$

This ϕ is a morphism in **Set**, and it is the witness required by Eq. (32.1).

32.3. Set-enriched DPs (DPIs)

Proposition 32.11. **DP** is enriched in **Pos**. Equivalently, it is a locally-posetal 2-category: a 2-category where the hom-categories are also posets.

Proof. Suppose that $f, g : P \leftrightarrow Q$ and $j, k : Q \leftrightarrow R$ are design problems and that $f \Rightarrow g$ and $j \Rightarrow k$. We need to show that $(f ; j) \Rightarrow (g ; k)$ in **DP**.



Assume $f \Rightarrow g$ and $j \Rightarrow k$, and choose $p \in P$ and $r \in R$ such that $(f ; j)(p^*, r) = \top$; we must show that $(g ; k)(p^*, r) = \top$. By assumption, there is some $q \in Q$ such that $f(p^*, q) \wedge j(q^*, r) = \top$. But then also $g(p^*, q) \wedge k(q^*, r) = \top$, and this implies the result. \square

Since posets are also categories, monotone maps between posets can be defined as functors between poset categories (Lemma 14.4). Furthermore, posets are enriched in **Bool** (Example 32.4), i.e. $\text{Hom}_A(a, b) \in \text{Bool}$ for all $A \in \text{Pos}$. So when we say that a design problem f is “enriched in **Bool**”, we mean that is not only a profunctor $f : A \leftrightarrow B$ between posets $A^{\text{op}} \times B$ and **Bool**, but that its action on hom-sets is always a morphism in **Bool**. Explicitly,

$$f : \text{Hom}_{A^{\text{op}} \times B}(\langle a_1^*, b_1 \rangle, \langle a_2^*, b_2 \rangle) \rightarrow \text{Hom}_{\text{Bool}}(f(\langle a_1^*, b_1 \rangle), f(\langle a_2^*, b_2 \rangle))$$

is a morphism in **Bool**—in other words, one among the set $1 \leq_{\text{Bool}} 1, 0 \leq_{\text{Bool}} 0$, and $0 \leq_{\text{Bool}} 1$ familiar from Example 11.5.

Definition 32.12 (Enriched functor). Given two categories **C** and **D** enriched in the same monoidal category **V**, an enriched functor $F : \mathbf{C} \rightarrow \mathbf{D}$ consists of:

1. A map $F : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{D}}$ that maps objects of **C** to objects of **D**.
2. For each x, y in $\text{Ob}_{\mathbf{C}}$, there exists a morphism in **V** given by

$$F_{x,y} : \text{Hom}_{\mathbf{C}}(x, y) \rightarrow \text{Hom}_{\mathbf{D}}(F(x), F(y)),$$

such that composing maps “across F ” respects the composition in **C** and the unit in **V** in the obvious ways:

$$\begin{array}{ccc} \text{Hom}_{\mathbf{C}}(a, b) \otimes \text{Hom}_{\mathbf{C}}(b, c) & \xrightarrow{\circ_{a,b,c}} & \text{Hom}_{\mathbf{C}}(a, c) \\ F_{a,b} \otimes F_{b,c} \downarrow & & \downarrow F_{a,c} \\ \text{Hom}_{\mathbf{D}}(F(b), F(c)) \otimes \text{Hom}_{\mathbf{D}}(F(a), F(b)) & \xrightarrow{\circ_{F(a),F(b),F(c)}} & \text{Hom}_{\mathbf{D}}(F(a), F(c)) \end{array} \quad (32.3)$$

and

$$\begin{array}{ccc} \mathbf{1} & \searrow^{\text{id}_{F(a)}} & \\ \downarrow \text{id}_a & & \\ \text{Hom}_{\mathbf{C}}(a, a) & \xrightarrow{F_{a,a}} & \text{Hom}_{\mathbf{D}}(F(a), F(a)) \end{array}$$

where \otimes and $\mathbf{1}$ are the monoidal product and monoidal unit in **V**.

Proposition 32.13. Design problems in **DP** are Boolean-enriched profunctors.

Proof. We will show that any monotone map $f : A \rightarrow_{\text{Pos}} B$ between posets is a Boolean-enriched functor. Then a design problem is simply a Boolean-enriched functor that happens to be a profunctor.

Monotonicity of f means that $a_1 \leq_A a_2$ implies $f(a_1) \leq_B f(a_2)$. Rewriting this condition considering A, B as categories enriched in **Bool**, we have that

$$\text{Hom}_A(a_1, a_2) \rightarrow \text{Hom}_B(f(a_1), f(a_2)),$$

which can be re-stated as proclaiming the existence of a morphism f_{a_1, a_2} in **Bool**:

$$f_{a_1, a_2} : \text{Hom}_A(a_1, a_2) \rightarrow_{\text{Bool}} \text{Hom}_B(f(a_1), f(a_2)).$$

It remains to be checked that the diagrams in Definition 32.12 commute. The first is trivial since $a \leq_A a$ is true by definition in all posets:

$$\text{id}_a ; f_{a,a}(*) = \text{id}_{f(a)=\top}.$$

The second is also relatively trivial: the monoidal product is just the conjunction \wedge in **Bool**, so commuting with the composition in the poset can be thought of as a verification of the monotonicity of f via the natural \Rightarrow inside **Bool**:

$$\begin{array}{ccc} (a \leq b) \wedge (b \leq c) & \rightarrow & (a \leq c) \\ \Downarrow & & \Downarrow \\ (f(a) \leq f(b)) \wedge (f(b) \leq f(c)) & \rightarrow & (f(a) \leq f(c)) \end{array}$$

□

Having defined morphisms in **DP** as Boolean-enriched design problems, we can directly generalize these Boolean-enriched design problems to **Set**-enriched design problems, or esign problems with implementations.

Definition 32.14 (Design problems with implementation as monotone functions). Suppose that F , R are posets. A *design problem with implementation* is a monotone map (a **Set**-enriched profunctor) $\langle I_d, \text{prov}, \text{req} \rangle : F \nrightarrow R$, where I_d is a set, **prov** and **req** are functions from I_d to F and R , respectively

$$F \xleftarrow{\text{prov}} I \xrightarrow{\text{req}} R,$$

and $\langle I_d, \text{prov}, \text{req} \rangle : F \nrightarrow R$ is given by

$$\begin{aligned} \langle I_d, \text{prov}, \text{req} \rangle : F \nrightarrow R : F^{\text{op}} \times R \rightarrow_{\text{Pos}} \mathcal{P}(I_d) \\ \langle f^*, r \rangle \mapsto \{i \in I_d : (f \leq_F \text{prov}(i)) \wedge (\text{req}(i) \leq_R r)\}, \end{aligned}$$

where the partial order on $\mathcal{P}(I)$ is given by subset inclusion.

Intended semantics When we consider a design problem with implementation $\langle I_d, \text{prov}, \text{req} \rangle : F \nrightarrow R$, we imagine the poset F to represent the functionality to be provided and the poset R to represent the resources required. The object $\langle I_d, \text{prov}, \text{req} \rangle : F \nrightarrow R$ is the set of ways to provide $f \in F$ using $r \in R$.

The monotonicity of d represents the two assumptions:

1. If f is feasible with r in a set i of ways, then any $f' \leq_F f$ is feasible with r in a set $i' \supseteq i$ of ways.
2. If f is feasible with r in a set i of ways, then f is feasible with any $r' \geq_R r$ in a set $i' \supseteq i$ of ways.

Definition 32.15 (Series composition of design problems with implementation). Given two design problems with implementation $\langle I_f, \text{prov}_f, \text{req}_f \rangle : A \nrightarrow B$ and $\langle I_g, \text{prov}_g, \text{req}_g \rangle : B \nrightarrow C$, we can define their series interconnection

$$\langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle : A \nrightarrow C.$$

as follows. With reference to this diagram:

$$A \xleftarrow{\text{prov}_f} I_f \xrightarrow{\text{req}_f} B \xleftarrow{\text{prov}_g} I_g \xrightarrow{\text{req}_g} C \quad (32.4)$$

we let the implementation space be the *pullback*

$$I_{f \circ g} = I_f \times_B I_g \doteq \{(i_f, i_g) \in I_f \times I_g : \text{req}_f(i_f) \leq_B \text{prov}_g(i_g)\}, \quad (32.5)$$

and the two maps prov , req defined as:

$$\begin{aligned}\text{req} &: \langle i_f, i_g \rangle \mapsto \text{req}_2(i_g) \\ \text{prov} &: \langle i_f, i_g \rangle \mapsto \text{prov}_1(i_f).\end{aligned}\tag{32.6}$$

In terms of the profunctors, one has

$$\begin{aligned}\langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle : A \times C \rightarrow_{\text{Pos}} \mathcal{P}(I_f \times_B I_g) \\ \langle a^*, c \rangle \mapsto \bigcup_{\substack{(b, b') \in B^{\text{op}} \times B^{\text{op}} \\ b \leq_B b'}} \left[\langle I_f, \text{prov}_f, \text{req}_f \rangle (a^*, b) \times \langle I_g, \text{prov}_g, \text{req}_g \rangle (b'^*, c) \right].\end{aligned}\tag{32.7}$$

Lemma 32.16. The series composition operation for design problems with implementation as in Eq. (32.7) is monotone in a and c .

Proof. Consider Eq. (32.7). By choosing $a' \geq_A a$, one has

$$\langle I_f, \text{prov}_f, \text{req}_f \rangle (a'^*, c) \subseteq \langle I_f, \text{prov}_f, \text{req}_f \rangle (a^*, c),\tag{32.8}$$

and hence

$$\langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle (a'^*, c) \subseteq \langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle (a^*, c).\tag{32.9}$$

Similarly, by choosing $c' \geq_C c$, one has

$$\langle I_f, \text{prov}_f, \text{req}_f \rangle (a^*, c') \supseteq \langle I_f, \text{prov}_f, \text{req}_f \rangle (a^*, c)\tag{32.10}$$

and hence

$$\langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle (a^*, c') \supseteq \langle I_{f \circ g}, \text{prov}_{f \circ g}, \text{req}_{f \circ g} \rangle (a^*, c).\tag{32.11}$$

This shows monotonicity, and hence shows that the series composition of two DPs is a DP. \square

Lemma 32.17. The series composition operation for design problems with implementation as in Eq. (32.7) is associative, i.e. given three (composable) odesign problems with implementation f, g, h :

$$(f \circ g) \circ h = f \circ (g \circ h).\tag{32.12}$$

Proof. Consider three design problems with implementation:

$$\begin{aligned}f &= \langle I_f, \text{prov}_f, \text{req}_f \rangle : A \leftrightarrow B, \\ g &= \langle I_g, \text{prov}_g, \text{req}_g \rangle : B \leftrightarrow C, \\ h &= \langle I_h, \text{prov}_h, \text{req}_h \rangle : C \leftrightarrow D.\end{aligned}\tag{32.13}$$

First of all, one has:

$$A \xleftarrow{\text{prov}_f} I_f \xrightarrow{\text{req}_f} B \xleftarrow{\text{prov}_g} I_g \xrightarrow{\text{req}_g} C \xleftarrow{\text{prov}_h} I_h \xrightarrow{\text{req}_h} D. \quad (32.14)$$

We first consider the composition $f ; g$. One has:

$$\begin{aligned} I_{f;g} &= \{ \langle i_f, i_g \rangle \in I_f \times I_g : \text{req}_f(i_f) \leq_B \text{prov}_g(i_g) \} \\ \text{req}_{f;g} &: \langle i_f, i_g \rangle \mapsto \text{req}_g(i_g) \\ \text{prov}_{f;g} &: \langle i_f, i_g \rangle \mapsto \text{prov}_f(i_f). \end{aligned} \quad (32.15)$$

We can now look at $(f ; g) ; h$. One has:

$$\begin{aligned} I_{(f;g);h} &= \{ \langle i_{f;g}, i_h \rangle \in I_{f;g} \times I_h : \text{req}_{f;g}(i_{f;g}) \leq_C \text{prov}_h(i_h) \} \\ &= \{ \langle \langle i_f, i_g \rangle, i_h \rangle \in (I_f \times I_g) \times I_h : (\text{req}_f(i_f) \leq_B \text{prov}_g(i_g)) \wedge (\text{req}_g(i_g) \leq_C \text{prov}_h(i_h)) \}, \end{aligned} \quad (32.16)$$

$$\begin{aligned} \text{req}_{(f;g);h} &: \langle i_{f;g}, i_h \rangle \mapsto \text{req}_h(i_h) \\ \text{req}_{(f;g);h} &: \langle \langle i_f, i_g \rangle, i_h \rangle \mapsto \text{req}_h(i_h), \end{aligned} \quad (32.17)$$

and

$$\begin{aligned} \text{prov}_{(f;g);h} &: \langle i_{f;g}, i_h \rangle \mapsto \text{prov}_{f;g}(i_{f;g}) \\ \text{prov}_{(f;g);h} &: \langle \langle i_f, i_g \rangle, i_h \rangle \mapsto \text{prov}_f(i_f). \end{aligned} \quad (32.18)$$

Since

$$(I_f \times I_g) \times I_h \cong I_f \times (I_g \times I_h), \quad (32.19)$$

the above is exactly what we would obtain for $f ; (g ; h)$, so we can say that $f ; (g ; h) \cong (f ; g) ; h$, meaning that this composition is associative up to isomorphism. \square

Definition 32.18 (Identity design problem with implementation). The *identity design problem with implementation* $\langle I_{\text{id}_A}, \text{prov}, \text{req} \rangle : A \leftrightarrow A$ is given by implementation set $I_{\text{id}_A} = A$ and $\text{prov} = \text{req}$ being the identity on A . The profunctor is defined as

$$\langle I_{\text{id}_A}, \text{prov}, \text{req} \rangle : A^{\text{op}} \times A \rightarrow_{\text{Pos}} \mathcal{P}(A) \quad (32.20)$$

$$\langle a^*, a' \rangle \mapsto (\uparrow a) \cap (\downarrow a') \quad (32.21)$$

Remark 32.19. Alternatively, one can define the identity profunctor as

$$\langle I_{\text{id}_A}, \text{prov}, \text{req} \rangle : A^{\text{op}} \times R \rightarrow_{\text{Pos}} \mathcal{P}A \quad (32.22)$$

$$\langle a^*, a' \rangle \mapsto \begin{cases} \{a\}, & a \leq_A a' \\ \emptyset, & \text{otherwise.} \end{cases} \quad (32.23)$$

Lemma 32.20. The series composition operation for design problems with implementation as in Eq. (32.7) satisfies the left and right unit laws (unitality).

Proof.

Do the proof and see if we need to change some definitions to make it work

□

Definition 32.21 (Category of **Set**-enriched design problems). The category of **Set**-enriched design problems, **DPI**, consists of the following data:

1. *Objects*: Objects of **DPI** are posets.
2. *Morphisms*: The morphisms of **DPI** are design problems with implementation (Definition 32.14).
3. *Identity morphism*: The identity morphism is given by Definition 32.18.
4. *Composition operation*: Given two composable morphisms f and g , their composition $f \circ g$ is given by Definition 32.15.

Lemma 32.22. **DPI** is a category.

Proof. We have already shown that the composition operator in **DPI** is associative and unital, and that the composition of two design problems with implementation is a design problem with implementation (closure). Therefore, **DPI** is a valid category. □

Like **DP**, **DPI** is also a traced symmetric monoidal category with monoidal product \times and biproduct given by $+$; we will skip the proofs for **DPI** since most are directly analogous to those for **DP**. We already saw an example of **DPI** in Example 32.1 above. It remains to verify that morphisms in **DPI** are indeed enriched in **Set**:

Proposition 32.23. Design problems with implementation are **Set**-enriched profunctors.

Proof. Fix a design problem $f : A^{\text{op}} \times B \rightarrow_{\text{Pos}} \mathcal{P}(I)$.

Finish the proof

□

We introduce **DPI** mainly as a point of comparison; enriching in **Set** is the most obvious, if not the most elegant, way of representing and reasoning about implementations of design problems. Indeed, the way we defined morphisms is rather clunky (note how I_d is a set rather than a poset, and the extra provisions for **prov** and **req** in the identity morphism), and it also essentially restricts implementations I_d to being subsets of $F^{\text{op}} \times R$, i.e. an implementation *rigidly* provides a fixed functionality f with a fixed r .

The far more natural option, from the perspective of enriched category theory, is to enrich design problems in **DP** itself.

Example 32.24.

$$\begin{aligned} \text{engine} : \text{Thrust}^{\text{op}} \times \text{Fuel} &\rightarrow_{\text{Pos}} \mathcal{P}(\text{Engines}) \\ \langle t^*, f \rangle &\mapsto \{e \in \text{Engines} : e(t^*, f) = \top\} \end{aligned} \tag{32.24}$$

where **Engines** is the hom-poset $\text{Hom}_{\text{DP}}(\text{Thrust}, \text{Fuel})$ from Example 37.7.

Operations on DPIS

33. Negative designs

to write



Part E.

Operadic structures

34. Wiring diagrams

to write



35. Operads

to write



36. Recursion

36.1. Recursive Design Problems

37. Higher-order design

to write

Contents

37.1. Diagrams	276
37.2. Diagram as a monotone function	276
37.3. Representation Results	276
37.4. Compact closed structure	276
37.5. A locally-posetal pro-arrow equipment	280



Remark 37.1. In Lemma 27.3, we showed that $\text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B})$ is a bounded lattice, and in particular a poset. The fact that $\text{Hom}_{\mathbf{DP}}$ is a poset means that design problems in $\text{Hom}_{\mathbf{DP}}(A, B)$ can be counted as functionalities and/or resources in a design problem.

Example 37.2. Jeb's Spaceship Parts lost their last contract to Starshow Bob after submitting a completely inferior engine in every respect ($\text{Jeb-XX} \Rightarrow \text{Bob-Roc}$). In response, they've decided to invest heavily in R&D, which can be thought of as a design problem of its own: Given time and money as resources, what kind of engine technology can they produce as a ‘functionality’?

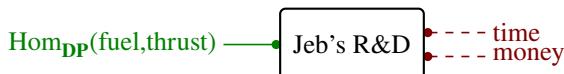


Figure 37.1.: Example of order in **DP**.

37.1. Diagrams

37.2. Diagram as a monotone function

37.3. Representation Results

37.4. Compact closed structure

Definition 37.3 (External and internal hom). The set $\text{Hom}_{\mathbf{C}}(A, B)$ of morphisms between A and B is known as the *external hom*, and is canonically defined for every category \mathbf{C} . For certain categories, however, there is also an *internal hom* $[A, B] \in \mathbf{C}$ which satisfies

$$\text{Hom}_{\mathbf{C}}(A, B) \simeq \{f : \mathbf{1} \rightarrow [A, B]\}, \quad (37.1)$$

where $\mathbf{1}$ is the monoidal unit in \mathbf{C} ; we say that $\text{Hom}_{\mathbf{C}}(A, B)$ is the set of *generalized elements* of $[A, B]$.

Example 37.4. The set of functions between A and B is the internal hom between sets $A, B \in \mathbf{Set}$ (it also happens to be equivalent to the external hom in \mathbf{Set}).

In \mathbf{DP} , the external hom and the internal hom are not equivalent, but the isomorphism $\text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B}) \simeq \{f : \mathbf{1} \leftrightarrow [\mathbf{A}, \mathbf{B}]\}$ still holds, allowing us to study the properties of $\text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B})$ from “inside” \mathbf{DP} .

To say that a category \mathbf{C} is *closed* is to say, roughly, that the internal hom exists for all pairs of objects in \mathbf{C} . In \mathbf{DP} , this means that there exists a unique poset $[\mathbf{A}, \mathbf{B}]$ associated to every pair \mathbf{A}, \mathbf{B} and satisfying Eq. (37.1).

To say that a category \mathbf{C} is *compact closed* is, to say that every object $A \in \mathbf{C}$ has a dual object $A^* \in \mathbf{C}$, and that, for any $B \in \mathbf{C}$, A and A^* satisfy a unique formula for the internal hom: $[A, B] = A^* \times B$. In \mathbf{DP} , the dual of a poset $\mathbf{A} \in \mathbf{DP}$ is \mathbf{A}^{op} , and the formula for the internal hom is $[A, B] = \mathbf{A}^{\text{op}} \times \mathbf{B}$.

Definition 37.5 (Compact closed category). Let $\langle \mathbf{C}, \otimes, I, \sigma \rangle$ be a symmetric monoidal category. It is called *compact closed* if, for all $C \in \mathbf{C}$ there exists some object $C^* \in \mathbf{C}$ (called the *dual of C*), a morphism $\eta_C : I \rightarrow C^* \otimes C$ (called the *unit for C*), and a morphism $\epsilon_C : C \otimes C^* \rightarrow I$ (called the *counit of C*) such that the following diagrams commute for all $C \in \mathbf{C}$:

$$\begin{array}{ccc} C & \xlongequal{\quad} & C \\ \rho^{-1} \downarrow & & \uparrow \lambda \\ C \otimes I & \xrightarrow{\eta} & C \otimes (C^* \otimes C) \xrightarrow{\alpha} (C \otimes C^*) \otimes C \xrightarrow{\epsilon} I \otimes C \end{array} \quad (37.2)$$

$$\begin{array}{ccc} C^* & \xlongequal{\quad} & C^* \\ \lambda^{-1} \downarrow & & \uparrow \rho \\ I \otimes C^* & \xrightarrow{\eta} & (C^* \otimes C) \otimes C^* \xrightarrow{\alpha} C \otimes (C \otimes C^*) \xrightarrow{\epsilon} C^* \otimes I \end{array}$$

Lemma 37.6 (\mathbf{DP} is compact closed). The symmetric monoidal category $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is compact closed.

Proof. Note that we have already shown that $\langle \mathbf{DP}, \otimes, \{1\}, \sigma \rangle$ is a symmetric monoidal category (Lemma 25.19). Define the unit $\eta_P : \{1\} \leftrightarrow P^{\text{op}} \times P$ as

$$\begin{aligned} \eta_P : \{1\} \times (P^{\text{op}} \times P) &\rightarrow_{\text{Pos Bool}} \\ \langle 1, \langle p^*, q \rangle \rangle &\mapsto p \leq_P q, \end{aligned} \quad (37.3)$$

and define the counit $\epsilon_P : P^{\text{op}} \times P \leftrightarrow \{1\}$ as

$$\begin{aligned} \epsilon_P : (P^{\text{op}} \times P)^{\text{op}} \times \{1\} &\rightarrow_{\text{Pos Bool}} \\ \langle \langle p, q^* \rangle^*, 1 \rangle &\mapsto p \leq_P q. \end{aligned} \quad (37.4)$$

We now check that the first diagram (Eq. (37.2)) holds. To show that the second holds, is similar. We will show that the composite $\rho^{-1} ; \eta ; \alpha ; \epsilon ; \lambda$, call it $f : P \leftrightarrow P$, is

equal to id_P . First, let's consider each morphism in this composite morphism f . For ρ^{-1} one has:

$$\rho^{-1}(\langle p_1^*, \langle p_2, * \rangle \rangle) = p_1 \leq p_2. \quad (37.5)$$

For η_P one has

$$\eta(\langle p_2, * \rangle^*, \langle p_3, \langle p_4^*, p_5 \rangle \rangle) = (p_2 \leq p_3) \wedge (p_4 \leq p_5). \quad (37.6)$$

For α one has

$$\alpha(\langle p_3, \langle p_4^*, p_5 \rangle \rangle^*, \langle \langle p_6, p_7^* \rangle, p_8 \rangle) = (p_3 \leq p_6) \wedge (p_7 \leq p_4) \wedge (p_5 \leq p_8). \quad (37.7)$$

For ϵ_P one has

$$\epsilon(\langle \langle p_6, p_7^* \rangle, p_8 \rangle^*, \langle *, p_9 \rangle) = (p_6 \leq p_7) \wedge (p_8 \leq p_9). \quad (37.8)$$

Finally, for λ one has

$$\lambda(\langle *, p_9 \rangle^*, p_{10}) = p_9 \leq p_{10}. \quad (37.9)$$

The composition formula then says that f is given by:

$$\begin{aligned} f(p_1^*, p_{10}) &= \bigvee_{p_2, \dots, p_9} (p_1 \leq p_2 \leq p_3 \leq p_6 \leq p_7 \leq p_4 \leq p_5 \leq p_8 \leq p_9 \leq p_{10}) \\ &= p_1 \leq p_{10} \\ &= \text{id}_P(p_1^*, p_{10}). \end{aligned} \quad (37.10)$$

□

Example 37.7. In the simplest case, fix some $\text{engine} \in \text{Engines} := \text{Hom}_{\mathbf{DP}}(\text{thrust}, \text{fuel})$. Since \mathbf{DP} is compact closed, we can rewrite the R&D design problem $\text{R\&D}(\text{engine}^*, \langle t, m \rangle)$ from Example 37.2 as a design problem of the form $\mathbf{1} \leftrightarrow \text{time} \times \text{money}$ (Fig. 37.2). where

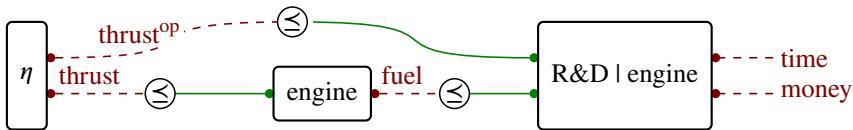


Figure 37.2.: Example for compact closure.

$\eta : \mathbf{I} \leftrightarrow \text{thrust}^{\text{op}} \times \text{thrust}$ is the *unit* design problem described in Eq. (37.3) and the dependent design problem ' $\text{R\&D} \mid \text{engine}$ ' is defined by

$$\begin{aligned} (\text{R\&D} \mid \text{engine}) : (\text{thrust}^{\text{op}} \times \text{fuel})^{\text{op}} \times (\text{time} \times \text{money}) &\rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \langle \langle h^*, f \rangle^*, \langle t, m \rangle \rangle &\mapsto \text{R\&D}(\text{engine}, \langle t, m \rangle). \end{aligned} \quad (37.11)$$

As expected, ' $\text{R\&D} \mid \text{engine}$ ' simply throws away the thrust and fuel information. The composition $(\eta ; \text{engine} ; \text{R\&D} \mid \text{engine})(t, m)$ is defined by

$$\text{R\&D}(\text{engine}, \langle t, m \rangle) \wedge \bigvee_{h^*, f} \text{engine}(h^*, f). \quad (37.12)$$

In other words, the composition represents a simple threshold of time and money (actually, an antichain in the poset Time \times Money), above which it is feasible to construct ‘engine’, and below which it is not.

This design problem can then simply be tensored with the original engine design problem to create a compiled design problem of the form $(\text{engine} \otimes \text{R\&D}) : \text{Thrust} \leftrightarrow \text{Fuel} \times \text{Time} \times \text{Money}$. Since we are simply tensoring, the original engine design problem can still be plugged into a larger design problem in all the usual ways.

Dependent design problems

Example 37.7 was relatively simplistic, since we fixed a specific engine $\in \text{Hom}_{\mathbf{DP}}(\text{thrust}, \text{fuel})$. Now, suppose we want to make the engine design problem—“available technologies”—in a rocket design problem *dependent* on the amount of time and money committed in the R&D design problem, given that time and money could be spent on other things, like building the actual rocket. In other words, we want a design problem involving R&D, which still retains Fuel and Thrust as “open” inputs and outputs. This is a typical kind of design problem one considers when funding science and R&D projects—a very relevant subject for the authors of this article!

Recall that, for a fixed choice of $f \in \text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B})$, any ‘design problem of design problems’ $k : \text{Hom}_{\mathbf{DP}}(\mathbf{A}, \mathbf{B}) \leftrightarrow \mathbf{C}$ can be reframed as a composition of three maps: the unit $\eta_A : \mathbf{1} \leftrightarrow \mathbf{A}^{\text{op}} \times \mathbf{A}$, $f : \mathbf{A} \leftrightarrow \mathbf{B}$, and the dependent design problem $(k \mid f) : \mathbf{A}^{\text{op}} \times \mathbf{B} \leftrightarrow \mathbf{C}$.

I’m stuck on this; not quite sure how to define these kinds of problems. We may not be able to do this in **DP** as defined... maybe it would be easier in **DPI**?

Before going to the next example, we introduce a special design problem for adding the output of two wires.

Example 37.8. Jeb’s R&D team is made up of three rocket scientists: Howie, who has watched every Fast&Furious movie, Shirley, an unpaid intern, and Mabie, who was recently prosecuted for insurance fraud. Howie says that given 3 months and \$100,000, he can make the engine much faster at higher fuel inputs. Shirley says that given 1 month and \$10,000, she can make the engine slightly more fuel-efficient at all currently-feasible thrust performances. Mabie promises that with just 1 week and \$10 million, she can make the engine perform spectacularly at a very low fuel input.

One can go to even higher design problems. In the example above, Jeb might want to know whether it’s worth spending all that time and money to improve the Jeb-XX so that it is competitive with the Bob-Roc, given the value of the contract from NASA. But this requires us to think about R&D processes themselves; i.e. it should cost less time and money to achieve Bob-Roc-grade performance, given the Jeb-XX as a starting point, than it does it to achieve Bob-Roc-grade performance, given the Wright engine as a starting point. That is, we would like to treat the set of engine technologies

$$\text{Engines} = \text{Hom}_{\mathbf{DP}}(\text{thrust}, \text{fuel})$$

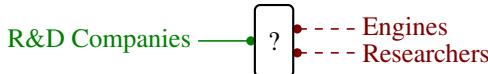
as a resource input to a design problem which outputs R&D processes,

$$\text{R\&D Companies} = \text{Hom}_{\mathbf{DP}}(\text{Engines}, \text{Time} \times \text{Money}).$$

Notably, the design problem



is an element of R&D Companies and thus we can consider, simultaneously, ...



Each R&D process itself outputs an engine technology, so one can imagine tracing the diagram above.

37.5. A locally-posetal pro-arrow equipment

Definition 37.9 (2-category). A *strict 2-category* is a category enriched over **Cat**.

Definition 37.10 (Locally posetal 2-category). A 2-category **C** is *locally posetal* (or enriched in **Pos**) if every hom-category $\text{Hom}_C(x, y)$ is a poset.

In Definition 28.6 on companions and conjoints, we saw how any monotone map in **Pos** can be turned into a design problem. But **DP** is not a subcategory of **Pos**, nor vice versa. Using the 2-category language, we can now precisely characterize the relationship between **Pos** and **DP**.

Example 37.11. Take any category **C**, and enrich it in $\langle \text{Bool}, \wedge, \top \rangle$. The resulting category is composed of:

- Objects are Ob_C ;
- For each $x, y \in \text{Ob}_C$, we have $\text{Hom}_C(x, y) \in \text{Ob}_{\text{Bool}}$. In words, we assign each morphism to either \top or \perp ;
- For any $x, y, z \in \text{Ob}_C$, we define the composition operator

$$\circ : \text{Hom}_C(x, y) \wedge \text{Hom}_C(y, z) \leq \text{Hom}_C(x, z).$$

This composition operator describes transitivity.

- Given $x \in \text{Ob}_C$, we define $\text{id}_x : \top \leq \text{Hom}_C(x, x)$, which implies $x \leq x$ (reflexivity).

In which sense this corresponds to a preorder? If a morphism $f \in \text{Hom}_C(x, y)$ is \top , $x \sim y$, else not. This, together, with transitivity and reflexivity, defines a preorder.

Proposition 37.12. The companion functor $(\hat{\cdot}) : \text{Pos} \rightarrow \text{DP}$ preserves the 2-structure; i.e. if $f \Rightarrow g$ with $f, g : P \rightarrow Q$, then $\hat{f} \geq_{\text{DP}} \hat{g}$, where $\hat{f}, \hat{g} : \text{Q} \leftrightarrow \text{P}$. In fact, it is locally fully faithful (Definition 15.2): $f \Rightarrow g$ iff $\hat{f} \geq_{\text{DP}} \hat{g}$.

Proof. Let f, g be as in the proposition statement. We have the following chain of

equivalences:

$$\begin{aligned} f \Rightarrow g &\text{ iff } \forall p \in P, f(p) \leq g(p) \\ &\text{ iff } \forall p \in P, \forall q \in Q, (q \leq f(p)) \Rightarrow (q \leq g(p)) \\ &\text{ iff } \widehat{f} \geq_{\mathbf{DP}} \widehat{g}. \end{aligned} \quad (37.13)$$

□

Proposition 37.13. For any monotone map $f : P \rightarrow Q$ we have implications

$$\text{id}_P \geq_{\mathbf{DP}} (\check{f} ; \widehat{f}) \quad \text{and} \quad (\widehat{f} ; \check{f}) \geq_{\mathbf{DP}} \text{id}_Q \quad (37.14)$$

Proof. For any $p_1, p_2 \in P$, we have

$$\text{id}_P(\textcolor{green}{p_1}^*, \textcolor{red}{p_2}) = P(\textcolor{green}{p_1}, \textcolor{red}{p_2}) \Rightarrow \bigvee_{q \in Q} Q(f(\textcolor{green}{p}_1), \textcolor{red}{q}) \wedge Q(\textcolor{green}{q}, f(\textcolor{red}{p}_2)) = (\widehat{f} ; \check{f})(\textcolor{green}{p}_1^*, \textcolor{red}{p}_2) \quad (37.15)$$

where the implication arrow comes, e.g. from taking $q = f(p_1)$. For transitivity, we have for any $q_1, q_2 \in Q$:

$$(\check{f} ; \widehat{f})(\textcolor{green}{q}_1^*, \textcolor{red}{q}_2) = \bigvee_{q \in Q} Q(\textcolor{green}{q}_1, f(\textcolor{red}{p})) \wedge Q(f(\textcolor{green}{p}), \textcolor{red}{q}_2) \Rightarrow Q(\textcolor{green}{q}_1, \textcolor{red}{q}_2) = \text{id}_Q(\textcolor{green}{q}_1^*, \textcolor{red}{q}_2). \quad (37.16)$$

□

For the sake of completeness, we add the following theorem, which is in fact a summary of every proposition we have developed since the beginning of the section.

Theorem 37.14. The 2-categories \mathbf{Pos} and \mathbf{DP}^{op} , together with the companion and conjoint functors from Lemma 28.7, form a locally-posetal pro-arrow equipment.

Proof. A locally-posetal pro-arrow equipment consists of the following data:

- A locally posetal 2-category \mathbf{P} (Definition 37.10),
- a locally-posetal 2-category \mathbf{D} , and
- a 2-functor $c : \mathbf{P} \rightarrow \mathbf{D}$, having the properties that
 - c is bijective on objects,
 - c is locally fully faithful, and
 - for every 1-morphism $f : p \rightarrow q$ in \mathbf{P} , there is a morphism $c'(f)$ such that

$$\text{id}_p \Rightarrow c(f) ; c'(f) \quad \text{and} \quad c'(f) ; c(f) \Rightarrow \text{id}_q \quad (37.17)$$

In our situation, the locally-posetal 2-categories are $\mathbf{P} := \mathbf{Pos}$, $\mathbf{D} := \mathbf{DP}$; see ???, Definition 27.1, Proposition 32.11, and Lemma 32.3. The 2-functor is basically the conjoint map; on objects it is the identity, on morphisms it is the conjoint $c(f) := \check{f}$ as in Definition 28.6, and it is 2-functorial and fully faithful by Proposition 37.12. For every f , the final property is satisfied by the companion $c'(f) := \widehat{f}$. □

Can we give a reference for “locally posetal proarrow equipment”? We should do this in many parts of the paper. At the beginning, give references to the usual introductory books. And when we use some non-trivial words, give references to the literature somewhere, even if it is a bit too difficult for the reader of the paper.

AC: Is the 2-category structure preserved by the trace? I suspect it follows from something we already say, but it would be nice to point it out for the slow children.

Part F.

Networks and systems

38. Decorated co-spans

Part G.

Extensions of co-design theory

39. Linear logic

to write



40. Linear DPs

to write



41. Temporal DPs

to write



Part H.

Control theory in category theory

This part is a restatement of control theory results in category theory. This part is developed on the wiki¹ and will be later included in the book.

¹wiki.functorialwiki.org/act4e

Part I.

Case study: co-design of AV fleets

42. Co-design of control systems

43. Co-design of autonomous systems

44. Co-design of mobility systems

Bibliography

- [1] Erik K Antonsson and Jonathan Cagan. *Formal engineering design synthesis*. Cambridge University Press, 2005.
- [2] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Walter de Gruyter GmbH, 2009. DOI: [10.1515/9781400831050](https://doi.org/10.1515/9781400831050).
- [3] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. «Theory and Applications of Robust Optimization.» In: *SIAM Review* 53.3 (2011), pp. 464–501. DOI: [10.1137/080734510](https://doi.org/10.1137/080734510).
- [4] François Bourdoncle. «Efficient chaotic iteration strategies with widenings». In: *Formal Methods in Programming and Their Applications*. Springer Science + Business Media, 2005, pp. 128–141. DOI: [10.1007/bfb0039704](https://doi.org/10.1007/bfb0039704).
- [5] Andrea Censi. «A Class of Co-Design Problems with Cyclic Constraints and Their Solution». In: *Robotics and Automation Letters* (2016).
- [6] Andrea Censi. «A Mathematical Theory of Co-Design». In: *CoRR* abs/1512.08055 (2015). URL: <http://arxiv.org/abs/1512.08055>.
- [7] Andrea Censi. «Efficient Neuromorphic Optomotor Heading Regulation». In: () .
- [8] Andrea Censi. «Efficient neuromorphic optomotor heading regulation». In: *Proceedings of the American Control Conference* 2015-July (2015), pp. 3854–3861. ISSN: 07431619. DOI: [10.1109/ACC.2015.7171931](https://doi.org/10.1109/ACC.2015.7171931).
- [9] Andrea Censi. «Handling Uncertainty in Monotone Co-Design Problems». In: *cs.RO* abs/1609.03103 (2016). URL: <https://arxiv.org/abs/1609.03103>.
- [10] Andrea Censi. «Monotone Co-Design Problems; or, everything is the same». In: *Proceedings of the American Control Conference (ACC)*. 2016.
- [11] Edgar F Codd. «A relational model of data for large shared data banks». In: *Software pioneers*. Springer, 2002, pp. 263–294.
- [12] Agostino Cortesi and Matteo Zanioli. «Widening and narrowing operators for abstract interpretation». In: *Computer Languages, Systems & Structures* 37.1 (2011), pp. 24 –42. ISSN: 1477-8424. DOI: [10.1016/j.cl.2010.09.001](https://doi.org/10.1016/j.cl.2010.09.001).
- [13] Patrick Cousot. *Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice*. Res. rep. R.R. 88. Grenoble, France: Université scientifique et médicale de Grenoble, 1977.
- [14] Patrick Cousot and Radhia Cousot. «Abstract Interpretation: Past, Present and Future». In: *Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. Vienna, Austria, 2014, 2:1–2:10. ISBN: 978-1-4503-2886-9. DOI: [10.1145/2603088.2603165](https://doi.org/10.1145/2603088.2603165).
- [15] 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).

- [16] R.J Duffin. «Topology of series-parallel networks». In: *Journal of Mathematical Analysis and Applications* 10.2 (1965), pp. 303–318. DOI: [10.1016/0022-247x\(65\)90125-3](https://doi.org/10.1016/0022-247x(65)90125-3).
- [17] 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).
- [18] Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press, 2019.
- [19] G. Gierz et al. *Continuous Lattices and Domains*. Cambridge University Press, 2003. DOI: [10.1017/cbo9780511542725](https://doi.org/10.1017/cbo9780511542725).
- [20] 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.
- [21] 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.
- [22] G. Grisetti, C. Stachniss, and W. Burgard. «Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters». In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 34–46. ISSN: 1552-3098. DOI: [10.1109/TRO.2006.889486](https://doi.org/10.1109/TRO.2006.889486).
- [23] Jarda Jezek. *Universal Algebra*. 2008.
- [24] André Joyal, Ross Street, and Dominic Verity. «Traced monoidal categories». In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 119. 3. Cambridge University Press, 1996, pp. 447–468.
- [25] André Joyal, Ross Street, and Dominic Verity. «Traced monoidal categories». In: *Math. Proc. Camb. Phil. Soc.* 119.03 (1996), p. 447. DOI: [10.1017/s0305004100074338](https://doi.org/10.1017/s0305004100074338).
- [26] 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).
- [27] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, May 2006. URL: <http://planning.cs.uiuc.edu/>.
- [28] 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.
- [29] Nancy. Leveson. *Engineering a safer world systems thinking applied to safety*. eng. Engineering systems. Cambridge, Mass: MIT Press.
- [30] 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).
- [31] 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.

- [32] 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>.
- [33] 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).
- [34] Donald B. McIntyre. «The Role of Composition in Computer Programming». In: *SIGAPL APL Quote Quad* 25.4 (June 1995), pp. 116–133. ISSN: 0163-6006. DOI: [10.1145/206944.206985](https://doi.acm.org/10.1145/206944.206985). URL: <http://doi.acm.org/10.1145/206944.206985>.
- [35] 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)*. 2015.
- [36] Jason M. O’Kane and Steven M. LaValle. «On comparing the power of robots». In: *International Journal of Robotics Research* 27.1 (2008), pp. 5–23.
- [37] Gerhard Pahl et al. *Engineering Design: A Systematic Approach*. 3rd. Springer, Jan. 2007. ISBN: 1846283183.
- [38] Douglas Stott Parker Jr. «Partial Order Programming». In: *Principles of Programming Languages* (1989). Also see the extended technical report (CSD-870067) available on the author’s webpage at the url <http://web.cs.ucla.edu/~stott/pop/>, pp. 260–266.
- [39] 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).
- [40] Herbert A. Simon. *The Sciences of the Artificial (3rd Ed.)* Cambridge, MA, USA: MIT Press, 1996. ISBN: 0262691914.
- [41] 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>.
- [42] David I Spivak. «Categorical databases». In: *Presented at Kensho* (2019).
- [43] David I Spivak. *Category theory for the sciences*. MIT Press, 2014.
- [44] David I. Spivak. *Category Theory for the Sciences*. MIT, 2014.
- [45] 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).
- [46] Gheorghe Ștefănescu. *Network Algebra*. Springer Science + Business Media, 2000. DOI: [10.1007/978-1-4471-0479-7](https://doi.org/10.1007/978-1-4471-0479-7).
- [47] N.P. Suh. *Axiomatic Design: Advances and Applications*. Oxford University Press, 2001. ISBN: 9780195134667.
- [48] Maria Svorenova et al. *Resource-Performance Trade-off Analysis for Mobile Robots*. 2016. eprint: [arXiv:1609.04888](https://arxiv.org/abs/1609.04888).

- [49] Olivier de Weck, Daniel Roos, and Christopher Magee. *Engineering Systems: Meeting Human Needs in a Complex Technological World*. Jan. 2011. ISBN: 9780262298513. DOI: [10.7551/mitpress/8799.001.0001](https://doi.org/10.7551/mitpress/8799.001.0001).
- [50] 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

- Bool**, 72
- Cat**, 110
 - monoid*, 178
 - DP**, 172
 - FinSet**, 56
 - Grph**, 132
 - InjSet**, 57
 - Pos**, 90
 - Rel**, 43
 - Set**, 54
 - Vect**, 131
- adjunction, 114, 115
- antitone map, 88
- binary relation, 42
- cartesian product, 122
- category, 20
 - category of design problems, 172
 - Compact closed category, 277
 - coproduct, 128
- design problem, 166
- design problem with implementation, 148, 260
- Drawings, 56
- endofunctor, 94
- endorelation, 46
- Enriched category, 256
- Enriched functor, 259
- equivalence relation, 47
 - function, 44
 - Functor, 94
 - functor, 94
- graph, 37
- Hasse diagram, 71
 - identity design problem, 171
 - initial object, 68
 - Injective function, 57
 - inverse, 62
 - isomorphism, 62
- Monoidal category, 181
 - monotone map, 88
 - morphism, 20
- natural isomorphism, 111
- natural transformation, 110
- opposite category, 107
- partition, 47
- permutations, 64
- poset, 71
- preorder, 73
- product, 124
- profunctor, 176
- relation, 44
- span, 206
- Strong monoidal functor, 197
- subcategory, 56
- Symmetric monoidal category, 184
- terminal object, 68

Part J.

To move

45. Paper to dismember and move

Abstract One of the challenges of modern engineering, and robotics in particular, is designing complex systems, composed of many subsystems, rigorously and with optimality guarantees. This paper introduces a theory of co-design that describes “design problems”, defined as tuples of “functionality space”, “implementation space”, and “resources space”, together with a feasibility relation that relates the three spaces. Design problems can be interconnected together to create “co-design problems”, which describe possibly recursive co-design constraints among subsystems. A co-design problem induces a family of optimization problems of the type “find the minimal resources needed to implement a given functionality”; the solution is an antichain (Pareto front) of resources. A special class of co-design problems are Monotone Co-Design Problems (MCDPs), for which functionality and resources are complete partial orders and the feasibility relation is monotone and Scott continuous. The induced optimization problems are multi-objective, nonconvex, nondifferentiable, noncontinuous, and not even defined on continuous spaces; yet, there exists a complete solution. The antichain of minimal resources can be characterized as a least fixed point, and it can be computed using Kleene’s algorithm. The computation needed to solve a co-design problem can be bounded by a function of a graph property that quantifies the interdependence of the subproblems. These results make us much more optimistic about the problem of designing complex systems in a rigorous way.

45.1. Introduction

One of the great engineering challenge of this century is dealing with the design of “complex” systems. A complex system is complex because its components cannot be decoupled; otherwise, it would be just a (simple) product of simple systems. The *design* of a complex system is complicated because of the “co-design constraints”, which are the constraints that one subsystem induces on another. This paper is an attempt towards formalizing and systematically solving the problem of “co-design” of complex systems with recursive design constraints.

45.1.1. Robotic systems as the prototype of complex systems

Robotics is the prototypical example of a field that includes heterogeneous multi-domain co-design constraints. The design of a robotic system involves the choice of physical components, such as the actuators, the sensors, the power supply, the computing units, the network links, etc. Not less important is the choice of the software components, including perception, planning, and control modules. All these components induce co-design constraints on each other. Each physical component has SWAP characteristics such as its shape (which must be contained somewhere), weight (which adds to the payload), power (which needs to be provided by something else), excess heat (which must be dissipated

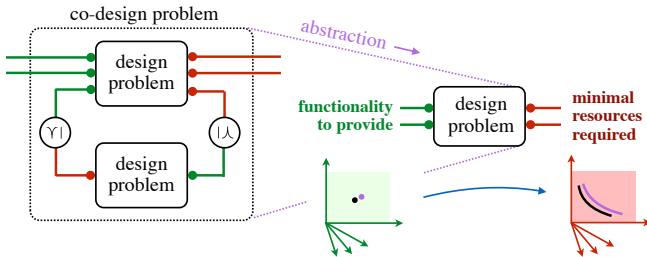


Figure 45.1.: A *design problem* is a relation that relates the implementations available to the **functionality provided** and the **resources required**, both represented as partially ordered sets. A *co-design problem* is the interconnection of two or more design problems. An edge in a co-design diagram like in the figure represent a *co-design constraint*: the resources required by the first design problem are a lower bound for the functionality to be provided by the second. The optimization problem to be solved is: find the solutions that are minimal in resources usage, given a lower bound on the functionality to be provided.

somehow), etc. Analogously, the software components have similar co-design constraints. For example, a planner needs a state estimate. An estimator provides a state estimate, and requires the data from a sensor, which requires the presence of a sensor, which requires power. Everything costs money to buy or develop or license.

What makes system design problems non trivial is that the constraints might be recursive. This is a form of *feedback* in the problem of design (Fig. 45.1). For example, a battery provides power, which is used by actuators to carry the payload. A larger battery provides more power, but it also increases the payload, so more power is needed. Extremely interesting trade-offs arise when considering constraints between the mechanical system and the embodied intelligence. For control, typically a better state estimate saves energy in the execution, but requires better sensors (which increase the cost and the payload) or better computation (which increases the power consumption).

45.1.2. Contribution: A Principled Theory of Co-Design

This paper describes a theory to deal with arbitrarily complex co-design problems in a principled way. A *design problem* is defined as a tuple of functionality space, implementation space, and a resources space, plus the two maps that relate an implementation to functionality provided and resources required. A design problem defines a family of optimization problems of the type “find the minimal resources needed to implement a given functionality”. A *co-design problem* is an interconnection of design problems according to an arbitrary graph structure, including feedback connections. Monotone Co-Design Problems (MCDPs) are the composition of design problems for which both functionality and resources are complete partial orders, and the relation between functionality implemented and resources needed is monotone (order-preserving) and Scott continuous. The first main result in this paper (Theorem 30.11) is that the class of MCDPs is closed with respect to interconnection. The second main result (Theorem 30.19) is that there exists a systematic procedure to solve an MCDP, assuming there is a procedure to solve the

primitive design problems. The solution of an MCDP—a Pareto front, or “antichain” of minimal resources—can be found by solving a least fixed point iteration in the space of antichains. The complexity of this iteration depends on the structure of the co-design diagram.

This paper is a generalization of previous work [5], where the interconnection was limited to one cycle. A conference version of this work appeared as [10].

45.1.3. Outline

?? recalls necessary background about partial orders. Section 22.1 defines co-design problems. Section 30.3 contains a brief statement of results. Section 30.4 describes composition operators for design problems. Section 30.5 shows how any interconnection of design problems can be described using three composition operators (series, parallel, feedback). XXX describes the invariance of a monotonicity property that is preserved by the composition operators. Section 30.7 describes solution algorithms for MCDPs. Section 30.10 shows numerical examples. ?? discusses related work.

Definition 45.1 (Width and height of a poset). $\text{width}(\mathcal{P})$ is the maximum cardinality of an antichain in \mathcal{P} and $\text{height}(\mathcal{P})$ is the maximum cardinality of a chain in \mathcal{P} .

Uppercase “Min” will denote the *minimal* elements of a set. The minimal elements are the elements that are not dominated by any other in the set. Lowercase “min” denotes *the least* element, an element that dominates all others, if it exists. (If $\min S$ exists, then $\text{Min } S = \{\min S\}$.)

The set of minimal elements of a set are an antichain, so Min is a map from the power set $\mathcal{P}(\mathcal{P})$ to the antichains \mathcal{AP} :

$$\begin{aligned}\text{Min} : \mathcal{P}(\mathcal{P}) &\rightarrow \mathcal{AP}, \\ S &\mapsto \{x \in S : (y \in S) \wedge (y \leq x) \Rightarrow (x = y)\}.\end{aligned}$$

Max and max are similarly defined.

45.1.4. Order on antichains

The upper closure operator “ \uparrow ” maps a subset of a poset to an upper set.

Definition 45.2 (Upper closure). The operator \uparrow maps a subset to the smallest upper set that includes it:

$$\begin{aligned}\uparrow : \mathcal{P}(\mathcal{P}) &\rightarrow \mathcal{UP}, \\ S &\mapsto \{y \in \mathcal{P} : \exists x \in S : x \leq y\}.\end{aligned}$$

By using the upper closure operator, we can define an order on antichains using the order on the upper sets (Fig. 45.2).

Lemma 45.3. \mathcal{AP} is a poset with the relation $\leq_{\mathcal{AP}}$ defined by

$$A \leq_{\mathcal{AP}} B \quad \equiv \quad \uparrow A \supseteq \uparrow B.$$

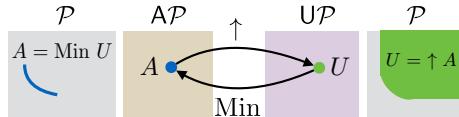


Figure 45.2.

In the poset $\langle \mathcal{AP}, \leq_{\mathcal{AP}} \rangle$, the top is the empty set: $\top_{\mathcal{AP}} = \emptyset$. If a bottom for \mathcal{P} exists, then the bottom for \mathcal{AP} is the singleton containing only the bottom for \mathcal{P} : $\perp_{\mathcal{AP}} = \{\perp_{\mathcal{P}}\}$.

Call “Monotone Co-Design Problems” (MCDPs) the set of CDPIs for which all subproblems respect the conditions in ???. I will show two main results:

- 1) A **modeling result** (Theorem 30.11) says that the class of MCDPs is closed with respect to arbitrary interconnections. Therefore, given a co-design diagram, such as the one in Fig. 22.35, if we know that each design problem is an MCDP, we can conclude that the diagram represents an MCDP as well.
- 2) An **algorithmic result** (Theorem 30.19) says that the functionality-resources map h for the entire MCDP has an explicit expression in terms of the maps $\{h_v, v \in \mathcal{V}\}$ for the subproblems. If there are cycles in the co-design diagram, the map h involves the solution of a *least fixed point* equation in the space of antichains. This equation can be solved using Kleene’s algorithm to find the antichain containing all minimal solutions at the same time.

45.2. Conclusions

This paper described a mathematical theory of co-design, in which the primitive objects are design problems, defined axiomatically as relations between functionality, resources, and implementation. Monotone Co-Design Problems (MCDPs) are the interconnection of design problems whose functionality and resources are complete partial orders and the relation is Scott continuous. These were shown to be non-convex, non-differentiable, and not even defined on continuous spaces. Yet, MCDPs have a systematic solution procedure in the form of a least fixed point iteration, whose complexity depends on a measure of interdependence between design problems—it is easier to design a system composed of subsystems that are only loosely coupled. Based on this theory, it is possible to create modeling languages and optimization tools that allow the user to quickly define and solve multi-objective design problems in heterogeneous domains.

46. Other paper

The work presented here contributes to a compositional theory of “co-design” that allows to optimally design a robotic platform. In this framework, a user models each subsystem as a monotone relation between **functionality provided** and **resources required**. These models can be easily composed to express the co-design constraints between different subsystems. The user then queries the model, to obtain the design with minimal resources usage, subject to a lower bound on the provided functionality. This paper concerns the introduction of uncertainty in the framework. Uncertainty has two roles: first, it allows to deal with limited knowledge in the models; second, it also can be used to generate consistent relaxations of a problem, as the computation requirements can be lowered should the user accept some uncertainty in the answer.

46.1. Introduction

The design of a robotic platform involves the choice and configuration of many hardware and software subsystems (actuation, energetics, perception, control, ...) in an harmonious entity in which all *co-design constraints* are respected. Because robotics is a relatively young discipline, there is still little work towards obtaining systematic procedures to derive optimal designs. Therefore, robot design is a lengthly design process mainly based on empirical evaluation and trial and error. The work presented here contributes to a theory of co-design that allows to optimally design a robotic platform based on formal models of the performance of its subsystems. The goal is to allow a designer to create better designs, faster. This paper describes the introduction of uncertainty in the theory.

Previous work

In previous work [5, 10, 6], I have proposed a compositional theory for co-design. The user defines “design problems” (DPs) that describe the constraints for each subsystem. These DPs can then be hierarchically composed and interconnected to obtain the class of Monotone Co-Design Problems (MCDPs).

An example of MCDP is sketched in Fig. 46.1. The design problem consists in finding an optimal configuration of a UAV, optimizing over actuators, sensors, processors, and batteries. Each design problem (DP) is formalized as a relation between **functionality** and **resources**. For example, the functionality of the UAV is parameterized by three numbers: the **distance to travel** for each mission; the **payload to transport**; the **number of missions** to fly. The optimal design is defined as the one that satisfies the functionality constraints while using the minimal amount of **resources** (**cost** and **mass**).

The convenience of the MCDP framework is that the user can define design problems for each subsystem and then compose them. The definition of the DPs is specified using a

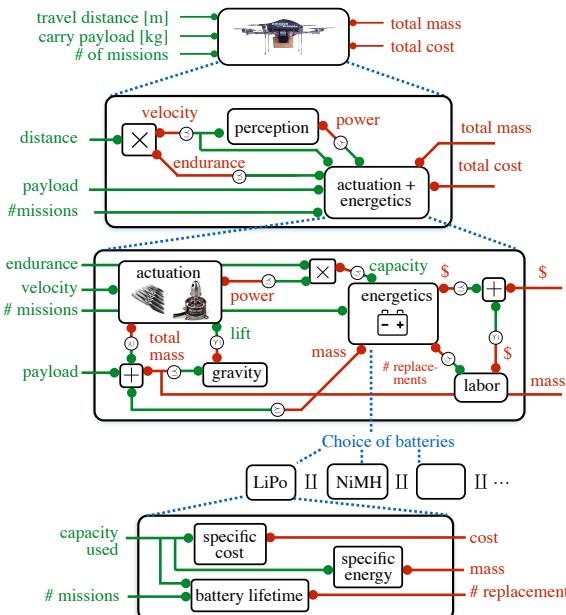


Figure 46.1.: Monotone Co-Design Problems (MCDPs) can capture much of the complexity of the optimal robot design process. The user defines a co-design diagram by hierarchical composition and arbitrary interconnection of primitive “design problems”, modeled as monotone relations between **functionality** and **resources**. The semantics of the MCDP of the figure is the minimization of the **total mass** and **cost** of the platform, subject to functionality constraints (**distance**, **payload**, **number of missions**). This paper describes how to introduce uncertainty in this framework, which allows, for example, to introduce parametric uncertainty in the definition of components properties (e.g. specific cost of batteries).

domain-specific language that promotes composition and code reuse; the formal specification is contained in the supplementary materials. In the figure, the model is exploded to show how actuation and energetics are modeled. Perception is modeled as a relation between the **velocity of the platform** and the **power** required. Actuation is modeled as a relation between **lift** and **power/cost**. Batteries are described by a relation between **capacity** and **mass/cost**. The interconnection between these describe the “co-design constraints”: e.g., actuators must lift the batteries, the batteries must power the actuators. In this example, there are different battery technologies (LiPo, etc.), each specified by specific energy, specific cost, and lifetime, thus characterized by a different relation between **capacity**, **number of missions** and **mass** and **cost**.

Once the model is defined, it can be queried to obtain the *minimal* solution in terms of resources — here, **total cost** and **total mass**. The output to the user is the Pareto front containing all non-dominated solutions. The corresponding optimization problem is, in general, nonconvex. Yet, with few assumptions, it is possible to obtain a systematic solution procedure, and show that there exists a dynamical system whose fixed point corresponding to the set of minimal solutions.

Contribution

This paper describes how to add a notion of *uncertainty* in the MCDP framework. The model of uncertainty considered is interval uncertainty on arbitrary partial orders. For a poset $\langle \mathcal{P}, \leq \rangle$, these are sets of the type $\{x \in \mathcal{P} : a \leq x \leq b\}$. I will show how one can introduce this type of uncertainty in the MCDP framework by considering ordered pairs of design problems. Each pair describes lower and upper bounds for resources usage. These *uncertain design problems* (UDPs) can be composed using series, parallel, and feedback interconnection, just like their non-uncertain counterparts.

The user is then presented with *two* Pareto fronts, corresponding to a lower bound and an upper bound for resource consumption, in the best case and in the worst case, respectively.

This is different from the usual formalization of “robust optimization” (see e.g., [3, 2]), usually formulated as a “worst case” analysis, in which one the uncertainty in the problem is described by a set of possible parameters, and the optimization problem is posed as finding the one design that is valid for all cases.

Uncertainty plays two roles: it can be used as a *modeling tool*, where the relations are uncertain because of our limited knowledge, and it can be used as a *computational tool*, in which we deliberately choose to consider uncertain relations as a relaxation of the problem, to reduce the computational load, while maintaining precise consistency guarantees. With these additions, the MCDP framework allows to describe even richer design problems and to efficiently solve them.

Paper organization

They give a formal definition of design problems (DPs) and their composition, called Monotone Co-Design Problems (MCDPs). Section 46.6 through Section 46.12 describe the notion of Uncertain Design Problem (UDP), the semantics of their interconnection, and the general theoretical results. Section 46.13 describes three specific applications of the theory with numerical results. The supplementary materials include detailed models written

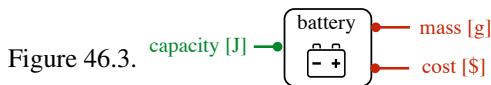
in MCDPL and pointers to obtain the source code and a virtual machine for reproducing the experiments.

46.2. Design Problems

A *design problem* (DP) is a monotone relation between *provided functionality* and *required resources*. *Functionality* and *resources* are complete partial orders (CPO) [15], indicated by $\langle F, \leq_F \rangle$ and $\langle R, \leq_R \rangle$. The graphical representations uses nodes for DPs and green (red) edges for *functionality* and *resources* (Fig. 46.2).



Example 46.1. The first-order characterization of a battery is as a store of energy, in which the *capacity* [kWh] is the *functionality* (what the battery provides) and *mass* [kg] and *cost* [\$] are *resources* (what the battery requires) (Fig. 46.3).



In general, fixed a functionality $f \in F$, there will be multiple resources in R sufficient to perform the functionality that are incomparable with respect to \leq_R . For example, in the case of a battery one might consider different battery technologies that are incomparable in the *mass/cost* resource space (Fig. 46.4).



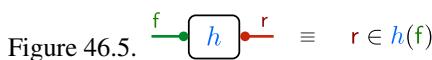
A subset with “minimal”, “incomparable” elements is called “antichain”. This is the mathematical formalization of what is informally called a “Pareto front”.

Definition 46.2. An *antichain* S in a poset $\langle P, \leq \rangle$ is a subset of P such that no element of S dominates another element: if $x, y \in S$ and $x \leq y$, then $x = y$.

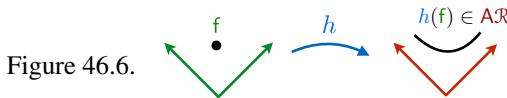
Lemma 46.3. Let \mathcal{AP} be the set of antichains of P . \mathcal{AP} is a poset itself, with the partial order $\leq_{\mathcal{AP}}$ defined as

$$S_1 \leq_{\mathcal{AP}} S_2 \equiv \uparrow S_1 \supseteq \uparrow S_2. \quad (46.1)$$

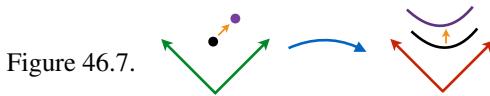
Definition 46.4. A *monotone design problem* (DP) is a tuple $\langle F, R, h \rangle$ such that F and R are CPOs, and $h : f \rightarrow \mathcal{AR}$ is a monotone and Scott-continuous function ([19] or [6, Definition 11]).



Each functionality f corresponds to an antichain of resources $h(f) \in \mathcal{AR}$ (Fig. 46.6).

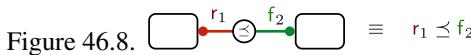


Monotonicity implies that, if the functionality is increased, then the required resources increase as well (Fig. 46.7).

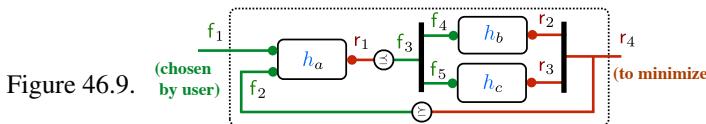


46.3. Monotone Co-Design Problems

A Monotone Co-Design Problem is a multigraph of DPs. Two DPs can be connected by adding an edge (Fig. 46.8). The semantics of the interconnection is that the resources required by the first DP must be provided by the second DP. Mathematically, this is a partial order inequality constraint of the type $r_1 \leq f_2$. Self-loops are allowed as well.



Example 46.5. The MCDP in Fig. 46.9 is the interconnection of 3 DPs h_a , h_b , h_c . The semantics of the MCDP as an optimization problem is shown in Fig. 46.10.



$$\text{Figure 46.10. } f_1 \mapsto \begin{cases} \text{Min } r_4 & r_1 \in h_a(f_1, f_2) \quad r_1 \preceq f_3 \quad f_3 = \langle f_4, f_5 \rangle \\ & r_2 \in h_b(f_4) \quad r_4 \preceq f_2 \quad r_4 = \langle r_2, r_3 \rangle \\ & r_3 \in h_c(f_5) \end{cases}$$

To describe the interconnection, the obvious choice is to describe it as a graph, as a set of nodes and of edges. For our goals, it is more convenient to use an algebraic definition. In the algebraic definition, the graph is represented by a tree, where the leaves are the nodes, and the junctions are one of three operators (series, par, loop), as in Fig. 46.11.

Similar constructions are widespread in computer science. One can see this in the spirit of series-parallel graphs (see, e.g., [16]), with an additional feedback operator to be able to represent all graphs. Equivalently, we are defining a symmetric traced monoidal category (see, e.g., [25] or [44] for an introduction); note that the loop operator is related to the “trace” operator but not exactly equivalent, though they can be defined in terms of each other. An equivalent construction for network processes is given in Stefanescu [46].

Let us use a standard definition of “operators”, “terms”, and “atoms” (see, e.g., [23, p.41]). Given a set of operators ops and a set of atoms \mathcal{A} , let $\text{Terms}(\text{ops}, \mathcal{A})$ be the set of all inductively defined expressions. For example, if the operator set contains only an operator f of arity 1, and there is only one atom a , then the terms are $\text{Terms}(\{f\}, \{a\}) = \{a, f(a), f(f(a)), \dots\}$.

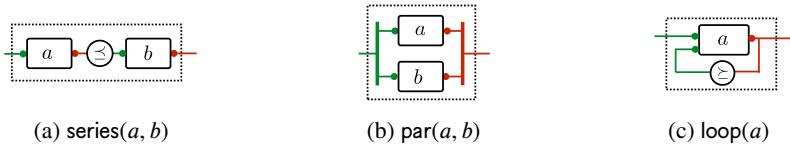


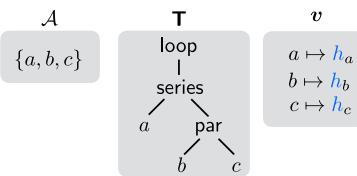
Figure 46.11.: The three operators used in the inductive definition of MCDPs.

Definition 46.6 (Algebraic definition Monotone Co-Design Problems). An MCDP is a tuple $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$, where:

1. \mathcal{A} is any set of atoms, to be used as labels.
 2. The term \mathbf{T} in the {series, par, loop} algebra describes the structure of the graph:
- $$\mathbf{T} \in \text{Terms}(\{\text{series, par, loop}\}, \mathcal{A}).$$
3. The *valuation* \mathbf{v} is a map $\mathbf{v} : \mathcal{A} \rightarrow \mathbf{DP}$ that assigns a DP to each atom.

Example 46.7. The MCDP in Fig. 46.9 can be described by the atoms $\mathcal{A} = \{a, b, c\}$, the term $\mathbf{T} = \text{loop}(\text{series}(a, \text{par}(b, c)))$, plus the valuation $\mathbf{v} : \{a \mapsto h_a, b \mapsto h_b, c \mapsto h_c\}$. The tuple $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$ for this example is shown in Fig. 46.12.

Figure 46.12.



Example 46.8. A sketch of the algebraic representation for part of the example in Fig. 46.1 is shown in Fig. 46.13. The supplementary materials contain more detailed visualizations of the trees for the numerical examples, which take too much space for including in this paper.

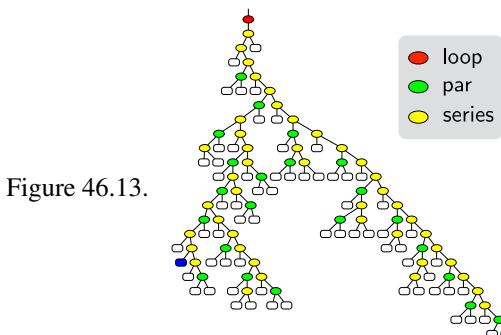


Figure 46.13.

46.4. Semantics of MCDPs

We can now define the *semantics* of an MCDP. The *semantics* is a function φ that, given an algebraic definition of an MCDP, returns a \mathbf{DP} . Thanks to the algebraic definition, to

define φ , we need to only define what happens in the base case (equation 46.2), and what happens for each operator `series`, `par`, `loop` (Eqs. (46.3) to (46.5)).

Definition 46.9 (Semantics of MCDP). Given an MCDP in algebraic form $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$, the semantics

$$\varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle] \in \mathbf{DP}$$

is defined as follows:

$$\varphi[\langle \mathcal{A}, a, \mathbf{v} \rangle] \doteq \mathbf{v}(a), \quad \text{for all } a \in \mathcal{A}, \quad (46.2)$$

$$\varphi[\langle \mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] \doteq \varphi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle] \odot \varphi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle], \quad (46.3)$$

$$\varphi[\langle \mathcal{A}, \text{par}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] \doteq \varphi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle] \otimes \varphi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle], \quad (46.4)$$

$$\varphi[\langle \mathcal{A}, \text{loop}(\mathbf{T}), \mathbf{v} \rangle] \doteq \varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle]^\dagger. \quad (46.5)$$

The operators \odot, \otimes, \dagger are defined in Definitions 46.11 to 46.12. Please see [6, Section VI] for details about the interpretation of these operators and how they are derived.

The \otimes operator is a regular product in category theory: we are considering all possible combinations of resources required by \mathbf{h}_1 and \mathbf{h}_2 .

Definition 46.10 (Product operator \otimes). For two maps $\mathbf{h}_1 : \mathbf{F}_1 \rightarrow \mathcal{A}\mathbf{R}_1$ and $\mathbf{h}_2 : \mathbf{F}_2 \rightarrow \mathcal{A}\mathbf{R}_2$, define

$$\begin{aligned} \mathbf{h}_1 \otimes \mathbf{h}_2 : (\mathbf{F}_1 \times \mathbf{F}_2) &\rightarrow \mathcal{A}(\mathbf{R}_1 \times \mathbf{R}_2), \\ \langle \mathbf{f}_1, \mathbf{f}_2 \rangle &\mapsto \mathbf{h}_1(\mathbf{f}_1) \times \mathbf{h}_2(\mathbf{f}_2), \end{aligned}$$

where \times is the product of two antichains.

The \odot operator is similar to a convolution: fixed \mathbf{f}_1 , one evaluates the resources $\mathbf{r}_1 \in \mathbf{h}_1(\mathbf{f})$, and for each \mathbf{r}_1 , $\mathbf{h}_2(\mathbf{r}_1)$ is evaluated. The Min operator then chooses the minimal elements.

Definition 46.11 (Series operator \odot). For two maps $\mathbf{h}_1 : \mathbf{F}_1 \rightarrow \mathcal{A}\mathbf{R}_1$ and $\mathbf{h}_2 : \mathbf{F}_2 \rightarrow \mathcal{A}\mathbf{R}_2$, if $\mathbf{R}_1 = \mathbf{F}_2$, define

$$\begin{aligned} \mathbf{h}_1 \odot \mathbf{h}_2 : \mathbf{F}_1 &\rightarrow \mathcal{A}\mathbf{R}_2, \\ \mathbf{h}_1 &\mapsto \text{Min}_{\leq_{\mathbf{R}_2}} \bigcup_{\mathbf{r}_1 \in \mathbf{h}_1(\mathbf{f})} \mathbf{h}_2(\mathbf{r}_1). \end{aligned}$$

The dagger operator \dagger is actually a standard operator used in domain theory (see, e.g., [19, pp. II–2.29]).

Definition 46.12 (Loop operator \dagger). For a map $\mathbf{h} : \mathbf{F}_1 \times \mathbf{F}_2 \rightarrow \mathcal{A}\mathbf{R}$, define

$$\begin{aligned} \mathbf{h}^\dagger : \mathbf{F}_1 &\rightarrow \mathcal{A}\mathbf{R}, \\ \mathbf{f}_1 &\mapsto \text{lfp} \left(\Psi_{\mathbf{f}_1}^{\mathbf{h}} \right), \end{aligned} \quad (46.6)$$

where lfp is the least-fixed point operator, and $\Psi_{\mathbf{f}_1}^{\mathbf{h}}$ is defined as

$$\begin{aligned} \Psi_{\mathbf{f}_1}^{\mathbf{h}} : \mathcal{A}\mathbf{R} &\rightarrow \mathcal{A}\mathbf{R}, \\ \mathbf{R} &\mapsto \text{Min}_{\leq_{\mathbf{R}}} \bigcup_{\mathbf{r} \in \mathbf{R}} \mathbf{h}(\mathbf{f}_1, \mathbf{r}) \cap \uparrow \mathbf{r}. \end{aligned}$$

46.5. Solution of MCDPs

Definition 46.9 gives a way to evaluate the map \mathbf{h} for the graph, given the maps $\{\mathbf{h}_a \mid a \in \mathcal{A}\}$ for the leaves. Following those instructions, we can compute $\mathbf{h}(\mathbf{f})$, and thus find the minimal resources needed for the entire MCDP.

Example 46.13. The MCDP in Fig. 46.9 is so small that we can do this explicitly. From Definition 46.9, we can compute the semantics as follows:

$$\begin{aligned}\mathbf{h} &= \varphi \llbracket (\mathcal{A}, \text{loop}(\text{series}(a, \text{par}(b, c)), \mathbf{v})) \rrbracket \\ &= (\mathbf{h}_a \odot (\mathbf{h}_b \otimes \mathbf{h}_c))^\dagger.\end{aligned}$$

Substituting the definitions Definitions 46.10 to 46.12 above, one finds that $\mathbf{h}(\mathbf{f}) = \text{lfp}(\Psi_{\mathbf{f}})$, with

$$\begin{aligned}\Psi_{\mathbf{f}} : \mathcal{A}\mathbf{R} &\rightarrow \mathcal{A}\mathbf{R}, \\ \mathbf{R} &\mapsto \bigcup_{r \in \mathbf{R}} \left[\text{Min}_{\leq} \uparrow \bigcup_{s \in \mathbf{h}_a(\mathbf{f}_1, r)} \mathbf{h}_b(s) \times \mathbf{h}_c(s) \right] \cap \uparrow r.\end{aligned}$$

The least fixed point equation can be solved using Kleene's algorithm [15, CPO Fixpoint theorem I, 8.15]. A dynamical system that computes the set of solutions is given by

$$\begin{cases} \mathbf{R}_0 &\leftarrow \{\perp_{\mathbf{R}}\}, \\ \mathbf{R}_{k+1} &\leftarrow \Psi_{\mathbf{f}}(\mathbf{R}_k). \end{cases}$$

The limit sup \mathbf{R}_k is the set of minimal solutions, which might be an empty set if the problem is unfeasible.

This dynamical system is a proper algorithm only if each step can be performed with bounded computation. An example in which this is not the case are relations that give an infinite number of solutions for each functionality. For example, the very first DP appearing in Fig. 46.1 corresponds to the relation `travel distance` \leq `velocity` \times `endurance`, for which there are infinite numbers of pairs $\langle \text{velocity}, \text{endurance} \rangle$ for each value of `travel distance`. The machinery developed in this paper will make it possible to deal with these infinite-cardinality relations by relaxation.

46.6. Uncertain Design Problems

We now consider the introduction of uncertainty. This section describes objects called Uncertain DPs (UDPs), which are an ordered pair of DPs. Each pair can be interpreted as upper and lower bounds for resource consumptions (Fig. 46.14).

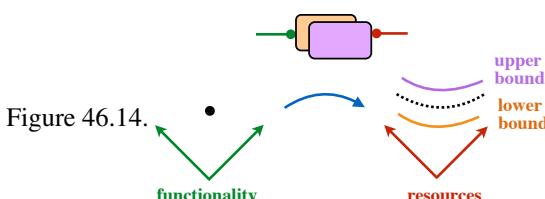


Figure 46.14.

We will be able to propagate this interval uncertainty through an arbitrary interconnection of DPs. The result presented to the user will be a *pair* of antichains — a lower and an upper bound for the resource consumption.

46.7. Partial order \leq_{DP}

Being able to provide both upper and lower bounds comes from the fact that in this framework everything is ordered—there are a poset of resources, lifted to posets of antichains, which is lifted to posets of DPs, and finally, to the poset of uncertain DPs.

The first step is defining a partial order \leq_{DP} on DP .

Definition 46.14 (Partial order \leq_{DP}). Consider two DPs $h_1, h_2 : \mathcal{F} \rightarrow \mathcal{AR}$. The DP h_1 precedes h_2 if it requires fewer resources for all functionality f :

$$h_1 \leq_{\text{DP}} h_2 \quad \equiv \quad h_1(f) \leq_{\mathcal{AR}} h_2(f), \text{ for all } f \in \mathcal{F}.$$

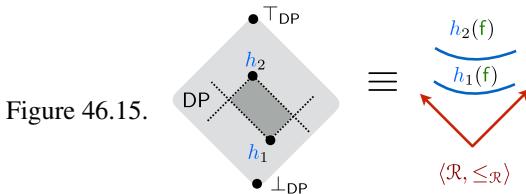


Figure 46.15.

In this partial order, there is both a top T_{DP} and a bottom \perp_{DP} , defined as follows:

$$\begin{aligned} \perp_{\text{DP}} : \mathcal{F} &\rightarrow \mathcal{AR}, & T_{\text{DP}} : \mathcal{F} &\rightarrow \mathcal{AR}, \\ f &\mapsto \{\perp_{\mathcal{R}}\}. & f &\mapsto \emptyset. \end{aligned} \quad (46.7)$$

\perp_{DP} means that any functionality can be done with zero resources, and T_{DP} means that the problem is always infeasible (“the set of feasible resources is empty”).

46.8. Uncertain DPs (UDPs)

Definition 46.15 (Uncertain DPs). An Uncertain DP (UDP) u is a pair of DPs $(\mathbf{L}u, \mathbf{U}u)$ such that $\mathbf{L}u \leq_{\text{DP}} \mathbf{U}u$.

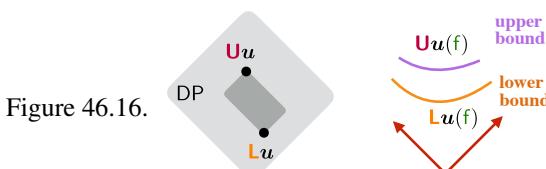


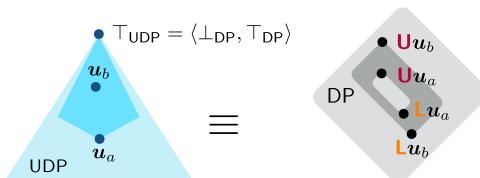
Figure 46.16.

46.9. Order on UDP

Definition 46.16 (Partial order \leq_{DP}). A UDP u_a precedes another UDP u_b if the interval $[\mathbf{L}u_a, \mathbf{U}u_a]$ is contained in the interval $[\mathbf{L}u_a, \mathbf{U}u_a]$ (Fig. 46.17):

$$u_a \leq_{\text{DP}} u_b \equiv \mathbf{L}u_b \leq_{\text{DP}} \mathbf{L}u_a \leq_{\text{DP}} \mathbf{U}u_a \leq_{\text{DP}} \mathbf{U}u_b.$$

Figure 46.17.



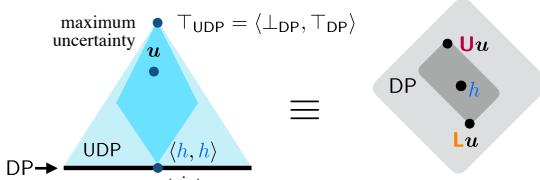
The partial order \leq_{DP} has a top $T_{\text{DP}} = \langle \perp_{\text{DP}}, T_{\text{DP}} \rangle$. This pair describes maximum uncertainty about the DP: we do not know if the DP is feasible with 0 resources (\perp_{DP}), or if it is completely infeasible (T_{DP}).

46.10. DPs as degenerate UDPs

A DP h is equivalent to a degenerate UDP $\langle h, h \rangle$.

A UDP u is a bound for a DP h if $u \leq_{\text{DP}} \langle h, h \rangle$, or, equivalently, if $\mathbf{L}u \leq_{\text{DP}} h \leq_{\text{DP}} \mathbf{U}u$.

Figure



A pair $\langle h, h \rangle$ is a minimal element of DP , because it cannot be dominated by any other. Thus, we can imagine the space DP as a pyramid (Fig. 46.18), with the space DP forming the base. The base represents non-uncertain DPs. The top of the pyramid is T_{DP} , which represents maximum uncertainty.

46.11. Interconnection of Uncertain Design Problems

We now define the interconnection of UDPs, in an equivalent way to the definition of MCDPs. The only difference between Definition 46.6 and Definition 46.17 below is that the valuation assigns to each atom an UDP, rather than a DP.

Definition 46.17 (Algebraic definition of UMCDPs). An Uncertain MCDP (UMCDP) is a tuple $\langle \mathcal{A}, \mathbf{T}, \nu \rangle$, where \mathcal{A} is a set of atoms, $\mathbf{T} \in \text{Terms}(\{\text{series}, \text{par}, \text{loop}\}, \mathcal{A})$ is the algebraic representation of the graph, and $\nu : \mathcal{A} \rightarrow \text{DP}$ is a valuation that assigns to each atom a UDP.

Next, the semantics of a UMCMDP is defined as a map Φ that computes the UDP. Definition 46.18 below is analogous to Definition 46.9.

Definition 46.18 (Semantics of UMCMDPs). Given an UMCMDP $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$, the semantics function Φ computes a UDP

$$\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle] \in \mathbf{DP},$$

and it is recursively defined as follows:

$$\Phi[\langle \mathcal{A}, a, \mathbf{v} \rangle] = \mathbf{v}(a), \quad \text{for all } a \in \mathcal{A}.$$

$$\begin{aligned}\mathbf{L}\Phi[\langle \mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] &= (\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle]) \odot (\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle]), \\ \mathbf{U}\Phi[\langle \mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] &= (\mathbf{U}\Phi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle]) \odot (\mathbf{U}\Phi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle]),\end{aligned}$$

$$\begin{aligned}\mathbf{L}\Phi[\langle \mathcal{A}, \text{par}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] &= (\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle]) \otimes (\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle]), \\ \mathbf{U}\Phi[\langle \mathcal{A}, \text{par}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v} \rangle] &= (\mathbf{U}\Phi[\langle \mathcal{A}, \mathbf{T}_1, \mathbf{v} \rangle]) \otimes (\mathbf{U}\Phi[\langle \mathcal{A}, \mathbf{T}_2, \mathbf{v} \rangle]),\end{aligned}$$

$$\begin{aligned}\mathbf{L}\Phi[\langle \mathcal{A}, \text{loop}(\mathbf{T}), \mathbf{v} \rangle] &= (\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle])^\dagger, \\ \mathbf{U}\Phi[\langle \mathcal{A}, \text{loop}(\mathbf{T}), \mathbf{v} \rangle] &= (\mathbf{U}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle])^\dagger.\end{aligned}$$

The operators \dagger, \odot, \otimes are defined in Definitions 46.11 to 46.12.

46.12. Approximation results

The main result of this section is a relaxation result stated as Theorem 46.20 below.

Informal statement

Suppose that we have an MCDP composed of many DPs, and one of those is h_a (Fig. 46.19).

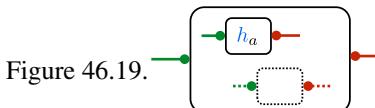


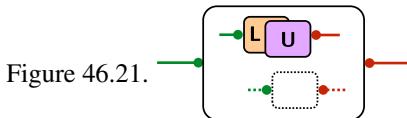
Figure 46.19.

Suppose that we can find two DPs \mathbf{L}, \mathbf{U} that bound the DP h_a (Fig. 46.20).

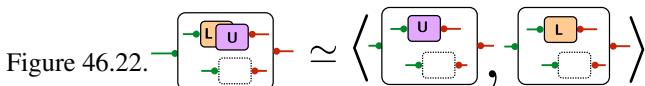


This can model either (a) uncertainty in our knowledge of h_a , or (b) a relaxation that we willingly introduce.

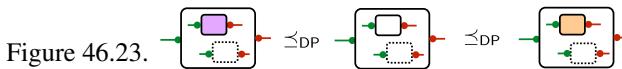
Then we can consider the pair \mathbf{L}, \mathbf{U} as a UDP $\langle \mathbf{L}, \mathbf{U} \rangle$ and we can plug it in the original MCDP in place of h_a (Fig. 46.21).



Given the semantics of interconnections of UDPs (Definition 46.18), this is equivalent to considering a pair of MCDPs, in which we choose either the lower bound or the upper bound (Fig. 46.22).



We can then show that the solution of the original MCDP is bounded below and above by the solution of the new pair of MCDPs (Fig. 46.23).



This result generalizes for any number of substitutions.

Formal statement

First, we define a partial order on the valuations. A valuation precedes another if it gives more information on each DP.

Definition 46.19 (Partial order \leq_V on valuations). For two valuations $v_1, v_2 : \mathcal{A} \rightarrow \mathbf{DP}$, say that $v_1 \leq_V v_2$ if $v_1(a) \leq_{\text{DP}} v_2(a)$ for all $a \in \mathcal{A}$.

At this point, we have enough machinery in place that we can simply state the result as “the semantics is monotone in the valuation”.

Theorem 46.20 (Φ is monotone in the valuation). If $v_1 \leq_V v_2$, then

$$\Phi[\langle \mathcal{A}, \mathbf{T}, v_1 \rangle] \leq_{\text{DP}} \Phi[\langle \mathcal{A}, \mathbf{T}, v_2 \rangle].$$

The proof is given in Appendix Section 46.18.4 in the supplementary materials.

This result says that we can swap any DP in a MCDP with a UDP relaxation, obtain a UMCMDP, which we can solve to obtain inner and outer approximations to the solution of the original MCDP. This shows that considering uncertainty in the MCDP framework is easy; as the problem reduces to solving a pair of problems instead of one. This

The rest of the paper consists of applications of this result.

46.13. Applications

This section shows three example applications of the theory:

1. The first example deals with *parametric uncertainty*.
2. The second example deals with the idea of relaxation of a scalar relation. This is equivalent to accepting a tolerance for a given variable, in exchange for a reduced number of iterations.
3. The third example deals with the relaxation of relations with infinite cardinality. In particular it shows how one can obtain consistent estimates with a finite and prescribed amount of computation.

46.14. Application: Dealing with Parametric Uncertainty

To instantiate the model in Fig. 46.1, we need to obtain numbers for energy density, specific cost, and operating life for all batteries technologies we want to examine.

By browsing Wikipedia, one can find the figures in Table 46.1.

Table 46.1.: Specifications of common batteries technologies

<i>technology</i>	<i>energy density</i> [Wh/kg]	<i>specific cost</i> [Wh/\$]	<i>operating life</i> # 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	150	2.50	600
LFP	90	1.50	1500

Should we trust those figures? Fortunately, we can easily deal with possible mistrust by introducing uncertain DPs.

Formally, we replace the DPs for *energy density*, *specific cost*, *operating life* in Fig. 46.1 with the corresponding Uncertain DPs with a configurable uncertainty. We can then solve the UDPs to obtain a lower bound and an upper bound to the solutions that can be presented to the user.

Fig. 46.24 shows the relation between the provided **endurance** and the minimal **total mass** required, when using uncertainty of 5%, 10%, 25% on the numbers above. Each panel shows two curves: the lower bound (best case analysis) and the upper bound (worst case analysis). In some cases, the lower bound is feasible, but the upper bound is not. For example, in panel *b*, for 10% uncertainty, we can conclude that, notwithstanding the uncertainty, there exists a solution for endurance ≤ 1.3 hours, while for higher endurance, because the upper bound is infeasible, we cannot conclude that there is a solution — though, because the lower bound is feasible, we cannot conclude that a solution does not exist (Fig. 46.24c).

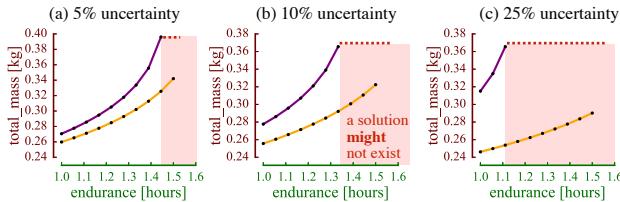


Figure 46.24.: Uncertain relation between endurance and the minimal total mass required, obtained by solving the example in Fig. 46.1 for different values of the uncertainty on the characteristics of the batteries. As the uncertainty increases, there are no solutions for the worst case.

46.15. Application: Introducing Tolerances

Another application of the theory is the introduction of tolerances for any variable in the optimization problem. For example, one might not care about the variations of the battery mass below, say, 1 g. One can then introduce a ± 1 g uncertainty in the definition of the problem by adding a UDP hereby called “uncertain identity”.

46.15.1. The uncertain identity

Let $\alpha > 0$ be a step size. Define floor_α and ceil_α to be the floor and ceil with step size α (Fig. 46.25). By construction, $\text{floor}_\alpha \leq_{\text{DP}} \text{Id} \leq_{\text{DP}} \text{ceil}_\alpha$.

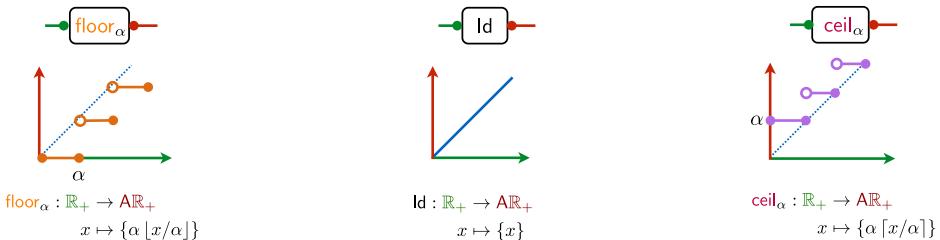


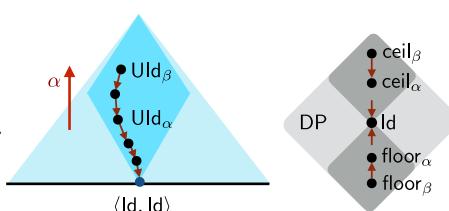
Figure 46.25.: The identity and its two relaxations floor_α and ceil_α .

Let $\text{UId}_\alpha \doteq \langle \text{floor}_\alpha, \text{ceil}_\alpha \rangle$ be the “uncertain identity”. For $0 < \alpha < \beta$, it holds that

$$\text{Id} \prec_{\text{DP}} \text{UId}_\alpha \prec_{\text{DP}} \text{UId}_\beta.$$

Therefore, the sequence UId_α is a descending chain that converges to Id as $\alpha \rightarrow 0$ (Fig. 46.26).

Figure 46.26.



46.15.2. Approximations in MCDP

We can take any edge in an MCDP and apply this relaxation. Formally, we first introduce an identity Id and then relax it using UId_α (Fig. 46.27).

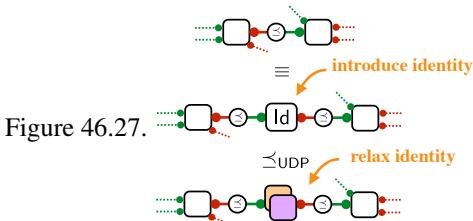


Figure 46.27.

Mathematically, given an MCDP $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$, we generate a UMCDP $\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_\alpha \rangle$, where the new valuation \mathbf{v}_α agrees with \mathbf{v} except on a particular atom $a \in \mathcal{A}$, which is replaced by the series of the original $\mathbf{v}(a)$ and the approximation UId_α :

$$\mathbf{v}_\alpha(a) \doteq \text{series}(\text{UId}_\alpha, \mathbf{v}(a))$$

Call the original and approximated DPs dp and dp_α :

$$\text{dp} \doteq \Phi \llbracket \langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle \rrbracket, \quad \text{dp}_\alpha \doteq \Phi \llbracket \langle \mathcal{A}, \mathbf{T}, \mathbf{v}_\alpha \rangle \rrbracket.$$

Because $\mathbf{v} \preceq_V \mathbf{v}_\alpha$ (in the sense of ???), Theorem 46.20 implies that

$$\text{dp} \leq_{\text{DP}} \text{dp}_\alpha.$$

This means that we can solve Ldp_α and Udp_α and obtain upper and lower bounds for dp . Furthermore, by varying α , we can construct an approximating sequence of DPs whose solution will converge to the solution of the original MCDP.

Numerical results This procedure was applied to the example model in Fig. 46.1 by introducing a tolerance to the “power” variable for the actuation. The tolerance α is chosen at logarithmic intervals between 0.01 mW and 1 W. Fig. 46.28a shows the solutions of the minimal mass required for Ldp_α and Udp_α , as a function of α . Fig. 46.28a confirms the consistency results predicted by the theory. First, if the solutions for both Ldp_α and Udp_α exist, then they are ordered ($\text{Ldp}_\alpha(f) \leq \text{Udp}_\alpha(f)$). Second, as α decreases, the interval shrinks. Third, the bounds are consistent (the solution for the original DP is always contained in the bound).

Next, it is interesting to consider the computational complexity. Fig. 46.28b shows the number of iterations as a function of the resolution α , and the trade-off of the uncertainty of the solution and the computational resources spent. This shows that this approximation scheme is an effective way to reduce the computation load while maintaining a consistent estimate.

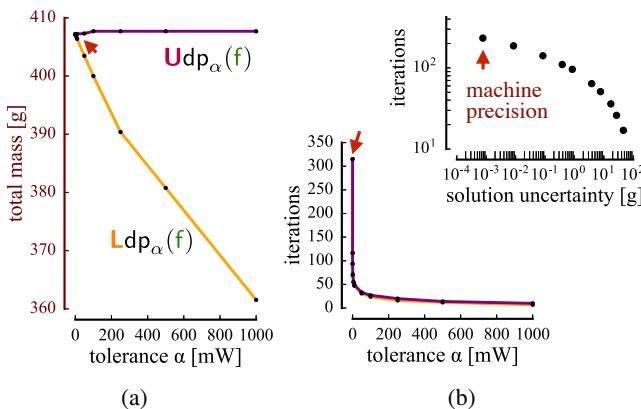


Figure 46.28.: Results of model in Fig. 46.1 when tolerance is applied to the actuation power resource. Please see the supplementary materials for more details.

46.16. Application: Relaxation for relations with infinite cardinality

Another way in which uncertain DPs can be used is to construct approximations of DPs that would be too expensive to solve exactly. For example, consider a relation like

$$\text{travel_distance} \leq \text{velocity} \times \text{endurance}, \quad (46.8)$$

which appears in the model in Fig. 46.1. If we take these three quantities in Eq. (46.8) as belonging to \mathbb{R} , then, for each value of the travel distance, there are infinite pairs of $\langle \text{velocity}, \text{endurance} \rangle$ that are feasible. (On a computer, where the quantities could be represented as floating point numbers, the combinations are properly not “infinite”, but still, extremely large.)

We can avoid considering all combinations by creating a sequence of uncertain DPs that use finite and prescribed computation.

46.16.1. Relaxations for addition

Consider a monotone relation between some functionality $f_1 \in \mathbb{R}_+$ and resources $r_1, r_2 \in \mathbb{R}_+$ described by the constraint that $f_1 \leq r_1 + r_2$ (Fig. 46.29). For example, this could represent the case where there are two batteries providing the power f_1 , and we need to decide how much to allocate to the first (r_1) or the second (r_2).

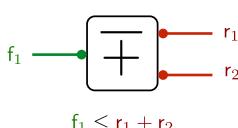


Figure 46.29.

$$f_1 \leq r_1 + r_2$$

The formal definition of this constraint as an DP is

$$\begin{aligned}\bar{+} : \mathbb{R}_+ &\rightarrow \mathcal{A}(\mathbb{R}_+ \times \mathbb{R}_+), \\ \bar{f}_1 &\mapsto \{\langle x, \bar{f}_1 - x \rangle \mid x \in \mathbb{R}_+\}.\end{aligned}$$

Note that, for each value \bar{f}_1 , $\bar{+}(\bar{f}_1)$ is a set of infinite cardinality.

We will now define two sequences of relaxations for $\bar{+}$ with a fixed number of solutions $n \geq 1$.

Using uniform sampling

We will first define a sequence of UDPs S_n based on uniform sampling. Let $\mathbf{U}S_n$ consist of n points sampled on the segment with extrema $\langle 0, \bar{f}_1 \rangle$ and $\langle \bar{f}_1, 0 \rangle$. For $\mathbf{L}S_n$, sample $n+1$ points on the segment and take the *meet* of successive points (Fig. 46.30).

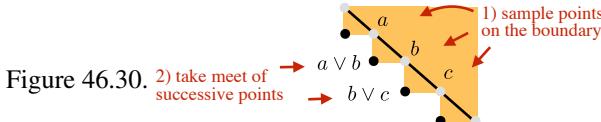


Figure 46.30. 2) take meet of successive points

The first elements of the sequences are shown in Fig. 46.31. One can easily prove that $\mathbf{L}S_n \leq_{\text{DP}} \bar{+} \leq_{\text{DP}} \mathbf{U}S_n$, and thus S_n is a relaxation of $\bar{+}$, in the sense that $\bar{+} \leq_{\text{DP}} S_n$. Moreover, S_n converges to $\bar{+}$ as $n \rightarrow \infty$.

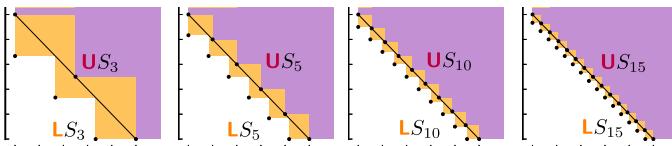


Figure 46.31.: Approximations to $\bar{+}$ using the uniform sampling sequence S_n .

However, note that the convergence is not monotonic: $S_{n+1} \not\leq_{\text{DP}} S_n$. The situation can be represented graphically as in Fig. 46.34a. The sequence S_n eventually converges to $\bar{+}$, but it is not a descending chain. This means that it is not true, in general, that the solution to the MCDP obtained by plugging in S_{n+1} gives smaller bounds than S_n .

Relaxation based on Van Der Corput sequence

We can easily create an approximation sequence $V : \mathbb{N} \rightarrow \mathbf{DP}$ that converges monotonically using Var Der Corput (VDC) sampling [27, Section 5.2]. Let $\text{vdc}(n)$ be the VDC sequence of n elements in the interval $[0, 1]$. The first elements of the VDC are $0, 0.5, 0.25, 0.75, 0.125, \dots$. The sequence is guaranteed to satisfy $\text{vdc}(n) \subseteq \text{vdc}(n+1)$ and

to minimize the discrepancy. The upper bound $\mathbf{U}V_n$ is defined as sampling the segment with extrema $\langle \mathbf{f}_1 \rangle$ and $\langle \mathbf{f}_1, 0 \rangle$ using the VDC sequence:

$$\mathbf{U}V_n : \mathbf{f}_1 \mapsto \{\langle \mathbf{f}_1 x, \mathbf{f}_1(1-x) \rangle \mid x \in \text{vdc}(n)\}.$$

The lower bound $\mathbf{L}V_n$ is defined by taking meets of successive points, according to the procedure in Fig. 46.30.

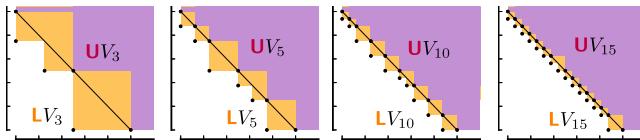


Figure 46.32.: Approximations to $\bar{+}$ using the Van Der Corput sequence V_n .

For this sequence, one can prove that not only $\bar{+} \leq_{\text{DP}} V_n$, but also that the convergence is uniform, in the sense that $\bar{+} \leq_{\text{DP}} V_{n+1} \leq_{\text{DP}} V_n$. The situation is represented graphically in Fig. 46.34b: the sequence is a descending chain that converges to $\bar{+}$.

46.16.2. Inverse of multiplication

The case of multiplication can be treated analogously to the case of addition. By taking the logarithm, the inequality $\mathbf{f}_1 \leq \mathbf{r}_1 \mathbf{r}_2$ can be rewritten as $\log(\mathbf{f}_1) \leq \log(\mathbf{r}_1) + \log(\mathbf{r}_2)$. So we can repeat the constructions done for addition. The VDC sequences are shown in Fig. 46.33.

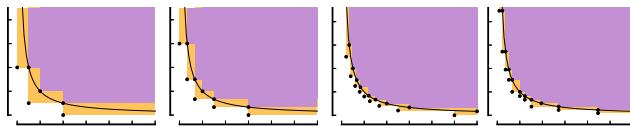


Figure 46.33.: Van Der Corput relaxations for the relation $\mathbf{f}_1 \leq \mathbf{r}_1 \mathbf{r}_2$.

46.16.3. Numerical example

We have applied this relaxation to the relation `travel distance` \leq `velocity` \times `endurance` in the MCDP in Fig. 46.1. Thanks to this theory, we can obtain estimates of the solutions using bounded computation, even though that relation has infinite cardinality.

Fig. 46.34c shows the result using uniform sampling, and Fig. 46.34d shows the result using VDC sampling. As predicted by the theory, uniform sampling does not give monotone convergence, while VDC sampling does.

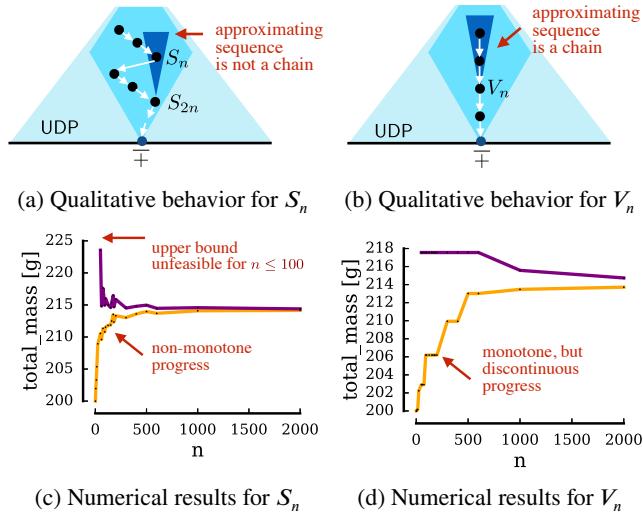


Figure 46.34.: Solutions to the example in Fig. 46.1, applying relaxations for the relation `travel_distance` \leq `velocity` \times `endurance` using the uniform sampling sequence and the VDC sampling sequence. The uniform sampling sequence S_n does not converge monotonically (panel *a*); therefore the progress is not monotonic (panel *c*). Conversely, the Van Der Corput sequence V_n is a descending chain (panel *b*), which results in monotonic progress (panel *d*).

46.17. Conclusions and future work

Monotone Co-Design Problems (MCDPs) provide a compositional theory of “co-design” that describes co-design constraints among different subsystems in a complex system, such as a robotic system.

This paper dealt with the introduction of uncertainty in the framework, specifically, interval uncertainty.

Uncertainty can be used in two roles. First, it can be used to describe limited knowledge in the models. For example, in Section 46.14, we have seen how this can be applied to model mistrust about numbers from Wikipedia. Second, uncertainty allows to generate relaxations of the problem. We have seen two applications: introducing an allowed tolerance in one particular variable (Section 46.15), and dealing with relations with infinite cardinality using bounded computation resources (Section 46.16).

Future work includes strengthening these results. For example, we are not able to predict the resulting uncertainty in the solution before actually computing it; ideally, one would like to know how much computation is needed (measured by the number of points in the antichain approximation) for a given value of the uncertainty that the user can accept.

46.18. Proofs

46.18.1. Proofs of well-formedness of Definition 46.18

As some preliminary business, we need to prove that Definition 46.18 is well formed, in the sense that the way the semantics function Φ is defined, it returns a UDP for each argument. This is not obvious from Definition 46.18.

For example, for $\Phi[\![\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v}]\!]$, the definition gives values for $\mathbf{L}\Phi[\![\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v}]\!]$ and $\mathbf{U}\Phi[\![\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v}]\!]$ separately, without checking that

$$\mathbf{L}\Phi[\![\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v}]\!] \leq_{\mathbf{DP}} \mathbf{U}\Phi[\![\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v}]\!].$$

The following lemma provides the proof for that.

Lemma 46.21. Definition 46.18 is well formed, in the sense that

$$\mathbf{L}\Phi[\!(\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v})\!] \leq_{\mathbf{DP}} \mathbf{U}\Phi[\!(\mathcal{A}, \text{series}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v})\!], \quad (46.9)$$

$$\mathbf{L}\Phi[\!(\mathcal{A}, \text{par}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v})\!] \leq_{\mathbf{DP}} \mathbf{U}\Phi[\!(\mathcal{A}, \text{par}(\mathbf{T}_1, \mathbf{T}_2), \mathbf{v})\!], \quad (46.10)$$

$$\mathbf{L}\Phi[\!(\mathcal{A}, \text{loop}(\mathbf{T}), \mathbf{v})\!] \leq_{\mathbf{DP}} \mathbf{U}\Phi[\!(\mathcal{A}, \text{loop}(\mathbf{T}), \mathbf{v})\!]. \quad (46.11)$$

Proof. Proving Eq. (46.9)—Eq. (46.11) can be reduced to proving the following three results, for any $x, y \in \mathbf{DP}$:

$$\begin{aligned} (\mathbf{L}x \odot \mathbf{L}y) &\leq_{\mathbf{DP}} (\mathbf{U}x \odot \mathbf{U}y), \\ (\mathbf{L}x \otimes \mathbf{L}y) &\leq_{\mathbf{DP}} (\mathbf{U}x \otimes \mathbf{U}y), \\ (\mathbf{L}x)^\dagger &\leq_{\mathbf{DP}} (\mathbf{U}x)^\dagger. \end{aligned}$$

These are given in Lemma 46.22, Lemma 46.23, Lemma 46.24. □

Lemma 46.22. $(\mathbf{L}x \odot \mathbf{L}y) \leq_{\mathbf{DP}} (\mathbf{U}x \odot \mathbf{U}y)$.

Proof. First prove that \odot is monotone in each argument (proved as Lemma 46.25). Then note that

$$(\mathbf{L}x \odot \mathbf{L}y) \leq_{\mathbf{DP}} (\mathbf{L}x \odot \mathbf{U}y) \leq_{\mathbf{DP}} (\mathbf{U}x \odot \mathbf{U}y).$$

□

Lemma 46.23. $(\mathbf{L}x \otimes \mathbf{L}y) \leq_{\mathbf{DP}} (\mathbf{U}x \otimes \mathbf{U}y)$.

Proof. The proof is entirely equivalent to the proof of Lemma 46.22. First prove that par is monotone in each argument (proved as Lemma 46.26). Then note that

$$(\mathbf{L}x \otimes \mathbf{L}y) \leq_{\mathbf{DP}} (\mathbf{L}x \otimes \mathbf{U}y) \leq_{\mathbf{DP}} (\mathbf{U}x \otimes \mathbf{U}y).$$

□

Lemma 46.24. $(\mathbf{L}x)^\dagger \leq_{\mathbf{DP}} (\mathbf{U}x)^\dagger$.

Proof. This follows from the fact that \dagger is monotone (Lemma 46.28). \square

46.18.2. Monotonicity lemmas for DP

These lemmas are used in the proofs above.

Lemma 46.25. $\circledcirc : \mathbf{DP} \times \mathbf{DP} \rightarrow \mathbf{DP}$ is monotone on $\langle \mathbf{DP}, \leq_{\mathbf{DP}} \rangle$.

Proof. In Definition 46.11, \circledcirc is defined as follows for two maps $h_1 : F_1 \rightarrow \mathcal{A}R_1$ and $h_2 : F_2 \rightarrow \mathcal{A}R_2$:

$$h_1 \circledcirc h_2 = \text{Min}_{\leq_{R_2}} \uparrow \bigcup_{s \in h_1(F)} h_2(s).$$

It is useful to decompose this expression as the composition of three maps:

$$h_1 \circledcirc h_2 = m \circ g[h_2] \circ h_1,$$

where “ \circ ” is the usual map composition, and g and m are defined as follows:

$$\begin{aligned} g[h_2] : \mathcal{A}R_1 &\rightarrow \mathcal{U}R_2, \\ R &\mapsto \bigcup_{s \in R} h_2(s), \end{aligned}$$

and

$$\begin{aligned} m : \mathcal{U}R_2 &\rightarrow \mathcal{A}R_2, \\ R &\mapsto \text{Min}_{\leq_{R_2}} R. \end{aligned}$$

From the following facts:

- m is monotone.
- $g[h_2]$ is monotone in h_2 .
- $f_1 \circ f_2$ is monotone in each argument if the other argument is monotone.

Then the thesis follows. \square

Lemma 46.26. $\otimes : \mathbf{DP} \times \mathbf{DP} \rightarrow \mathbf{DP}$ is monotone on $\langle \mathbf{DP}, \leq_{\mathbf{DP}} \rangle$.

Proof. The definition of \otimes (Definition 46.10) is:

$$\begin{aligned} h_1 \otimes h_2 : (F_1 \times F_2) &\rightarrow \mathcal{A}(R_1 \times R_2), \\ \langle f_1, f_2 \rangle &\mapsto h_1(f_1) \times h_2(f_2). \end{aligned}$$

Because of symmetry, it suffices to prove that \otimes is monotone in the first argument, leaving the second fixed.

We need to prove that for any two DPs $\mathbf{h}_a, \mathbf{h}_b$ such that

$$\mathbf{h}_a \leq_{\text{DP}} \mathbf{h}_b, \quad (46.12)$$

and for any fixed $\bar{\mathbf{h}}$, then

$$\mathbf{h}_a \otimes \bar{\mathbf{h}} \leq_{\text{DP}} \mathbf{h}_b \otimes \bar{\mathbf{h}}.$$

Let $R = \bar{\mathbf{h}}(\mathbf{f}_2)$. Then we have that

$$\begin{aligned} [\mathbf{h}_a \otimes \bar{\mathbf{h}}](\mathbf{f}_1, \mathbf{f}_2) &= \mathbf{h}_a(\mathbf{f}_1) \times R, \\ [\mathbf{h}_b \otimes \bar{\mathbf{h}}](\mathbf{f}_1, \mathbf{f}_2) &= \mathbf{h}_b(\mathbf{f}_1) \times R. \end{aligned}$$

Because of Eq. (46.12), we know that

$$\mathbf{h}_a(\mathbf{f}_1) \leq_{\mathcal{A}\mathbf{R}_1} \mathbf{h}_b(\mathbf{f}_1).$$

So the thesis follows from proving that the product of antichains is monotone (Lemma 46.27). \square

Lemma 46.27. The product of antichains $\times : \mathcal{A}\mathbf{R}_1 \times \mathcal{A}\mathbf{R}_2 \rightarrow \mathcal{A}(\mathbf{R}_1 \times \mathbf{R}_2)$ is monotone.

Lemma 46.28. $\dagger : \mathbf{DP} \rightarrow \mathbf{DP}$ is monotone on $\langle \mathbf{DP}, \leq_{\text{DP}} \rangle$.

Proof. Let $\mathbf{h}_1 \leq_{\text{DP}} \mathbf{h}_2$. Then we can prove that $\mathbf{h}_1^\dagger \leq_{\text{DP}} \mathbf{h}_2^\dagger$. From the definition of \dagger (Definition 46.12), we have that

$$\begin{aligned} \mathbf{h}_1^\dagger(\mathbf{f}_1) &= \text{lfp}(\Psi_{\mathbf{f}_1}^{\mathbf{h}_1}), \\ \mathbf{h}_2^\dagger(\mathbf{f}_2) &= \text{lfp}(\Psi_{\mathbf{f}_2}^{\mathbf{h}_2}), \end{aligned}$$

with $\Psi_{\mathbf{f}_1}^{\mathbf{h}}$ defined as

$$\begin{aligned} \Psi_{\mathbf{f}_1}^{\mathbf{h}} : \mathcal{A}\mathbf{R} &\rightarrow \mathcal{A}\mathbf{R}, \\ \mathbf{R} &\mapsto \underset{\leq_{\mathbf{R}}}{\text{Min}} \bigcup_{r \in \mathbf{R}} \mathbf{h}(\mathbf{f}_1, r) \cap \uparrow r. \end{aligned}$$

The least fixed point operator lfp is monotone, so we are left to check that the map

$$\mathbf{h} \mapsto \Psi_{\mathbf{f}_1}^{\mathbf{h}}$$

is monotone. That is the case, because if $\mathbf{h}_1 \leq_{\text{DP}} \mathbf{h}_2$ then

$$\left[\bigcup_{r \in \mathbf{R}} \mathbf{h}_1(\mathbf{f}_1, r) \cap \uparrow r \right] \leq_{\mathcal{A}\mathbf{R}} \left[\bigcup_{r \in \mathbf{R}} \mathbf{h}_2(\mathbf{f}_1, r) \cap \uparrow r \right].$$

\square

46.18.3. Monotonicity of semantics φ

Lemma 46.29 (φ is monotone in the valuation). Suppose that $v_1, v_2 : \mathcal{A} \rightarrow \mathbf{DP}$ are two valuations for which it holds that $v_1(a) \leq_{\mathbf{DP}} v_2(a)$. Then $\varphi[\langle \mathcal{A}, \mathbf{T}, v_1 \rangle] \leq_{\mathbf{DP}} \varphi[\langle \mathcal{A}, \mathbf{T}, v_2 \rangle]$.

Proof. Given the recursive definition of Definition 46.9, we need to prove this just for the base case and for the recursive cases.

The base case, given in Eq. (46.2), is

$$\varphi[\langle \mathcal{A}, a, v \rangle] = v(a), \quad \text{for all } a \in \mathcal{A}.$$

We have

$$\begin{aligned}\varphi[\langle \mathcal{A}, \mathbf{T}, v_1 \rangle] &= v_1(a) \\ \varphi[\langle \mathcal{A}, \mathbf{T}, v_2 \rangle] &= v_2(a)\end{aligned}$$

and $v_1(a) \leq_{\mathbf{DP}} v_2(a)$ by assumption.

For the recursive cases, Eqs. (46.3) to (46.5), the thesis follows from the monotonicity of \odot, \otimes, \dagger , proved in Lemma 46.26, Lemma 46.25, Lemma 46.28. \square

46.18.4. Proof of the main result, Theorem 46.20

We restate the theorem.

Theorem 46.20. If

$$v_1 \leq_V v_2$$

then

$$\Phi[\langle \mathcal{A}, \mathbf{T}, v_1 \rangle] \leq_{\mathbf{DP}} \Phi[\langle \mathcal{A}, \mathbf{T}, v_2 \rangle].$$

Proof. From the definition of Φ and φ , we can derive that

$$\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}, v \rangle] = \varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{L}\circ v \rangle]. \quad (46.13)$$

In particular, for $v = v_1$,

$$\mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}, v_1 \rangle] = \varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{L}\circ v_1 \rangle]. \quad (46.14)$$

Because $v_1(a) \leq_{\mathbf{DP}} v_2(a)$, from Lemma 46.29,

$$\varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{L}\circ v_1 \rangle] \leq_{\mathbf{DP}} \varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{L}\circ v_2 \rangle]. \quad (46.15)$$

From Eq. (46.13) again,

$$\varphi[\langle \mathcal{A}, \mathbf{T}, \mathbf{L}\circ v_2 \rangle] = \mathbf{L}\Phi[\langle \mathcal{A}, \mathbf{T}, v_2 \rangle]. \quad (46.16)$$

From Eq. (46.14), Eq. (46.15), Eq. (46.16) together,

$$\textcolor{brown}{L}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_1 \rangle] \leq_{\text{DP}} \textcolor{brown}{L}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_2 \rangle].$$

Repeating the same reasoning for $\textcolor{red}{U}$, we have

$$\textcolor{red}{U}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_2 \rangle] \leq_{\text{DP}} \textcolor{red}{U}\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_1 \rangle].$$

Therefore

$$\Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_1 \rangle] \leq_{\text{DP}} \Phi[\langle \mathcal{A}, \mathbf{T}, \mathbf{v}_2 \rangle].$$

□

46.19. Software

46.20. Source code

The implementation is available at the repository <http://github.com/AndreaCensi/mcdp/>, in the branch “uncertainty_sep16”.

46.21. Virtual machine

A VMWare virtual machine is available to reproduce the experiments at the URL <https://www.dropbox.com/sh/nfpnfgjh9hpcgvh/AACVZfdVXxMoVqTYiHWaOwHAA?dl=0>.

To reproduce the figures, log in with user password “mcdp”/“mcdp”. Then execute the following commands:

C Supplementary materials: model definition

The figures contained in this Appendix describe a subset of the models used for optimization.

The goal is to give an idea of how a Monotone Co-Design Problem (MCDP) is formalized using the formal language MCDPL.

This Appendix does not explain the syntax of MCDPL. For details, please see the manual available at <http://mcdp.mit.edu>.

C.1 General template

All the MCDPs used in the experiments are instantiation of the same *template*, whose code is shown in Fig. C.1.

Figure C.1: Template `DroneCompleteTemplate`

```
template [
    Battery: `BatteryInterface,
    Actuation: `ActuationInterface,
    Perception: `PerceptionInterface,
    PowerApprox: `PowerApprox]
mcdp {
    provides travel_distance [km]
    provides num_missions [R]
    provides carry_payload [g]

    requires total_cost_ownership [$]
    requires total_mass [g]

    strategy = instance `droneD_complete_v2.Strategy

    actuation_energetics =
        instance specialize [
            Battery: Battery,
            Actuation: Actuation,
            PowerApprox: PowerApprox
        ] `ActuationEnergeticsTemplate

    actuation_energetics.endurance >= strategy.endurance
    actuation_energetics.velocity >= strategy.velocity
    actuation_energetics.num_missions >= num_missions
    actuation_energetics.extra_payload >= carry_payload
    strategy.distance >= travel_distance

    perception = instance Perception
    perception.velocity >= strategy.velocity

    actuation_energetics.extra_power >= perception.power

    required total_mass >= actuation_energetics.total_mass
    total_cost_ownership >= actuation_energetics.total_cost
}
```

Note the top-level functionality:

- `travel_distance` [km];
- `num_missions` (unitless);
- `carry_payload` [g]

and the top-level resources:

- `total_mass` [g]
- `total_cost_ownership` [USD]

The template has four parameters:

- `Battery`: MCDP for energetics;
- `Actuation`: MCDP for actuation;
- `Perception`: MCDP for perception;
- `PowerApprox`: MCDP describing the tolerance for the power variable. This is used in Section VII-B.

Every experiment chooses different values for the parameters of this template.

The graphical representation of the template is shown in Fig. C.3. The dotted blue containers represent the “holes” that need to be filled to instantiate the template.

In turn, the template contains a specialization call to another template, called `ActuationEnergeticsTemplate`, whose code is shown in Fig. C.2 and whose graphical representation is shown in Fig. C.4.

Figure C.2: Template `ActuationEnergeticsTemplate`

```
template [Battery: `BatteryInterface,
          Actuation: `ActuationInterface,
          PowerApprox: `PowerApprox
] mcdp {
    provides endurance [s]
    provides extra_payload [kg]
    provides extra_power [W]
    provides num_missions [R]
    provides velocity [m/s]

    requires total_cost [$]

    battery = instance Battery
    actuation = instance Actuation

    total_power0 = power required by actuation + extra_power
    power_approx = instance PowerApprox
    total_power0 <= power_approx.power
    total_power = power required by power_approx

    capacity provided by battery >= provided endurance * total_power

    total_mass = (
        mass required by battery +
        actuator_mass required by actuation
        + extra_payload)

    gravity = 9.81 m/s^2
    weight = total_mass * gravity

    lift provided by actuation >= weight
    velocity provided by actuation >= velocity

    labor_cost = (10 $) * (maintenance required by battery)

    required total_cost >= (
        cost required by actuation +
        cost required by battery +
        labor_cost)

    battery.missions >= num_missions

    requires total_mass >= total_mass
}
```

The template has functionality `endurance`, `extra_payload`, `extra_power`, `num_missions`, `velocity`, and two resources, `total_mass` and `total_cost`

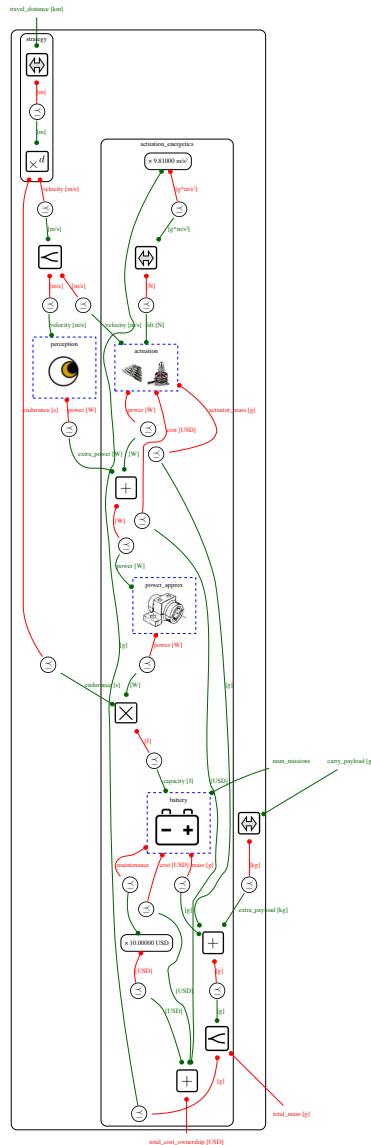
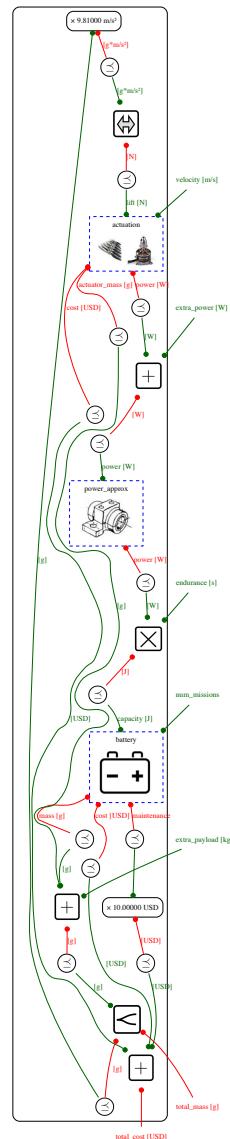
Figure C.3: Graphical representation for the template `droneCompleteTemplate` (Fig. ??)

Figure C.4: Graphical representation for the template `ActuationEnergeticsTemplate` (Fig. C.2)

C.2 MCDP defining batteries properties

Fig. C.5 shows the definition of a single battery technology in terms of specific energy, specific cost, and lifetime (number of cycles).

Figure C.5: Definition of `Battery_LiPo` MCDP

```

1 mcdp {
2   provides capacity [J]
3   provides missions [R]
4
5   requires mass      [g]
6   requires cost      [$]
7
8   # Number of replacements
9   requires maintenance [R]
10
11  # Battery properties
12  specific_energy = 150 Wh/kg
13  specific_cost = 2.50 Wh/$
14  cycles = 600 []
15
16  # Constraint between mass and capacity
17  mass >= capacity / specific_energy
18
19  # How many times should it be replaced?
20  num_replacements = ceil(missions / cycles)
21  maintenance >= num_replacements
22
23  # Cost is proportional to number of replacements
24  cost >= (capacity / specific_cost) * num_replacements
25 }
```

Here a battery is abstracted as a DP with `functionality`:

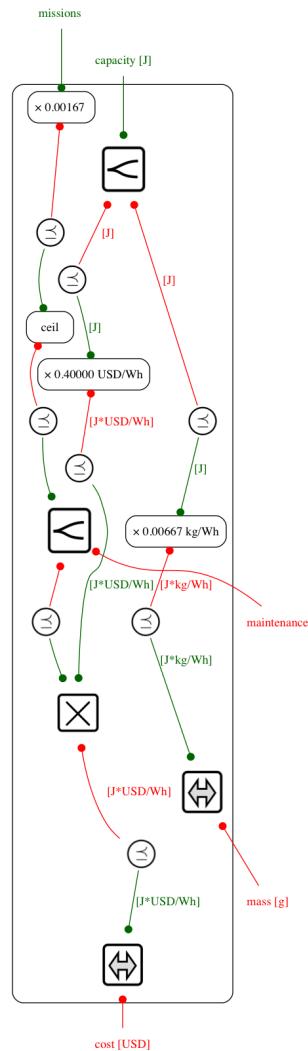
- `capacity` [J];
- `missions` (unitless)

and with `resources`:

- `mass` [g]
- `cost` [USD]

The corresponding graphical representation is shown in Fig. C.6.

Figure C.6: Definition of battery technology



Because this MCDP is completely specified, as opposed to the two *templates* shown earlier, we can show its algebraic representation, as defined in Def. 6.

The MCDP interpreter takes the code shown in Fig. C.5 and then builds an intermediate graphical representation like the one shown in Fig. C.6. Finally, it is compiled to an algebraic representation $\langle \mathcal{A}, \mathbf{T}, \mathbf{v} \rangle$, where \mathbf{T} is a tree in the $\{\text{series}, \text{par}, \text{loop}\}$ algebra.

A representation of \mathbf{T} for this example is shown in Fig. C.7.

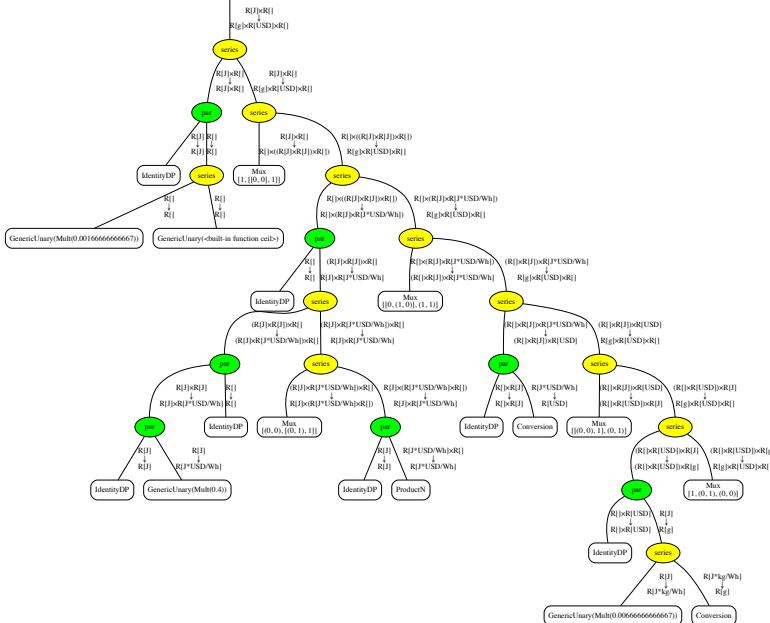
Each edge is the tree is labeled with the signature of the DP, in the form $\text{F} \rightarrow \text{R}$. The junctions are one of the $\{\text{series}, \text{par}, \text{loop}\}$ operators. (The operator *loop* does not appear in this example.)

The leaves are labeled with a representation of the Python class that implements them. In particular, the frequently-appearing *mux* type represents various multiplexing operations, such as

$$\langle x, y, z \rangle \mapsto \langle \langle z, y \rangle, x \rangle.$$

These are necessary to transform a graph into a tree representation.

Figure C.7: Algebraic representation for the example in Fig. C.5



C.3 Choice between different batteries

Just like we defined `Battery_LiPo` (Fig. C.5), other batteries technologies are similarly defined, such as `Battery_NiMH`, `Battery_LCO`, etc.

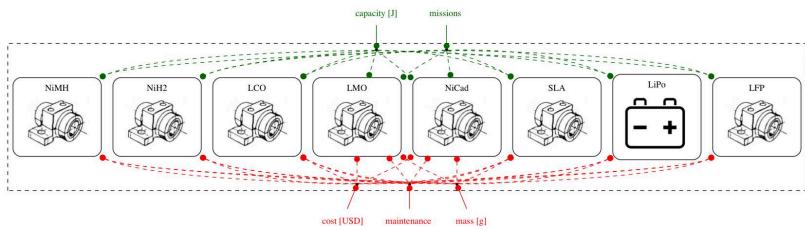
Then we can easily express the choice between any of them using the keyword `choose`, as in Fig. C.8.

Figure C.8: Definition of the `Batteries` MCDP

```

1  choose(
2      NiMH: (load Battery_NiMH),
3      NiH2: (load Battery_NiH2),
4      LCO: (load Battery_LCO),
5      LMO: (load Battery_LMO),
6      NiCad: (load Battery_NiCad),
7      SLA: (load Battery_SLA),
8      LiPO: (load Battery_LiPO),
9      LFP: (load Battery_LFP)
10 )

```



The choice between different batteries is modeled by a *coproduct* operator. This is another type of junction, in addition to `series`, `par`, `loop` that was not described in the paper.

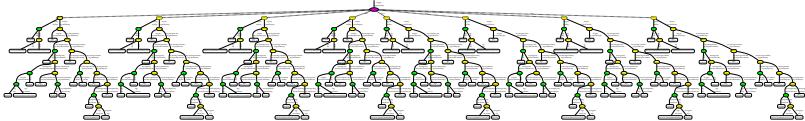
Formally, the coproduct operator it is defined as follows:

$$h_1 \sqcup \cdots \sqcup h_n : \mathcal{F} \rightarrow \text{AR}, \quad (1)$$

$$\text{f} \mapsto \underset{\preceq_{\mathcal{R}}}{\text{Min}} (h_1(\text{f}) \sqcup \cdots \sqcup h_n(\text{f})). \quad (2)$$

The algebraic representation (Fig. C.9) contains then one branch for each type of battery.

Figure C.9: Algebraic representation of the `Batteries` MCDP



C.4 Describing uncertainty

This is a description of the Uncertain MCDPs used in the experiments in Section VII-A.

MCDPL has an `Uncertain` operator that can describe interval uncertainty.

For example, the MCDP in Fig. C.5 is rewritten with uncertainty to obtain the code in Fig. C.10.

The figures have a 5% uncertainty added to them.

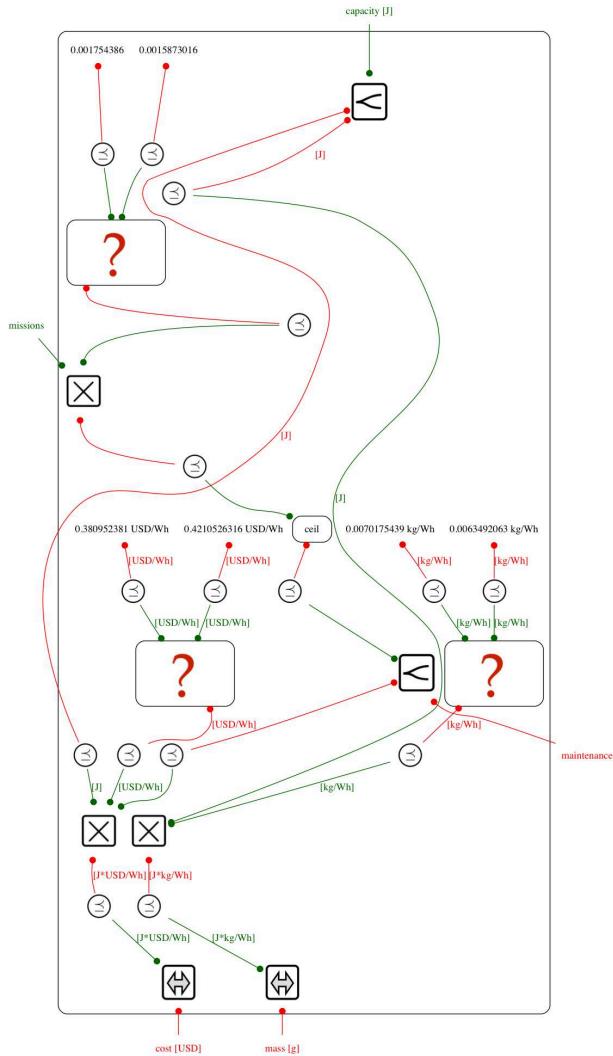
Figure C.10: Definition of `Battery_LiPo` MCDP with 5% uncertainty on parameters

```

1 mcdp {
2   provides capacity [J]
3   provides missions [R]
4
5   requires mass      [g]
6   requires cost      [USD]
7
8   # Number of replacements
9   requires maintenance [R]
10
11  # Battery properties
12  specific_energy_inv = Uncertain(1.0 [] / 157.5 Wh/kg, 1.0 [] / 142.5 Wh/kg)
13  specific_cost_inv = Uncertain(1.0 [] / 2.625 Wh/USD, 1.0 [] / 2.375 Wh/USD)
14  cycles_inv = Uncertain(1.0 []/630.0 [], 1.0[]/ 570.0 [])
15
16  # Constraint between mass and capacity
17  massc = provided capacity * specific_energy_inv
18
19  # How many times should it be replaced?
20  num_replacements = ceil(provided missions * cycles_inv)
21  required maintenance >= num_replacements
22
23  # Cost is proportional to number of replacements
24  costc = (provided capacity * specific_cost_inv) * num_replacements
25
26  required cost >= costc
27  required mass >= massc
28 }
```

In the graphical representation, the uncertainty is represented as “uncertainty gates” that have two branches: one for best case and one for worst case (Fig. C.11).

Figure C.11: Graphical representation of uncertain MCDP in Fig. C.10



C.5 Specialization of templates

Once all the single pieces are defined, then the final MCDP is assembled using the `specialize` keyword.

For example, the following code specializes the template using only the `Battery_LiPo` MCDP.

Figure C.12:

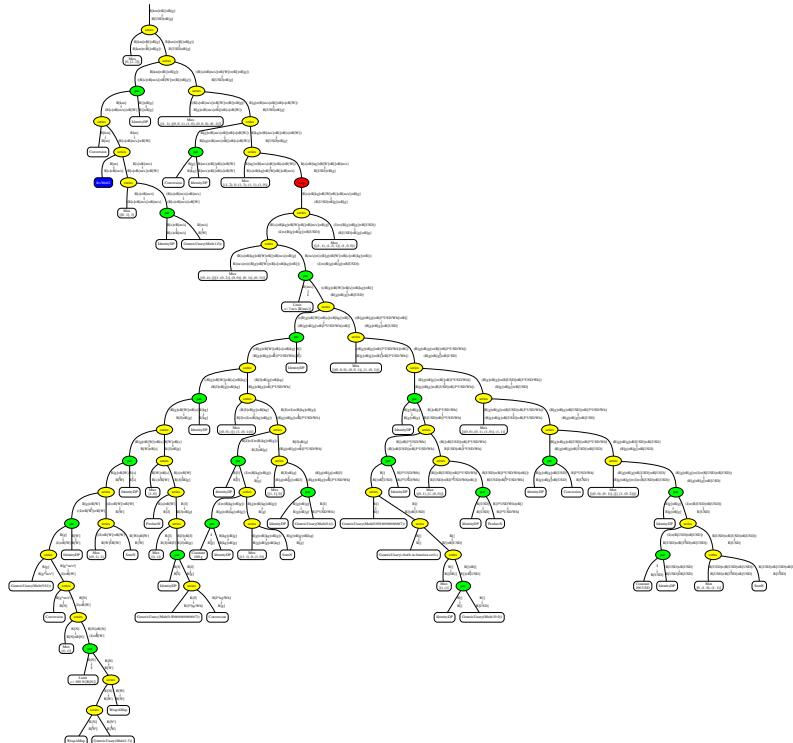
```

1 specialize [
2   Battery: `batteries_nodisc.Battery_LiPo,
3   Actuation: `droneD_complete_v2.Actuation,
4   Perception: `Perception1,
5   PowerApprox: `PowerApprox
6 ]
7 DroneCompleteTemplate

```

The algebraic representation is shown in Fig. C.13.

Figure C.13: Algebraic representation of MCDP in Fig. C.12



The following code specializes the template using the coproduct of all batteries, each having an uncertain specification.

Figure C.14:

```

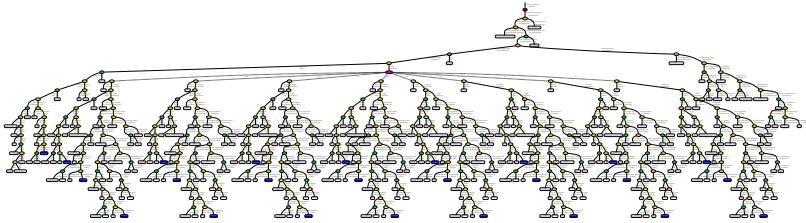
1 specialize [
2   Battery: batteries_uncertain1.batteries,
3   Actuation: droneD_complete_v2.Actuation,
4   PowerApprox: mcdp {
5     provides power [W]
6     requires power [W]
7     required power >= approxu(provided power, 1 mW)
8   }
9 ]
10 ] `ActuationEnergeticsTemplate

```

The algebraic representation is shown in Fig. C.15.

The blue nodes are the uncertain nodes (UDPs).

Figure C.15: Algebraic representation of MCDP in Fig. C.14



47. Relationship between products

47.1. Biproduct, Product, and Coproduct of Design Problems

Example 47.1. Just as Beau is about to connect the engine diagram to the larger design problem for the X103, his supervisor, Elly May, comes up behind him and catches a glance at his diagram. “Beau, that’s almost right: Jeb-XX \wedge Bob-Roc does indeed work as an approximation of a generic “engine” design problem. The problem is that the choice of engine is so important and expensive that it isn’t up to the engineering team—it’s up to the politicians! So take $f \vee g$ out and stick the one reported in Fig. 47.1 instead.

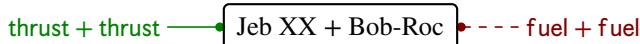


Figure 47.1.: The choice of engine is up to the politicians: Biproduct.

We’ll come back and adjust the parameter to either the Jeb XX or the Bob-Roc after the politicians make their choice.”

Definition 47.2 (Biproduct of design problems). Given design problems $f : \mathbf{A} \leftrightarrow \mathbf{B}$ and $g : \mathbf{C} \leftrightarrow \mathbf{D}$, their *biproduct* $(f + g) : \mathbf{A} + \mathbf{C} \leftrightarrow \mathbf{B} + \mathbf{D}$ is defined by

$$\begin{aligned}
 (f + g) : (\mathbf{A} + \mathbf{C})^{\text{op}} \times (\mathbf{B} + \mathbf{D}) &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\
 \langle \langle 1, \mathbf{a} \rangle^*, \langle 1, \mathbf{b} \rangle \rangle &\mapsto f(\mathbf{a}^*, \mathbf{b}), \\
 \langle \langle 2, \mathbf{c} \rangle^*, \langle 1, \mathbf{b} \rangle \rangle &\mapsto \perp, \\
 \langle \langle 1, \mathbf{a} \rangle^*, \langle 2, \mathbf{d} \rangle \rangle &\mapsto \perp, \\
 \langle \langle 2, \mathbf{c} \rangle^*, \langle 2, \mathbf{d} \rangle \rangle &\mapsto g(\mathbf{c}^*, \mathbf{d}),
 \end{aligned} \tag{47.1}$$

and represented as in Fig. 47.2.

In particular, when $f, g : \mathbf{A} \leftrightarrow \mathbf{B}$, $(f + g) : \mathbf{A} + \mathbf{A} \leftrightarrow \mathbf{B} + \mathbf{B}$ is defined by

$$\begin{aligned}
 (f + g) : (\mathbf{A} + \mathbf{A})^{\text{op}} \times (\mathbf{B} + \mathbf{B}) &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\
 \langle \langle 1, \mathbf{a} \rangle^*, \langle 1, \mathbf{b} \rangle \rangle &\mapsto f(\mathbf{a}^*, \mathbf{b}), \\
 \langle \langle 2, \mathbf{a} \rangle^*, \langle 1, \mathbf{b} \rangle \rangle &\mapsto \perp, \\
 \langle \langle 1, \mathbf{a} \rangle^*, \langle 2, \mathbf{b} \rangle \rangle &\mapsto \perp, \\
 \langle \langle 2, \mathbf{a} \rangle^*, \langle 2, \mathbf{b} \rangle \rangle &\mapsto g(\mathbf{a}^*, \mathbf{b}).
 \end{aligned} \tag{47.2}$$

Assume $f, g : \mathbf{A} \leftrightarrow \mathbf{B}$. Intuitively, $f + g$ can be thought of as: pick either f or g , then throw away the other one, whereas on any $\langle \mathbf{a}^*, \mathbf{b} \rangle$, $f \vee g$ always picks the better (more

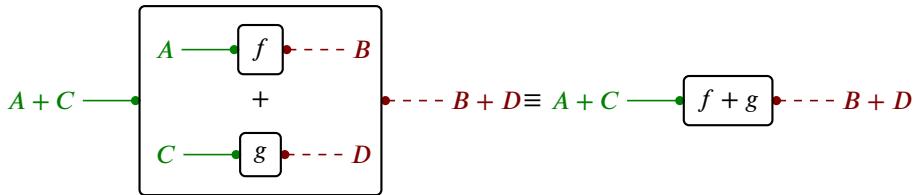


Figure 47.2.: Diagrammatic representation of the biproduct of design problems.

feasible) of either $f(a^*, b)$ or $g(a^*, b)$. Note that $f + g$ introduces an extra parameter, since the choice of f or g has to be hard-coded into the larger design problem.

In general, the biproduct is defined first on objects of the category, so what we are calling a biproduct of design problems is actually the unique map derived from the biproduct $A + B$ of posets in **DP**, namely the disjoint union. In the more general setting $f : A \leftrightarrow B$ and $g : C \leftrightarrow D$, the biproduct on objects $A + C$ models A and C as *interchangeable entities*: $f + g$ may output one functionality from either A or C , and similarly it requires only one resource from either B or D . We can emphasize one or the other condition by letting $A = C$ or $B = D$ above, thus deriving the product of design problems and coproduct of design problems, respectively.

Definition 47.3 (Coproduct of design problems). Given two design problems $f : A \leftrightarrow C$ and $g : B \leftrightarrow C$, their *coproduct* $(f \sqcup g) : A + B \leftrightarrow C$ is defined by

$$(f \sqcup g) : (A + B)^{\text{op}} \times C \rightarrow_{\text{Pos}} \text{Bool},$$

$$\begin{aligned} \langle a^*, c \rangle &\mapsto f(a^*, c), \\ \langle b^*, c \rangle &\mapsto g(b^*, c), \end{aligned} \quad (47.3)$$

and represented as in Fig. 47.3.

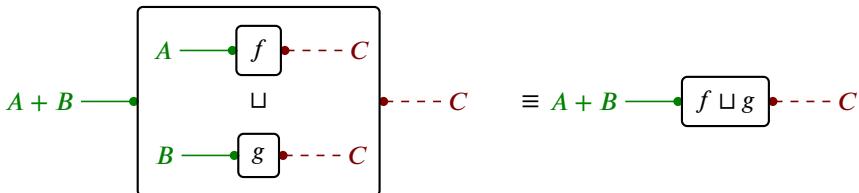


Figure 47.3.: Diagrammatic representation of the coproduct of design problems.

Definition 47.4 (Product of design problems). Given design problems $f : C \leftrightarrow A$, $g : C \leftrightarrow B$, their *product* $(f \times g) : C \leftrightarrow A + B$ is defined by

$$(f \times g) : C^{\text{op}} \times (A + B) \rightarrow_{\text{Pos}} \text{Bool},$$

$$\begin{aligned} \langle c^*, a \rangle &\mapsto f(c^*, a), \\ \langle c^*, b \rangle &\mapsto g(c^*, b), \end{aligned} \quad (47.4)$$

and represented as in Fig. 47.4.

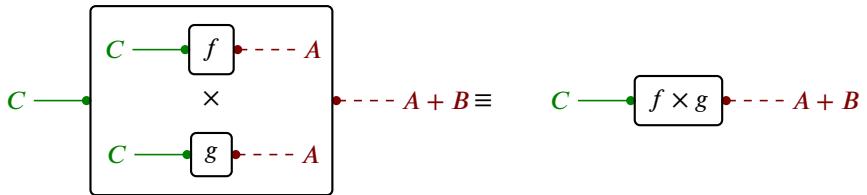


Figure 47.4.: Diagrammatic representation of the product of design problems.

To show that \$A + B\$ is in fact a biproduct in the categorical sense, we need to show that it satisfies certain properties. These properties guarantee that the biproduct is the most “efficient” way of combining two objects in **DP** in a certain sense, and that the resulting combination is unique when it exists. Specifically, we need to show that \$A + B\$ is both a product \$\langle +, \pi_A, \pi_B \rangle\$ and a coproduct \$\langle +, \iota_A, \iota_B \rangle\$ in **DP**, and satisfies an extra coherence condition: \$\pi_B \circ \iota_B = \text{id}_B\$.

Definition 47.5 (Initial and terminal object). Let **C** be a category and let \$A \in \mathbf{C}\$ be an object. We say that \$A\$ is an *initial object* if, for all \$B \in \mathbf{C}\$, the hom-set \$\text{Hom}_{\mathbf{C}}(A, B)\$ has exactly one element. We say that \$A\$ is a *terminal object* if, for all \$D \in \mathbf{C}\$, the hom-set \$\text{Hom}_{\mathbf{C}}(D, A)\$ has exactly one element.

Definition 47.6 (Finite coproducts and products). We say that **C** has *finite coproducts* if it has an initial object and every pair of objects in **C** has a coproduct. We say that **C** has *finite products* if it has a terminal object and every pair of objects in **C** has a product.

Example 47.7. The category **Pos** has finite coproducts and finite products. The coproduct \$\sqcup_{\mathbf{Pos}}\$ is the disjoint union \$+\$. The product \$\times_{\mathbf{Pos}}\$ is the Cartesian product \$\times\$.

Lemma 47.8. The category **DP** has finite coproducts and finite products. The coproduct \$\sqcup_{\mathbf{DP}}\$ is the disjoint union \$+\$ (Definition 19.14). The product \$\times_{\mathbf{DP}}\$ is also the disjoint union \$+\$.

Proof. Let \$\iota_A\$ be the injection of posets \$A \rightarrow A + B\$. We define the design problems

$$\begin{aligned} \hat{\iota}_A : A &\leftrightarrow A + B \\ \langle a^*, c \rangle &\mapsto \iota_A(a) \leq_{A+B} c, \end{aligned} \tag{47.5}$$

and

$$\begin{aligned} \hat{\iota}_B : B &\leftrightarrow A + B \\ \langle b^*, c \rangle &\mapsto \iota_B(b) \leq_{A+B} c. \end{aligned} \tag{47.6}$$

Recall the coproduct of design problems in Definition 47.3. To show that $A + B$ is a coproduct (of posets) in **DP**, we need to show that the coproduct $f \sqcup g$ of design problems is unique and that it satisfies $f = \widehat{\iota_A} \circ (f \sqcup g)$ and $g = \widehat{\iota_B} \circ (f \sqcup g)$. This can be verified by simply writing out the composition. In the following we denote \leq_{A+B} by \leq . One has

$$\begin{aligned}
& (\widehat{\iota_A} \circ (f \sqcup g))(\textcolor{green}{a}^*, \textcolor{red}{c}) \\
&= \bigvee_{x' \in A+B} \widehat{\iota_A}(\textcolor{green}{a}^*, \textcolor{red}{x}') \wedge (f \sqcup g)(\textcolor{green}{x}'^*, \textcolor{red}{c}) \\
&= \bigvee_{\langle 1, x \rangle \in A+B} (\iota_A(\textcolor{green}{a}) \leq \langle 1, \textcolor{red}{x} \rangle) \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c}) \vee \bigvee_{\langle 2, x \rangle \in A+B} (\iota_A(\textcolor{green}{a}) \leq \langle 2, \textcolor{red}{x} \rangle) \wedge g(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= \bigvee_{\langle 1, x \rangle \in A+B} ((\textcolor{green}{a} \leq \textcolor{red}{x}) \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c})) \vee \bigvee_{\langle 2, x \rangle \in A+B} (\perp \wedge g(\textcolor{green}{x}^*, \textcolor{red}{c})) \\
&= \bigvee_{\langle 1, x \rangle \in A+B} (\textcolor{green}{a} \leq \textcolor{red}{x}) \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= \bigvee_{x \in A} (\textcolor{green}{a} \leq \textcolor{red}{x}) \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= (\text{id}_A \circ f)(\textcolor{green}{a}^*, \textcolor{red}{c}) \\
&= f(\textcolor{green}{a}^*, \textcolor{red}{c}). \tag{47.7}
\end{aligned}$$

Similarly:

$$\begin{aligned}
& (\widehat{\iota_B} \circ (f \sqcup g))(\textcolor{blue}{b}^*, \textcolor{red}{c}) \\
&= \bigvee_{x' \in A+B} \widehat{\iota_B}(\textcolor{blue}{b}^*, \textcolor{red}{x}) \wedge (f \sqcup g)(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= \bigvee_{\langle 1, x \rangle \in A+B} (\iota_B(\textcolor{blue}{b}) \leq \langle 1, \textcolor{red}{x} \rangle) \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c}) \vee \bigvee_{\langle 2, x \rangle \in A+B} (\iota_B(\textcolor{blue}{b}) \leq \langle 2, \textcolor{red}{x} \rangle) \wedge g(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= \bigvee_{\langle 1, x \rangle \in A+B} (\perp \wedge f(\textcolor{green}{x}^*, \textcolor{red}{c})) \vee \bigvee_{\langle 2, x \rangle \in A+B} ((\textcolor{blue}{b} \leq \textcolor{red}{x}) \wedge g(\textcolor{green}{x}^*, \textcolor{red}{c})) \\
&= \bigvee_{\langle 2, x \rangle \in A+B} (\textcolor{blue}{b} \leq \textcolor{red}{x}) \wedge g(\textcolor{green}{x}^*, \textcolor{red}{c}) \\
&= g(\textcolor{blue}{b}^*, \textcolor{red}{c}). \tag{47.8}
\end{aligned}$$

The proof that the disjoint union $+$ is also a categorical product in **DP** is analogous, now with $f : \textcolor{blue}{C} \leftrightarrow \textcolor{red}{A}$, $g : \textcolor{blue}{C} \leftrightarrow \textcolor{red}{B}$, and replacing the injections ι_A with projections:

$$\begin{aligned}
& \widehat{\pi_A} : \textcolor{blue}{A} + \textcolor{blue}{B} \leftrightarrow \textcolor{red}{A} \\
& \langle \textcolor{green}{c}^*, \textcolor{red}{a} \rangle \mapsto \pi_A(\textcolor{green}{c}) \leq_A \textcolor{red}{a}, \tag{47.9}
\end{aligned}$$

and

$$\begin{aligned}
& \widehat{\pi_B} : \textcolor{blue}{A} + \textcolor{blue}{B} \leftrightarrow \textcolor{red}{B} \\
& \langle \textcolor{green}{c}^*, \textcolor{red}{b} \rangle \mapsto \pi_B(\textcolor{green}{c}) \leq_B \textcolor{red}{b}, \tag{47.10}
\end{aligned}$$

and the coproduct $f \sqcup g$ with the product (on morphisms) $f \times g : \textcolor{blue}{C} \leftrightarrow \textcolor{red}{A} + \textcolor{red}{B}$

from Definition 47.4. We have

$$\begin{aligned}
 & ((f \times g) ; \widehat{\pi}_A) (\textcolor{blue}{c}^*, \textcolor{red}{a}) \\
 &= \bigvee_{x' \in A+B} (f \times g)(\textcolor{blue}{c}^*, \textcolor{red}{x}') \wedge \widehat{\pi}_A(\textcolor{blue}{x}'^*, \textcolor{red}{a}) \\
 &= \bigvee_{\langle 1, x \rangle \in A+B} (f(\textcolor{blue}{c}^*, x) \wedge \widehat{\pi}_A(\langle 1, \textcolor{blue}{x} \rangle^*, \textcolor{red}{a})) \vee \bigvee_{\langle 2, x \rangle \in A+B} (g(\textcolor{blue}{c}^*, x) \wedge \widehat{\pi}_A(\langle 1, \textcolor{blue}{x} \rangle^*, \textcolor{red}{a})) \\
 &= \bigvee_{\langle 1, x \rangle \in A+B} (f(\textcolor{blue}{c}^*, x) \wedge (x \leq \textcolor{red}{a})) \vee \bigvee_{\langle 2, x \rangle \in A+B} (g(\textcolor{blue}{c}^*, x) \wedge \perp) \\
 &= \bigvee_{\langle 1, x \rangle \in A+B} f(\textcolor{blue}{c}^*, x) \wedge (x \leq \textcolor{red}{a}) \\
 &= f(\textcolor{blue}{c}^*, \textcolor{red}{a}).
 \end{aligned} \tag{47.11}$$

Similarly:

$$\begin{aligned}
 & ((f \times g) ; \widehat{\pi}_B) (\textcolor{blue}{c}^*, \textcolor{red}{b}) \\
 &= \bigvee_{x' \in A+B} (f \times g)(\textcolor{blue}{c}^*, \textcolor{red}{x}') \wedge \widehat{\pi}_B(\textcolor{blue}{x}'^*, \textcolor{red}{b}) \\
 &= \bigvee_{\langle 1, x \rangle \in A+B} (f(\textcolor{blue}{c}^*, x) \wedge \widehat{\pi}_B(\langle 1, \textcolor{blue}{x} \rangle^*, \textcolor{red}{b})) \vee \bigvee_{\langle 2, x \rangle \in A+B} (g(\textcolor{blue}{c}^*, x) \wedge \widehat{\pi}_B(\langle 2, \textcolor{blue}{x} \rangle^*, \textcolor{red}{b})) \\
 &= \bigvee_{\langle 1, x \rangle \in A+B} (f(\textcolor{blue}{c}^*, x) \wedge \perp) \vee \bigvee_{\langle 2, x \rangle \in A+B} (g(\textcolor{blue}{c}^*, x) \wedge (x \leq \textcolor{red}{b})) \\
 &= \bigvee_{\langle 2, x \rangle \in A+B} g(\textcolor{blue}{c}^*, x) \wedge (x \leq \textcolor{red}{b}) \\
 &= g(\textcolor{blue}{c}^*, \textcolor{red}{b}).
 \end{aligned} \tag{47.12}$$

□

Remark 47.9. Where it is clear from context, we will not distinguish between the injection of posets ι_A and its corresponding design problem $\widehat{\iota}_A$, and refer to both by ι_A . The same holds for the projections π_A and $\widehat{\pi}_A$.

Definition 47.10 (Zero object and morphism). We call an object that is both initial and terminal the *zero object* and we indicate it as 0. For any other pairs of objects $A, B \in \mathbf{C}$, there is a unique morphism of the form $A \rightarrow 0 \rightarrow B$; we call it the *zero morphism* and denote it $0_{A,B} : A \rightarrow B$.

Example 47.11. \mathbf{Pos} has an initial object \emptyset and a terminal object $\{1\}$, but no zero object.

Lemma 47.12. In the category \mathbf{DP} , the empty poset \emptyset is a zero object. The zero morphism $0_{P,Q} : P \rightarrow Q$ is the design problem that is always infeasible.

Proof. First of all, \emptyset is an initial object in $\$DP$. Indeed, for any poset P , there is a unique design problem $\emptyset \leftrightarrow P$ given by the unique monotone map $\emptyset = \emptyset^{\text{op}} \times P \rightarrow \text{Bool}$. Similarly, there is a unique design problem $P \leftrightarrow \emptyset$, so \emptyset is also terminal. For any posets P, Q , the zero map $0 : P \leftrightarrow Q$ is the monotone map $P^{\text{op}} \times Q \rightarrow_{\text{Pos}} \text{Bool}$ sending everything to \perp . \square

Definition 47.13 (Biproduct). Let \mathbf{C} be a category with a zero-object 0 , and let $C_1, C_2 \in \mathbf{C}$ be objects. Suppose that $\langle D, \pi_1, \pi_2, \iota_1, \iota_2 \rangle$ is a tuple such that $\langle D, \pi_1, \pi_2 \rangle$ is a product of C_1 and C_2 and $\langle D, \iota_1, \iota_2 \rangle$ is a coproduct of C_1 and C_2 . We say that it is a *biproduct* if, for each $1 \leq i, j \leq 2$, we have

$$\iota_i ; \pi_j = \begin{cases} \text{id}_{C_i}, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases} \quad (47.13)$$

as maps $C_i \rightarrow C_j$ in \mathbf{C} .

Lemma 47.14. The disjoint union is a biproduct for \mathbf{DP} .

Proof. We have already shown that the disjoint union is both a product and a coproduct. We just need to verify that Eq. (47.13) holds for the design problems $\pi_A, \pi_B, \iota_A, \iota_B$. This amounts to checking the four conditions:

$$\begin{aligned} (\iota_A ; \pi_A) &= \text{id}_A, \\ (\iota_A ; \pi_B) &= 0_{A,B}, \\ (\iota_B ; \pi_A) &= 0_{B,A}, \\ (\iota_B ; \pi_B) &= \text{id}_B. \end{aligned} \quad (47.14)$$

We check only the first two, as the other two are similar. To check the second condition, we compute an explicit expression for $(\iota_A ; \pi_B) : A \leftrightarrow B$, using the definition of

Category	Set	Pos	DP
Objects	sets	posets	posets
Morphisms	functions	monotone functions	design problems
Product (objects)	Cartesian product	Cartesian product	disjoint union
Product (morphisms)	(not used)	(not used)	$\times_{\mathbf{DP}}$
Coproduct (objects)	disjoint union	disjoint union	disjoint union
Coproduct (morphisms)	(not used)	(not used)	$\sqcup_{\mathbf{DP}}$
Biproduct (morphisms)	none	none	disjoint union
Tensor Product	\times or \sqcup	\times	\times
Initial object	\emptyset	\emptyset	\emptyset
Terminal object	{1}	{1}	{1}

Table 47.1.: A comparison of **Pos**, **Set**, and **DP**.

design problem series composition:

$$\begin{aligned} (\iota_A \circ \pi_B) : \mathbf{A}^{\text{op}} \times \mathbf{B} &\rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \langle \mathbf{x}^*, \mathbf{z} \rangle &\mapsto \bigvee_{y \in A+B} \iota_A(\mathbf{x}^*, \mathbf{y}) \wedge \pi_B(\mathbf{y}^*, \mathbf{z}). \end{aligned} \quad (47.15)$$

We can do a case analysis for $y \in A + B$. Suppose $y \in A$. Then $\pi_B(\mathbf{y}^*, \mathbf{z}) = \perp$. If $y \in B$, then $\iota_A(\mathbf{x}^*, \mathbf{y}) = \perp$. Therefore, the sum is always false, and hence $(\iota_A \circ \pi_B) = 0_{A,B}$. For the first condition, we compute an explicit expression for $(\iota_A \circ \pi_A) : \mathbf{A} \leftrightarrow \mathbf{A}$:

$$\begin{aligned} (\iota_A \circ \pi_A) : \mathbf{A}^{\text{op}} \times \mathbf{A} &\rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \langle \mathbf{x}^*, \mathbf{z} \rangle &\mapsto \bigvee_{y \in A+B} \iota_A(\mathbf{x}^*, \mathbf{y}) \wedge \pi_A(\mathbf{y}^*, \mathbf{z}). \end{aligned} \quad (47.16)$$

Let us divide the “sum” in two parts:

$$\bigvee_{y \in A} \iota_A(\mathbf{x}^*, \mathbf{y}) \wedge \pi_A(\mathbf{y}^*, \mathbf{z}) \quad \vee \quad \bigvee_{y \in B} \iota_A(\mathbf{x}^*, \mathbf{y}) \wedge \pi_A(\mathbf{y}^*, \mathbf{z}). \quad (47.17)$$

For $y \in B$, $\iota_A(\mathbf{x}^*, \mathbf{y})$ and $\pi_A(\mathbf{y}^*, \mathbf{z})$ are both false, and we can therefore ignore the second sum. From the definition of ι_A we have that for $y \in A$, $\iota_A(\mathbf{x}^*, \mathbf{y}) = \mathbf{x} \leq_A \mathbf{y}$, and from the definition of π_A we have that for $y \in A$, $\pi_A(\mathbf{y}^*, \mathbf{z}) = \mathbf{y} \leq_A \mathbf{z}$. Thus we compute:

$$\begin{aligned} (\iota_A \circ \pi_A)(\mathbf{x}^*, \mathbf{z}) &= \bigvee_{y \in A} \iota_A(\mathbf{x}^*, \mathbf{y}) \wedge \pi_A(\mathbf{y}^*, \mathbf{z}) \\ &= \bigvee_{y \in A} (\mathbf{x} \leq_A \mathbf{y}) \wedge (\mathbf{y} \leq_A \mathbf{z}) \\ &= \mathbf{x} \leq_A \mathbf{z} \\ &= \text{id}_A(\mathbf{x}^*, \mathbf{z}) \end{aligned} \quad (47.18)$$

□

47.2. Relationship between intersection and monoidal product

To define the precise relationship between the monoidal product $f \otimes g$ (Definition 25.15) and the intersection $f \wedge g$, we first define two operations, **split** : $\mathbf{A} \leftrightarrow \mathbf{A} \times \mathbf{A}$ and **fuse** : $\mathbf{A} \times \mathbf{A} \leftrightarrow \mathbf{A}$, which correspond to splitting and fusing wires in a diagram:

$$\begin{aligned} \text{split} : \mathbf{A}^{\text{op}} \times (\mathbf{A} \times \mathbf{A}) &\rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \langle \mathbf{x}^*, \langle \mathbf{y}, \mathbf{z} \rangle \rangle &\mapsto (\mathbf{x} \leq_A \mathbf{y}) \wedge (\mathbf{x} \leq_A \mathbf{z}) \end{aligned} \quad (47.19)$$

$$\begin{aligned} \text{fuse} : (\mathbf{A} \times \mathbf{A})^{\text{op}} \times \mathbf{A} &\rightarrow_{\mathbf{Pos}} \mathbf{Bool} \\ \langle \langle \mathbf{x}^*, \mathbf{y}^* \rangle, \mathbf{z} \rangle &\mapsto (\mathbf{x} \leq_A \mathbf{z}) \wedge (\mathbf{y} \leq_A \mathbf{z}). \end{aligned} \quad (47.20)$$

Lemma 47.15. For $f, g : \mathbf{A} \leftrightarrow \mathbf{B}$,

$$f \wedge g = \text{split} \circ (f \otimes g) \circ \text{fuse}, \quad (47.21)$$

as shown in Fig. 47.5.

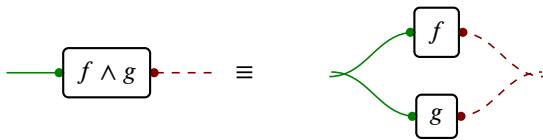


Figure 47.5.: $f \wedge g = \text{split} ; (f \otimes g) ; \text{fuse}$.

Proof. Recall that

$$(f \otimes g)((\mathbf{a}_1, \mathbf{a}_2)^*, (\mathbf{b}_1, \mathbf{b}_2)) = f(\mathbf{a}_1^*, \mathbf{b}_1) \wedge g(\mathbf{a}_2^*, \mathbf{b}_2). \quad (47.22)$$

We have

$$\begin{aligned} & ((f \otimes g) ; \text{fuse})((\mathbf{a}_1, \mathbf{a}_2)^*, \mathbf{b}) \\ &= \bigvee_{\langle b', b'' \rangle \in B \times B} (f(\mathbf{a}_1^*, \mathbf{b}') \wedge g(\mathbf{a}_2^*, \mathbf{b}'')) \wedge ((\mathbf{b}' \leq \mathbf{b}) \wedge (\mathbf{b}'' \leq \mathbf{b})) \\ &= f(\mathbf{a}_1^*, \mathbf{b}) \wedge g(\mathbf{a}_2^*, \mathbf{b}). \end{aligned} \quad (47.23)$$

Thus, we have

$$\begin{aligned} & (\text{split} ; f \otimes g ; \text{fuse})(\mathbf{a}^*, \mathbf{b}) \\ &= \bigvee_{\langle a', a'' \rangle \in A \times A} \text{split}(\mathbf{a}^*, \langle \mathbf{a}', \mathbf{a}'' \rangle) \wedge (f(\mathbf{a}'^*, \mathbf{b}) \wedge g(\mathbf{a}''^*, \mathbf{b})) \\ &= \bigvee_{\langle a', a'' \rangle \in A \times A} (\mathbf{a}^* \leq \mathbf{a}') \wedge (\mathbf{a}^* \leq \mathbf{a}'') \wedge f(\mathbf{a}'^*, \mathbf{b}) \wedge g(\mathbf{a}''^*, \mathbf{b}) \\ &= f(\mathbf{a}^*, \mathbf{b}) \wedge g(\mathbf{a}^*, \mathbf{b}) \\ &= (f \wedge g)(\mathbf{a}^*, \mathbf{b}). \end{aligned} \quad (47.24)$$

□

Is “ $X \times Y$ ” the same as “ $Y \times X$ ”? It depends on the context. Intuitively, for the categories of resources treated in this work, we would not make a distinction between “having X and Y ” and “having Y and X ”. However, there are contexts in which this is not valid. For example, if we are using $X \times Y$ to mean that we will have the resource X today, and the resource Y tomorrow, then $Y \times X$ would not be the same as $X \times Y$.

In the contexts in which this symmetry holds, we call the category “symmetric monoidal”: we will talk about this more in detail later in the book.

The word “symmetric” is well-suited, because to say that the two objects are equivalent, or “isomorphic”, we can postulate that we always have a way to get “ $X \times Y$ ” starting from “ $Y \times X$ ” and viceversa. Diagrammatically, this is depicted by the two arrows that connect them (Fig. 47.6).

JL: This opening bit seems confusing/misleading to me. I can edit/correct later; also I would move it to the section on monoidal products.



Figure 47.6.: Objects equivalence in the symmetric case.

AC: yes, move to the part on monoidal products, when talking about symmetry

48. Computation

Design problems are relations between different posets, and they answer the question, “can I provide a given functionality f with resources r ?”.

In engineering design, we are interested not only in whether a certain functionality can be provided with a given resource, but also in the *minimal* amount of resources with which we can provide that functionality. That is, given a required functionality, what are the *minimal* sets of resources which provide it?

Let us define this query more precisely. For any design problem d , we can define a monotone function $\textcolor{blue}{h}_d$ that sends a functionality $f \in F$ to the *minimum* antichain of resources which provide f :

$$\begin{aligned} \textcolor{blue}{h}_d : F &\rightarrow \mathcal{A}R, \\ f &\mapsto \text{Min}\{r \in R : d(f^*, r) = \top\}. \end{aligned} \tag{48.1}$$

Definition 48.1. A design problem is *computable* if an exact solution to $\textcolor{blue}{h}_d$ can be computed in time polynomial in the computation time of its component design problems. If a design problem has no component design problems, then we say that it is *primitive*.

In this section, we assume that all primitive design problems are computable, that is, $\textcolor{blue}{h}_d$ terminates in finite time.

Is this map computable? In this section we will show that the answer is "no" in general, and we will find sufficient conditions for the answer to be "yes". The fact that **DP** is compact closed, which we proved in Lemma 37.6, will help us describe and solve these optimization problems.

48.0.1. Preliminaries on antichains

We now use the antichains machinery developed in Section 11.3. For P a poset, $\mathcal{U}P$ denotes the set of upper sets, and $\mathcal{A}P$ the set of antichains.

We will see $\mathcal{A}P$ itself as a poset, with the order given by

$$S \leq_{\mathcal{A}P} T \quad \equiv \quad \uparrow S \supset \uparrow T,$$

where \uparrow is the upper closure of the set (Definition 11.17).

The top element of the poset $\mathcal{A}P$ is the empty set \emptyset .

When we talk about computing minimal sets of resources, we always mean computing an appropriate antichain, i.e. an element in $\mathcal{A}P$.

Sufficient conditions for bijection between DPs and antichains

We *almost* have a bijection between design problems and antichains.

The natural bijection would be

$$\begin{aligned} \text{Hom}_{\mathbf{DP}}(F, R) &\xleftrightarrow{?} \mathcal{A}(F^{\text{op}} \times R), \\ d &\mapsto \text{Min } K_d, \end{aligned} \tag{48.2}$$

where K_d is the feasible set of d , and the map Min returns the minimal elements of the subset (Definition 11.31).

The problem is that not all upper sets of P are representable by their minimal elements. For example, take the two distinct upper sets $S_1 = \emptyset$ and $S_2 = \{x \in \mathbb{R} : x > 0\}$. We have $\text{Min } S_1 = \text{Min } S_2 = \emptyset$.

Suppose we restrict to only subsets which are *downward-closed*, i.e. $\text{Min } S = \emptyset \Rightarrow S = \emptyset$.

AC:

I am not sure the above is what we want. This is more explicit:

Suppose that we restrict to the upper sets that are downward-closed, in the sense that

$$S = \uparrow \text{Min } S. \tag{48.3}$$

Are these equivalent? In any case I put that definition now as Definition 11.34.

JT: They're not equivalent but on upper sets they're very close. Assume my definition, and let $\text{Min } S = \infty$, then $S = \infty$, which does not satisfy $S = \uparrow \text{Min } S$. On the other hand, suppose $\text{Min } S = K \neq \infty$ for some set K . Then $S \neq \infty$ trivially, and it remains to determine $\uparrow K$. But since S is an upper set, every $s \in S$ satisfies $s \geq k$ for some $k \in K$, so $s \in \uparrow K$; on the other hand this is the same definition of a typical element $k \in \uparrow K$.

So basically, with respect to all the proofs below (which involve only upper sets and exclude the case ∞) both definitions work fine. Mine feels a bit weaker (though neither implies the other due to how they behave on ∞) but yours is maybe more clear. So I'm happy to use yours.

We say that a design problem $d : F \leftrightarrow R$ is *downward-closed* if $K_d \subset F^{\text{op}} \times R$ is downward-closed.

Lemma 48.2. There exists a bijection between the set $\text{Hom}_{\mathbf{DP}}(F, R)$ of downward-closed design problems $d : F \leftrightarrow R$ and the set $\mathcal{A}(F^{\text{op}} \times R)$ of antichains of $F^{\text{op}} \times R$,

$$\begin{aligned} \text{Hom}_{\underline{\mathbf{DP}}}(F, R) &\leftrightarrow \mathcal{A}(F^{\text{op}} \times R) \\ d &\mapsto \text{Min } K_d. \end{aligned} \tag{48.4}$$

Proof. Every d can be represented by its feasible set K_d , so $\text{Hom}_{\mathbf{DP}}(F, R) \simeq \mathcal{U}(F^{\text{op}} \times R)$. In effect, the proof comes down to the fact that antichains are compact representations of upper sets which are downward-closed. We will show that, for any poset P , there is a bijection between the antichains of P and the set of upper sets of P which are downward-closed.

Pick $A \in \mathcal{A}P$. Then A generates an upper set $\uparrow A = \{x \in P : a \leq x \text{ for some } a \in A\}$, and $\text{Min } \uparrow A = A$. From the other direction, consider $\mathcal{U}P$, the set of upper sets of P which are downward-closed. Pick a non-empty $U \in \mathcal{U}P$. Since U is downward-closed, $\text{Min } U$ is a non-empty antichain given by $\{a \in \underline{U} : a \leq x \text{ for all } x \in \underline{U}\}$,

and $\uparrow \text{Min } \underline{U} = \underline{U}$. □

Remark 48.3. Another way of looking at the bijection: a poset P considered with respect to only those upper sets which are downward-closed is an approximation of a well-founded poset, a.k.a. a poset which satisfies the descending chain condition. In such posets, there is a well-known bijection between upper sets and antichains (assuming the axiom of dependent choice).

AC: I don't understand what "approximation" means here. JT: I wrote approximation because restricting attention to just those downward-closed upper sets was less strong than just assuming DCC on the whole poset (I think of it as focusing attention on parts of the poset while leaving the funky other parts blurred out), but allowed you to prove a similar bijection. But I'm not familiar with the arguments in terms of trees that Hamkin brought up, I'll think about this later.

AC: By the way, when I asked, I was told that you need Choice to make the bijection work. See this discussion. JT: I added the mention of choice in the remark. For the main proof see above comment.

Why does this bijection matter?

Looking at the definition of h_d :

$$\begin{aligned} h_d : F &\rightarrow \mathcal{AR} \\ f &\mapsto \text{Min } \{r \in R : d(f^*, r) = \top\} \end{aligned} \tag{48.5}$$

Assuming $K_d \neq \emptyset$, one immediately observes that an exact solution exists if and only if d is downward-closed. So the bijection tells us how to restrict to design problems which have exact solutions.

Computability of design problems

A *primitive* design problem is any design problem which is not constructed using any of the operators we have already defined. A primitive design problem d is computable when h_d is computable in polynomial time.

AC: for the above: what is a "primitive" design problem? JT: See above.

AC: And what does it mean "almost"? JT: See above.

AC: "querying" a lookup table might take infinitely long if such lookup table is infinite... JT: Fair enough. I've changed the characterization.

Every combination of computable design problems we have so far looked at—excepting loop—can be easily seen to be computable:

Proposition 48.4. The composition, monoidal product, and coproduct of computable design problems are also computable.

Proof. These are direct properties of **DP**.

AC: This is not really a proof, it is just a statement of facts without proofs. JT: Hmm I thought I was copying a proposition in your original paper but now that I go back I can't find it. In that case I will need to think about this a bit more. The issues are the Min in the composition and the coproduct in the cases with an infinite antichain.

Composition. Suppose $d_1 : A \leftrightarrow B$ and $d_2 : B \leftrightarrow C$. Then $h_{d_1 \circ d_2}$ is defined by

$$\begin{aligned} h_{d_1 \circ d_2} : A &\rightarrow AC \\ a &\mapsto \text{Min} \bigcup_{b \in h_{d_1}(a)} h_{d_2}(b). \end{aligned} \tag{48.6}$$

Clearly this is computable if h_{d_1}, h_{d_2} are computable.

AC: Is it obvious that Min itself is computable? JT: See above.

Monoidal product. Similarly with the monoidal product: if $d_1 : A \leftrightarrow C$ and $d_2 : B \leftrightarrow D$, then $h_{d_1 \otimes d_2}$ is defined by

$$\begin{aligned} h_{d_1 \otimes d_2} : (A \times B) &\rightarrow \mathcal{A}(C \times D) \\ \langle a, b \rangle &\mapsto h_{d_1}(a) \times h_{d_2}(b) \end{aligned} \tag{48.7}$$

Coproduct. Similarly with the coproduct, for $d_1, d_2 : A \leftrightarrow B$ below:

$$\begin{aligned} h_{d_1 + d_2} : A &\rightarrow AB \\ \langle a, b \rangle &\mapsto \text{Min}(h_{d_1}(a) \cup h_{d_2}(b)). \end{aligned} \tag{48.8}$$

AC: This last one should be $d_1 + d_2$, not $d_1 \sqcup d_2$, right? JT: I think so; I'm not sure what the most recent notation is. \square

Loops are significantly harder. To see that we can compute them in finite time, we will need the following fact:

Lemma 48.5. If a design problem $d : F \leftrightarrow R$ is downward-closed, then h_d is Scott-continuous.

Proof. Recall that a monotone function is *Scott-continuous* if it preserves all directed suprema, i.e. if D is a directed subset and D^\uparrow is its supremum, then $f(D^\uparrow) = f(D)^\uparrow$. d Scott-continuous just means that the supremum of any feasible, directed subset must also be feasible. If d is downward-closed, then d is Scott-continuous.

AC: I am confused. In "d Scott-continuous just means...", what does it mean for a DP to be Scott-continuous? Scott-continuous is a property of a map. JT: Recall that d is defined to be a monotone map from a given poset $F^{\text{op}} \times R$ to Bool . The sentence above is a direct translation of what it means for $d(D^\uparrow) = d(D)^\uparrow$, where $D \subset F^{\text{op}} \times R$.

If d is Scott-continuous, then h_d is Scott-continuous, since h_d only observes the

lower boundary of the feasible set, which is controlled by the downward-closed condition. \square

For a design problem of the form $d : (B \times A) \leftrightarrow (B \times C)$, $\textcolor{blue}{h}_d : B \times A \rightarrow \mathcal{A}(B \times C)$ is defined as normal. For any input a we define a recursion operator $\psi_{a,d}$ for $\textcolor{blue}{h}_d$ by

$$\begin{aligned}\psi_{a,d} : \mathcal{A}(B \times C) &\rightarrow \mathcal{A}(B \times C) \\ S &\mapsto \text{Min } \bigcup_{\langle b,c \rangle \in S} \{\langle b',c' \rangle \in \textcolor{blue}{h}_d(b,a) : b' \leq b\}.\end{aligned}\quad (48.9)$$

In that case, $h_{\text{Tr } d}$ is defined by

$$\begin{aligned}h_{\text{Tr } d} : A &\rightarrow \mathcal{AC} \\ a &\mapsto \text{Min } \pi_C(\text{lfp}(\psi_{a,d})).\end{aligned}\quad (48.10)$$

where $\pi_C : B \times C \rightarrow C$ is the projection on C .

AC: TODO: define π_C (projection on C) **JT:** Done.

AC: Where does lfp come from? I think there are many passages omitted here. **JT:** I assume they know what a lfp is, or can look it up.

AC: Also, this formula is not equivalent to what I had found; see below for more comments.

JT: The main thing that is different is the fact that we're carrying around an extra variable's worth of data, namely C .

To say that $\text{Tr } d$ is computable is to say that the least fixed point $\text{lfp}(\psi_{a,d})$ exists.

What's going on here?

1. First, calculate $\textcolor{blue}{h}_d$ for some fixed, initial $\langle b_{\text{in}}, a \rangle$. The goal is to compute the antichain of minimal elements $c \in C$ which provide a , across all possible choices of b_{in} .
2. Throw away all those minimal elements in $\textcolor{blue}{h}_d(b_{\text{in}}, a)$ that do not satisfy the trace constraint, $b_{\text{in}} \leq b_{\text{out}}$,¹ where $b_{\text{out}} \in \pi_B(\textcolor{blue}{h}_d(b_{\text{in}}, a))$.
3. At this point, we are left with an antichain $S \subset \mathcal{A}(B \times C)$ of minimal elements $\langle b_{\text{out}}, c \rangle$ that satisfy the trace constraint, but the resources $c \in S|_C$ (which do not necessarily form an antichain) are not necessarily minimal with respect to the functionality a , since we might have chosen some other initial b_{in} and gotten better results. Alternately, if we had started with the wrong b_{in} , the entire computation might have terminated with \emptyset .
4. Take the list of b_{out} from Step 2 as the new b_{in} , and run through steps 1–2 in parallel for each new b_{in} . Take the union of the results, and take the Min of the union to obtain a new list of b_{out} . Each result will be a smaller antichain in C , i.e. a “better” result.
5. Keep iterating Step 4 until the result converges to a least fixed point: the antichain of minimal elements $\langle b, c \rangle \in B \times C$ which provide $\langle b, a \rangle$, across all possible choices of b_{in} .
6. Lastly, given an antichain in $\mathcal{A}(B \times C)$, we need to convert it into an antichain \mathcal{AC} . The obvious way to do this is to project the antichain to C and take the minimal elements of the projection.

¹If the loop is composed with an addition behind the input, which is often the case, then this condition would change to reflect the addition.

JT: Something about this last step feels wrong. I feel that $\mathcal{A}(B \times C)$ is more important than $\mathcal{A}C$. Wouldn't you prefer to retain the information there, rather than throw it all away? Projecting to the minimal elements of the projection would also distinguish a subset of the “stable” b , namely those in the pre-image under the projection of those minimal elements (pullback?).

AC: In fact, I don't think that Eq. (48.10) is correct. There are some gaps in the derivation. Try to fill those gaps and see if the result changes? JT: I'll look at it again. If you think of something specific, could you help me by pointing it out? In reference to “magic equation”, see comment above.

To ensure that the least fixed point exists, we require that the posets B, C (thus $\mathcal{A}(B \times C)$) be a *directed complete partial order* or *dcpo*, i.e. that there exists a supremum for any directed subset $S \subset \mathcal{A}(B \times C)$. To ensure that we do not miss any solutions by starting with the wrong b_{in} , we also require that B be *pointed*, i.e. that it possess a unique bottom element.

Lemma 48.6. If \mathbf{h}_d is Scott-continuous, then so is $\psi_{a,d}$.

Proof. The statement is Exercise 8.26 in Davey and Priestly [cite]. □

Proposition 48.7. If d is downward-closed and $\mathcal{A}(B \times C)$ is a pointed dcpo, then $\text{Tr } d$ is computable.

Proof. The proposition is a direct application of Adámek's theorem for algebras over an endofunctor.

Recall the statement of Adámek's theorem for algebras over an endofunctor: let \mathcal{C} be a category with an initial object 0 and transfinite composition of length ω , hence colimits of sequences $\omega \rightarrow \mathcal{C}$ (where ω is the first infinite ordinal), and suppose $F : \mathcal{C} \rightarrow \mathcal{C}$ preserves colimits of ω -chains. Then the colimit γ of the chain

$$0 \xrightarrow{i} F(0) \xrightarrow{F(i)} \dots \xrightarrow{F^{(n)}(0)} F^{(n+1)}(0) \rightarrow \dots$$

carries the structure of an initial F -algebra.

The category \mathcal{C} is the poset $\mathcal{A}(B \times C)$, whose initial object is just the bottom element \perp . The colimit of a (possibly infinite) sequence or chain of elements in $\mathcal{A}(B \times C)$ is simply the supremum of that chain. The endofunctor F is given by the recursion operator $\psi_{a,d} : \mathcal{A}(B \times C) \rightarrow \mathcal{A}(B \times C)$, and by the previous lemmas, $\psi_{a,d}$ is Scott-continuous, thus preserves suprema. Finally, under repeated applications of $\psi_{a,d}$, the colimit γ (now defined in **Pos** as opposed to $\mathcal{A}B$) of that sequence of monotone maps is the fixed point of $\psi_{a,d}$; the fact that it is an initial F -algebra—i.e. an element S in the poset $\mathcal{A}(B \times C)$ that is “initial”, i.e. $S \leq T$ for all $T \in \mathcal{A}B$ —means that γ is the least fixed point. □

AC: I'm trusting you on the above.

Directed complete partial orders and Scott-continuous functions define their own category of design problems, \mathbf{DP}_S . We can think of \mathbf{DP}_S as the subcategory of \mathbf{DP} whose morphisms preserve all sequential colimits.

Duality

We can also define the dual problem; suppose we fix a given resource $r \in R$ and ask for the *maximal antichain* of functionalities that can be provided with r . Then we can define a function

$$\begin{aligned} g_d : R &\rightarrow \mathcal{AF}^{\text{op}} \\ r &\mapsto \text{Min } \{f^* \in F^{\text{op}} : d(f^*, r) = \top\} \end{aligned} \tag{48.11}$$

with analogous properties to $\textcolor{blue}{h}_d$.

AC: Do we know if $\textcolor{blue}{h}_d$ computable implies g_d computable? **JT:** I have not checked and am fine with removing this comment in the interests of time.

To put back somewhere

Note that $\textcolor{blue}{h}_d$ is a monotone function but is *not* a design problem. Given that it is a monotone function, what happens if we try to turn it into a design problem? The conjoint of $\textcolor{blue}{h}_d$ is given by

$$\begin{aligned} \hat{\textcolor{blue}{h}}_d : F &\leftrightarrow \mathcal{AR} \simeq \text{Hom}_{\mathbf{DP}}(*, R) \\ \langle f^*, S \rangle &\mapsto \mathcal{AR}(\textcolor{blue}{h}_d(f), S) \leftrightarrow \bigwedge_{s \in S} d(f^*, s) \end{aligned}$$

In other words, the conjoint of $\textcolor{blue}{h}_d$ is an indicator function for subsets of feasible resources.

Contents

9. Sameness	83
9.1. Sameness in category theory	85
9.2. Isomorphism is not identity	88
10. Universal properties	89
10.1. Universal properties	91
11. Trade-offs	93
11.1. Trade-offs	95
11.2. Ordered sets	96
11.3. Chains and Antichains	99
11.4. Upper and lower sets	100
11.5. From antichains to uppersets, and viceversa	101
11.6. Lattices	105
12. Poset constructions	113
12.1. Opposite of a poset	117
12.2. Poset of intervals	118
12.3. A different poset of intervals	118
13. Life is hard	119
13.1. Monotone maps	121
13.2. Compositionality of monotonicity	124
13.3. The category Pos of posets and monotone maps	124
13.4. Why Pos is not sufficient for design theory	125
14. Functors	127
14.1. Functors	129
14.2. A poset as a category	129
14.3. Other examples of functors	131
15. Up the ladder	133
15.1. Functor composition	135
15.2. A category of categories	135
15.3. Full and faithful functors	136
15.4. Forgetful functor	137
16. Duality	139
16.1. Galois connections	141
17. Naturality	147
17.1. Natural transformations	149
18. Adjunctions	153
18.1. An example	155
18.2. Adjunctions: hom-set definition	155
18.3. Adjunctions: (co)unit definition	155
18.4. Example of a “Product-Hom” adjunction	157
18.5. Example of a “Free-Forgetful” adjunction	159
18.6. Relating the two definitions	160
19. Combination	163
19.1. Products	165

19.2. Coproduct	172
19.3. Other examples	180
20. Counter-examples	183
20.1. Not quite categories	185
20.2. Not quite functors	185
B. Monotone Co-Design	187
21. Design	189
21.1. What is “design”?	191
21.2. What is “co-design”?	191
21.3. Basic concepts of formal engineering design	193
21.4. Queries in design	196
22. Design problems	197
22.1. Design Problems	199
22.2. Querying a DPI	210
22.3. Co-design problems	211
22.4. Discussion of related work	214
23. Feasibility	219
23.1. Design problems as monotone maps	221
23.2. Series composition of design problems	223
23.3. The category of design problems	229
23.4. DP Isomorphisms	230
24. Profunctors	231
24.1. Profunctors	233
24.2. Hom Profunctor	233
24.3. Other examples of profunctors	233
24.4. The bicategory of profunctors	233
24.5. DPI as profunctors	233
25. Parallelism	235
25.1. Monoids	237
25.2. Monoids homomorphism	238
25.3. Dynamical systems and monoids	238
25.4. Monoidal posets	239
25.5. Monoidal categories	240
25.6. DP is a monoidal category	246
26. Feedback	255
26.1. Trace of a linear transformation	257
26.2. Continuous LTI	257
26.3. Feedback in category theory	258
26.4. Feedback in design problems	259
27. Ordering design problems	267
27.1. Ordering DPs	269
27.2. Restrictions and alternatives	270
27.3. Interaction with composition	271

Contents

28. Constructing design problems	275
28.1. Creating design problems from catalogues	277
28.2. Companions and conjoint	280
28.3. Sum and intersection with companion and conjoints	282
28.4. Monoidal DPs	285
C. Computation	287
29. From math to implementation	289
30. Solving	293
30.1. Types of queries	295
30.2. Computational representation of DPs	295
30.3. Problem statement and summary of results	296
30.4. Composition operators for design problems	298
30.5. Decomposition of MCDPs	301
30.6. Monotonicity as compositional property	303
30.7. Solution of MCDPs	309
30.8. Example: Optimizing over the natural numbers	310
30.9. Complexity of the solution	314
30.10. Extended Numerical Examples	316
30.11. Monotonicity and fixed points	323
31. Uncertainty	327
31.1. Monads	329
31.2. Using monads to understand uncertainty	330
31.3. L and U monads	331
D. Higher-order theory	337
32. Enrichments	339
32.1. Enrichments	341
32.2. Enriched categories	342
32.3. Set-enriched DPs (DPIs)	344
33. Negative designs	351
E. Operadic structures	353
34. Wiring diagrams	355
35. Operads	357
36. Recursion	359
36.1. Recursive Design Problems	359
37. Higher-order design	361
37.1. Diagrams	363
37.2. Diagram as a monotone function	363
37.3. Representation Results	363

37.4. Compact closed structure	363
37.5. A locally-posetal pro-arrow equipment	368
F. Networks and systems	371
38. Decorated co-spans	373
G. Extensions of co-design theory	375
39. Linear logic	377
40. Linear DPs	379
41. Temporal DPs	381
H. Control theory in category theory	383
I. Case study: co-design of AV fleets	387
42. Co-design of control systems	389
43. Co-design of autonomous systems	391
44. Co-design of mobility systems	393
J. To move	395
45. Paper to dismember and move	397
45.1. Introduction	397
45.2. Conclusions	400
46. Other paper	403
46.1. Introduction	403
46.2. Design Problems	406
46.3. Monotone Co-Design Problems	407
46.4. Semantics of MCDPs	410
46.5. Solution of MCDPs	411
46.6. Uncertain Design Problems	412
46.7. Partial order \leq_{DP}	413
46.8. Uncertain DPs (UDPs)	413
46.9. Order on UDP	414
46.10 DPs as degenerate UDPs	414
46.11 Interconnection of Uncertain Design Problems	415
46.12 Approximation results	416
46.13 Applications	418
46.14 Application: Dealing with Parametric Uncertainty	418
46.15 Application: Introducing Tolerances	419
46.16 Application: Relaxation for relations with infinite cardinality	422

Contents

46.17	Conclusions and future work	425
46.18	Proofs	427
46.19	Software	433
46.20	Source code	433
46.21	Virtual machine	433
47.	Relationship between products	447
47.1.	Biproduct, Product, and Coproduct of Design Problems	447
47.2.	Relationship between intersection and monoidal product	456
48.	Computation	461

1. Introduction

to write

Contents

1.1.	Thesis	15
1.2.	Systems and components	15
1.3.	What ACT can do for you	16
1.4.	What ACT cannot do	16
1.5.	Book organization	16
1.6.	Contributors	17



1. Introduction

1.1. Thesis

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.

1.2. Systems and components

What is a “system”?

Here is a great quote¹:

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

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.

Haiken 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” [leveson12engineering].

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 (Fig. 1.1). 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.

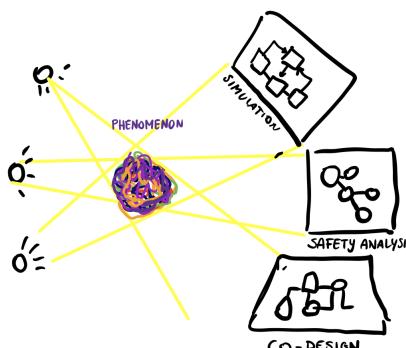


Figure 1.1.: Engineers live in a Plato’s cave with multiple light sources. A certain phenomenon can be illuminated from different angles and look very different, very simple or complex.

Make better figure

¹This 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 in [McIntyre1999Role], but ultimately sourceless and probably apocryphal.

1.3. What ACT can do for you

describe how ACT can help

1.4. What ACT cannot do

describe limitations of ACT

1.5. Book organization

To write.

1.6. Contributors

To write.

1. Introduction

Part A.

A first look at Applied Category Theory

2. Overview

to write

Contents

2.1. Everything is the same	23
2.2. Guidebook	23



2. Overview

2.1. Everything is the same

Reproduce here the discussion from the first lecture, about many different things that are actually the same.

2.2. Guidebook

Broader overview of the abstractions that we will develop in this book, including posets, lattices, etc. up to operad.

3. Transmutation

to write

Contents

3.1. Interfaces and transformations	27
3.2. Formal definition of category	30
3.3. Currency categories	33



3. Transmutation

3.1. Interfaces and transformations

Gives more examples before getting to the definition. Give more examples with different conventions for the arrow direction.

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 chain 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 “transmuters”: the **motor** and **wheels**.
2. The “transmuted”: the **electricity**, the **rotation**, the **translation**.

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. 3.1).



Figure 3.1.: 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

$$X \xrightarrow{\text{transmuter}} Y \tag{3.1}$$

means that “using transmuter, having **Y** is sufficient to produce **X**”.

Remark 3.1 (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 Eq. (3.1) would be as follows:

$$\text{transmuter} : X \rightarrow Y. \tag{3.2}$$

This is now to you something syntactically familiar; when we study the categories of sets and functions between sets we will see that in that context the familiar meaning is also the correct meaning.

With these conventions, we can describe the two transmuters as these arrows:

$$\text{motor} : \text{rotation} \rightarrow \text{electricity}, \tag{3.3}$$

$$\text{wheels} : \text{translation} \rightarrow \text{rotation}. \tag{3.4}$$

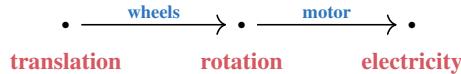


Figure 3.2.: Transmuters are arrows between resources.

We can put these arrows in the diagrams, and obtain the following (Fig. 3.2).

In this representation, the arrows are the components of the system. We will learn how to compose these arrows according to the rules of category theory. The basic rule will be *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. 3.3).

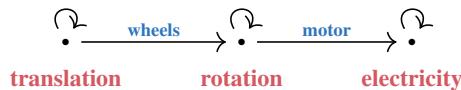


Figure 3.3.: System components and identities.

Furthermore, we might consider the idea of composition of arrows. Suppose that we know that

$$X \xrightarrow{a} Y \quad \text{and} \quad Y \xrightarrow{b} Z,$$

that is, using a b we can get a Z from a Y , and using an a we can get a X from a Y , then we conclude that using an a and a b 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. 3.4).

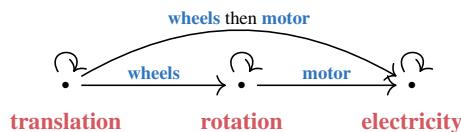


Figure 3.4.: Composition of system components.

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. 3.5 shows two models of motors (**motor A**, and **motor B**) and two models of wheels (**wheels U** and **wheels V**).

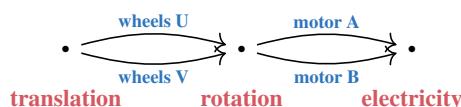


Figure 3.5.: Multiple models for wheels and motors.

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**;

3. Transmutation

A “category” is an abstract mathematical structure that captures the properties of these systems of points and arrows and the logic of composition.

The basic terminology is that the points are called **objects**, and the arrows are called **morphisms**.

In our example, the **motor** and the **wheels** are the morphisms, and **electricity**, **rotation**, **translation** are the objects.

Many things can be defined as categories and we will see many examples in this book.

We are now just biding our time before introducing the formal definition of category. At first sight it will be intimidating: there are four parts to the definition, two axioms to define. Moreover, it is quite a bit technical and it takes half a page to write.

3.2. Formal definition of category

The following is the formal definition.

Definition 3.2 (Category). A *category* **C** is specified by four components:

1. **Objects**: a collection¹ Ob_C , whose elements are called *objects*.
2. **Morphisms**: for every pair of objects $X, Y \in \text{Ob}_C$, there is a set $\text{Hom}_C(X, Y)$, elements of which are called *morphisms* from X to Y . The set is called the “hom-set from X to Y ”.
3. **Identity morphisms**: for each object X , there is an element $\text{id}_X \in \text{Hom}_C(X, X)$ which is called the *identity morphism of X* .
4. **Composition rules**: given any morphism $f \in \text{Hom}_C(X, Y)$ and any morphism $g \in \text{Hom}_C(Y, Z)$, there exists a morphism $f \circ g \in \text{Hom}_C(X, Z)$ which is the *composition of f and g* .

Furthermore, the constituents are required to satisfy the following conditions:

1. **Unitality**: for any morphism $f \in \text{Hom}_C(X, Y)$,

$$\text{id}_X \circ f = f = f \circ \text{id}_Y. \quad (3.5)$$

2. **Associativity**: for $f \in \text{Hom}_C(X, Y)$, $g \in \text{Hom}_C(Y, Z)$, and $h \in \text{Hom}_C(Z, W)$,

$$(f \circ g) \circ h = f \circ (g \circ h). \quad (3.6)$$

Remark 3.3 (Are we sure we are not going in the wrong direction?). We denote composition of morphisms in a somewhat unusual way—sometimes preferred by category-theorists and computer scientists—namely in *diagrammatic order*.

That is, given $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, we denote their composite by $(f \circ g) : X \rightarrow Z$, pronounced “ f then g ”. This is in contrast to the more typical notation for composition, namely $g \circ f$, or simply gf , which reads as “ g after f ”. The notation $f \circ g$ is sometimes called *infix notation*.

We promise, at some point it will be clear what are the advantages of seemingly doing everything in the wrong direction.

Note that we may save some ink when drawing diagrams of morphisms:

¹A “collection” is something which may be thought of as a set, but may be “too large” to technically be a set in the formal sense. This distinction is necessary in order to avoid such issues as Russel’s paradox.

- We do not need to draw the identity arrows from one object to itself, because, by Definition 3.2, they always exist.
- Given arrows $X \rightarrow Y$ and $Y \rightarrow Z$, we do not need to draw their composition because, by Definition 3.2, 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. 3.6).

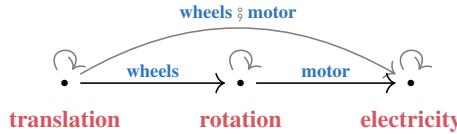


Figure 3.6.: Electric car example. The grey arrows are implied by the properties of a category.

In particular, the electric car example corresponds to the category C specified by

- *Objects:* $\text{Ob}_C = \{\text{electricity}, \text{rotation}, \text{translation}\}$.
- *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. 3.7, Fig. 3.8).

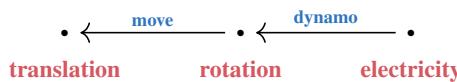


Figure 3.7.: Electric power can be produced from motion.

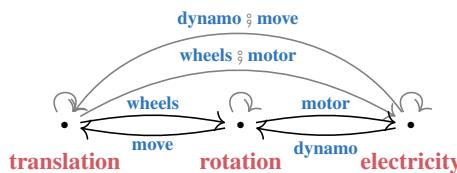


Figure 3.8.: Electric car example: forward and backward transformations.

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 **wheels ; motor ; dynamo ; move**, 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. 3.9). 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.

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. Fig. 3.10 shows an example of a diagram that describes a process which is definitely not invertible.

3. Transmutation

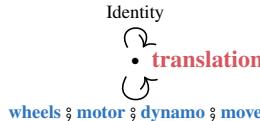


Figure 3.9.: There can be multiple morphisms from an object to itself.



Figure 3.10.: An example of a process which is not invertible.

3.3. 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, e.g., as offered by a currency exchange service.

We start with a set **C** of labels for all the currencies we wish to consider, i.e.:

$$\mathbf{C} = \{\mathbf{EUR}, \mathbf{USD}, \mathbf{CHF}, \mathbf{SGD}, \dots\}.$$

For each currency $c \in \mathbf{C}$ we define an object $\mathbb{R} \times \{c\}$ which represents possible amounts of the given currency c (we will ignore the issue of rounding currencies to an accuracy of two decimal places, and we allow negative amounts). The currency label keeps track of which “units” we are using.

Now consider two such objects, say $\mathbb{R} \times \{\mathbf{USD}\}$ and $\mathbb{R} \times \{\mathbf{EUR}\}$. 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 a and a commission b for the transaction. Given an amount $x \in \mathbb{R}$ of USD, we define a morphism (a currency exchanger) as:

$$E_{a,b} : \mathbb{R} \times \{\mathbf{USD}\} \rightarrow \mathbb{R} \times \{\mathbf{EUR}\},$$

by the formula

$$\langle x, \mathbf{USD} \rangle \mapsto \langle ax - b, \mathbf{EUR} \rangle.$$

Note that the commission is given in the units of 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 $\mathbb{R} \times \{\mathbf{USD}\}$ to $\mathbb{R} \times \{\mathbf{EUR}\}$.

To build our category, we also need to specify how currency exchangers compose. Given currencies c_1, c_2, c_3 , and given currency exchangers

$$E_{a,b} : \mathbb{R} \times \{c_1\} \rightarrow \mathbb{R} \times \{c_2\} \quad \text{and} \quad E_{a',b'} : \mathbb{R} \times \{c_2\} \rightarrow \mathbb{R} \times \{c_3\}$$

we define the composition $E_{a,b} \circ E_{a',b'}$ to be the currency exchanger

$$E_{aa',a'b+b'} : \mathbb{R} \times \{c_1\} \rightarrow \mathbb{R} \times \{c_3\}. \tag{3.7}$$

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 unit morphisms for our category. These are currency exchangers which “do nothing”. For any object $\mathbb{R} \times \{c\}$, its identity morphism is

$$E_{1,0} : \mathbb{R} \times \{c\} \rightarrow \mathbb{R} \times \{c\},$$

the currency exchanger with exchange rate “1” and commission “0”.

It is straightforward to check that the composition of currency exchangers as defined above obeys the associative law, and that the identity morphisms act neutrally for composition. Thus we indeed have a category!

Remark 3.4. In the above specification of our category of currency exchangers, we can actually just work with the set of currency labels \mathbf{C} as our objects, instead of using “amounts” of the form $\mathbb{R} \times \{\mathbf{c}\}$ as our objects. Indeed, on a mathematical level, the definition of currency exchangers and their composition law Eq. (3.7) do not depend on using amounts! Namely, a currency exchanger $E_{a,b}$ is specified by the pair of numbers $\langle a, b \rangle$, and the composition law Eq. (3.7) may then, in this notation, be written as

$$\langle a, b \rangle ; \langle a', b' \rangle = \langle a'a, a'b + b' \rangle. \quad (3.8)$$

The interpretation is still that currency exchangers change amounts of one currency to amounts in another currency, but for this we do not need to carry around copies of \mathbb{R} in our notation.

Following the above remark:

Definition 3.5 (Category \mathbf{Curr}). The *category of currencies* \mathbf{Curr} is specified by:

1. *Objects*: a collection of currencies \mathbf{C} .
2. *Morphisms*: given two currencies $\mathbf{c}_1, \mathbf{c}_2 \in \mathbf{C}$, morphisms between them are currency exchangers $\langle a, b \rangle$ from \mathbf{c}_1 to \mathbf{c}_2 .
3. *Identity morphism*: given an object $\mathbf{c} \in \mathbf{C}$, 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 Eq. (3.8).

As an illustration, consider three currency exchange companies **ExchATM**, **MoneyLah**, and **Frankurrencies**, which operate on several currencies (Table 3.1).

Company name	Exchanger label	Direction	a (exchange rate)	b (fixed commission)
ExchATM	<i>A</i>	USD to CHF	0.95 CHF/USD	2.0 CHF
ExchATM	<i>B</i>	CHF to USD	1.05 USD/CHF	1.5 USD
ExchATM	<i>C</i>	USD to SGD	1.40 SGD/USD	1.0 SGD
MoneyLah	<i>D</i>	USD to CHF	1.00 CHF/USD	1.0 CHF
MoneyLah	<i>E</i>	SGD to USD	0.72 USD/SGD	3.0 USD
Frankurrencies	<i>F</i>	EUR to CHF	1.20 CHF/EUR	0.0 CHF
Frankurrencies	<i>G</i>	CHF to EUR	1.00 EUR/CHF	1.0 EUR

Table 3.1.: Three currency exchange companies operating different currencies.

We can represent this information as a graph, where the nodes are the currencies and the edges are particular exchange operations (Fig. 3.11).

There is a currency category built from the information in Table 3.1 and the graph in Fig. 3.11. Its collection of objects is the set $\{\mathbf{EUR}, \mathbf{USD}, \mathbf{CHF}, \mathbf{SGD}\}$, 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 3.1,
- all possible compositions of the currency exchangers listed in Table 3.1.

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. 3.11, and obtaining

3. Transmutation

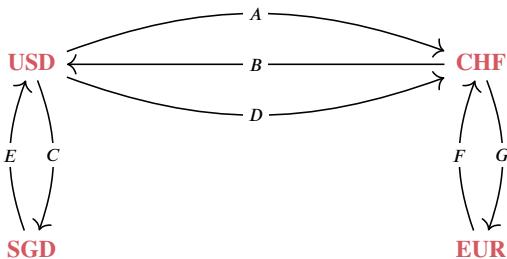


Figure 3.11.: Three currency exchange companies operating different currencies as a graph.

from it an associated category, called the *free category on G* . We introduce this concept in the next section.

Exercise 3.6 (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.

What relation is there with the currency category?

4. Culture

to write

Contents

4.1. Definitional impetus vs. computational goals	39
4.2. Things that don't matter	40



4. Culture

4.1. 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:

$$\underbrace{\langle eca}_{g}, \underbrace{ecb + ed + f}_{h} \rangle.$$

Assuming $e = a^{-1}$ (i.e., an exchange rate direction is not more profitable than the other), and $h \neq 0$, because of the commissions one 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.

4.2. 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.
- 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*.¹ 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**.

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 in an intuitive way.

4.2.1. 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 $f \circ g$ (“ 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 viceversa. For example, the formula

$$(f \circ g) \circ h = f \circ (g \circ h)$$

would be transformed in

$$h \circ (g \circ f) = (h \circ g) \circ f.$$

(A bit more advanced category theory can describe this transformation more precisely.)

The same considerations apply for the convention regarding the arrow directions. If we have a category with morphisms such as

$$\text{motor} : \text{rotation} \rightarrow \text{electricity}$$

with the semantics of “the **motor** can produce **rotation** given **electricity**”, we could define a *different* category, where the conventions are inverted. In this other category, for which we use arrows of different color, we would write

$$\text{motor} : \text{electricity} \rightarrow \text{rotation}$$

and the semantics would be “the **motor** consumes **electricity** to produce **rotation**” (Fig. 4.1).

These two categories would have the same objects, and the same number of arrows; it's just that the arrows change direction when moving from one category to the other. In particular, there exists a transformation which maps every black arrow to a blue arrow, revertin the direction (Fig. 4.2). This transformation is invertible. Intuitively, we would not expect anything substantial to change, because we are just changing a convention. We will see that there is a concept called *opposite category* that formalizes this idea of reversing the direction of the arrows.

¹IEEE Spectrum

4. Culture

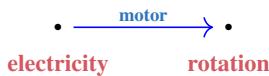
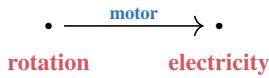


Figure 4.1.: Opposite convention for arrows direction.

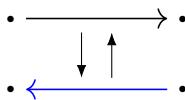


Figure 4.2.

Diagrams conventions don't matter

Now that we are flexed our isomorphism muscles, we can also talk about the isomorphisms of the visual language.

In engineering, “boxes and wires” diagrams are commonly used to talk about materials transformations and signal flows. In those diagrams one would use boxes to describe the processes and the wires to describe the materials or information that is being transformed. Boxes have “inputs” and “outputs”, and arrows have directions representing the causality. From left to right, what is to the left causes what is to the right. The left-to-right directionality seems an utterly obvious choice for most of you who learned languages that are written left-to-right, top-to-bottom as in this book².

Fig. 4.3 shows how we would have visually described the first example using the boxes-and-wires conventions. Again, we say that this is just a different convention, because we have a procedure to transform one diagram into the other. This is not as simple as changing the direction of the arrows as in the case of an opposite category. Rather, to go from points-and-arrows to boxes-and-wires:

- Arrows that describe transmutes become boxes that describe processes;
- The points that describe the resources become wires between the boxes.

Expand example as in the slides

²Note that this paragraph cannot be translated literally to Japanese. It breaks the assumption that we made before, about the fact that we can have a 1:1 literal translation of this book without changing the meaning. You might think that our future hypothetical Japanese translator can make an outstanding job and translate also our figures to go right-to-left, then saying that right-to-left is natural to people that write right-to-left. However, that does not work, because in fact Japanese engineers also use left-to-right diagrams.

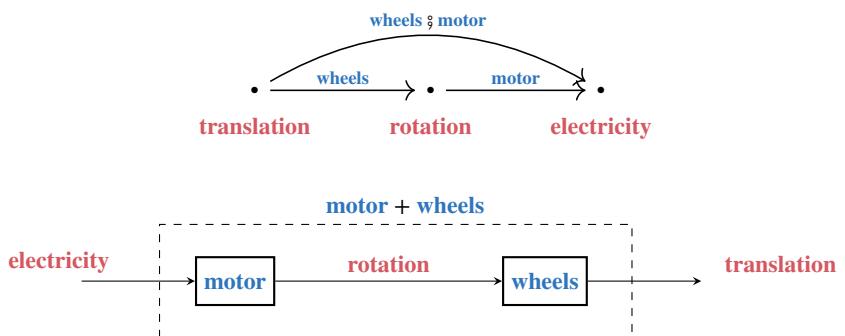


Figure 4.3.: Isomorphisms of resource diagrams.

4. Culture

5. Connection

Currency categories illustrated how one can use category theory to think about things transforming into each other. In this chapter, we want to think about how things connect to each other.

Contents

5.1. Mobility	47
5.2. Trekking in the Swiss Mountains	49
5.3. Generating categories from graphs	51



5. Connection

5.1. Mobility

For a specific mode of transportation, say a car, we can define a graph

$$G_c = \langle V_c, A_c, s_c, t_c \rangle,$$

where V_c represents geographical locations which the car can reach and A_c represents the paths it can take (e.g. roads). Similarly, we consider a graph $G_s = \langle V_s, A_s, s_s, t_s \rangle$, representing the subway system of a city, with stations V_s and subway lines going through paths A_s , and a graph $G_b = \langle V_b, A_b, s_b, t_b \rangle$, representing onboarding and offboarding 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 $G = (V, A, s, t)$ with $V = V_c \cup V_s \cup V_b$ and $A = A_c \cup A_s \cup A_b$, we obtain the desired intermodality graph. Graph G can be seen as a new category, with objects V and morphisms A .

Example 5.1. Consider the **car** category, describing your road trip in California, with

$$V_c = \{SFO_c, S. Mateo, Half Moon Bay, SBP_c, Lake Balboa, LAX_c\},$$

and arrows as in Fig. 5.1. The nodes represent typical touristic road-trip checkpoints in California and the arrows represent famous highways connecting them.

SFO_c — US101 → S. Mateo — CA92 → H. M. Bay — CA1 → SBP_c — US101S → Lake Balboa — I405 → LAX_c

Figure 5.1.: The **car** category.

Furthermore, consider the **flight** category with $V_f = \{SFO_f, SJC, SBP_f, LAX_f\}$ and arrows as in Fig. 5.2. The nodes represent airports in California and the arrows represent connections, offered by specific flight companies.

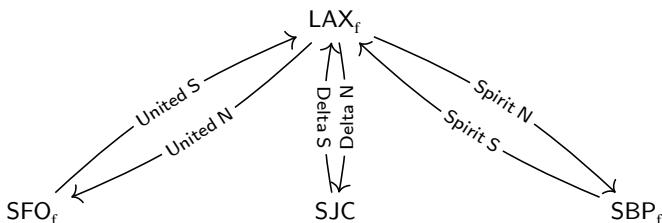


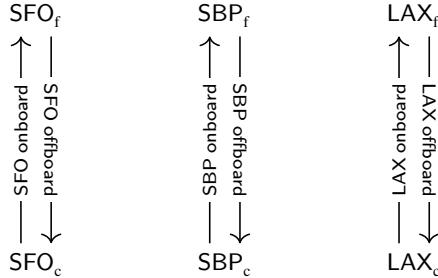
Figure 5.2.: The **flight** category.

We then consider the **board** category, with nodes

$$V_b = \{SFO_f, SFO_c, SBP_f, SBP_c, LAX_f, LAX_c\}$$

and arrows as in Fig. 5.3. Nodes represent airports and airport parkings, and arrows represent the onboarding and offboarding paths one has to walk to get from the parkings to the airport and vice-versa.

The combination of the three, which we call the *intermodal graph*, can be represented as a graph, with red arrows for the car network, blue arrows for the flight network, green arrows for the boarding


 Figure 5.3.: The **board** category.

network, and black dashed arrows for intermodal morphisms, arising from composition of morphisms involving multiple modes (Fig. 5.4). Imagine that you are in the parking lot of LAX airport and you want to reach S. Mateo. From there, you will e.g. onboard to a United flight to SFO_f , will then offboard reaching the parking lot SFO_c , and drive on highway US-101 reaching S. Mateo. This is intermodality.

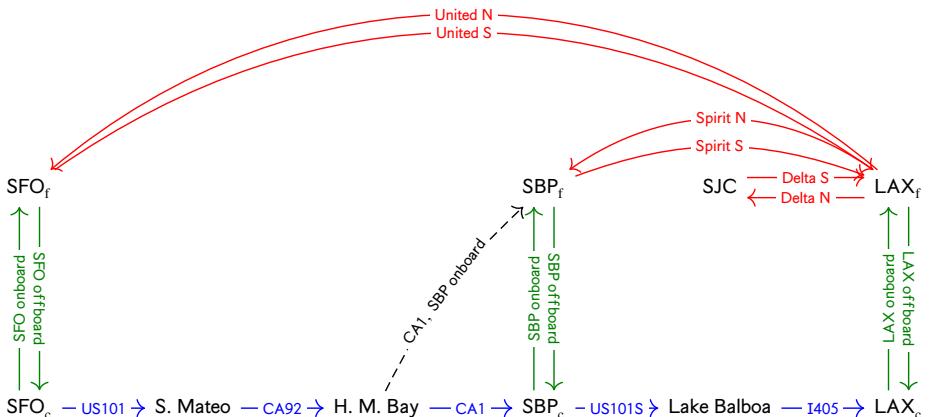


Figure 5.4.: Intermodal graph. The dashed arrows represent intermodal morphisms, and we depict just one of them for simplicity.

The intermodal network category **intermodal** is the free category on the graph illustrated in Fig. 5.4.

5.2. Trekking in the Swiss Mountains

In the section we'll 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

5. Connection

the z -th coordinate is given by an “elevation function”, $z = h(x, y)$, and that h is C^1 , i.e. continuously differentiable. This means that our map of the landscape forms a C^1 -manifold; let’s call it 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.

Definitely we need a picture of Swiss mountains

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 5.2 (Berg). Let **Berg** be the category defined as follows:

- 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).
- 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).
- For any object $\langle p, v \rangle$, we define its identity morphism $1_{\langle p, v \rangle} = \langle \gamma, 0 \rangle$ formally: its path γ is defined on the closed interval $[0, 0]$, i.e. $T = 0$ and $\gamma(0) = p$. We declare this path to be C^1 by convention, and declare its derivative at 0 to be v .
- Given morphisms $\langle \gamma_1, T_1 \rangle : \langle p_1, v_1 \rangle \rightarrow \langle p_2, v_2 \rangle$ and $\langle \gamma_2, T_2 \rangle : \langle p_2, v_2 \rangle \rightarrow \langle p_3, v_3 \rangle$, their composition is $\langle \gamma, T \rangle$ with $T = T_1 + T_2$ and

$$\gamma(t) = \begin{cases} \gamma_1(t) & 0 \leq t \leq T_1 \\ \gamma_2(t - T_1) & T_1 \leq t \leq T_1 + T_2. \end{cases} \quad (5.1)$$

Make a technical sketch of the manifold showing what are the velocities, how do paths look like, etc.

Since we are only amateurs, we don’t feel comfortable with 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 homset $\text{Hom}(\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle)$, a function

$$\text{MaxSteepness} : \text{Hom}(\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle) \longrightarrow [0, 1].$$

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*. Let’s 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.

5.3. Generating categories from graphs

Put this section after all concrete examples

To begin, we recall some formal definitions related to (directed) graphs.

Need nice pictures of graphs and various quantities.

Definition 5.3 (Graph). A (directed) *graph* $G = \langle V, A, s, t \rangle$ consists of a set of vertices V , a set of arrows A , and two functions $s, t : A \rightarrow V$, called the *source* and *target* functions, respectively. Given $a \in A$ with $s(a) = v$ and $t(a) = w$, we say that a is an *arrow* from v to w .

Remark 5.4. Both directed graphs and undirected graphs play a prominent role in many kinds of mathematics. In this text, we work primarily with directed graphs and so, from now on, we will drop the “directed”: unless indicated otherwise, the word “graph” will mean “directed graph”.

Definition 5.5 (Path). Let G be a graph. A *path* in G is a sequence of arrows such that the target of one arrow is the source of the next. The *length* of a path is the number of arrows in the sequence. We also formally allow for sequences made up of “zero-many” arrows (such paths therefore have length zero). We call such paths *trivial* or *empty*. If paths describe a journey, then trivial paths correspond to “not going anywhere”. The notions of source and target for arrows extend, in an obvious manner, to paths. For trivial paths, the source and target always coincide.

The following definition provides a way of turning any graph into a category.

Definition 5.6 (Free category on a graph). Let $G = \langle V, A, s, t \rangle$ be a graph. The *free category on G* , denoted $\mathbf{Free}(G)$, has as objects the vertices V of G , and given vertices $x \in V$ and $y \in V$, the morphisms $\mathbf{Free}(G)(x, y)$ are the paths from x to y . The composition of morphisms is given by concatenation of paths, and for any object $x \in V$, the associated identity morphism id_x is the trivial path which starts and ends at x .

Show a picture of a graph and its induced category.

We leave it to the reader to check that the above definition does indeed define a category.

Let's do it ourselves

Exercise 5.7. Consider the following five graphs. For each graph G , how many morphisms in total are there in the associated category $\mathbf{Free}(G)$?



5. Connection

6. Relation

to write

Contents

6.1. Distribution networks	55
6.2. Relations	58
6.3. Relational Databases	66



6. Relation

6.1. 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, low voltage transmission lines and nodes, and consumers (e.g. homes and businesses). The situation is depicted in Fig. 6.1.

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 6.1.: Components of electrical power grids.

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. 6.2.

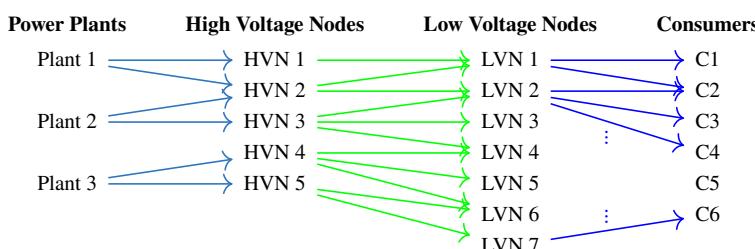


Figure 6.2.: Connectivity between components in electric power grids.

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 paths traced by sequences of arrows, as in Fig. 6.3. 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. 6.2, we can also include, for example, information about the connectivity of high voltage nodes among themselves, as in Fig. 6.4.

The information encoded in Fig. 6.4 and Fig. 6.2 can also be displayed as a single graph, see Fig. 6.5, Fig. 6.4. If we ignore the directionality of the arrows, this is analogous to a depiction of type shown in Fig. 6.6, which is a schema of a power grid.¹

¹See https://en.wikipedia.org/wiki/Electrical_grid and <https://doi.org/10.1109/JYST.2015.2427994>

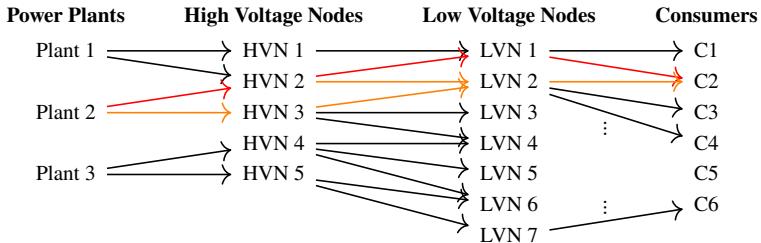


Figure 6.3.: Connection between consumers and power plants.

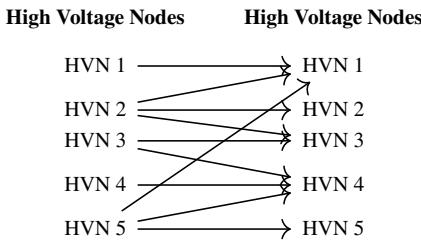


Figure 6.4.: Connectivity between high voltage nodes.

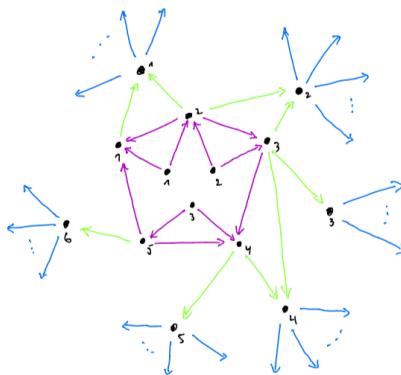


Figure 6.5.: Alternative visualization for connectivity.

6.2. Relations

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

Definition 6.1 (Binary relation). A *binary relation* from a set X to a set Y is a subset of the Cartesian product $X \times Y$.

Remark 6.2. We will often drop the word “binary” and simply use the name “relation”.

If X and Y are finite sets, we can depict a relation $R \subseteq X \times Y$ graphically as in Fig. 6.7. For each

6. Relation

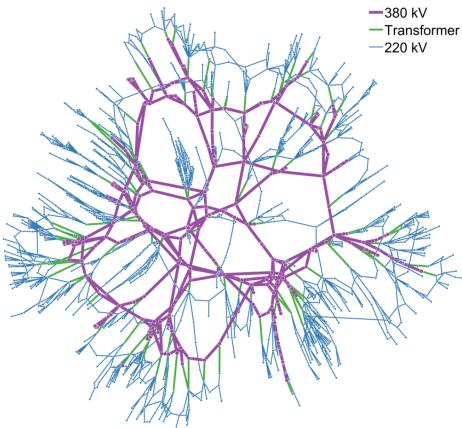


Figure 6.6.: A schematic view of a power grid.

element $\langle x, y \rangle \in X \times Y$, we draw an arrow from x to y if and only if $\langle x, y \rangle \in R \subseteq X \times Y$.

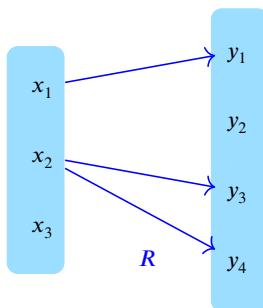


Figure 6.7.

We can also depict this relation graphically as a subset of $X \times Y$ in a “coordinate system way”, as in Fig. 6.8. The shaded grey area is the subset R defining the relation.

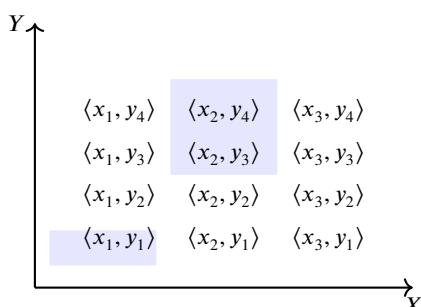


Figure 6.8.: Relations visualized in “coordinate systems”.

Exercise 6.3. Let $X = Y = \{1, 2, 3, 4\}$ and consider the relation $R \subseteq X \times Y$ defined by

$$R = \{\langle x, y \rangle \in X \times Y \mid x \leq y\}. \quad (6.1)$$

Visualize the relation R via the method in Fig. 6.7 and Fig. 6.8 each.

The visualization in Fig. 6.7 hints at the fact that we can think of a relation $R \subseteq X \times Y$ as a *morphism* from X to Y .

Definition 6.4 (Category Rel). The category **Rel** of relations **Rel** is given by:

1. *Objects:* The objects of this category are all sets.
2. *Morphisms:* Given sets X, Y , the homset $\text{Hom}_{\text{Rel}}(X, Y)$ consists of all relations $R \subseteq X \times Y$.
3. *Identity morphisms:* Given a set X , its identity morphism is

$$1_X := \{\langle x, y \rangle \mid x = y\}. \quad (6.2)$$

4. *Composition:* Given relations $R : X \rightarrow Y, S : Y \rightarrow Z$, their composition is given by

$$R ; S := \{\langle x, z \rangle \mid \exists y \in Y : (\langle x, y \rangle \in R) \wedge (\langle y, z \rangle \in S)\}. \quad (6.3)$$

To illustrate the composition rule in Eq. (6.3) for relations, let's consider a simple example, involving sets X, Y , and Z , and relations $R : X \rightarrow Y$ and $S : Y \rightarrow Z$, as depicted graphically below in Fig. 6.9. Now, according to the rule in Eq. (6.3), the composition $R ; S \subseteq X \times Z$ will be such

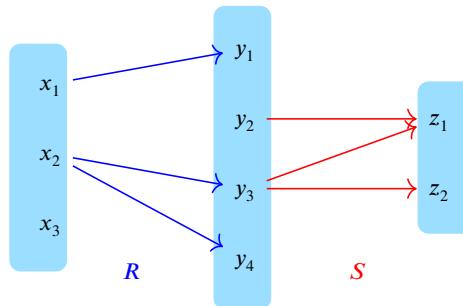


Figure 6.9.: Relations compatible for composition.

that $\langle x, z \rangle \in R ; S$ if and only if there exists some $y \in Y$ such that $\langle x, y \rangle \in R$ and $\langle y, z \rangle \in S$, which, graphically, means that for $\langle x, z \rangle$ to be an element of the relation $R ; S$, x and y 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. 6.9, there is an arrow from x_2 to y_3 , and from there on to z_1 , and therefore, in the composition $R ; S$ depicted in Fig. 6.10, there is an arrow from x_2 to z_1 .

Remark 6.5. Relations with the same source and target can be *compared* via inclusion. Given $R \subseteq X \times Y$ and $R' \subseteq X \times Y$, we can ask whether $R \subseteq R'$ or $R' \subseteq R$.

A question on your mind at this point might be: what is the relationship between relations and functions? One point of view is that functions are special kinds of relations.

Definition 6.6 (Functions as relations). Let X and Y be sets. A relation $R \subseteq X \times Y$ is a **function** if it satisfies the following two conditions:

6. Relation

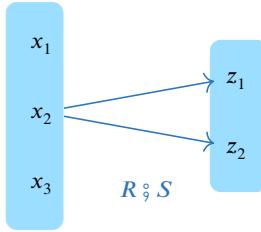


Figure 6.10.: Composition of relations.

1. \$\forall x \in X \quad \exists y \in Y : \langle x, y \rangle \in R\$
2. \$\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R\$ holds : \$x_1 = x_2 \Rightarrow y_1 = y_2\$.

What does this definition have to do with the “usual” way that we think about functions?

Let start with a relation \$R \subseteq X \times Y\$ satisfying the conditions of Definition 6.6. We’ll build from it a function \$f_R : X \rightarrow Y\$. Choose an arbitrary \$x \in X\$. According to point 1. in Definition 6.6, there exists a \$y \in Y\$ such that \$\langle x, y \rangle \in R\$. So let’s choose such a \$y\$, and call it \$f_R(x)\$. This gives us recipe to get from any \$x\$ to a \$y\$. But maybe you are worried: given a specific \$x \in X\$, 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 X\$ 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\$.

Given a function \$f : X \rightarrow Y\$, we can turn it into a relation in a simple way: we consider its graph

$$R_f := \text{graph}(f) = \{\langle x, y \rangle \in X \times Y \mid y = f(x)\}.$$

The relation \$R_f\$ encodes the same information that \$f\$ encodes – simply in a different form.

In this text, we take Definition 6.6 as our rigorous definition of a what a function is. Nevertheless, we’ll often use functions “in the usual way”, e.g. we’ll write things like \$y = f(x)\$.

Another question you may be wondering about is this: if we define functions as special kinds of relations, how then do we define the composition of functions? The answer is that we compose functions simply by the rule for composing relations.

Lemma 6.7. Let \$R \subseteq X \times Y\$ and \$S \subseteq Y \times Z\$ be relations which are functions. Then their composition \$R ; S \subseteq X \times Z\$ is again a function.

Proof. We check that \$R ; S\$ satisfies the two conditions stated in Definition 6.6.

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

$$\langle x_1, y_1 \rangle \in R \text{ and } \langle y_1, z_1 \rangle \in S,$$

$$\langle x_2, y_2 \rangle \in R \text{ and } \langle y_2, z_2 \rangle \in S.$$

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.

□

Example 6.8. Can we have a function (or relation) whose source is the empty set \emptyset ? Given any set Y , such a relation would be of the form $R \subseteq \emptyset \times Y := \emptyset$. This implies that $R = \emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $\emptyset \rightarrow Y$:

- For all $x \in X = \emptyset$, $\exists y \in Y$ such that $\langle x, y \rangle \in R$ (trivially satisfied).
- Clearly, given $\langle x, y \rangle, \langle x', y' \rangle \in R = \emptyset$, having $x = x'$ implies $y = y'$.

Therefore, the answer to the original question is yes.

Example 6.9. Can we have a function (or relation) whose target is the empty set \emptyset ? Again, given any set X , such a relation would be of the form $R \subseteq X \times \emptyset := \emptyset$. This, again, implies $R\emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $X \rightarrow \emptyset$:

- For all $x \in X$, $\exists y \in Y = \emptyset$ such that $\langle x, y \rangle \in R$? Unless $X = \emptyset$, this is not satisfied.

Therefore, given $X \neq \emptyset$, there is no function (or relation) $X \rightarrow \emptyset$.

Definition 6.10 (Properties of a relation). Let $R \subseteq X \times Y$ be a relation. R is:

1. *Surjective* if $\forall y \in Y \exists x \in X : \langle x, y \rangle \in R$;
2. *Injective* if $\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R$ it holds: $y_2 = y_1 \Rightarrow x_1 = x_2$;
3. *Defined-everywhere* if $\forall x \in X \exists y \in Y : \langle x, y \rangle \in R$;
4. *Single-valued* if $\forall \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R$ it holds: $x_1 = x_2 \Rightarrow y_1 = y_2$.

Example 6.11. The relation depicted in Fig. 6.7 is injective but not surjective, i.e. if $\langle x, y \rangle, \langle x', y' \rangle \in R$ and $y = y'$, then $x = x'$.

One can notice a certain duality in the properties listed in Definition 6.10. This is made more precise through the following definition.

Definition 6.12 (Transpose of a relation). Let $R \subseteq X \times Y$ be a relation. The *transpose* (or *opposite*, *reverse*) of R is the relation given by:

$$R^\top := \{ \langle y, x \rangle \in Y \times X \mid \langle x, y \rangle \in R \}.$$

note that $R^\top : Y \rightarrow X$, while $R : X \rightarrow Y$.

Remark 6.13. In the following, we list some properties which refer to relations and their opposites. It is a good exercise to prove them:

- $(R^\top)^\top = R$;
- If R is everywhere-defined, then R^\top is surjective;
- If R is single-valued, then R^\top is injective.
- If R is everywhere defined, then $\text{id}_X \subseteq R \circ R^\top$;
- If R is single-valued, then $R^\top \circ R \subseteq \text{id}_Y$.

Remark 6.14. The aforementioned duality can be seen by “reading the relations (arrows) backwards” (Fig. 6.11).

Definition 6.15 (Endorelation). An *endorelation* on a set X is a relation $R \subseteq X \times X$.

Example 6.16. “Equality” is an endorelation of the form

$$\{ \langle x_1, x_2 \rangle \in X \times X \mid x_1 = x_2 \}.$$

Example 6.17. Take $X = \mathbb{N}$. The relation “less than or equal” is an endorelation of the form

$$\{ \langle m, n \rangle \in \mathbb{N} \times \mathbb{N} \mid m \leq n \}.$$

6. Relation

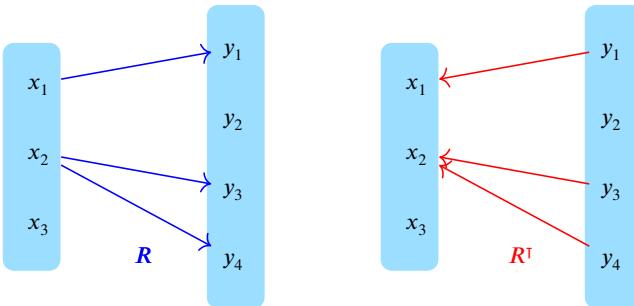


Figure 6.11.

Example 6.18. The relation depicted in Fig. 6.4 is an endorelation between the set of high voltage nodes.

Definition 6.19 (Properties of endorelations). Let $R \subseteq X \times X$ be an endorelation. R is:

- *Symmetric* if $\forall x, x' \in X : \langle x, x' \rangle \in R \Leftrightarrow \langle x', x \rangle \in R$;
- *Reflexive* if $\forall x \in X : \langle x, x \rangle \in R$;
- *Transitive* if $\forall \langle x, x' \rangle \in R$ and $\langle x', x'' \rangle \in R$, we have $\langle x, x'' \rangle \in R$.

Example 6.20. 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 $l \leq n$.

Example 6.21. The relation depicted in Fig. 6.4 is reflexive (each node is connected with itself).

Example 6.22. The endorelation reported in Fig. 6.12 is a symmetric relation on $X = \{x_1, x_2\}$.

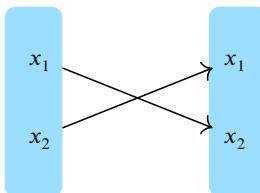


Figure 6.12.: Example of symmetric endorelation.

Definition 6.23 (Equivalence relation). An endorelation $R \subseteq X \times X$ is an *equivalence relation* if it is symmetric, reflexive, and transitive. We write $x \sim x'$ if $\langle x, x' \rangle \in R$.

Example 6.24. The relation “equals” on \mathbb{N} is an equivalence relation. The relation “less than or equal” on \mathbb{N} is not.

Example 6.25. 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 everyone has the same birthday as itself. It 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.

Example 6.26. Let $f : X \rightarrow Y$ be a function between sets. The following defines an equivalence relation:

$$x \sim x' \Leftrightarrow f(x) = f(x').$$

Definition 6.27 (Partition). A *partition* of a set X is a collection $\{X_i\}_{i \in I}$ of subsets $X_i \subseteq X$ such that

1. $X_i \cap X_j = \emptyset \quad \forall i \neq j;$
2. $\bigcup_{i \in I} X_i = X.$

Remark 6.28. Equivalence relations are a way to group together elements of a set which we think of as “the same” in some respect. There is a one-to-one correspondence between equivalence relations on a set X and partitions on X .

Example 6.29. An example of partitions can be shown through information networks. An exemplary network is reported in Fig. 6.13. Here, nodes represent data centers, and the arrows represent information flows. We say that data centers x and y are equivalent (i.e., $x \sim y$) if and only if there is a path from x to y and a path from y to x . In this case, we have $a \sim b$, $e \sim d$, and all centers equivalent with themselves.

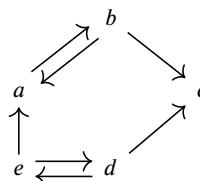


Figure 6.13.

6.3. Relational Databases

A *relational database* like PostgreSQL, MySQL, etc. presents the data to the user as relations. This does not necessarily mean that the data is stored as tuples, as in the mathematical model, but rather that what the user can do is query and manipulate relations. This conceptual model is now 50 years old.

Can we use the category **Rel** to represent databases [[codd2002relational](#)]?

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 motors².

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 6.1.

Such database table can be seen as representing an n -ary relation with $n = 7$, as we are expressing a relation over the sets

$$M \times C \times S \times W \times J \times P,$$

²See pololu.com for a standard catalogue of electric stepper motors.

6. Relation

Motor ID	Company	Size [mm ³]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	20 x 20 x 30	60.0	2.34	19.95
1206	SOYO	28 x 28 x 45	140.0	3.00	19.95
1207	SOYO	35 x 35 x 26	130.0	2.07	12.95
2267	SOYO	42 x 42 x 38	285.0	4.76	16.95
2279	Sanyo Denki	42 x 42 x 31.5	165.0	5.40	164.95
1478	SOYO	56.4 x 56.4 x 76	1,000	8.96	49.95
2299	Sanyo Denki	50 x 50 x 16	150.0	5.90	59.95

Table 6.1.: A simplified catalogue of motors.

where M represents the set of motor IDs, C the set of companies producing motors, S the set of motor sizes, W the set of motor weights, J the set of possible maximal powers, and P the set of possible prices. An n -ary relation is a relation over n sets, just like a binary relation is a relation over 2 sets.

Definition 6.30 (n -ary relation). An n -ary relation on n sets $\langle X_1, X_2, \dots, X_n \rangle$ is a subset of the product set

$$X_1 \times X_2 \times \dots \times X_n.$$

Rel only allows binary relations. Morphisms in **Rel** have 1 source and 1 target. There is no immediate and natural way to represent n -ary relations using **Rel**.

To represent relational databases categorically, there are at least 3 options.

Option 1: Hack it We will introduce the notion of *products* and *isomorphisms*. This will allow us to say that because

$$X_1 \times X_2 \times X_3 \times \dots \times X_n,$$

is isomorphic to

$$X_1 \times (X_2 \times (X_3 \times \dots \times X_n))$$

we can talk about n -ary relations in terms of binary relations. This is not really a natural way to do it.

Option 2: Mutant Morphisms What if morphisms could have more than “two legs”? There are indeed theories that work with more complicated arrows. For example: [multicategories](#), [polycategories](#), [operads](#).

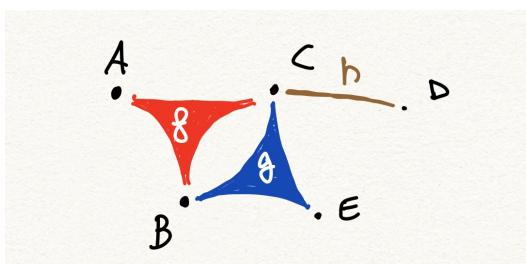


Figure 6.14.: Can you imagine how to define composition with mutant morphisms with more than two legs?

Option 3: Categorical databases A different perspective is that of *categorical databases* [spivak2019cat]. In this modeling framework one does not model the data tables as relations directly. Rather the data is described as a functor from a category representing the schema to **Set**.

6. Relation

7. Computing

to write

Contents

7.1. Databases, sets, functions	71
7.2. The Category Set	73
7.3. Propositions	74



7. Computing

7.1. Databases, sets, functions

In this section, transmuted and transmuter are switched

AC: I'm not sure about this example. I feel that if one is happy with relational databases as a trivial example, they already know Intuitively functions as morphisms (maybe because they know types). If you don't know databases, then that topic cannot be used to introduce a simple concept such as databases.

JL: In the second sentence above, is "that topic" referring to "functions as morphisms"? I'm also not sure if I understand the overall comment. In any case, if it's helpful: the original idea behind this database example was to do things in a way that is compatible with how databases are treated in Seven Sketches. (I'm not necessarily attached to keeping this example, this is just to explain the idea/intention.) I don't know anything about databases anyway :) I just thought it probably works well if it is in line with the FQL way of seeing things.

AC: I propose to have here the **Spreadsheet** category: objects are cells, morphisms are the formulas. There is an object 1 that can be the domain for the constant value of cells.

We continue the discussion of Section 6.3.

In the particular case of tables with primary keys, things are easier. In relational databases, a table column is a primary key if the values of that column are guaranteed to be unique.

If the values in the column are unique, the column serves as the name of the row. In the table above the motor ID serves as the primary key.

Consider a table with columns $P \times X_1 \times X_2 \times \dots \times X_n$, where P is the primary key column. Then, given a key $p \in P$, we can obtain the value in the other columns. We first find the unique row with the key p , and then we read out the values.

Therefore, a table with columns $P \times X_1 \times X_2 \times \dots \times X_n$ can be seen as a tuple $\langle S, \{f_i\}_i \rangle$:

- A subset $A \subseteq P$ that gives us the available keys.
- n read-out functions $f_i : P \rightarrow X_i$, each giving the corresponding value of the i -th attribute.

JL: I don't see yet how the set A comes into play.

GZ:here maybe we don't want color, maybe let's not use transmuted thing?

In this example, we can consider the primary key to be the set

$$M := \{1204, 1206, 1207, 2267, 2279, 1478, 2299\},$$

of models of motors. The other columns are given by the set

$$C := \{\text{SOYO, Sanyo Denki}\}$$

of manufacturing companies, the set S of possible motor sizes, the set W of possible weights, the set J of possible maximal powers, and the set P of possible prices. Each attribute of a motor may be thought of as a function from the set M to set of possible values for the given attribute. For example, there is a function **Company** : $M \rightarrow C$ which maps each model to the corresponding company that manufactures it. So, according to Table 6.1, we have e.g. **Company(1204) = SOYO**, and **Company(2279) = Sanyo Denki**, etc.

Note that in "real life", the catalogue of motors might not have seven entries, as in Table 6.1, but has in fact hundreds of entries, and is implemented digitally as a database, i.e. a collection of interrelated tables. In this case, we will want to be able to search and filter the data based on various criteria. Many natural operations on tables and databases may be described simply in terms of operations with functions. We will use this setting as a way to introduce compositional aspects of working with sets and functions, and a preview of how this might be useful for thinking, in particular, about databases.

Sticking with Table 6.1, suppose, for instance, that we want to consider only motors from Company **Sanyo Denki**. In terms of the function

$$\text{Company} : M \rightarrow C$$

this corresponds to the preimage $\text{Company}^{-1}(\{\text{Sanyo Denki}\}) = \{2279, 2299\}$, which is a subset of the set M . Or, we may want to consider only motors which cost between 40 and 200 USD. In terms of the obvious function

$$\text{Price} : M \rightarrow P,$$

this means we wish to restrict ourselves to the preimage

$$\text{Price}^{-1}(\{49.95, 59.95, 164.95\}) = \{1478, 2299, 2279\} \subseteq M.$$

Now suppose we wish to add a column to our table for “volume”, because we may want to only consider motors that have, at most, a certain volume. For this we define a set V of possible volumes (let's take $V = \mathbb{R}_{\geq 0}$, the non-negative real numbers), and define a function

$$\begin{aligned} \text{Multiply} : S &\rightarrow V \\ \langle l, w, h \rangle &\mapsto l \cdot w \cdot h, \end{aligned}$$

which maps any size of motor to its corresponding volume by multiplying together the given numbers for length, width, and height. Now we can compose this function with the function

$$\text{Size} : M \rightarrow S$$

to obtain a function

$$\text{Volume} : M \rightarrow V,$$

which defines a new column in our table. The composition of functions is usually written as $\text{Volume} = \text{Multiply} \circ \text{Size}$, however we stick to our convention of writing $\text{Volume} = \text{Size} ; \text{Multiply}$. Schematically, we can represent what we did as a diagram (Fig. 7.1).

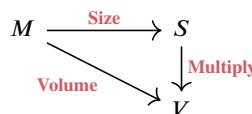


Figure 7.1.: A diagram of functions.

We can interpret arrows in this diagram as being part of a category, where M , S , and V are among the objects, and where the functions **Size**, **Multiply** and **Volume** are morphisms. We probably want to consider the other sets associated with our database as also part of this category, and the other functions which we defined so far, too. One idea might be to just include all the sets and functions that we've defined so far, as well as all possible compositions of those functions, and obtain a category, which we call **Database**, in a way that is similar to how one can build a category from a graph (Section 5.3). This would be an option. However, we may want soon to add new sets and functions to our database framework, or think about new kinds of functions between them that we had not considered before. And we might not want to re-think each time precisely which category we are working with.

7.2. The Category Set

A helpful concept here is to think of our specific sets and functions as living in a very (very) large category which contains all possible sets as its objects and all possible functions as its morphisms. This category is known as the category of sets, and it is an important protagonist in category theory. We will denote it by **Set**. It is a short exercise to check that the following does indeed define a category.

Definition 7.1 (Category of sets). The category of sets **Set** is defined by:

1. *Objects*: all sets.
2. *Morphisms*: given sets X and Y , the homset $\text{Hom}_{\text{Set}}(X, Y)$ is the set of all functions from X to Y .
3. *Identity morphism*: given a set X , its identity morphism id_X is the identity function $X \rightarrow X$, $\text{id}_X(x) = x$.
4. *Composition operation*: the composition operation is the usual composition of functions.

We did say above, however, that we could build a category **Database** which only involves the sets that we are using for our database, and the functions between them that we are working with. What we would need for **Database** to be a category is that if any function is in **Database**, then also its sources and target sets are, and we would need that any composition of functions in **Database** is again in **Database**. (Also, we define the identity morphism for any set in **Database** to be the identity function on that set.) If these conditions are met, **Database** is what is called a *subcategory* of **Set**.

7.3. Propositions

Define the category of propositions where objects are propositions and morphisms are proofs.
Also use to introduce the sequent notations

8. Specialization

to write

Contents

8.1. Notion of subcategory	77
8.2. Drawings	77
8.3. Other examples of subcategories in engineering	79
8.4. Subcategories of Berg	81



8. Specialization

8.1. Notion of subcategory

Definition 8.1 (Subcategory). A *subcategory* **D** of a category **C** is a category for which:

1. All the objects in $\text{Ob}_{\mathbf{D}}$ are in $\text{Ob}_{\mathbf{C}}$;
2. For any objects $X, Y \in \text{Ob}_{\mathbf{D}}$, $\text{Hom}_{\mathbf{D}}(X, Y) \subseteq \text{Hom}_{\mathbf{C}}(X, Y)$;
3. If $X \in \text{Ob}_{\mathbf{D}}$, then $\text{id}_X \in \text{Hom}_{\mathbf{C}}(X, X)$ is in $\text{Hom}_{\mathbf{D}}(X, X)$ and acts as its identity morphism;
4. If $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ in **D**, then the composite $f \circ g$ in **C** is in **D** and represents the composite in **D**.

Two important examples of subcategory are the following.

Example 8.2 (Finite Sets). **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 8.3 (**Set** and **Rel**). The category **Set** is a subcategory of **Rel**. To show this, we need to prove the conditions presented in Definition 8.1.

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 $\text{Hom}_{\text{Set}}(X, Y) \subseteq \text{Hom}_{\text{Rel}}(X, Y)$, i.e., that all functions between sets X, Y are a particular subset of all relations between X, Y .
3. 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**.
4. 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 $R \circ S \subseteq X \times Z$, is again a function. This was proven in Lemma 6.7.

8.2. Drawings

Definition 8.4 (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 8.5. Check whether just considering

- affine invertible transformations, or
- rototranslations, or
- scalings, or
- translations, or
- rotations,

as morphisms forms a subcategory of **Draw**.

Add a few figures here.

We can now think about the different types of transformations.

- **Scalings.** Let $s, t \in \mathbb{R}$. Scalings can be represented as functions of the form

$$\begin{aligned} f_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle sx, ty \rangle, \end{aligned}$$

- **Translations.** Let $s, t \in \mathbb{R}$. Translations are functions of the form

$$\begin{aligned} f_{s,t} : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle x + s, y + t \rangle. \end{aligned}$$

- **Rotations.** Let $\theta \in [0, 2\pi)$. Rotations are functions of the form

$$\begin{aligned} f_\theta : \mathbb{R}^2 &\rightarrow \mathbb{R}^2 \\ \langle x, y \rangle &\mapsto \langle x \cos(\theta) + y \sin(\theta), y \cos(\theta) - x \sin(\theta) \rangle. \end{aligned}$$

- ...

Finish the above, show which ones are subcategories, etc.

8.3. 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 forms a subcategory of Set

Definition 8.6 (Injective function). Let $f : X \rightarrow Y$ be a function. The function f is *injective* if, for all $x, x' \in X$ holds: $f(x) = f(x') \implies x = x'$.

Example 8.7. We can define a category **InjSet** which has the same objects as **Set** but restricts the morphisms to be *injective functions*. 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 Definition 8.1 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 $\text{Hom}_{\text{InjSet}}(X, X)$, i.e., the identity function is injective. This proves the third condition. To check the fourth 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 $x, x' \in X$, $f(x) = f(x') \Leftrightarrow x = x'$ and $y, y' \in Y$, $g(y) = g(y') \Leftrightarrow y = y'$. Furthermore, we have:

$$\begin{aligned} (f \circ g)(x) = (f \circ g)(x') &\implies f(x) = f(x') \\ &\implies x = x', \end{aligned}$$

which proves the fourth condition of Definition 8.1, i.e. that the composition of injective functions is injective.

Definition 8.8 (Continuous functions). Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be a function. We call f *continuous* at $c \in \mathbb{R}$ if $\lim_{x \rightarrow c} f(x) = f(c)$; f is continuous over \mathbb{R} if the condition is satisfied for all $c \in \mathbb{R}$.

Example 8.9. We can define a category **Cont** which $\text{Ob}_{\text{Cont}} = \mathbb{R}$ and in which the morphisms are given by continuous functions. Composition and identity are as in **Set**. We want to show that **Cont** is a subcategory of **Set**.

write down formally and use that composition of continuous is continuous

8. Specialization

Continuous functions (topologies)

Differentiable functions: Set to Manifolds

Definition 8.10 (Differentiable functions). A function $f : U \subset \mathbb{R} \rightarrow \mathbb{R}$, defined on an open set U , is *differentiable* at $a \in U$ if the derivative

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h} \quad (8.1)$$

exists; f is differentiable on U if it is differentiable at every point of U .

Example 8.11. the composition of differentiable functions is differentiable

Lipschitz bounded

Definition 8.12 (Lipschitz continuous function). A real valued function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called *Lipschitz* continuous if there exists a positive real constant κ such that, for all $x_1, x_2 \in \mathbb{R}$:

$$|f(x_1) - f(x_2)| \leq \kappa|x_1 - x_2|. \quad (8.2)$$

Example 8.13. the composition of differentiable functions is differentiable

smooth

cont diff, composition is compdiff

Generalization outside of \mathbb{R}

Generalization to more general spaces. We used the fact that \mathbb{R} is:

- For defining continuous functions, we used the fact that \mathbb{R} is topological space (minimum needed for defining a continuous function.). In fact, the real definitioin of continuous function is:

ADD: continuous function

- To define Lipschitz we needed the fact that \mathbb{R} is a metric space
- For differentiable, smooth, you define this on Manifolds. Exists tangent space.

Hence in general, the objects of these are different, so it's not really a relation of subcategory, that requires the objects to be the same. However we will see later that you can generalize this notion using functors.

functor $F:C \rightarrow D$ that is both injective on objects and a faithful functor.

8.4. Subcategories of **Berg**

Recall the category **Berg** presented in Section 5.2. In the following, we want to give both a positive and a negative example of subcategories related to **Berg**.

We first start our discussion by introducing an *amateur* version of **Berg**, called **BergAma**, which only considers paths (morphisms) in **Berg**, whose steepness does not exceed a critical value, say 1/2. Is **BergAma** a subcategory of **Berg**? Let's check the different conditions:

1. The constraint on the maximum steepness restricts the objects which are acceptable in **BergAma** 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, **BergAma** will only contain objects whose identity morphisms do not exceed the steepness constraint, i.e. $\text{Ob}_{\text{BergAma}} \subseteq \text{Ob}_{\text{Berg}}$.
2. For $A, B \in \text{Ob}_{\text{BergAma}}$, we know that paths satisfying the steepness constraint are specific paths in **Berg**, i.e. $\text{Hom}_{\text{BergAma}} \subseteq \text{Hom}_{\text{Berg}}$.
3. The identity morphisms in **Berg** which satisfy the steepness constraint are, by definition, in **BergAma** and they act as identities there.
4. Given two morphisms f, g which can be composed in **BergAma**, the maximum steepness of their composition $f \circ g$ is given by:

$$\text{MaxSteepness}(f \circ g) = \max \{ \text{MaxSteepness}(f), \text{MaxSteepness}(g) \} < 1/2.$$

This shows that **BergAma** is a subcategory of **Berg**. What would an example of non-subcategory of **Berg** be? Let's define 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 fourth property of Definition 8.1.

8. Specialization

9. Sameness

to write

Contents

9.1. Sameness in category theory	85
9.2. Isomorphism is not identity	88



9. Sameness

9.1. 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 $f : X \rightarrow Y$ of sets to be bijective:

Definition 1: “ $f : X \rightarrow Y$ is bijective if, for every $y \in Y$ there exists precisely one $x \in X$ such that $f(x) = y$;

Definition 2: “ $f : X \rightarrow Y$ is bijective if there exists a function $g : Y \rightarrow X$ such that $f \circ g = \text{id}_X$ and $g \circ f = \text{id}_Y$ ”.

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 X and Y . 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 9.1 (Isomorphism). Let \mathbf{C} be a category, let $X, Y \in \mathbf{C}$ 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 9.2. 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 9.3. In Remark 9.2 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 $g_1 = g_2$.

Definition 9.4 (Isomorphic). Let \mathbf{C} be a category, and let $X, Y \in \mathbf{C}$ be objects. 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 9.2 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, let’s look at some examples using sets.

Example 9.5 (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 $f : M \rightarrow \{\text{Si, No}\}$ which maps a given dish $m \in M$ to the statement “Si” (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

$$\{m \in M \mid f(m) = \text{Si} \quad \text{and} \quad g(m) = \text{Ja}\}.$$

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 (“Si” and “Ja”) to “ \heartsuit ”, and their respective negative answers to “ \clubsuit ”. This defines isomorphisms

$$\{\text{Si, No}\} \longleftrightarrow \{\heartsuit, \clubsuit\} \longleftrightarrow \{\text{Ja, Nein}\}$$

whose compositions provide a translation between the Italian and German two-element sets. Using these isomorphisms, we obtain, by composition, new functions

$$\tilde{f} : M \longrightarrow \{\heartsuit, \clubsuit\}, \quad \tilde{g} : M \longrightarrow \{\heartsuit, \clubsuit\},$$

and the set of dishes that Francesca and Gabriel would be willing to share can be written as

$$\{m \in M \mid \tilde{f}(m) = \heartsuit \quad \text{and} \quad \tilde{g}(m) = \heartsuit\}.$$

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

$$\{0, 1\}, \{\text{true, false}\}, \{\perp, \top\}, \{\text{left, right}\}, \{-, +\}, \text{etc.}$$

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, i.e. how they are being used in interaction with other mathematical constructs.

Example 9.6 (Sizes). Suppose we are a manufacturer and we are counting how many wheels are in a certain warehouse. If W denotes the set of wheels that we have, then counting can be modelled as a function $f : W \rightarrow \mathbb{N}$ to the natural numbers. If we find that there are, say, 273 wheels, then our counting procedure gives us a bijective function from W to the set $\{1, 2, 3, \dots, 272, 273\}$. In this case, we don’t care which specific wheel we counted first, second, or last. We could just as well have counted in a different order, which would amount to a different function $f' : W \rightarrow \mathbb{N}$. The only thing we care about is the fact that the sets W and $\{1, 2, 3, \dots, 272, 273\}$ are *isomorphic*; we don’t need to keep track of which counting isomorphism exhibits this fact.

Example 9.7 (Relabelling). Consider the little catalogue in Table 6.1. 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, 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.

9. Sameness

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

$$\text{Company} : M \rightarrow C.$$

The updated version of the “Company” label, using the new set N of model IDs, is obtained by the composition

$$N \xrightarrow{\text{relabel}} M \xrightarrow{\text{Company}} C.$$

Example 9.8. Going back to currency exchangers, recall that any currency exchanger $E_{a,b}$, given by

$$E_{a,b} : \mathbb{R} \times \{\text{USD}\} \rightarrow \mathbb{R} \times \{\text{EUR}\}$$
$$\langle x, \text{USD} \rangle \mapsto \langle ax - b, \text{EUR} \rangle$$

is an isomorphism, since one can define a currency exchanger $E_{a',b'}$ such that

$$E_{a,b} \circ E_{a',b'} = E_{a',b'} \circ E_{a,b} = E_{1,0}.$$

Example 9.9. 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 (i.e., the set has n elements), the number of isomorphisms is given by the number of ways in which one can “rearrange” n elements of the set, which is $n!$.

Example 9.10. In **Set**, isomorphisms between $\mathbb{R} \rightarrow \mathbb{R}$ correspond to invertible functions.

9.2. Isomorphism is not identity

Example 9.11. Let’s consider currencies, and in particular the sets $\mathbb{R} \times \{\text{USD}\}$ and $\mathbb{R} \times \{\text{USD cents}\}$. 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, 10 **USD** and 1,000 **USD cents** are different elements of different sets, but there exists an isomorphism between the two. For one direction, the isomorphism transforms **USD** into **USD cents** (multiplying the real number by 100); the other direction transforms **USD cents** into **USD** (dividing the real number by 100).

Invertible functions are isomorphisms

Strictly monotone functions are invertible from \mathbb{R} to \mathbb{R} ?

isomorphisms in 2D

isomorphisms are permutations

Examples from before: Dynamical systems (open, $d=0$). Objects are sets, morphisms given by state update and readout. Can go back and forth. (category of processes in computer science)

10. Universal properties

to write

Contents

10.1. Universal properties	91
--------------------------------------	----



move universal properties somewhere else

10.1. Universal properties

JL: I think this might be a good place to introduce universal properties via initial and terminal objects (and in particular using the examples of Set and Rel)

Definition 10.1 (Initial and terminal object). Let \mathbf{C} be a category and let $A \in \mathbf{C}$ be an object. We say that A is an *initial object* if, for all $B \in \mathbf{C}$, the hom-set $\text{Hom}_{\mathbf{C}}(A, B)$ has exactly one element. We say that A is a *terminal object* if, for all $D \in \mathbf{C}$, the hom-set $\text{Hom}_{\mathbf{C}}(D, A)$ has exactly one element.

11. Trade-offs

to write

Contents

11.1. Trade-offs	95
11.2. Ordered sets	96
11.3. Chains and Antichains	99
11.4. Upper and lower sets	100
11.5. From antichains to uppersets, and viceversa	101
11.6. Lattices	105



11. Trade-offs

11.1. Trade-offs

Add the examples from session 6:

Trade-offs characterize all engineering disciplines, and can be literally found everywhere. A typical trade-off is the one reported in Fig. 11.1. When designing a product, you want it to be *good*, *fast*, and *cheap*, and typically you can just choose between two of these qualities.

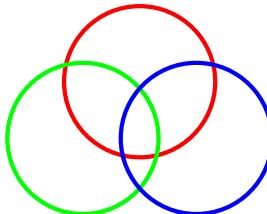


Figure 11.1.

finish example

Functionality, requirements, 3 types of achievable accuracy plots

example: Trade-offs for the human body

example: Masks

example: Hats and headphones

11.2. Ordered sets

So far, the discussion has been purely qualitative. While we discussed how categories can describe the way in which one resource can be turned into another, this kind of modelling did not allow for quantitative statements. For example, it is good to know that we can obtain motion from electric power, but, how fast can we go with a certain amount of power?

To achieve a quantitative theory, we need to specify various degrees of resources and functionality. One way of doing this, is through the idea of partial orders.

Such orderings arise naturally in engineering as criteria for judging whether one design is better or worse than another. As an example, suppose you need to prepare some pizza, i.e., you have to buy specific ingredients and cook them, using a recipe you decide to follow. In this simple example, you can think of having two resources: time and money. A quicker recipe might include more expensive ingredients, and a slower recipe could feature more affordable ones. How to choose among recipes, if you do not prefer one resource over the other? How to model this? In this section, we will assume that functionality and resources are *partially-ordered sets (posets)*.

Davey and Priestley [[davey02](#)] and Roman [[roman08](#)] are possible reference texts.

Introduce at the same time pre-order, poset, total order, like in slides.

Remark: add remark about more general preference functions (Semiorder etc.)

Definition 11.1 (Partially ordered set). A *partially-ordered set (poset)* is a tuple $\langle P, \leq \rangle$, where P is a set (also called the *carrier set*), together with a relation \leq on P that is

1. *Reflexive*: For all $x \in P$, $x \leq x$.
2. *Antisymmetric*: For all $x, y \in P$, if $x \leq y$ and $y \leq x$, then $x = y$.
3. *Transitive*: For all $x, y, z \in P$, if $x \leq y$ and $y \leq z$, then $x \leq z$.

A *Hasse diagram* is an economical (in terms of arrows) way to visualize a poset. In a Hasse diagram elements are points, and if $x \leq y$ then x is drawn lower than y and with an edge connected to it, if no other point is in between. Hasse diagrams are directed graphs.

In the example of the pizza recipes, both time and money can be thought of as partially ordered sets $\langle \mathbb{R}_{\geq 0}, \leq \rangle$. Imagine that you have recipes costing 1 **USD**, 2 **USD**, and 3 **USD**. This can be represented as in Fig. 11.2.



Figure 11.2.: The cost of pizza ingredients can be represented as a poset.

Example 11.2. Consider a poset $P = \{a, b, c, d, e\}$ with $a \leq b$, $a \leq c$, $d \leq c$, and $d \leq e$. This can be represented with a Hasse diagram as in Fig. 11.3.

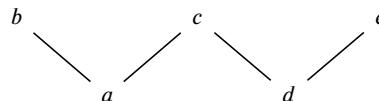


Figure 11.3.: Example of Hasse diagram of P .

Example 11.3 (Singleton poset). If a set has only one element, say $\{1\}$, then there is a unique order relation on it (Fig. 11.4). We denote the resulting poset again by $\{1\}$.



Figure 11.4.: The singleton poset.

Example 11.4. In this example, we represent all posets up to isomorphisms on up to 4 elements. For one element, one has only the singleton poset (Fig. 11.4). On 2-elements sets, one has the posets reported in Fig. 11.5. On 3-elements sets, one has the posets reported in Fig. 11.6. On 4-elements sets, one has the posets reported in Fig. 11.7.

Example 11.5 (Booleans). The booleans is a poset with carrier set $\{\top, \perp\}$ and the order relation given by $b_1 \leq_{\text{Bool}} b_2$ iff $b_1 \Rightarrow b_2$, that is, $\perp \leq_{\text{Bool}} \top$ (Fig. 11.8).

This relation should be familiar from Table 11.1.

11. Trade-offs



Figure 11.5.: All posets on 2-elements sets, up to isomorphisms.

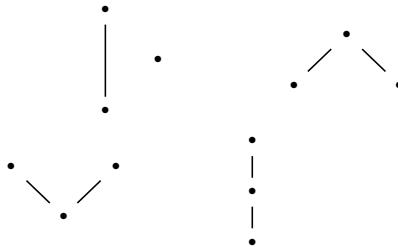


Figure 11.6.: All posets on 3-elements sets, up to isomorphisms.

a	b	$a \leq_{\text{Bool}} b$	$a \wedge b$	$a \vee b$
T	T	T	T	T
T	⊥	⊥	⊥	T
⊥	T	T	⊥	T
⊥	⊥	T	⊥	⊥

Table 11.1.: Properties of the **Bool** poset.

In addition to the operation

$$\Rightarrow : \text{Bool} \times \text{Bool} \rightarrow \text{Bool},$$

called *implies*, 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.

Example 11.6 (Reals). The real numbers \mathbb{R} form a poset with carrier \mathbb{R} and order relation given by the usual ordering $r_1 \leq r_2$.

Example 11.7 (Discrete partially ordered sets). Every set X can be considered as a *discrete poset* $\langle X, = \rangle$. Discrete posets are represented as collection of points (Fig. 11.9).

Example 11.8. Given a set $X = \{a, b, c\}$, consider its power set $\mathcal{P}(X)$. Define sets as the objects of this new category and define the morphisms to be inclusions (Fig. 11.10).

The identity morphism of each set is the inclusion with itself (every set is a subset of itself). Composition is given by composition of inclusions, i.e., if $X \subseteq Y \subseteq Z$, then $X \subseteq Z$.

A note on preorders

Definition 11.9 (Preorder). A *preorder* is a tuple $\langle P, \leq \rangle$, where P is a set (also called the *carrier set*), together with a relation \leq on P that is

1. *Reflexive*: For all $x \in P$, $x \leq x$.
2. *Transitive*: For all $x, y, z \in P$, if $x \leq y$ and $y \leq z$, then $x \leq z$.

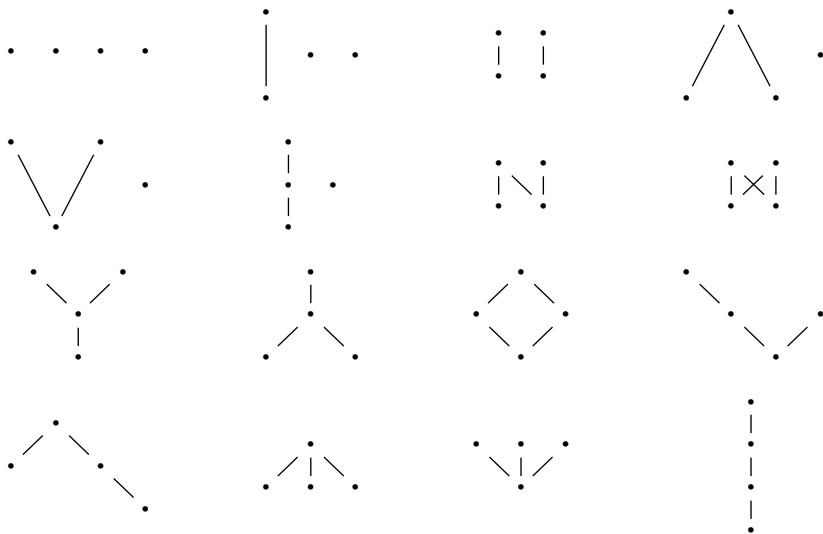


Figure 11.7.: All posets on 4-elements sets, up to isomorphisms.

$$\begin{array}{c} \top \\ | \leq \\ \perp \end{array}$$

Figure 11.8.

The theory of design problems can be easily generalized to preorders. This means that there could be two elements x and y such that $x \leq y$ and $x \geq y$ but $x \neq y$ (Fig. 11.11).

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 preorder, computation will always involve passing to the poset representation.

This means that, given a preorder, we can consider the poset of its isomorphism classes, by means of the following equivalence relation:

$$x \simeq y \quad \equiv \quad (x \leq y) \wedge (y \leq x). \quad (11.1)$$

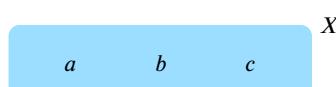


Figure 11.9.: Example of a discrete poset.

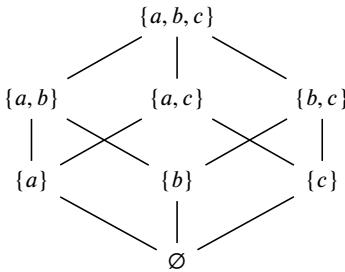
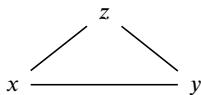


Figure 11.10.: Power set as a category.



Let's not use Hasse for preorders, it's not a thing.

Figure 11.11.: Example of a preorder.

11.3. Chains and Antichains

This part is partially repeated from above. Do chains, antichains, upper/lower sets before "poset as a category".

Definition 11.10 (Chain in a poset). Given a poset S , a *chain* is a sequence of elements s_i in S where two successive elements are comparable, i.e.:

$$i \leq j \Rightarrow s_i \leq s_j. \quad (11.2)$$

Definition 11.11 (Antichain in a poset). An *antichain* is a subset S of a poset where no elements are comparable. If $a, b \in S$, then $a \leq b$ implies $a = b$.

Remark 11.12. We denote the set of antichains of a poset P by \mathcal{AP} .

Remark 11.13. Note that given a poset $\langle P, \leq \rangle$, $\emptyset \in \mathcal{AP}$ since \emptyset contains no elements (and hence no comparable elements).

In the context of pizza recipes, consider the diagram reported in Fig. 11.12. The blue points represent an antichain of recipes $\{\langle 1 \text{ USD}, 2 \text{ h} \rangle, \langle 2 \text{ USD}, 1 \text{ h} \rangle\}$, i.e. recipes which do not dominate each other (one is cheaper but takes longer and the other is more expensive but quicker). The red point represents a recipe which cannot be part of the antichain, since it is dominated by $\langle 2 \text{ USD}, 1 \text{ h} \rangle$.

Example 11.14. Let's consider the poset $\langle P, \leq \rangle$ where $a \leq b$ if a is a divisor of b and $P = \{1, 5, 10, 11, 13, 15\}$. A chain of P is $\{1, 5, 10, 15\}$. An antichain of P is $\{10, 11, 13\}$.

Example 11.15. Consider Example 11.8. Examples of chains are

$$\{\emptyset, \{a\}, \{a, b\}, \{a, b, c\}\}, \quad \{\emptyset, \{b\}, \{b, c\}, \{a, b, c\}\}. \quad (11.3)$$

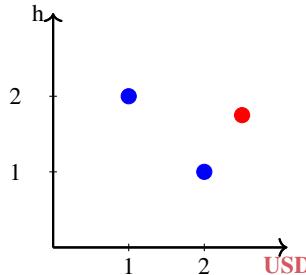


Figure 11.12.: Example of an antichain of pizza recipes.

Examples of antichains are

$$\{\{a\}, \{b\}, \{c\}\}, \quad \{\{a, b\}, \{a, c\}, \{b, c\}\}. \quad (11.4)$$

Example 11.16. Suppose you have to choose a battery model based on its cost and its weight, both to be minimized. There may be models which dominate others. For instance, a model $\langle 10 \text{ USD}, 1 \text{ kg} \rangle$ is better than a model $\langle 11 \text{ USD}, 1.1 \text{ kg} \rangle$. Also, there may be models which are incomparable, i.e. which form an antichain. For example, you cannot say whether $\langle 10 \text{ USD}, 1 \text{ kg} \rangle$ is better than $\langle 5 \text{ USD}, 2 \text{ kg} \rangle$. The incomparable models form an antichain.

11.4. Upper and lower sets

Definition 11.17 (Upper set). An *upper set* is a subset U of a poset P such that, if an element is inside, all elements above it are inside as well. In formulas:

$$U \text{ is an upperset} \equiv \forall x \in U, \forall y \in P : x \leq y \Rightarrow y \in U. \quad (11.5)$$

Remark 11.18. We call UP the set of upper sets of P .

Definition 11.19 (Lower set). A *lower set* is a subset L of a poset P if, if a point is inside, all points below it are inside as well. In formulas:

$$L \text{ is a lower set} \equiv \forall x \in L, \forall y \in P : y \leq x \Rightarrow y \in L. \quad (11.6)$$

Remark 11.20. We call LP the set of lower sets of P .

Remark 11.21. Note that if A is an antichain of a poset P , then the set

$$I(A) = \{x : x \leq y, y \in A\} \quad (11.7)$$

is a lower set of P .

Consider the blue poset of pizza recipes from before. The upper and lower sets of this poset can be represented as in Fig. 11.13. The upper set can be interpreted as all the potential pizza recipes for which we can find better alternatives in the poset. Similarly, the lower set can be interpreted as all the potential pizza recipes which would be better than the ones in the poset.

Example 11.22 (Upper and lower sets in **Bool**). The booleans $\{\perp, \top\}$ form a poset with $\perp \leq \top : (\text{Bool}, \leq)$. The subset $\{\perp\} \subseteq \text{Bool}$ is not an upper set, since $\perp \leq \top$ and $\top \notin \{\perp\}$.

11. Trade-offs

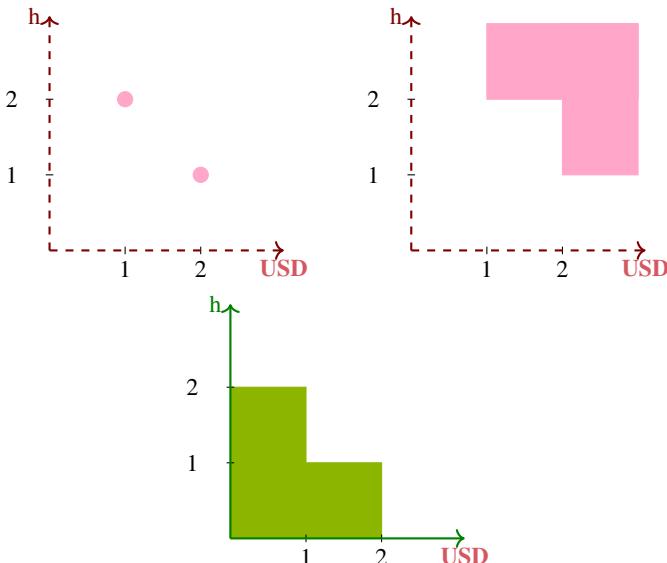


Figure 11.13.: Example of upper and lower sets of a poset of pizza recipes.

11.5. From antichains to uppersets, and viceversa

Definition 11.23 (Upper closure operator). The *upper closure operator* \uparrow maps a subset to the smallest upper set that includes it, i.e.:

$$\begin{aligned} \uparrow: \mathcal{P}(P) &\rightarrow \text{UP} \\ S &\mapsto \{y \in P \mid \exists x \in S : x \leq y\}. \end{aligned} \tag{11.8}$$

Remark 11.24. Note that, by definition, an upper set is closed to upper closure.

Remark 11.25. For any $S \in \mathcal{P}(P)$, $\uparrow S$ is in fact an upper set.

Proof. Suppose $y \in \uparrow S$ and $z \in P$, and suppose $y \leq z$. By definition $\exists x \in S$ s.t. $x \leq y$, meaning that $x \leq z$. Thus, $z \in \uparrow S$, as was to be shown. \square

Lemma 11.26. The upper closure operator \uparrow is a monotone map.

Proof. Consider the posets $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle \text{UP}, \supseteq \rangle$, and $S_1, S_2 \in \mathcal{P}(P)$. It is clear that given $S_1 \subseteq S_2$, one has

$$\{y \in P \mid \exists x \in S_1 : x \leq y\} \supseteq \{y \in P \mid \exists x \in S_2 : x \leq y\}.$$

Therefore, $\uparrow S_1 \supseteq \uparrow S_2$, satisfying the monotonicity property for \uparrow . \square

Lemma 11.27. Let A and B be subsets of P that are antichains. Then

$$\uparrow A = \uparrow B \Rightarrow A = B.$$

Proof. First, let's fix an $a \in A$. From $\uparrow A = \uparrow B$ we know that in particular $A \subseteq \uparrow B$. This means that for our fixed $a \in A$ there exists $b \in B$ such that $b \leq a$. From $\uparrow A = \uparrow B$ it also follows that $B \subseteq \uparrow A$, so to the $b \in B$ given above, there exists $a' \in A$ such that $a' \leq b$. In total, we have $a' \leq b \leq a$, and since A is an antichain, we must have $a' = a$. This implies that $a' = b = a$. In particular, we have $a \in B$.

The above shows that $A \subseteq B$. To show $B \subseteq A$, we can fix any $b \in B$ and repeat the above argumentation, now with the roles of A and B exchanged. \square

In the example of the pizza recipes, first, consider the upper set of a single element of the poset, e.g. $p_1 = \langle 1 \text{ USD}, 2 \text{ h} \rangle$ (Fig. 11.14). Then, consider the case of two elements, with $p_2 = \langle 2 \text{ USD}, 1 \text{ h} \rangle$

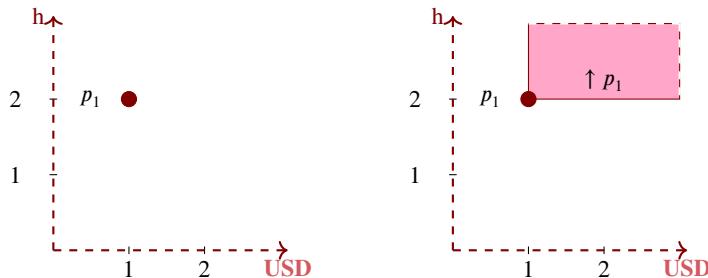


Figure 11.14.: The upper closure of a singleton set of pizza recipes.

(Fig. 11.15).

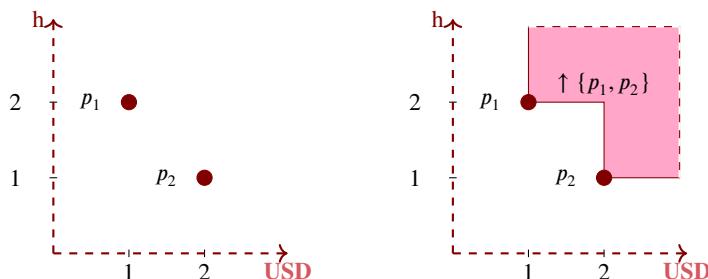


Figure 11.15.: The upper closure of a set of pizza recipes.

Note that the upper set of the subset formed by the two elements is the union of the upper sets of the single elements.

Definition 11.28 (Lower closure operator). The *lower closure operator* \downarrow maps a subset to the smallest lower set that includes it, i.e.

$$\begin{aligned}\downarrow : \mathcal{P}(P) &\rightarrow \mathsf{LP} \\ S &\mapsto \{y \in P \mid \exists x \in S : y \leq x\}.\end{aligned}$$

Lemma 11.29. The lower closure operator \downarrow is a monotone map.

11. Trade-offs

JL: The following proof is a bit redundant... we can say “analogous to the case of the upper closure operation” and/or invoke the principle of duality for posets.

Proof. Consider the posets $\langle \mathcal{P}(P), \subseteq \rangle$ and $\langle \mathcal{LP}, \subseteq \rangle$, and let $S_1, S_2 \in \mathcal{P}(P)$. It is clear that given $S_1 \subseteq S_2$, one has

$$\{y \in P \mid \exists x \in S_1 : y \leq x\} \subseteq \{y \in P \mid \exists x \in S_2 : y \leq x\}. \quad (11.9)$$

Therefore, $\uparrow S_1 \subseteq \uparrow S_2$, satisfying the monotonicity property for \downarrow . \square

Example 11.30. Consider the battery example of Example 11.16, and the antichain given by the battery models $a = \langle 10 \text{ USD}, 1 \text{ kg} \rangle$, $b = \langle 20 \text{ USD}, 0.5 \text{ kg} \rangle$, and $c = \langle 30 \text{ USD}, 0.25 \text{ kg} \rangle$ (Fig. 11.16). The lower closure operator $\downarrow \{a, b, c\}$ represents all the battery models which, if existing, would dominate $\{a, b, c\}$.

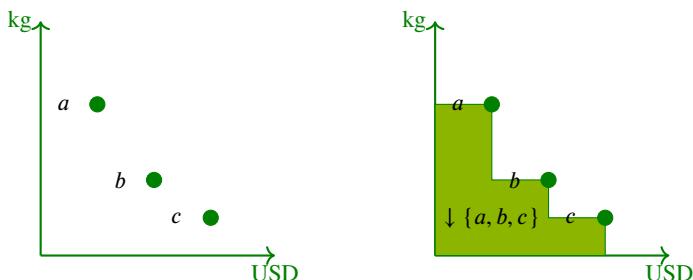


Figure 11.16.: Battery example. From the left: antichain, upper closure, and lower closure.

Definition 11.31 (Min). $\text{Min} : \mathcal{P}(P) \rightarrow \mathcal{AP}$ is the map that sends a subset S of a poset to the minimal elements of that subset, i.e., those elements $a \in S$ such that $a \leq b$ for all $b \in S$. In formulas:

$$\begin{aligned} \text{Min} &: \mathcal{P}(P) \rightarrow \mathcal{AP} \\ S &\mapsto \{x \in S : (y \in S) \wedge (y \leq x) \Rightarrow (x = y)\}. \end{aligned}$$

Note that $\text{Min}(S)$ could be empty.

Definition 11.32 (Max). $\text{Max} : \mathcal{P}(P) \rightarrow \mathcal{AP}$ is the map that sends a subset S of a poset to the maximal elements of that subset, i.e., those elements $a \in S$ such that $a \geq b$ for all $b \in S$. In formulas:

$$\begin{aligned} \text{Max} &: \mathcal{P}(P) \rightarrow \mathcal{AP} \\ S &\mapsto \{x \in S : (y \in S) \wedge (y \geq x) \Rightarrow (x = y)\}. \end{aligned}$$

Note that $\text{Max}(S)$ could be empty.

This is a remnant of older times. To remove.

Lemma 11.33. Given a poset $\langle P, \leq \rangle$, $\langle \mathcal{AP}, \leq_{\mathcal{AP}} \rangle$ is a poset with

$$A \leq_{\mathcal{AP}} B \text{ if and only if } \uparrow A \supseteq \uparrow B. \quad (11.10)$$

Furthermore, it is bounded by the top $\top_{\mathcal{AP}} = \emptyset$ and the bottom $\perp_{\mathcal{AP}} = \{\perp_P\}$.

Proof. We need to show the poset properties (Definition 11.1). We can prove the following:

- *Reflexivity:* From $\langle P, \leq \rangle$ being a poset we know that

$$\begin{aligned} \{y \in P \mid \exists x \in A : x \leq y\} &\supseteq \{y \in P \mid \exists x \in A : x \leq y\}, \\ \uparrow A &= \uparrow A \end{aligned} \tag{11.11}$$

and hence $A \leq_{AP} A$.

- *Antisymmetry:* One has

$$\begin{aligned} (A \leq_{AP} B) \wedge (B \leq_{AP} A) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow A) \\ &\Leftrightarrow \uparrow A = \uparrow B \\ &\Rightarrow A = B. \end{aligned} \tag{11.12}$$

The last implication is by ??.

- *Transitivity:* One has

$$\begin{aligned} (A \leq_{AP} B) \wedge (B \leq_{AP} C) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow C) \\ &\Rightarrow \uparrow A \supseteq \uparrow C \\ &\Rightarrow A \leq_{AP} C. \end{aligned} \tag{11.13}$$

In order to find the top, we need to find the smallest set T_{AP} such that $A \leq_{AP} T_{AP}$ for all $A \in AP$. In other words, such that $\uparrow A \supseteq \uparrow T_{AP}$ for all $A \in AP$. This is clearly \emptyset , since $\uparrow \emptyset = \emptyset$. Similarly, in order to find the bottom, we need to find the set \perp_{AP} such that $\perp_{AP} \leq_{AP} A$ for all $A \in AP$. In other words, such that $\uparrow \perp_{AP} \supseteq \uparrow A$ for all $A \in AP$. We obtain a bottom if we set $\perp_{AP} := T_p$, since $T_p \supseteq A$ for all $A \subseteq P$, and hence, by monotonicity of \uparrow , we have in particular $\uparrow T_p \supseteq \uparrow A$ for all antichains A .

□

Definition 11.34 (Downward closed set). An upper set S is *downward-closed* in a poset P if

$$S = \uparrow \text{Min } S. \tag{11.14}$$

Remark 11.35. The set of downward-closed upper sets of P is denoted \underline{UP} .

11.6. Lattices

Definition 11.36 (Lattice). A *lattice* is a poset $\langle P, \leq \rangle$ with some additional properties:

- Given two points $p, q \in P$, it is always possible to define their least upper bound, called *join*, and indicated as $p \vee q$.
- Given two points $p, q \in P$, it is always possible to define their greatest lower bound, called *meet*, and indicated as $p \wedge q$.

Add the wooden lattice figure here.

These can be defined for preorders as well... move before.

11. Trade-offs

Remark 11.37 (Bounded lattices). If there is a least upper bound for the entire lattice A , it is called the *top* (\top). If a greatest lower bound exists it is called the *bottom* (\perp). If both a top and a bottom exist, we call the lattice *bounded*, and denote it by $\langle A, \leq, \vee, \wedge, \perp, \top \rangle$.

Example 11.38. In Example 11.8 we presented the poset arising from the power set $\mathcal{P}(A)$ of a set A and ordered via subset inclusion. This is a lattice, bounded by A and by the empty set \emptyset . Note that this lattice possesses two (dual) monoidal structures $\langle \mathcal{P}(A), \subseteq, \emptyset, \cup \rangle$ and $\langle \mathcal{P}(A), \subseteq, A, \cap \rangle$.

Example 11.39. Consider the set $\{1, 2, 3, 6\}$ ordered by divisibility. For instance, since 2 divides by 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. 11.17).

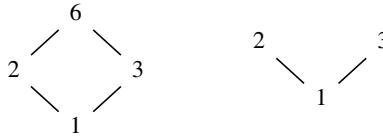


Figure 11.17.: Examples of a lattice and a non-lattice.

Lemma 11.40. UR is a bounded lattice (Definition 11.36) with

$$\langle UR, \leq_{UR}, \perp_{UR}, \top_{UR}, \vee_{UR}, \wedge_{UR} \rangle = \langle UR, \supseteq, R, \emptyset, \cap, \cup \rangle. \quad (11.15)$$

Proof. Consider the poset $\langle UR, \supseteq \rangle$ and $P, Q \in UR$.

- First, we need to show that $P \cap Q \in UR$. One has $P \subseteq UR$ and $Q \subseteq UR$, meaning that by definition, if $x \in P \cap Q$, we have $x \in P \wedge x \in Q$. It follows that $x \in UR$ for all $x \in P \cap Q$. Furthermore, we need to show that $P \cap Q$ is the least upper bound of P, Q . Assume this is not true, i.e. there exists a $T \in UR$, $T \neq P \cap Q$, such that $P \supseteq T \supseteq P \cap Q$ and $Q \supseteq T \supseteq P \cap Q$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} P \cap Q &\supseteq T \cap T \supseteq P \cap Q \\ P \cap Q &\supseteq T \supseteq P \cap Q \\ T &= P \cap Q, \end{aligned} \quad (11.16)$$

which contradicts the assumption. Therefore, $P \cap Q$ is the least upper bound of P, Q .

- Second, we need to show that $P \cup Q \in LF$. One has $P \subseteq UR$ and $Q \subseteq UR$, meaning that by definition, if $x \in P \cup Q$, we have either $x \in P$ or $x \in Q$. If $x \in P$, then $x \in UR$. If $x \in Q$, then $x \in UR$. It follows that $x \in UR$ for all $x \in P \cup Q$. Furthermore, we need to show that $P \cup Q$ is the greatest lower bound of P, Q . Assume this is not true, i.e. there exists a $T \in UR$, $T \neq P \cup Q$, such that $P \cup Q \supseteq T \supseteq P$ and $P \cup Q \supseteq T \supseteq Q$. Using the fact that union preserves inclusions, one has

$$\begin{aligned} (P \cup Q) \cup (P \cup Q) &\supseteq T \cup T \supseteq P \cup Q \\ P \cup Q &\supseteq T \supseteq P \cup Q \\ T &= P \cup Q, \end{aligned} \quad (11.17)$$

which contradicts the assumption. Therefore, $P \cup Q$ is the greatest lower bound of P, Q .

We have therefore proved that $\langle UR, \supseteq \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq T$ for any $T \in UR$, meaning that \emptyset is the top. Furthermore, we notice that $T \subseteq R$ for any $T \in UR$, meaning that R is a bottom. Therefore, the lattice is bounded. \square

Lemma 11.41. LF is a bounded lattice (Definition 11.36) with

$$\langle \text{LF}, \leq_{\text{LF}}, \perp_{\text{LF}}, \top_{\text{LF}}, \vee_{\text{LF}}, \wedge_{\text{LF}} \rangle = \langle \text{LF}, \subseteq, \emptyset, F, \cup, \cap \rangle. \quad (11.18)$$

Proof. Consider the poset $\langle \text{LF}, \subseteq \rangle$ and $P, Q \in \text{LF}$.

- First, we need to show that $P \cup Q \in \text{LF}$. One has $P \subseteq \text{LF}$ and $Q \subseteq \text{LF}$, meaning that by definition, if $x \in P \cup Q$, either $x \in P$ or $x \in Q$. If $x \in P$, then $x \in \text{LF}$. If $x \in Q$, then $x \in \text{LF}$. It follows that $x \in \text{LF}$ for all $x \in P \cup Q$. Furthermore, we need to show that $P \cup Q$ is the least upper bound of P, Q . Assume this is not true, i.e. there exists a $T \in \text{LF}$, $T \neq P \cup Q$, such that $P \subseteq T \subseteq P \cup Q$ and $Q \subseteq T \subseteq P \cup Q$. Using the fact that union preserves inclusions, one has

$$\begin{aligned} P \cup Q &\subseteq T \cup T \subseteq P \cup Q \\ P \cup Q &\subseteq T \subseteq P \cup Q \\ T &= P \cup Q, \end{aligned} \quad (11.19)$$

which contradicts the assumption. Therefore, $P \cup Q$ is the least upper bound of P, Q .

- Second, we need to show that $P \cap Q \in \text{LF}$. One has $P \subseteq \text{LF}$ and $Q \subseteq \text{LF}$, meaning that by definition, if $x \in P \cap Q$, we have $x \in P \wedge x \in Q$, i.e. $x \in \text{LF}$, for all $x \in P \cap Q$. Furthermore, we need to show that $P \cap Q$ is the greatest lower bound of P, Q . Assume this is not true, i.e. there exists a $T \in \text{LF}$, $T \neq P \cap Q$, such that $P \cap Q \subseteq T \subseteq P$ and $P \cap Q \subseteq T \subseteq Q$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} (P \cap Q) \cap (P \cap Q) &\subseteq T \cap T \subseteq P \cap Q \\ P \cap Q &\subseteq T \subseteq P \cap Q \\ T &= P \cap Q, \end{aligned} \quad (11.20)$$

which contradicts the assumption. Therefore, $P \cap Q$ is the greatest lower bound of P, Q .

We have therefore proved that $\langle \text{LF}, \subseteq \rangle$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq T$ for any $T \in \text{LF}$, meaning that \emptyset is the bottom. Furthermore, we notice that $T \subseteq F$ for any $T \in \text{LF}$, meaning that F is a top. Therefore, the lattice is bounded. \square

11. Trade-offs

12. Poset constructions

to write

Contents

12.1. Opposite of a poset	117
12.2. Poset of intervals	118
12.3. A different poset of intervals	118



12.0.1. Product of posets

We can think of the product of posets.

Definition 12.1 (Product of posets). Given two posets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, the *product poset* is $\langle X \times Y, \leq_{X \times Y} \rangle$, where $X \times Y$ is the Cartesian product of two sets (Definition 19.1) and the order $\leq_{X \times Y}$ is given by:

$$\langle x_1, y_1 \rangle \leq_{X \times Y} \langle x_2, y_2 \rangle \Leftrightarrow (x_1 \leq_X x_2) \wedge (y_1 \leq_Y y_2). \quad (12.1)$$

Recalling the pizza recipes example, we have the two posets representing time and money. Given that we want to minimize both time and costs, by considering the money poset containing elements 1 USD, 2 USD, and 3 USD, and the time poset containing elements 1 h, and 2 h, one can represent the product as in Fig. 12.1.

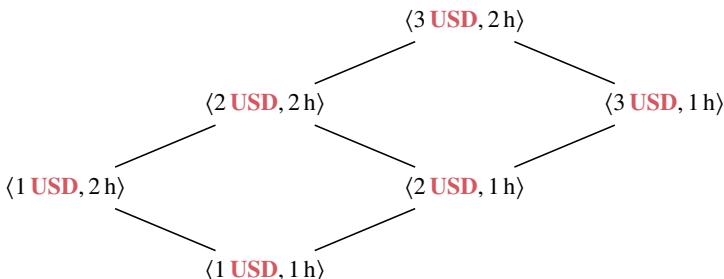


Figure 12.1.: Product poset of time and cost for pizza recipes.

Example 12.2. Consider now the two posets given in Fig. 12.2. Their product is depicted in Fig. 12.3.

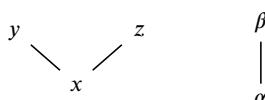


Figure 12.2.: Two posets.

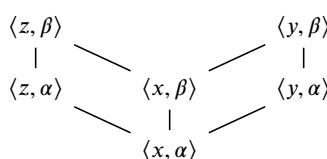


Figure 12.3.: Product of two posets.

12.0.2. Disjoint union of posets

Similarly to what we have done for sets in Section 19.2, we can think of alternatives in the poset case through their disjoint union.

Definition 12.3 (Disjoint union of posets). Given posets $\langle X, \leq_X \rangle$ and $\langle Y, \leq_Y \rangle$, we can define their *disjoint union* $\langle X + Y, \leq_{X+Y} \rangle$, where $X + Y$ is the disjoint union of the sets X and Y (Definition 19.14), and the order \leq_{X+Y} is given by:

$$x \leq_{X+Y} y \equiv \begin{cases} x \leq_A y, & x, y \in X, \\ x \leq_B y, & x, y \in Y. \end{cases} \quad (12.2)$$

$$\begin{aligned} \leq_{X+Y} : (X + Y) \times (X + Y) &\rightarrow \text{Bool} \\ \langle 1, x_1 \rangle, \langle 1, x_2 \rangle &\mapsto (x_1 \leq_X x_2) \\ \langle 2, y \rangle, \langle 1, x \rangle &\mapsto \perp \\ \langle 1, x \rangle, \langle 2, y \rangle &\mapsto \perp \\ \langle 2, y_1 \rangle, \langle 2, y_2 \rangle &\mapsto (y_1 \leq_Y y_2). \end{aligned} \quad (12.3)$$

Example 12.4. Consider the posets $X = \langle \diamond, \star \rangle$ with $\diamond \leq_X \star$, and $Y = \langle \dagger, * \rangle$, with $* \leq_Y \dagger$. Their disjoint union can be represented as in Fig. 12.4.

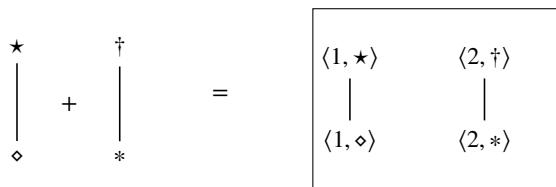


Figure 12.4.: Disjoint union of posets.

12.1. Opposite of a poset

Definition 12.5. The *opposite* of a poset $\langle A, \leq \rangle$ is the poset denoted as $\langle A^{\text{op}}, \leq^{\text{op}} \rangle$ that has the same elements as A and the reverse ordering (Fig. 12.5). For a given $x \in A$, we use x^* to represent its corresponding copy in A^{op} ; note that x and x^* belong to distinct posets. Reversing the order means that, for all $x, y \in A$,

$$x \leq y \Leftrightarrow y^* \leq^{\text{op}} x^*. \quad (12.4)$$

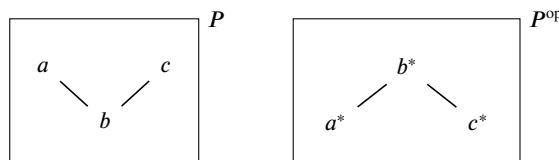


Figure 12.5.: Opposite of a poset.

Example 12.6 (Credit and debt). Let us define the set

$$\text{USD} = \{\$0.00, \$0.01, \$0.02, \dots\}$$

of all US dollars monetary quantities approximated to the cent. From this set we can define two posets: $\text{USD}^+ = \langle \text{USD}, \leq \rangle$ and $\text{USD}^- = \langle \text{USD}, \geq \rangle$, that are the opposite of each other. If the context is that, given two quantities \$1 and \$2, we prefer \$1 to \$2 (for example because it is a cost to pay to acquire a component), then we are working in USD^+ , otherwise we are working in USD^- (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”.

12.2. Poset of intervals

Definition 12.7 (Poset of intervals). An interval is an ordered pair of elements $\langle l, u \rangle$ of P , such that $l \leq_P u$. Given a poset P , one can define a *poset of intervals* on P . Intervals can be ordered by inclusion, e.g.:

$$\langle l_1, u_1 \rangle \leq_{\text{Int } P} \langle l_2, u_2 \rangle \Leftrightarrow (l_1 \leq_P l_2) \wedge (u_2 \leq_P u_1).$$

12.3. A different poset of intervals

to write

13. Life is hard

to write

Contents

13.1. Monotone maps	121
13.2. Compositionality of monotonicity	124
13.3. The category Pos of posets and monotone maps	124
13.4. Why Pos is not sufficient for design theory	125



13.1. Monotone maps

AC: Add some motivation from design. E.g. generalization of "non-decreasing" and "non-increasing" from real numbers to partial orders

Definition 13.1 (Monotone map). A *monotone map* between two posets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$ is a map that preserves the ordering, in the sense that

$$a \leq_A b \Rightarrow f(a) \leq_B f(b).$$

Remark 13.2. Given a poset A , the map id_A is monotone, since for $a_1, a_2 \in A$, one has:

$$\begin{aligned} a_1 \leq_A a_2 &\Rightarrow a_1 = a_1 \circ \text{id}_A \\ &\leq_A a_2 \circ \text{id}_A \\ &= a_2. \end{aligned}$$

Definition 13.3 (Antitone map). An *antitone map* between two posets $\langle A, \leq_A \rangle$ and $\langle B, \leq_B \rangle$ is a map that reverses the ordering, in the sense that

$$a \leq_A b \Rightarrow f(a) \geq_B f(b).$$

Example 13.4 (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 $t : \mathbb{N} \rightarrow \mathbb{R}_+$ is a map between the ordered sets \mathbb{N} and \mathbb{R}_+ , and maps each quantity of widgets to a total manufacturing cost (Fig. 13.2). Clearly, t is a monotone function. Conversely, the unit cost function $u : \mathbb{N} \rightarrow \mathbb{R}_+$ is antitone (Fig. 13.1).

Figure 13.1.:

add picture

Figure 13.2.:

add picture

Example 13.5 (Rounding functions).

to write

Example 13.6 (Cardinality map). In Example 11.8 we presented the poset arising from the power set of a set $A = \{a, b, c\}$ and ordered via subset inclusion.

The map $|\cdot| : \mathcal{P}(A) \rightarrow \mathbb{N}$ (cardinality), is a monotone map (Fig. 13.3).

Lemma 13.7. Consider a discrete poset A and a poset B . Any map $f : A \rightarrow B$ is monotone.

Proof. Since A is a discrete poset, one has

$$a_1 \leq_A a_2 \iff a_1 = a_2. \quad (13.1)$$