

**Applied
Compositional
Thinking
for
Engineers**

Volume 1



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

applied-compositional-thinking.engineering

Please visit the website to get the last updated edition.

Compiled on June 3, 2021.

Volume Class

Volume Class	iii
1. Exercises	1
1.1. Introduction	1
1.2. Code exercises	2
Prerequisites	2
Sign up on Github	2
Option 1: Using Github Classroom	2
Option 2: Independent use without Github Classroom	3
Docker installation	3
1.3. Exercise tutorial: Finite sets	4
Walk-through for first exercise	6
1.4. Turning in exercises	9
Support	9
2. Videos of the class	11
3. Solutions to selected exercises	35
A. E PLURIBUS	37
4. Putting things together	39
4.1. Stacking blocks	39
4.2. Mixing colors	43
4.3. Commutativity and associativity	46
4.4. Composing recipes	48
4.5. Isomorphisms	49
5. Sets and functions	53
5.1. Sets	53
5.2. Functions	54
5.3. Code exercises	55
Properties	55
Mappings	55
Set products	56
Set union \star	58
6. Unus Pro Omnibus, Omnes Pro Uno	61
6.1. Semigroups	62
Induced n -ary multiplication	66
Code exercises	66

6.2.	Monoids	69
6.3.	Groups	73
7.	Morphisms	77
7.1.	Semigroup Morphisms	77
7.2.	Monoid morphisms	81
7.3.	Group morphisms	82
8.	Actions	85
8.1.	Matrix groups	85
	Special Euclidean group	86
8.2.	Actions	87
9.	Solutions to selected exercises	91
 B. POINTS AND ARROWS		 95
10.	(Semi)categories	97
10.1.	Interfaces	97
10.2.	Formal definition of (semi) category	98
11.	Dependencies	101
11.1.	Design	101
12.	Transmutation	107
12.1.	Currency categories	107
13.	Culture	111
13.1.	Definitional impetus vs. computational goals	111
13.2.	Things that don't matter	112
	Typographical conventions don't matter	113
14.	Connection	117
14.1.	Mobility	117
14.2.	Trekking in the Swiss Mountains	119
15.	Relation	121
15.1.	Distribution networks	121
15.2.	Relations	123
15.3.	Relations and functions	125
15.4.	Properties of relations	126
15.5.	Endorelations	127
15.6.	Relational Databases	128
15.7.	Exercises	130
16.	Mapping	131
16.1.	Databases, sets, functions	131

16.2.	Functions as relations	133
16.3.	The Category Set	134
16.4.	Constructing sets	135
16.5.	Exercises	135
17.	Processes	137
17.1.	Moore machines	137
17.2.	Action on sequences	140
17.3.	Other machines	141
17.4.	Action of a category	144
17.5.	Procedures	145
17.6.	Propositions	145
18.	Graphs	147
18.1.	Graphs	147
18.2.	Graph homomorphisms	148
19.	Solutions to selected exercises	149
 C. ORDER		 151
20.	Trade-offs	153
20.1.	Trade-offs	153
20.2.	The 3 diagrams	154
Trade-offs for the human body	154	
Masks	155	
Hats and headphones	156	
20.3.	Ordered sets	158
20.4.	Chains and Antichains	163
20.5.	Upper and lower sets	164
20.6.	From antichains to uppersets, and viceversa	165
Measuring posets	168	
20.7.	Lattices	169
21.	Poset constructions	173
21.1.	Product of posets	173
21.2.	Disjoint union of posets	174
21.3.	Opposite of a poset	175
21.4.	Poset of intervals	175
21.5.	A different poset of intervals	175
22.	Life is hard	177
22.1.	Monotone maps	177
22.2.	Compositionality of monotonicity	179
22.3.	The category Pos of posets and monotone maps	180
Why Pos is not sufficient for design theory	180	

23. Solutions to selected exercises	183
D. COMBINATION	185
24. Combination	187
24.1. Products	187
24.2. Coproducts	195
24.3. Other examples	200
Product and coproduct for power set	200
Product and coproduct for logical sequents	201
24.4. Biproducts	201
24.5. Occultism	201
25. Sameness	203
25.1. Sameness in category theory	203
25.2. Isomorphism is not identity	206
26. Universal properties	207
26.1. Universal properties	207
27. Constructing categories	209
27.1. Opposite Category	209
27.2. Generating categories from graphs	210
27.3. Intervals as categories	211
Twisted arrow category	211
Arrow category	211
27.4. Arrow construction	211
27.5. Slice construction	211
28. Solutions to selected exercises	213
E. THICKER ARROWS	215
29. Functors	217
29.1. Modeling translations	217
29.2. Functors	218
29.3. A poset as a category	219
Monotone maps are functors	220
29.4. Other examples of functors	220
The list functor	220
Planning as the search of a functor	220
29.5. Categorical Databases	221
30. Specialization	223
30.1. Notion of subcategory	223

30.2. Drawings	224
30.3. Other examples of subcategories in engineering	225
30.4. Subcategories of Berg	226
30.5. Embeddings	227
31. Up the ladder	229
31.1. Functor composition	229
31.2. A category of categories	230
31.3. Full and faithful functors	230
31.4. Forgetful functor	231
32. Naturality	233
32.1. Natural transformations	233
33. Adjunctions	237
33.1. Galois connections	237
33.2. An example	240
33.3. Adjunctions: hom-set definition	240
33.4. Adjunctions: (co)unit definition	241
33.5. Example of a “Product-Hom” adjunction	242
33.6. Example of a “Free-Forgetful” adjunction	243
33.7. Relating the two definitions	245
34. Solutions to selected exercises	247
F. DESIGN	249
35. Design	251
35.1. What is “design”?	251
35.2. What is “co-design”?	252
“Co” for “compositional”	252
“Co” for “collaborative”	252
“Co” for “computational”	253
“Co” for “continuous”	253
35.3. Basic concepts of formal engineering design	254
35.4. Queries in design	255
36. Design problems	257
36.1. Design Problems	257
Mechatronics	259
Geometrical constraints	260
Inference	260
Communication	261
Multi-robot systems	261
Computation	262
Other examples in minimal robotics	263

36.2. Queries	266
36.3. The semi-category DPI	267
36.4. Co-design problems	269
36.5. Discussion of related work	274
Theory of design	274
Partial Order Programming	274
Abstract interpretation	274
37. Feasibility	277
37.1. Design problems as monotone maps	277
37.2. Querying design problems	280
37.3. Series composition of design problems	282
37.4. The category of design problems	285
37.5. DP Isomorphisms	286
38. Profunctors	287
38.1. Profunctors	287
38.2. Hom Profunctor	288
38.3. Other examples of profunctors	289
38.4. The bicategory of profunctors	289
38.5. DPI as profunctors	289
39. Parallelism	291
39.1. Modeling parallelism	291
39.2. Monoidal categories	291
39.3. DPI is a monoidal category	298
DPI is a symmetric monoidal category	300
39.4. Pre-monoidal categories	302
39.5. Monoidal functors	302
39.6. Dualizable objects	303
40. Feedback	307
40.1. Feedback in design problems	307
40.2. Feedback examples	307
40.3. Feedback in category theory	307
41. Ordering design problems	313
41.1. Ordering DPs	313
41.2. Restrictions and alternatives	314
Union of Design Problems	314
In DPI	315
Intersection of Design Problems	315
In DPI	316
41.3. Interaction with composition	316
42. Constructing design problems	319
42.1. DPIs are spans	319

42.2. Companions and conjoint	320
43. Solutions to selected exercises	323
G. COMPUTATION	325
44. From math to implementation	327
44.1. From math to implementation	327
45. Solving	329
45.1. Problem statement	329
Expressivity of MCDPs	330
Approach	331
45.2. Computational representation of DPis	331
45.3. Composition operators for design problems	332
45.4. Monotonicity as compositional property	333
45.5. Monotonicity and fixed points	336
45.6. Solution of MCDPs	338
Example: Optimizing over the natural numbers	339
Guarantees of Kleene ascent	341
45.7. Extended Numerical Examples	341
Language and interpreter/solver	341
Model of a battery	343
Competing battery technologies	343
Introducing other variations or objectives	343
From component to system co-design	345
45.8. Complexity of the solution	350
Complexity of fixed point iteration	350
Relating complexity to the graph properties	350
Considering relations with infinite cardinality	351
45.9. Decomposition of MCDPs	351
Equivalence	352
Plumbing	352
45.10. Related work	354
46. Generalized computation	355
46.1. Generalized objects and operations	355
Modeling nondeterministic uncertainty	355
Modeling interval uncertainty	357
Upper sets	359
Keeping track of resource usage	359
46.2. Monads	360
46.3. The Kleisli construction	364
46.4. Algebras of a monad	364
46.5. Monads, computer science definition	368

47. Uncertainty	369
47.1. Introduction	369
47.2. UPos and LPos categories	369
47.3. From DP to UPos and LPos	373
47.4. \mathcal{L} and \mathcal{U} monads	373
48. Solutions to selected exercises	381
49. Enrichments	383
49.1. Enriched categories	383
50. Operads	387
BACK MATTER	391
Solutions to selected exercises	393
Nomenclature	395
Tables and maps	407
References	407
Example exams	415



1. Exercises

1.1. Introduction

This book contains three types of exercises:

- ▷ Solved theoretical exercises, such as Exercise 1 below. These are solved at the end of the part.
- ▷ Graded theoretical exercises, such as Graded exercise 1 below. You will not find the solutions in the book.
- ▷ Graded code exercises, such as Code exercise 1 below. These are autograded and the solution is not publicly available.

Remark 1.1. Instructors: if you want to teach from this book, we are happy to provide the solutions to you.

Exercise 1. Can you solve this riddle?

*Thirty white horses on a red hill,
First they champ,
Then they stamp,
Then they stand still.*

What are they?

1.1 Introduction	1
1.2 Code exercises	2
Prerequisites	2
Sign up on Github	2
Option 1: Using Github Classroom2	
Option 2: Independent use without Github Classroom	3
Docker installation	3
1.3 Exercise tutorial: Finite sets	4
Walk-through for first exercise	6
1.4 Turning in exercises	9
Support	9

See solution on page 35.

Graded exercise 1. This is a graded exercise.

Code exercise 1. This is a code exercise.

You will have the chance to hand in the graded and code exercises, following the procedure described in Section 1.4.

1.2. Code exercises

Prerequisites

You will need the following skills:

- ▷ Python programming. If you want to learn Python, we suggest you look at [EdX's courses](#).
- ▷ Git / Github usage. If you want to learn Git and Github, go to [try.github.io](#).

Additionally, it might be useful to have some idea of what Docker is. In theory you don't need to know anything about Docker, if everything goes well in our autograding scripts.

Sign up on Github

Sign up on Github if you haven't already.

Option 1: Using Github Classroom

If you are following a class where the instructors are using Github Classroom, this is the section for you. Otherwise, jump to Section 1.2.

Get invite link and accept assignment

Github Classroom is a product that helps coordinate assignments for large classes. Your instructors will send you an invite link to join the classroom. An invite link is similar to this:

<https://classroom.github.com/a/XXXXXXX>

If you are not enrolled in an official class, then you can get the invite link on Zulip.

In any case, do not pass invite links around: there are different “classrooms”. Do not use old links because the classroom is refreshed after each season.

Clone the repository

Once you go through the procedure, you will have created a repository

```
https://github.com/INSTRUCTORS/solution-YOU
```

where INSTRUCTORS is the Github organization for the instructors, and you is your username.

Clone the repository just created and jump into the directory.

You should see the content as in Fig. 1.1.

In the following we list two options for you to be able to use the ACT4E interfaces when you code.

Interfaces

The next step is to make sure that you can use the “interfaces repository”. That repository contains all the Python snippets you see in this book. Your IDE needs it so that it can help you write the correct code.

For this, we offer two options. You can clone the “interfaces” repository (from within your repository):

```
https://github.com/ACT4E/ACT4E-exercises.git
```

and in there do

```
python setup.py develop
```

Alternatively, you can directly install the python package:

```
pip install ACT4E-exercises
```

In both cases, please pull (in case you clone the repository) or update (in case you install the package) weekly, to always use the most updated version.

Option 2: Independent use without Github Classroom

If you are not following a class where the instructors are using Github Classroom, this is the section for you. Otherwise, jump to Section 1.2.

In this section we assume that you are not part of a class.

In that case, you can fork directly the repository

```
https://github.com/ACT4E/ACT4E-exercises-template
```

Note that there might be different branches to choose from. The instructions in this book correspond to the branch `spring2021`.

You can do the exercises on your own by using this repository.

In the following, ignore any other mention of Github Classroom.

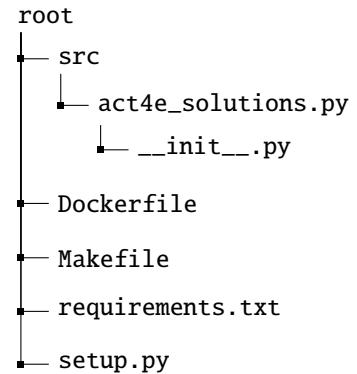


Figure 1.1.: Original content of the exercise template repository.

Docker installation

Install Docker following the information at

```
https://docs.docker.com/get-docker/
```

Inside your repository, do

```
make pull
```

before starting your coding session. This will make sure that you are using the most updated version of the infrastructure provided by us.

1.3. Exercise tutorial: Finite sets

During the exercises, you are going to build in Python a library to build and manipulate most of the concepts in the book. As a first step, we are going to build together the support for sets.

The exercises will give you the *interface* of an *abstract class* to be generated.

Let's define three interfaces:

- ▷ `Setoid`, for a generic set;
- ▷ `EnumerableSet`, for a set that can enumerate its elements;
- ▷ `FiniteSet`, for a set that has a finite set of elements.

They are defined in the package `act4e_interfaces` which is in the repository ACT4E/ACT4E-interfaces.

Figure 1.2 shows their relation in an UML diagram. The arrow means “inherits from”.

Listing 1 shows the interface for `Setoid`.

Listing 1: The `Setoid` interface.

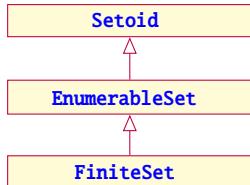


Figure 1.2.: UML inheritance diagram

```

class Setoid(ABC):
    """
    A setoid is something to which elements may belong,
    has a way of distinguishing elements,
    and is able to (de)serialize its elements.
    """

    @abstractmethod
    def contains(self, x: Element) -> bool:
        """ Returns true if the element is in the set. """

    def equal(self, x: Element, y: Element) -> bool:
        """ Returns True if the two elements are to be considered equal. """
        return x == y # default is to use the Python equality

    def apart(self, x: Element, y: Element) -> bool:
        return not self.equal(x, y)

    @abstractmethod
    def save(self, h: IOHelper, x: Element) -> ConcreteRepr:
        ...

    @abstractmethod
    def load(self, h: IOHelper, o: ConcreteRepr) -> Element:
        ...
  
```

The class `Setoid` derives from `ABC`, which means *abstract base class*; it tells the interpreter that some of the methods of the class will have to be implemented by the subclasses. This is the equivalent of the “virtual” methods in C++. The

methods that need to be implemented are marked by a decorator `@abstractmethod`. For `Setoid`, we ask that the implementer implements the `contains()` function, and they can optionally implement `equal()`.

This is the semantics of the first three methods:

- ▷ The method `contains()` checks if an element is part of this set.
- ▷ The method `equal()` checks if two elements are the same. The default way to do this is to use Python's equality operator (which calls the `__eq__` method of the class of the first object); later we will see cases in which we might want to choose something different.
- ▷ The method `apart()` checks that two elements are “apart”. You can ignore this for the entire first volume, but it will become more relevant later when we talk more in detail about constructivism.[†]

There are other methods that have to do with serializing/deserializing the data:

- ▷ The method `save()` saves an object into a “concrete representation” `ConcreteRepr`. By that we mean something that can be serialized to YAML.
- ▷ The method `load()` does the inverse.

Note that YAML supports the following types: `int`, `bool`, `float`, `datetime`, , `dict`, `list`. Most of Python values are representable in YAML. The exception is class instances. For that you need to come up with a way to represent the data. For example, you might have the class `Vector`:

```
@dataclass
class Vector:
    x: int
    y: int
    z: int

a = V(1,2,3)
```

You might choose one of these possible variations for serialization:

Serialize as array [1, 2, 3]

Serialize as dictionary {'x': 1, 'y': 2, 'z': 3}

The `Setoid` is the one that knows how to serialize its elements.

The code contains also typing annotations. The symbol `Element`, like the others we will see later, such as `Object` and `Morphism`, are simple aliases to Python's `object` class; which means that they are not specifying the type at all. However, they are useful for intuition. The latest versions of Python have supported typing annotations with generics, so we could have chosen to use those and have more accurate typing, however we want these exercises to be accessible to all.

Listing 2 shows the interface for `EnumerableSet`:

Listing 2: The `EnumerableSet` interface.

```
class EnumerableSet(Setoid, ABC):
    @abstractmethod
    def elements(self) -> Iterator[Element]:
        """ Note: possibly non-terminating. """
```

The class inherits from `Setoid` and adds one methods:

[†] See [Apartment relation](#) on Wikipedia.

- ▷ The method `elements()` returns a Python `Iterator`: this is something that you can iterate.

Listing 3 shows the interface for `FiniteSet`.

Listing 3: The `FiniteSet` interface.

```
class FiniteSet(EnumerableSet, ABC):
    """ A finite set has a finite size. """

    @abstractmethod
    def size(self) -> int:
        """ Return the size of the finite set. """
```

The additional method:

- ▷ The method `size()` returns the size of the set.

You will have to implement the `FiniteSet` class as part of the exercise.

We have defined formats for all the structures to be used in the exercises. The file format for `FiniteSet` is shown in Fig. 1.3. The file represents a dictionary with only one field called `elements`, which contains a list of elements. YAML can represent most primitive data format of Python, as well as lists and dictionaries. We visualize the data samples in the book using YAML, but you never have to deal with YAML yourself. You can expect to receive a Python data structure as in Fig. 1.4.

elements: [a, b, c]

Figure 1.3.: Example shown in YAML format.

{ "elements": ["a", "b", "c"] }

Figure 1.4.: We show data formats in YAML because it is terse, but in the exercises you will receive the Python data structure directly.

Code exercise 2 (`TestFiniteSetRepresentation`). The first exercise is to implement the methods that read and write from this file format. You have to implement the interface `FiniteSetRepresentation` shown in Listing 4. As part of the exercise, you will also have to implement a concrete subclass of `FiniteSet`.

Listing 4

```
class FiniteSetRepresentation(ABC):
    @abstractmethod
    def load(self, h: IOHelper, data: FiniteSet_desc) -> FiniteSet:
        """Load a finite set from data structure.
        Throw InvalidFormat if the format is incorrect.
        """
    @abstractmethod
    def save(self, h: IOHelper, f: FiniteSet) -> FiniteSet_desc:
        """Serializes into a data structure """
```

For now, you can ignore the argument `h: IOHelper`. That is an interface we will need later when serializing large objects.

Walk-through for first exercise

We are going to solve this exercise together.

We assume that you have checked out the repository as explained in the setup section.

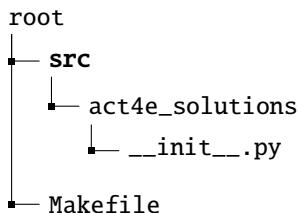


Figure 1.5.: Relevant files for the moment

Of all the files in the repository, we only need the ones displayed in Fig. 1.5.

We can start by checking if this is a valid solution. There is a recipe for this in the `Makefile`. Type this:

```
make check-TestFiniteSetRepresentation
```

Here `TestFiniteSetRepresentation` is the code name for the exercise. You can check other exercises by changing the name.

This will fail; it will complain saying that it didn't even find any code implementing the exercise.

Create a file called `first.py` in the `src/act4e_solutions` directory. This file will contain the implementation of the classes.

For now we are just going to add non-functional code for the classes, as in Listing 5.

Listing 5

```
from typing import Collection, Iterator, List

import act4e_interfaces as I

class MyFiniteSet(I.FiniteSet):
    _elements: List[I.Element]

    def __init__(self, elements: Collection[I.Element]):
        ... # to implement

    def size(self) -> int:
        ... # to implement

    def contains(self, x: I.Element) -> bool:
        ... # to implement

    def elements(self) -> Iterator[I.Element]:
        ... # to implement

    def save(self, h: I.IOHelper, x: I.Element) -> I.ConcreteRepr:
        ... # to implement

    def load(self, h: I.IOHelper, o: I.ConcreteRepr) -> I.Element:
        ... # to implement

class MyFiniteSetRepresentation(I.FiniteSetRepresentation):
    def load(self, h: I.IOHelper, data: I.FiniteSet_desc) -> I.FiniteSet:
        ... # to implement

    def save(self, h: I.IOHelper, f: I.FiniteSet) -> I.FiniteSet_desc:
        ... # to implement
```

Now modify `__init__.py` to import all the symbols from the `first` module.

```
from .first import *
```

At this point you can check again by typing:

```
make check-TestFiniteSetRepresentation
```

The program will tell you that it found the code supposedly implementing the function, but that the tests failed. All of our functions returned `None`.

At this point we need to implement the rest of the code.

For example, Listing 6 is a valid implementation of `FiniteSet`.

Listing 6

```
from typing import Collection, Iterator

import act4e_interfaces as I

class MyFiniteSet(I.FiniteSet):

    def __init__(self, elements: Collection[I.Element]):
        self._elements = list(elements) # new list

    def size(self) -> int:
        return len(self._elements)

    def contains(self, x: I.Element) -> bool:
        for y in self._elements:
            if self.equal(x, y):
                return True
        return False

    def elements(self) -> Iterator[I.Element]:
        # return a copy of our list
        for _ in self._elements:
            yield _

    def save(self, h: I.IOHelper, x: I.Element) -> I.ConcreteRepr:
        return x

    def load(self, h: I.IOHelper, o: I.ConcreteRepr) -> I.Element:
        return o
```

The loading and saving of the data is very simple. To load just take the `elements` field of the data structure and give it to `MyFiniteSet`. To save, reconstruct the list elements from the `MyFiniteSet` instance by using the method `elements()`, which returns an iterator. Note that the `save()` function you have to implement must work for all `FiniteSet` instances, not just your particular implementation of `FiniteSet`; therefore, you cannot access the `_elements` field directly. An example of this is shown in Listing 7.

Listing 7

```
import act4e_interfaces as I

class MyFiniteSetRepresentation(I.FiniteSetRepresentation):
    def load(self, h: I.IOHelper, data: I.FiniteSet_desc) -> I.FiniteSet:
        if not isinstance(data, dict):
            raise I.InvalidFormat()
        if not "elements" in data:
            raise I.InvalidFormat()
        if not isinstance(data["elements"], list):
```

```
    raise I.InvalidFormat()
elements = data["elements"]
return MyFiniteSet(elements)

def save(self, h: I.IOHelper, f: I.FiniteSet) -> I.FiniteSet_desc:
    all_elements = [f.save(h, _) for _ in f.elements()]
    return {"elements": all_elements}
```

With this code, the tests should pass. Try again to run:

```
make check-TestFiniteSetRepresentation
```

Note that the types used by the function `save()` of `MyFiniteSetRepresentation` is `FiniteSet`, not `MyFiniteSet`. The method must be able to work for *any* implementation of `FiniteSet`, not just yours. In fact, the tester will try to call that function with a different implementation.

Note that the implementation above uses this line to get the elements:

```
all_elements = sorted(f.elements())
```

This is correct because it is using the method `elements()` that all implementations of `FiniteSet` need to have. What could be wrong is using code like the following:

```
all_elements = sorted(f._elements)
```

This code accesses the attribute `_elements` of the class `MyFiniteSet`. It will not work with other implementations. In fact, the tests will fail.

1.4. Turning in exercises

Via Github Classroom you can easily turn in your exercises. Each graded exercise is characterized by a name (for instance `TestFiniteSetRepresentation`).

To hand in an exercise, simply commit it with the name as the commit message, and push it to your solutions branch.

In the case of code exercises, the push will automatically result in an evaluation, which will assign you points (running the same tests as the ones you can run before pushing).

In the case of theory exercises, a TA will have to correct them manually. For this reason, we kindly ask you to push a single pdf file per exercise, named after the exercise (up to you to choose the style, either written by hand, or in TeX, ...).

Support

Support will happen via Zulip, on the appropriate streams.

2. Videos of the class

Watch online video (5 minutes).

Matrix groups

Groups of square matrices, with some property preserved by composition.

The "Special" versions of the groups are subgroups with determinant 1.

Definition (Special linear group $SL(n)$). The special linear group of order n , written $SL(n)$, is the group of $n \times n$ invertible matrices with determinant equal to 1.

Definition (Special orthogonal group $SO(n)$). The special orthogonal group of order n , written $SO(n)$, is the group of $n \times n$ square matrices that satisfy

$$M^T M = I$$

and $\det(M) = 1$.

Definition (Special Euclidean group $SE(n)$). The special Euclidean group of order n , written $SE(n)$, is the group of $(n+1) \times (n+1)$ square matrices of the form

$$\begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

where $R \in SO(n)$ and $t \in \mathbb{R}^n$.



Watch online video (33 minutes).

More rewriting

Definition (Semigroup left action (preliminary version)). A semigroup left action of a semigroup S onto a set A is

$$Lact : S \rightarrow \text{End}(A)$$

such that, for all $a \in A$,

$$Lact(x, Lact(y, a)) = Lact(x \cdot y, a).$$

Now rewrite the condition as equality of functions

$$Lact(x) \circ_{\text{End}(A)} Lact(y) =_{\text{End}(A)} Lact(x \cdot y)$$

- This is the semigroup morphism condition.

Definition (Semigroup left action). A semigroup left action of a semigroup S onto a set A is a semigroup morphism

$$Lact : S \rightarrow \text{End}(A).$$

Definition (Semigroup right action). A semigroup right action of a semigroup S onto a set A is a semigroup morphism

$$Ract : S^* \rightarrow \text{End}(A).$$



Watch online video (25 minutes).

Defining the semi-category of Moore machines

$f : (U_f, X_f, Y_f, \text{dyn}_f, m_f, \text{start}_f) \xrightarrow{\text{equal}} (U_g, X_g, Y_g, \text{dyn}_g, m_g, \text{start}_g)$

$U_f \cong U_g$
 $X_f \cong X_g \times X_g$
 $\text{start}_f \cong (\text{start}_g, \text{start}_g)$

$Y_f \cong Y_g$
 $\text{dyn}_f : U_f \times (X_f \times X_g) \longrightarrow (X_f \times X_g)$

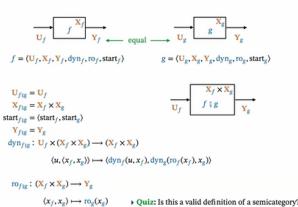
$(u, (x_f, x_g)) \mapsto (\text{dyn}_f(u, x_f), \text{dyn}_f(v, x_g), x_g)$

$m_f : (X_f \times X_g) \longrightarrow Y_g$
 $(x_f, x_g) \mapsto m_f(x_g)$



Watch online video (19 minutes).

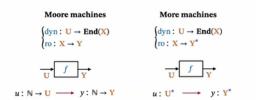
Defining the semi-category of Moore machines



Watch online video (5 minutes).

More machines

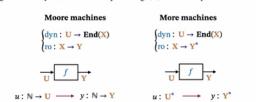
Let's give the ability to the machine of producing 0, 1, or more output.



Watch online video (3 minutes).

More machines

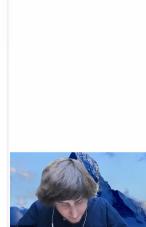
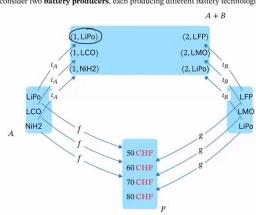
Let's give the ability to the machine of producing 0, 1, or more output.



Watch online video (13 minutes).

Coproduct

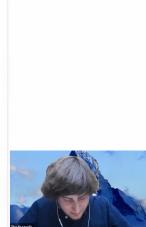
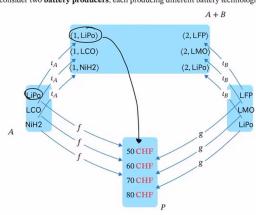
Let's consider two battery producers, each producing different battery technologies



Watch online video (8 minutes).

Coproduct

Let's consider two battery producers, each producing different battery technologies



Watch online video (1 minutes).

Coproduct

Definition (Coproduct). Let \mathbf{C} be a category and let $X, Y \in \text{Ob}_{\mathbf{C}}$ be objects. The coproduct of X and Y is:

1. an object $Z \in \text{Ob}_{\mathbf{C}}$ ("the coproduct" of X and Y)
 2. injection morphisms $i_1 : X \rightarrow Z$ and $i_2 : Y \rightarrow Z$
- Conditions**
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 $\hat{f} \circ g : Z \rightarrow T$ such that $f = i_1 \circ \hat{f} \circ g$ and $g = i_2 \circ \hat{f} \circ g$.

- Via commuting diagram:

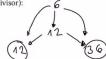
$$\begin{array}{ccc} & T & \\ f \swarrow & \downarrow i_1 \circ f \circ g & \searrow g \\ X & \xrightarrow{i_1} & X+Y & \xleftarrow{i_2} & Y \end{array}$$



Watch online video (4 minutes).

Product vs. Coproduct: Example

- Let $A, B \in \mathbb{N}$
- We draw an arrow between A and B if A divides B . For instance $6 \rightarrow 12$



Watch online video (43 minutes).

Upper / lower closure

- The **upper closure** of an element is the set of elements that dominate it:
 $\uparrow a = \{x \in P : a \leq x\}$

- Upper closure is a **monotone function**:

$$\begin{array}{c} \uparrow \\ (P, \leq_P) \xrightarrow{\quad \uparrow \quad} (\{a\}, \supseteq) \\ \frac{a \leq_P b}{\uparrow a \supseteq \uparrow b} \end{array}$$



Watch online video (13 minutes).

Form ever follows function

Whether it be the sweeping eagle in his flight, or the open apple-blossom, the tellingly work-horse, the little sparrow, the branching oak, the winding stream at its base, the living creature, the dead thing, the plant, the mineral—each has its form. This is its life. Where function does not change, form does not change. The granite rocks, the over-bounding hills, remain for ages; the lightning lives, comes into shape, and, dies, in a moment.

It is the pervading law of all things organic and inorganic, of all things physical and metaphysical, of all things human and all things超自然的, of all manifestations of the law of the heart, of the soul, that life is recognizable in its expression, that form ever follows function. This is the law.

Louis Sullivan

"function" conflates two dual aspects



Watch online video (11 minutes).

Defining composition for boolean profunctors

- Definition of composition:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ f : A^{\text{op}} \times B \rightarrow_{\text{Pos}} \text{Bool} & & g : B^{\text{op}} \times C \rightarrow_{\text{Pos}} \text{Bool} \\ & \swarrow \quad \searrow & \\ & \text{A} & \end{array}$$



Watch online video (2 minutes).

What is design?

- We take a broad view of what design is, the same as Herbert Simon:

Engineering is a field of professional design.
Everyone designs who devises solutions in 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 strategy for a company or the one that designs a bridge for a river. Design, as we shall see, is the core of all professional training; it is the principal mark that distinguishes the profession 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.

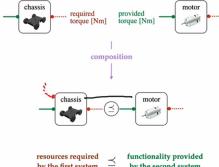
Herbert Simon, The sciences of the artificial, Chapter 5




Watch online video (4 minutes).

Composition semantics

- Composition in DIP reflects an intuitive notion of composition in engineering:



resources required by the first system \leq functionality provided by the second system

Herbert Simon, The sciences of the artificial, Chapter 5



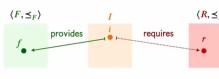

Watch online video (7 minutes).

Design problem with implementation (DPIs)

Definition (Design problem with implementation). A design problem with implementation (DPI) is a tuple $(F, R, I, prov, req)$

where:

- F is a poset, called *functionality space*;
- R is a poset, called *requirement space*;
- I is a poset, called *implementation space*;
- the map $prov : I \rightarrow F$ maps an implementation to the functionality it provides;
- the map $req : I \rightarrow R$ maps an implementation to the resources it requires;



(F, \leq_F) provides i \vdash I requires r \vdash (R, \leq_R)

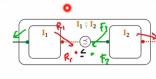
Herbert Simon, The sciences of the artificial, Chapter 5




Watch online video (5 minutes).

DPI composition

Definition (DPI composition). The series composition of two DPs

$$\begin{aligned} dp_1 &= (F_1, R_1, I_1, prov_1, req_1), & I_1 \\ dp_2 &= (F_2, R_2, I_2, prov_2, req_2), & I_2 \\ \text{for which } F_2 = R_1 \text{ is} & & I_1, I_2, I_1 \cup I_2 = I_1 \cup I_2 \\ (dp_1 \circ dp_2) &:= (F_1, R_2, I_1 \cup I_2, prov_1 \circ prov_2, req_1 \circ req_2), & I_1 \cup I_2 \\ I &= \{i_1 | i_2 \in (I_1 \cup I_2) \text{ and } req_2(i_2) \leq_{R_2} prov_1(i_1)\}, & i_1 \leq_{I_1} i_2 \\ prov &: I \rightarrow F, & i_1 \leq_{I_1} i_2 \mapsto prov_1(i_1), \\ req &: I \rightarrow R, & i_1 \leq_{I_1} i_2 \mapsto req_2(i_2). \end{aligned}$$


$I_1 \cup I_2 = I$

Herbert Simon, The sciences of the artificial, Chapter 5




Watch online video (2 minutes).

From DPI to feasibility relation

- From a DPI we can find a monotone function

$$r: F^{\text{op}} \times R \rightarrow \text{Bool}$$

defined as follows:

$$(f^{\text{op}}, r) \iff \exists i \in I : (f \leq_F \text{provides}(i)) \wedge (\text{requires}(i) \leq_R r)$$

Alternative: first define the set of all feasible implementations:

$$(f^{\text{op}}, r) \iff \{i \in I : (f \leq_F \text{provides}(i)) \wedge (\text{requires}(i) \leq_R r)\}$$

then ask if non empty.

Herbert Simon, The sciences of the artificial, Chapter 5

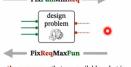



Watch online video (3 minutes).

Design queries

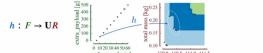
- FixFunMinReq: Fix a lower bound on functionality, minimize the resources.
- FixReqMaxFun: Fix an upper bound on the resource, maximize the functionality

Given the functionality to be provided, what are the minimal resources required?



Given the resources that are available, what is the maximal functionality that can be provided?

- We are looking for a map from functionality to upper sets of feasible resources:



Watch online video (2 minutes).

Upper / lower closure

- The upper closure of an element is the set of elements that dominate it:

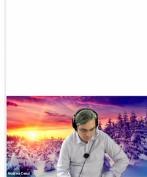
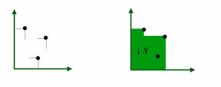
$$\uparrow a = \{x \in P : a \leq x\}$$

- Upper closure of a set:

$$\uparrow S = \bigcup_{a \in S} \uparrow a$$



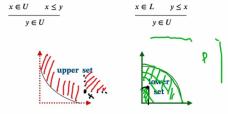
- Lower closure is defined symmetrically:



Watch online video (4 minutes).

Upper/lower sets

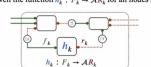
- An upper set is a subset of a poset such that, if x is in the poset, all "higher" elements are.
- A lower set is a subset of a poset such that, if x is in the poset, all "lower" elements are.



Watch online video (39 minutes).

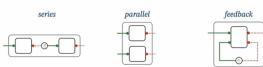
Solving DP queries

- Suppose we are given the function $h_k : F_k \rightarrow AR_k$ for all nodes in the co-design graph.



- Can we find the map $h : F \rightarrow AR$ for the entire diagram?

- Recursive approach based on functionality: We just need to work out the composition formulas for all operations we have defined.



Watch online video (7 minutes).

Reducing to a normal form

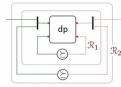
- Reducing to a normal form



Watch online video (6 minutes).

What about multiple loops?

- Generally speaking, a fixed point iteration will converge at the ω -th step, where ω is the first infinite ordinal - a countable number of steps.
- But if we close 2 loops, we need to compute a fixed point of a fixed point: this will take ω^2 steps.

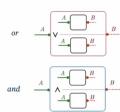


Watch online video (10 minutes).

Introducing new ways of composing

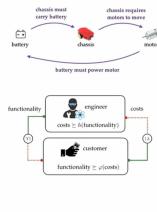
- If you have a compositional/functorial solution framework, you can easily introduce new ways of composing models in a modular way:
 - Introduce the new composition on the category.
 - Extend the solution functor on the new type of composition.

Example: introducing "and" and "or" for DP.



Watch online video (6 minutes).

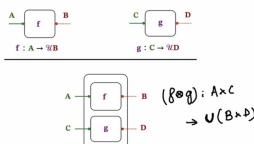
What about loops?



Watch online video (3 minutes).

Parallel composition for DP

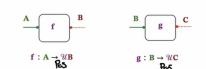
- Definition of composition:



Watch online video (8 minutes).

Series composition for DP

- Definition of composition:



$$(f \circ g)(a) = \bigcup_{b \in g(a)} f(b)$$

Below this, a diagram illustrates the execution flow: an arrow from A to B is labeled f , an arrow from B to C is labeled g , and an arrow from A to C is labeled $(f \circ g)$. The intermediate states $b \in g(a)$ are shown as green dots along the path from A to B.



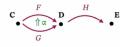
Watch online video (3 minutes).

Big picture

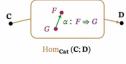
- In a category, **morphisms** are arrows between objects.



- Functors** are arrows between categories in a category of categories called **Cat**.



- Natural transformations** are arrows between functors with same domain/codomain.



Watch online video (19 minutes).

Semi-category action

Definition (Semi-category action). A semi-category action of a semi-category \mathbf{C} is defined by

- a map φ that associates from each object $X \in \text{Ob}_{\mathbf{C}}$, a set $\varphi(X)$:

$$\varphi : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\text{Set}}$$

- a map γ that associates to each morphism a function:

$$\gamma : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\text{Set}}(\varphi(X); \varphi(Y))$$

Moreover, this condition must hold:

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



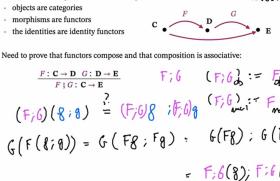
- A semi-category action is a functor to **Set**.



Watch online video (6 minutes).

A category of categories

- There exists a category of (small) categories called **Cat**:
- objects are categories
- morphisms are functors
- the identities are identity functors



$$G(f(l; g)) = G(Fg; fg) = G(Fg) \circ G(fg)$$

$$G(f(l; g)) = G(Fg; fg) = G(Fg) \circ G(fg)$$

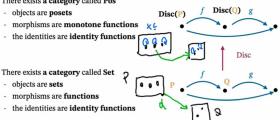
$$= F_g(g(l; g)) = F_g(g(l; g)) = F_g(g(l; g))$$



Watch online video (7 minutes).

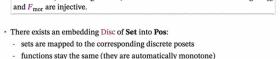
Embedding Set in Pos

- There exists a category called **Pos**
- objects are posets
- morphisms are monotone functions
- the identities are identity functions



Definition (Embedding functor). A functor $F : \mathbf{C} \rightarrow \mathbf{D}$ is an embedding if F_{ob} and F_{mor} are injective.

- There exists an embedding Disc of **Set** into **Pos**:
- sets are mapped to the corresponding discrete posets
- functions stay the same (they are automatically monotone)



Watch online video (2 minutes).

Semi-category action

Definition (Semi-category action). A semi-category action of a semi-category \mathbf{C} is defined by

- a map φ that associates from each object $X \in \text{Ob}_{\mathbf{C}}$, a set $\varphi(X)$:

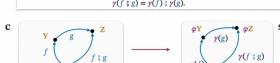
$$\varphi : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\text{Set}}$$

- a map γ that associates to each morphism a function:

$$\gamma : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\text{Set}}(\varphi(X); \varphi(Y))$$

Moreover, this condition must hold:

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



- A semi-category action is a functor to **Set**.



Watch online video (2 minutes).

Semigroup morphisms

Definition (Semigroup morphism). A morphism $F : S \rightarrow T$ between semi-groups $S = \langle \cdot, \cdot_g \rangle$ and $T = \langle \cdot, \cdot_T \rangle$ is a function $F : S \rightarrow T$ such that for all $x, y \in S$, $F(x \cdot_g y) = F(x) \cdot_T F(y)$.

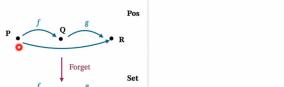
- Semigroup morphisms are semi-functors between semi-categories with 1 object.



Watch online video (3 minutes).

Forgetful functors

- There exists a category called **Pos**
- objects are **posets**
- morphisms are **monotone functions**
- the identities are **identity functions**



- There exists a category called **Set**
- objects are **sets**
- morphisms are **functions**
- the identities are **identity functions**

- A "forgetful functor" **Forget** from **Pos** to **Set** forgets the extra structure:
- each poset is mapped to the underlying set
- each monotone function is mapped to itself (we forget it is monotone)



Watch online video (1 minutes).

Semi-functors

Definition (Semi-functor). A semi-functor $F : \mathbf{C} \rightarrow \mathbf{D}$ between two semi-categories is defined by a map

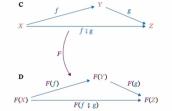
$$F_{\text{Ob}} : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{D}}$$

and, for every pair of objects X, Y , a map

$$F_{\text{Mor}} : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{D}}(F_{\text{Ob}}(X); F_{\text{Ob}}(Y))$$

such that

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{F_{\text{Mor}}(f \circ g) = F_{\text{Mor}}(f) \circ F_{\text{Mor}}(g)}.$$



Watch online video (12 minutes).

Monotone functions as functors

- A function $f : P \rightarrow Q$ between two posets P and Q is monotone iff

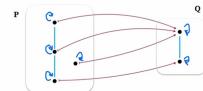
$$\frac{x \leq y}{f(x) \leq f(y)}$$

$$\frac{x \leq y}{x \leq y}$$

- A functor need satisfy two conditions:

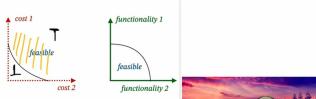
- Identities map to identities: $F_{\text{Mor}}(\text{Id}_X) = \text{Id}_{F_{\text{Ob}}(X)}$

- Composition is respected: $\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{F_{\text{Mor}}(f \circ g) = F_{\text{Mor}}(f) \circ F_{\text{Mor}}(g)}$



Watch online video (2 minutes).

Monotone functions



Watch online video (7 minutes).

Functions

• Recall how to define a function:

$$f : \mathbb{R} \rightarrow \mathbb{R}$$

\downarrow

$a = f(a)$

- Top line: name of function, domain, codomain.

- Second line: a recipe to compute the function.

- " \rightarrow " is pronounced "to", " $=$ " is pronounced "maps to".

$$\begin{array}{c} f : A \rightarrow B \quad x \in A \\ f(x) \in B \end{array} \qquad \begin{array}{c} f : A \rightarrow B \quad x_1 \in A \quad x_2 \in A \quad x_1 = x_2 \\ f(x_1) = f(x_2) \end{array}$$



• Recall that two functions are the same if and only if (iff):

- they have same domain and same codomain.

- they agree on the results for every input in the domain.

$$\begin{array}{c} f : A \rightarrow B \quad g : A \rightarrow B \\ f = g \\ \hline x \in A \\ f(x) = g(x) \end{array}$$

Watch online video (34 minutes).

USASCII code chart

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000 0000	0000 0001	0000 0010	0000 0011	0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111
0000 0010	0000 0011	0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111
0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111



Watch online video (4 minutes).

Group morphisms

Definition (Group morphism). A morphism $F : G \rightarrow H$ between groups

$$G = \langle G, \cdot_G, \text{id}_G, \text{inv}_G \rangle \quad \text{and} \quad H = \langle H, \cdot_H, \text{id}_H, \text{inv}_H \rangle$$

is a function $F : G \rightarrow H$ such that for all x, y in G ,

$$F(x \cdot_G y) = F(x) \cdot_H F(y)$$

• Why is the inverse inv purple?

It's definitely not a group morphism!

$$\text{inv}(x \cdot y) = \text{inv}(y) \cdot \text{inv}(x)$$



Watch online video (7 minutes).

Monoid morphisms

Definition (Monoid morphism). A morphism $F : M \rightarrow N$ between monoids

$$M = \langle M, \cdot_M, \text{id}_M \rangle \quad \text{and} \quad N = \langle N, \cdot_N, \text{id}_N \rangle$$

is a function $F : M \rightarrow N$ such that for all x, y in M ,

$$F(x \cdot_M y) = F(x) \cdot_N F(y)$$

and

$$F(\text{id}_M) = \text{id}_N \quad \text{new}$$



Watch online video (23 minutes).

USASCII code chart

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000 0000	0000 0001	0000 0010	0000 0011	0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111
0000 0010	0000 0011	0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111
0000 0100	0000 0101	0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 0110	0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 0111	0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1000	0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1001	0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1010	0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1011	0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1100	0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1101	0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1110	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111
0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111	0000 1111



Watch online video (9 minutes).

ASCII encoding

- The 7-bit ASCII code is a correspondence between 128 characters (alphabet, numbers, symbols, special characters) and the numbers in $[0, 127]$.
- Let's call this alphabet $\mathcal{A} = \{A, B, C, \dots, a, \dots, 3, \dots, 8, \dots, 7, \dots, \text{DEL}, \dots\}$

Quiz:

Suppose we also include the step of converting numbers to binary digits.

We can then define an **invertible function**
 $\text{ASCII} : \mathcal{A} \rightarrow \{0, 1\}^7$ $\text{ASCII}^{-1} : \{0, 1\}^7 \rightarrow \mathcal{A}$

1. True or false: the function above induces a semigroup morphism.
 $\text{ASCII} : \mathcal{A}^* \rightarrow \{0, 1\}^7$

2. True or false: this function induces a semigroup isomorphism.

$$\text{ASCII}(\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}) = \text{"AF"} \\ \text{ASCII}(\begin{pmatrix} 0 & 0 & 1 & 0 & 1 \end{pmatrix}) = \text{"AM"} \\ \text{ASCII}^{-1}(\begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix})$$



Watch online video (8 minutes).

Morse code

- No, we do not have a semigroup morphism:
 $\text{morse} : (\{A \in Z\} \cup \{0 \text{ to } 9\} \cup \{\#\})^* \rightarrow \{*, s_1, s_2, s_3\}^*$

- Suppose this was possible. Then the message "I AM MAX".

MAX

- We could also decompose it as follows:

I **A** **M** **M** **A** **X**

- Then we would be able to use this translation:

$$\text{morse}(I A) ; \text{morse}(M) ; \text{morse}(I) ; \text{morse}(M) ; \text{morse}(M) ; \text{morse}(AX)$$

$$\text{morse}(M) ; \text{morse}(M)$$

$$\begin{matrix} * & s_1 & s_2 & * & s_3 & s_4 & * & s_5 & s_6 & s_7 & * & s_8 & s_9 & s_{10} \\ * & s_1 & s_2 & * & s_3 & s_4 & * & s_5 & s_6 & s_7 & * & s_8 & s_9 & s_{10} \\ I & A & M & \text{ } & M & A & X & & & & & & & \end{matrix}$$

$$\begin{matrix} * & s_1 & s_2 & * & s_3 & s_4 & * & s_5 & s_6 & s_7 & * & s_8 & s_9 & s_{10} \\ * & s_1 & s_2 & * & s_3 & s_4 & * & s_5 & s_6 & s_7 & * & s_8 & s_9 & s_{10} \\ I & A & M & \text{ } & M & A & X & & & & & & & \end{matrix}$$

- Something to think about: can you find a way to modify the formalization such that Morse encoding is a semigroup morphism?



Watch online video (3 minutes).

Semigroup isomorphisms

Definition (Semigroup isomorphism): A morphism of semigroups $F : S \rightarrow T$ is called a semigroup isomorphism if there exists a morphism of semigroups $G : T \rightarrow S$ such that

$$F \circ G = \text{Id}_S \quad \text{and} \quad G \circ F = \text{Id}_T.$$

- Example: Logarithm and exponential

$$\log(a \cdot b) = \log(a) + \log(b)$$

$$\log : \mathbb{R}_{>0} \rightarrow \mathbb{R}$$

$$\log \circ \exp = \text{Id}_{\mathbb{R}_{>0}} \quad C \quad (\mathbb{R}_{>0}, +) \circlearrowleft \quad (\mathbb{R}, +) \circlearrowleft \quad \exp \circ \log = \text{Id}_{\mathbb{R}}$$

$$\exp : \mathbb{R} \rightarrow \mathbb{R}_{>0}$$

$$\exp(c+d) = \exp(c) \cdot \exp(d)$$



Watch online video (11 minutes).

Composing commutative squares

- Suppose we have commutative squares like this

$$\begin{array}{ccc} T & \xrightarrow{f} & U \\ \downarrow u & \downarrow & \downarrow v \\ V & \xrightarrow{i} & W \end{array} \quad \begin{array}{ccc} U & \xrightarrow{g} & X \\ \downarrow g & \downarrow & \downarrow e \\ W & \xrightarrow{j} & Y \end{array}$$

- We can compose them into one large diagram

$$\begin{array}{ccccc} T & \xrightarrow{f} & U & \xrightarrow{g} & X \\ \downarrow u & \downarrow & \downarrow v & \downarrow & \downarrow e \\ V & \xrightarrow{i} & W & \xrightarrow{j} & Y \end{array}$$

- Quiz: how many paths are there in this diagram from T to Y ?

- Quiz: if the component squares are commutative, is the whole diagram commutative, too? What's your argumentation?



Watch online video (36 minutes).

Functors with the same source and target

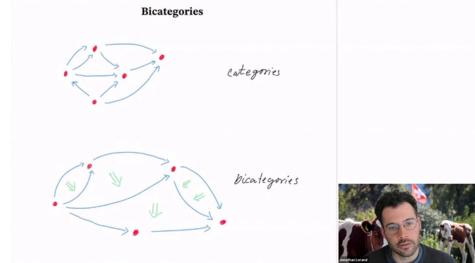
- If F and G are both functors $C \rightarrow D$ we can try to compare them:

- Given a diagram in C , can we relate its image under F to its image under G ?

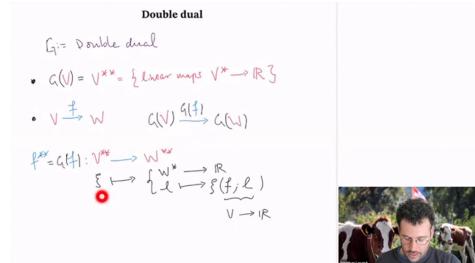
$$\begin{array}{ccc} C & & D \\ \downarrow & \nearrow F & \downarrow \nearrow G \\ X & \xrightarrow{f} & Y \\ \downarrow & \nearrow f_1 & \downarrow & \nearrow f_2 \\ Z & \xrightarrow{g} & W & \xrightarrow{h} & V \\ \downarrow & \nearrow g_1 & \downarrow & \nearrow g_2 & \downarrow & \nearrow h_1 \\ X' & \xrightarrow{f'} & Y' & \xrightarrow{g'} & W' & \xrightarrow{h'} & V' \end{array}$$



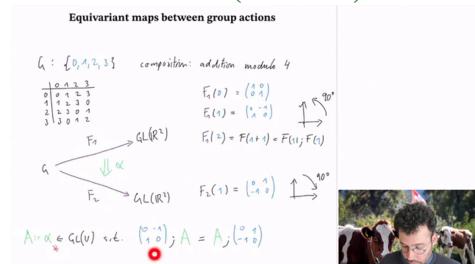
Watch online video (1 minutes).



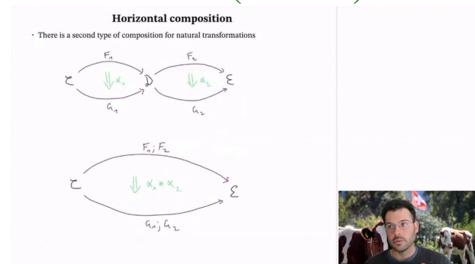
Watch online video (10 minutes).



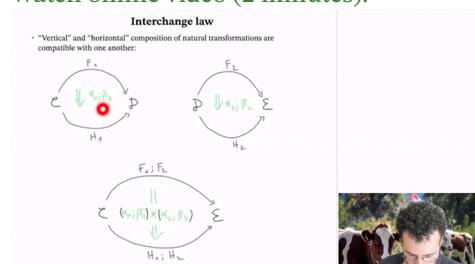
Watch online video (7 minutes).



Watch online video (5 minutes).



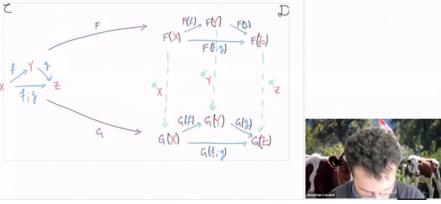
Watch online video (2 minutes).



Watch online video (8 minutes).

 **Functors with the same source and target**

- If F and G are both functors $C \rightarrow D$ we can try to compare them:
- Given a diagram in C , we can relate its image under F to its image under G .




Watch online video (2 minutes).

Natural transformations are morphisms between functors

$F \xrightarrow{\text{def}} \mathcal{C}$ means: $\forall x \in \mathcal{C} \exists \eta_x : F(x) \rightarrow G(x)$ s.t.

$$\forall x \xrightarrow{f} y \quad \begin{array}{c} F(y) \xrightarrow{F(f)} F(y) \\ \eta_x \downarrow \qquad \qquad \downarrow \eta_y \\ G(x) \xrightarrow{G(f)} G(y) \end{array}$$

$G \xrightarrow{\text{def}} \mathcal{H}$ means: $\forall x \in \mathcal{C} \exists \mu_x : G(x) \rightarrow H(x)$ s.t.

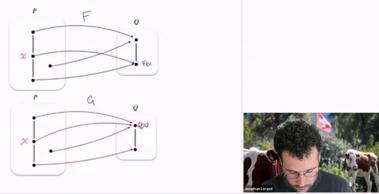
$$\forall x \xrightarrow{g} y \quad \begin{array}{c} G(y) \xrightarrow{G(g)} G(y) \\ \mu_x \downarrow \qquad \qquad \downarrow \mu_y \\ H(x) \xrightarrow{H(g)} H(y) \end{array}$$


Watch online video (3 minutes).

Relating monotone maps

- Posets can be viewed as categories
- Monotone maps can be viewed as functors
- Monotone maps $P \rightarrow Q$ also form a poset: $F \leq G$ iff $F(x) \leq G(x)$ for all $x \in P$

Example




Watch online video (2 minutes).

 **Braided monoidal categories**

Definition 1.6 (Braided monoidal category). A braided monoidal category is a symmetric monoidal category $(\mathcal{C}, \otimes, \mathbb{I})$, cf. Definition 1.3, equipped with a braiding, which is specified by

Compositions

- A natural isomorphism bc , called the braiding, whose components are of the type

$$bc_{cd} : (c \otimes d) \xrightarrow{\sim} (d \otimes c), \quad c, d \in \mathbf{Ob}_{\mathcal{C}}$$

Explicitly, this means that for any morphisms $f_1 : c_1 \rightarrow d_1$ and $f_2 : c_2 \rightarrow d_2$, the following diagram

$$\begin{array}{ccc} c_1 \otimes c_2 & \xrightarrow{f_1 \otimes f_2} & d_1 \otimes d_2 \\ \downarrow bc_{c_1, c_2} & & \downarrow id_{d_1, d_2} \\ c_2 \otimes c_1 & \xrightarrow{f_2 \otimes f_1} & d_2 \otimes d_1 \end{array}$$

commutes.




Watch online video (10 minutes).

Evaluation and coevaluation

Example: $\mathbf{Vect}_{\mathbb{R}}$

$$V^* := (\text{linear maps } V \rightarrow \mathbb{R}) = \text{Hom}_{\mathbb{R}}(V, \mathbb{R})$$

$V \cong (V^*)^*$ if and only if $\dim V < \infty$

- Another way to characterize this:
 $V \in \mathbf{Ob}_{\mathbf{Vect}_{\mathbb{R}}}$ is dualizable if and only if $\dim V < \infty$.
- We will use "evaluation" and "coevaluation" maps.
- Evaluation:** $e_V : V^* \otimes V \rightarrow \mathbb{R}, (l, v) \mapsto l(v)$
- Coevaluation:** $\eta_V : \mathbb{R} \rightarrow V \otimes V^*, \lambda \mapsto \sum_{i=1}^n e_i \otimes e_i^*$

$V \in \mathbf{Ob}_{\mathbf{Vect}_{\mathbb{R}}}$ is dualizable if

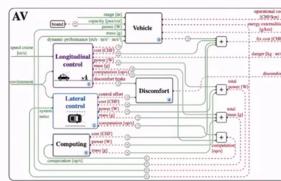
$$(e_V \otimes \text{Id}_{V^*}) \circ (\text{Id}_V \otimes e_V) = \text{Id}_V$$

$$(\text{Id}_{V^*} \otimes \eta_V) \circ (e_V \otimes \text{Id}_{V^*}) = \text{Id}_{V^*}$$


Watch online video (8 minutes).

Design problems

- Example: design problems



Watch online video (19 minutes).

Monoidal posets

Definition 1.1 (Monoidal poset). A monoidal structure on a poset $P = (\mathbb{P}, \leq)$ is specified by:

Conditions

1. A monotone map $\otimes : P \times P \rightarrow P$, called the *monoidal product*.

Note that here we are implicitly assuming $P \times P$ as having the product order. In detail, monotonicity means that, for all $x_1, x_2, y_1, y_2 \in \mathbb{P}$,

$$x_1 \leq y_1 \text{ and } x_2 \leq y_2 \Rightarrow (x_1 \otimes x_2) \leq (y_1 \otimes y_2).$$

2. An element $1 \in \mathbb{P}$, called the *monoidal unit*.

Conditions

1. Associativity: for all $x, y, z \in \mathbb{P}$,

$$(x \otimes y) \otimes z = x \otimes (y \otimes z).$$

2. Left and right unitarity: for all $x \in \mathbb{P}$,

$$1 \otimes x = x \quad \text{and} \quad x \otimes 1 = x.$$

A poset equipped with a monoidal structure is called a *monoidal poset*.



Watch online video (8 minutes).

Monoidal categories

- Example: $\mathbf{Vect}_{\mathbb{R}}$

$$\begin{array}{c} \otimes : \mathbf{Vect}_{\mathbb{R}} \times \mathbf{Vect}_{\mathbb{R}} \rightarrow \mathbf{Vect}_{\mathbb{R}} \\ \text{or } \otimes_{\mathbf{Vect}_{\mathbb{R}}} \end{array}$$

"product"

"unit object"

• **Functorial** $\otimes : \mathbf{Vect}_{\mathbb{R}} \times \mathbf{Vect}_{\mathbb{R}} \rightarrow \mathbf{Vect}_{\mathbb{R}}$ is a functor

• **Associativity** $(U \otimes V) \otimes W \cong U \otimes (V \otimes W)$

• **Unitarity** $I_{\mathbf{Vect}_{\mathbb{R}}} \otimes U \cong U$

$$U \otimes I_{\mathbf{Vect}_{\mathbb{R}}} \cong U$$



Watch online video (5 minutes).

Monoidal posets

Definition 1.1 (Monoidal poset). A monoidal structure on a poset $P = (\mathbb{P}, \leq)$ is specified by:

Conditions

1. A monotone map $\otimes : P \times P \rightarrow P$, called the *monoidal product*.

Note that here we are implicitly assuming $P \times P$ as having the product order. In detail, monotonicity means that, for all $x_1, x_2, y_1, y_2 \in \mathbb{P}$,

$$x_1 \leq y_1 \text{ and } x_2 \leq y_2 \Rightarrow (x_1 \otimes x_2) \leq (y_1 \otimes y_2).$$

2. An element $1 \in \mathbb{P}$, called the *monoidal unit*.

Conditions

1. Associativity: for all $x, y, z \in \mathbb{P}$,

$$(x \otimes y) \otimes z = x \otimes (y \otimes z).$$

2. Left and right unitarity: for all $x \in \mathbb{P}$,

$$1 \otimes x = x \quad \text{and} \quad x \otimes 1 = x.$$

A poset equipped with a monoidal structure is called a *monoidal poset*.



Watch online video (6 minutes).

String diagrams

- The idea: to visualize monoidal categories,

wires for objects, boxes for morphisms

- connect matching wires to compose morphisms

$$\xrightarrow{\square \square} \xrightarrow{\square \square}$$

" $f \circ g$ "

- monoidal product = place morphisms in parallel

$$\xrightarrow{\square \square} \xrightarrow{\square \square}$$

" $f \otimes g$ "

- monoidal unit = empty space

$$\square \square$$

" $1 \otimes 1$ "

- Example: monoidal posets

$$\begin{array}{c} \xrightarrow{\square \square} \xrightarrow{\square \square} \\ \square \square \end{array}$$

" $f \otimes g$ "

$$\xrightarrow{\square \square} \xrightarrow{\square \square}$$

" $f \circ g$ "



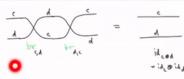
Watch online video (5 minutes).

Symmetric monoidal categories

Definition 1.4 (Symmetric monoidal category). A symmetric monoidal category is a braided monoidal category $(\mathcal{C}, \otimes, 1, \eta, \delta)$ for which the braiding satisfies the symmetry condition

$$\text{br}_{c,d} \circ \text{br}_{d,c} = \text{id}_{c \otimes d} \quad (1.1)$$

for all $c, d \in \text{Ob}_{\mathcal{C}}$



•



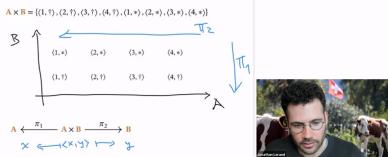
Watch online video (36 minutes).

Cartesian product

Definition (Cartesian product of sets). Given two sets A, B , their cartesian product is denoted $A \times B$ and defined as

$$A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$$

• Example $A = \{1, 2, 3, 4\}$ $B = \{*, †\}$



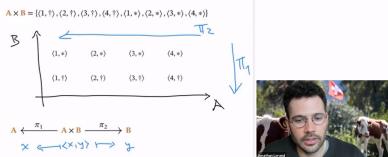
Watch online video (3 minutes).

Cartesian product

Definition (Cartesian product of sets). Given two sets A, B , their cartesian product is denoted $A \times B$ and defined as

$$A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$$

• Example $A = \{1, 2, 3, 4\}$ $B = \{*, †\}$



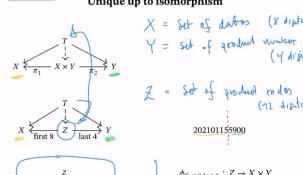
Watch online video (33 minutes).

Choosing



Watch online video (12 minutes).

Unique up to isomorphism



Watch online video (22 minutes).

Cartesian product

Definition (Cartesian product of sets). Given two sets A, B , their cartesian product is denoted $A \times B$ and defined as

$$A \times B = \{(x,y) \mid x \in A \text{ and } y \in B\}.$$

* Example $A = \{1, 2, 3, 4\}$ $B = \{*, †\}$

$$A \times B = \{(1, *), (2, *), (3, *), (4, *), (1, †), (2, †), (3, †), (4, †)\}$$

$\begin{array}{c} B \\ \downarrow \\ \begin{array}{cccc} (1,*) & (2,*) & (3,*) & (4,*) \\ | & | & | & | \\ (*,1) & (2,1) & (3,1) & (4,1) \\ | & | & | & | \\ (†,1) & (2,†) & (3,†) & (4,†) \end{array} \end{array} \quad \begin{array}{c} A \\ \leftarrow \pi_1 \qquad \pi_2 \rightarrow \\ \begin{array}{c} A \times B \\ \xleftarrow{\pi_1 \times \pi_2} \xrightarrow{\pi_2} \end{array} \end{array}$



Watch online video (42 minutes).

Distribution networks



Watch online video (2 minutes).

Relations form a category

Definition (Category Rel). The category of relations \mathbf{Rel} is given by:

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

$$\mathrm{Id}_X := \{(x, x) \mid x \in X\}.$$

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

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


Watch online video (3 minutes).

Composition of relations

$$R \subseteq A \times B \quad S \subseteq B \times C$$

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

* Example

$R = \{(z_1, z_1), (z_2, z_3), (z_3, z_4)\} \subseteq A \times B$
 $S = \{(z_1, z_1), (z_2, z_2)\} \subseteq A \times B$

$R ; S = \{(z_2, z_1), (z_3, z_2)\} \subseteq A \times B$



Watch online video (7 minutes).

Distribution networks

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

Electricity generation, transmission, and distribution

Source: Adapted from National Energy Education Development Project website.



Watch online video (12 minutes).

Endorelations

$A = B = \mathbb{R}^2$ $R = " \leq "$

$\langle x_1, x_2 \rangle \leq \langle y_1, y_2 \rangle \Leftrightarrow x_1 \leq y_1 \text{ and } x_2 \leq y_2$



Watch online video (2 minutes).

Equivalence relations

- Quiz: which of the following are equivalence relations?
 - Having the same blood type
 - For natural numbers: "differing by a multiple of 3"
 $xRy \Leftrightarrow (x - y) \bmod 3 = 0$
 - Equality " $=$ " of real numbers
 - Inequality " \leq " of natural numbers



Watch online video (6 minutes).

Functions as relations

$f : X \rightarrow Y$ $R_f := \{(x, y) \in X \times Y \mid y = f(x)\}$



Watch online video (7 minutes).

Properties of relations

Definition (Properties of all). Let $R \subseteq A \times B$ be a relation. R is:

- Surjective if $\forall y \in B$ there exists an $x \in A$ such that $(x, y) \in R$.
- Injective if for all $(x_1, y_1), (x_2, y_2) \in R$ it holds: $y_1 = y_2 \Rightarrow x_1 = x_2$.
- Defined-everywhere if for all $x \in A$ there exists an $y \in B$: $(x, y) \in R$.
- Single-valued if $\forall x \forall y_1, y_2 \in B$ holds: $x = y_1 \wedge x = y_2 \Rightarrow y_1 = y_2$.



Watch online video (5 minutes).

Relations

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

Example $A = \{x_1, x_2, x_3\}$ $B = \{y_1, y_2, y_3, y_4\}$

$R = \{(x_1, y_1), (x_2, y_3), (x_3, y_4)\} \subseteq A \times B$

x	y	(x, y)
x_1	y_1	(x_1, y_1)
x_1	y_2	(x_1, y_2)
x_1	y_3	(x_1, y_3)
x_1	y_4	(x_1, y_4)
x_2	y_1	(x_2, y_1)
x_2	y_2	(x_2, y_2)
x_2	y_3	(x_2, y_3)
x_2	y_4	(x_2, y_4)
x_3	y_1	(x_3, y_1)
x_3	y_2	(x_3, y_2)
x_3	y_3	(x_3, y_3)
x_3	y_4	(x_3, y_4)



Watch online video (2 minutes).

Transposition

Definition (Transpose of relations). Let $R \subseteq A \times B$ be a relation. The transpose (or opposite, reverse) of R is the relation given by:

$$R^T := \{(y, x) \in B \times A \mid (x, y) \in R\}.$$

note that $R: B \rightarrow A$, while $R^T: A \rightarrow B$.



Watch online video (10 minutes).

Groups

Definition (Group). A group is a monoid together with an “inverse” operation.

In more detail, a group M is

- Constituents
 - 1. a set M ;
 - 2. a binary operation $\circ: M \times M \rightarrow M$, called composition;
 - 3. a specified element $\text{id} \in M$;
 - new** 4. a map $\text{inv}: M \rightarrow M$ called “inverse”.
- Conditions
 - 1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z)$;
 - 2. Neutrality Laws: $\text{id} \circ x = x = x \circ \text{id}$;
 - new** 3. Inverse law: $\text{inv}(x) \circ x = \text{id} = \text{inv}(x) \circ x$.

Quiz: which of these monoids are groups? Give the expression for the inverse operation.

$(\mathbb{N}, +, 0)$	$X \xrightarrow{\quad} \neg X$	$(\mathbb{N}, +, 0)$	\times	$(\mathbb{N}, \max, 0)$
$(\mathbb{N}, \cdot, 1)$	$\times \xrightarrow{\quad} \frac{1}{x}$	$(\mathbb{N}, \cdot, 1)$	\times	\times



Watch online video (7 minutes).

Semigroups

Definition (Semigroup). A semigroup S is a set S , together with a binary operation $\circ: S \times S \rightarrow S$, called composition, which satisfies the associative law:

$$(x \circ y) \circ z = x \circ (y \circ z)$$

for all $x, y, z \in S$.

Example: Consider the discrete time linear time-invariant systems of the form:

$$\begin{aligned} x_{k+1} &= A x_k + B u_k \\ y_k &= C x_k \end{aligned}$$

with the constraint that input and output have the same dimension.

- The composition is the series composition.
- The composition of linear system is linear



Watch online video (13 minutes).

Monoids

Definition (Monoid). A monoid M is:

- Constituents
 - 1. a set M ;
 - 2. a binary operation $\circ: M \times M \rightarrow M$;
 - new** 3. a specified element $\text{id} \in M$, called neutral element.
- Conditions
 - 1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z)$;
 - 2. Neutrality Laws: $\text{id} \circ x = x = x \circ \text{id}$.

Quiz: Which of these semigroups have a neutral element and are therefore monoids?

$(\mathbb{N}, +)$	O	I	(\mathbb{N}, \max)	O	(\mathbb{N}, \min)	X
$(\mathbb{R}, +)$	O	I	(\mathbb{R}, \max)	X	(\mathbb{R}, \min)	O
$(\{x \in \mathbb{N} : x \geq 2021\}, +)$	O	I	$(\{x \in \mathbb{N} : x \geq 2021\}, \max)$	O	$(\{x \in \mathbb{N} : x \geq 2021\}, \min)$	O



Watch online video (3 minutes).

Summary so far

- One way to define algebraic structure is refinement: progressively add more properties that the structure must satisfy.
- Today we defined these structures:
 - A magma is a set with a binary operation.
 - A semigroup is a set with an associative operation.
 - A monoid is a semigroup with a neutral element.
 - A group has an “inverse” operation.
- The other direction, which requires more imagination creatively, is generalization, in the sense of imagining a broader structure that contains the current structure as a particular case.

Spoiler: we are already doing category theory.



Watch online video (24 minutes).

From semigroups to monoids

- Sometime we can "formally add" a neutral element to a semigroup.
- Example:

(\mathbb{R}, \max)



Watch online video (14 minutes).

Semigroups

Definition (Semigroup). A semigroup S is a set S , together with a binary operation $\cdot : S \times S \rightarrow S$, called composition, which satisfies the associativity law:
 $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in S$.

- Examples:

(\mathbb{R}, \cdot)
 $(\mathbb{R}, +)$ (\mathbb{N}, \min) (\mathbb{N}, \max) (\mathbb{R}, \max)



Watch online video (8 minutes).

Semicategories

Definition 11.4 (Semicategory). A semicategory \mathbf{C} is:

Commitments

- Objects: a collection $\mathbf{Ob}_{\mathbf{C}}$, whose elements are called objects.
- Morphisms: for every pair of objects $X, Y \in \mathbf{Ob}_{\mathbf{C}}$, there is a set $\text{Hom}_{\mathbf{C}}(X, Y)$, elements of which are called morphisms from X to Y . The set is called the "hom-set from X to Y ".
- Composition: given any morphism $f \in \text{Hom}_{\mathbf{C}}(X, Y)$ and any morphism $g \in \text{Hom}_{\mathbf{C}}(Y, Z)$, there exists a morphism $f \circ g \in \text{Hom}_{\mathbf{C}}(X, Z)$ which is the composition of f and g .

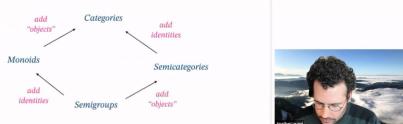
Conditions

- Associativity: for any morphisms $f \in \text{Hom}_{\mathbf{C}}(X, Y)$, $g \in \text{Hom}_{\mathbf{C}}(Y, Z)$, and $h \in \text{Hom}_{\mathbf{C}}(Z, W)$,

$$(f \circ g) \circ h = f \circ (g \circ h). \quad (11.1)$$


Watch online video (1 minutes).

Summary so far



Watch online video (3 minutes).

Logic notation

Logic constants:

true	\top	false	\perp
"top"			"bottom"

Operations:

$a \wedge b$	$a \text{ and } b$
$a \vee b$	$a \text{ or } b$

(Constructive) implication:

$a \Rightarrow b$	If you give me a proof of a , I can give you a proof of b .
" a implies b "	If you give me a refutation of b , I can you give you a refutation of a .

$a \Rightarrow a$



Watch online video (3 minutes).

Equivalence relations

- Equivalence relation: a reflexive, transitive, symmetric relation.
- Recall: equivalence relations define partitions of a set.

reflexive	transitive	symmetric
$\frac{T}{aRa}$	$\frac{aRb \ bRc}{aRc}$	$\frac{aRb}{bRa}$

- Examples:
 - "Having the same blood type" is an equivalence relation.
 - For natural numbers: "differing by a multiple of 3" is an equivalence relation.

$$aRb \Leftrightarrow (a - b) \bmod 3 = 0$$

- \equiv is an equivalence relation.



Watch online video (2 minutes).

Sequent notation

- This is a compact way to describe implications.

$\frac{\frac{a}{b} \ c}{a \ c}$ \Leftrightarrow $\Leftrightarrow b \ c$	$\frac{a}{\frac{b}{c}}$ $a \Leftrightarrow b \Leftrightarrow c$
$\frac{a \ b}{c}$ $(a \wedge b) \Rightarrow c$	$f: \frac{\frac{a}{b}}{c}$ $g: \frac{b}{c}$ $f \dashv g: \frac{a}{c}$
$\frac{T}{a}$ $"a \text{ is true}"$	$\frac{a}{L}$ $"\text{if } a \text{ is true the world explodes}"$



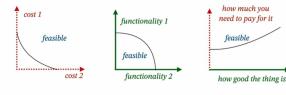
Watch online video (47 minutes).

Trade-offs

- We distinguish (semantically) between functionality and requirements/costs.



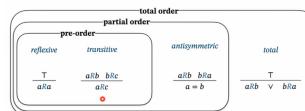
- Exercise: Open an engineering book. Find the graphs talking about "achievable performance". What colors should the axes be? Classify into one of these:



Watch online video (31 minutes).

Definition of pre-order, partial order, total order

- Pre-order: a relation that is transitive and reflexive.
- Partial order: a relation that is transitive, reflexive, and antisymmetric.
- Total order: a relation that is transitive, reflexive, antisymmetric and total.



- Customary symbols for a pre-order relation:

$$\leq \quad \preceq$$

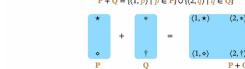


Watch online video (3 minutes).

Disjoint union of posets

- Given two sets P and Q , we have defined the disjoint union $P + Q$.

$$P + Q = \{(1, \circ), (2, \circ)\} \cup \{(1, \star), (2, \star)\}$$



- If P and Q are posets, we can give $P + Q$ the structure of a poset.

$$\preceq_{P+Q}: (P + Q) \times (P + Q) \rightarrow \text{Bool}$$

$$(1, \circ), (1, \circ) \rightarrow (1, \circ) \preceq_{P+Q} (2, \circ)$$

$$(2, \circ), (1, \circ) \rightarrow \perp$$

$$(1, \circ), (2, \circ) \rightarrow T$$

$$(2, \circ_1), (2, \circ_2) \rightarrow (2, \circ_1) \preceq_{P+Q} (2, \circ_2)$$

$$\bullet$$



Watch online video (2 minutes).

Disjoint union of posets

- Given two sets P and Q , we have defined the disjoint union $P + Q$.

$$\begin{array}{c} * \\ P \end{array} + \begin{array}{c} * \\ Q \end{array} = \begin{array}{c} (1,*) \\ (1,*), (2,*) \\ (1,*), (2,*) \\ P+Q \end{array}$$

- If P and Q are posets, we can give $P + Q$ the structure of a poset.

$$\leq_{P+Q} : (P + Q) \times (P + Q) \rightarrow \text{Bool}$$

$$(1, z_1), (1, z_2) \mapsto (z_1 \leq_P z_2)$$

$$(2, z_1), (1, z_2) \mapsto \perp$$

$$(z_1, z_2), (2, z_3) \mapsto T$$

$$(2, z_1), (2, z_2) \mapsto (z_1 \leq_Q z_2).$$

•

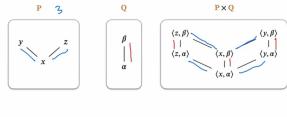


Watch online video (2 minutes).

Product of posets

- Given two sets A and B , we have defined the product $A \times B$.
- If A and B are posets, we can give $A \times B$ the structure of a poset.

$$\begin{array}{c} (y_1, z_1) \leq_{P \times Q} (y_2, z_2) \\ (y_1 \leq_P y_2) \wedge (z_1 \leq_Q z_2) \end{array}$$



Watch online video (7 minutes).

Counting orders

Quiz:

- How many pre-orders exist for a set of $n = 2$ elements?
- How many partial orders exist for a set of $n = 2$ elements?
- How many total orders exist for a set of $n = 5$ elements?

<http://bit.ly/3yPdRzg>



Watch online video (1 minutes).

There are more general ways to describe "preferences"

- Hierarchy of order types:
 - Quasittransitive relation
 - Interval order
 - Semiorder
 - Preorder
 - Partial order
 - Total order

more specific

Example of semi-order preferences:
I am indifferent between 10 and 11,
and indifferent between 11 and 12.
But I prefer 10 to 12.



Watch online video (2 minutes).

Hasse Diagrams

- Hasse Diagrams are an economical way to draw partial orders.



Watch online video (3 minutes).

Example: set-based filtering

- Here's an example of **set-based filtering** ("filtering"; online inference).

Scenario:

- Suppose we want to track the value of a quantity x in the interval $[0, 100]$.
 - We don't know anything about x a priori.
 - We have sensors that periodically measure the quantity with some variable precision.
 - The quantity fluctuates randomly:
- $x_t \in [l_t, u_t]$
- $x_t \in [-1, +1]$ (except at boundaries)
- $x_t \in [l_t, u_t]$

- When dealing with inference, these are the basic questions:
 - What are the possible information states?
 - Is it possible to represent the information states exactly?
 - Do I need an approximation?
 - What is the prior information?
 - What are the update rules?

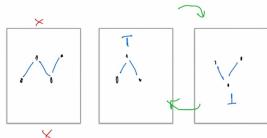


Watch online video (1 minutes).

Top and bottom

- The **Top** of a partial order is the element that dominates all others.

- The **Bottom** of a partial order is an element that is dominated by all others.

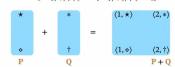


Watch online video (5 minutes).

Disjoint union of posets

- Given two sets P and Q , we have defined the disjoint union $P + Q$.

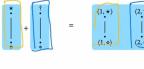
$$P + Q = \{(1, \cdot) \mid 1 \in P\} \cup \{(2, \cdot) \mid 2 \in Q\}$$



- If P and Q are posets, we can give $P + Q$ the structure of a poset.

$$\leq_{P+Q} : (P + Q) \times (P + Q) \rightarrow \text{Bool}$$

$$\begin{aligned} (1, \cdot_1), (1, \cdot_2) &\mapsto (1_1 \leq_P 1_2) \\ (1, \cdot_1), (1, \cdot_3) &\mapsto \perp \\ (1, \cdot_1), (2, \cdot_2) &\mapsto \perp \\ (2, \cdot_1), (2, \cdot_2) &\mapsto (2_1 \leq_Q 2_2) \end{aligned}$$



3. Solutions to selected exercises

Solution of Exercise 1. *Teeth*, Gollum answered correctly.

PART A.E PLURIBUS



4. Putting things together	39
5. Sets and functions	53
6. Unus Pro Omnibus, Omnes Pro Uno	61
7. Morphisms	77
8. Actions	85
9. Solutions to selected exercises	91



4. Putting things together

In this chapter we discuss the various types of “compositions” we find in applications.

4.1. Stacking blocks

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

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

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

$$\begin{array}{ccc} \text{ingredient} & \text{ingredient} & \text{ingredient} \\ \hline & \text{result} & \text{result} \end{array} \quad (4.1)$$

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

4.1 Stacking blocks	39
4.2 Mixing colors	43
4.3 Commutativity and associativity	46
4.4 Composing recipes	48
4.5 Isomorphisms	49

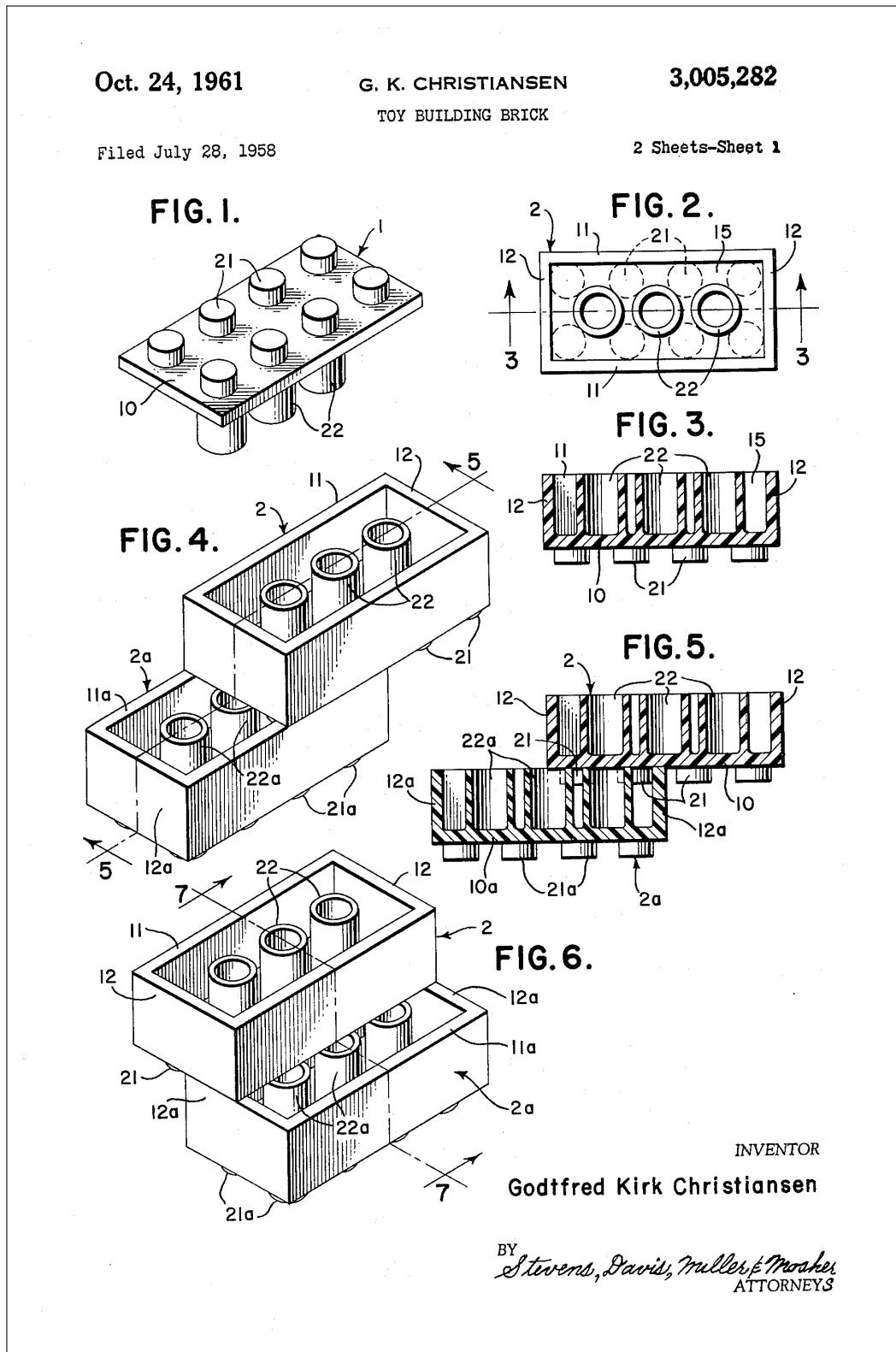
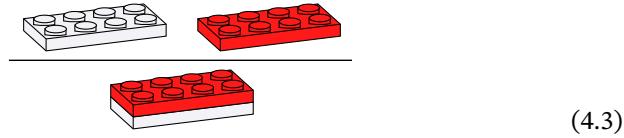
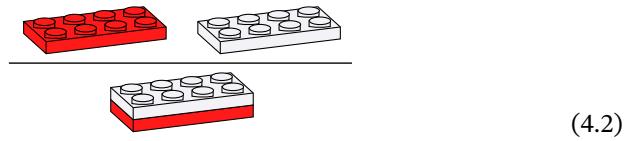
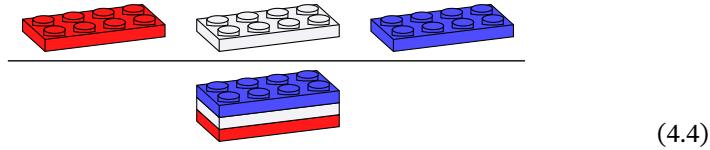


Figure 4.1.: The 1961 Lego patent.

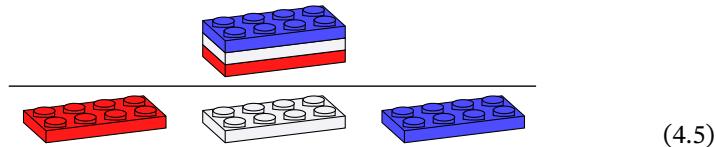
on the table. (4.2) shows that composing red and white produces a red-white brick; (4.3) shows that composing white and red produces a white-red brick.



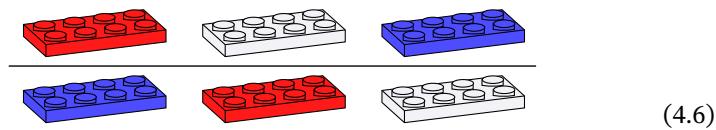
We can compose more than one brick. For example, red, white, blue, make a red-white-blue brick.



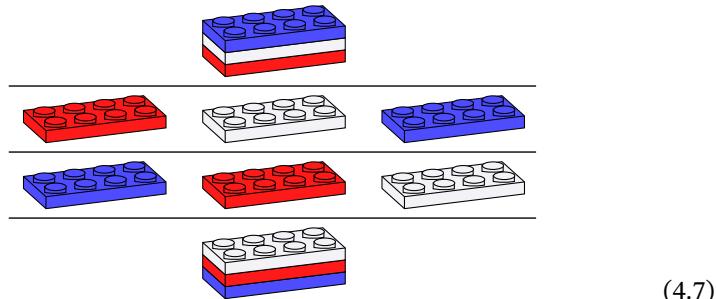
In Lego we can also de-compose. If we have a red-white-blue brick, we can also recover the single bricks.



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



Consequently, if you have a red-white-blue brick, you can disassemble, permute, reassemble to obtain a blue-white-red.



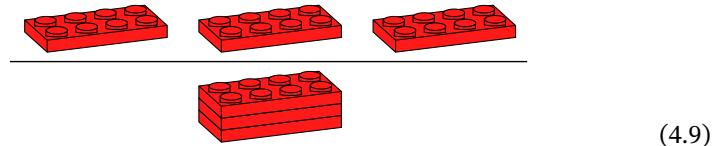
The aforementioned recipe contains several concrete steps to go from the initial ingredient to the final result. If we do not care about the detailed steps, we can

summarize the recipe as follows, by eliding the intermediate steps and only remember the ingredient and the results.

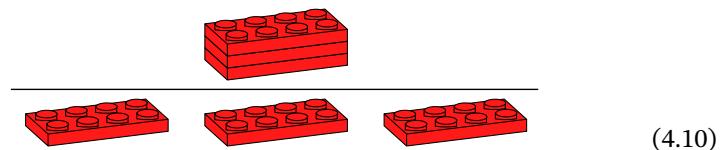


Alternatively, you can think of (4.8) as the statement of a theorem, and of (4.7) as the proof of the theorem.

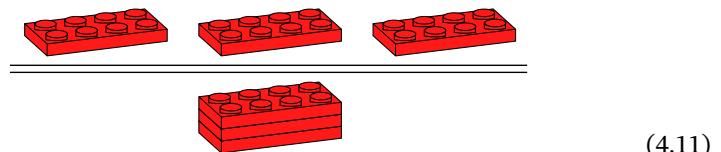
Sometimes we want to think about the transformations that are reversible. For example, we can assemble 3 red bricks into a red-red-red brick:



We can also do the opposite:



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



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



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

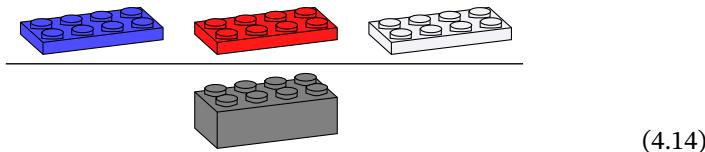
as a full block, but not vice versa.



(4.13)

4.2. Mixing colors

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



(4.14)

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



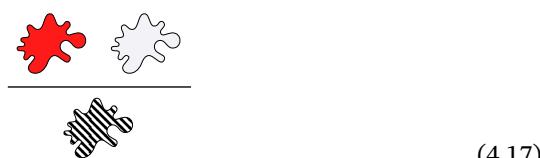
(4.15)

Furthermore, white and white make white:



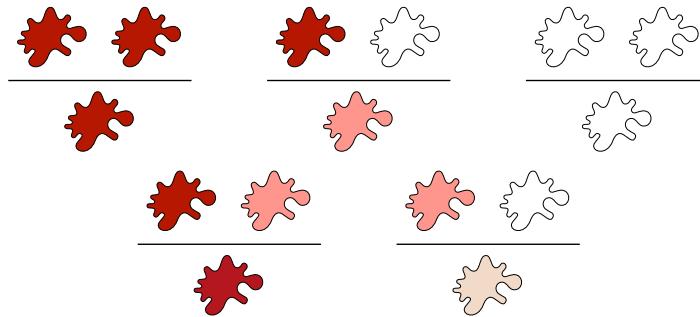
(4.16)

However, white and red make *horrible*.



(4.17)

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



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

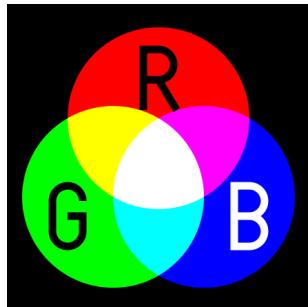
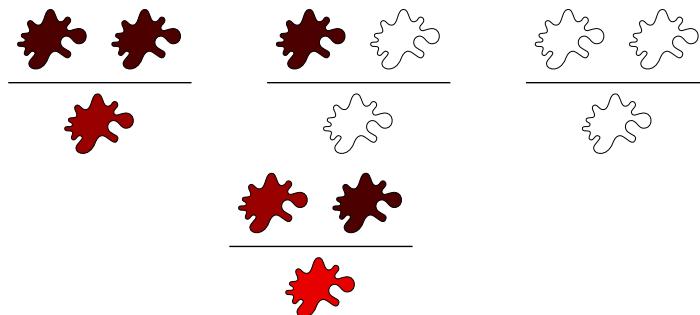
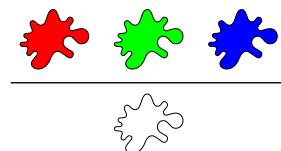


Figure 4.2.: Additive composition



Green, red, blue additively make white:



(4.18)

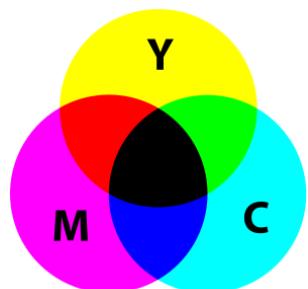
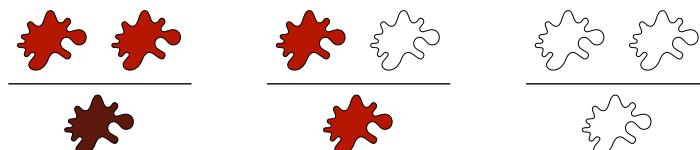
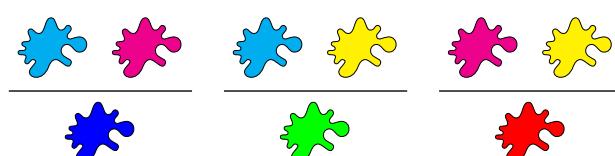


Figure 4.3.: Subtractive composition

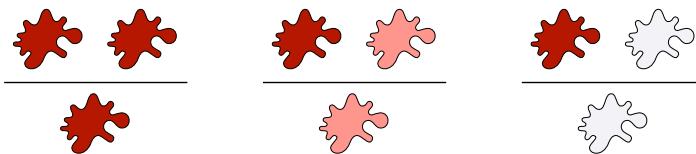
A different way to compose colors is by using the **subtractive** rules in the CMY (cyan, magenta, yellow) color space. These rules formalize the physical process of offset printing: we produce colors by putting pigments that block the other colors.



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

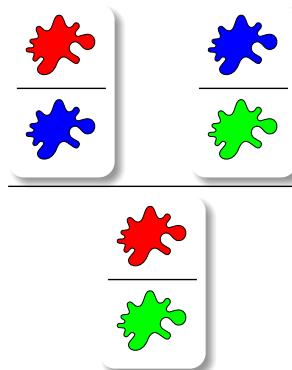


Finally, we can think of a **paint-over-it** composition rule: the first color is replaced by the second.



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

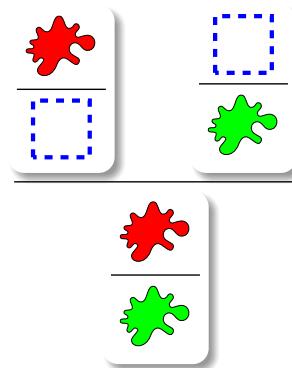
For example, the following shows that if a red stain gives you a blue stain, and a blue stain gives you a green stain, you can produce a green stain from a red stain.



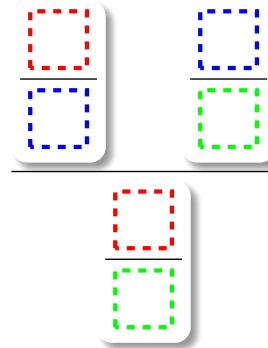
Note that to activate the meta-recipe above, no red stain was needed. In fact, for the above to be valid, it is not even necessary to postulate that red stains exist.

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

For example, if a red stain gives me an B, and B gives me a green stain, then a red stain gives me a green stain, not matter what B is.



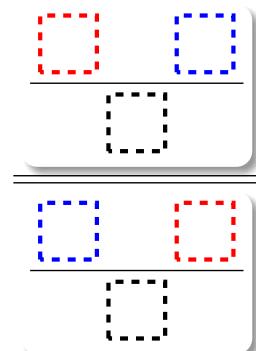
We can abstract further by saying that: if A gives me B, and B gives me C, then A gives me C.



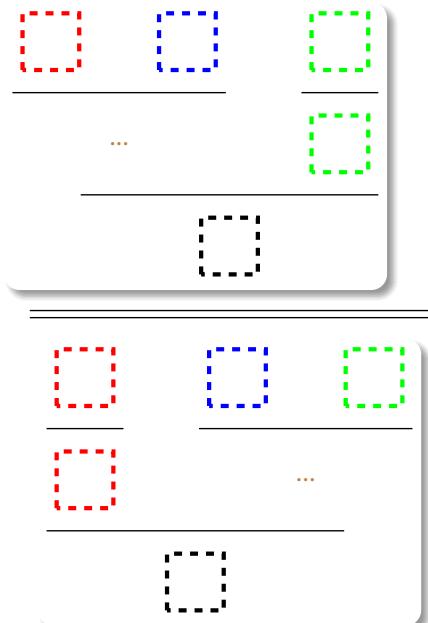
4.3. Commutativity and associativity

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

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



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



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

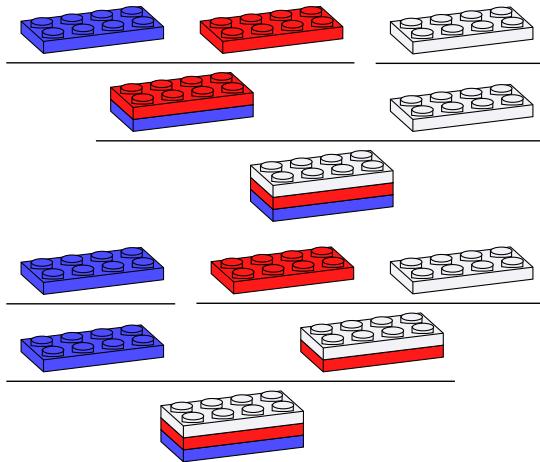
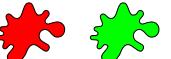


Table 4.1 shows the properties of the 4 composition rules for composing colors that we described earlier.

The table also notes the presence of a *neutral element* and an *annihilating element*.

Table 4.1.: Properties of color composition rules

	Lego purist	Mixing paint	Apply paint	Additive	Subtractive
	 	 	 	 	 
Commutative?	yes	yes	no	yes	yes
Associative?	yes	no	yes	yes	yes
annihilative element?		no			
neutral element?	no	no	no		

4.4. Composing recipes

We can also compose recipes themselves.

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

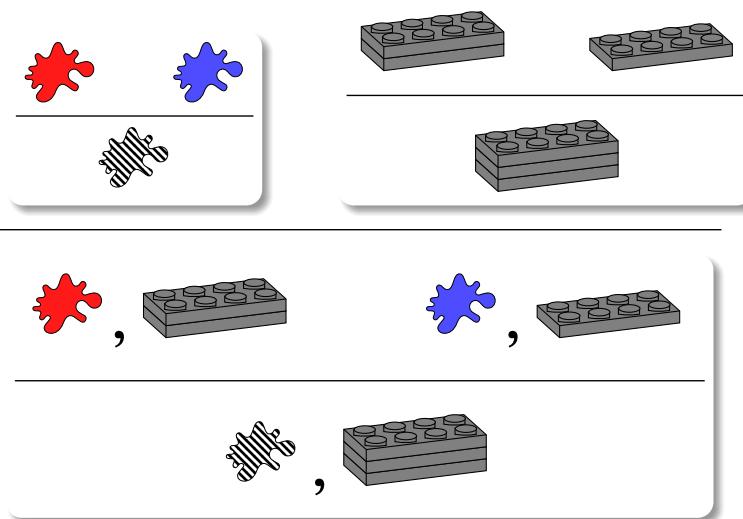
$$\begin{array}{c}
 \text{red} \\
 \hline\hline
 \text{red}, \text{grey}
 \end{array} \tag{4.19}$$

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

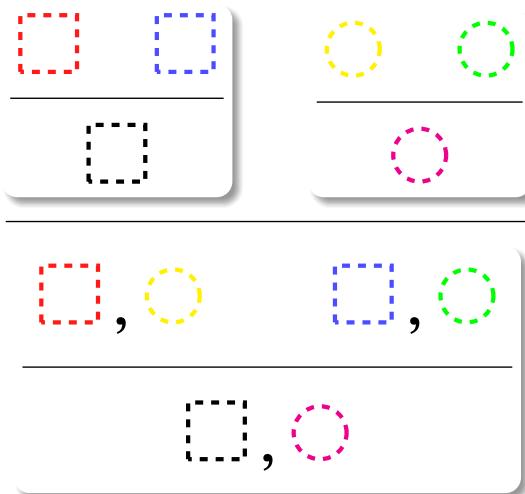
For example, given two pairs

$$\text{red}, \text{grey} \quad \text{and} \quad \text{blue}, \text{grey}$$

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



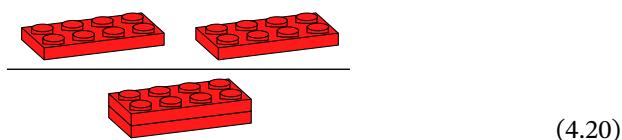
We can generalize this as follows



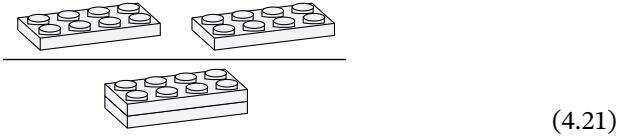
4.5. Isomorphisms

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

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



and

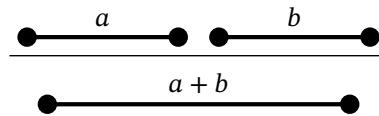


We could create Lego theories for each of the worlds. Although they would describe different worlds, the theories would be *isomorphic*.

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

$$\frac{1 \quad 1}{2} \quad (4.22)$$

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



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

We want to show you a rope trick. Suppose that we want to be more precise than Section 4.5 to describe the process of composing ropes, by keeping track of the extra rope that is needed to make a knot (Fig. 4.4).

One first attempt would be to call k the extra rope for the knot, and have rules like (4.23): from $a+k$ and $k+b$ we obtain a piece of rope of $a+b$.

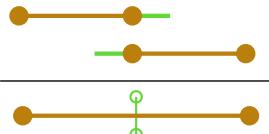
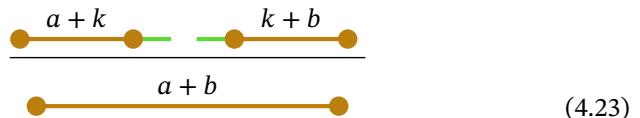
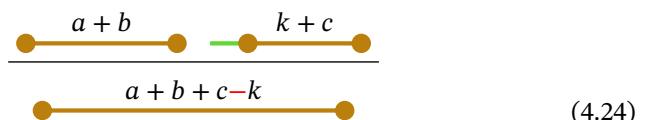


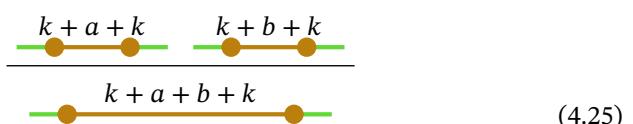
Figure 4.4.: Keeping track of knot material



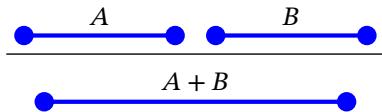
This is fine but not elegant. If you want to compose further, you need to introduce a notion of subtraction.



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



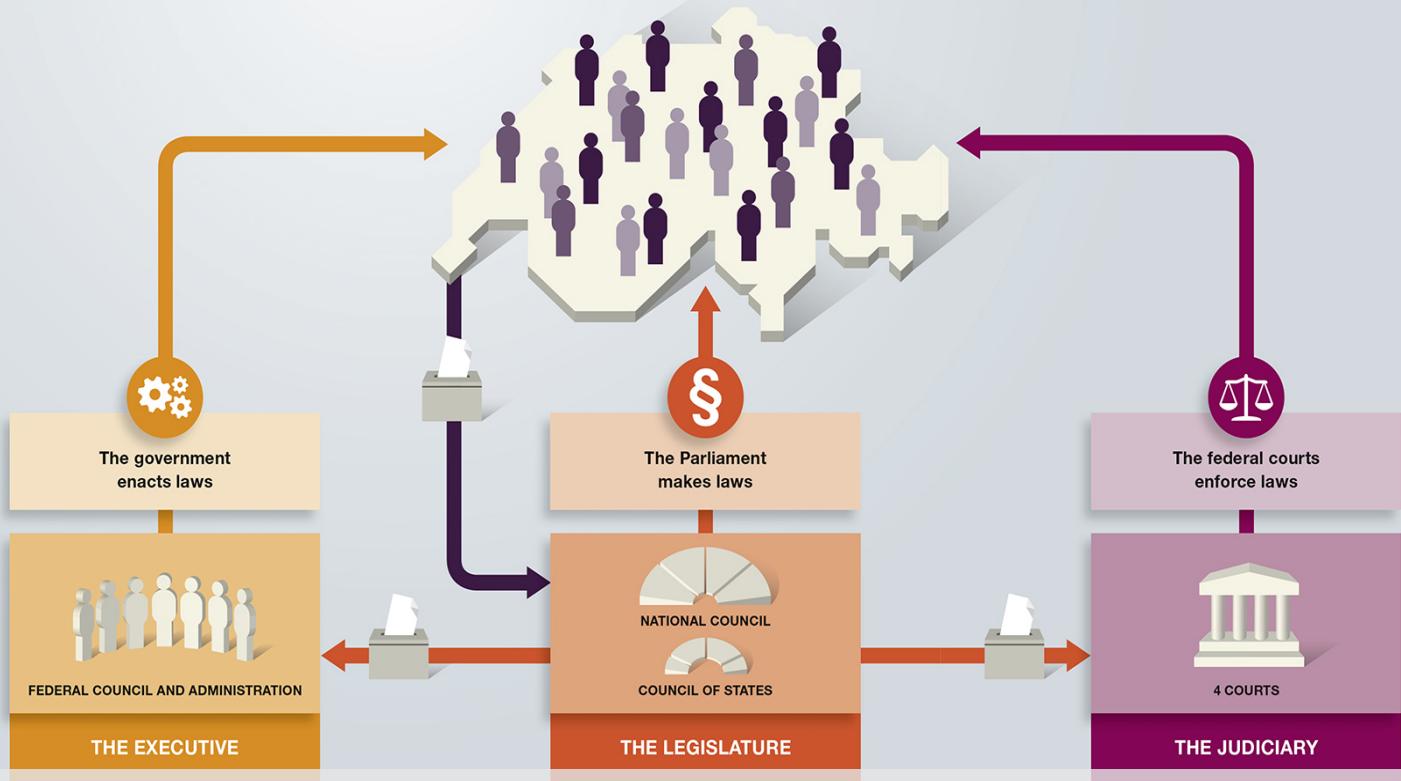
Now when we compose, the 2 ks on the inside they elide, and we are left with 2 ks at either end, ready to be knotted with other pieces of rope. Notice that all ropes so created have the 2 extra ks . We can just remove them from the notation. We obtain new rules for ropes that take into account the knots:



And here's the magic trick: if we don't take into account the knot materials we have the simple rule Section 4.5; if we do take into account the knot materials, *for any arbitrary length k* , we obtain Section 4.5 which is exactly the same as Section 4.5.

Exercise 2. Explain the trick: Where did the extra material go?

See solution on page 91.



5. Sets and functions

5.1. Sets

5.1 Sets	53
5.2 Functions	54
5.3 Code exercises	55
Properties	55
Mappings	55
Set products	56
Set union *	58

5.2. Functions

5.3. Code exercises

We go on assuming that you have already completed the tutorial in Section 1.3.

Properties

Code exercise 1 (`TestFiniteSetProperties`). Given two finite sets, check if they are a subset of the other. Implement the interface in Listing 8.

Listing 8: The `FiniteSetProperties` interface.

```
class FiniteSetProperties(ABC):
    @abstractmethod
    def is_subset(self, a: FiniteSet, b: FiniteSet) -> bool:
        """ True if `a` is a subset of `b`. """
    def equal(self, a: FiniteSet, b: FiniteSet) -> bool:
        return self.is_subset(a, b) and self.is_subset(b, a)
    def is_strict_subset(self, a: FiniteSet, b: FiniteSet) -> bool:
        return self.is_subset(a, b) and not self.is_subset(b, a)
```

As you can see, once you define `is_subset()`, you get for free the implementation of `equal()` and `is_strict_subset()`.

Check your results with

```
make check-TestFiniteSetProperties
```

Mappings

Listing 9 shows our interface for `Mapping`.

Listing 9: The `Mapping` interface.

```
class Mapping(ABC):
    @abstractmethod
    def source(self) -> Setoid:
        ...
    @abstractmethod
    def target(self) -> Setoid:
        ...
    @abstractmethod
    def __call__(self, a: Element) -> Element:
        ...
```

A `Mapping` has a source and a target, which, in general, are setoids.

A `Mapping` is able to take an element from the source setoid and return an element in the second setoid.

Here we use Python's syntax sugar. If an object defines an `__call__()` method, then these are equivalent:

```
assert myobject(x) == myobject.__call__(x)
```

Therefore, once you implement a `Mapping` you can pass it around like if it was a Python function.

We also define a subclass `FiniteMap` which is constrained to have both source and target be a `FiniteSet` rather than a `Setoid` (Listing 10).

Listing 10: The `FiniteMap` interface.

```
class FiniteMap(Mapping, ABC):
    @abstractmethod
    def source(self) -> FiniteSet:
        ...

    @abstractmethod
    def target(self) -> FiniteSet:
        ...
```

```
source:
  elements: [a, b, c]
target:
  elements: [1, 2]
values:
  - [a, 1]
  - [b, 2]
  - [c, 1]
```

Figure 5.1.: Format for representing maps.

We define a YAML representation for finite maps as in Section 5.3.

The fields `source` and `target` are two sets. These are expressions that you can pass to the set constructors you developed previously.

The field `values` is an array of pairs: a pair $\langle x, y \rangle$ means that x maps to y .

For the data to be well-formed it is necessary that for each element in the domain, there is exactly one row for that element. If that is not the case, throw an exception `InvalidFormat`.

Code exercise 2 (`TestFiniteMapRepresentation`). Create a function to load the data by implementing the interface in Listing 11.

Listing 11: The `FiniteMapRepresentation` interface.

```
class FiniteMapRepresentation(ABC):
    @abstractmethod
    def load(self, h: IOHelper, s: FiniteMap_desc) -> FiniteMap:
        ...

    @abstractmethod
    def save(self, h: IOHelper, m: FiniteMap) -> FiniteMap_desc:
        ...
```

Set products

When we compute the union of two sets, we want to forget the structure. With products we want to remember the structure. So we define a `SetProduct` to be a special `Setoid` that remembers its factors.

Listing 12: The `SetProduct` interface.

```

class SetProduct(Setoid, ABC):
    """ A set product is a setoid that can be factorized. """

    @abstractmethod
    def components(self) -> List[Setoid]:
        """ Returns the components of the product"""

    @abstractmethod
    def pack(self, *args: Element) -> Element:
        """ Packs an element of each setoid into an element of the mapping"""

    @abstractmethod
    def projections(self) -> List[Mapping]:
        """ Returns the projection mappings. """

```

The semantics is the following:

- ▷ The method `components()` returns the ordered list of components.
- ▷ The method `pack()` takes a number of arguments and creates an element of the product.
- ▷ The method `projections()` returns the list of projection mappings. The i -th projection mapping takes an element of the product and returns the i -th component.

Code exercise 3 (`TestMakeSetProduct`). Given two sets, compute their product. Implement the interface in Listing 13.

Test your results using

```
make check-TestMakeSetProduct
```

```

product:
- elements:
  - a
  - b
- elements:
  - 1
  - 2

```

Listing 13: The `MakeSetProduct` interface.

```

class MakeSetProduct(ABC):
    @overload
    def product(self, components: List[Setoid]) -> SetProduct:
        ...

    @abstractmethod
    def product(self, components: List[FiniteSet]) -> FiniteSetProduct:
        ...

```

Figure 5.2.: $\{a, b\} \times \{1, 2\}$

```

product:
- elements:
  - 1
- elements:
  - 1
- elements:
  - 1

```

The interface above means that if you are passed 2 `FiniteSets`, you should return a `FiniteSet`. Otherwise, you should return a `Setoid`.

Figure 5.3.: $\{1\} \times \{1\} \times \{1\}$

Code exercise 4 (`TestFiniteSetRepresentationProduct`). Extend now the code you wrote for loading sets to allow the format in Figs. 5.2 to 5.4. We add another clause to the parsing algorithm:

1. If it has a field `elements`, it is a finite set described with elements directly.
2. If there is a field `product`, it must be a list of sets, and the semantics is the product of those sets.

```

product:
- elements:
  - 1
- elements: []

```

Figure 5.4.: $\{1\} \times \emptyset$

3. Otherwise, it is an error — for now; we will introduce many more ways to describe sets.

Test your results using

```
make check-TestFiniteSetRepresentationProduct
```

Set union ★

Remark 5.1. This is a more advanced exercise; feel free to skip the first time around.

When in mathematics we compute the union of two sets, we obtain a set but we forget the structure. We cannot do this when we take a constructive approach.

For example, suppose that we have two **Setoids** **A** = {*a*, *b*, *c*} and **B** = {*a*, *b*, 3}. By all accounts the elements *a*, *b* are in both. However, how can we judge that *a* = *b*? Should we use the first setoid or the second? Both? What if they disagree?

We choose this semantics: when comparing two elements, we see the set of components to which they both belong. Then we compare them using all the equalities. The elements are equal if they are equal according to all of them.

From the point of view of the implementation, we create an interface like in Listing 14, together with **EnumerableSetUnion** and **FiniteSetUnion**, not shown.

Listing 14: The **SetUnion** interface.

```
class SetUnion(Setoid, ABC):
    """A set product is a setoid that can be factorized. """

    @abstractmethod
    def components(self) -> List[Setoid]:
        """Returns the components of the union"""
```

The method **components()** allows us to recover the components for the union.

Code exercise 5 (`TestMakeSetUnion` (for advanced students)). Implement the interface in Listing 15.

The **@overload** annotations means that:

- ▷ if the method is given all **FiniteSet**, it should return a **FiniteSet**;
- ▷ if the method is give all **EnumerableSet**, it should return an **EnumerableSet**;
- ▷ otherwise, in the general case, it should return a **Setoid**.

Listing 15: The **MakeSetUnion** interface.

```
class MakeSetUnion(ABC):
    @overload
    def union(self, components: List[FiniteSet]) -> FiniteSetUnion:
        ...

    @overload
```

```
def union(self, components: List[EnumerableSet]) -> EnumerableSetUnion:
    ...
    @abstractmethod
def union(self, components: List[Setoid]) -> SetUnion:
    ...
```

Check your results with

```
make check-TestMakeSetUnion
```

Extend now the code you wrote for loading sets to allow the format in Fig. 5.2.

This is the algorithm to implement.

Given a dictionary:

1. If it has a field `elements`, it is a finite set described with elements directly.
2. If it has a field `union`, then the value must be a list of set representation dictionaries.
3. Otherwise, it is an error — for now; we will introduce many more ways to describe sets.

In particular, note that these are valid representations.

```
union:
- elements:
  - 1
  - 2
- elements:
  - a
  - b
  - 1
```

Figure 5.5.: Union

An empty set

The union of 1 empty set

The union of 2 empty sets

The union of zero sets

```
elements: []
```

```
union:
- elements: []
```

```
union:
- elements: []
- elements: []
```

```
union: []
```




6. Unus Pro Omnibus, Omnes Pro Uno

This chapter introduces three of the most basic algebra structures: semigroups, monoids, and groups.

Oftentimes we are going to studies certain *structures* and then their *refinements*. By *refinement* we mean another type of structure that has additional properties/-constraints.

The simplest structure that has to do with composition is that of a *magma*: it just assumes that there is a set with a binary operation defined on it.

Definition 6.1 (Magma). A *magma* \mathbf{S} is a set \mathbf{S} , together with a binary operation

$$\circ : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}. \quad (6.1)$$

Given a finite set \mathbf{A} , one way to specify a composition operation \circ on \mathbf{A} is simply by writing out what it does with each pair of elements of \mathbf{A} . Since \circ is a function of two variables, this can be conveniently displayed as a table, sometimes called a *multiplication table* or a *Cayley table*. We will use the name *composition table*. Consider for example the set $\mathbf{A} = \{\otimes, \approx\}$. A composition operation \circ is specified in Table 6.1. We read it as saying

$$\otimes \circ \otimes = \otimes, \quad \otimes \circ \approx = \otimes, \quad \approx \circ \otimes = \approx, \quad \approx \circ \approx = \otimes.$$

Magmas are quite general and simplistic. We can build more interesting structures by considering, for example, properties that the composition operation might

6.1 Semigroups	62
Induced n -ary multiplication	66
Code exercises	66
6.2 Monoids	69
6.3 Groups	73

Table 6.1.: Composition table.

\circ	\otimes	\approx
\otimes	\otimes	\otimes
\approx	\approx	\otimes

have.

6.1. Semigroups

Watch online video (7 minutes).

Semigroups

Definition (Semigroup). A semigroup \mathbf{S} is a set \mathbf{S} , together with a binary operation $\circ : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}$, called composition, which satisfies the associative law:

$$(x \circ y) \circ z = x \circ (y \circ z)$$

for all $x, y, z \in \mathbf{S}$.

Example: Consider the discrete time linear time-invariant systems of the form

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned}$$

with the constraint that input and output have the same dimension.

- The composition is the series composition.
- The composition of linear system is linear
- we have a semigroup.





Definition 6.2 (Semigroup). A *semigroup* \mathbf{S} is a set \mathbf{S} , together with a binary operation:

$$;\mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}, \quad (6.2)$$

called *composition*, which satisfies the *associative law*:

$$(x ; y) ; z = x ; (y ; z) \quad (6.3)$$

for all $x, y, z \in \mathbf{S}$.

Remark 6.3. Given a fixed set \mathbf{S} , there will in general be many different choices of composition operation which make \mathbf{S} into a semigroup. So, technically, a semigroup is a pair $(\mathbf{S}, ;)$ consisting of a set \mathbf{S} and a choice of composition $;$. The set \mathbf{S} is the *underlying set* of the semigroup. Often we will be slightly imprecise and refer to a semigroup simply by the name of its underlying set; this is practical when it is clear from context which multiplication operation we are considering, or when it is not necessary to refer to the multiplication explicitly.

Example 6.4. Pair-wise average on \mathbb{R} ,

$$x ; y := \frac{x + y}{2}, \quad (6.4)$$

does not define semigroup composition, because it is not associative.

A counterexample could be:

$$(4 ; 8) ; 16 = 11 \neq 8 = 4 ; (8 ; 16). \quad (6.5)$$

Example 6.5. Consider $(\mathbb{N}, +)$, defined as:

$$x ; y := x + y.$$

This is a semigroup, since, for all $l, m, n \in \mathbb{N}$, we have

$$(l + m) + n = l + (m + n). \quad (6.6)$$

Example 6.6 (Booleans). Consider the set $\mathbf{B} = \{\perp, \top\}$, and $\langle \mathbf{B}, \wedge \rangle$, where the operation \wedge is defined via Table 6.2.

This forms a semigroup, given the associativity of \wedge .

Graded exercise 1 ([CompositionTable](#)). Consider composition presented in Table 6.1. Does this composition operation define a semigroup?

Table 6.2.: Composition table for booleans.

\wedge	\perp	\top
\perp	\perp	\perp
\top	\perp	\top

Exercise 3. [[CrossProduct](#)] Consider $\mathbf{S} = \mathbb{R}^3$ and the operation usually referred to as the “cross-product”:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \circ \begin{pmatrix} x \\ y \\ z \end{pmatrix} := \begin{pmatrix} bz - cy \\ cx - az \\ ay - bx \end{pmatrix}$$

This is a binary operation and therefore $\langle \mathbb{R}^3, \circ \rangle$ forms a magma. Show that this does not form a semigroup.

See solution on page 91.

Example 6.7. Consider a finite set \mathbf{A} , which we think of as an alphabet. For instance, consider

$$\mathbf{A} = \{\bullet, \circ\}. \quad (6.7)$$

Let \mathbf{S} be the set of non-empty strings of elements of A . For example,

$$\bullet \bullet \bullet \bullet \bullet \quad (6.8)$$

is a non-empty string of elements of \mathbf{A} .

We may define a multiplication operation on \mathbf{S} simply by concatenating strings. Given the strings

$$\bullet \bullet \bullet \bullet \bullet \text{ and } \bullet \bullet \bullet$$

their concatenation

$$\bullet \bullet \bullet \bullet \bullet \circ \bullet \bullet$$

is the string

$$\bullet \bullet \bullet \bullet \bullet \bullet \bullet \dots$$

It is readily seen that concatenation satisfies the associative law, so \mathbf{S} , together with this multiplication, forms a semigroup, often called *free semigroup*.

Graded exercise 2 ([VariationsOnConcatenation](#)). Consider the set \mathbf{S} of finite non-empty strings of symbols from the alphabet \mathbf{A} , as in Example 6.7.

Can you think of other candidates for multiplication operations on \mathbf{S} , besides the straightforward concatenation of strings considered above? Do your candidates define semigroup multiplications – that is, do they obey the associative law?

For example, one might consider the operation where, given an ordered

pair of strings, one first doubles the last symbol of the first string, and then concatenates. Is this operation associative? Justify your answers.

Example 6.8. The function $\max : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ defines a multiplication operation which equips \mathbb{N} with the structure of a semigroup. It is easy to show that it satisfies associativity. Given $x, y, z \in \mathbb{N}$, we have:

$$\max(\max(x, y), z) = \max(x, \max(y, z)).$$

Exercise 4. Verify the statement made in Example 6.8; that is, check that the associative law holds.

Does $\min : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ also define a semigroup structure on \mathbb{N} ?

See solution on page 91.

Example 6.9. Consider the set $\mathbf{A} = \{\text{sprout}, \text{young}, \text{mature}, \text{old}, \text{dead}\}$ which describes five possible states of a plant. Let $f : \mathbf{A} \rightarrow \mathbf{A}$ be the function that describes “development” (Fig. 6.1):

$$\begin{aligned} f(\text{sprout}) &= \text{young} \\ f(\text{young}) &= \text{mature} \\ f(\text{mature}) &= \text{old} \\ f(\text{old}) &= \text{dead} \\ f(\text{dead}) &= \text{dead} \end{aligned}$$

In other words, we think of f as the change of state of the plant during a given time



Figure 6.1.: Graphical representation of plant transitions.

interval (say, three months). Composing the function f with itself corresponds to considering multiples of the given time interval. For example, the function

$$f \circ f : \mathbf{A} \rightarrow \mathbf{A}$$

models the change over the course of nine months. In general, for the n -fold composition of f with itself we write f^n . The set $\mathbf{S} = \{f^n \mid n \in \mathbb{N}\}$ is a semigroup, with the multiplication given by the composition operation.

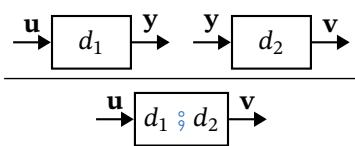


Figure 6.2.: Composition of discrete-time linear systems.

Definition 6.10 (Discrete-time linear systems). A discrete-time linear time-invariant proper open system is defined by three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$. Together they give a recurrence of the type

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{Ax}_k + \mathbf{Bu}_k \\ \mathbf{y}_k &= \mathbf{Cx}_k \end{aligned} \tag{6.9}$$

If \mathbf{x} has dimension $n \geq 1$, u dimension $m \geq 1$ and \mathbf{y} dimension $p \geq 1$, then \mathbf{A} has dimension $n \times n$, \mathbf{B} has dimension $n \times m$, and \mathbf{C} has dimension $p \times n$.

Consider now the systems where $m = p$ (but possibly different n): these are systems with input and output of the same size. Hence, we can compose them in series. Series composition is the composition in which the output of a system is the input of another system (Fig. 6.2). The composition forms a new discrete-time linear system, which has the input of the first system and the output of the second system. Let's consider the first system d_1 :

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{Ax}_k + \mathbf{Bu}_k \\ \mathbf{y}_k &= \mathbf{Cx}_k\end{aligned}$$

and the second system (which has \mathbf{y} as input) d_2 :

$$\begin{aligned}\mathbf{z}_{k+1} &= \mathbf{Ez}_k + \mathbf{Fy}_k \\ \mathbf{v}_k &= \mathbf{Gz}_k\end{aligned}$$

We can write their composition $d_1 \circ d_2$ compactly as a discrete linear system with input \mathbf{u} and output \mathbf{v} :

$$\begin{aligned}\begin{pmatrix} \mathbf{x}_{k+1} \\ \mathbf{z}_{k+1} \end{pmatrix} &= \begin{pmatrix} \mathbf{A} & 0 \\ \mathbf{FC} & \mathbf{E} \end{pmatrix} \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \end{pmatrix} + \begin{pmatrix} \mathbf{B} \\ 0 \end{pmatrix} \mathbf{u}_k \\ \mathbf{v}_k &= (0 \quad \mathbf{G}) \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \end{pmatrix}\end{aligned}$$

We now want to show that the composition of discrete linear systems is associative (graphically reported in Fig. 6.3). To do so, we need a third system, d_3 , with input \mathbf{v} and output \mathbf{w} :

$$\begin{aligned}\mathbf{s}_{k+1} &= \mathbf{Hs}_k + \mathbf{Iv}_k \\ \mathbf{w}_k &= \mathbf{Js}_k\end{aligned}$$

We now want to show that $(d_1 \circ d_2) \circ d_3 = d_1 \circ (d_2 \circ d_3)$. Proceeding as we did before, we can write the composition $(d_1 \circ d_2) \circ d_3$ as:

$$\begin{aligned}\begin{pmatrix} \mathbf{x}_{k+1} \\ \mathbf{z}_{k+1} \\ \mathbf{s}_{k+1} \end{pmatrix} &= \begin{pmatrix} \mathbf{A} & 0 & 0 \\ \mathbf{FC} & \mathbf{E} & 0 \\ 0 & \mathbf{IG} & \mathbf{H} \end{pmatrix} \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix} + \begin{pmatrix} \mathbf{B} \\ 0 \\ 0 \end{pmatrix} \mathbf{u}_k \\ \mathbf{w}_k &= (0 \quad 0 \quad \mathbf{J}) \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix}\end{aligned}$$

We now want to write the composition $d_1 \circ (d_2 \circ d_3)$. To do so, we start by writing $(d_2 \circ d_3)$ as:

$$\begin{aligned}\begin{pmatrix} \mathbf{z}_{k+1} \\ \mathbf{s}_{k+1} \end{pmatrix} &= \begin{pmatrix} \mathbf{E} & 0 \\ \mathbf{IG} & \mathbf{H} \end{pmatrix} \begin{pmatrix} \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix} + \begin{pmatrix} \mathbf{F} \\ 0 \end{pmatrix} \mathbf{y}_k \\ \mathbf{w}_k &= (0 \quad \mathbf{J}) \begin{pmatrix} \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix}\end{aligned}$$

From this we can write $d_1 \circ (d_2 \circ d_3)$ as:

$$\begin{aligned} \begin{pmatrix} \mathbf{x}_{k+1} \\ \mathbf{z}_{k+1} \\ \mathbf{s}_{k+1} \end{pmatrix} &= \begin{pmatrix} \mathbf{A} & 0 & 0 \\ \mathbf{FC} & \mathbf{E} & 0 \\ 0 & \mathbf{IG} & \mathbf{H} \end{pmatrix} \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix} + \begin{pmatrix} \mathbf{B} \\ 0 \\ 0 \end{pmatrix} \mathbf{u}_k \\ \mathbf{w}_k &= (0 \quad 0 \quad \mathbf{J}) \begin{pmatrix} \mathbf{x}_k \\ \mathbf{z}_k \\ \mathbf{s}_k \end{pmatrix}, \end{aligned}$$

showing associativity.

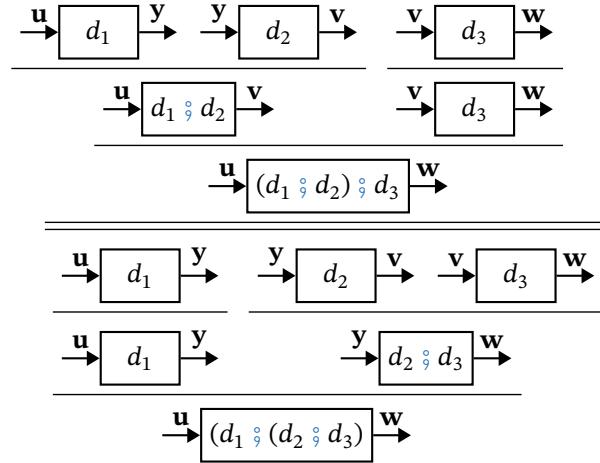


Figure 6.3.: Associativity law for the composition of discrete-time linear systems.

Induced n -ary multiplication

Given a semigroup $\langle \mathbf{S}, \circ \rangle$, for each $n \in \mathbb{N}$, we can define an induced n -ary multiplication operation

$$\mathbf{S}^n \rightarrow \mathbf{S}, \langle s_1, s_2, \dots, s_n \rangle \mapsto s_1 \circ s_2 \dots \circ s_n.$$

Thanks to the associative law, this is well-defined – that is, we do not need to set parentheses. We will say that an element $s \in \mathbf{S}$ is an n -fold multiplication if it is in the image of this n -ary multiplication operation. At times, we may not wish to specify the arity of the multiplication, in which case we just speak of a multiplication.

Code exercises

Remark 6.11. In this course we assume, unless otherwise noted, that you have done the previous exercises, so that you can build on top of the code that you already have. The first exercise and tutorial was in Section 1.3.

We define two classes: [Semigroup](#) and [FiniteSemigroup](#). Their relations are shown in Section 6.1.

A [Semigroup](#) is something that has a carrier set, which is a [Setoid](#). It also has a compose function.

Listing 16: The `Semigroup` interface.

```
class Semigroup(ABC):
    @abstractmethod
    def carrier(self) -> Setoid:
        ...

    @abstractmethod
    def composition(self) -> Mapping:
        ...
```

Code exercise 6 (`TestFiniteSemigroupConstruct`). Given a finite set, construct the free semigroup. Implement the interface in Listing 17

Listing 17: The `FiniteSemigroupConstruct` interface.

```
class FiniteSemigroupConstruct(ABC):
    @abstractmethod
    def free(self, fs: FiniteSet) -> FreeSemigroup:
        """ Construct the free semigroup on a set. """
```

A `FreeSemigroup` is a semigroup with an additional method, `unit()`, which converts an element of the carrier set to an element of the free group. (e.g. from element to a list with one element.)

Listing 18: The `FreeSemigroup` interface.

```
class FreeSemigroup(Semigroup, ABC):

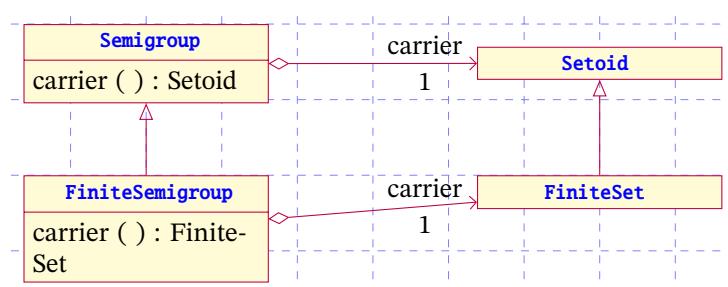
    @abstractmethod
    def unit(self, a: Element) -> Element:
        """ From an element of the carrier, returns the element of the free
        ↵ semigroup """
```

A `FiniteSemigroup` is a special `Semigroup` whose carrier is a `FiniteSet`.

Listing 19: The `FiniteSemigroup` interface.

```
class FiniteSemigroup(Semigroup, ABC):
    @abstractmethod
    def carrier(self) -> FiniteSet:
        ...

    @abstractmethod
```



```
def composition(self) -> FiniteMap:
    ...
```

The data format to define a finite semigroup is given in Listing 20.

The data structure contains two fields:

1. `carrier` is the serialization of a `FiniteSet A`.
2. `compose` is the serialization of a `FiniteMap`. The source for this map must be a product of sets (`A × A`), and the target must be equal to `A`.

```
carrier:
  elements: [0, 1]
composition:
  source:
    product:
      - elements: [0, 1]
      - elements: [0, 1]
  target:
    - elements: [0, 1]
  values:
    - [[0, 0], 0]
    - [[0, 1], 0]
    - [[1, 0], 0]
    - [[1, 1], 1]
```

Listing 20

Code exercise 7 (`TestFiniteSemigroupRepresentation`). Create a function to load the data, with the interface shown in Listing 21. Hint: you want to reuse the code you already have to load a `FiniteSet` and a `FiniteMap`.

Listing 21: The `FiniteSemigroupRepresentation` interface.

```
class FiniteSemigroupRepresentation(ABC):
    @abstractmethod
    def load(self, h: IOHelper, s: FiniteSemigroup_desc) -> FiniteSemigroup:
        """ Load the data """

    @abstractmethod
    def save(self, h: IOHelper, m: FiniteSemigroup) -> FiniteSemigroup_desc:
        """ Save the data """
```

6.2. Monoids

Watch online video (13 minutes).

Monoids

- A monoid is a semigroup with a neutral element.

Definition (Monoid). A monoid \mathbf{M} is:

Constituents

1. a set \mathbf{M} ;
2. a binary operation $\circ : \mathbf{M} \times \mathbf{M} \rightarrow \mathbf{M}$;
3. a specified element $\text{id} \in \mathbf{M}$, called *neutral element*.

Conditions

1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z) \quad \forall x, y, z \in \mathbf{M}$;
2. Neutrality Laws: $\text{id} \circ x = x = x \circ \text{id} \quad \forall x \in \mathbf{M}$.

Quiz: Which of these semigroups have a neutral element and are therefore monoids?

$(\mathbb{N}, +)$	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{N}, \cdot)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
$(\mathbb{R}, +)$	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{R}, \cdot)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{N}, \max)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{N}, \min)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$

Quiz: Which of these semigroups have a neutral element and are therefore monoids?

$(\mathbb{N}, +)$	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{N}, \cdot)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{N} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
$(\mathbb{R}, +)$	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{R}, \cdot)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{R}, \max)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$
(\mathbb{R}, \min)	$\text{id} :$	$\text{id} :$	$\forall x \in \mathbb{R} \text{ : } \text{id} \circ x = x = x \circ \text{id}$

Quiz: Which of these semigroups have a neutral element and are therefore monoids?

$((x \in \mathbb{N} : x \geq 2021), +)$	$\text{id} :$	$\text{id} :$	$\forall x \in ((x \in \mathbb{N} : x \geq 2021), +) \text{ : } \text{id} \circ x = x = x \circ \text{id}$
$((x \in \mathbb{N} : x \geq 2021), \cdot)$	$\text{id} :$	$\text{id} :$	$\forall x \in ((x \in \mathbb{N} : x \geq 2021), \cdot) \text{ : } \text{id} \circ x = x = x \circ \text{id}$
$((x \in \mathbb{N} : x \geq 2021), \max)$	$\text{id} :$	$\text{id} :$	$\forall x \in ((x \in \mathbb{N} : x \geq 2021), \max) \text{ : } \text{id} \circ x = x = x \circ \text{id}$
$((x \in \mathbb{N} : x \geq 2021), \min)$	$\text{id} :$	$\text{id} :$	$\forall x \in ((x \in \mathbb{N} : x \geq 2021), \min) \text{ : } \text{id} \circ x = x = x \circ \text{id}$



Algebraic structures are often defined in *layers*. For example, in the definition of semigroup, we start with a set \mathbf{S} as a basic building block, and we add a layer of structure to it, namely a multiplication operation $\circ : \mathbf{S} \times \mathbf{S} \rightarrow \mathbf{S}$. The multiplication operation for semigroups was not only a new *structure* that we added, but we also required that this structure obey a *condition*, namely that it satisfy the associative law. One might also say that the multiplication operation was a new *constituent* or a new *datum*, and that satisfying the associative law is a *property*. Mathematicians often use such words in an intuitive, non-rigorous way as a tool for structuring their thinking. We will do the same. For clarity, we will aim to stick with the words *constituents* and *conditions*. Roughly speaking, we think of constituents as building blocks, and we think of conditions as rules for how those blocks fit together and behave.

Using the constituent vs. condition distinction we will, in particular, present some definitions in the following succinct, list-like fashion:

Definition 6.12 (Monoid). A *monoid* \mathbf{M} is:

Constituents

1. A set \mathbf{M} ;
2. A binary operation $\circ : \mathbf{M} \times \mathbf{M} \rightarrow \mathbf{M}$;
3. A specified element $\text{id} \in \mathbf{M}$, called *neutral element*.

Conditions

1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z) \quad \forall x, y, z \in \mathbf{M}$;
2. Neutrality Laws: $\text{id} \circ x = x = x \circ \text{id} \quad \forall x \in \mathbf{M}$.

Remark 6.13. The way that we presented the definition of a monoid is certainly not unique. For example, we could have done the following.

A *monoid* \mathbf{M} is:

Constituents

1. a semigroup $\langle \mathbf{M}, \circ \rangle$;
2. a specified element $\text{id} \in \mathbf{M}$, called *neutral element*.

Conditions

1. Neutrality laws: $\text{id} \circ x = x = x \circ \text{id}$.

In this version, two constituents and one condition from Definition 6.12 are “compressed” into the information that we are using here a semigroup as a constituent. This kind of “compression” has its pros and cons; depending on the context will

use it to varying degrees.

There is a similar dilemma when considering the software interfaces to describe these structures. In terms of software engineering, the two strategies are *composition* (a monoid has a semigroup as a constituent) and *inheritance* (a monoid is a semigroup with additional data).

Example 6.14. Consider $\langle \mathbb{R}, +, 0 \rangle$.

This is a monoid, since, for all $x, y, z \in \mathbb{R}$, we have

$$(x + y) + z = x + (y + z),$$

and

$$x + 0 = x = 0 + x.$$

Example 6.15. The set \mathbb{Z} , together with the operation of multiplication of whole numbers, forms a monoid. The neutral element is the number 1.

Example 6.16. Consider $\langle \mathbf{B}, \wedge \rangle$ as in Example 6.6, and consider T as neutral element. This forms a monoid, since $b \wedge T = b = T \wedge b$, for all $b \in \mathbf{B}$.

Lemma 6.17. Let $\langle \mathbf{S}, \circ \rangle$ be a semigroup. If there exists elements $1 \in \mathbf{S}$ and $1' \in \mathbf{S}$ such that $\langle \mathbf{S}, \circ, 1 \rangle$ and $\langle \mathbf{S}, \circ, 1' \rangle$ are each monoids, then $1 = 1'$ must hold. In other words, the neutral element of a monoid is uniquely determined by the underlying semigroup structure.

Graded exercise 3 (UniqueNeutralMonoid). Prove Lemma 6.17.

Example 6.18. 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

$$\max(x, 0) = x = \max(0, x).$$

Remark 6.19. Note that in the above example, we could have just as well instead considered the set $\mathbb{R}_{\geq 7.5}$ of real numbers greater than 7.5, together with “max” as composition and 7.5 as neutral element. In other words, we can choose any real number $a \in \mathbb{R}$ and obtain a monoid $\langle \mathbb{R}_{\geq a}, \max, a \rangle$.

Example 6.20. We can extend the semigroup introduced in Example 6.7 by defining the notion of an empty string $\langle \rangle$. This functions as a neutral element, so we obtain a monoid.

Example 6.21 (Transition functions).

Definition 6.22 (Continuous-time dynamical system). A dynamical system on \mathbb{R}^n may be defined by a function

$$f : \mathbb{R}^n \rightarrow \mathbb{R}^n. \tag{6.10}$$

A trajectory of a dynamical system is a function $x : \mathbb{R} \rightarrow \mathbb{R}^n$ such that

$$\dot{x}_t = f(x_t). \quad (6.11)$$

We use the notation \dot{x} to abbreviate dx/dt .

Suppose that we are dealing with dynamical systems such that for any point x_0 , there is exactly one trajectory beginning at x_0 . (For this we need to put reasonable constraints on the function f .)

We can then ask the following: given a point x , where would its trajectory be after δ ? This question induces a family of functions T_δ , called transition functions. For each particular δ , we have a function

$$T_\delta : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

that maps a point to its position δ in the future.

We can spot here a semigroup structure. Suppose we want to know the position of a point $\delta_1 + \delta_2$ in the future. We can take T_{δ_1} and compose it with T_{δ_2} ; or take directly $T_{\delta_1 + \delta_2}$. By construction we will have that

$$T_{\delta_1 + \delta_2} = T_{\delta_1} \circ T_{\delta_2} \quad (6.12)$$

We can also easily prove associativity:

$$\begin{aligned} T_{\delta_1} \circ (T_{\delta_2} \circ T_{\delta_3}) &= T_{\delta_1} \circ T_{\delta_2 + \delta_3} \\ &= T_{\delta_1 + \delta_2 + \delta_3} \\ &= T_{\delta_1 + \delta_2} \circ T_{\delta_3} \\ &= (T_{\delta_1} \circ T_{\delta_2}) \circ T_{\delta_3}. \end{aligned} \quad (6.13)$$

This shows that the set of transition functions for a particular system with the operation of function composition form a semigroup.

This semigroup is a monoid because there is an identity. The identity is T_0 , the map that tells us what happens after 0 seconds. That is $\text{id}_{\mathbb{R}^n}$, the identity on \mathbb{R}^n . To show that $T_0 = \text{id}_{\mathbb{R}^n}$ is an identity, we can fix any δ and substituting in (6.12) we have

$$\begin{aligned} T_{\delta+0} &= T_\delta \circ T_0 \\ &= T_\delta \end{aligned}$$

Exercise 5. [Discrete-time linear systems] In Section 6.1 we have constructed the semigroup of linear discrete-time dynamical systems.

1. Show that it is not a monoid, by showing that one cannot find an identity.
2. Suggest an extension to Def. 6.10 so that you can obtain a monoid.

See solution on page 91.

Listing 22: The `Monoid` interface.

```
class Monoid(Semigroup, ABC):
    @abstractmethod
    def identity(self) -> Element:
        ...
```

Listing 23: The `FiniteMonoid` interface.

```
class FiniteMonoid(Monoid, FiniteSemigroup, ABC):
    """
```

Representation

The file format for monoids is an extension of those for semigroups. There is an extra field `neutral` which gives you the neutral element. Listing 24 shows an example.

Listing 24

```
carrier:
    elements: [0, 1]
composition:
    source:
        product:
            - elements: [0, 1]
            - elements: [0, 1]
    target:
        - elements: [0, 1]
values:
    - [[0, 0], 0]
    - [[0, 1], 0]
    - [[1, 0], 0]
    - [[1, 1], 1]
neutral: 1
```

Code exercise 8 (`TestFiniteMonoidRepresentation`). Create a function to load the data.

Listing 25: The `FiniteMonoidRepresentation` interface.

```
class FiniteMonoidRepresentation(ABC):
    @abstractmethod
    def load(self, h: IOHelper, s: FiniteMonoid_desc) -> FiniteMonoid:
        """ Load the data """

    @abstractmethod
    def save(self, h: IOHelper, m: FiniteMonoid) -> FiniteMonoid_desc:
        """ Save the data """
```

6.3. Groups

Watch online video (10 minutes).

Groups

Definition (Group). A group is a monoid together with an “inverse” operation.

In more detail, a group \mathbf{G} is

Constituents

1. a set \mathbf{G} ;
2. a binary operation $\circ : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$, called *composition*;
3. a specified element $\text{id} \in \mathbf{G}$;

new 4. a map $\text{inv} : \mathbf{G} \rightarrow \mathbf{G}$ called “inverse”.

Conditions

1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z)$;
2. Neutrality laws: $\text{id} \circ x = x = x \circ \text{id}$;
3. Inverse law: $\text{inv}(x) \circ x = \text{id} = \text{inv}(x) \circ x$.

Quiz: which of these monoids are groups? Give the expression for the inverse operation.

$(\mathbb{R}, +, 0)$ $x \mapsto -x$ $(\mathbb{N}, +, 0)$ $x \mapsto x$ $(\mathbb{N}, \max, 0)$ $x \mapsto x$

$(\mathbb{R}, \cdot, 1)$ $x \mapsto \frac{1}{x}$ $(\mathbb{N}, \cdot, 1)$ $x \mapsto x$ $(\mathbb{N}, \min, 1)$ $x \mapsto x$



Definition 6.23 (Group). A *group* is a monoid together with an “inverse” operation. In more detail, a group \mathbf{G} is

Constituents

1. a set \mathbf{G} ;
2. a binary operation $\circ : \mathbf{G} \times \mathbf{G} \rightarrow \mathbf{G}$, called *composition*;
3. a specified element $\text{id} \in \mathbf{G}$;
4. a map $\text{inv} : \mathbf{G} \rightarrow \mathbf{G}$, called *inverse*.

Conditions

1. Associative law: $(x \circ y) \circ z = x \circ (y \circ z)$, $\forall x, y, z \in \mathbf{G}$;
2. Neutrality laws: $\text{id} \circ x = x = x \circ \text{id}$, $\forall x \in \mathbf{G}$;
3. Inverse laws:

$$\text{inv}(x) \circ x = \text{id} = x \circ \text{inv}(x), \quad \forall x \in \mathbf{G}. \quad (6.14)$$

Example 6.24. The following is a group: the set \mathbb{Z} , together with addition as the composition operation, the element 0 as neutral element, and “taking the negative” as the inverse operation:

$$\text{inv}(x) := -x, \quad \forall x \in \mathbb{Z}. \quad (6.15)$$

Example 6.25. The monoid $\langle \mathbb{R}_{\setminus \{0\}}, \cdot, 1 \rangle$ becomes a group when equipped with the inverse operation defined by

$$\text{inv}(x) := \frac{1}{x}, \quad \forall x \in \mathbb{R}. \quad (6.16)$$

Example 6.26. The monoid $\langle \mathbf{B}, \wedge, \top \rangle$ from Example 6.16 cannot become a group, because there cannot be an inverse for \perp . In other words, there is no inverse for which $\text{inv}(\perp) \wedge \perp = \top$.

Example 6.27. Consider the set $\mathbf{G} := \{1, e^{\frac{1}{3}2\pi i}, e^{\frac{2}{3}2\pi i}\} \subseteq \mathbb{C}$. Taking the usual multiplication of complex numbers as the composition operation, these three elements from a group.

Graded exercise 4 (GroupWithThreeElements). In Example 6.27, what is the neutral element? What is the inverse operation? Draw the composition table for this group.

Lemma 6.28. Let $\mathbf{G} = \langle \mathbf{G}, \circ, \text{id}, \text{inv} \rangle$ be a group and let $x, y \in \mathbf{G}$. If x and y satisfy the equation

$$x \circ y = \text{id}. \quad (6.17)$$

Then $y = \text{inv}(x)$ and $x = \text{inv}(y)$.

Proof. If $x \circ y = \text{id}$, then, by composing both sides of this equation with $\text{inv}(x)$ from the left, and using associativity to remove brackets, we find $\text{inv}(x) \circ x \circ y = \text{id} \circ \text{inv}(x)$. Applying the inverse laws on the left-hand side, we obtain $\text{id} \circ y = \text{id} \circ \text{inv}(x)$, and using the neutrality laws on both side of this equation yields $y = \text{inv}(x)$. The fact that $x = \text{inv}(y)$ may be proved similarly. \square

Corollary 6.29. Let $\mathbf{M} = \langle \mathbf{M}, \circ, \text{id} \rangle$ be a monoid. If inv and $\widetilde{\text{inv}}$ are both operations of inverse which make \mathbf{M} into a group, then $\text{inv} = \widetilde{\text{inv}}$ holds. In other words, if a monoid can be made into a group by adding an inverse operation, then the resulting group is uniquely determined by the underlying monoid.

Proof. Let $x \in \mathbf{M}$. Since $\langle \mathbf{M}, \circ, \text{id}, \widetilde{\text{inv}} \rangle$ is a group, we have $x \circ \widetilde{\text{inv}}(x) = \text{id}$. Also $\langle \mathbf{M}, \circ, \text{id}, \text{inv} \rangle$ is a group, and in terms of this group, the equation $x \circ \text{inv}(x) = \text{id}$ implies, by Lemma 6.28, that $\widetilde{\text{inv}}(x) = \text{inv}(x)$, by thinking of $\widetilde{\text{inv}}(x)$ in the role of y from Lemma 6.28. Since $x \in \mathbf{M}$ was arbitrary, $\widetilde{\text{inv}} = \text{inv}$ as functions on \mathbf{M} . \square

Lemma 6.30. Let $\langle \mathbf{G}, \circ, \text{id}, \text{inv} \rangle$ be a group. Then

1. $\text{inv}(\text{id}) = \text{id}$;
2. $\text{inv}(\text{inv}(x)) = x, \quad \forall x \in \mathbf{G}$;
3. $\text{inv}(x \circ y) = \text{inv}(y) \circ \text{inv}(x), \quad \forall x, y \in \mathbf{G}$.

Graded exercise 5 (GroupInverseProperties). Prove Lemma 6.30.

Example 6.31 (Square matrices with full rank). Fix an integer $n \geq 1$ and consider the set of square matrices with full rank $\mathbf{A} \in \mathbb{R}^{n \times n}$. This set, equipped with the usual matrix multiplication as the binary operation ($\mathbf{A} \circ \mathbf{B} := \mathbf{AB}$), the identity matrix I as the neutral element, and matrix inverse as the inverse ($\text{inv}(\mathbf{A}) := \mathbf{A}^{-1}$), forms a group.

Example 6.32 (Orthogonal matrices). Fix an integer $n \geq 1$ and consider the set of *orthogonal* matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$. Orthogonal matrices are real, square matrices with orthonormal columns and rows. One way to express orthogonality of a matrix is:

$$\mathbf{A}^\top \mathbf{A} = \mathbf{A} \mathbf{A}^\top = I,$$

where I is the identity matrix. The set $\mathbb{R}^{n \times n}$ equipped with matrix multiplication as a binary operation, the identity matrix as the neutral element, and the transposition $(\cdot)^\top$ (which for this specific type of matrices corresponds to the inverse) forms a group, usually denoted $O(n)$. Any orthogonal matrix \mathbf{A} has the property $\det(\mathbf{A}) \in \{-1, 1\}$. The subset of orthogonal $n \times n$ matrices with determinant 1 forms the so-called *special* orthogonal group $SO(n)$.



7. Morphisms

In this chapter we look at morphisms, which are maps between two semigroups (or monoids, groups) that “preserve the structure”.

7.1 Semigroup Morphisms	77
7.2 Monoid morphisms	81
7.3 Group morphisms	82

7.1. Semigroup Morphisms

Definition 7.1 (Semigroup morphism). A morphism $F : S \rightarrow T$ between semigroups

$$S = \langle S, \circ_S \rangle \quad \text{and} \quad T = \langle T, \circ_T \rangle \quad (7.1)$$

is a function $F : S \rightarrow T$ such that for all $x, y \in S$,

$$F(x \circ_S y) = F(x) \circ_T F(y). \quad (7.2)$$

Note that we use $F : S \rightarrow T$ when we want to highlight the function between sets, and we use $F : S \rightarrow T$ when we want to highlight the relation between semigroup structures. We think of (7.2) as a way of saying that the function $F : S \rightarrow T$ is *compatible* with the multiplication operations on S and T , respectively.

Definition 7.2 (Identity morphism). Let \mathbf{S} be a semigroup. The identity function $\text{Id}_{\mathbf{S}} : \mathbf{S} \rightarrow \mathbf{S}$ is always a morphism of semigroups. We can easily check that (7.2) is satisfied:

$$\text{Id}_{\mathbf{S}}(x \circ_{\mathbf{S}} y) = x \circ_{\mathbf{S}} y = \text{Id}_{\mathbf{S}}(x) \circ_{\mathbf{S}} \text{Id}_{\mathbf{S}}(y). \quad (7.3)$$

We call this the *identity morphism* of \mathbf{S} .

Definition 7.3 (Semigroup isomorphism). A morphism of semigroups $F : \mathbf{S} \rightarrow \mathbf{T}$ is called a *semigroup isomorphism* if there exists a morphism of semigroups $G : \mathbf{T} \rightarrow \mathbf{S}$ such that

$$F \circ G = \text{Id}_{\mathbf{S}} \quad \text{and} \quad G \circ F = \text{Id}_{\mathbf{T}}. \quad (7.4)$$

Lemma 7.4. The composition of semigroup morphisms is a morphism:

$$\frac{F : \mathbf{S} \rightarrow \mathbf{T} \quad G : \mathbf{T} \rightarrow \mathbf{U}}{F \circ G : \mathbf{S} \rightarrow \mathbf{U}} \quad (7.5)$$

Exercise 6. Prove Lemma 7.4.

See solution on page 91.

Example 7.5 (Logarithms and exponentials). The positive reals with multiplication $\langle \mathbb{R}_{>0}, \cdot \rangle$ is a semigroup. The reals with addition $\langle \mathbb{R}, + \rangle$ is a semigroup.

Now consider as a bridge between the two: the logarithmic function.

$$\log : \mathbb{R}_{>0} \rightarrow \mathbb{R},$$

and its inverse

$$\exp : \mathbb{R} \rightarrow \mathbb{R}_{>0}.$$

We already know that these are inverse of each other:

$$\exp \circ \log = \text{Id}_{\mathbb{R}}$$

$$\log \circ \exp = \text{Id}_{\mathbb{R}_{>0}}$$

We can verify that \log is also a semigroup morphism, because of this property of the logarithms:

$$\log(a \cdot b) = \log(a) + \log(b). \quad (7.6)$$

Because \log is a bijection and \exp is its inverse, it already follows that \exp is a morphism in the opposite direction. Alternatively we can see that is the case because of the property of the exponentials:

$$\exp(c + d) = \exp(c) \cdot \exp(d). \quad (7.7)$$

Equations (7.6) and (7.7) are both (7.2) in disguise.

USASCII code chart																	
		Column		0		o		o		o		o		o			
		b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	0	1	1	1	1	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	0	1	0	2	2	2	2	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	1	3	3	3	3	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	4	4	4	4	4	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	5	5	5	5	5	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	6	6	6	6	6	ACK	SYN	8	6	F	V	f	v
0	1	1	1	7	7	7	7	7	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	8	8	8	8	8	BS	CAN	(8	H	X	h	x
1	0	0	1	9	9	9	9	9	9	HT	EM)	9	I	Y	i	y
1	0	1	0	10	10	10	10	10	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	11	11	11	11	11	VT	ESC	+	:	K	C	k	c
1	1	0	0	12	12	12	12	12	12	FF	FS	-	<	L	\	l	\
1	1	0	1	13	13	13	13	13	13	CR	GS	=	=	M	J	m	j
1	1	1	0	14	14	14	14	14	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	15	15	15	15	15	SI	US	/	?	O	-	o	DEL

Figure 7.1.: 7-bit US-ASCII encoding.

Example 7.6 (ASCII code). ASCII encoding takes any alphanumerical characters and symbols into a number between 0 and 127 (Fig. 7.1). Let's call \mathbb{A} the set of those 128 symbols. We can see ASCII encoding as a semigroup morphism of \mathbb{A}^* to the free semigroup on the integers $[0, 127]^*$:

$$\text{ASCII} : \mathbb{A}^* \rightarrow [0, 127]^*$$

Because we can also go back, by using the inverse function,

$$\text{ASCII}^{-1} : [0, 127]^* \rightarrow \mathbb{A}^*,$$

ASCII encoding is also an isomorphisms of semigroups.

Example 7.7 (ASCII code to binary). Currently, computers use binary to store data. (There were, in fact, *trinary* computers.) In Fig. 7.1, you can see represented also the binary encoding of each character. Therefore, we can see ASCII as a morphism between \mathbb{A}^* and binary strings $\{0, 1\}^*$.

Exercise 7. Show that the morphism

$$\text{ASCII} : \mathbb{A}^* \rightarrow \{0, 1\}^*$$

is *not* an isomorphism.

See solution on page 91.

Example 7.8 (Morse code). Consider the Morse code: a way to encode the letters and numerals to an alphabet of dots (\bullet) and dashes ($-$). The encoding is shown in Table 7.1. Here, the alphabet \mathcal{A} is the letters A–Z and the numbers 0–9. There is no difference between upper and lower case, and there are no punctuation marks.

Transcribing a text in Morse code is not just a matter of creating the right sequence of dots and dashes. The standard also requires a certain timing of the events. If the length of \bullet is 1, then the length of $-$ must be 3. There must be an interval of 3 between letters, and 7 between words.

Therefore, there are 5 symbols in the Morse alphabet (Table 7.2); each representing a *signal*.

Define now the extended alphabet \mathcal{A}' to be the union of \mathcal{A} and the set $\{\textcolor{red}{|}, \textcolor{green}{|}\}$, where $\textcolor{red}{|}$ is interletter space, and $\textcolor{green}{|}$ is inter-word space.

Therefore, to encode the sentence

“I am well”

we first transform it to upper case

“I AM WELL”

Then we note the inter-letter space and the interword spaces:

I $\textcolor{red}{|}$ A $\textcolor{red}{|}$ M $\textcolor{red}{|}$ W $\textcolor{red}{|}$ E $\textcolor{red}{|}$ L $\textcolor{red}{|}$ L

At this point we can substitute the Morse code to obtain

$$\bullet s_1 \bullet s_7 \bullet s_1 - s_3 - s_1 - s_7 \bullet s_1 - s_1 - s_3 \bullet s_3 - s_3 -$$

Table 7.1.: Morse encoding

A	..	
B	—...	
C	—...	
D	—..	
E	.	
F	
G	---	
H	
I	..	0
J	-----	1
K	---	2
L	---.	3
M	--	4
N	--.	5
O	---	6
P	---.	7
Q	---.	8
R	---.	9
S	...	
T	-	
U	...	
V	...	
W	---	
X	---	
Y	---	
Z	---	

Table 7.2.: 5 symbols for Morse encoding

•	beep of length ℓ	
—	beep of length 3ℓ	
s_1	silence of length ℓ	
s_3	silence of length 3ℓ	
s_7	silence of length 7ℓ	

In signal space—what somebody would hear—this becomes



With this representation it is clear that 5 symbols are redundant: if we have a 1-period beep and a 1-period silence, we can obtain the 3-period silence and beeps and the 7-period silence.

In the end, the Morse alphabet is *binary* in the sense that it all reduces to two symbols: not $\{\bullet, -\}$ but rather the alphabet $\{\blacksquare, \blacksquare\}$.

Exercise 8. [MorseMorphism] We have seen that Morse code transforms a word in the alphabet

$$\mathcal{A} = (\text{A to Z}) \cup (0 \text{ to } 9) \cup \{\blacksquare\}$$

to the alphabet

$$\mathcal{B} = \{\blacksquare, \blacksquare\}$$

Is this map a morphism of semigroups?

See solution on page 92.

Example 7.9 (Transition function, continuation of Example 6.21). Consider the map $f : \mathbb{R}_{\geq 0} \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^n)$ that associates to a delta δ its transition function T_δ . Re-reading (6.12), we can see that it is a morphism between the semigroup $\langle \mathbb{R}_{\geq 0}, + \rangle$ and the semigroup of endomorphisms of \mathbb{R}^n .

Graded exercise 6 (IsoViaTables). Consider the set $\mathbf{A} = \{\circlearrowleft, \approx\}$ and the following three composition tables, each of which defines a semigroup structure on \mathbf{A} .

\circlearrowleft_1	\circlearrowleft	\approx
\circlearrowleft	\circlearrowleft	\circlearrowleft
\approx	\approx	\approx

\circlearrowleft_2	\circlearrowleft	\approx
\circlearrowleft	\circlearrowleft	\approx
\approx	\approx	\approx

\circlearrowleft_3	\circlearrowleft	\approx
\circlearrowleft	\approx	\approx
\approx	\approx	\approx

Which of the three semigroups defined in this way are isomorphic to each other? Justify your answer.

Graded exercise 7 (SemigroupUpToIso). How many different non-isomorphic semigroups are there with precisely one element? How many with precisely two elements? Can you prove your answer?

Graded exercise 8 (CharacterizeSemigroupIsos). Let $F : S \rightarrow T$ be a morphism of semigroups. Prove that F is an isomorphism of semigroups if and only if the function $F : S \rightarrow T$ is bijective.

7.2. Monoid morphisms

We have defined semigroup morphism. A monoid morphism has the same properties, and one additional one: the constraint that it transforms identities to identities.

Definition 7.10 (Monoid morphism). A morphism $F : \mathbf{M} \rightarrow \mathbf{N}$ between monoids

$$\mathbf{M} = \langle \mathbf{M}, \circ_{\mathbf{M}}, \text{id}_{\mathbf{M}} \rangle \quad \text{and} \quad \mathbf{N} = \langle \mathbf{N}, \circ_{\mathbf{N}}, \text{id}_{\mathbf{N}} \rangle \quad (7.8)$$

is a function $F : \mathbf{M} \rightarrow \mathbf{N}$ such that for all x, y in \mathbf{M} ,

$$F(x \circ_{\mathbf{M}} y) = F(x) \circ_{\mathbf{N}} F(y) \quad (7.9)$$

and

$$F(\text{id}_{\mathbf{M}}) = \text{id}_{\mathbf{N}} \quad (7.10)$$

Example 7.11. The set $\mathbf{M} = \{-1, 0, 1\}$, together with multiplication of whole numbers and with 1 as neutral element, forms a monoid. The inclusion map $\mathbf{M} \rightarrow \mathbb{Z}$ is a morphism of monoids.

Example 7.12. Consider the monoid \mathbf{N} in Example 6.20 and the monoid $\mathbf{M} = \langle \mathbb{N}, +, 0 \rangle$. Define a map

$$\text{length} : \mathbf{N} \rightarrow \mathbf{M}$$

which maps each string to its length. This is a monoid morphism, since:

$$\text{length}(x \circ_{\mathbf{N}} y) = \text{length}(x) \circ_{\mathbf{M}} \text{length}(y).$$

In other words, the length of the concatenation of two strings is the sum of the lengths of the two strings.

Definition 7.13 (Identity morphism). Let \mathbf{M} be a monoid. Similar to the case of semigroups, the identity function induces a morphism of monoids $\text{Id}_{\mathbf{M}} : \mathbf{M} \rightarrow \mathbf{M}$.

Definition 7.14 (Monoid morphism). A morphism of monoids $F : \mathbf{M} \rightarrow \mathbf{N}$ is called a *monoid isomorphism* if there is a morphism of monoids $G : \mathbf{N} \rightarrow \mathbf{M}$ such that

$$F \circ G = \text{Id}_{\mathbf{M}} \quad \text{and} \quad G \circ F = \text{Id}_{\mathbf{N}}. \quad (7.11)$$

Graded exercise 9 ([MorphismMonoidIsomorphism](#)). Prove: a morphism of monoids $F : \mathbf{M} \rightarrow \mathbf{N}$ is an isomorphism of monoids if and only if the function $F : \mathbf{M} \rightarrow \mathbf{N}$ is bijective.

Graded exercise 10(TraceAndDeterminant). Let $\mathbb{R}^{n \times n}$ denote the set of real $n \times n$ matrices. These form a monoid $\langle \mathbb{R}^{n \times n}, \cdot, \mathbb{1} \rangle$, with matrix multiplication as composition, and the identity matrix as neutral element. They also form a monoid $\langle \mathbb{R}^{n \times n}, +, 0 \rangle$ with matrix addition as composition, and the zero matrix as neutral element.

Recall that the trace of a real $n \times n$ matrix is the sum of its diagonal entries. This defines a function

$$\text{Tr} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}. \quad (7.12)$$

Computing the determinant also corresponds to a function

$$\text{Det} : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}. \quad (7.13)$$

We have seen that $\langle \mathbb{R}, \cdot, 1 \rangle$ and $\langle \mathbb{R}, +, 0 \rangle$ are two different monoid structures on \mathbb{R} .

1. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbb{1} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
2. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbb{1} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
3. Does Det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbb{1} \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
4. Does Det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, \cdot, \mathbb{1} \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
5. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, 0 \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
6. Does Tr define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, 0 \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?
7. Does Det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, 0 \rangle \rightarrow \langle \mathbb{R}, \cdot, 1 \rangle$?
8. Does Det define a morphism of monoids $\langle \mathbb{R}^{n \times n}, +, 0 \rangle \rightarrow \langle \mathbb{R}, +, 0 \rangle$?

Short answers (without proof) are fine.

7.3. Group morphisms

Definition 7.15 (Group morphism). A morphism $F : G \rightarrow H$ between groups

$$G = \langle \textcolor{brown}{G}, \circ_G, \text{id}_G, \text{inv}_G \rangle \quad \text{and} \quad H = \langle \textcolor{brown}{H}, \circ_H, \text{id}_H, \text{inv}_H \rangle \quad (7.14)$$

is a function $F : G \rightarrow H$ such that for all x, y in G ,

$$F(x \circ_G y) = F(x) \circ_H F(y). \quad (7.15)$$

Where are the equations

$$F(\text{id}_G) = \text{id}_H \quad (7.16)$$

and

$$F(\text{inv}_G(x)) = \text{inv}_H(F(x)) ? \quad (7.17)$$

They are not needed; they are implied by the group axioms.

Exercise 9. Prove that (7.16) and (7.17) are implied by Definition 7.15.

See solution on page 92.

Exercise 10. Let $\mathbf{G} = \{1, -1, i, -i\}$. Consider the group $\mathbf{G} = \langle \mathbf{G}, \cdot, 1, \text{inv}_{\mathbf{G}} \rangle$ and the group \mathbf{H} of all integers under addition. Prove that $F : \mathbf{H} \rightarrow \mathbf{G}$ such that $f(n) = i^n$ for all $n \in \mathbf{H}$ is a group morphism.

See solution on page 93.

Example 7.16. Consider the group

$$\mathbf{G} = \left\langle \mathbb{R}^{2 \times 2}, +, \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, - \right\rangle \quad (7.18)$$

of real 2×2 matrices, together with sum of matrices as a binary operation, “zero” matrix as the neutral element, and the “-” operation as inverse. Furthermore, consider the group $\langle \mathbb{R}, +, 0, - \rangle$. Taking the *trace* of a matrix corresponds to a group morphism. Indeed, the operation

$$\begin{aligned} \text{Tr} : \mathbb{R}^{2 \times 2} &\rightarrow \mathbb{R} \\ \begin{pmatrix} a & b \\ c & d \end{pmatrix} &\mapsto a + d \end{aligned}$$

satisfies the required condition:

$$\begin{aligned} \text{Tr} \left(\begin{pmatrix} a & b \\ c & d \end{pmatrix} \circ_{\mathbf{G}} \begin{pmatrix} e & f \\ g & h \end{pmatrix} \right) &= \text{Tr} \left(\begin{pmatrix} a+e & b+f \\ c+g & d+h \end{pmatrix} \right) \\ &= \text{Tr} \left(\begin{pmatrix} a & b \\ c & d \end{pmatrix} \right) \circ_{\mathbf{H}} \text{Tr} \left(\begin{pmatrix} e & f \\ g & h \end{pmatrix} \right) \\ &= (a+d) \circ_{\mathbf{H}} (e+h). \end{aligned}$$



8. Actions

8.1 Matrix groups	85
Special Euclidean group	86
8.2 Actions	87

8.1. Matrix groups

There are many *matrix groups* (Table 8.1) that represent linear transformations of a vector space that have some special properties.

Definition 8.1 (General linear group $\text{GL}(n)$). The general linear group of order n , written $\text{GL}(n)$, is the group of $n \times n$ invertible matrices.

We have used this in the past.

Definition 8.2 (General orthogonal group $\text{O}(n)$). The general orthogonal group of order n , written $\text{O}(n)$, is the group of $n \times n$ square matrices that satisfy

$$\mathbf{M}\mathbf{M}^T = \mathbf{M}^T\mathbf{M} = \mathbb{1}. \quad (8.1)$$

Definition 8.3 (Special linear group $\text{SL}(n)$). The special linear group of order n , written $\text{SL}(n)$, is the group of $n \times n$ invertible matrices with determinant equal to 1.

Table 8.1.: Matrix groups

$\text{GL}(n)$	General linear group	invertible matrices
$\text{O}(n)$	Orthogonal group	preserves length of vectors
$\text{E}(n)$	Euclidean groups	preserves distances and angles
$\text{SO}(n)$	Special orthogonal group	
$\text{SL}(n)$	Special linear group	
$\text{SE}(n)$	Special Euclidean group	

Definition 8.4 (Special orthogonal group $\text{SO}(n)$). The special orthogonal group of order n , written $\text{SO}(n)$, is the group of $n \times n$ square matrices that satisfy

$$\mathbf{M}\mathbf{M}^\top = \mathbf{M}^\top\mathbf{M} = \mathbb{1} \quad (8.2)$$

and $\det(\mathbf{M}) = 1$.

Definition 8.5 (General Euclidean group $\text{E}(n)$). The general Euclidean group of order n , written $\text{E}(n)$, is the group of $(n+1) \times (n+1)$ square matrices of the form

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \quad (8.3)$$

where $\mathbf{R} \in \text{O}(n)$ and $\mathbf{t} \in \mathbb{R}^n$.

Definition 8.6 (Special Euclidean group $\text{SE}(n)$). The special Euclidean group of order n , written $\text{SE}(n)$, is the group of $(n+1) \times (n+1)$ square matrices of the form

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \quad (8.4)$$

where $\mathbf{R} \in \text{SO}(n)$ and $\mathbf{t} \in \mathbb{R}^n$.

We also have 3 “special” versions: $\text{SL}(n)$, $\text{SO}(n)$, $\text{SE}(n)$. These are subgroups of those that have determinant equal to 1.

These matrix groups are also transformations of \mathbb{R}^n .

Special Euclidean group

The groups $\text{SE}(2)$ and $\text{SE}(3)$ are particular important in robotics because they represent the roto-translations of the plane and 3D space, respectively.

From (8.3) we know we can represent one by a pair $\langle \mathbf{R}, \mathbf{t} \rangle$, with $\mathbf{R} \in \text{SO}(n)$ and $\mathbf{t} \in \mathbb{R}^n$.

If we look at how matrices compose, we get

$$\begin{pmatrix} \mathbf{R}_2 & \mathbf{t}_2 \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{R}_1 & \mathbf{t}_1 \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}_2\mathbf{R}_1 & \mathbf{R}_2\mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{0} & 1 \end{pmatrix}. \quad (8.5)$$

The formula for composition is

$$\langle \mathbf{R}_1, \mathbf{t}_1 \rangle \circ_{\text{SE}(n)} \langle \mathbf{R}_2, \mathbf{t}_2 \rangle = \langle \mathbf{R}_2\mathbf{R}_1, \mathbf{R}_2\mathbf{t}_1 + \mathbf{t}_2 \rangle \quad (8.6)$$

The group $\text{SE}(n)$ induces a transformation on the points of \mathbb{R}^n . We are going to call this an *action*.

The action is the following function:

$$\begin{aligned}\text{apply} : \text{SE}(n) \times \mathbb{R}^n &\longrightarrow \mathbb{R}^n \\ \langle \langle \mathbf{R}, \mathbf{t} \rangle, \mathbf{p} \rangle &\longmapsto \mathbf{R}\mathbf{p} + \mathbf{t}\end{aligned}\tag{8.7}$$

Given a roto-translation and a point, the function returns the roto-translated point. We can also see this in matrix form as follows. We need to substitute for a point $\mathbf{p} \in \mathbb{R}^n$ a homogenous point $\begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} \in \mathbb{R}^{n+1}$.

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}\mathbf{p} + \mathbf{t} \\ 1 \end{pmatrix}\tag{8.8}$$

If we apply two rototranslations, first $\mathbf{x} = \langle \mathbf{R}_x, \mathbf{t}_x \rangle$ and then $\mathbf{y} = \langle \mathbf{R}_y, \mathbf{t}_y \rangle$, we find:

$$\begin{aligned}\text{apply}(\langle \mathbf{R}_y, \mathbf{t}_y \rangle, \text{apply}(\langle \mathbf{R}_x, \mathbf{t}_x \rangle, \mathbf{p})) &= \text{apply}(\langle \mathbf{R}_y, \mathbf{t}_y \rangle, \mathbf{R}_x\mathbf{p} + \mathbf{t}_x) \\ &= \mathbf{R}_y\mathbf{R}_x\mathbf{p} + \mathbf{R}_y\mathbf{t}_x + \mathbf{t}_y.\end{aligned}\tag{8.9}$$

It is easy to see that it is equal to compose the two transformations in the inverse order

$$\langle \mathbf{R}_x, \mathbf{t}_x \rangle \circ_{\text{SE}(n)} \langle \mathbf{R}_y, \mathbf{t}_y \rangle = \langle \mathbf{R}_y\mathbf{R}_x, \mathbf{R}_y\mathbf{t}_x + \mathbf{t}_y \rangle\tag{8.10}$$

and then apply it to the object

$$\text{apply}(\langle \mathbf{R}_y\mathbf{R}_x, \mathbf{R}_y\mathbf{t}_x + \mathbf{t}_y \rangle, \mathbf{p}) = \mathbf{R}_y\mathbf{R}_x\mathbf{p} + \mathbf{R}_y\mathbf{t}_x + \mathbf{t}_y.\tag{8.11}$$

Thus we have proved this property

$$\text{apply}(\mathbf{y}, \text{apply}(\mathbf{x}, \mathbf{p})) = \text{apply}(\mathbf{x} \circ \mathbf{y}, \mathbf{p}),\tag{8.12}$$

which is graphically reported in Fig. 8.1.

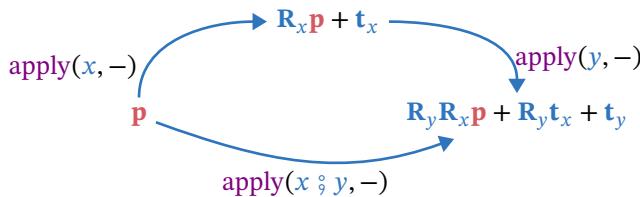


Figure 8.1.: Graphical representation of roto-translation action.

The notion of semigroup action generalizes this property.

8.2. Actions

There are actually 2 versions for actions: a covariant and a contravariant version, depending on the order of the transformations.

Definition 8.7 (Semigroup covariant action (preliminary version)). A *semigroup covariant action* of a semigroup \mathbf{S} onto a set \mathbf{A} is a map

$$\text{Covact} : \mathbf{S} \times \mathbf{A} \rightarrow \mathbf{A} \quad (8.13)$$

such that, for all $a \in \mathbf{A}$,

$$\text{Covact}(y, \text{Covact}(x, a)) = \text{Covact}(x \circ_{\mathbf{S}} y, a). \quad (8.14)$$

Definition 8.8 (Semigroup contravariant action (preliminary version)). A *semigroup contravariant action* of a semigroup \mathbf{S} onto a set \mathbf{A} is a map

$$\text{Contravact} : \mathbf{A} \times \mathbf{S} \rightarrow \mathbf{A} \quad (8.15)$$

such that, for all $a \in \mathbf{A}$,

$$\text{Contravact}(\text{Contravact}(a, y), x) = \text{Contravact}(a, x \circ_{\mathbf{S}} y). \quad (8.16)$$

This definition is the standard algebraic definition. We will compress it a bit using the language of category theory.

We need to introduce Dr. Haskell Curry (Fig. 8.2). His first name, Haskell, named the language. His last name, Curry, named the operation of *currying* that we are going to need.

Curry noticed that for describing the domain of a function, we do not need to have the Cartesian product. If we have a function

$$f : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C} \quad (8.17)$$

we can rewrite it as a function of higher type

$$f : \mathbf{A} \rightarrow (\mathbf{B} \rightarrow \mathbf{C}). \quad (8.18)$$

If you feel like a cool computer scientist, you can also drop the parenthesis and write

$$f : \mathbf{A} \rightarrow \mathbf{B} \rightarrow \mathbf{C}, \quad (8.19)$$

because the expression is not ambiguous, as \rightarrow associates from the left.

This specifies f as a function that, given a \mathbf{A} , provides a function that, given a \mathbf{B} , provides a \mathbf{C} ; this is the same as a function that needs the two arguments \mathbf{A} and \mathbf{B} before giving the \mathbf{C} . To describe the isomorphism we can write it as

$$\mathbf{A} \rightarrow (\mathbf{B} \rightarrow \mathbf{C}) \simeq \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C}, \quad (8.20)$$

or, more precisely, highlighting that these are morphism arrows in \mathbf{Set} , we have

$$\text{Hom}_{\mathbf{Set}}(\mathbf{A}; \text{Hom}_{\mathbf{Set}}(\mathbf{B}; \mathbf{C})) \simeq \text{Hom}_{\mathbf{Set}}(\mathbf{A} \times \mathbf{B}; \mathbf{C}). \quad (8.21)$$



Figure 8.2.: Haskell Curry

Keep this in mind, it will show up later.

Now armed with currying, we can take a second look at (8.13) and realize that we can rewrite

$$\text{Covact} : \mathbf{S} \times \mathbf{A} \rightarrow \mathbf{A} \quad (8.22)$$

as

$$\text{Covact} : \mathbf{S} \rightarrow (\mathbf{A} \rightarrow \mathbf{A}). \quad (8.23)$$

We gave a name to functions of type $\mathbf{A} \rightarrow \mathbf{A}$: these are the endomorphisms of \mathbf{A} , written as $\text{End}(\mathbf{A})$. Thus, we rewrite this as

$$\text{Covact} : \mathbf{S} \rightarrow \text{End}(\mathbf{A}). \quad (8.24)$$

Now we take a second look at (8.14)

$$\text{Covact}(y, \text{Covact}(x, a)) = \text{Covact}(x \circ_{\mathbf{S}} y, a). \quad (8.25)$$

If we rewrite it as an equality of functions, we obtain

$$\text{Covact}(x) \circ_{\text{End}(\mathbf{A})} \text{Covact}(y) =_{\text{End}(\mathbf{A})} \text{Covact}(x \circ_{\mathbf{S}} y). \quad (8.26)$$

Looking at Eqs. (8.24) and (8.26) we recognize that together they indicated that **Covact** is a semigroup morphism (Definition 7.1). This brings us to a more compact description of what is a semigroup action.

Definition 8.9 (Semigroup covariant action). A *semigroup covariant action* of a semigroup \mathbf{S} onto a set \mathbf{A} is a semigroup morphism

$$\text{Covact} : \mathbf{S} \rightarrow \text{End}(\mathbf{A}). \quad (8.27)$$

As it turns out, what could look like a new notion, is actually a special case of a general notion we already encountered.

For completeness, we also define monoid action.

Definition 8.10 (Monoid covariant action). A *monoid covariant action* of a monoid \mathbf{M} onto a set \mathbf{A} is a monoid morphism

$$\text{Covact} : \mathbf{M} \rightarrow \text{End}(\mathbf{A}). \quad (8.28)$$

The neutral element of the monoid $\text{End}(\mathbf{A})$ is the identity function $\text{Id}_{\mathbf{A}}$. Thus, a monoid action must map the neutral element of \mathbf{M} to $\text{Id}_{\mathbf{A}}$.

For defining a group covariant action, we must introduce a slight variation. The endomorphisms $\text{End}(\mathbf{A})$ are not a group, because they also contain non-invertible maps.

We need to reference the *automorphisms* $\text{Aut}(\mathbf{A})$, which is a group.

Definition 8.11 (Group covariant action). A *group covariant action* of a group \mathbf{G} onto a set \mathbf{A} is a group morphism

$$\mathbf{Covact} : \mathbf{G} \rightarrow \mathbf{Aut}(\mathbf{A}). \quad (8.29)$$

A contravariant semigroup action is, by definition, a covariant action of the opposite semigroup.

Definition 8.12 (Semigroup contravariant action). A *semigroup contravariant action* of a semigroup \mathbf{S} onto a set \mathbf{A} is a semigroup morphism

$$\mathbf{Contravact} : \mathbf{S}^{\text{op}} \rightarrow \mathbf{End}(\mathbf{A}). \quad (8.30)$$

Graded exercise 11 (`MatrixMultAction`). Let $V = \mathbb{R}^n$, and let $G = GL_n(\mathbb{R})$ be the group of invertible $n \times n$ matrices. Let

$$\alpha : G \times V \rightarrow V, \langle M, v \rangle \mapsto Mv \quad (8.31)$$

be the usual multiplication of matrices with vectors. Check that (8.31) defines a covariant action of G on V .

9. Solutions to selected exercises

Solution of Exercise 2. Note that Section 4.5 describes composition for ropes of any arbitrary size, while Section 4.5 describes compositions for ropes whose length can be written as $k + a + k$, hence with a minimum size of $2k$. Therefore, the rope 1 in the first theory describes a physical rope of length 1, while the rope 1 in the second theory describes a rope of length $1 + 2k$. They are different theories that happen to have isomorphic rules. Note that the rope $k + k$ acts just like the identity 0. If you connect $k + a + k$ to $k + k$, you obtain $k + a + k$, for all values of a .

Solution of Exercise 3.

Solution of Exercise 4.

Solution of Exercise 5.

Solution of Exercise 6. We have:

$$\begin{aligned} (\mathbf{F} \circledast \mathbf{G})(\mathbf{x} \circledast_{\mathbf{S}} \mathbf{y}) &= \mathbf{G}(\mathbf{F}(\mathbf{x} \circledast_{\mathbf{S}} \mathbf{y})) \\ &= \mathbf{G}(\mathbf{F}(\mathbf{x}) \circledast_{\mathbf{T}} \mathbf{F}(\mathbf{y})) \\ &= \mathbf{G}(\mathbf{F}(\mathbf{x})) \circledast_{\mathbf{U}} \mathbf{G}(\mathbf{F}(\mathbf{y})) \\ &= (\mathbf{F} \circledast \mathbf{G})(\mathbf{x}) \circledast_{\mathbf{U}} (\mathbf{F} \circledast \mathbf{G})(\mathbf{y}). \end{aligned} \tag{9.1}$$

Solution of Exercise 7. We can show that we cannot find an inverse morphism

$$\text{ASCII}^{-1} : \{0, 1\}^* \rightarrow \mathbf{A}^* \tag{9.2}$$

At first sight everything seems in order: if we could find an isomorphism into $[0, 127]^*$ and we can express integers in binary, what could hold us back?

What fails here is something so simple it could go unnoticed: the hypothetical function g is not well defined for all points of its domain. We know how to translate a binary string of length 7, 14, 21, ... back to symbols; but what would be the output of g on the string 111?

The function g is a left inverse for **ASCII**, in the sense that $\text{ASCII} \circ g = \text{id}_{\mathbb{A}^*}$, but it is not a right inverse.

Solution of Exercise 8. The answer is **no** because the encoding is context dependent; I don't know if a single letter is followed by a space or another letter. For example, take the string

$$\text{I } \boxed{\text{A}} \text{ M } \boxed{\text{A}} \text{ X} \quad (9.3)$$

We can decompose it as follows

$$\text{I } \boxed{\text{A}} \circ \text{M} \circ \boxed{\text{A}} \circ \text{M} \circ \text{AX} \quad (9.4)$$

If Morse encoding was a morphism F then we would be able to encode the string as follows:

$$\text{morse}(\text{I } \boxed{\text{A}}) \circ \text{morse}(\text{M}) \circ \text{morse}(\boxed{\text{A}}) \circ \text{morse}(\text{M}) \circ \text{morse}(\text{AX}) \quad (9.5)$$

However, this cannot work, because in the second instance of M we would need to output a letter separator, while in the first case we don't.

Can you find a way to fix it?

For example you can consider the alphabet

$$((\text{A to Z}) \cup (\text{0 to 9})) \times \{\boxed{\text{A}}, \boxed{\text{B}}\} \quad (9.6)$$

where we annotate if each symbol is followed by a letter or by a space.

In this representation, the string can be written as

$$\langle \text{I}, \boxed{\text{A}} \rangle \langle \text{A}, \boxed{\text{B}} \rangle \langle \text{M}, \boxed{\text{A}} \rangle \langle \text{M}, \boxed{\text{B}} \rangle \langle \text{A}, \boxed{\text{B}} \rangle \langle \text{X}, \boxed{\text{A}} \rangle. \quad (9.7)$$

Based on this representation we can define context-independent rules that make a morphism.

Solution of Exercise 9. We start with the first one. Consider $x \in \mathbf{G}$. We have: We know that

$$F(\text{id}_{\mathbf{G}} \circ_{\mathbf{G}} x) = F(x).$$

On the other hand, we know:

$$F(\text{id}_{\mathbf{G}} \circ_{\mathbf{G}} x) = F(\text{id}_{\mathbf{G}}) \circ_{\mathbf{H}} F(x).$$

These two are equivalent if and only if $F(\text{id}_{\mathbf{G}}) = \text{id}_{\mathbf{H}}$.

For the second statement, consider again $x \in \mathbf{G}$. We now that

$$\begin{aligned} F(\text{inv}_{\mathbf{G}}(x) \circ_{\mathbf{G}} x) &= F(\text{id}_{\mathbf{G}}) \\ &= \text{id}_{\mathbf{H}}, \end{aligned}$$

and

$$F(\text{inv}_{\mathbf{G}}(x) \circ_{\mathbf{G}} x) = F(\text{inv}_{\mathbf{G}}(x)) \circ_{\mathbf{H}} F(x).$$

These two are equivalent if and only if $\textcolor{violet}{F}(\text{inv}_{\mathbf{G}}(\textcolor{blue}{x})) = \text{inv}_{\mathbf{H}}(\textcolor{violet}{F}(\textcolor{blue}{x}))$.

Solution of Exercise 10. We have:

$$\begin{aligned}\textcolor{violet}{F}(m+n) &= i^{m+n} \\ &= i^m \cdot i^n \\ &= \textcolor{violet}{F}(m) \cdot \textcolor{violet}{F}(n).\end{aligned}$$

PART B.POINTS AND ARROWS



10. (Semi)categories	97
11. Dependencies	101
12. Transmutation	107
13. Culture	111
14. Connection	117
15. Relation	121
16. Mapping	131
17. Processes	137
18. Graphs	147
19. Solutions to selected exercises	149



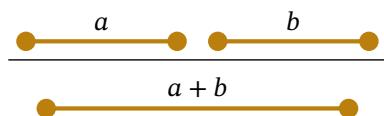
10. (Semi)categories

10.1 Interfaces	97
10.2 Formal definition of (semi) category	98

10.1. Interfaces

One way to understand categories is to see them as generalization of monoids. In semigroups, monoids, groups, we could take any two elements and compose them: the elements always had a “compatible” interface.

To motivate the need for interfaces, consider the ropes of ?? 3, which had this composition rule:



Two chapters later, it is easy to see that we were describing the monoid $\langle \mathbb{R}_{\geq 0}, +, 0 \rangle$. Being a monoid, all pieces of rope are compatible and can be composed.

The first step towards discussing interfaces is to think of things that have a direction. For example, consider extension cords. Let be an extension cord of length c . If you have an extension cord of length c and another of length d ,

you can plug them together to get an extension cord of length $c + d$.

$$\begin{array}{c} \text{---} c \text{---} \text{---} d \text{---} \text{---} \\ \hline \text{---} c + d \text{---} \end{array} \quad (10.1)$$

In this form, this is still the same monoid.

But suppose now that, reading this book, you fall in love with Switzerland and want to visit. As you start to plan your trip, at some point you need to think about electrical adapters.

As you read this book and start to plan to visit Switzerland, at some point you need to buy some adapters. Switzerland uses the connector of type N (Fig. 10.1). If you come from Ireland, your appliances use type G.

Now we when think of extension cords, we might allow either end to have a plug type. These would be Irish and Swiss extension cords of length ℓ :

- A
- B
- C
- D
- E
- F
- G
- H
- I
- L
- M
- N

$$\begin{array}{c} \text{---} \text{---} \text{---} \ell \text{---} \text{---} \text{---} \\ \hline \text{---} \text{---} \text{---} \end{array} \quad (10.2)$$

You might want a cord that has a Swiss male end and an Irish female end:

$$\begin{array}{c} \text{---} \text{---} \text{---} \ell \text{---} \text{---} \text{---} \\ \hline \text{---} \text{---} \text{---} \end{array} \quad (10.3)$$

Unfortunately these devices don't exist. What you can buy are adapters, which we can think of extension cord of length zeros:

$$\begin{array}{c} \text{---} \text{---} \text{---} 0 \text{---} \text{---} \text{---} \\ \hline \text{---} \text{---} \text{---} \end{array} \quad (10.4)$$

If you have an adapter, then you can attach an extension cord to it to obtain (10.3):

$$\begin{array}{c} \text{---} \text{---} \text{---} \ell \text{---} \text{---} \text{---} \text{---} \text{---} 0 \text{---} \text{---} \text{---} \\ \hline \text{---} \text{---} \text{---} \text{---} c \text{---} \text{---} \text{---} \\ \hline \text{---} \text{---} \text{---} \end{array} \quad (10.5)$$

The general formula to compose cords with generic types X, Y, Z is

$$\begin{array}{c} X \text{---} a \text{---} \text{---} Y \text{---} Y \text{---} b \text{---} \text{---} Z \\ \hline X \text{---} a + b \text{---} \text{---} Z \end{array} \quad (10.6)$$

This kind of composition of things that have an input and an output interfaces, like cords, can be modeled by the notions of semicategory and category.

10.2. Formal definition of (semi) category

Let us formally define *semicategories*.

Definition 10.1 (Semicategory). A *semicategory* \mathbf{C} is specified by:
Constituents

Figure 10.1.: Plug/socket types used in the world

1. Objects: A collection[†] $\text{Ob}_{\mathbf{C}}$ whose elements are called *objects*.
2. Morphisms: For every pair of objects X, Y in $\text{Ob}_{\mathbf{C}}$, there is a set $\text{Hom}_{\mathbf{C}}(X; Y)$, elements of which are called *morphisms*. We write

$$f : X \rightarrow_{\mathbf{C}} Y \quad (10.7)$$

to indicate

$$f \in \text{Hom}_{\mathbf{C}}(X; Y). \quad (10.8)$$

3. Composition operations: For every three objects X, Y, Z in $\text{Ob}_{\mathbf{C}}$ there is a composition map

$$\circ_{X,Y,Z} : \text{Hom}_{\mathbf{C}}(X; Y) \times \text{Hom}_{\mathbf{C}}(Y; Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Z) \quad (10.9)$$

We omit mentioning the three objects and just write

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z}{f \circ g : X \rightarrow Z} \quad (10.10)$$

The morphism $f \circ g$ is called the *composition* of f and g .

Conditions

1. Associativity: it holds that

$$\frac{f : X \rightarrow Y \quad g : Y \rightarrow Z \quad h : Z \rightarrow W}{(f \circ g) \circ h = f \circ (g \circ h)} \quad (10.11)$$

The following is the formal definition of a *category*.

Definition 10.2 (Category). A *category* \mathbf{C} is a semicategory with an additional constituent and rule:

Constituents

1. Identity morphisms: for each object X , there is a morphism $\text{Id}_X : X \rightarrow X$ called the *identity morphism* of X .

Conditions

1. Unitality: It holds that:

$$\frac{f : X \rightarrow Y}{\text{Id}_X \circ f = f = f \circ \text{Id}_Y} \quad (10.12)$$

Remark 10.3. 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

[†] A “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.

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.

Graded exercise 1 (CategorySemigroups). There is a category where the objects are semigroups and the morphisms are semigroup homomorphisms. Spell out explicitly what this category is: check in detail each of the points of Definition 10.2.



11. Dependencies

11.1 Design 101

11.1. Design

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. 11.1).

Figure 11.1.: Resources in the electric car example.

• • •
translation **rotation** **electricity**

Now, we can draw arrows between two points if there is a transmuter between them.

We choose the direction of the arrow such that

$$\text{transmuter} : \mathbf{X} \xrightarrow{\quad} \mathbf{Y} \quad (11.1)$$

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

Remark 11.1 (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 (11.1) would be as follows:

$$\text{transmuter} : \mathbf{X} \rightarrow \mathbf{Y}. \quad (11.2)$$

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:

$$\begin{aligned} \text{motor} &: \mathbf{rotation} \rightarrow \mathbf{electricity}, \\ \text{wheels} &: \mathbf{translation} \rightarrow \mathbf{rotation}. \end{aligned} \quad (11.3)$$

We can put these arrows in the diagrams, and obtain the following (Fig. 11.2).

• **wheels** • **motor** •
translation **rotation** **electricity**

Figure 11.2.: 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. 11.3).

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

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

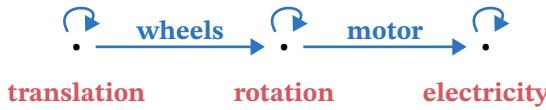


Figure 11.3.: System components and identities.

that is, using a b we can get a Y from a Z , 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. 11.4).

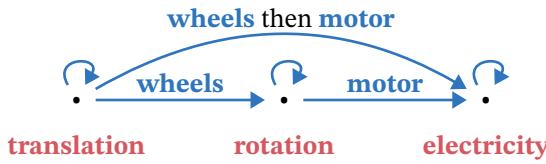


Figure 11.4.: Composition of system components.

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

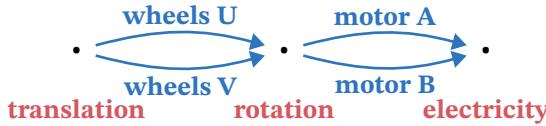


Figure 11.5.: Multiple models for wheels and motors.

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.

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 10.2, they always exist.
- ▷ Given arrows $X \rightarrow Y$ and $Y \rightarrow Z$, we do not need to draw their composition because, by Definition 10.2, 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. 11.6).

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

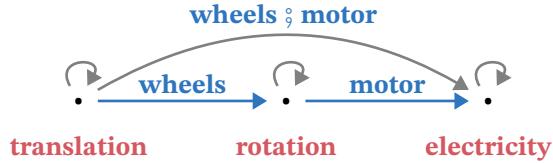


Figure 11.6.: Electric car example. The gray arrows are implied by the properties of a category.

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. 11.7, Fig. 11.8).

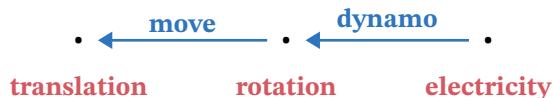


Figure 11.7.: Electric power can be produced from motion.

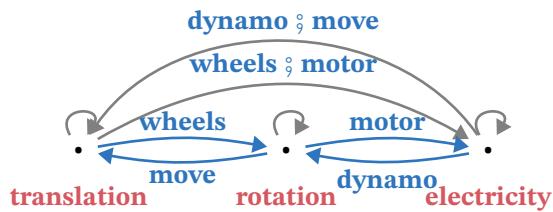


Figure 11.8.: Electric car example: forward and backward transformations.

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

$$\text{wheels ; motor ; dynamo ; move},$$

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. 11.9). 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.

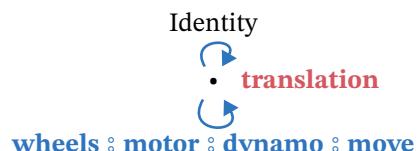


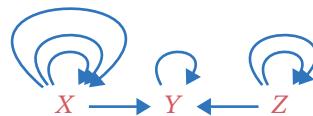
Figure 11.9.: There can be multiple morphisms from an object to itself.

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



Figure 11.10.: An example of a process which is not invertible.

Example 11.2. 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. 11.11 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 \circ to be the composition \circ in \mathbf{C} , and the neutral element to be the identity Id_X . The associativity and unitality laws of the category \mathbf{C} coincide with the ones of the monoid's definition, and are satisfied. Therefore, we can identify a monoid as a one-object category.





12. Transmutation

12.1 Currency categories 107

12.1. Currency categories

In this section, we introduce a kind of category for describing currency exchangers. Our idea is to model currencies as objects of a category, and morphisms will describe ways of exchanging between those currencies. As an example, currency exchangers offer this service.

We start with a set \mathbf{C} of labels for all the currencies we wish to consider:

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

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 $\mathbb{R} \times \{\text{USD}\}$ 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

Fondue is a Swiss dish consisting of melted cheese, served in a *caquelon* (communal pot) over a *réchaud* (portable stove). It is best enjoyed with boiled potatoes, bread, and pickled vegetables.

exchanger) as:

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

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

Note that 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 X, Y, Z , and given currency exchangers

$$E_{a,b} : \mathbb{R} \times \{X\} \rightarrow \mathbb{R} \times \{Y\} \text{ and } E_{a',b'} : \mathbb{R} \times \{Y\} \rightarrow \mathbb{R} \times \{Z\},$$

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

$$E_{aa',a'b+b'} : \mathbb{R} \times \{X\} \rightarrow \mathbb{R} \times \{Z\}. \quad (12.1)$$

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”.

We now want to check that the composition of currency exchangers as defined above obeys unitality and associativity. Given $E_{a,b} : \mathbb{R} \times \{c_1\} \rightarrow \mathbb{R} \times \{c_2\}$ we have:

$$\begin{aligned} E_{1,0} \circ E_{a,b} &= E_{1 \cdot a, a \cdot 0 + b} \\ &= E_{a,b}, \end{aligned}$$

and

$$\begin{aligned} E_{a,b} \circ E_{1,0} &= E_{a \cdot 1, 1 \cdot b + 0} \\ &= E_{a,b}, \end{aligned}$$

This is unitality. Furthermore, given $E_{a',b'} : \mathbb{R} \times \{c_2\} \rightarrow \mathbb{R} \times \{c_3\}$, and $E_{a'',b''} : \mathbb{R} \times \{c_3\} \rightarrow \mathbb{R} \times \{c_4\}$ we have:

$$\begin{aligned} (E_{a,b} \circ E_{a',b'}) \circ E_{a'',b''} &= E_{aa',a'b+b'} \circ E_{a'',b''} \\ &= E_{aa'a'',a''(a'b+b')+b''} \\ &= E_{a,b} \circ E_{a'a'',a''b'+b''} \\ &= E_{a,b} \circ (E_{a',b'} \circ E_{a'',b''}). \end{aligned}$$

This is associativity. Thus we indeed have a category!

Remark 12.1. 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 (12.1) 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 (12.1) may then, in this

notation, be written as

$$\langle a, b \rangle \circ \langle a', b' \rangle = \langle a'a, a'b + b' \rangle. \quad (12.2)$$

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 12.2 (Category Curr). The *category of currencies* **Curr** is specified by:

1. *Objects*: a collection of currencies.
2. *Morphisms*: given two currencies c_1, c_2 , morphisms between them are currency exchangers $\langle a, b \rangle$ from c_1 to c_2 .
3. *Identity morphism*: given a currency 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 (12.2).

As an illustration, consider three currency exchange companies ExchATM, MoneyLah, and Frankurrencies, which operate on several currencies (Table 12.1).

Table 12.1.: Three currency exchange companies operating different currencies.

Company name	Exchanger label	Direction	Exchange rate a	Fixed commission b
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

We can represent this information as a graph, where the nodes are the currencies and the edges are particular exchange operations (Fig. 12.1).

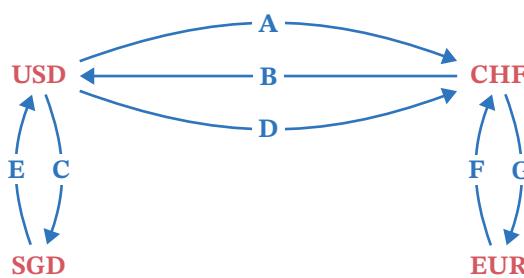


Figure 12.1.: Three currency exchange companies operating different currencies as a graph.

There is a currency category built from the information in Table 12.1 and the graph in Fig. 12.1. 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 12.1,
- ▷ all possible compositions of the currency exchangers listed in Table 12.1.

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. 12.1, and obtaining from it an associated category, called the *free category on G* .

Exercise 11. [Temperatures] Define a category of temperature converters, where the objects are **Celsius**, **Kelvin**, **Fahrenheit**, and the morphisms are the rules to transform a measurement from one unit to another.

Prove that this forms a category.

See solution on page 149.



13. Culture

13.1. 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

13.1 Definitional impetus vs. computational goals	111
13.2 Things that don't matter	112
Typographical conventions don't matter	113

Schwingen is the traditional Swiss wrestling native to the pre-alpine parts of German-speaking Switzerland. Wrestlers wear *Schwingerhosen* that can be used to hold and grapple the adversary.

things, the engineer is typically interested in representing concrete things, and solve some computational problem on the represented structure.

For example, in the case of the currency exchangers, the problem might be that of finding the sequence of the best conversions between a source and a target currency.

First, the engineer would add more constraints to the definition to work with more well-behaved objects. For example, it is reasonable to limit the universe of morphisms in such a way that the action of converting back and forth the same currency to have a cost (through the commission) higher than 0.

In that case, we will find that the optimal paths of currencies never pass through a currency more than once. To see this, consider three currencies **A**, **B**, **C**, a currency exchanger $\langle a, b \rangle$ from **A** to **B**, a currency exchanger $\langle c, d \rangle$ from **B** to **C**, and a currency exchanger $\langle e, f \rangle$ from **C** to **A**. The composition of the currency exchangers reads:

$$\langle eca, ecb + ed + f \rangle = \langle g, h \rangle.$$

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

13.2. 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*.[†] 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.

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 would 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 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. 13.1).

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, reverting the direction (Fig. 13.2). 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

[†] IEEE Spectrum

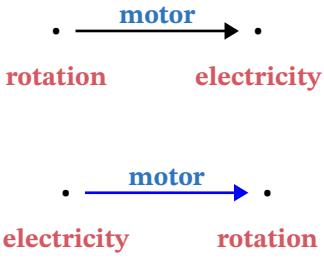


Figure 13.1.: Opposite convention for arrows direction.

called *opposite category* that formalizes this idea of reversing the direction of the arrows.

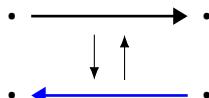


Figure 13.2.

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 book[‡].

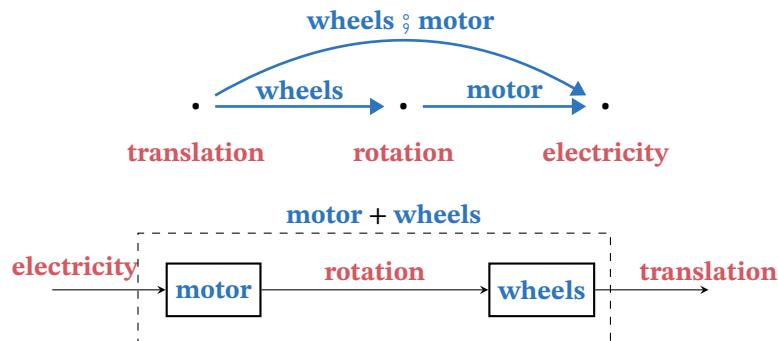


Figure 13.3.: Isomorphisms of resource diagrams.

Fig. 13.3 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

[‡] Note 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.

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 (Fig. 13.4):

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

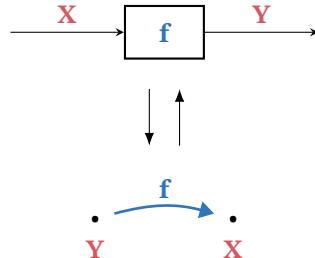


Figure 13.4.: Switching conventions in resource diagrams.

Another example is the following. Consider the bottom representation in Fig. 13.5 and recall that it implies the existence of four composed arrows (wheels and motor pairings): **wheels U ; motor A**, **wheels U ; motor B**, **wheels V ; motor A**, and **wheels V ; motor B**, all going from **translation** to **electricity**. The four possibilities can be represented via boxes and wires, as listed in the upper part of Fig. 13.5.

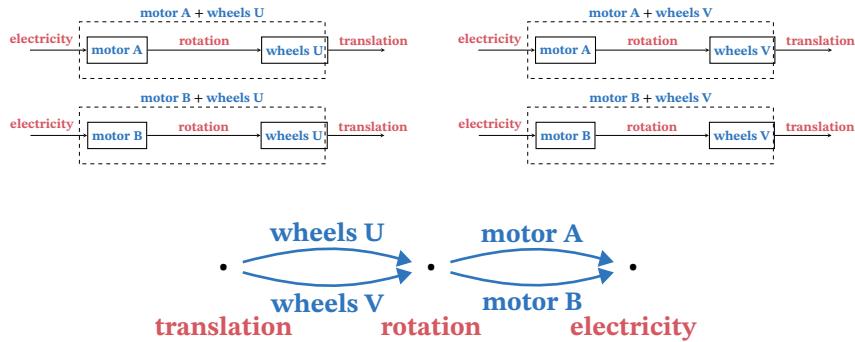


Figure 13.5.: Multiple models for wheels and motors.



14. 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.

14.1 Mobility	117
14.2 Trekking in the Swiss Mountains	119

14.1. Mobility

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

$$\mathcal{G}_c = \langle \mathbf{V}_c, \mathbf{A}_c, \text{src}_c, \text{tgt}_c \rangle,$$

where \mathbf{V}_c represents geographical locations which the car can reach and \mathbf{A}_c represents the paths it can take, such as roads. Similarly, we consider a graph $\mathcal{G}_s = \langle \mathbf{V}_s, \mathbf{A}_s, \text{src}_s, \text{tgt}_s \rangle$, representing the subway system of a city, with stations \mathbf{V}_s and subway lines going through paths \mathbf{A}_s , and a graph $\mathcal{G}_b = \langle \mathbf{V}_b, \mathbf{A}_b, \text{src}_b, \text{tgt}_b \rangle$, 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 $\mathcal{G} = (\mathbf{V}, \mathbf{A}, \text{src}, \text{tgt})$ with $\mathbf{V} = \mathbf{V}_c \cup \mathbf{V}_s \cup \mathbf{V}_b$ and $\mathbf{A} = \mathbf{A}_c \cup \mathbf{A}_s \cup \mathbf{A}_b$, we obtain the desired intermodality graph. Graph \mathcal{G} can be seen as

The Polybahn is an autonomous funicular railway that connects the Central square with the terrace by the main building of ETH Zürich. It was opened in 1889 and it is owned by the banking group UBS AG.

a new category, with objects V and morphisms A .

Example 14.1. Consider the **Car** category, describing your road trip through Italy and Switzerland, with

$$\text{V}_c = \{\text{FCO}_c, \text{Florence}, \text{Bologna}, \text{MPX}_c, \text{Gotthard}, \text{ZRH}_c\},$$

and arrows as in Fig. 14.1. The nodes represent typical touristic road-trip checkpoints in Italy and Switzerland and the arrows represent highways connecting them.



Figure 14.1.: The **Car** category.

Furthermore, consider the **Flight** category with $\text{V}_f = \{\text{FCO}_f, \text{LIN}, \text{MPX}_f, \text{ZRH}_f\}$ and arrows as in Fig. 14.2. The nodes represent airports in Italy and Switzerland and the arrows represent connections, offered by specific flight companies.

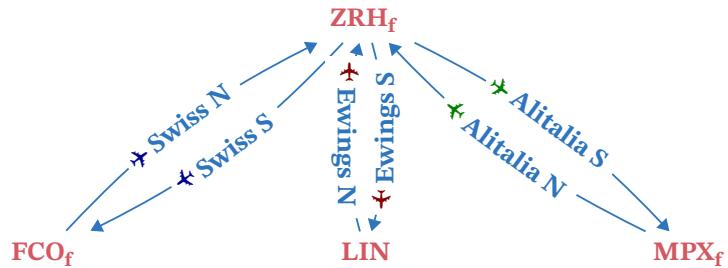


Figure 14.2.: The **Flight** category.

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

$$\text{V}_b = \{\text{FCO}_f, \text{FCO}_c, \text{MPX}_f, \text{MPX}_c, \text{ZRH}_f, \text{ZRH}_c\}$$

and arrows as in Fig. 14.3. 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.

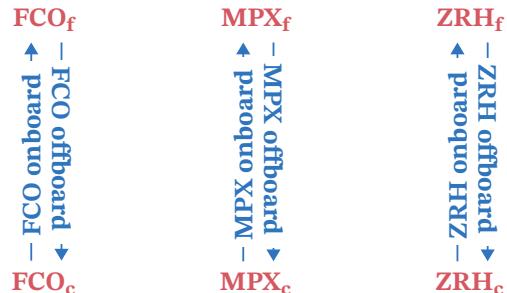


Figure 14.3.: The **Board** category.

The combination of the three, which we call the *intermodal graph*, can be represented as a graph, in which we use dashed arrows for intermodal morphisms,

arising from composition of morphisms involving multiple modes (Fig. 14.4). Imagine that you are in the parking lot of **ZRH** airport and you want to reach **Florence**. From there, you can onboard to a **Swiss** flight to **FCO_f**, will then offboard reaching the parking lot **FCO_c**, and drive on highway **A1** reaching **Florence**. This is intermodality.

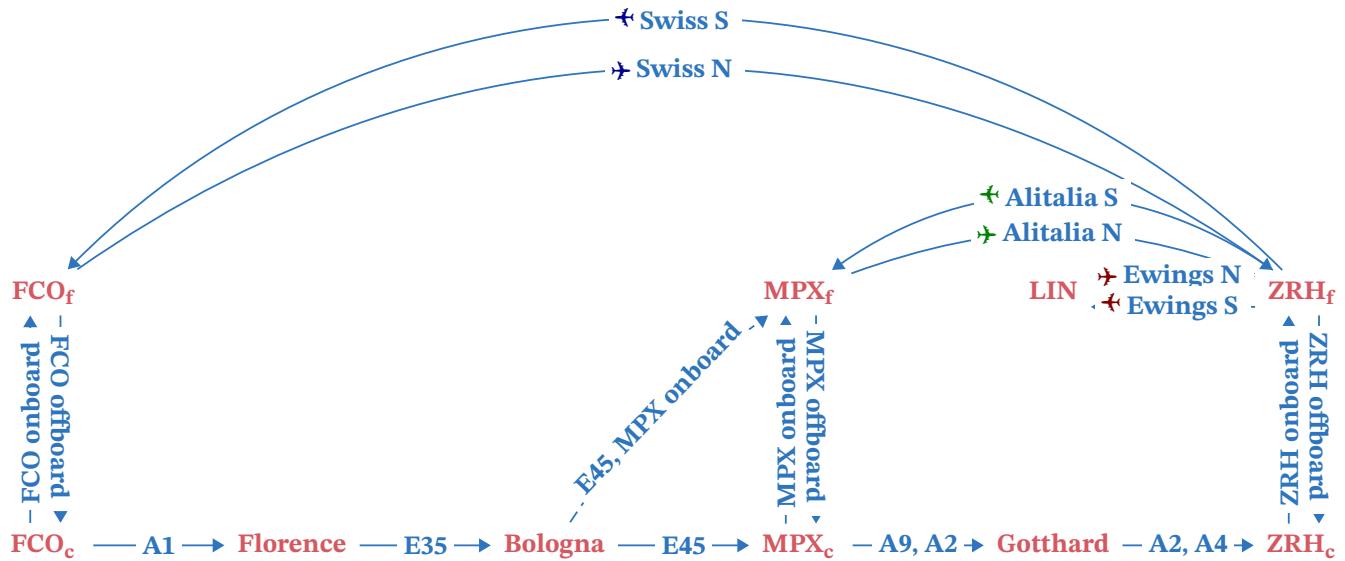


Figure 14.4.: Intermodal graph. The dashed arrows represent intermodal morphisms, and we depict just one of them for simplicity.

The intermodal network category **Intermodal** is the free category on the graph illustrated in Fig. 14.4.

14.2. 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 (x, y, z) . We assume the z -th coordinate is given by an “elevation function”, $z = h(x, y)$, and that h is C^1 (a continuously differentiable function). 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.

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 14.2 (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 $\text{Id}_{\langle p, v \rangle} = \langle \gamma, 0 \rangle$ formally: its path γ is defined on the closed interval $[0, 0]$, (with $T = 0$ and $\gamma(0) = p$). We declare this path to be C^1 by convention, and declare its derivative at 0 to be v .
- ▷ 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

$$\gamma(t) = \begin{cases} \gamma_1(t) & 0 \leq t \leq T_1 \\ \gamma_2(t - T_1) & T_1 \leq t \leq T_1 + T_2. \end{cases} \quad (14.1)$$

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 $\text{Hom}_{\text{Berg}}(\langle p_1, v_1 \rangle; \langle p_2, v_2 \rangle)$, a function

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

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.



15. Relation

15.1. 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, such as homes and businesses. The situation is depicted in Fig. 15.1.

To model the connectivity between the components of the power grid, we now draw arrows between components that are connected. We set the direction of the arrows to flow from energy production, via transmission components, to energy consumption, as depicted in Fig. 15.2.

A possible question one asks about such a power distribution network is: which consumers are serviced by which power sources? For example, power sources such as a solar power plant may fluctuate due to weather conditions, while other power sources, such as a nuclear power plant, may shut down every once in a while due to maintenance work. To see which consumers are connected to which power plants, we can follow “connectivity paths” traced by sequences of arrows, as in Fig. 15.3. There, two possible connectivity paths are depicted (in red and

15.1 Distribution networks	121
15.2 Relations	123
15.3 Relations and functions	125
15.4 Properties of relations	126
15.5 Endorelations	127
15.6 Relational Databases	128
15.7 Exercises	130

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 15.1.: Components of electrical power grids.

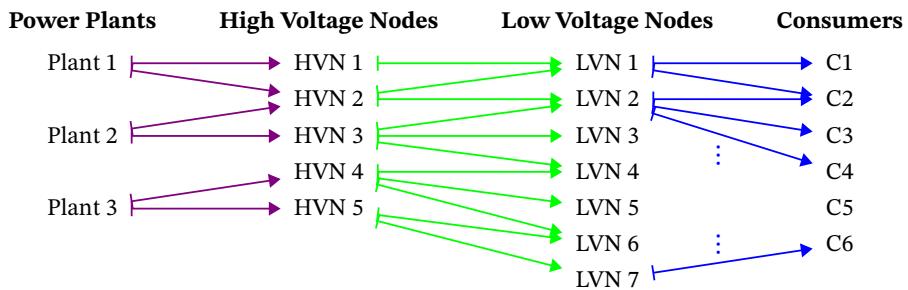


Figure 15.2.: Connectivity between components in electric power grids.

orange, respectively).

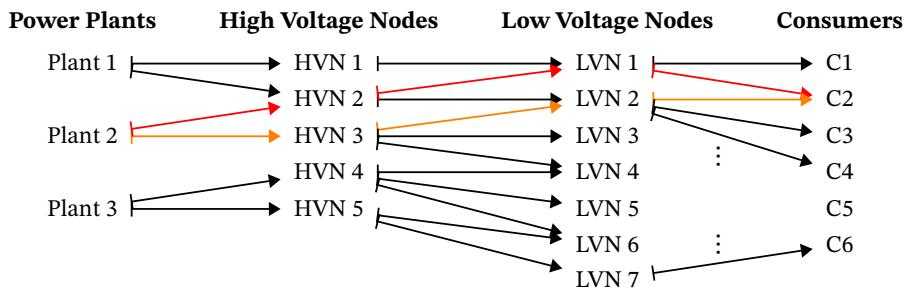


Figure 15.3.: Connection between consumers and power plants.

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. 15.2, we can also include, for example, information about the connectivity of high voltage nodes among themselves, as in Fig. 15.4.

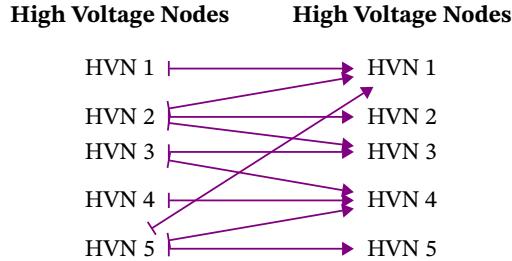


Figure 15.4.: Connectivity between high voltage nodes.

The information encoded in Fig. 15.4 and Fig. 15.2 can also be displayed as a single graph, see Figs. 15.4 and 15.5.

If we ignore the directionality of the arrows, this is analogous to a depiction of type shown in Fig. 15.6, which is a schema of a power grid [8][†].

15.2. Relations

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

Definition 15.1 (Binary relation). A *binary relation* from a set **A** to a set **B** is a subset of the Cartesian product $\mathbf{A} \times \mathbf{B}$.

Remark 15.2. We will often drop the word “binary” and simply use the name “relation”.

Example 15.3. Let $\mathbf{A} = \{x_1, x_2, x_3\}$ and $\mathbf{B} = \{y_1, y_2, y_3, y_4\}$. An example of a relation is the subset

$$\mathcal{R} = \{\langle x_1, y_1 \rangle, \langle x_2, y_3 \rangle, \langle x_1, y_4 \rangle\} \subseteq \mathbf{A} \times \mathbf{B}. \quad (15.1)$$

If **A** and **B** are finite sets, we can depict a relation $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{B}$ graphically as in Fig. 15.7. For each element $\langle x, y \rangle \in \mathbf{A} \times \mathbf{B}$, we draw an arrow from **x** to **y** if and only if $\langle x, y \rangle \in \mathcal{R} \subseteq \mathbf{A} \times \mathbf{B}$.

We can also depict this relation graphically as a subset of $\mathbf{A} \times \mathbf{B}$ in a “coordinate system way”, as in Fig. 15.8.

The shaded grey area is the subset \mathcal{R} defining the relation.

The visualization in Fig. 15.7 hints at the fact that we can think of a relation $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{B}$ as a *morphism* from **A** to **B**.

Definition 15.4 (Category **Rel**). The category of relations **Rel** is given by:

1. *Objects:* The objects of this category are all sets.

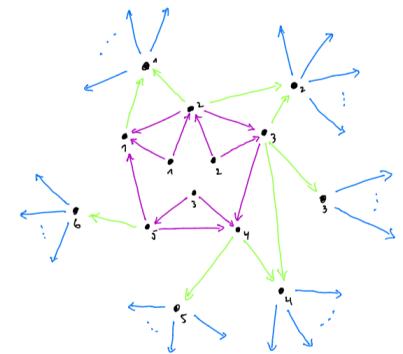


Figure 15.5.: Alternative visualization for connectivity.

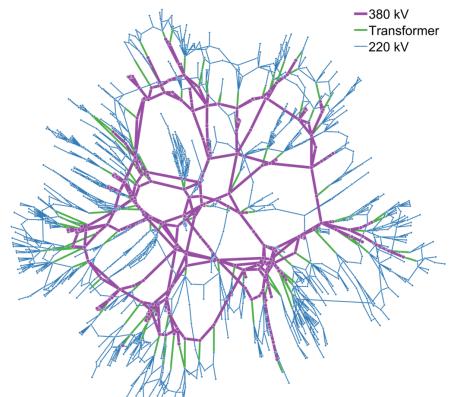


Figure 15.6.: A schematic view of a power grid.

[†] See https://en.wikipedia.org/wiki/Electrical_grid

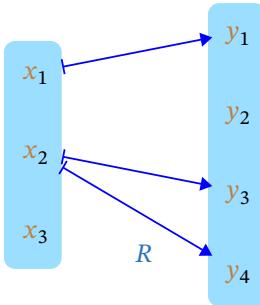


Figure 15.7.

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

$$\text{Id}_X := \{\langle x, x \rangle \mid x \in X\}. \quad (15.2)$$

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

$$R ; S := \{\langle x, z \rangle \mid \exists y \in Y : (\langle x, y \rangle \in R) \wedge (\langle y, z \rangle \in S)\}. \quad (15.3)$$

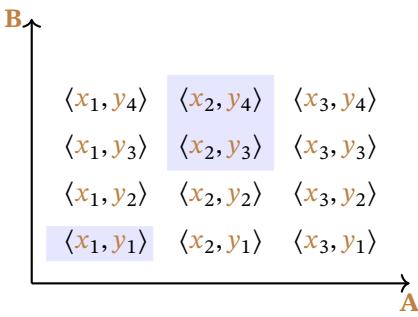
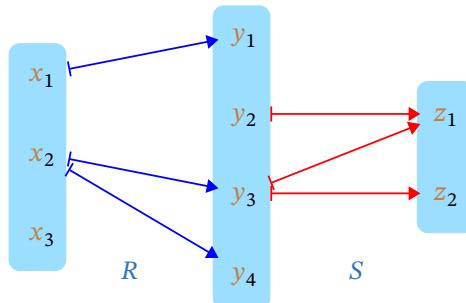


Figure 15.8.: Relations visualized in “coordinate systems”.

Figure 15.9.: Relations compatible for composition.

To illustrate the composition rule in (15.3) for relations, let us consider a simple example, involving sets A, B , and C , and relations $R : A \rightarrow B$ and $S : B \rightarrow C$, as depicted graphically below in Fig. 15.9. Now, according to the rule in (15.3), the



composition $R ; S \subseteq A \times C$ will be such that $\langle x, z \rangle \in R ; S$ if and only if there exists some $y \in B$ 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. 15.9, 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. 15.10, there is an arrow from x_2 to z_1 .

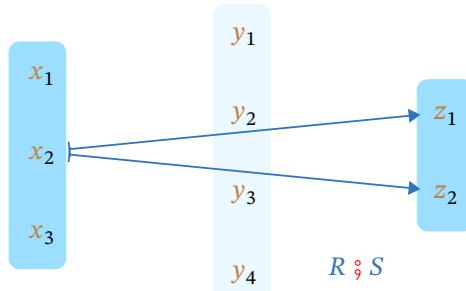


Figure 15.10.: Composition of relations.

Remark 15.5. Relations with the same source and target can be *compared* via inclusion. Given $R \subseteq A \times B$ and $R' \subseteq A \times B$, we can ask whether $R \subseteq R'$ or $R' \subseteq R$.

15.3. Relations and functions

Every function between sets can be thought as a relation. For example, consider the sets $\mathbf{A} = \{x_1, x_2, x_3\}$ and $\mathbf{B} = \{y_1, y_2, y_3, y_4\}$, and the function $f : \mathbf{A} \rightarrow \mathbf{B}$ defined by

$$f(x_1) = y_1, f(x_2) = y_3, f(x_3) = y_4. \quad (15.4)$$

This can be depicted graphically as in Fig. 15.11 and can be written as the following relation

$$\{(x_1, y_1), (x_2, y_3), (x_3, y_4)\} \subseteq \mathbf{A} \times \mathbf{B}. \quad (15.5)$$

This relation (15.5) that we associated to the function f may be thought of as its *graph*. That is, it is the set of tuples in $\mathbf{A} \times \mathbf{B}$ which are a pairing of an element of the source set \mathbf{A} with the element which is its image under f . In Fig. 15.12, the graph of (15.5) is visualized by highlighting in color the elements of the graph.

Not every relation comes from a function in this way, however. Part of the definition of the notion of a “function” is that it maps *every* element of its source set to some element of its target. And a function must map each element of its source to *exactly one* element of its target (“multivalued functions” are not functions in the strict sense). So, for example, we see that the relation in Fig. 15.7 does not arise from a function. Not every element of the source set is related to some element of the target: x_3 is not related to anything. Nor is the relation “single-valued”: the element x_2 is related to two distinct elements of the target set.

Remark 15.6. We can however think about relations *in terms of* functions. Here are three ways:

1. We can think of a relation $R \subseteq \mathbf{A} \times \mathbf{B}$ as a function $\mathbf{A} \times \mathbf{B} \rightarrow \{\perp, \top\}$, where we think of “ \perp ” as “false” and “ \top ” as “true”.

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

$$\phi_R(\langle x, y \rangle) = \begin{cases} \top & \text{if } \langle x, y \rangle \in R \\ \perp & \text{if } \langle x, y \rangle \notin R. \end{cases} \quad (15.6)$$

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

$$R_\phi = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid \phi_R(\langle x, y \rangle) = \top\}. \quad (15.7)$$

These two constructions are inverse to one-another.

2. We can think of a relation $R \subseteq \mathbf{A} \times \mathbf{B}$ as a function $\mathbf{A} \rightarrow \mathcal{P}(\mathbf{B})$.

Given R , we can define a function $\hat{\phi}_R : \mathbf{A} \rightarrow \mathcal{P}(\mathbf{B})$ via

$$\hat{\phi}_R(x) = \{y \in \mathbf{B} \mid \langle x, y \rangle \in R\}. \quad (15.8)$$

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

$$R_{\hat{\phi}} = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid y \in \hat{\phi}_R(x)\}. \quad (15.9)$$

These two constructions are inverse to one another, too.

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

Given R , we can define a function $\check{\phi}_R : \mathbf{B} \rightarrow \mathcal{P}(\mathbf{A})$ via

$$\check{\phi}_R(y) = \{x \in \mathbf{A} \mid \langle x, y \rangle \in R\}. \quad (15.10)$$

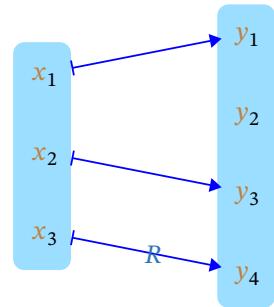


Figure 15.11.: Visualization of the function (15.5).

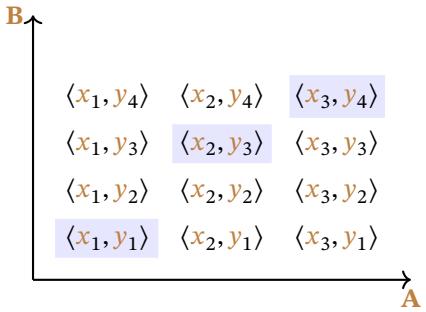


Figure 15.12.: The graph of the function (15.5).

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

$$\mathcal{R}_{\check{\phi}} = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid x \in \check{\phi}_R(y)\}. \quad (15.11)$$

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

Graded exercise 2 (Rel3Functions). For the relation Fig. 15.13, write out the three functions that describe it, respectively, in the three ways outlined in Remark 15.6.

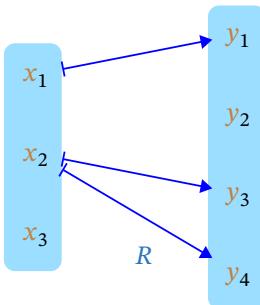


Figure 15.13.

15.4. Properties of relations

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

Definition 15.7 (Properties of a relation). Let $R \subseteq \mathbf{A} \times \mathbf{B}$ be a relation. R is:

1. *Surjective* if for all $y \in \mathbf{B}$ there exists an $x \in \mathbf{A}$ such that $\langle x, y \rangle \in R$;
2. *Injective* if for all $\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \in R$ it holds: $y_1 = y_2 \Rightarrow x_1 = x_2$;
3. *Everywhere-defined* if for all $x \in \mathbf{A}$ there exists an $y \in \mathbf{B}$: $\langle x, y \rangle \in R$;
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 15.8. The relation depicted in Fig. 15.7 is injective but not surjective. It is not single-valued, nor everywhere-defined.

One can notice a certain duality in the properties listed in Def. 15.7. This is made more precise through the following definition.

Definition 15.9 (Transpose of a relation). Let $R \subseteq \mathbf{A} \times \mathbf{B}$ be a relation. The *transpose* (or *opposite*, or *reverse*) of R is the relation given by:

$$R^\top := \{\langle y, x \rangle \in \mathbf{B} \times \mathbf{A} \mid \langle x, y \rangle \in R\}.$$

note that $R^\top : \mathbf{B} \rightarrow \mathbf{A}$, while $R : \mathbf{A} \rightarrow \mathbf{B}$.

Remark 15.10. Some useful properties of a relation $R : \mathbf{A} \rightarrow \mathbf{B}$ and its opposite $R^\top : \mathbf{B} \rightarrow \mathbf{A}$:

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

Graded exercise 3 (RelProperties). Provide a proof of each of the properties listed in Remark 15.10.

Remark 15.11. The aforementioned duality can be seen by “reading the relations (arrows) backwards” (Fig. 15.14).

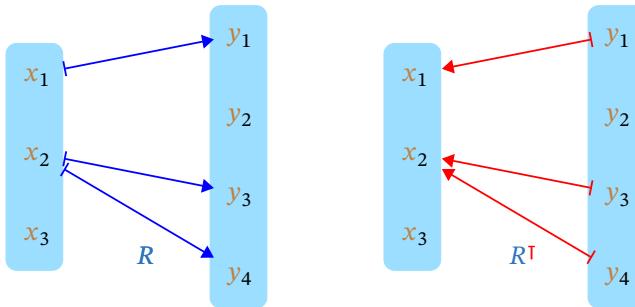


Figure 15.14.

15.5. Endorelations

Definition 15.12 (Endorelation). An *endorelation* on a set \mathbf{A} is a relation $R \subseteq \mathbf{A} \times \mathbf{A}$.

Example 15.13. “Equality” is an endorelation of the form

$$\{(x_1, x_2) \in \mathbf{A} \times \mathbf{A} \mid x_1 = x_2\}.$$

Example 15.14. Take $\mathbf{A} = \mathbb{N}$. The relation “less than or equal” is an endorelation of the form

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

Example 15.15. The relation depicted in Fig. 15.4 is an endorelation between the set of high voltage nodes.

Definition 15.16 (Properties of endorelations). Let $R \subseteq \mathbf{A} \times \mathbf{A}$ be an endorelation. R is:

- ▷ *Symmetric* if for all $x, x' \in \mathbf{A}$ it holds $\langle x, x' \rangle \in R \Leftrightarrow \langle x', x \rangle \in R$;
- ▷ *Reflexive* if for all $x \in \mathbf{A}$ it holds $\langle x, x \rangle \in R$;
- ▷ *Transitive* if for all $\langle x, x' \rangle \in R$ and $\langle x', x'' \rangle \in R$, we have $\langle x, x'' \rangle \in R$.

Example 15.17. 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 15.18. The relation depicted in Fig. 15.4 is reflexive (each node is connected with itself).

Example 15.19. The endorelation reported in Fig. 15.15 is a symmetric relation on $X = \{x_1, x_2\}$.

Definition 15.20 (Equivalence relation). An endorelation $R \subseteq \mathbf{A} \times \mathbf{A}$ is an *equivalence relation* if it is symmetric, reflexive, and transitive. We write $x \sim x'$ if $\langle x, x' \rangle \in R$.

Example 15.21. The relation “equals” on \mathbb{N} is an equivalence relation. The relation “less than or equal” on \mathbb{N} is not.

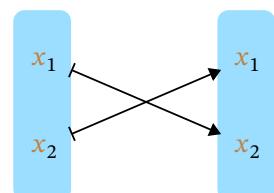


Figure 15.15.: Example of symmetric endorelation.

Example 15.22. 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 15.23. Let $f : \mathbf{A} \rightarrow \mathbf{B}$ be a function between sets. The following defines an equivalence relation:

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

Definition 15.24 (Partition). A *partition* of a set \mathbf{A} is a collection $\{\mathbf{A}_i\}_{i \in I}$ of subsets $\mathbf{A}_i \subseteq \mathbf{A}$ such that

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

Remark 15.25. 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 \mathbf{A} and partitions on \mathbf{A} .

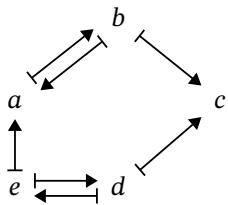


Figure 15.16.

Example 15.26. An example of partitions can be shown through information networks. An exemplary network is depicted in Fig. 15.16. Here, nodes represent data centers, and the arrows represent information flows. We say that data centers x and y are equivalent ($x \sim y$) if and only if there is a path from x to y and a path from y to x . In Fig. 15.16, we have that $a \sim b$, $e \sim d$, and also every center is equivalent with itself.

Graded exercise 4 (CountingEquivalenceRelations). Let $\mathbf{A} = \{1, 2, 3, 4\}$. How many different equivalence relations are there on \mathbf{A} ? Explain how you found your answer.

15.6. 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 [4]?

Suppose we want to buy an electric stepper motor for a robot that we are building, and for this we consult a catalogue of electric stepper motors[‡].

The catalogue might be organized as a large table, where on the left-hand side there is a column listing all available motors (identified with a model ID), and the remaining columns correspond to different attributes that each of the models of motor might have, such as the name of the company that manufactures the

[‡] See pololu.com for a standard catalogue of electric stepper motors.

motor, the size dimensions, the weight, the maximum power, the price, etc.. A simple illustration is provided in Table 15.1.

Table 15.1.: A simplified catalogue of motors.

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

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 15.27 (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](#).

Option 3: Categorical databases A different perspective is that of *categorical databases* [31]. 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**.

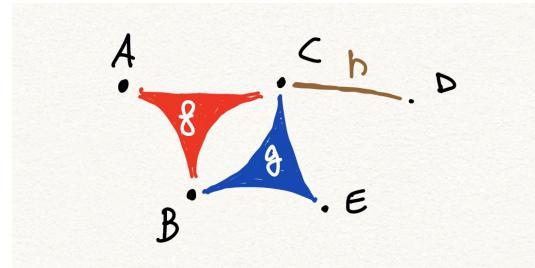


Figure 15.17.: Can you imagine how to define composition with mutant morphisms with more than two legs?

15.7. Exercises

Graded exercise 5 (VisualizeLeqRelation). Let $\mathbf{A} = \mathbf{B} = \{1, 2, 3, 4\}$ and consider the relation $R \subseteq \mathbf{A} \times \mathbf{B}$ defined by

$$R = \{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid x \leq y\}. \quad (15.12)$$

Visualize the relation R via the method in Fig. 15.7 and Fig. 15.8 each.



16. Mapping

16.1. Databases, sets, functions

We continue the discussion of Section 15.6.

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.

16.1 Databases, sets, functions	131
16.2 Functions as relations	133
16.3 The Category Set	134
16.4 Constructing sets	135
16.5 Exercises	135

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

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

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

$$\mathbf{C} := \{\text{Soyo, Sanyo Denki}\}$$

of manufacturing companies, the set \mathbf{S} of possible motor sizes, the set \mathbf{W} of possible weights, the set \mathbf{J} of possible maximal powers, and the set \mathbf{P} of possible prices. Each attribute of a motor may be thought of as a function from the set \mathbf{M} to set of possible values for the given attribute. For example, there is a function $\text{Company} : \mathbf{M} \rightarrow \mathbf{C}$ which maps each model to the corresponding company that manufactures it. So, according to Table 15.1, we have mappings of the kind $\text{Company}(1204) = \text{Soyo}$, and $\text{Company}(2279) = \text{Sanyo Denki}$, etc.

Note that in “real life”, the catalogue of motors might not have seven entries, as in Table 15.1, but has in fact hundreds of entries, and is implemented digitally as a database (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 15.1, suppose, for instance, that we want to consider only motors from Company Sanyo Denki. In terms of the function

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

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

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

this means we wish to restrict ourselves to the preimage

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

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 \mathbf{V} of possible volumes (let’s take $\mathbf{V} = \mathbb{R}_{\geq 0}$, the non-negative real numbers), and define a function

$$\text{Multiply} : \mathbf{S} \rightarrow \mathbf{V}$$

$$\langle l, w, h \rangle \mapsto l \cdot w \cdot h,$$

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

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

to obtain a function

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

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. 16.1).

We can interpret arrows in this diagram as being part of a category, where \mathbf{M} , \mathbf{S} , and \mathbf{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 27.2). 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.

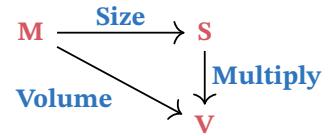


Figure 16.1.: A diagram of functions.

16.2. Functions as relations

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

1. $\forall x \in \mathbf{A} \quad \exists y \in \mathbf{B} : \langle x, y \rangle \in R$
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's start with a relation $R \subseteq \mathbf{A} \times \mathbf{B}$ satisfying the conditions of Def. 16.1. We will build from it a function $f_R : \mathbf{A} \rightarrow \mathbf{B}$. Choose an arbitrary $x \in \mathbf{A}$. According to point 1. in Def. 16.1, there exists a $y \in \mathbf{B}$ 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 $x \in \mathbf{A}$, what if we choose y differently each time we apply the recipe? Point 2. guarantees that this can't happen: it says that the element $f_R(x)$ that we associate to a given $x \in \mathbf{A}$ is in fact uniquely determined by that x . Put another way, the condition 2. says: if $f_R(x_1) \neq f_R(x_2)$, then $x_1 \neq x_2$.

Given a function $f : \mathbf{A} \rightarrow \mathbf{B}$, we can turn it into a relation in a simple way: we consider its graph

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

The relation R_f encodes the same information that f encodes – simply in a different form.

In this text, we take Def. 16.1 as our rigorous definition of what a function is.

Nevertheless, we'll often use functions "in the usual way" and we 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 16.2. Let $R \subseteq A \times B$ and $S \subseteq B \times C$ be relations which are functions. Then their composition $R ; S \subseteq A \times C$ is again a function.

Proof. We check that $R ; S$ satisfies the two conditions stated in Def. 16.1.

1. Choose an arbitrary $x \in A$. We need to show that there exists $z \in C$ such that $\langle x, z \rangle \in R ; S$. Since R is a function, there exists $y \in B$ such that $\langle x, y \rangle \in R$. Choose such a $y \in B$. Then, because S is a function, there exists $z \in C$ 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 $y_1, y_2 \in B$ 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 16.3. Can we have a function (or relation) whose source is the empty set \emptyset ? Given any set B , such a relation would be of the form $R \subseteq \emptyset \times B := \emptyset$. This implies that $R = \emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $\emptyset \rightarrow B$:

- ▷ For all $x \in \emptyset = \emptyset$, there exists a $y \in B$ 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 16.4. Can we have a function (or relation) whose target is the empty set \emptyset ? Again, given any set A , such a relation would be of the form $R \subseteq A \times \emptyset := \emptyset$. This, again, implies $R = \emptyset$. We now need to check whether $R = \emptyset$ corresponds to a function $A \rightarrow \emptyset$:

- ▷ For all $x \in A$, there exists a $y \in \emptyset = \emptyset$ such that $\langle x, y \rangle \in R$? Unless $A = \emptyset$, this is not satisfied.

Therefore, given $A \neq \emptyset$, there is no function (or relation) $A \rightarrow \emptyset$.

16.3. 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 16.5 (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**.

16.4. Constructing sets

16.5. Exercises



17. Processes

17.1. Moore machines

We now look at processes, especially dynamical systems, and see them as categories that act on sequences.

We have already seen linear discrete time systems. We can generalize them by allowing non-linear functions, so to have the definition as follows. We call these **Moore** machines, and describe them as:

$$\begin{cases} \text{dyn} : \mathbf{U} \times \mathbf{X} \rightarrow \mathbf{X} \\ \text{ro} : \mathbf{X} \rightarrow \mathbf{Y}, \end{cases} \quad (17.1)$$

where \mathbf{U} represents inputs, \mathbf{X} states, \mathbf{Y} outputs, dyn the dynamics, and ro the readout. As introduced in Section 8.2, we can apply currying to dyn , to obtain a map from inputs to endomorphisms on the states:

$$\begin{cases} \text{dyn} : \mathbf{U} \rightarrow \text{End}(\mathbf{X}) \\ \text{ro} : \mathbf{X} \rightarrow \mathbf{Y} \end{cases} \quad (17.2)$$

Exercise 12. fake one

17.1 Moore machines	137
17.2 Action on sequences	140
17.3 Other machines	141
17.4 Action of a category	144
17.5 Procedures	145
17.6 Propositions	145

See solution on page 149.

We also need to have an start $\in \mathbf{X}$ to act as the initial state.

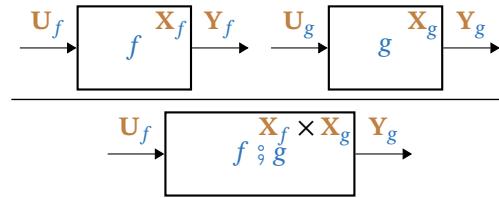
Suppose we have two morphisms

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{start}_f \rangle \quad (17.3)$$

and

$$g = \langle \mathbf{U}_g, \mathbf{X}_g, \mathbf{Y}_g, \text{dyn}_g, \text{ro}_g, \text{start}_g \rangle, \quad (17.4)$$

such that $\mathbf{Y}_f \subseteq \mathbf{U}_g$. The composition of these two systems should have a joint state that is the product of the states.



Here is one way to do it. We specify the spaces:

$$\begin{aligned} \mathbf{U}_{f \circ g} &= \mathbf{U}_f \\ \mathbf{X}_{f \circ g} &= \mathbf{X}_f \times \mathbf{X}_g \\ \text{start}_{f \circ g} &= \langle \text{start}_f, \text{start}_g \rangle \\ \mathbf{Y}_{f \circ g} &= \mathbf{Y}_g \end{aligned} \quad (17.5)$$

Furthermore, we specify the dynamics

$$\begin{aligned} \text{dyn}_{f \circ g} : \mathbf{U}_f \times (\mathbf{X}_f \times \mathbf{X}_g) &\longrightarrow (\mathbf{X}_f \times \mathbf{X}_g) \\ \langle u, \langle x_f, x_g \rangle \rangle &\mapsto \langle \text{dyn}_f(u, x_f), \text{dyn}_g(\text{ro}_f(x_f), x_g) \rangle \end{aligned} \quad (17.6)$$

and the “readout”:

$$\begin{aligned} \text{ro}_{f \circ g} : (\mathbf{X}_f \times \mathbf{X}_g) &\longrightarrow \mathbf{Y}_g \\ \langle x_f, x_g \rangle &\mapsto \text{ro}_g(x_g) \end{aligned} \quad (17.7)$$

We represent the composition graphically as in Fig. 17.1.

However, if we define these using the Cartesian product $\mathbf{X}_f \times \mathbf{X}_g$, we cannot compose the systems in an associative way. When we have three systems, composing in the two ways would bring to $(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h$ and $\mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h)$, which are *isomorphic* sets but not equal. Elements of these sets are of the form $\langle \langle a, b \rangle, c \rangle$ and $\langle a, \langle b, c \rangle \rangle$. You can clearly spot the isomorphism:

$$(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h \neq \mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h) \quad (17.8)$$

$$(\mathbf{X}_f \times \mathbf{X}_g) \times \mathbf{X}_h \simeq \mathbf{X}_f \times (\mathbf{X}_g \times \mathbf{X}_h) \quad (17.9)$$

We can avoid lengthy book-keeping by using a slightly different construction.

Let's consider the monoid of sequences of sets:

$$\mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h \in \mathbf{Set}^*, \quad (17.10)$$

for which we have, for sets

$$(\mathbf{X}_f \circ \mathbf{X}_g) \circ \mathbf{X}_h = \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h = \mathbf{X}_f \circ (\mathbf{X}_g \circ \mathbf{X}_h), \quad (17.11)$$

and for elements

$$[a] = a \quad (17.12)$$

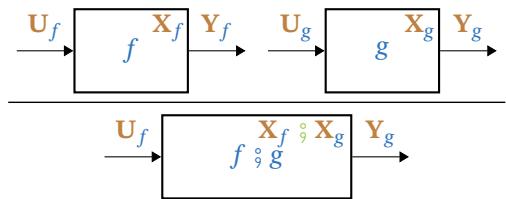
$$[a; b]; c = [a; b; c] \quad (17.13)$$

$$\frac{a : A \quad b : B}{[a; b] : (A \circ B)} \quad (17.14)$$

$$\frac{a : A \quad [a; b] : (A \circ B)}{b : B} \quad (17.15)$$

and

$$\frac{f : A \rightarrow C \quad g : B \rightarrow C}{[f; g] : (A \circ B) \rightarrow C}. \quad (17.16)$$



Here is a different way to do composition (Fig. 17.2). First, we write spaces as:

$$\begin{aligned} \mathbf{U}_{f \circ g} &= \mathbf{U}_f \\ \mathbf{X}_{f \circ g} &= \mathbf{X}_f \circ \mathbf{X}_g \\ \text{start}_{f \circ g} &= [\text{start}_f; \text{start}_g] \\ \mathbf{Y}_{f \circ g} &= \mathbf{Y}_g \end{aligned} \quad (17.17)$$

We then write the dynamics

$$\text{dyn}_{f \circ g} : \mathbf{U}_f \times (\mathbf{X}_f \circ \mathbf{X}_g) \longrightarrow (\mathbf{X}_f \circ \mathbf{X}_g) \quad (17.18)$$

$$\langle u, [x_f; x_g] \rangle \mapsto [\text{dyn}_f(u, x_f); \text{dyn}_g(\text{ro}_f(x_f), x_g)]$$

$$\mathbf{X}_f \circ \mathbf{X}_g \quad (17.19)$$

and the readout

$$\begin{aligned} \text{ro}_{f \circ g} : (\mathbf{X}_f \circ \mathbf{X}_g) &\longrightarrow \mathbf{Y}_g \\ [x_f ; x_g] &\longmapsto \text{ro}_g(x_g) \end{aligned} \tag{17.20}$$

With this definition we can define the semi-category **Moo** of Moore machines.

Definition 17.1 (Moo). The *semi-category of Moore machines* **Moo** is given by:

1. *Objects*: sets.
2. *Morphisms*: A morphism is a tuple

$$f = \langle \mathbf{U}_f, \mathbf{X}_f, \mathbf{Y}_f, \text{dyn}_f, \text{ro}_f, \text{start}_f \rangle, \tag{17.21}$$

where:

- ▷ $\mathbf{U}, \mathbf{X}, \mathbf{Y}$ are sets;
- ▷ $\text{dyn} : \mathbf{U} \rightarrow \mathbf{End}(\mathbf{X})$;
- ▷ $\text{ro} : \mathbf{X} \rightarrow \mathbf{Y}$.

3. *Composition of morphisms*: Composition is given by:

$$\begin{aligned} \mathbf{U}_{f \circ g} &= \mathbf{X}_f \\ \mathbf{X}_{f \circ g} &= \mathbf{X}_f \circ \mathbf{X}_g \\ \text{start}_{f \circ g} &= [\text{start}_f ; \text{start}_g] \\ \mathbf{Y}_{f \circ g} &= \mathbf{Y}_g, \end{aligned} \tag{17.22}$$

with

$$\begin{aligned} \text{dyn}_{f \circ g} : \mathbf{U}_f \times (\mathbf{X}_f \circ \mathbf{X}_g) &\longrightarrow (\mathbf{X}_f \circ \mathbf{X}_g) \\ \langle u, [x_f ; x_g] \rangle &\longmapsto [\text{dyn}_f(u, x_f) ; \text{dyn}_g(\text{ro}_f(x_f), x_g)], \end{aligned} \tag{17.23}$$

and

$$\begin{aligned} \text{ro}_{f \circ g} : (\mathbf{X}_f \circ \mathbf{X}_g) &\longrightarrow \mathbf{Y}_g \\ [x_f ; x_g] &\longmapsto \text{ro}_g(x_g) \end{aligned} \tag{17.24}$$

Exercise 13. Show that indeed **Moo** is a semi-category.

See solution on page 149.

17.2. Action on sequences

Let's now look at how machines like the above acts on sequences.

For now we only have defined semi-group, monoid, and group actions, and have not talked yet about (semi)category actions. Let's consider the set of machines systems with $\mathbf{U} = \mathbf{Y} = \mathbf{A}$, which is the homset $\mathbf{Hom}_{\mathbf{Moo}}(\mathbf{A}; \mathbf{A})$.

Given a finite input sequence $u : \mathbb{N} \rightarrow \mathbf{A}$ of length n , the output is an instanta-

neous transformation of the state:

$$\begin{aligned} y_0 &= \text{ro}(x_0) \\ y_1 &= \text{ro}(x_1) \\ y_2 &= \text{ro}(x_2) \\ &\dots = \dots \\ y_k &= \text{ro}(x_{k-1}) \end{aligned} \tag{17.25}$$

The state is computed recursively as follows:

$$\begin{aligned} x_0 &= \text{dyn}(u_0, \text{start}) \\ x_1 &= \text{dyn}(u_1, x_0) \\ x_2 &= \text{dyn}(u_2, x_1) \\ &\dots = \dots \\ x_k &= \text{dyn}(u_k, x_{k-1}) \end{aligned} \tag{17.26}$$

Therefore, given a machine $f : \mathbf{A} \rightarrow_{\mathbf{Moo}} \mathbf{A}$ we have defined a map from $\mathbb{N} \rightarrow \mathbf{A}$ to itself. Let's call it act . It is defined as a map of the form

$$\text{act}_f : (\mathbb{N} \rightarrow \mathbf{A}) \rightarrow (\mathbb{N} \rightarrow \mathbf{A}), \tag{17.27}$$

or, more formally,

$$\text{act} : \text{Hom}_{\mathbf{Moo}}(\mathbf{A}; \mathbf{A}) \rightarrow \text{End}(\mathbb{N} \rightarrow \mathbf{A}). \tag{17.28}$$

Note that both $\text{Hom}_{\mathbf{Moo}}(\mathbf{A}; \mathbf{A})$ and $\text{End}(\mathbb{N} \rightarrow \mathbf{A})$ are semigroups. Could it be that act is a semigroup morphism? (and consequently, is act a covariant semigroup action)?

Let's check the condition for it being a morphism ((7.2)):

$$\text{act}(f \circ_{\mathbf{Moo}} g) \stackrel{?}{=} \text{act}(f) \circ_{\text{End}(\mathbb{N} \rightarrow \mathbf{A})} \text{act}(g) \tag{17.29}$$

$$\text{act}(g \circ_{\mathbf{Moo}} f) \stackrel{?}{=} \text{act}(g) \circ_{\text{End}(\mathbb{N} \rightarrow \mathbf{A})} \text{act}(f) \tag{17.30}$$

17.3. Other machines

But there are many different types of machines.

A Moore machine outputs 1 element at each time step; what if the machine was able to output more than one or zero output?

The signature of this would be this:

$$\begin{cases} \text{dyn} : \mathbf{U}^* \rightarrow \text{End}(\mathbf{X}) \\ \text{ro} : \mathbf{X} \rightarrow \mathbf{Y}^* \end{cases} \tag{17.31}$$

where the output is not just \mathbf{Y} but \mathbf{Y}^* : the machine can produce zero or more outputs. We call these *More* machines.

Exercise 14. Define the semi-category **Mor** of More machines.

See solution on page ??.

We have these act on A^* ; the timing information is not there.

Another type of machine are the event-based machines, with signature

$$\begin{cases} \text{dyn} : (\mathbb{N} \times \text{U})^* \rightarrow \text{End}(\text{X}) \\ \text{ro} : \text{X} \rightarrow (\mathbb{N} \times \text{Y})^* \end{cases} \quad (17.32)$$

The natural numbers are the deltas between events. (This could be generalized to $\mathbb{R}_{\geq 0}$.)

Continuous-time dynamical systems would be described as

$$\begin{cases} \text{dyn} : \text{U} \rightarrow \text{VF}(\text{X}) \\ \text{ro} : \text{X} \rightarrow \text{Y} \end{cases} \quad (17.33)$$

One should put more conditions on the objects—typically, that they are manifolds, and more constraints about the dynamics for the trajectories to exist.

Table 17.1.: Some types of signals and processes

	Signals		Processes
	one-sided	two-sided	
Moore machines (Moo)	$\mathbb{N} \rightarrow \textcolor{brown}{A}$	$\mathbb{Z} \rightarrow \textcolor{brown}{A}$	$\begin{cases} \text{dyn} : \textcolor{brown}{U} \rightarrow \text{End}(\textcolor{brown}{X}) \\ \text{ro} : \textcolor{brown}{X} \rightarrow \textcolor{brown}{Y} \end{cases}$
More machines (Mor)	$\textcolor{brown}{A}^*$	$\textcolor{brown}{A}^*$	$\begin{cases} \text{dyn} : \textcolor{brown}{U}^* \rightarrow \text{End}(\textcolor{brown}{X}) \\ \text{ro} : \textcolor{brown}{X} \rightarrow \textcolor{brown}{Y}^* \end{cases}$
event-based (EB)	$(\mathbb{N} \times \textcolor{brown}{A})^*$	$(\mathbb{N} \times \textcolor{brown}{A})^*$	$\begin{cases} \text{dyn} : (\mathbb{N} \times \textcolor{brown}{U})^* \rightarrow \text{End}(\textcolor{brown}{X}) \\ \text{ro} : \textcolor{brown}{X} \rightarrow (\mathbb{N} \times \textcolor{brown}{Y})^* \end{cases}$
continuous (DS)	$\mathbb{R}_{\geq 0} \rightarrow \textcolor{brown}{A}$	$\mathbb{R} \rightarrow \textcolor{brown}{A}$	$\begin{cases} \text{dyn} : \textcolor{brown}{U} \rightarrow \text{VF}(\textcolor{brown}{X}) \\ \text{ro} : \textcolor{brown}{X} \rightarrow \textcolor{brown}{Y} \end{cases}$

17.4. Action of a category

Now it is time to generalize from actions of a semigroup to action of a semicategory.

Let \mathbf{C} a general process category, as in Table 17.1.

We want a process in $\text{Hom}_{\mathbf{C}}(\mathbf{X}; \mathbf{Y})$ to induce a map between signals.

Note that each process type has a different signal types.

For example, a Moore machine in $\text{Hom}_{\text{Moo}}(\mathbf{X}; \mathbf{Y})$ will map the set $\mathbb{N} \rightarrow \mathbf{X}$ to the set $\mathbb{N} \rightarrow \mathbf{Y}$. A More machine in $\text{Hom}_{\text{Mor}}(\mathbf{X}; \mathbf{Y})$ will map the set \mathbf{X}^* to the set \mathbf{Y}^* . An event based machine will map the set $\mathbb{N} \times \mathbf{X}^*$ to the set $\mathbb{N} \times \mathbf{Y}^*$.

To look at this generically, we need to consider a map φ that maps the base set \mathbf{X} to the actual set $\varphi(\mathbf{X})$. For example:

$$\begin{aligned}\varphi_{\text{Moo}} : \mathbf{Set} &\longrightarrow \mathbf{Set} \\ \mathbf{X} &\longmapsto (\mathbb{N} \rightarrow \mathbf{X})\end{aligned}\tag{17.34}$$

$$\begin{aligned}\varphi_{\text{Mor}} : \mathbf{Set} &\longrightarrow \mathbf{Set} \\ \mathbf{X} &\longmapsto \mathbf{X}^*\end{aligned}\tag{17.35}$$

$$\begin{aligned}\varphi_{\text{EB}} : \mathbf{Set} &\longrightarrow \mathbf{Set} \\ \mathbf{X} &\longmapsto (\mathbb{N} \times \mathbf{X})^*\end{aligned}\tag{17.36}$$

$$\begin{aligned}\varphi_{\text{DS}} : \mathbf{Set} &\longrightarrow \mathbf{Set} \\ \mathbf{X} &\longmapsto (\mathbb{R}_{\geq 0} \rightarrow \mathbf{X})\end{aligned}\tag{17.37}$$

$$\varphi_{\text{Moo}} : \mathbf{X} \longmapsto (\mathbb{N} \rightarrow \mathbf{X})\tag{17.38}$$

$$\varphi_{\text{Mor}} : \mathbf{X} \longmapsto \mathbf{X}^*\tag{17.39}$$

$$\varphi_{\text{EB}} : \mathbf{X} \longmapsto (\mathbb{N} \times \mathbf{X})^*\tag{17.40}$$

$$\varphi_{\text{DS}} : \mathbf{X} \longmapsto (\mathbb{R}_{\geq 0} \rightarrow \mathbf{X})\tag{17.41}$$

Note for the last one we need to define manifolds.

Then we need another map γ that given a process in $\text{Hom}_{\mathbf{C}}(\mathbf{X}; \mathbf{Y})$ produces a map from $\varphi(\mathbf{X})$ to $\varphi(\mathbf{Y})$.

$$\gamma : \text{Hom}_{\mathbf{C}}(\mathbf{X}; \mathbf{Y}) \rightarrow (\varphi(\mathbf{X}) \rightarrow \varphi(\mathbf{Y}))\tag{17.42}$$

Interpreting the arrows as morphisms in sets, we can say:

$$\gamma : \text{Hom}_{\mathbf{C}}(\mathbf{X}; \mathbf{Y}) \rightarrow \text{Hom}_{\mathbf{Set}}(\varphi(\mathbf{X}); \varphi(\mathbf{Y}))\tag{17.43}$$

Definition 17.2 (Semi-category action). A *semi-category action* of a semi-category \mathbf{C} is defined by

- ▷ a map φ that associates from each object $X \in \text{Ob}_{\mathbf{C}}$, a set $\varphi(X)$:

$$\varphi : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{Set}} \quad (17.44)$$

- ▷ a map γ that associates to each morphism a function:

$$\gamma : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{Set}}(\varphi(X); \varphi(Y)) \quad (17.45)$$

Moreover, this condition must hold:

$$\gamma(f ; g) = \gamma(f) ; \gamma(g). \quad (17.46)$$

17.5. Procedures

17.6. Propositions



18. Graphs

18.1 Graphs	147
18.2 Graph homomorphisms	148

18.1. Graphs

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

Definition 18.1 (Graph). A *graph* $\mathcal{G} = \langle \mathbf{V}, \mathbf{A}, \text{src}, \text{tgt} \rangle$ consists of a set of vertices \mathbf{V} , a set of arrows \mathbf{A} , and two functions $\text{src}, \text{tgt} : \mathbf{A} \rightarrow \mathbf{V}$, called the *source* and *target* functions, respectively. Given $a \in \mathbf{A}$ with $\text{src}(a) = v$ and $\text{tgt}(a) = w$, we say that a is an *arrow* from v to w .

Remark 18.2. 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 18.3 (Path). Let \mathcal{G} be a graph. A *path* in \mathcal{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.

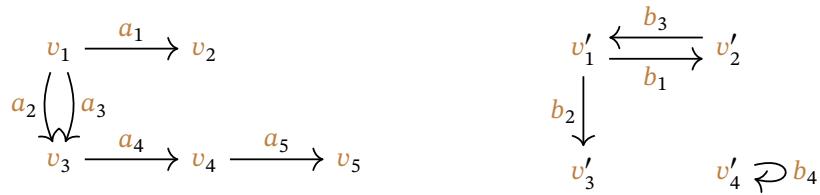
18.2. Graph homomorphisms

Definition 18.4 (Graph homomorphism). Given graphs $\mathcal{G} = \langle \mathbf{V}, \mathbf{A}, \text{src}, \text{tgt} \rangle$ and $\mathcal{G}' = \langle \mathbf{V}', \mathbf{A}', \text{src}', \text{tgt}' \rangle$, a graph homomorphism $f : \mathcal{G} \rightarrow \mathcal{G}'$ is given by maps $f_0 : \mathbf{V} \rightarrow \mathbf{V}'$ and $f_1 : \mathbf{A} \rightarrow \mathbf{A}'$, such that the following diagrams commute:

$$\begin{array}{ccc} \mathbf{A} & \xrightarrow{f_1} & \mathbf{A}' \\ \downarrow \text{src} & & \downarrow \text{src}' \\ \mathbf{V} & \xrightarrow{f_0} & \mathbf{V}' \end{array} \quad \begin{array}{ccc} \mathbf{A} & \xrightarrow{f_1} & \mathbf{A}' \\ \downarrow \text{tgt} & & \downarrow \text{tgt}' \\ \mathbf{V} & \xrightarrow{f_0} & \mathbf{V}' \end{array}$$

Remark 18.5. Intuitively, all this is saying is that “arrows are bound to their vertices”, meaning that if a vertex v_1 is connected to v_2 via an arrow a , the vertices $f_0(v_1)$ and $f_0(v_2)$ have to be connected via an arrow $f_1(a)$.

Example 18.6. Let us consider the two graphs, \mathcal{G} and \mathcal{G}' depicted in Example 18.6.



Example 18.7 (Counterexample).

Exercise 15.

See solution on page ??.

19. Solutions to selected exercises

Solution of Exercise 11.

Solution of Exercise 12.

Solution of Exercise 13. Given valid, composable machines f, g, h , we can specify the spaces and initial conditions:

$$\begin{aligned}\mathbf{U}_{(f \circ g) \circ h} &= \mathbf{U}_{f \circ (g \circ h)} = \mathbf{U}_f, \\ \mathbf{X}_{(f \circ g) \circ h} &= \mathbf{X}_{f \circ (g \circ h)} = \mathbf{X}_f \circ \mathbf{X}_g \circ \mathbf{X}_h \\ \text{start}_{(f \circ g) \circ h} &= \text{start}_{f \circ (g \circ h)} = [\text{start}_f ; \text{start}_g ; \text{start}_h] \\ \mathbf{Y}_{(f \circ g) \circ h} &= \mathbf{Y}_{f \circ (g \circ h)} = \mathbf{Y}_h\end{aligned}$$

Starting from (17.23) one can now check associativity of the dynamics:

$$\begin{aligned}\mathbf{dyn}_{(f \circ g) \circ h} : \mathbf{U}_f \times ((\mathbf{X}_f \circ \mathbf{X}_g) \circ \mathbf{X}_h) &\rightarrow (\mathbf{X}_f \circ \mathbf{X}_g) \circ \mathbf{X}_h \\ \langle u, [x_f ; x_g ; x_h] \rangle &\mapsto [\mathbf{dyn}_{f \circ g}(u, [x_f ; x_g]) ; \mathbf{dyn}_h(\mathbf{ro}_{f \circ g}([x_f ; x_g]), x_h)] \\ &= [\mathbf{dyn}_f(u, x_f) ; \mathbf{dyn}_g(\mathbf{ro}(x_f), x_g) ; \mathbf{dyn}_h(\mathbf{ro}_g(x_g), x_h)].\end{aligned}$$

On the other hand, one has:

$$\begin{aligned}\mathbf{dyn}_{f \circ (g \circ h)} : \mathbf{U}_f \times (\mathbf{X}_f \circ (\mathbf{X}_g \circ \mathbf{X}_h)) &\rightarrow \mathbf{X}_f \circ (\mathbf{X}_g \circ \mathbf{X}_h) \\ \langle u, [x_f ; x_g ; x_h] \rangle &\mapsto [\mathbf{dyn}_f(u, x_f) ; \mathbf{dyn}_{g \circ h}(\mathbf{ro}_f(x_f), [x_g ; x_h])] \\ &= [\mathbf{dyn}_f(u, x_f) ; \mathbf{dyn}_g(\mathbf{ro}(x_f), x_g) ; \mathbf{dyn}_h(\mathbf{ro}_g(x_g), x_h)].\end{aligned}$$

Finally, one can check associativity of the readout:

$$\text{ro}_{(f \circ g) \circ h} : (\mathbf{X}_f \circ \mathbf{X}_g) \circ \mathbf{X}_h \rightarrow \mathbf{Y}_h$$

$$[x_f ; x_g ; x_h] \mapsto \text{ro}_h(x_h),$$

and

$$\text{ro}_{f \circ (g \circ h)} : \mathbf{X}_f \circ (\mathbf{X}_g \circ \mathbf{X}_h) \rightarrow \mathbf{Y}_h$$

$$[x_f ; x_g ; x_h] \mapsto \text{ro}_h(x_h).$$

PART C.ORDER



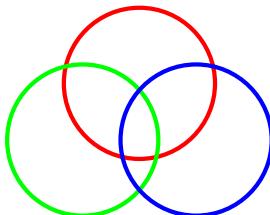
20. Trade-offs	153
21. Poset constructions	173
22. Life is hard	177
23. Solutions to selected exercises	183



20. Trade-offs

20.1. Trade-offs

Trade-offs characterize all engineering disciplines, and can be literally found everywhere. A typical trade-off is the one reported in Fig. 20.1. When designing a product, you want it to be *good*, *fast*, and *cheap*, and typically you can just choose between two of these qualities.



20.1 Trade-offs	153
20.2 The 3 diagrams	154
Trade-offs for the human body .	154
Masks	155
Hats and headphones	156
20.3 Ordered sets	158
20.4 Chains and Antichains	163
20.5 Upper and lower sets	164
20.6 From antichains to uppersets, and viceversa	165
Measuring posets	168
20.7 Lattices	169

Figure 20.1.

20.2. The 3 diagrams

In this sections, we introduce concepts which will be important throughout the book, when talking about theories of design. We distinguish semantically between **functionalities** and **requirements/costs**. In general, you prefer **functionalities** to be “large” (Fig. 20.3) and **requirements/costs** to be “small” (Fig. 20.2).

We think of three achievable accuracy plots (Fig. 20.4).

First, we can plot trade-offs in costs and add a “feasibility” curve.

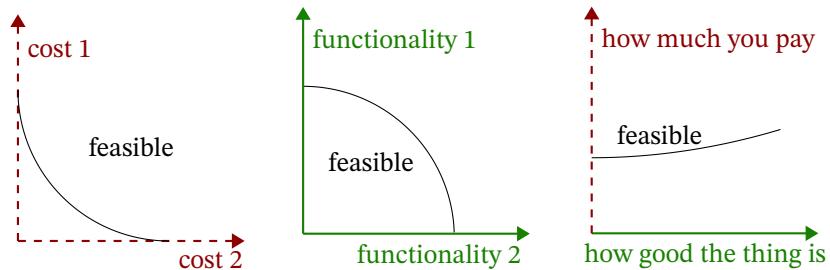


Figure 20.2.: Requirements/costs.



Figure 20.3.: Functionality.

Figure 20.4.



Everything above this curve is feasible and will cost more than what is *on* the curve. Second, we can plot trade-offs in functionalities and add a “feasibility” curve. Everything below the curve is feasible, but is below the “standards” required by the curve. Finally, we can plot functionality and resource together, representing the trade-offs between “how good a product is” and “how much one needs to pay for it. Feasible pairs are represented via the feasibility curve. Everything above the curve will be feasible (by paying more). A good exercise would be to open any engineering book, find the graphs talking about “achievable” performance and “resources” needed, and classify into one of the ones reported in Fig. 20.4. In the following we will have a careful look at specific examples involving trade-offs.

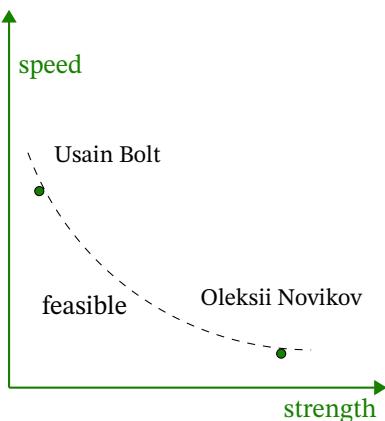


Figure 20.5.

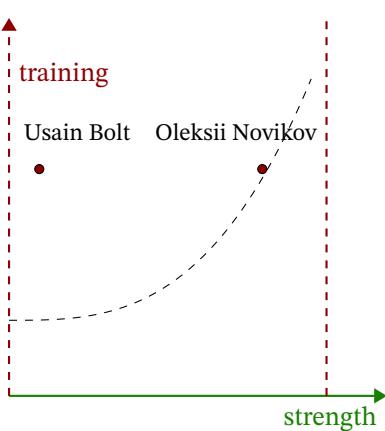
Trade-offs for the human body

A good example for trade-offs for the human body is given by sports. Indeed, when looking at different disciplines, various physical abilities are desired and trade-offs between them characterize athletes. For instance, we can think about trade-offs between *speed* and *strength* for humans (Fig. 20.5).

These are functionalities, which different athletes might want to maximize. Let’s consider Usain Bolt, who owns the 100 metres, 200 metres, and 4×100 metres relay world records. Without doubts, in the human speed-strength trade-off curve he positions himself close to the highest achievable speeds. At the same time, however, Usain Bolt is not among the strongest men in the world. To see the other end of the curve, we need to introduce Oleksii Novikov, who won the 2020 World’s Strongest Man competition. Similarly to Bolt, he is among the best in his discipline, reaching very high strength. Again, the speed-strength trade-off implies that Oleksii cannot be among the fastest men in the world, if he wants to be among the strongest ones. For this specific case, we can think of resources that make functionalities possible. In sports, resources are typically related to training (Fig. 20.6).

If we want to relate the invested training and the resulting strength reached by the athletes, we will notice that with a lot of training, Novikov will improve his results,

Figure 20.6.



approaching perfection. On the other hand, the kind of training Bolt undergoes is not optimizing strength, and therefore his results will be less effective.

Masks

Orders give us a rich way to describe products under various lenses. Recently, we all needed to become experts of protective masks. In this section, we will show various ways in which one can order the latter by functionality. By first thinking about the effectiveness of the mask in protecting the wearer from a virus, we can order masks as in Fig. 20.7.

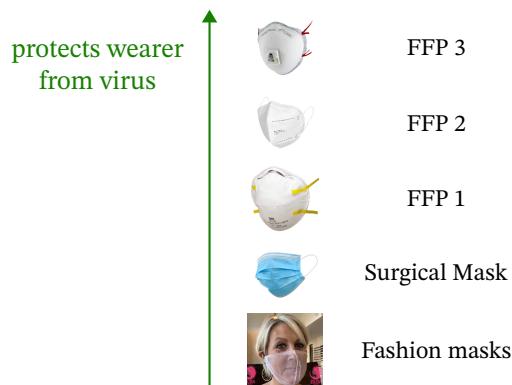


Figure 20.7.

In general masks are classified following their filter abilities and inward leakages. The FFP1 class filters at least 80 % of airborne particles and allows less than 22 % inward leakage. The FFP2 class filters at least 96 % of airborne particles and allows less than 8 % inward leakage, and the FFP3 class filters at least 99 % of airborne particles and allows less than 2 %. inward leakage

Obviously, based on the protection level, the most performant in Fig. 20.7 is FFP3, and the worst is the fashion one. However, this is not the only way in which we can classify masks. If, for instance, we want to consider a functionality “how much does the mask say about the wearer”, we can order the masks differently. Arguably, the ordering could look like the one in Fig. 20.8.

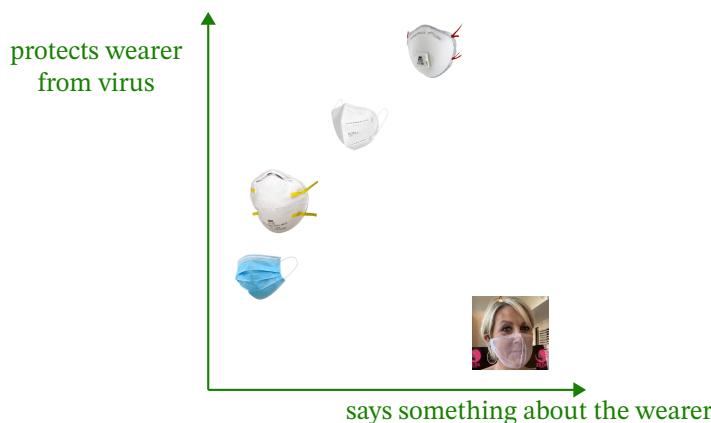


Figure 20.8.

Indeed, choosing a fashion mask might say that the wearer cares more about

aesthetics than safety, and choosing a FFP3 highlights responsible behaviors, care, and research in masks models.

Similarly, one could order masks based on different performance criteria, adding the functionality “how much does it protect others?” (Fig. 20.9).

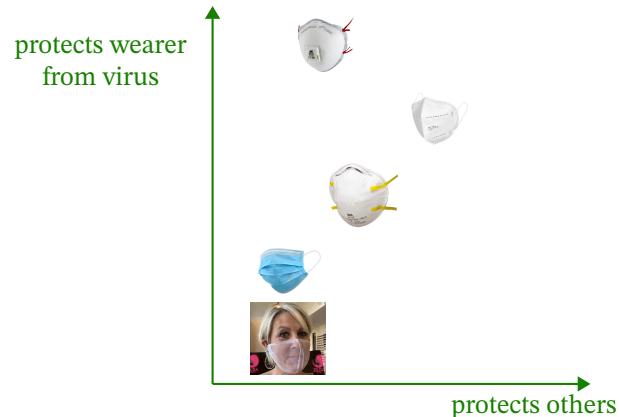


Figure 20.9.

On the other hand, one could think about the trade-offs between the mask performance and its cost, presenting a functionality-resource plot (Fig. 20.10).

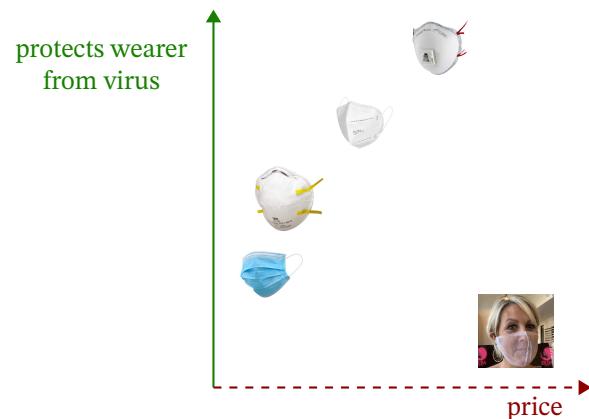


Figure 20.10.

More performant masks are typically more expensive, and the fashion mask will be probably the least performance and most expensive.

This example once again highlights the flexibility and richness of the “orders approach”. This will be much more evident in the next example.

Hats and headphones

Another good example of ordering of multiple functionalities and costs is the one of headphones. Let’s consider a set of headphones and let’s order them based on their abilities to “keep warm” and to “reproduce music” (Fig. 20.11).

Clearly, these two functionalities represent different objectives and diverse product ranges will satisfy them in different ways. For instance, winter hats clearly cannot reproduce music, but keep very warm. On the other hand, large headphones are the best in reproducing music, but cannot keep as warm as winter

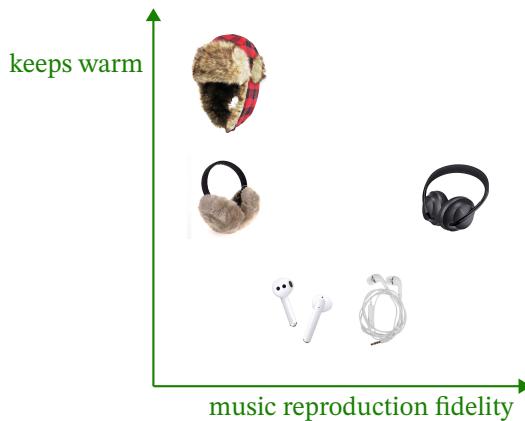


Figure 20.11.

hats. Functionalities come at a cost. For instance we could plot the trade-off between “keep warm” and price (Fig. 20.12).



Figure 20.12.

Other interesting costs could be expressed via the frequency of charging (Fig. 20.13), or the hassle of dealing with wires (Fig. 20.14).

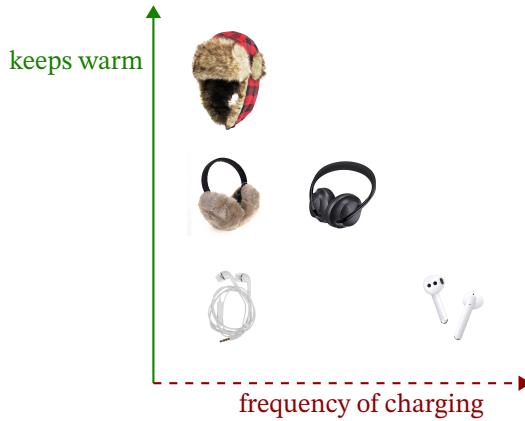


Figure 20.13.

It is interesting to notice that by considering all the aforementioned characteris-



Figure 20.14.

tics

$$(\text{keeps warm} \times \text{music fidelity}) \times (\text{price} \times \text{frequency of charging} \times \text{wires hassle}),$$

no product dominates another. This is also known as the *law of successful products*. At equilibrium, in an efficient and free market, no product completely dominates another by both functionality and costs. Otherwise, the dominated product would not sell. Once we *specify the design purpose* and the related constraints, we can (partially) order things.

20.3. 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 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: 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 *ordered sets*, and will introduce pre-orders, partial orders, and total orders.

Davey and Priestley [9] and Roman [28] are possible reference texts.

We introduce these concepts by adding levels of specificity.

Definition 20.1 (Pre-ordered set). A *pre-ordered set* is a tuple $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$, where \mathbf{P} is a set (also called the *carrier set*), together with a relation $\leq_{\mathbf{P}}$ that is:

▷ *Reflexive*:

$$\frac{\top}{p \leq_{\mathbf{P}} p} \quad (20.1)$$

▷ *Transitive*:

$$\frac{p \leq_{\mathbf{P}} q \quad q \leq_{\mathbf{P}} r}{p \leq_{\mathbf{P}} r} \quad (20.2)$$

By adding a property called *antisymmetry*, one obtains a partially ordered set.

Definition 20.2 (Partially ordered set). A pre-ordered set $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ is a *partially-ordered set (poset)* if the relation $\leq_{\mathbf{P}}$ is *antisymmetric*. In other words, if:

$$\frac{p \leq_{\mathbf{P}} q \quad q \leq_{\mathbf{P}} p}{p = q} \quad (20.3)$$

Definition 20.3 (Totally ordered set). A partially ordered set $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ is a *totally ordered set* if the relation $\leq_{\mathbf{P}}$ is *total*. In other words, if:

$$\frac{\top}{(p \leq_{\mathbf{P}} q) \vee (q \leq_{\mathbf{P}} p)} \quad (20.4)$$

Remark 20.4. There are more general ways to describe preferences. One has:

▷ *Quasitransitive relations*, which are relations $\leq_{\mathbf{P}}$ over a set \mathbf{P} for which

$$\frac{(p \leq_{\mathbf{P}} q) \quad \neg(q \leq_{\mathbf{P}} p) \quad (q \leq_{\mathbf{P}} r) \quad \neg(r \leq_{\mathbf{P}} q)}{(p \leq_{\mathbf{P}} r) \wedge \neg(r \leq_{\mathbf{P}} p)} \quad (20.5)$$

▷ *Semiorders*, which are relations $\leq_{\mathbf{P}}$ over a set \mathbf{P} such that:

- *Asymmetry* (not antisymmetry!) holds:

$$\frac{p \leq_{\mathbf{P}} q}{\neg(p \leq_{\mathbf{P}} p)} \quad (20.6)$$

- Let's denote two elements $q, r \in \mathbf{P}$ which are *incomparable* by $q \sim r$.

One has:

$$\frac{p \leq_{\mathbf{P}} q \quad q \sim r \quad r \leq_{\mathbf{P}} s}{p \leq_{\mathbf{P}} s} \quad (20.7)$$

- One has:

$$\frac{p \leq_{\mathbf{P}} q \quad q \leq_{\mathbf{P}} r}{(p \leq_{\mathbf{P}} s) \vee (s \leq_{\mathbf{P}} p)} \quad (20.8)$$

An example of semiorder is the following. Say that you have to express your preference over numbers in $\mathbf{P} = \{10, 11, 12\}$. You are indifferent between 10 and 11, and between 11 and 12. Though, you prefer 10 to 12.

A *Hasse diagram* is an economical (in terms of arrows) way to visualize a poset. In a Hasse diagram elements are points, and if $p \leq_{\mathbf{P}} q$ then p is drawn lower than q 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 $\langle \mathbb{R}_{\geq 0}, \leq \rangle$ (actually, both are particularly totally ordered sets). Imagine that you have recipes costing 1 ₩, 2 ₩, and 3 ₩. This can be represented as in Fig. 20.15.

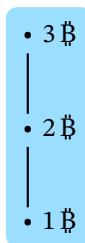


Figure 20.15.: The cost of pizza ingredients can be represented as a poset.

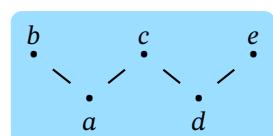


Figure 20.16.: Example of Hasse diagram of \mathbf{P} .



Figure 20.17.: The singleton poset.

Example 20.5. Consider a poset $\mathbf{P} = \{a, b, c, d, e\}$ with $a \leq_{\mathbf{P}} b$, $a \leq_{\mathbf{P}} c$, $d \leq_{\mathbf{P}} c$, and $d \leq_{\mathbf{P}} e$. This can be represented with a Hasse diagram as in Fig. 20.16.

Example 20.6 (Singleton poset). If a set has only one element, say \bullet , then there is a unique order relation on it (Fig. 20.17). We denote the resulting poset again by $\{\bullet\}$.

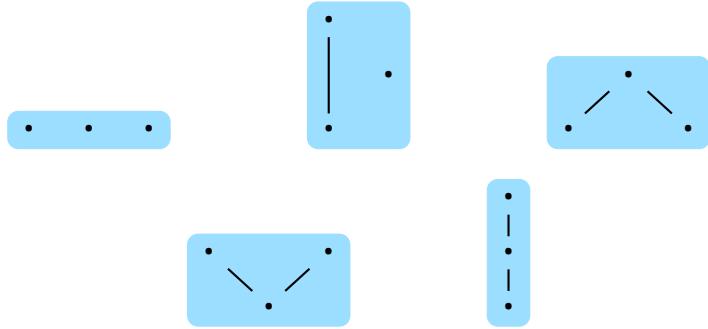


Figure 20.18.: All posets on 3-elements sets, up to isomorphisms.

Example 20.7. 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. 20.17). On 2-elements sets, one has the posets reported in Fig. 20.19. On 3-elements sets,



Figure 20.19.: All posets on 2-elements sets, up to isomorphisms.

one has the posets reported in Fig. 20.18.

On 4-elements sets, one has the posets reported in Fig. 20.20.

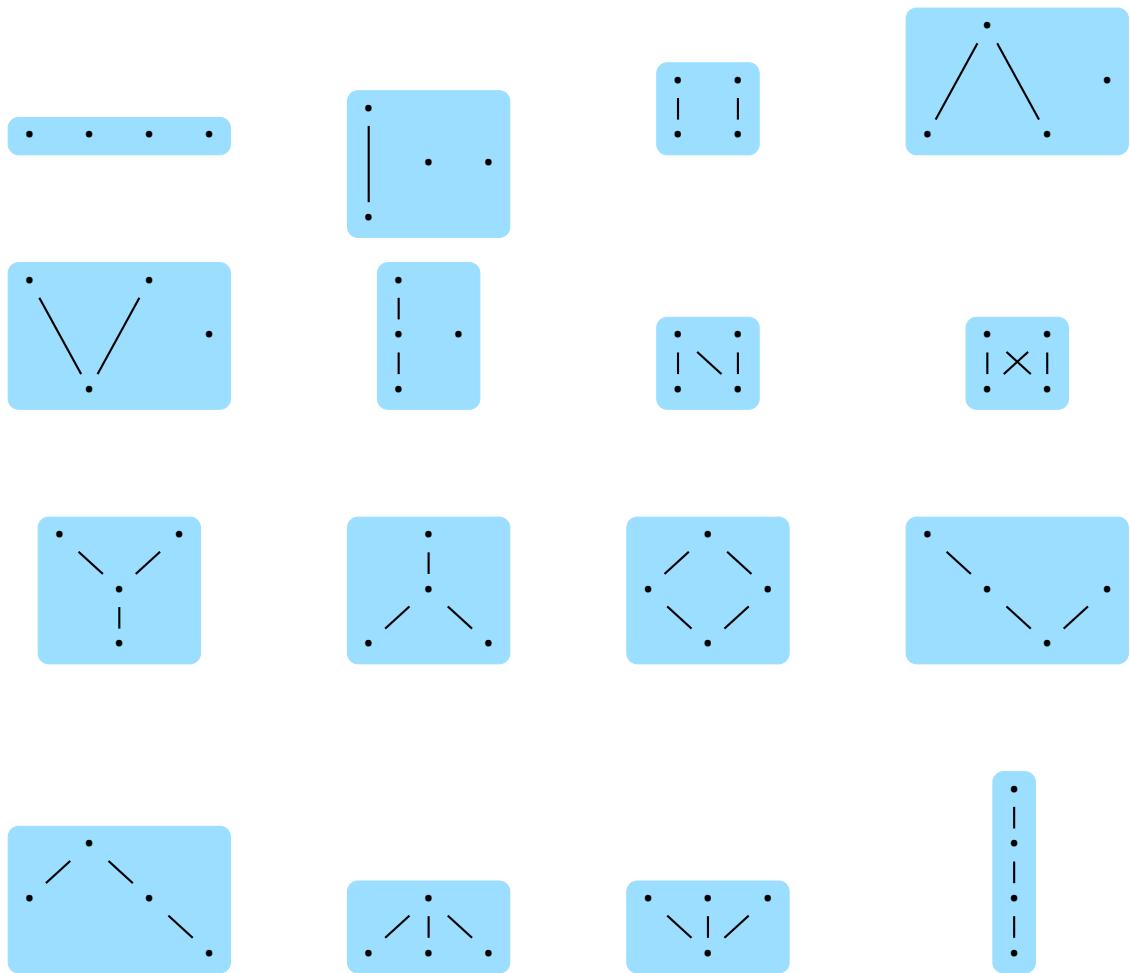


Figure 20.20.: All posets on 4-elements sets, up to isomorphisms.



Figure 20.21.

a	b	$a \leq b$	$a \wedge b$	$a \vee b$
T	T	T	T	T
T	⊥	⊥	⊥	T
⊥	T	T	⊥	T
⊥	⊥	T	⊥	⊥

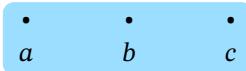
Table 20.1.: Properties of the **Bool** poset. Note that $\leq \Rightarrow$.

Figure 20.22.: Example of a discrete poset.

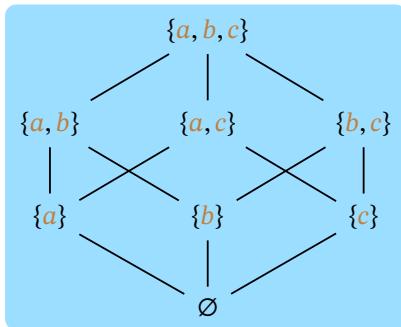


Figure 20.23.: Power set as a category.

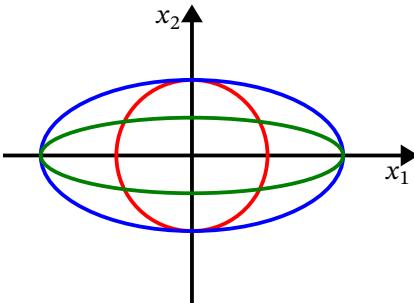


Figure 20.24.: Example of ellipses representing positive semi-definite matrices.

Example 20.8 (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. 20.21). This relation should be familiar from Table 20.1.

In addition to the operation

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

called *Imp*, 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 20.9 (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 20.10 (Discrete partially ordered sets). Every set A can be considered as a *discrete poset* $\langle A, = \rangle$. Discrete posets are represented as collection of points (Fig. 20.22).

Example 20.11. A symmetric matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ is *positive semi-definite* if $x^T \mathbf{M} x \geq 0$ for all non-zero $x \in \mathbb{R}^n$. We call the set of all such matrices \mathcal{P}^n . Such matrices have real, semi-positive eigenvalues, which can be interpreted as axes lengths of ellipsoids. Any matrix $\mathbf{A} \in \mathcal{P}^n$ describes an ellipsoid, descriptive equation of which can be written as a quadratic form:

$$x^T \mathbf{A} x = 1, \quad x \in \mathbb{R}^n.$$

We can define a partial order on \mathcal{P}^n as:

$$\mathbf{A} \leq \mathbf{B} \Leftrightarrow (\mathbf{A} - \mathbf{B}) \in \mathcal{P}^n, \quad \mathbf{A}, \mathbf{B} \in \mathcal{P}^n$$

The order can be interpreted as an inclusion of ellipsoids. Take for instance the matrices

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}.$$

The order on the set $\{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$ is reported in Fig. 20.25, and it is easily explained via Fig. 20.24. The ellipse representing \mathbf{A} (in red) is included by the one representing matrix \mathbf{B} (in blue), but not by the one representing matrix \mathbf{C} (in green). Furthermore, the one representing \mathbf{B} includes the one representing \mathbf{C} .

Example 20.12. Given a set $A = \{a, b, c\}$, consider its power set $\mathcal{P}(A)$. Define sets as the objects of this new category and define the morphisms to be inclusions (Fig. 20.23).

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, meaning that if $\mathbf{A} \subseteq \mathbf{B} \subseteq \mathbf{C}$, then $\mathbf{A} \subseteq \mathbf{C}$.

A note on preorders The theory of design problems can be easily generalized to preorders. This means that there could be two elements p and q such that $p \leq_p q$ and $p \geq_p q$ but $p \neq q$.

This is actually common in practice. For example, if the order relation comes from human judgement, such as customer preference, all bets are off regarding the consistency of the relation. We will only refer to posets for two reasons:

1. The exposition is smoother.
2. Given a pre-order, computation will always involve passing to the poset representation.

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

$$p \simeq q \equiv (p \leq_{\mathbf{P}} q) \wedge (q \leq_{\mathbf{P}} p). \quad (20.9)$$

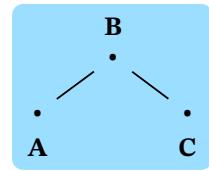


Figure 20.25.: Example of order between positive semi-definite matrices.

Graded exercise 1 ([PolynomialDivisibility](#)). Let \mathbf{A} be the set of all polynomials with coefficients in \mathbb{R} . Recall that a polynomial p divides a polynomial q if there exists a polynomial m such that $p \cdot m = q$. If p divides q we denote this by $p|q$. Divisibility defines an endorelation on \mathbf{A} by saying p is related to q iff $p|q$. Does this define a preorder structure on \mathbf{A} ? Does this define a poset structure on \mathbf{A} ? Justify your answer.

20.4. Chains and Antichains

Definition 20.13 (Chain in a poset). Given a poset \mathbf{P} , a *chain* is a sequence of elements p_i in \mathbf{P} where two successive elements are comparable:

$$\frac{i \leq j}{p_i \leq_{\mathbf{P}} p_j} \quad (20.10)$$

Definition 20.14 (Antichain in a poset). An *antichain* is a subset \mathbf{S} of a poset where no elements are comparable. If $a, b \in \mathbf{S}$, then:

$$\frac{a \leq_{\mathbf{P}} b}{a = b} \quad (20.11)$$

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

Remark 20.15. Note that given a poset $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$, the empty set \emptyset is both a chain and an antichain.

In the context of pizza recipes, consider the diagram reported in Fig. 20.26. The blue points represent an antichain of recipes $\{\langle 1 \text{ \AA}, 2 \text{ h} \rangle, \langle 2 \text{ \AA}, 1 \text{ h} \rangle\}$. It is a set of antichains because they 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 $\langle 2 \text{ \AA}, 1 \text{ h} \rangle$.

Example 20.16. Let's consider the poset $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ where $p \leq_{\mathbf{P}} q$ if p is a divisor of q and $\mathbf{P} = \{1, 5, 10, 11, 13, 15\}$. A chain of \mathbf{P} is $\{1, 5, 10, 15\}$. An antichain of \mathbf{P} is $\{10, 11, 13\}$.

Example 20.17. Consider Example 20.12. Examples of chains are

$$\{\emptyset, \{a\}, \{a, b\}, \{a, b, c\}\}, \quad \{\emptyset, \{b\}, \{b, c\}, \{a, b, c\}\}. \quad (20.12)$$

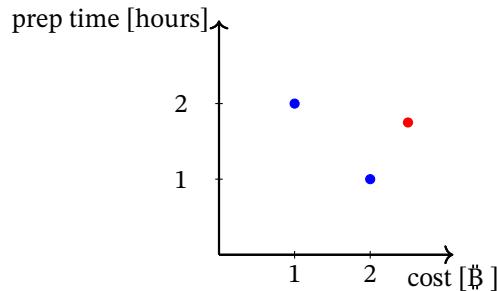


Figure 20.26.: Example of antichains.

Examples of antichains are

$$\{\{a\}, \{b\}, \{c\}\}, \quad \{\{a, b\}, \{a, c\}, \{b, c\}\}. \quad (20.13)$$

Example 20.18. 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 $\langle 10 \text{ €}, 1 \text{ kg} \rangle$ is better than a model $\langle 11 \text{ €}, 1.1 \text{ kg} \rangle$. Also, there may be models which are incomparable, which form an antichain. For example, you cannot say whether $\langle 10 \text{ €}, 1 \text{ kg} \rangle$ is better than $\langle 5 \text{ €}, 2 \text{ kg} \rangle$. The incomparable models form an antichain.

20.5. Upper and lower sets

Definition 20.19 (Upper set). An *upper set* \mathbf{U} is a subset of a poset \mathbf{P} such that, if an element is inside, all elements above it are inside as well. In other words:

$$\frac{x \in \mathbf{U} \quad x \leq_{\mathbf{P}} y}{y \in \mathbf{U}} \quad (20.14)$$

We call $\mathcal{U}\mathbf{P}$ the set of upper sets of \mathbf{P} .

Definition 20.20 (Lower set). A *lower set* \mathbf{L} is a subset of a poset \mathbf{P} if, if a point is inside, all points below it are inside as well. In other words:

$$\frac{x \in \mathbf{L} \quad y \leq_{\mathbf{P}} x}{y \in \mathbf{L}} \quad (20.15)$$

We call $\mathcal{L}\mathbf{P}$ the set of lower sets of \mathbf{P} .

Consider the blue poset of pizza recipes from before. The upper and lower sets of this poset can be represented as in Fig. 20.27. 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.

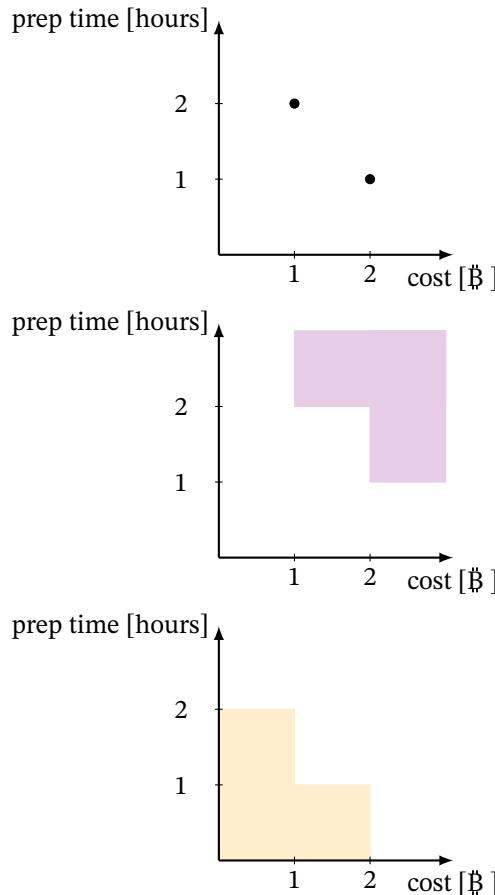


Figure 20.27.: Example of upper and lower sets of a poset of pizza recipes.

20.6. From antichains to uppersets, and viceversa

Definition 20.21 (Upper closure operator). The *upper closure operator* \uparrow maps a subset to the smallest upper set that includes it:

$$\begin{aligned} \uparrow : \mathcal{P}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{P} \\ S &\mapsto \{y \in \mathbf{P} \mid \exists x \in S : x \leq_{\mathbf{P}} y\}. \end{aligned} \tag{20.16}$$

Remark 20.22. Note that, by definition, an upper set is closed to upper closure.

Lemma 20.23. For any $S \in \mathcal{P}\mathbf{P}$, $\uparrow S$ is in fact an upper set.

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

Lemma 20.24. The upper closure operator \uparrow is a monotone map.

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

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

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

Lemma 20.25. Let A and B be subsets of P that are antichains. Then

$$\frac{\uparrow A = \uparrow B}{A = B}$$

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. $p_1 = \langle 1 \text{ \AA}, 2 \text{ h} \rangle$ (Fig. 20.28). Then, consider the case of two

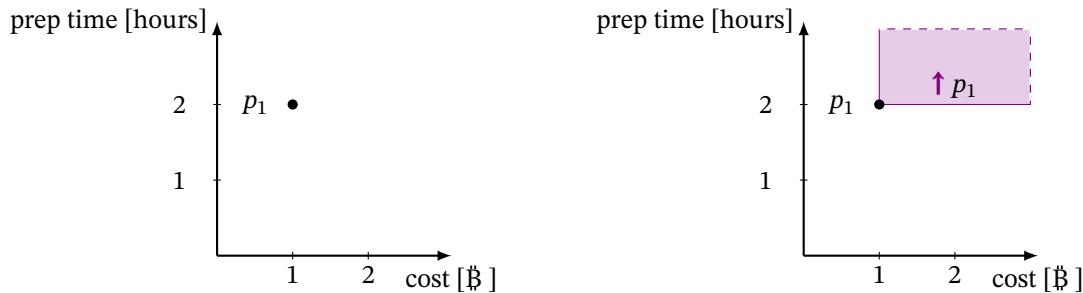


Figure 20.28.: The upper closure of a singleton set of pizza recipes.

elements, with $p_2 = \langle 2 \text{ \AA}, 1 \text{ h} \rangle$ (Fig. 20.29).

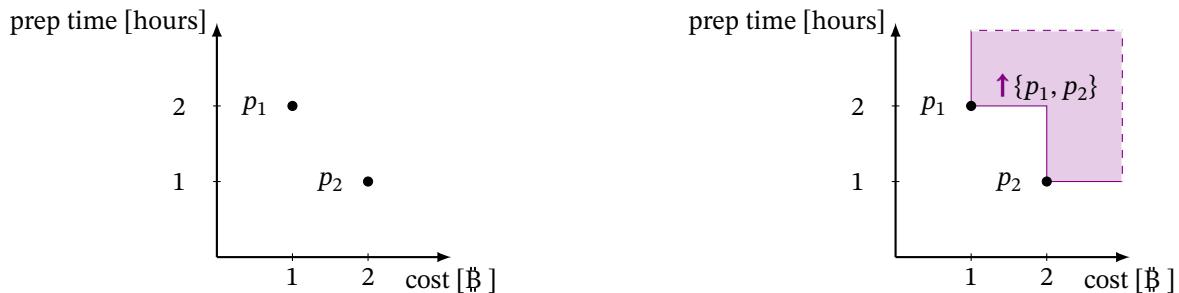


Figure 20.29.: The upper closure of a set of pizza recipes.

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 20.26 (Lower closure operator). The *lower closure operator* \downarrow maps

a subset to the smallest lower set that includes it:

$$\downarrow : \mathcal{P} \rightarrow \mathcal{L}\mathbf{P}$$

$$S \mapsto \{y \in P \mid \exists x \in S : y \leq_P x\}.$$

Lemma 20.27. The lower closure operator \downarrow is a monotone map.

Exercise 16. Prove Lemma 20.27.

See solution on page 183.

Example 20.28. Consider the battery example of Example 20.18, and the antichain given by the battery models $a = \langle 10 \text{ \AA}, 1 \text{ kg} \rangle$, $b = \langle 20 \text{ \AA}, 0.5 \text{ kg} \rangle$, and $c = \langle 30 \text{ \AA}, 0.25 \text{ kg} \rangle$ (Fig. 20.30). The lower closure operator $\downarrow \{a, b, c\}$ represents all the battery models which, if existing, would dominate $\{a, b, c\}$.



Figure 20.30.: Battery example. From the left: antichain, upper closure, and lower closure.

Definition 20.29 (Min). $\text{Min} : \mathcal{P} \rightarrow \mathcal{A}\mathbf{P}$ 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 $a \leq_P b$ for all $b \in S$. In formulas:

$$\text{Min} : \mathcal{P} \rightarrow \mathcal{A}\mathbf{P}$$

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

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

Definition 20.30 (Max). $\text{Max} : \mathcal{P} \rightarrow \mathcal{A}\mathbf{P}$ 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:

$$\text{Max} : \mathcal{P} \rightarrow \mathcal{A}\mathbf{P}$$

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

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

Lemma 20.31. Given a poset $\langle P, \leq_P \rangle$, $\langle \mathcal{A}\mathbf{P}, \leq_{\mathcal{A}\mathbf{P}} \rangle$ is a poset with

$$A \leq_{\mathcal{A}\mathbf{P}} B \text{ if and only if } \uparrow A \supseteq \uparrow B. \quad (20.17)$$

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

Proof. We need to show the poset properties (Def. 20.2). We can prove the following:

▷ *Reflexivity:* From $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ being a poset we know that

$$\{y \in \mathbf{P} \mid \exists x \in A : x \leq_{\mathbf{P}} y\} \supseteq \{y \in \mathbf{P} \mid \exists x \in A : x \leq_{\mathbf{P}} y\}, \quad (20.18)$$

$$\uparrow A = \uparrow A$$

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

▷ *Antisymmetry:* One has

$$\begin{aligned} (A \leq_{\mathcal{AP}} B) \wedge (B \leq_{\mathcal{AP}} A) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow A) \\ &\Leftrightarrow \uparrow A = \uparrow B \\ &\Rightarrow A = B. \end{aligned} \quad (20.19)$$

The last implication is by Lemma 20.25.

▷ *Transitivity:* One has

$$\begin{aligned} (A \leq_{\mathcal{AP}} B) \wedge (B \leq_{\mathcal{AP}} C) &\Leftrightarrow (\uparrow A \supseteq \uparrow B) \wedge (\uparrow B \supseteq \uparrow C) \\ &\Rightarrow \uparrow A \supseteq \uparrow C \\ &\Rightarrow A \leq_{\mathcal{AP}} C. \end{aligned} \quad (20.20)$$

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

□

Definition 20.32 (Downward closed set). An upper set S is *downward-closed* in a poset \mathbf{P} if

$$S = \uparrow \text{Min } S. \quad (20.21)$$

The set of downward-closed upper sets of \mathbf{P} is denoted $\underline{\mathcal{U}}\mathbf{P}$.

Measuring posets

Definition 20.33 (Width of a poset). The *width* of a poset, denoted $\text{width}(\mathbf{P})$, is the maximum cardinality of an antichain in \mathbf{P} .

Definition 20.34 (Height of a poset). The *height* of a poset, denoted $\text{height}(\mathbf{P})$, is the maximum cardinality of a chain in \mathbf{P} .

Exercise 17. If you know the width of the posets \mathbf{P} and \mathbf{Q} , can you compute the width of $\mathbf{P} \times \mathbf{Q}$?

See solution on page 183.

Exercise 18. If you know the height of the posets \mathbf{P} and \mathbf{Q} , can you compute the height of $\mathbf{P} \times \mathbf{Q}$?

See solution on page 183.

20.7. Lattices

Definition 20.35 (Lattice). A *lattice* is a poset $\langle P, \leq_P \rangle$ with some additional properties:

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

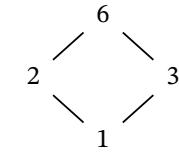
Remark 20.36 (Bounded lattices). If there is a least upper bound for the entire lattice P , it is called the *top* (\top). If a greatest lower bound exists it is called the *bottom* (\perp). If both a top and a bottom exist, we call the lattice *bounded*, and denote it by $\langle P, \leq, \vee, \wedge, \perp, \top \rangle$.

Example 20.37. In Example 20.12 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 $\langle \mathcal{P}A, \subseteq, A, \cap \rangle$.

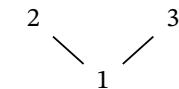
Example 20.38. Consider the set $\{1, 2, 3, 6\}$ ordered by divisibility. For instance, since 2 divides 6, we have $2 \leq 6$. This is a lattice. However, the set $\{1, 2, 3\}$ ordered by divisibility is not, since 2 and 3 lack a meet (Fig. 20.31).

Graded exercise 2 (UpperLowerBounds). Let $A = \{a, b, c, d, e\}$. Give examples of the following situations using Hasse diagrams. In each case, provide a poset structure on A and a subset $B \subseteq A$ such that:

1. B has a least upper bound;
2. B has a greatest lower bound;
3. B has no least upper bound;
4. B has no greatest lower bound.



(a)



(b)

Figure 20.31.: Examples of a lattice and a non-lattice.

Lemma 20.39. $\mathcal{U}P$ is a bounded lattice (Def. 20.35) with

$$\begin{aligned}
 \leq_{\mathcal{U}P} &:= \supseteq \\
 \perp_{\mathcal{U}P} &:= P \\
 \top_{\mathcal{U}P} &:= \emptyset \\
 \vee_{\mathcal{U}P} &:= \cap \\
 \wedge_{\mathcal{U}P} &:= \cup.
 \end{aligned} \tag{20.22}$$

Proof. Consider the poset $\langle \mathcal{U}P, \supseteq \rangle$ and $A, B \in \mathcal{U}P$.

- ▷ First, we need to show that $A \cap B \in \mathcal{U}P$. One has $A \subseteq \mathcal{U}P$ and $B \subseteq \mathcal{U}P$, meaning that by definition, if $a \in A \cap B$, we have $a \in A \wedge a \in B$. It follows that $a \in \mathcal{U}P$ for all $a \in A \cap B$. Furthermore, we need to show that $A \cap B$ is the least upper bound of A and B . Assume this

is not true, meaning that there exists a $\mathbf{C} \in \mathcal{U}\mathbf{P}$, $\mathbf{C} \neq \mathbf{A} \cap \mathbf{B}$, such that $\mathbf{A} \supseteq \mathbf{C} \supseteq \mathbf{A} \cap \mathbf{B}$ and $\mathbf{B} \supseteq \mathbf{C} \supseteq \mathbf{A} \cap \mathbf{B}$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} \mathbf{A} \cap \mathbf{B} &\supseteq \mathbf{C} \cap \mathbf{C} \supseteq \mathbf{A} \cap \mathbf{B} \\ \mathbf{A} \cap \mathbf{B} &\supseteq \mathbf{C} \supseteq \mathbf{A} \cap \mathbf{B} \\ \mathbf{C} &= \mathbf{A} \cap \mathbf{B}, \end{aligned} \tag{20.23}$$

which contradicts the assumption. Therefore, $\mathbf{A} \cap \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} .

- ▷ Second, we need to show that $\mathbf{A} \cup \mathbf{B} \in \mathcal{U}\mathbf{P}$. One has $\mathbf{A} \subseteq \mathcal{U}\mathbf{P}$ and $\mathbf{B} \subseteq \mathcal{U}\mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cup \mathbf{B}$, we have either $a \in \mathbf{A}$ or $a \in \mathbf{B}$. If $a \in \mathbf{A}$, then $a \in \mathcal{U}\mathbf{P}$. If $a \in \mathbf{B}$, then $a \in \mathcal{U}\mathbf{P}$. It follows that $a \in \mathcal{U}\mathbf{P}$ for all $a \in \mathbf{A} \cup \mathbf{B}$. Furthermore, we need to show that $\mathbf{A} \cup \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} . Assume this is not true, meaning that there exists a $\mathbf{C} \in \mathcal{U}\mathbf{P}$, $\mathbf{C} \neq \mathbf{A} \cup \mathbf{B}$, such that $\mathbf{A} \cup \mathbf{B} \supseteq \mathbf{C} \supseteq \mathbf{A}$ and $\mathbf{A} \cup \mathbf{B} \supseteq \mathbf{C} \supseteq \mathbf{B}$. Using the fact that union preserves inclusions, one has

$$\begin{aligned} (\mathbf{A} \cup \mathbf{B}) \cup (\mathbf{A} \cup \mathbf{B}) &\supseteq \mathbf{C} \cup \mathbf{C} \supseteq \mathbf{A} \cup \mathbf{B} \\ \mathbf{A} \cup \mathbf{B} &\supseteq \mathbf{C} \supseteq \mathbf{A} \cup \mathbf{B} \\ \mathbf{C} &= \mathbf{A} \cup \mathbf{B}, \end{aligned} \tag{20.24}$$

which contradicts the assumption. Therefore, $\mathbf{A} \cup \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} .

We have therefore proved that $(\mathcal{U}\mathbf{P}, \supseteq)$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq \mathbf{C}$ for any $\mathbf{C} \in \mathcal{U}\mathbf{P}$, meaning that \emptyset is the top. Furthermore, we notice that $\mathbf{C} \subseteq \mathbf{P}$ for any $\mathbf{C} \in \mathcal{U}\mathbf{P}$, meaning that \mathbf{P} is a bottom. Therefore, the lattice is bounded. \square

Lemma 20.40. $\mathcal{L}\mathbf{P}$ is a bounded lattice (Def. 20.35) with:

$$\begin{aligned} \leq_{\mathcal{L}\mathbf{P}} &:= \subseteq \\ \perp_{\mathcal{L}\mathbf{P}} &:= \emptyset \\ \top_{\mathcal{L}\mathbf{P}} &:= \mathbf{P} \\ \vee_{\mathcal{L}\mathbf{P}} &:= \cup \\ \wedge_{\mathcal{L}\mathbf{P}} &:= \cap. \end{aligned} \tag{20.25}$$

Proof. Consider the poset $(\mathcal{L}\mathbf{P}, \subseteq)$ and $P, Q \in \mathcal{L}\mathbf{P}$.

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

one has

$$\begin{aligned} \mathbf{A} \cup \mathbf{B} &\subseteq \mathbf{C} \cup \mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B} \\ \mathbf{A} \cup \mathbf{B} &\subseteq \mathbf{C} \subseteq \mathbf{A} \cup \mathbf{B} \\ \mathbf{C} &= \mathbf{A} \cup \mathbf{B}, \end{aligned} \tag{20.26}$$

which contradicts the assumption. Therefore, $\mathbf{A} \cup \mathbf{B}$ is the least upper bound of \mathbf{A} and \mathbf{B} .

- ▷ Second, we need to show that $\mathbf{A} \cap \mathbf{B} \in \mathcal{L}\mathbf{P}$. One has $\mathbf{A} \subseteq \mathcal{L}\mathbf{P}$ and $\mathbf{B} \subseteq \mathcal{L}\mathbf{P}$, meaning that by definition, if $a \in \mathbf{A} \cap \mathbf{B}$, we have $a \in \mathbf{A} \wedge a \in \mathbf{B}$ ($a \in \mathcal{L}\mathbf{P}$, for all $a \in \mathbf{A} \cap \mathbf{B}$). Furthermore, we need to show that $\mathbf{A} \cap \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} . Assume this is not true, meaning there exists a $\mathbf{C} \in \mathcal{L}\mathbf{P}$, $\mathbf{C} \neq \mathbf{A} \cap \mathbf{B}$, such that $\mathbf{A} \cap \mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{A}$ and $\mathbf{A} \cap \mathbf{B} \subseteq \mathbf{C} \subseteq \mathbf{B}$. Using the fact that intersection preserves inclusions, one has

$$\begin{aligned} (\mathbf{A} \cap \mathbf{B}) \cap (\mathbf{A} \cap \mathbf{B}) &\subseteq \mathbf{C} \cap \mathbf{C} \subseteq \mathbf{A} \cap \mathbf{B} \\ \mathbf{A} \cap \mathbf{B} &\subseteq \mathbf{C} \subseteq \mathbf{A} \cap \mathbf{B} \\ \mathbf{C} &= \mathbf{A} \cap \mathbf{B}, \end{aligned} \tag{20.27}$$

which contradicts the assumption. Therefore, $\mathbf{A} \cap \mathbf{B}$ is the greatest lower bound of \mathbf{A} and \mathbf{B} .

We have therefore proved that $(\mathcal{L}\mathbf{P}, \subseteq)$ is a lattice. To show that it is bounded, we notice that $\emptyset \subseteq \mathbf{C}$ for any $\mathbf{C} \in \mathcal{L}\mathbf{P}$, meaning that \emptyset is the bottom. Furthermore, we notice that $\mathbf{C} \subseteq \mathbf{P}$ for any $\mathbf{C} \in \mathcal{L}\mathbf{P}$, meaning that \mathbf{P} is a top. Therefore, the lattice is bounded. \square



21. Poset constructions

21.1. Product of posets

We can think of the product of posets.

Definition 21.1 (Product of posets). Given two posets $\langle P, \leq_P \rangle$ and $\langle Q, \leq_Q \rangle$, the *product poset* is $\langle P \times Q, \leq_{P \times Q} \rangle$, where $P \times Q$ is the Cartesian product of two sets (Def. 24.1) and the order $\leq_{P \times Q}$ is given by:

$$\frac{\langle p_1, q_1 \rangle \leq_{P \times Q} \langle p_2, q_2 \rangle}{(p_1 \leq_P p_2) \wedge (q_1 \leq_Q q_2)} \quad (21.1)$$

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 ₣, 2 ₣, and 3 ₣, and the time poset containing elements 1 hours, and 2 hours, one can represent the product as in Fig. 21.1.

Example 21.2. Consider now two posets and their product, given in Fig. 21.2.

21.1 Product of posets	173
21.2 Disjoint union of posets	174
21.3 Opposite of a poset	175
21.4 Poset of intervals	175
21.5 A different poset of intervals	175

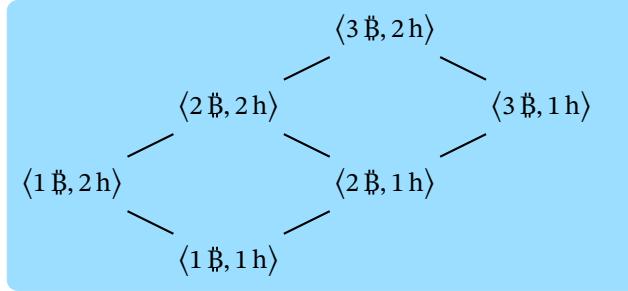


Figure 21.1.: Product poset of time and cost for pizza recipes.

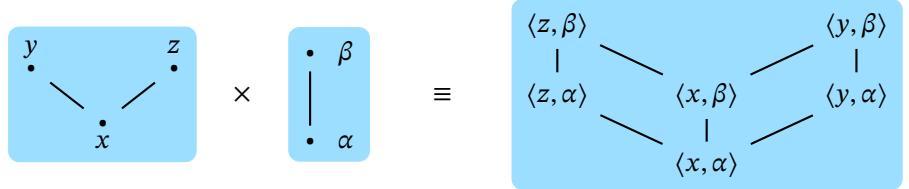


Figure 21.2.: Product of two posets.

21.2. Disjoint union of posets

Similarly to what we have done for sets in Section 24.2, we can think of alternatives in the poset case through their disjoint union.

Definition 21.3 (Disjoint union of posets). Given posets $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ and $\langle \mathbf{Q}, \leq_{\mathbf{Q}} \rangle$, we can define their *disjoint union* $\langle \mathbf{P} + \mathbf{Q}, \leq_{\mathbf{P}+\mathbf{Q}} \rangle$, where $\mathbf{P} + \mathbf{Q}$ is the disjoint union of the sets \mathbf{P} and \mathbf{Q} (Def. 24.22), and the order $\leq_{\mathbf{P}+\mathbf{Q}}$ is given by:

$$p \leq_{\mathbf{P}+\mathbf{Q}} q \equiv \begin{cases} p \leq_{\mathbf{P}} q, & p, q \in \mathbf{P}, \\ p \leq_{\mathbf{Q}} q, & p, q \in \mathbf{Q}, \end{cases} \quad (21.2)$$

with

$$\begin{aligned} \leq_{\mathbf{P}+\mathbf{Q}} : (\mathbf{P} + \mathbf{Q}) \times (\mathbf{P} + \mathbf{Q}) &\rightarrow \text{Bool} \\ \langle 1, p_1 \rangle, \langle 1, p_2 \rangle &\mapsto (p_1 \leq_{\mathbf{P}} p_2) \\ \langle 2, q_1 \rangle, \langle 1, q_2 \rangle &\mapsto \perp \\ \langle 1, p \rangle, \langle 2, q \rangle &\mapsto \perp \\ \langle 2, q_1 \rangle, \langle 2, q_2 \rangle &\mapsto (q_1 \leq_{\mathbf{Q}} q_2). \end{aligned} \quad (21.3)$$

Example 21.4. Consider the posets $\mathbf{P} = \langle \diamond, \star \rangle$ with $\diamond \leq_{\mathbf{P}} \star$, and $\mathbf{Q} = \langle \dagger, * \rangle$, with $* \leq_{\mathbf{Q}} \dagger$. Their disjoint union can be represented as in Fig. 21.3.



Figure 21.3.: Disjoint union of posets.

21.3. Opposite of a poset

Definition 21.5. The *opposite* of a poset $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ is the poset denoted as $\langle \mathbf{P}^{\text{op}}, \leq_{\mathbf{P}}^{\text{op}} \rangle$ that has the same elements as \mathbf{P} and the reverse ordering (Fig. 21.4). For a given $p \in \mathbf{P}$, we use p^* to represent its corresponding copy in \mathbf{P}^{op} ; note that p and p^* belong to distinct posets. Reversing the order means that, for all $p, q \in \mathbf{P}$,

$$\frac{p \leq_{\mathbf{P}} q}{q^* \leq_{\mathbf{P}}^{\text{op}} p^*} \quad (21.4)$$

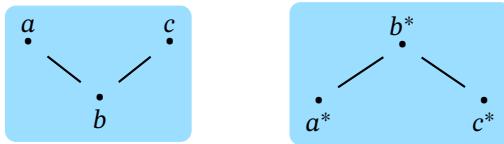


Figure 21.4.: Opposite of a poset.

Example 21.6 (Credit and debt). Let us define the set

$$\mathbf{P} = \mathbb{R} \times \{\text{€}\} = \{0.00, 0.01, 0.02, \dots\}$$

of all € monetary quantities approximated to the cent. From this set we can define two posets: $\mathbf{P}^+ = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ and $\mathbf{P}^- = \langle \mathbf{P}, \geq_{\mathbf{P}} \rangle$, 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 \mathbf{P}^+ , otherwise we are working in \mathbf{P}^- (for example because it represents the price at which we are selling our product). Traditionally, in double-entry ledger systems, the numbers were not written with negative signs, but rather in color: red and black. From this convention we get the idioms “being in the black” and “being in the red”.

21.4. Poset of intervals

Definition 21.7 (Poset of intervals). An interval is an ordered pair of elements $\langle p, q \rangle$ of \mathbf{P} , such that $p \leq_{\mathbf{P}} q$. Given a poset \mathbf{P} , one can define a *poset of intervals* on \mathbf{P} . Intervals can be ordered by inclusion:

$$\frac{\langle p_1, q_1 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_2, q_2 \rangle}{(p_1 \leq_{\mathbf{P}} p_2) \wedge (q_2 \leq_{\mathbf{P}} q_1)} \quad (21.5)$$

Exercise 19. Check that the relation defined in Def. 21.7 is indeed a poset.

See solution on page 183.

21.5. A different poset of intervals

Definition 21.8 (Another poset of intervals). Given a partially ordered set \mathbf{P} , an interval is an ordered pair of elements $\langle l, u \rangle$ of \mathbf{P} , such that $l \leq_{\mathbf{P}} u$. One can

define a *poset of intervals* on \mathbf{P} , denoted $\text{Int}'(\mathbf{P})$. Intervals can be ordered using the following rule:

$$\frac{\langle p_1, p_2 \rangle \leq_{\text{Int}'(\mathbf{P})} \langle q_1, q_2 \rangle}{(p_1 \leq_{\mathbf{P}} q_1) \wedge (p_2 \leq_{\mathbf{P}} q_2)} \quad (21.6)$$

This partially ordered set will be instrumental when we define uncertainty in design problems.

Exercise 20. Check that the relation defined in Def. 21.8 is indeed a poset.

See solution on page ??.



22. Life is hard

22.1. Monotone maps

Definition 22.1 (Monotone map). A *monotone map* between two posets $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ and $\langle \mathbf{Q}, \leq_{\mathbf{Q}} \rangle$ is a map f that preserves the ordering, in the sense that

$$\frac{p_1 \leq_{\mathbf{P}} p_2}{f(p_1) \leq_{\mathbf{Q}} f(p_2)} \quad (22.1)$$

Remark 22.2. Given a poset \mathbf{P} , the map $\text{id}_{\mathbf{P}}$ is monotone, since for $p_1, p_2 \in \mathbf{P}$, one has:

$$\begin{aligned} p_1 \leq_{\mathbf{P}} p_2 &\Rightarrow p_1 = p_1 \circ \text{id}_{\mathbf{P}} \\ &\leq_{\mathbf{P}} p_2 \circ \text{id}_{\mathbf{P}} \\ &= p_2. \end{aligned}$$

Definition 22.3 (Antitone map). An *antitone map* between two posets $\langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ and $\langle \mathbf{Q}, \leq_{\mathbf{Q}} \rangle$ is a map f that reverses the ordering, in the sense that

$$\frac{p_1 \leq_{\mathbf{P}} p_2}{f(p_1) \geq_{\mathbf{Q}} f(p_2)} \quad (22.2)$$

22.1 Monotone maps	177
22.2 Compositionality of monotonicity	179
22.3 The category Pos of posets and monotone maps	180
Why Pos is not sufficient for de- sign theory	180

Example 22.4 (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 $t : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is a map between the ordered sets \mathbb{N} and $\mathbb{R}_{\geq 0}$, and maps each quantity of widgets to a total manufacturing cost (Fig. 22.2). Clearly, t is a monotone function. Conversely, the unit cost function $u : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$ is antitone (Fig. 22.1).

Figure 22.1.:

Figure 22.2.:

Example 22.5 (Rounding functions). In this example we look at three rounding functions: ceil, floor, and rntte. Both the maps

$$\begin{aligned}\text{ceil} : & \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle \\ & x \mapsto i \in \mathbb{N} : i - 1 < x \leq i,\end{aligned}$$

and

$$\begin{aligned}\text{floor} : & \langle \mathbb{R}, \leq \rangle \rightarrow \langle \mathbb{N}, \leq \rangle \\ & x \mapsto i \in \mathbb{N} : i \leq x < i + 1.\end{aligned}$$

are monotone, since $x \leq y$ implies both $\text{ceil}(x) \leq \text{ceil}(y)$ and $\text{floor}(x) \leq \text{floor}(y)$.

Example 22.6 (Cardinality map). In Example 20.12 we presented the poset arising from the power set of a set $\mathbf{A} = \{a, b, c\}$ and ordered via subset inclusion. The map $|\cdot| : \mathcal{P}\mathbf{A} \rightarrow \mathbb{N}$ (cardinality), is a monotone map (Fig. 22.3).

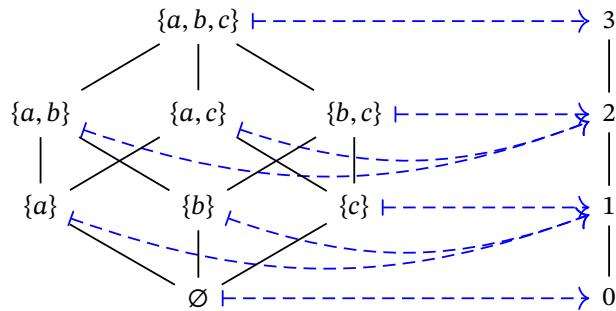


Figure 22.3.: The cardinality map is a monotone map.

Lemma 22.7. Consider a discrete poset \mathbf{P} and a poset \mathbf{Q} . Any map $f : \mathbf{P} \rightarrow \mathbf{Q}$ is monotone.

Proof. Since \mathbf{P} is a discrete poset, one has

$$p_1 \leq_{\mathbf{P}} p_2 \iff p_1 = p_2. \quad (22.3)$$

Therefore, one has

$$\begin{aligned}p_1 \leq_{\mathbf{P}} p_2 \Rightarrow & p_1 = p_2 \\ \Rightarrow & f(p_1) = f(p_2) \\ \Rightarrow & f(p_1) \leq_{\mathbf{Q}} f(p_2).\end{aligned} \quad (22.4)$$

□

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 22.8. 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. 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* (Def. 21.7). The Hasse diagram representing a situation related to this diagram could be as reported in Fig. 22.4.

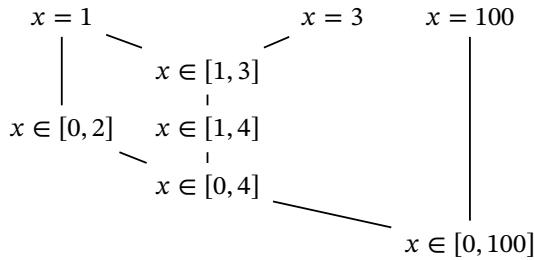


Figure 22.4.

22.2. Compositionality of monotonicity

Note that monotonicity is a compositional property.

Lemma 22.9. Given posets $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ and monotone maps $f : \mathbf{P} \rightarrow \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow \mathbf{R}$, the composite map $f ; g : \mathbf{P} \rightarrow \mathbf{R}$ is monotone as well.

Proof. Consider $p_1, p_2 \in \mathbf{P}$, $q_1, q_2 \in \mathbf{Q}$. We have, by definition,

$$\begin{aligned} p_1 \leq_{\mathbf{P}} p_2 &\Rightarrow f(p_1) \leq_{\mathbf{P}} f(p_2) \\ q_1 \leq_{\mathbf{Q}} q_2 &\Rightarrow g(q_1) \leq_{\mathbf{R}} g(q_2). \end{aligned} \tag{22.5}$$

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

$$p_1 \leq_{\mathbf{P}} p_2 \Rightarrow (f ; g)(p_1) \leq_{\mathbf{R}} (f ; g)(p_2), \tag{22.6}$$

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

22.3. 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 22.10 (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**.

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

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 22.11 (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 22.12 (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 distance \rightarrow velocity \times 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. 22.5.

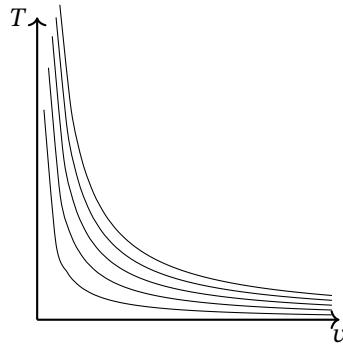


Figure 22.5.: Antichains in $\langle v, T \rangle$ for different values of d .

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 37.1), which will allow to describe such a theory.

23. Solutions to selected exercises

Solution of Exercise 16. Consider the posets $\langle \mathcal{P}^{\text{P}}, \subseteq \rangle$ and $\langle \mathcal{L}^{\text{P}}, \subseteq \rangle$, and let $\mathbf{S}_1, \mathbf{S}_2 \in \mathcal{P}^{\text{P}}$. It is clear that given $\mathbf{S}_1 \subseteq \mathbf{S}_2$, one has

$$\{y \in \mathbf{P} \mid \exists x \in \mathbf{S}_1 : y \leq_{\mathbf{P}} x\} \subseteq \{y \in \mathbf{P} \mid \exists x \in \mathbf{S}_2 : y \leq_{\mathbf{P}} x\}. \quad (23.1)$$

Therefore, $\downarrow \mathbf{S}_1 \subseteq \downarrow \mathbf{S}_2$, satisfying the monotonicity property for \downarrow .

Solution of Exercise 17.

Solution of Exercise 18.

Solution of Exercise 19. We prove the three conditions.

- ▷ First, we know that $\langle p_1, q_1 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_1, q_1 \rangle$, since $p_1 \leq_{\mathbf{P}} p_1$ and $q_1 \leq_{\mathbf{P}} q_1$.
- ▷ Second, $\langle p_1, q_1 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_2, q_2 \rangle$ and $\langle p_2, q_2 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_3, q_3 \rangle$ imply $\langle p_1, q_1 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_3, q_3 \rangle$.
- ▷ Third, if $\langle p_1, q_1 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_2, q_2 \rangle$ and $\langle p_2, q_2 \rangle \leq_{\text{Int}(\mathbf{P})} \langle p_1, q_1 \rangle$, then $p_1 = p_2$ and $q_1 = q_2$.

PART D.COMBINATION



24. Combination	187
25. Sameness	203
26. Universal properties	207
27. Constructing categories	209
28. Solutions to selected exercises	213



24. Combination

24.1. Products

We'll start off by recalling a familiar way of combining two sets, \mathbf{A} and \mathbf{B} .

Definition 24.1 (Cartesian product of sets). Given two sets \mathbf{A}, \mathbf{B} , their *cartesian product* is denoted $\mathbf{A} \times \mathbf{B}$ and defined as

$$\mathbf{A} \times \mathbf{B} = \{\langle x, y \rangle \mid x \in \mathbf{A} \text{ and } y \in \mathbf{B}\}.$$

Example 24.2. Consider the sets $\mathbf{A} = \{1, 2, 3, 4\}$ and $\mathbf{B} = \{\emptyset, \mathbb{P}\}$. We have

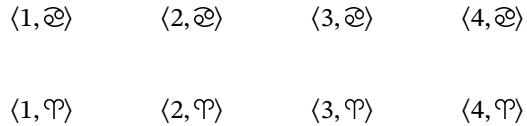
$$\mathbf{A} \times \mathbf{B} = \{\langle 1, \emptyset \rangle, \langle 1, \mathbb{P} \rangle, \langle 2, \emptyset \rangle, \langle 2, \mathbb{P} \rangle, \langle 3, \emptyset \rangle, \langle 3, \mathbb{P} \rangle, \langle 4, \emptyset \rangle, \langle 4, \mathbb{P} \rangle\}. \quad (24.1)$$

We can, however, also represent $\mathbf{A} \times \mathbf{B}$ in a way which highlights its structure more (Example 24.2).

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

$$\pi_1(\langle x, y \rangle) = x \text{ and } \pi_2(\langle x, y \rangle) = y. \quad (24.2)$$

24.1 Products	187
24.2 Coproducts	195
24.3 Other examples	200
Product and coproduct for power set	200
Product and coproduct for logical sequents	201
24.4 Biproducts	201
24.5 Occultism	201



We will often depict the situation like this:

$$\mathbf{A} \xleftarrow{\pi_1} \mathbf{A} \times \mathbf{B} \xrightarrow{\pi_2} \mathbf{B}$$

It turns out that this situation is part of a pattern that unites various different constructions across mathematics. This pattern, of which the cartesian product is a special case, is formalized in the notion of “categorical product”. Before introducing the rigorous definition of the categorical product, let us list a number of examples that are part of the pattern.

Example 24.3. For any $x_1, x_2 \in \mathbb{R}$ let us draw an arrow $x_1 \rightarrow x_2$ iff $x_1 \leq x_2$. Then “taking the minimum” is an example of the categorical product; see Fig. 24.1.

For instance, choosing $x_1 = 3, x_2 = 7$, we have $3 \geq \min\{3, 7\} \leq 7$.

$$x_1 \xleftarrow{\min\{x_1, x_2\}} x_2$$

Figure 24.1.: Taking the minimum

$$m \xleftarrow{\gcd\{m, n\}} n$$

Figure 24.2.: Taking the greatest common divisor

$$\mathbf{S}_1 \xleftarrow{\mathbf{S}_1 \cap \mathbf{S}_2} \mathbf{S}_2$$

Figure 24.3.: Taking the intersection.

$$p_1 \xleftarrow{p_1 \wedge p_2} p_2$$

Figure 24.4.: Taking the conjunction

$$x \xleftarrow{x \wedge y} y$$

Figure 24.5.: Taking the meet

$$X \xleftarrow{\text{“product of } X \text{ and } Y\text{”}} Y$$

Figure 24.6.

Example 24.4. For any $m, n \in \mathbb{N}$ let us draw an arrow $m \rightarrow n$ iff m divides n , which is written $m|n$. Then “taking the greatest common divisor” is an example of the categorical product; see Fig. 24.2.

For instance, choosing $m = 6, n = 9$, we have $\gcd\{6, 9\} = 3$, and $3|6$ and $3|9$.

Example 24.5. Let \mathbf{A} be a set. For any subsets $\mathbf{S}_1, \mathbf{S}_2 \subseteq \mathbf{A}$, let us draw an arrow $\mathbf{S}_1 \rightarrow \mathbf{S}_2$ iff $\mathbf{S}_1 \subseteq \mathbf{S}_2$. Then “taking the intersection” is an example of the categorical product; see Fig. 24.3.

For instance, let $\mathbf{A} = \{1, 2, 3, 4\}$, $\mathbf{S}_1 = \{1, 2, 3\}$, and $\mathbf{S}_2 = \{2, 3, 4\}$. Then $\mathbf{S}_1 \cap \mathbf{S}_2 = \{2, 3\}$, and $\{1, 2, 3\} \supseteq \{2, 3\} \subseteq \{2, 3, 4\}$.

Example 24.6. Let $\mathbf{A} = \{\top, \perp\}$ be the set of logical propositions consisting of true and false. For any propositions $p_1, p_2 \in \mathbf{A}$, let us draw an arrow $p_1 \rightarrow p_2$ iff $p_1 \Rightarrow p_2$. Then “taking the conjunction” (the operation “and”, denoted \wedge) of two statements is an example of the categorical product; see Fig. 24.4.

For instance, let $p_1 = \top, p_2 = \perp$. Then $\top \wedge \perp = \perp$ and $\top \Leftarrow \perp \Rightarrow \perp$ holds.

Example 24.7. Let ... be a lattice. For any element $x, y \in \dots$, let us draw an arrow $x \rightarrow y$ iff $x \leq y$. Then the “meet” (the operation \wedge) of two elements is an example of the categorical product; see Fig. 24.5.

As you can see from the above list of examples, the notion of categorical product involves diagrams of the type in Fig. 24.6.

There is more to the story, however. Loosely speaking, the categorical “product of X and Y ” is characterized by how it interacts, in a certain way, with all other diagrams which have a similar form. Let us explain using an “applied” example involving the cartesian product of sets.

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 $\text{Snacks} = \{\text{apple}, \text{banana}, \text{carrot}\}$ and a drink from $\text{Drinks} =$

$\{\text{apple}, \text{banana}, \text{carrot}\}$. Let **Participants** denote the set of participants. The choice of snacks could be organized as depicted in Fig. 24.7, in which each participant chooses a food, and chooses a drink. This can be described via functions $\text{eats} : \text{Participants} \rightarrow \text{Snacks}$ and $\text{drinks} : \text{Participants} \rightarrow \text{Drinks}$.

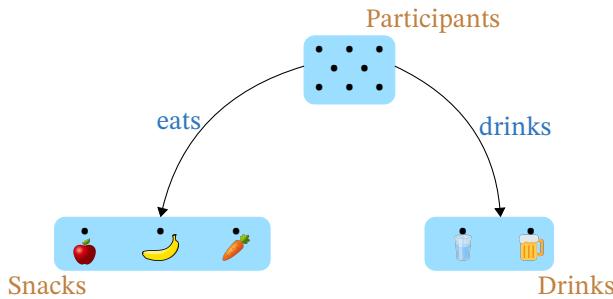


Figure 24.7.: Each participant chooses a food and a drink.

Alternatively, snacks could be pre-packaged in such a way as to allow all possible combinations of food and drink choices. This corresponds to $\text{Snacks} \times \text{Drinks}$. Then the choice participants make of which lunch package they'd like is described by a single function $\text{meal} : \text{Participants} \rightarrow \text{Snacks} \times \text{Drinks}$ (see Fig. 24.8).

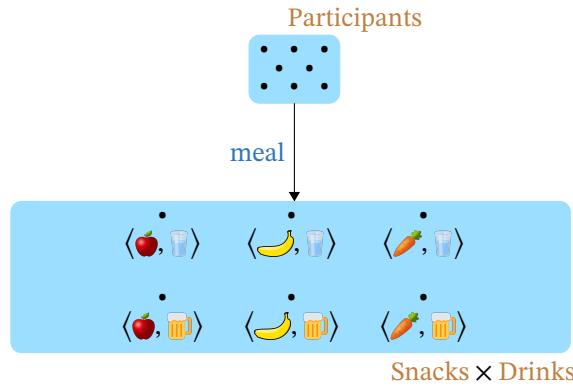


Figure 24.8.: Each participant chooses a combination of food and a drink.

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

$$\phi_{\text{eats,drinks}} : \text{Participants} \rightarrow \text{Snacks} \times \text{Drinks} \quad (24.3)$$

$$p \mapsto \langle \text{eats}(p), \text{drinks}(p) \rangle.$$

Furthermore, given $\phi_{\text{eats,drinks}}$, one can recover eats and drinks :

$$\text{eats} = \phi_{\text{eats,drinks}} \circ \pi_1 \quad \text{and} \quad \text{drinks} = \phi_{\text{eats,drinks}} \circ \pi_2. \quad (24.4)$$

These two equations say that the diagram in Fig. 24.9 is commutative. The whole situation can be summarized thus: given a set **Participants** and functions $\text{eats} : \text{Participants} \rightarrow \text{Snacks}$ and $\text{drinks} : \text{Participants} \rightarrow \text{Drinks}$ as in Fig. 24.8, there is a unique function

$$\phi_{\text{eats,drinks}} : \text{Participants} \rightarrow \text{Snacks} \times \text{Drinks} \quad (24.5)$$

such that the diagram in Fig. 24.9 commutes.

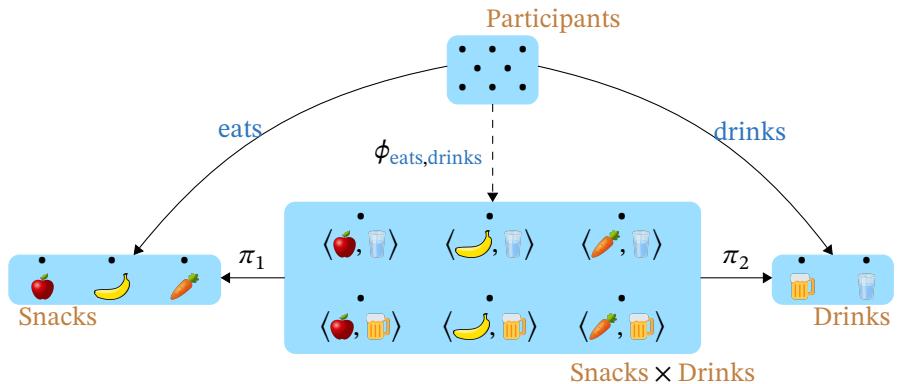


Figure 24.9: Choosing food and drink separately is essentially the same as choosing a combination of the two.

It turns out that this state of affairs *characterizes* the cartesian product **Snacks \times Drinks**. We think of the diagram

$$\text{Snacks} \leftarrow \text{Snacks} \times \text{Drinks} \rightarrow \text{Drinks} \quad (24.6)$$

as “interacting” with the diagram

$$\text{Snacks} \leftarrow \text{Participants} \rightarrow \text{Drinks} \quad (24.7)$$

via the fact that such a map $\phi_{\text{eats},\text{drinks}}$ exists which links the two by making the diagram Fig. 24.9 commute.

This describes the general pattern for the definition of the categorical product. Namely, the categorical product is a diagram of the kind

$$\textcolor{red}{X} \leftarrow \text{“product of } \textcolor{red}{X} \text{ and } \textcolor{red}{Y} \text{”} \rightarrow \textcolor{red}{Y} \quad (24.8)$$

which interacts with any other diagram of the form

$$\textcolor{red}{X} \leftarrow \textcolor{red}{T} \rightarrow \textcolor{red}{Y} \quad (24.9)$$

in a way which is analogous to the situation in Fig. 24.9 (here $\textcolor{red}{T}$ stands for any set that plays the role of the set **Participants**).

Let us now finally state the general definition of categorical product. It is probably helpful to read the definition together with the clarifying remarks that follow it.

Definition 24.8 (Categorical Product). Let \mathbf{C} be a category and let $\textcolor{red}{X}, \textcolor{red}{Y} \in \mathbf{Ob}_{\mathbf{C}}$ be objects. The *product* of $\textcolor{red}{X}$ and $\textcolor{red}{Y}$ is:

Constituents

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

Conditions

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

and $g = (\phi_{f,g}) \circ \pi_2$.

Remark 24.9. Diagrammatically, the condition above states that the diagrams of this form commute:

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

Figure 24.10.

Remark 24.10. 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 $\phi_{f,g}$. Then diagram Fig. 24.10 looks as in Fig. 24.11.

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

Figure 24.11.

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 24.12 illustrates this.

Remark 24.11. 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, for instance, to how, in electrodynamics, a “test charge” is used to probe an electromagnetic field).

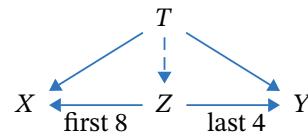
Example 24.12. Suppose that as a manufacturer, you want to label your products with

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

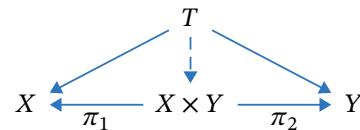
Instead of two separate labels, you can also make one

$$202101155900 \quad (24.10)$$

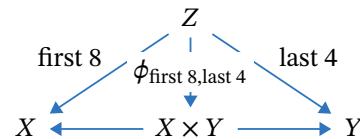
where the first 8 digits represent a date, and the last 4 digits are a model number. Let's call this single label the *product code*. Let Z denote the set of all product codes, and consider the maps $\pi_1 : Z \rightarrow X$, and $\pi_2 : Z \rightarrow Y$ which, respectively, map a 12-digit product code 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 ”.



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 24.10, 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:



Concretely,

$$\phi_{\text{first } 8, \text{last } 4} : Z \rightarrow X \times Y \quad (24.11)$$

maps for instance

$$202101155900 \mapsto \langle 20210115, 5900 \rangle. \quad (24.12)$$

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

Now let us revisit the examples that were given earlier, before we stated the definition of categorical product. The idea is that all of these are instantiations of the categorical product, but in each case, we are working the in context of a different category! In each case, arrows denote morphisms in the category in question, specific to the example.

Example 24.13. This is a continuation of Example 24.3. For any $x_1, x_2 \in \mathbb{R}$, we drew an arrow $x_1 \rightarrow x_2$ iff $x_1 \leq x_2$. The category in question here is the category whose objects are elements of \mathbb{R} , and whose morphisms are inequalities. The product is “taking the minimum”; its universal property is illustrated in Fig. 24.12. It says that if $t \in \mathbb{R}$ is such that $t \leq x_1$ and $t \leq x_2$, then $t \leq \min\{x_1, x_2\}$.

To make things concrete, choose $x_1 = 10$, $x_2 = 18$, and experiment with different choices of t . Verify that everything checks out.

Example 24.14. This is a continuation of Example 24.4. For any $m, n \in \mathbb{N}$, we drew an arrow $m \rightarrow n$ iff m divides n , written $m|n$. The category in question has natural numbers as its objects, and morphisms are given by the relation “divides”. Then product is “taking the greatest common divisor”; its universal property is visualized in Fig. 24.2.

For a concrete example, let $m = 12$ and $n = 18$, so $\gcd\{12, 18\} = 6$. If we take $t = 3$, which divides both 12 and 18, we see that, indeed, 3 also divides $6 = \gcd\{12, 18\}$. And if we take $t = 2$, which also divides both 12 and 18, we see that it is also true that 2 also divides $6 = \gcd\{12, 18\}$.

Example 24.15. This is a continuation of Example 24.5. Given a set \mathbf{A} and arbitrary subsets $S_1, S_2 \subseteq \mathbf{A}$, we drew an arrow $S_1 \rightarrow S_2$ iff $S_1 \subseteq S_2$. The category in question here has as its objects the subsets of \mathbf{A} , and its morphisms are inclusions between them. The product is “taking the intersection”; its universal property is visualized in Fig. 24.3.

As a concrete example, consider again $\mathbf{A} = \{1, 2, 3, 4\}$, $S_1 = \{1, 2, 3\}$, and $S_2 = \{2, 3, 4\}$. So $S_1 \cap S_2 = \{2, 3\}$. If we choose $T = \{2\}$, we see that $T \subseteq S_1$ and $T \subseteq S_2$, and that also $T \subseteq S_1 \cap S_2$ (as it must, according to the universal property). The situation is similar if we choose $T = \{1\}$ or $T = \emptyset$.

The following Lemma describes a general fact that was illustrated in Example 24.15.

Lemma 24.16. Let \mathbf{A} be any set. Its powerset $\mathcal{P}\mathbf{A}$, with the relation of inclusion, is a poset. View this poset as a category (this means there is a single morphism $S_1 \rightarrow S_2$ if and only if $S_1 \subseteq S_2$). For any two objects $S_1, S_2 \in \mathcal{P}\mathbf{A}$, their categorical product exists and is given by $S_1 \cap S_2 \in \mathcal{P}\mathbf{A}$.

Graded exercise 1 (`CatProductPowerset`). Prove Lemma 24.16 by checking that Definition 24.8 is satisfied.

Example 24.17. This is a continuation of Example 24.6. We considered $\mathbf{A} = \{\top, \perp\}$ as a set of logical propositions and for any $p_1, p_2 \in \mathbf{A}$, we drew an arrow $p_1 \rightarrow p_2$ iff $p_1 \Rightarrow p_2$. So, the category at play here has \mathbf{A} as its set of objects, and its morphisms are logical implications. The product is “taking the conjunction” (the logical operation “and”); the universal property is visible in Fig. 24.4.

Example 24.18. This is a continuation of Example 24.18. We considered ... and we drew an arrow $x \rightarrow y$ iff $x \leq y$. So the category at play here is the one corresponding to the poset underlying The categorical product of two elements is their meet (greatest lower bound); the universal property is illustrated in Fig. 24.5.

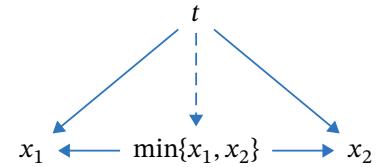


Figure 24.12.: Taking the minimum

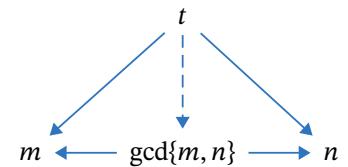


Figure 24.13.: Taking the greatest common divisor

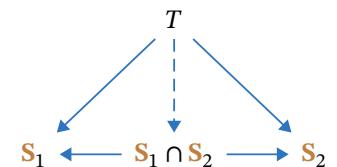


Figure 24.14.: Taking the intersection

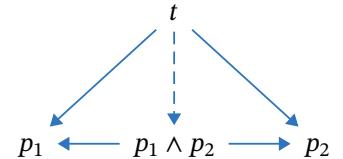


Figure 24.15.: Taking the conjunction

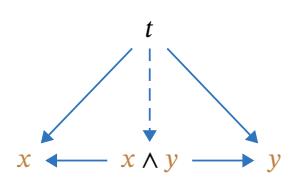


Figure 24.16.: Taking the meet

Example 24.19. 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 Fig. 24.17, 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.



Figure 24.17.: Alternative ways to generate **motion**.

We could also consider building a hybrid vehicle, where we can obtain **motion** from **either gasoline or electric energy** (Fig. 24.18).

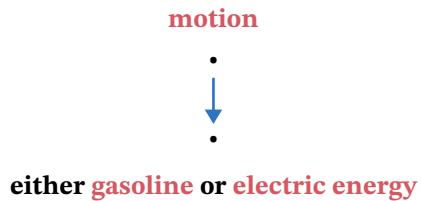


Figure 24.18.: We can generate **motion** from either **gasoline** or **electric energy**.

Definition 24.20 (Cartesian product of categories). Given two categories **C** and **D**, their *cartesian product* **C × D** is the category specified as follows:

1. *Objects*: Objects are pairs $\langle X, Y \rangle$, with $X \in \text{Ob}_C$ and $Y \in \text{Ob}_D$.
2. *Morphisms*: Morphisms are pairs of morphisms $\langle f, g \rangle : \langle X, Z \rangle \rightarrow \langle Y, W \rangle$, with $f : X \rightarrow Y$, $g : Z \rightarrow W$.
3. *Identity morphisms*: Given objects $X \in \text{Ob}_C$ and $Y \in \text{Ob}_D$, the identity morphism on $\langle X, Y \rangle$ is the pair $\langle \text{Id}_X, \text{Id}_Y \rangle$.
4. *Composition of morphisms*: The composition of morphisms is given by composing each component of the pair separately:

$$\langle f, g \rangle ;_{\mathbf{C} \times \mathbf{D}} \langle h, i \rangle = \langle f ;_C h, g ;_D i \rangle. \quad (24.13)$$

Example 24.21. Consider two posets **P**, **Q** as categories. The product poset **P × Q** (Def. 21.1) is the product category of the two posetal categories.

Graded exercise 2. Prove that the product poset $\mathbf{P} \times \mathbf{Q}$ “is” the categorical product of \mathbf{P} and \mathbf{Q} within the category of posets.

Graded exercise 3. Prove that the product category $\mathbf{C} \times \mathbf{D}$ of two small categories “is” the categorical product of \mathbf{C} and \mathbf{D} within the category of small categories.

24.2. Coproducts

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.

Given sets \mathbf{A} and \mathbf{B} , their disjoint union $\mathbf{A} + \mathbf{B}$ is a set that contains a distinct copy of \mathbf{A} and \mathbf{B} each. If an element is contained in both \mathbf{A} and \mathbf{B} , then there will be two distinct copies of it in the disjoint union $\mathbf{A} + \mathbf{B}$.

Definition 24.22 (Disjoint union of sets). The *disjoint union* (or sum) of sets \mathbf{A} and \mathbf{B} is

$$\mathbf{A} + \mathbf{B} = \{\langle 1, x \rangle \mid x \in \mathbf{A}\} \cup \{\langle 2, y \rangle \mid y \in \mathbf{B}\}. \quad (24.14)$$

Example 24.23. Consider the sets $\mathbf{A} = \{\star, \diamond\}$ and $\mathbf{B} = \{*, †\}$. Their disjoint union can be represented as in Fig. 24.19.

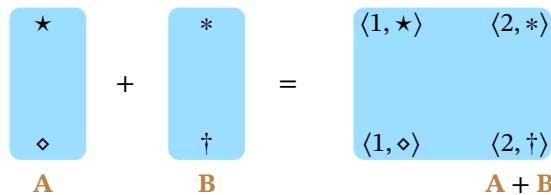
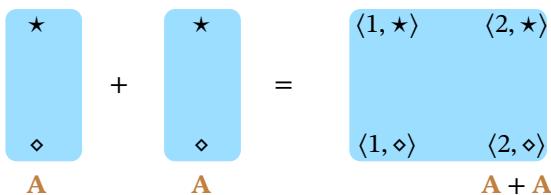


Figure 24.19.: Example of a disjoint union of sets.

We can define the disjoint union of a set with itself; this corresponds to having two distinct copies of the set (Fig. 24.20).



In the case of the cartesian product of two sets we had projection maps, as in [REF]. For the disjoint union of sets, we have instead *inclusion maps*. Thus we have a diagram of this form:

$$\mathbf{A} \xrightarrow{\iota_1} \mathbf{A} + \mathbf{B} \xleftarrow{\iota_2} \mathbf{B}$$

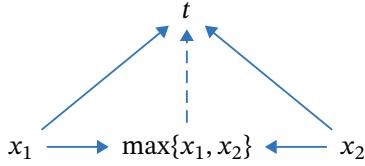


Figure 24.21.: Taking the minimum

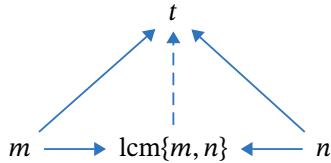


Figure 24.22.: Taking the least common multiple

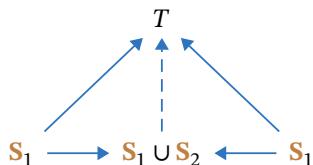


Figure 24.23.: Taking the union

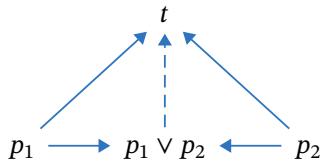


Figure 24.24.: Taking the disjunction

Example 24.24. This example is “dual” to Example 24.13. The category in question is the same one: objects are elements of \mathbb{R} and morphisms are inequalities. The coproduct is “taking the maximum”; its universal property is illustrated in Fig. 24.12. It says that if $t \in \mathbb{R}$ is such that $t \geq x_1$ and $t \geq x_2$, then $t \geq \min\{x_1, x_2\}$.

Example 24.25. This example is “dual” to Example 24.14. The category we’re working in has natural numbers as its objects, and morphisms are given by the relation “divides”. The coproduct is “taking the least common multiple”; its universal property is visualized in Fig. 24.2.

Example 24.26. This example is “dual” to Example 24.15. Given a set \mathbf{A} and arbitrary subsets $\mathbf{S}_1, \mathbf{S}_2 \subseteq \mathbf{A}$, we drew an arrow $\mathbf{S}_1 \rightarrow \mathbf{S}_2$ iff $\mathbf{S}_1 \subseteq \mathbf{S}_2$. The category in question here has, as its objects, the subsets of \mathbf{A} , and its morphisms are inclusions between them. The product is “taking the intersection”; its universal property is visualized in Fig. 24.3.

As a concrete example, consider again $\mathbf{A} = \{1, 2, 3, 4\}$, $\mathbf{S}_1 = \{1, 2, 3\}$, and $\mathbf{S}_2 = \{2, 3, 4\}$. So $\mathbf{S}_1 \cap \mathbf{S}_2 = \{2, 3\}$. If we choose $T = \{2\}$, we see that $T \subseteq \mathbf{S}_1$ and $T \subseteq \mathbf{S}_2$, and that also $T \subseteq \mathbf{S}_1 \cap \mathbf{S}_2$ (as it must, according to the universal property). The situation is similar if we choose $T = \{1\}$ or $T = \emptyset$.

Example 24.27. This example is “dual” to Example 24.15. Again consider the set $\mathbf{A} = \{\top, \perp\}$ of logical propositions and for any $p_1, p_2 \in \mathbf{A}$, we drew an arrow $p_1 \rightarrow p_2$ iff $p_1 \Rightarrow p_2$. The category we are working with has \mathbf{A} as its set of objects, and its morphisms are logical implications. The coproduct is “taking the disjunction” (the logical operation “or”); the universal property is shown in Fig. 24.4.

Example 24.28. This example is “dual” to Example 24.18. We considered ... and we drew an arrow $x \rightarrow y$ iff $x \leq y$. The category at play here is the one corresponding to the poset underlying The categorical product of two elements is their join (least upper bound); the universal property is illustrated in Fig. 24.5.

As you can see from the above list of examples, the notion of coproduct involves diagrams of the type in Fig. 24.26.

$$X \longrightarrow \text{“coproduct of } X \text{ and } Y \text{”} \longleftarrow Y$$

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!

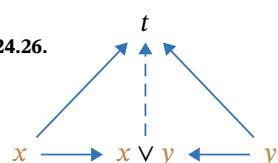


Figure 24.25.: Taking the join

Definition 24.29 (Coproduct). Let \mathbf{C} be a category and let $X, Y \in \text{Ob}_{\mathbf{C}}$ be objects. The *coproduct* of X and Y is:

Constituents

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

Conditions

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 24.30. Diagrammatically, the condition above states that diagrams of this form commute: Similarly as was the case with the categorical product, “the

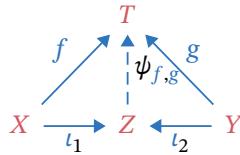


Figure 24.27.

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 $\psi_{f,g}$. The diagram in Fig. 24.27 then looks as in Fig. 24.28

Example 24.31. Let’s consider two battery producers, each producing specific battery technologies. The first company produces a set

$$\mathbf{A} = \{\text{LiPo, LCO, NiH2}\} \quad (24.15)$$

of technologies, and the second one a set

$$\mathbf{B} = \{\text{LFP, LMO, LiPo}\}. \quad (24.16)$$

Each technology has, for a specific desired battery mass, a specific price, belonging to a set of prices

$$\mathbf{P} = \{50, 60, 70, 80\} \times \text{CHF}. \quad (24.17)$$

We specify the price mappings for different technologies via the functions $f : \mathbf{A} \rightarrow \mathbf{P}$ and $g : \mathbf{B} \rightarrow \mathbf{P}$. A battery vendor wants to sell batteries from both producers and wants to create a battery catalogue, which needs to take into account which

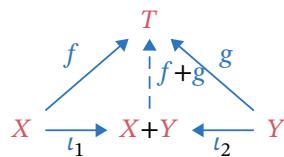


Figure 24.28.

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:

$$\mathbf{A} + \mathbf{B} = \{\langle 1, \text{LiPo} \rangle, \langle 1, \text{LCO} \rangle, \langle 1, \text{NiH2} \rangle, \langle 2, \text{LFP} \rangle, \langle 2, \text{LMO} \rangle, \langle 2, \text{LiPo} \rangle\}. \quad (24.18)$$

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

$$\begin{aligned} \iota_{\mathbf{A}} : \mathbf{A} &\rightarrow \mathbf{A} + \mathbf{B} \\ a &\mapsto \langle 1, a \rangle \end{aligned} \quad (24.19)$$

$$\begin{aligned} \iota_{\mathbf{B}} : \mathbf{B} &\rightarrow \mathbf{A} + \mathbf{B} \\ b &\mapsto \langle 2, b \rangle. \end{aligned} \quad (24.20)$$

This situation is graphically represented in Fig. 24.29, and mimics the coproduct diagram presented in Definition 24.29.

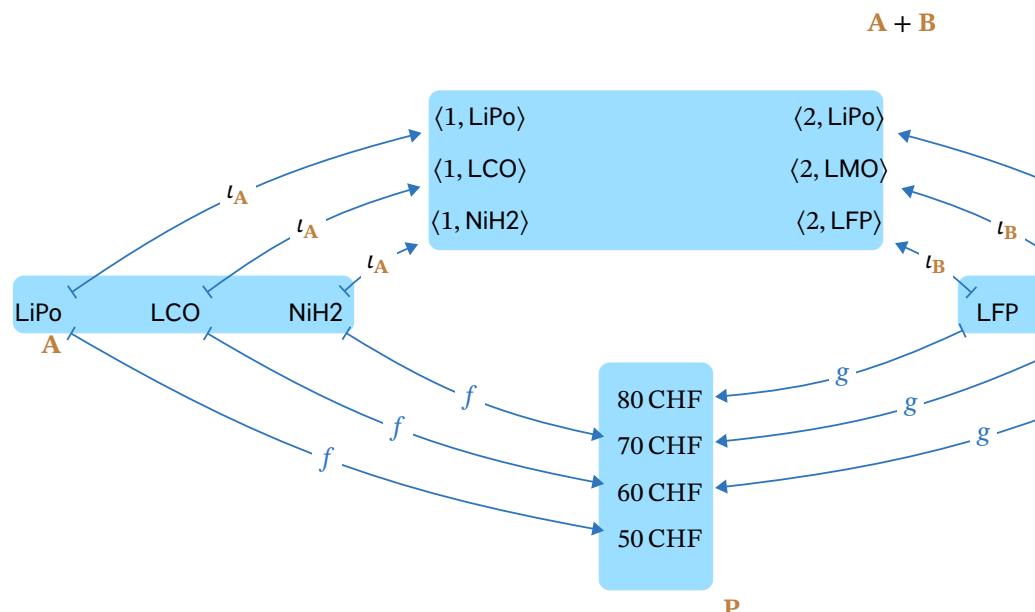


Figure 24.29.: Battery technologies, companies, prices, and a catalogue.

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

$$\iota_{\mathbf{A}} \circ (f+g) = f \text{ and } \iota_{\mathbf{B}} \circ (f+g) = g. \quad (24.21)$$

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

$$\text{either } \exists a \in \mathbf{A} : x = \iota_{\mathbf{A}}(a) \text{ or } \exists b \in \mathbf{B} : x = \iota_{\mathbf{B}}(b). \quad (24.22)$$

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

$$\begin{aligned} f+g : \mathbf{A} + \mathbf{B} &\rightarrow \mathbf{P} \\ x &\mapsto \begin{cases} f(x), & \text{if } x = \iota_{\mathbf{A}}(a), \quad a \in \mathbf{A}, \\ g(x), & \text{if } x = \iota_{\mathbf{B}}(b), \quad b \in \mathbf{B}. \end{cases} \end{aligned} \quad (24.23)$$

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 $\mathbf{A} \cup \mathbf{B}$ (Fig. 24.30). 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 $\text{LiPo} \in \mathbf{A} \cup \mathbf{B}$, we have no notion of the company which produces it, and we are therefore unsure whether to assign it to $f(\text{LiPo}) = 50 \text{ CHF}$ or $g(\text{LiPo}) = 60 \text{ CHF}$. Indeed, the unique map $f+g$ required by the universal property of the coproduct cannot exist, since in case $\mathbf{A} \cap \mathbf{B} \neq \emptyset$, any element $x \in \mathbf{A} \cap \mathbf{B}$ should be simultaneously sent to $f(x)$ and $g(x)$.

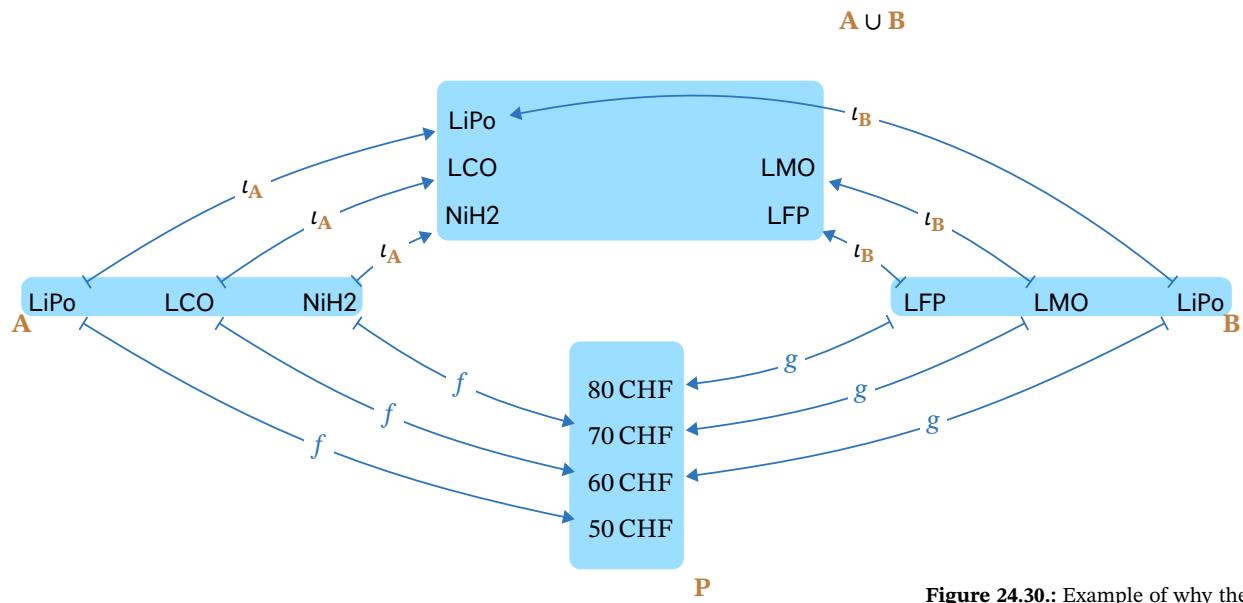


Figure 24.30.: Example of why the union is not the coproduct in **Set**.

Example 24.32. Given $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\text{Rel}}$ (so \mathbf{A} and \mathbf{B} are sets) their coproduct is the disjoint union $\mathbf{A} + \mathbf{B}$. The disjoint union of sets comes equipped with inclusion functions $i_{\mathbf{A}} : \mathbf{A} \rightarrow \mathbf{A} + \mathbf{B}$ and $i_{\mathbf{B}} : \mathbf{B} \rightarrow \mathbf{A} + \mathbf{B}$. If we turn these functions into relations

$$R_{i_{\mathbf{A}}} \subseteq \mathbf{A} \times (\mathbf{A} + \mathbf{B})$$

$$R_{i_{\mathbf{B}}} \subseteq \mathbf{B} \times (\mathbf{A} + \mathbf{B}).$$

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 might be expected.

Example 24.33. 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 24.34. Let’s consider the ordered set (\mathbb{R}, \leq) , 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*.

Example 24.35. Let S be a set, and $A, B \subseteq S$ subsets. We can draw an arrow $A \rightarrow B$ if $A \subseteq B$. By following the coproduct's commutative diagram, it is easy to see that the coproduct of A and B is given by $A \cup B$: the “smallest” set containing both A and B .

Definition 24.36 (Disjoint union category). Given two categories C and D , their *disjoint union* $C + D$ is the category specified as follows:

1. *Objects*: Objects are elements of $\text{Ob}_C + \text{Ob}_D$; that is, objects are tuples of the form $\langle X, i \rangle$, with $i = 1$ or $i = 2$, depending on whether $X \in \text{Ob}_C$ or $X \in \text{Ob}_D$.
2. *Morphisms*: Given objects $\langle X, i \rangle, \langle Y, j \rangle \in \text{Ob}_{C+D}$,

$$\text{Hom}_{C+D}(\langle X, i \rangle, \langle Y, j \rangle) := \begin{cases} \text{Hom}_C(X, Y) & \text{if } i = j = 1, \\ \text{Hom}_D(X, Y) & \text{if } i = j = 2, \\ \emptyset & \text{else.} \end{cases} \quad (24.24)$$

3. *Identity morphisms*:
4. *Composition of morphisms*:

24.3. Other examples

Product and coproduct for power set

Example 24.5 and Example 24.35 are specific instances of the power set lattice.

Definition 24.37 (Power set as 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:

$$A \leq B := A \subseteq B;$$

- ▷ The join is given by the union of sets:

$$A \vee B := A \cup B;$$

- ▷ The meet is given by the intersection of sets:

$$A \wedge B := A \cap B;$$

- ▷ The top element is the set S itself:

$$T = S;$$

- ▷ The bottom element is the empty set:

$$\perp = \emptyset.$$

The Hasse diagram reported in Fig. 24.31 illustrates the structure of the power set lattice for three sets $A, B, C \in \mathcal{P}S$.

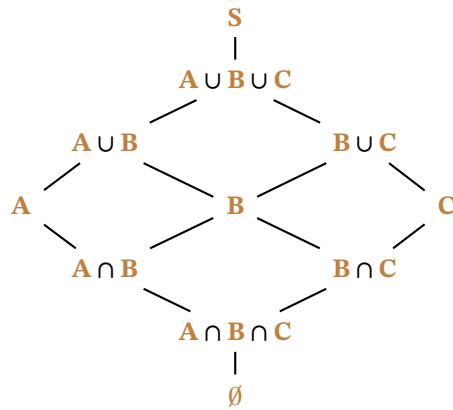


Figure 24.31.

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 $i_1 : A \rightarrow A \cup B$ and $i_2 : B \rightarrow A \cup B$ simply state the inclusion of A, B in $A \cup B$.

Product and coproduct for logical sequents

24.4. Biproducts

24.5. Occultism



25. Sameness

25.1 Sameness in category theory . . .	203
25.2 Isomorphism is not identity . . .	206

25.1. Sameness in category theory

One nice thing about the category of sets is that we are all used to working with sets and functions. And many concepts that are familiar in the setting of sets and functions can actually be reformulated in a way which makes sense for lots of other categories, if not for all categories. It can be fun, and insightful, to see known definitions transformed into “category theory language”. For example: the notion of a bijective function is a familiar concept. There are least two ways of saying what it means for a function $f : \mathbf{A} \rightarrow \mathbf{B}$ of sets to be bijective:

Definition 1: “ $f : \mathbf{A} \rightarrow \mathbf{B}$ is bijective if, for every $y \in \mathbf{B}$ there exists precisely one $x \in \mathbf{A}$ such that $f(x) = y$;

Definition 2: “ $f : \mathbf{A} \rightarrow \mathbf{B}$ is bijective if there exists a function $g : \mathbf{B} \rightarrow \mathbf{A}$ such that $f \circ g = \text{Id}_{\mathbf{A}}$ and $g \circ f = \text{Id}_{\mathbf{B}}$ ”.

It is a short proof to show that the above two definitions are equivalent. The first definition, however, does not lend itself well to generalization in category theory, because it is formulated using something that is very specific to sets: namely, it refers to *elements* of the sets \mathbf{A} and \mathbf{B} . And we have seen that the objects of a category need not be sets, and so in general we cannot speak of “elements” in the usual sense. Definition 2, on the other hand, can easily be generalized to

work in any category. To formulate this version, all we need are morphisms, their composition, the notion of identity morphisms, and the notion of equality of morphisms (for equations such as “ $f \circ g = \text{Id}_x$ ”). The generalization we obtain is the fundamental notion of an “isomorphism”.

Definition 25.1 (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 25.2. 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 21. In the previous remark we wrote *the* inverse. We do this because inverses are in fact unique. Can you prove this? That is, show that if $f : X \rightarrow Y$ is an isomorphism, and if $g_1 : Y \rightarrow X$ and $g_2 : Y \rightarrow X$ are morphisms such that $f \circ g_1 = \text{Id}_X$ and $g_1 \circ f = \text{Id}_Y$, and $f \circ g_2 = \text{Id}_X$ and $g_2 \circ f = \text{Id}_Y$, then necessarily $g_1 = g_2$.

See solution on page 213.

Definition 25.3 (Isomorphic objects). Let X, Y be two objects in a category. We say that X and Y are **isomorphic** if there exists an isomorphism $X \rightarrow Y$ or $Y \rightarrow X$.

For the formulation of the definition of “isomorphic”, mathematicians might often only require the existence of an isomorphism $X \rightarrow Y$, say, since by remark above we know there is then necessarily also an isomorphism in the opposing direction, namely the inverse. We choose here the longer, perhaps more cumbersome formulation just to emphasize the symmetry of the term “isomorphic”. Also note that the definition leaves unspecified whether there might be just one or perhaps many isomorphisms $X \rightarrow Y$.

When two objects are isomorphic, in some contexts we will want to think of them as “the same”, and in some contexts we will want to keep track of more information. In fact, in category theory, it is typical to think in terms of different kinds of “sameness”. To give a sense of this, let’s look at some examples using sets.

Example 25.4 (Semantic coherence). Suppose Francesca and Gabriel want to share a dish at a restaurant. Francesca only speaks Italian, and Gabriel only speaks German. Let M denote the set of dishes on the menu. For each dish, Francesca can say if she is willing to eat it, or not. This can be modeled by a function $f : M \rightarrow \{\text{Si}, \text{No}\}$ 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 $g : M \rightarrow \{\text{Ja}, \text{Nein}\}$. 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, such as how they are being used in interaction with other mathematical constructs.

Example 25.5 (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 25.6 (Relabelling). Consider the little catalogue in Table 15.1. 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 25.7. 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 25.8. In **FinSet**, isomorphisms from a set to itself are automorphisms, and correspond to *permutations* of the set. Assuming a cardinality of n for the set (for instance, the set has n elements), the number of isomorphisms is given by the number of ways in which one can “rearrange” n elements of the set, which is $n!$.

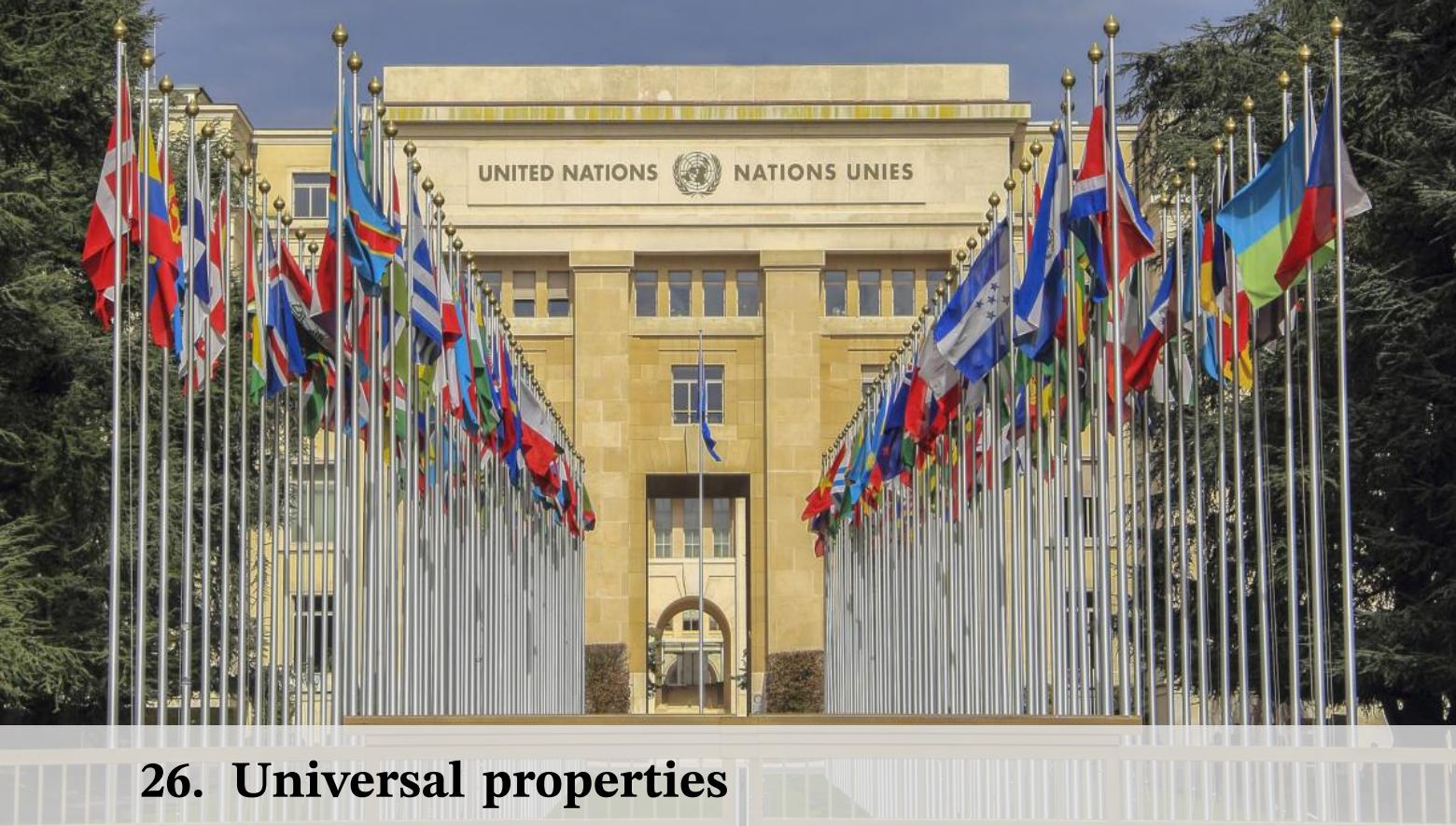
Example 25.9. In **Set**, isomorphisms between $\mathbb{R} \rightarrow \mathbb{R}$ correspond to invertible functions.

25.2. Isomorphism is not identity

Example 25.10. Let’s consider currencies, and in particular the sets $\mathbb{R} \times \{\text{B}\}$ and $\mathbb{R} \times \{\text{B cents}\}$. 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 ₣ and 1,000 ₣ cents are different elements of different sets, but there exists an isomorphism between the two. For one direction, the isomorphism transforms ₣ into ₣ cents (multiplying the real number by 100); the other direction transforms ₣ cents into ₣ (dividing the real number by 100).

Invertible functions are isomorphisms

isomorphisms in 2D



26. Universal properties

26.1 Universal properties 207

26.1. Universal properties

Definition 26.1 (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 $\mathbf{Hom}_{\mathbf{C}}(A, B)$ has exactly one element. We say that A is a *terminal object* if, for all $D \in \mathbf{C}$, the hom-set $\mathbf{Hom}_{\mathbf{C}}(D, A)$ has exactly one element.



27. Constructing categories

In this chapter we discuss various ways of building new categories from old ones. Given categories \mathbf{C} and \mathbf{D} , we saw in Definition 24.20 and Definition 24.36 how to build their cartesian product $\mathbf{C} \times \mathbf{D}$ and their direct sum $\mathbf{C} + \mathbf{D}$, respectively. In addition to these constructions, we will now see how to start with a category \mathbf{C} and construct its *opposite category* \mathbf{C}^{op} , as well as various other categories derived from \mathbf{C} by thinking of kinds of diagrams in \mathbf{C} as objects themselves.

27.1 Opposite Category	209
27.2 Generating categories from graphs	210
27.3 Intervals as categories	211
Twisted arrow category	211
Arrow category	211
27.4 Arrow construction	211
27.5 Slice construction	211

27.1. Opposite Category

Definition 27.1 (Opposite category). Given a category \mathbf{C} , its *opposite category* \mathbf{C}^{op} is specified by:

1. *Objects*: $\text{Ob}_{\mathbf{C}^{\text{op}}} = \text{Ob}_{\mathbf{C}}$.

Given $X \in \text{Ob}_{\mathbf{C}}$, we will sometimes (though not always) write X^{op} to signify when we are thinking of X as an object of $\text{Ob}_{\mathbf{C}}^{\text{op}}$.

2. *Morphisms*: Given objects $X^{\text{op}}, Y^{\text{op}} \in \text{Ob}_{\mathbf{C}^{\text{op}}} = \text{Ob}_{\mathbf{C}}$,

$$\text{Hom}_{\mathbf{C}^{\text{op}}}(X^{\text{op}}, Y^{\text{op}}) := \text{Hom}_{\mathbf{C}}(Y, X). \quad (27.1)$$

Given $f \in \text{Hom}_C(Y; X)$, when we are thinking of it as an element of $\text{Hom}_{C^{\text{op}}}(X^{\text{op}}; Y^{\text{op}})$, we will sometimes write f^{op} .

3. *Identity morphisms:* Given $X^{\text{op}} \in \text{Ob}_{C^{\text{op}}}$, its identity morphism is

$$\text{Id}_{X^{\text{op}}} := \text{Id}_X^{\text{op}}. \quad (27.2)$$

4. *Composition:* Let $f^{\text{op}} \in \text{Hom}_{C^{\text{op}}}(X^{\text{op}}; Y^{\text{op}})$ and $g^{\text{op}} \in \text{Hom}_{C^{\text{op}}}(Y^{\text{op}}; Z^{\text{op}})$, then

$$f^{\text{op}} \circ_{C^{\text{op}}} g^{\text{op}} := (g \circ_C f)^{\text{op}}. \quad (27.3)$$

Graded exercise 4. Verify that Definition 27.1 defines a category. In other words, check that its constituents satisfy the conditions of associative and unitality.

27.2. Generating categories from graphs

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

Definition 27.2 (Free category on a graph). Let $G = (V, A, s, t)$ be a graph. The *free category on G*, denoted $\text{Free}(G)$, has as objects the vertices V of G , and given vertices $x \in V$ and $y \in V$, the morphisms $\text{Free}(G)(x, y)$ 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 .

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

Exercise 22. Consider the following five graphs. For each graph G , how many morphisms in total are there in the associated category $\text{Free}(G)$?



See solution on page 213.

27.3. Intervals as categories

Twisted arrow category

Definition 27.3 (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 $f : X \rightarrow Y, g : Z \rightarrow W$ is given by a pair of arrows $\langle h, i \rangle$ such that the following diagram commutes:

$$\begin{array}{ccc} X & \xleftarrow{h} & Z \\ f \downarrow & & \downarrow g \\ Y & \xrightarrow{i} & W \end{array}$$

3. *Identity morphisms*:
4. *Composition*:

Graded exercise 5. Prove that Definition 27.3 does define a category.

Example 27.4 (Intervals). Consider a poset \mathbf{P} . The twisted arrow category $\text{Tw}(\mathbf{P})$ is isomorphic to the poset (viewed as a category) of nonempty *intervals* in \mathbf{P} . Recall that, given elements $a, b \in \mathbf{P}$, the interval $[a, b]$ is

$$[a, b] := \{p \in \mathbf{P} \mid a \leq_{\mathbf{P}} p \leq_{\mathbf{P}} b\}.$$

Exercise 23. Prove the statement in Example 27.4.

See solution on page ??.

Remark 27.5. Recall Section 29.3 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

Arrow category

27.4. Arrow construction

27.5. Slice construction

28. Solutions to selected exercises

Solution of Exercise 21.

Solution of Exercise 22.

PART E. THICKER ARROWS



29. Functors	217
30. Specialization	223
31. Up the ladder	229
32. Naturality	233
33. Adjunctions	237
34. Solutions to selected exercises	247



29. Functors

29.1. Modeling translations

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

29.1 Modeling translations	217
29.2 Functors	218
29.3 A poset as a category	219
Monotone maps are functors . . .	220
29.4 Other examples of functors	220
The list functor	220
Planning as the search of a func- tor	220
29.5 Categorical Databases	221

29.2. Functors

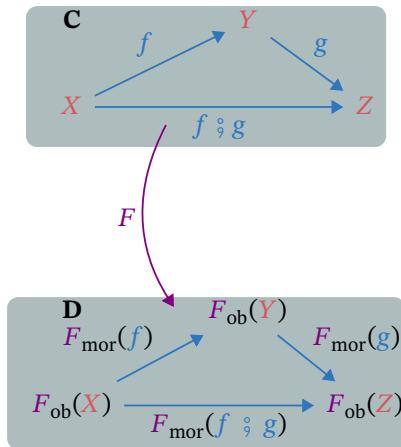


Figure 29.1.: Commuting diagram for semi-functors

Definition 29.1 (Semi-functor). Given two semi-categories **C** and **D**, a *semi-functor* $F : \mathbf{C} \rightarrow \mathbf{D}$ from **C** to **D** is defined by the following data and conditions.

Data:

- i) A map

$$F_{\text{ob}} : \text{Ob}_{\mathbf{C}} \rightarrow \text{Ob}_{\mathbf{D}}. \quad (29.1)$$

- ii) For every pair of objects X, Y of **C** a map

$$F_{\text{mor}} : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{D}}(F_{\text{ob}}(X); F_{\text{ob}}(Y)) \quad (29.2)$$

Conditions:

- 1. It holds that

$$\frac{f : X \rightarrow_{\mathbf{C}} Y \quad g : Y \rightarrow_{\mathbf{C}} Z}{F_{\text{mor}}(f \circ g) = F_{\text{mor}}(f) \circ F_{\text{mor}}(g)}. \quad (29.3)$$

This situation is graphically reported in Fig. 29.1.

It is common to overload the notation and use F to mean both F_{ob} and F_{mor} . The diagram with this notation overload is in Fig. 29.2.

Semi-functors are a generalization of the various semigroup morphisms that we saw in the previous chapter.

In particular, they are a generalization of semi-category actions (Definition 17.2), which we can re-define as follows.

Definition 29.2 (Semi-category actions, redefined). A semi-category action of **C** is a semi-functor $F : \mathbf{C} \rightarrow \mathbf{Set}$.

Definition 29.3 (Embedding functor). A functor $F : \mathbf{C} \rightarrow \mathbf{D}$ is an *embedding* if F_{ob} and F_{mor} are injective.

For categories we have the stronger concept of *functors*. Categories have identities, and functors are required to preserve the identities.

Definition 29.4 (Functor). A functor from category **C** to category **D** is a semi-functor $F : \mathbf{C} \rightarrow \mathbf{D}$ that satisfies the condition

$$F_{\text{mor}}(\text{Id}_X) = \text{Id}_{F_{\text{ob}}(X)}. \quad (29.4)$$

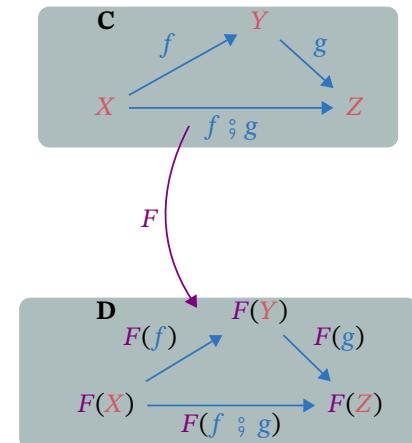


Figure 29.2.: Commuting diagram for semi-functors, overloading the notation.

Remark 29.5. A functor from a category to itself is called an *endofunctor*.

Example 29.6. Let \mathbf{C} be the category whose objects are all real vector spaces and whose morphisms are \mathbb{R} -linear maps. Composition is the usual composition of linear maps. There is an endofunctor $F : \mathbf{C} \rightarrow \mathbf{C}$ whose action on objects is

$$F(V) = V^{**}. \quad (29.5)$$

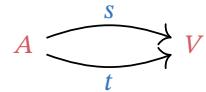
(Recall that $V^{**} = \{\text{linear maps } V^* \rightarrow \mathbb{R}\} = \text{Hom}_{\mathbf{C}}(V^*; \mathbb{R})$). The action of F on morphisms is as follows. Given a linear map $f : V \rightarrow W$,

$$F(f) : V^{**} \rightarrow W^{**}, \quad \xi \mapsto [l \mapsto \xi(f \circ l)]. \quad (29.6)$$

Graded exercise 1 (DoubleDualFunctor). Prove that F as defined in Example 29.6 is in fact a functor.

Graded exercise 2 (GraphsViaFunctors). Consider the following category, which has two objects, V and A , and four morphisms: besides the identity morphisms, there are two morphisms, s and t , from A to V . See Fig. 29.3. Call this category \mathbf{G} .

Can you explain the following statement? “Specifying a functor $\mathbf{G} \rightarrow \mathbf{Set}$ is the ‘same thing’ as specifying a directed graph”.



Graded exercise 3 (UpperSetsViaFunctors). Recall that **Bool** denotes the category with two objects, \top and \perp , and with precisely one non-identity morphism which goes from \perp to \top . Let \mathbf{P} be a poset. View it as a category \mathbf{P} , and let $F : \mathbf{P} \rightarrow \mathbf{Bool}$ be a functor. In other words, $F = F_{\text{ob}}$ is a monotone function. Prove that the set

$$\mathbf{S} := \{p \in \mathbf{P} \mid F(p) = \top\} \subseteq \mathbf{P} \quad (29.7)$$

is an upper set.

Figure 29.3.

29.3. A poset as a category

A single poset $\langle \mathbf{P}, \leq \rangle$ can be described as a category, in which each point is an object, and there is a morphism between two objects x and y if and only if $x \leq_{\mathbf{P}} y$. This is a “thin” category, which means that there is at most one morphism between two objects. For any $x, y \in \mathbf{P}$, there exist only one relation $x \leq_{\mathbf{P}} y$ in \mathbf{P} (Def. 20.2). The identity morphism is given by the reflexivity property of posets: for any $x \in \mathbf{P}$, we have $x \leq_{\mathbf{P}} x$. Furthermore, composition is given by the transitivity property of posets: for $x, y, z \in \mathbf{P}$, $x \leq_{\mathbf{P}} y$ and $y \leq_{\mathbf{P}} z$ implies $x \leq_{\mathbf{P}} z$.

Example 29.7. Let's revisit Example 20.12, in which we had a poset $\mathcal{P}(\{a, b, c\})$ with order given by inclusion (Fig. 29.4).

This is a category \mathbf{C} , with $\text{Ob}_{\mathbf{C}} = \mathcal{P}(\{a, b, c\})$, 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 $\{a\} \subseteq \{a, b, c\}$.

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 29.8. A monotone map F between posets \mathbf{P}, \mathbf{Q} is a functor between the “posetal categories” \mathbf{P} and \mathbf{Q} .

Proof. We start by specifying the functor F and two posets \mathbf{P} and \mathbf{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 $x \in \mathbf{P}$ to $F(x) \in \mathbf{Q}$, and it guarantees that for every $x, y \in \mathbf{P}$, if $x \leq_{\mathbf{P}} y$ then $F(x) \leq_{\mathbf{Q}} F(y)$. We now need to check the two conditions that a functor must satisfy. First, consider the identity morphism for $x \in \mathbf{P}$, namely $x \leq_{\mathbf{P}} x$. The application of the map F results in the condition $F(x) \leq_{\mathbf{Q}} F(x)$, which is the identity morphism on \mathbf{Q} . Second, morphisms $x \leq_{\mathbf{P}} y$ and $y \leq_{\mathbf{P}} z$ in \mathbf{P} , by applying the map F to the morphism composition $x \leq_{\mathbf{P}} z$ one obtains $F(x) \leq_{\mathbf{Q}} F(z)$, which is the same as the composition of $F(x) \leq_{\mathbf{Q}} F(y)$ and $F(y) \leq_{\mathbf{Q}} F(z)$. \square

29.4. Other examples of functors

The list functor

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

Planning as the search of a functor

Example 29.9. Recall the category **Berg** introduced in Section 14.2 and define a category **Plans** where objects are specific areas of the mountain and morphisms describing visiting order constraints, illustrated in Fig. 29.5. For instance, there

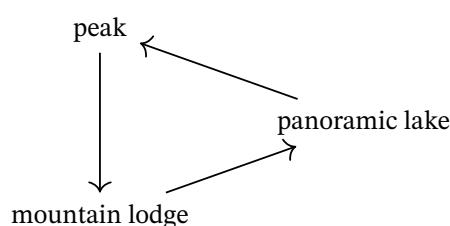


Figure 29.5.: Example of visiting order constraints on a mountain.

is a morphism from “mountain lodge” to “panoramic lake”, which describes the plan of going from the lodge area to the lake area. We call such morphisms *plans*. Plans can be composed via concatenation. For instance, given a plan to go from “mountain lodge” to “panoramic lake”, and a plan to go from “panoramic lake 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 $\langle p, v \rangle$, where p represent coordinates of a specific location and $v \in \mathbb{R}^3$ represents velocities. Morphisms in **Berg** are paths that connect locations. For the sake of our planning, we can identify areas of the mountain as sets of locations. Such areas are, for instance, the “mountain lodge”, “panoramic lake”, and the “peak” (note that the “peak” represents an area corresponding to a single location). Given some plans as in Fig. 29.5, 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. 29.6.

Figure 29.6.:

29.5. Categorical Databases

In this section we look at how a relational database can be modeled using the notion of functors. This view of databases is due to Spivak [31, 32].

To model a database, we need to model two things:

1. The *schema* of the database is its structure. This includes which tables are present, what fields are included in each table, and what constraints there are among tables.
2. The *data* that resides in the database.

Spivak’s intuition is that categories can be used for modeling the schema, not the data. The data can be modeled as a *functor*.



30. Specialization

30.1. Notion of subcategory

30.1 Notion of subcategory	223
30.2 Drawings	224
30.3 Other examples of subcategories in engineering	225
30.4 Subcategories of Berg	226
30.5 Embeddings	227

Definition 30.1 (Subcategory). A *subcategory* **D** of a category **C** is a category for which:

1. All the objects in Ob_D are in Ob_C ;
2. For any objects $X, Y \in \text{Ob}_D$, $\text{Hom}_D(X; Y) \subseteq \text{Hom}_C(X; Y)$;
3. If $X \in \text{Ob}_D$, then $\text{Id}_X \in \text{Hom}_C(X; X)$ is in $\text{Hom}_D(X; X)$ 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 30.2 (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

The *Pontifical Swiss Guard* is an armed forces that protects the Pope and the Apostolic Palace, serving as the military of Vatican City. Recruits to the guards must be Catholic, single males with Swiss citizenship, who have completed basic training with the Swiss Armed Forces.

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 30.3 (Set and Rel). The category Set is a subcategory of Rel . To show this, we need to prove the conditions presented in Definition 30.1.

1. In both Rel and Set , the collection of objects is all sets.
2. Given $X, Y \in \text{Ob}_{\text{Set}}$, we know that $\text{Hom}_{\text{Set}}(X; Y) \subseteq \text{Hom}_{\text{Rel}}(X; Y)$, since all functions between sets X, Y are a particular subset of all relations between X, Y .
3. For each $X \in \text{Ob}_{\text{Set}}$, the identity relation $\text{Id}_X = \{\langle x, x' \rangle \in X \times X \mid x = x'\}$ 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 $R ; S \subseteq X \times Z$, is again a function. This was proven in Lemma 16.2.

30.2. Drawings

Definition 30.4 (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 24. Check whether just considering

- ▷ affine invertible transformations, or
- ▷ rototranslations, or
- ▷ scalings, or
- ▷ translations, or
- ▷ rotations,

as morphisms forms a subcategory of Draw .

See solution on page 247.

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

$$f_\theta : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

$$\langle x, y \rangle \mapsto \langle x \cos(\theta) + y \sin(\theta), y \cos(\theta) - x \sin(\theta) \rangle.$$

▷ ...

30.3. 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 30.5 (Injective function). Let $f : \mathbf{A} \rightarrow \mathbf{B}$ be a function. The function f is *injective* if, for all $x, x' \in \mathbf{A}$ holds: $f(x) = f(x') \implies x = x'$.

Example 30.6. 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 30.1 is satisfied. Injective functions are a particular type of functions: this satisfies the second condition. Given $X \in \text{Ob}_{\text{InjSet}}$, the identity morphism $\text{Id}_X \in \text{Hom}_{\text{Set}}(X; X)$ corresponds to the identity morphism in $\text{Hom}_{\text{InjSet}}(X; X)$: the identity function is injective. This proves the 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:

$$(f \circ g)(x) = (f \circ g)(x') \implies f(x) = f(x')$$

$$\implies x = x',$$

which proves the fourth condition of Definition 30.1: the composition of injective functions is injective.

Definition 30.7 (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 30.8. We can define a category **Cont** which $\text{Ob}_{\text{Cont}} = \{\mathbb{R}, \mathbb{R}^2, \mathbb{R}^3, \dots\}$ 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**.

Definition 30.9 (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

$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a + h) - f(a)}{h} \tag{30.1}$$

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

Example 30.10. the composition of differentiable functions is differentiable

Definition 30.11 (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 $x_1, x_2 \in \mathbb{R}$:

$$|f(x_1) - f(x_2)| \leq \kappa|x_1 - x_2|. \quad (30.2)$$

Example 30.12. the composition of differentiable functions is differentiable

Generalization outside of \mathbf{R}

Generalization to more general spaces. We used the fact that \mathbf{R} is:

- ▷ For defining continuous functions, we used the fact that \mathbf{R} is topological space (minimum needed for defining a continuous function.). In fact, the real definition of continuous function is:
- ▷ To define Lipschitz we needed the fact that \mathbf{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.

30.4. Subcategories of **Berg**

Recall the category **Berg** presented in Section 14.2. 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 $\langle p, v \rangle \in \text{Ob}_{\mathbf{Berg}}$, 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. $\text{Ob}_{\mathbf{BergAma}} \subseteq \text{Ob}_{\mathbf{Berg}}$.
2. For $A, B \in \text{Ob}_{\mathbf{BergAma}}$, we know that paths satisfying the steepness constraint are specific paths in **Berg**, i.e. $\text{Hom}_{\mathbf{BergAma}} \subseteq \text{Hom}_{\mathbf{Berg}}$.
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 30.1.

30.5. Embeddings



31. Up the ladder

31.1. Functor composition

In the following, we want to show that functors compose. Given categories $\mathbf{A}, \mathbf{B}, \mathbf{C}$ 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 (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 we used that F, G are functors (they preserve composition).

We can define an identity functor. Given a category \mathbf{C} , we define it as $\text{Id}_{\mathbf{C}} : \mathbf{C} \rightarrow$

31.1 Functor composition	229
31.2 A category of categories	230
31.3 Full and faithful functors	230
31.4 Forgetful functor	231

\mathbf{C} , $\text{Id}_{\mathbf{C}}(x) = x$ for every object and morphism in \mathbf{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 \mathbf{C} , we have:

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

31.2. 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 \mathbf{Cat} .

Definition 31.1 (Category of small categories). There is a category, called \mathbf{Cat} , which is constituted of

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

31.3. Full and faithful functors

Definition 31.2 (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

$$F : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{D}}(F(X); F(Y)) \quad (31.1)$$

is surjective (respectively injective).

Example 31.3. Let \mathbf{C} be the category depicted in Fig. 31.1. Let $F : \mathbf{C} \rightarrow \mathbf{C}$ 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.

$$X \rightleftarrows Y$$

Example 31.4. Consider a functor which maps a category \mathbf{C} to its preorder structure \mathbf{D} , that is a category with the same objects as \mathbf{C} and a *unique* morphism from $X \in \text{Ob}_{\mathbf{D}}$ to $Y \in \text{Ob}_{\mathbf{D}}$ if and if only if there is at least one morphism from X

Figure 31.1.

to Y in \mathbf{C} . This functor is full by construction, and it is faithful if and only if \mathbf{C} is a preorder.

31.4. Forgetful functor

Example 31.5. The functor $\mathbf{Pos} \rightarrow \mathbf{Set}$ 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.

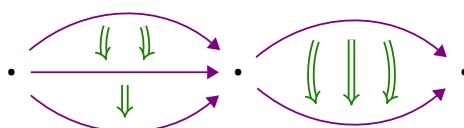


32. Naturality

32.1 Natural transformations 233

32.1. 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 32.1 (Natural transformation). Let \mathbf{C} and \mathbf{D} be categories, and let $F, G : \mathbf{C} \rightarrow \mathbf{D}$ be functors. A *natural transformation* $\alpha : F \Rightarrow G$ is specified by:

Constituents

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

Conditions

- For every morphism $f : X \rightarrow Y$ in \mathbf{C} , the components of α must satisfy the *naturality condition*

$$F(f) \circ \alpha_Y = \alpha_X \circ G(f). \quad (32.1)$$

In other words, the following diagram must commute:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(f)} & F(Y) \\ \alpha_X \downarrow & & \downarrow \alpha_Y \\ G(X) & \xrightarrow{G(f)} & G(Y) \end{array}$$

A natural transformation $\alpha : F \Rightarrow G$ is denoted visually as follows:

$$\begin{array}{ccc} & \curvearrowright^F & \\ \mathbf{C} & \Downarrow \alpha & \mathbf{D} \\ & \curvearrowright_G & \end{array}$$

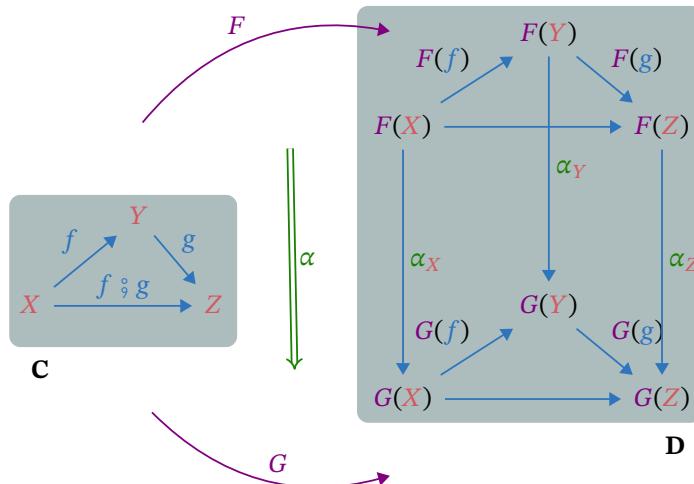


Figure 32.1.

Definition 32.2 (Natural isomorphism). A natural transformation $\alpha : F \Rightarrow G$ is called a *natural isomorphism* if each component morphism α_X in \mathbf{D} is

an isomorphism.

Example 32.3. 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 describing all linear maps from V to \mathbb{R} :

$$V^* := \mathbf{Hom}_{\mathbf{Vect}}(V; \mathbb{R}), \quad (32.2)$$

Also, recall the 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

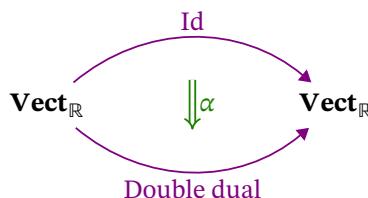
$$\text{Double dual} : \mathbf{Vect} \rightarrow \mathbf{Vect} \quad (32.3)$$

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

$$\alpha_V(v)(l) = l(v), \quad v \in V, l \in V^*. \quad (32.4)$$

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



Graded exercise 4 (`NatTrafosGraphs`). This exercise builds on Graded exercise 2. There, we defined a category \mathbf{G} which has precisely two objects and four morphisms, see Fig. 32.2 (the two identity morphisms are not drawn). The task there was to understand how specifying a functor from this category G into the category of sets is ‘the same thing’ as specifying a directed graph.

Now consider two functors $F_1, F_2 : \mathbf{G} \rightarrow \mathbf{Set}$. Spell out what it means to have a natural transformation $\alpha : F_1 \Rightarrow F_2$. What does this correspond to in the language of directed graphs?

Graded exercise 5 (`UpperSetsNatTrafos`). This exercise builds on Graded exercise 3. There we fixed a poset \mathbf{P} , viewed it as a category \mathbf{P} , and saw that functors $\mathbf{P} \rightarrow \mathbf{Bool}$ encode upper sets in \mathbf{P} . Suppose we have two functors $F_1, F_2 : \mathbf{P} \rightarrow \mathbf{Bool}$. What does a natural transformation $\alpha : F_1 \Rightarrow F_2$ correspond to in terms of the upper sets encoded by F_1 and F_2 , respectively?

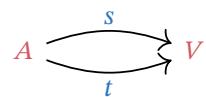


Figure 32.2.



33. Adjunctions

33.1. Galois connections

Definition 33.1 (Monotone Galois Connection). A *monotone Galois connection* between posets \mathbf{P} and \mathbf{Q} is a pair of monotone maps $f : \mathbf{P} \rightarrow \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow \mathbf{P}$ such that for all $p \in \mathbf{P}, q \in \mathbf{Q}$:

$$\frac{f(p) \leq_Q q}{p \leq_P g(q)} \tag{33.1}$$

This is equivalent to ask, for all $p \in \mathbf{P}, q \in \mathbf{Q}$, that:

$$(p \leq_P g(f(p))) \wedge (q \leq_Q f(g(q))). \tag{33.2}$$

33.1 Galois connections	237
33.2 An example	240
33.3 Adjunctions: hom-set definition	240
33.4 Adjunctions: (co)unit definition	241
33.5 Example of a “Product-Hom” adjunction	242
33.6 Example of a “Free-Forgetful” adjunction	243
33.7 Relating the two definitions . . .	245

The *Battle of Surfaces* was a men's tennis exhibition match that was held on May 2, 2007 between the Swiss Roger Federer and Rafael Nadal, respectively number 1 and 2 in the world in men's singles. Federer preferred grass—he was 5 years unbeaten on that terrain. Nadal preferred clay—he was 3 years unbeaten. To check who would win when averaging out the terrain, the match was played on a unique court with a clay surface on one side of the net and grass on the other.

Definition 33.2 (Antitone Galois Connection). An *antitone Galois connection* between \mathbf{P} and \mathbf{Q} is a pair of antitone maps $f : \mathbf{P} \rightarrow \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow \mathbf{P}$ such that for all $p \in \mathbf{P}, q \in \mathbf{Q}$:

$$\frac{q \leq_{\mathbf{Q}} f(p)}{p \leq_{\mathbf{P}} g(q)} \quad (33.3)$$

This is equivalent to ask for all $p \in \mathbf{P}, q \in \mathbf{Q}$:

$$(p \leq_{\mathbf{P}} g(f(p))) \wedge (q \leq_{\mathbf{Q}} f(g(q))). \quad (33.4)$$

Consider a boolean profunctor $d : \mathbf{P} \nrightarrow \mathbf{Q}$. We can define the maps that work on single functionality and resources:

$$\begin{aligned} \theta : \mathbf{P} &\rightarrow \mathcal{U}\mathbf{Q} \\ p &\mapsto \{q \in \mathbf{Q} : d(p, q)\}, \end{aligned} \quad (33.5)$$

$$\begin{aligned} \psi : \mathbf{Q} &\rightarrow \mathcal{L}\mathbf{P} \\ q &\mapsto \{p \in \mathbf{P} : d(p, q)\}. \end{aligned} \quad (33.6)$$

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

$$\begin{aligned} \alpha : \mathcal{L}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{Q} \\ S &\mapsto \{q \in \mathbf{Q} : \exists p \in S : d(p, q)\}, \end{aligned} \quad (33.7)$$

Alternatively, we can write

$$\begin{aligned} \alpha : \mathcal{L}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{Q} \\ S &\mapsto \bigcup_{p \in S} \theta(p). \end{aligned} \quad (33.8)$$

$$\begin{aligned} \beta : \mathcal{U}\mathbf{Q} &\rightarrow \mathcal{L}\mathbf{P} \\ T &\mapsto \{p \in \mathbf{P} : \exists q \in T : d(p, q)\}, \end{aligned} \quad (33.9)$$

Alternatively, we can write

$$\begin{aligned} \beta : \mathcal{U}\mathbf{Q} &\rightarrow \mathcal{L}\mathbf{P} \\ T &\mapsto \bigcup_{q \in T} \psi(q). \end{aligned} \quad (33.10)$$

$$\begin{aligned} \delta : \mathcal{L}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{Q} \\ S &\mapsto \{q \in \mathbf{Q} : \forall p \in S : d(p, q)\}, \end{aligned} \quad (33.11)$$

Alternatively, we can write

$$\begin{aligned} \delta : \mathcal{L}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{Q} \\ S &\mapsto \bigcap_{p \in S} \theta(p). \end{aligned} \quad (33.12)$$

$$\begin{aligned}\gamma: \mathcal{U}\mathbf{Q} &\rightarrow \mathcal{L}\mathbf{P} \\ T &\mapsto \{\mathbf{p} \in \mathbf{P}: \forall \mathbf{q} \in T: d(\mathbf{p}, \mathbf{q})\},\end{aligned}\tag{33.13}$$

Alternatively, we can write

$$\begin{aligned}\delta: \mathcal{U}\mathbf{Q} &\rightarrow \mathcal{L}\mathbf{P} \\ T &\mapsto \bigcap_{\mathbf{q} \in T} \psi(\mathbf{q}).\end{aligned}\tag{33.14}$$

Properties of these maps are reported in Table 33.1.

Table 33.1.: Properties of $\alpha, \beta, \delta, \gamma$

\star	X	Y	$\star(\perp)$	$\star(\top)$	$A \leq_X B$	$\star(A \vee_X B)$	$\star(A \wedge_X B)$
α	$\mathcal{L}\mathbf{P}$	$\mathcal{U}\mathbf{Q}$	$\alpha(\emptyset) = \emptyset$	$\alpha(\mathbf{P}) \geq_{\mathcal{U}\mathbf{Q}} \alpha(\cdot)$	$\alpha(A) \geq_{\mathcal{U}\mathbf{Q}} \alpha(B)$	$\alpha(A) \vee_{\mathcal{L}\mathbf{P}} \alpha(B)$	$\alpha(A) \wedge_{\mathcal{L}\mathbf{P}} \alpha(B)$
β	$\mathcal{U}\mathbf{Q}$	$\mathcal{L}\mathbf{P}$	$\beta(\mathbf{Q}) \geq_{\mathcal{L}\mathbf{P}} \beta(\cdot)$	$\beta(\emptyset) = \emptyset$	$\beta(A) \geq_{\mathcal{L}\mathbf{P}} \beta(B)$	$\beta(A) \vee_{\mathcal{L}\mathbf{P}} \beta(B)$	$\beta(A) \wedge_{\mathcal{L}\mathbf{P}} \beta(B)$
δ	$\mathcal{L}\mathbf{P}$	$\mathcal{U}\mathbf{Q}$	$\delta(\emptyset) = \mathbf{Q}$	$\delta(\mathbf{P}) \geq_{\mathcal{U}\mathbf{Q}} \delta(\cdot)$	$\delta(A) \leq_{\mathcal{U}\mathbf{Q}} \delta(B)$	$\delta(A) \wedge_{\mathcal{U}\mathbf{Q}} \delta(B)$	$\delta(A) \vee_{\mathcal{U}\mathbf{Q}} \delta(B)$
γ	$\mathcal{U}\mathbf{Q}$	$\mathcal{L}\mathbf{P}$	$\gamma(\mathbf{Q}) \leq_{\mathcal{L}\mathbf{P}} \gamma(\cdot)$	$\gamma(\emptyset) = \mathbf{P}$	$\gamma(A) \leq_{\mathcal{L}\mathbf{P}} \gamma(B)$	$\gamma(A) \wedge_{\mathcal{L}\mathbf{P}} \gamma(B)$	$\gamma(A) \vee_{\mathcal{L}\mathbf{P}} \gamma(B)$

Lemma 33.3. δ and γ are monotone maps.

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

$$\begin{aligned}\delta(A) &= \{\mathbf{q} \in \mathbf{Q}: \forall \mathbf{p} \in A: d(\mathbf{p}, \mathbf{q})\} \\ &\supseteq \{\mathbf{q} \in \mathbf{Q}: \forall \mathbf{p} \in B: d(\mathbf{p}, \mathbf{q})\} \\ &= \delta(B),\end{aligned}\tag{33.15}$$

meaning that $A \leq_{\mathcal{L}\mathbf{P}} B \Rightarrow \delta(A) \leq_{\mathcal{U}\mathbf{Q}} \delta(B)$. We now prove that γ is a monotone map. Given $C, D \in \mathcal{U}\mathbf{Q}$, with $C \supseteq D$, one has

$$\begin{aligned}\gamma(C) &= \{\mathbf{p} \in \mathbf{P}: \forall \mathbf{q} \in C: d(\mathbf{p}, \mathbf{q})\} \\ &\subseteq \{\mathbf{p} \in \mathbf{P}: \forall \mathbf{q} \in D: d(\mathbf{p}, \mathbf{q})\} \\ &= \gamma(D),\end{aligned}\tag{33.16}$$

meaning that $C \leq_{\mathcal{U}\mathbf{Q}} D \Rightarrow \gamma(C) \leq_{\mathcal{L}\mathbf{P}} \gamma(D)$. \square

Lemma 33.4. α and β are antitone maps.

Proof. We first prove that α is an antitone map. Given $A, B \in \mathcal{L}\mathbf{P}$, with $A \subseteq B$, one has

$$\begin{aligned}\alpha(A) &= \{\mathbf{q} \in \mathbf{Q}: \exists \mathbf{p} \in A: d(\mathbf{p}, \mathbf{q})\} \\ &\subseteq \{\mathbf{q} \in \mathbf{Q}: \exists \mathbf{p} \in B: d(\mathbf{p}, \mathbf{q})\} \\ &= \alpha(B),\end{aligned}\tag{33.17}$$

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

$$\begin{aligned}\beta(C) &= \{\mathbf{p} \in \mathbf{P}: \exists \mathbf{q} \in C: d(\mathbf{p}, \mathbf{q})\} \\ &\supseteq \{\mathbf{p} \in \mathbf{P}: \exists \mathbf{q} \in D: d(\mathbf{p}, \mathbf{q})\} \\ &= \beta(D),\end{aligned}\tag{33.18}$$

meaning that $C \leq_{\mathcal{U}\mathbf{Q}} D \Rightarrow \beta(C) \geq_{\mathcal{U}\mathbf{Q}} \beta(D)$. □

Lemma 33.5. (δ, γ) forms a **monotone** Galois connection between $\mathcal{L}\mathbf{P}$ and $\mathcal{U}\mathbf{Q}$.

Proof. In Lemma 33.3 we proved that δ and γ are monotone maps. We now need to show that for any lower set $L \subseteq \mathbf{P}$ of functionalities and upper set $U \subseteq \mathbf{Q}$ of resources, we have

$$L \subseteq \gamma(U) \iff \delta(L) \supseteq U \quad (33.19)$$

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

$$\begin{aligned} L \subseteq \gamma(U) &\equiv L \subseteq \{p \in \mathbf{P} : \forall q \in U : d(p, q)\} \\ &\equiv \forall p \in L, q \in U : d(p, q) = \top \\ &\equiv \forall q \in U, p \in L : d(p, q) = \top \\ &\equiv U \subseteq \{q \in \mathbf{Q} : \forall p \in L : d(p, q) = \top\} \\ &\equiv U \subseteq \delta(L). \end{aligned} \quad (33.20)$$

□

Lemma 33.6. (α, β) does not form an **antitone** Galois connection between $\mathcal{L}\mathbf{P}$ and $\mathcal{U}\mathbf{Q}$.

Proof. In Lemma 33.4 we have proved that α and β are antitone maps. For $L \in \mathbf{P}, U \in \mathbf{Q}$, we want to show that the following does not hold:

$$L \subseteq \beta(\alpha(L)) \quad (33.21)$$

and

$$U \supseteq \alpha(\beta(U)). \quad (33.22)$$

Example Consider d as the design problem which is always not feasible (the empty profunctor), which means $d(p, q) = \perp, \forall p \in \mathbf{P}, q \in \mathbf{Q}$. Take any $L \in \mathbf{P}$. We know that $\alpha(L) = \emptyset$, and $\beta(\alpha(L)) = \beta(\emptyset) = \emptyset$. But $L \subseteq \emptyset$ is not true. □

33.2. An example

33.3. Adjunctions: hom-set definition

In this section we give a definition of adjunction which can be viewed as an analogy with the following situation in linear algebra. Suppose V and W are finite-dimensional real vector spaces, equipped with inner products $(-, -)_V$ and $(-, -)_W$, respectively. The adjoint of a linear map $F : V \rightarrow W$ is a linear map $F^* : W \rightarrow V$ such that

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

Definition 33.7 (Adjunction, Version 1). Let \mathbf{C} and \mathbf{D} be categories. An adjunction from \mathbf{C} to \mathbf{D} is given by the following data:

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

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

Remark 33.8. Note that τ is a natural isomorphism between functors of the form

$$\mathbf{C}^{\text{op}} \times \mathbf{D} \longrightarrow \mathbf{Set} \quad (33.23)$$

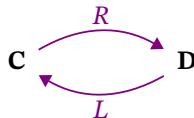
33.4. Adjunctions: (co)unit definition

Recall from Definition 25.1: in a category \mathbf{C} , a morphism $f : X \rightarrow Y$ is an isomorphism if there exists a morphism $g : Y \rightarrow X$ such that

$$f \circ g = \text{Id}_X \text{ and } g \circ f = \text{Id}_Y$$

Now let's think about this definition in the case where \mathbf{C} is the category \mathbf{Cat} of categories. We will consider weakenings of the notion of isomorphism in this setting, and this will lead to a second (but equivalent) definition of adjunction. The precise relationship between the two definitions will be spelled out Section 33.7.

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



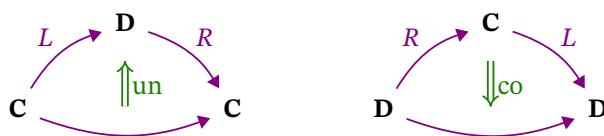
instead of requiring the equations

$$L \circ R = \text{Id}_{\mathbf{C}} \text{ and } R \circ L = \text{Id}_{\mathbf{D}}$$

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

$$L \circ R \xrightarrow{\text{un}} \text{Id}_{\mathbf{C}}, \quad R \circ L \xleftarrow{\text{co}} \text{Id}_{\mathbf{D}}$$

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



Definition 33.9 (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 $L : \mathbf{C} \rightarrow \mathbf{D}$;
2. A functor $R : \mathbf{D} \rightarrow \mathbf{C}$;
3. Natural isomorphisms $\text{un} : \text{Id}_{\mathbf{C}} \Rightarrow L \circ R$ and $\text{co} : R \circ L \Rightarrow \text{Id}_{\mathbf{D}}$.

Definition 33.10 (Adjunction, Version 2). Let \mathbf{C} and \mathbf{D} be categories. An adjunction from \mathbf{C} to \mathbf{D} is given by the following data, satisfying the following conditions.

Data:

1. A functor $L : \mathbf{C} \rightarrow \mathbf{D}$ (the left adjoint);
2. A functor $R : \mathbf{D} \rightarrow \mathbf{C}$ (the right adjoint);
3. Natural transformations $\text{un} : \text{Id}_{\mathbf{C}} \Rightarrow L \circ R$ and $\text{co} : R \circ L \Rightarrow \text{Id}_{\mathbf{D}}$

Conditions:

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

$$\text{Lun}_X \circ \text{co}_{LX} = \text{Id}_{LX} \text{ and } \text{un}_{RY} \circ R\text{co}_Y = \text{Id}_{RY}$$

i.e. that the following diagrams commute:

$$\begin{array}{ccc} LX & \xrightarrow{\text{Lun}_X} & LRLX \\ & \searrow \text{Id}_{LX} & \downarrow \text{co}_{LX} \\ & LX & \end{array} \quad \begin{array}{ccc} RY & \xrightarrow{\text{un}_{RY}} & RLRY \\ & \searrow \text{Id}_{RY} & \downarrow R\text{co}_Y \\ & RY & \end{array}$$

The 2-morphisms un and co 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 33.11. The conditions (triangle identities) from Definition 33.10 are “hidden” in Definition 33.7 in the condition that τ be an natural isomorphism. In Section 33.7 we spell out how the two definitions are related.

33.5. Example of a “Product-Hom” adjunction

We will consider an adjunction between the category \mathbf{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

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

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

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

In terms of Definition 33.7, there is a natural isomorphism

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

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

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

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 33.10, 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

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

33.6. 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 \mathbf{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 \mathbf{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 \mathbf{Y} on \mathbf{D} which has some structure, and “forgets” some of this structure, which results in us being able to view \mathbf{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 27.2). 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 $\mathbf{Ob}_{\mathbf{C}}$ 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 33.7. The natural isomorphism

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

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

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

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 it’s compositional properties and think of it just as a graph morphism).

Now let’s consider this adjunction from the perspective of Definition 33.10. 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? Its 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} .

33.7. 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 **C** to a category **D**, we have, by Definition 33.7 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 **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 **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 25. 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 .

See solution on page 247.

Exercise 26. Show that η and ϵ , as defined above, satisfy the triangle identities stated in Definition 33.10.

See solution on page 247.

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

See solution on page 247.

Exercise 28. 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 $\mathbf{C}^{\text{op}} \times \mathbf{D} \rightarrow \mathbf{Set}$.

See solution on page 247.

34. Solutions to selected exercises

Solution of Exercise 24.

Solution of Exercise 25.

Solution of Exercise 26.

Solution of Exercise 27.

Solution of Exercise 28.

PART F.DESIGN



35. Design	251
36. Design problems	257
37. Feasibility	277
38. Profunctors	287
39. Parallelism	291
40. Feedback	307
41. Ordering design problems	313
42. Constructing design problems	319
43. Solutions to selected exercises	323



35. Design

This chapter introduces basic concepts of engineering design, and describes what are the design queries we want to answer.

35.1. What is “design”?

We take a broad view of what it means to “design”, that is not limited to engineering. Citing at length Hebert Simon’s[†] *The sciences of the artificial* ([29], 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.

35.1 What is “design”?	251
35.2 What is “co-design”?	252
“Co” for “compositional”	252
“Co” for “collaborative”	252
“Co” for “computational”	253
“Co” for “continuous”	253
35.3 Basic concepts of formal engineering design	254
35.4 Queries in design	255

[†] Hebert A. Simon (1916-2001). Winner of the 1978 Nobel Prize in Economics.

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.

35.2. What is “co-design”?

The word “co-design” is not a new one. In this book, we will use a meaning that incorporates and extends the existing meaning.

We take the “Co” in “co-design” to have four meanings:

1. “co” for “compositional”;
2. “co” for “collaborative”;
3. “co” for “computational”;
4. “co” for “continuous”.

These meanings together describe the aspects of modern engineering design.

“Co” for “compositional”

The first meaning has to do with composition:

co-design = design everything together

We use the word “co-design” to refer to any decision procedure that has to do with making simultaneous choices about the components of a system to achieve system-level goals. This includes the choice of components, the interconnection of components, and the configuration of components. We will see that in most cases, choices that are made at the level of components without looking at the entire system are doomed to be suboptimal.

Slightly modifying Haiken’s quote in ??, we choose this as our slogan:

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

“Co” for “collaborative”

In a second broad meaning, “co-” stands for “collaborative”:

co-design = design everything, together

There are two types of collaborations. First, there is the collaboration between human and machine, in the definition and solution of design problems. Second, and most importantly, is the collaboration among different experts or teams of experts in the design process.

The typical situation is that the system design is suboptimal because every expert only knows one component and there are rigid interfaces/contracts designed early on. The problem here is sharing of knowledge across teams, specifically, knowledge about the design of systems.

In this case, this is the slogan:

«A system is composed of components;
a component is something that **somebody** understands **how to design**.»

There is a tight link between the “composition” and “collaboration” aspects.

As Conway[‡] first observed for software systems:

«Organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.»

This “mirroring” hypothesis between system and organization was explored formally and found to hold [22]. The ultimate reason is that “the organization’s governance structures, problem solving routines and communication patterns constrain the space in which it searches for new solutions”. This appears to be true for generic systems in addition to software.

In the end, civilization is about dividing up the work, and so we must choose where one’s work ends and the other’s work begins. But we need to keep talking if we want that everything works together.

“Co” for “computational”

The third meaning of “co-” in “co-design” will be **computational**. It is the age of machines and we need machines to understand what we are doing.

Therefore, we strive to create not only a qualitative modeling for co-design, but also a formal and quantitative description that will be suitable for setting up an optimization problem that can be solved to obtain an optimal design.

Our slogan becomes:

«A system is composed of components;
a component is something that **somebody** understands **how to design well enough to teach a computer**.»

“Co” for “continuous”

The fourth meaning of “co-” is **continuous**. We look at designs not as something that exists as a single decision in time, but rather as something that continuous to exist and evolve, independently on the designer.

[‡] John Horton Conway (1937–2020) was a mathematician. Probably the most popular idea of his was the invention of the Game of Life, which inspired countless works on cellular automata. We remember him for the discovery of the *surreal numbers*, which should be just called *numbers*, as they contain all other ordered fields.

35.3. Basic concepts of formal engineering design

We will informally introduce some of the basic nomenclature of engineering design [1, 36]. 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 35.1. 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 35.2. 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* [36] such as the extensibility and the maintainability of the system. Nevertheless, functionality and resources can express most of the requirements which can be quantitatively evaluated, at least prior to designing, assembling, and testing the entire system.

Implementation space The *implementation space* or *design space* is the set of all possible design choices that could be chosen; by *implementation*, or the word

“design”, used as a noun, we mean one particular set of choices. The implementation space \mathbf{I} is the set over which we are optimizing; an implementation $i \in \mathbf{I}$ is a particular point in that set (Fig. 35.1).

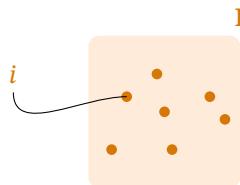


Figure 35.1.: An implementation i is a particular point in the implementation space \mathbf{I} .

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. 35.2).

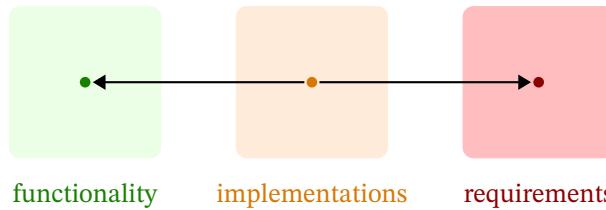


Figure 35.2.: Evaluation of specific implementations to get functionality and resources spaces.

Functional Interfaces and interconnection Components are *interconnected* to create a system. This implies that we have defined the *interfaces* of components, which have the dual function of delimiting when one component ends and another begins, and also to describe exactly what is the nature of their interaction.

We will develop a formalism in which the functionality and resources are the interfaces used for interconnection: two components are connected if the resources required by the first correspond to the functionality provided by the second.

Abstraction By *abstraction*, we mean that it is possible to “zoom out”, in the sense that a system of components can be seen as a component itself, which can be part of larger systems.

Compositionality A *compositional* property is a property that is preserved by interconnection and abstraction; assuming each component in a system satisfies that property, also the system as a whole satisfies the property.

Example 35.3. 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.

35.4. Queries in design

Suppose that we have a model with a functionality space \mathbf{F} , a requirements space \mathbf{R} , and an implementation space \mathbf{I} .

There are several queries we can ask of a model. They all look at the same phenomenon from different angles, so they look similar; however the computational cost of answering each one might be very different.

The first kind of query is one that asks if the design is feasible when fixed all variables.

Problem (Feasibility problem). Given a triplet of implementation $i \in \mathbf{I}$, functionality $f \in \mathbf{F}$, requirements $r \in \mathbf{R}$, determine if the design is feasible.

The second type of query is that which fixes the boundary conditions of functionality and requirements, and asks to find a solution.

Problem (Find implementation). Given a pair of minimal requested functionality $f \in \mathbf{F}$ and maximum allowed requirements $r \in \mathbf{R}$, determine if there is a an implementation $i \in \mathbf{I}$ that is feasible.

A different type of query is the one in which the design objective (the functionality) is fixed, and we ask what are the least resources necessary.

Problem (FixFunMinReq). Given a certain functionality $f \in \mathbf{F}$, find the set of “minimal” resources in \mathbf{R} that are needed to realize it (along with the implementations), or provide a proof that there are none.

Dually, we can ask, fixed the resources available, what are the functionalities that can be required.

Problem (FixReqMinFun). Given a certain requirement $r \in \mathbf{R}$, find the set of “maximal” functionalities in that can be realize 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.



36. Design problems

36.1. Design Problems

We start by defining a “design problem with implementation”, which is a tuple of “functionality space”, “implementation space”, and “resources space”, together with two maps that describe the feasibility relations between these three spaces (Fig. 36.1).

Definition 36.1 (Design problem with implementation). A *design problem with implementation* (DPI) is a tuple

$$\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle, \quad (36.1)$$

where:

- ▷ **F** is a poset, called *functionality space*;
- ▷ **R** is a poset, called *requirements space*;
- ▷ **I** is a set, called *implementation space*;
- ▷ the map $\text{prov} : \mathbf{I} \rightarrow \mathbf{F}$ maps an implementation to the functionality it provides;
- ▷ the map $\text{req} : \mathbf{I} \rightarrow \mathbf{R}$ maps an implementation to the resources it requires.

36.1 Design Problems	257
Mechatronics	259
Geometrical constraints	260
Inference	260
Communication	261
Multi-robot systems	261
Computation	262
Other examples in minimal robotics	263
36.2 Queries	266
36.3 The semi-category DPI	267
36.4 Co-design problems	269
36.5 Discussion of related work	274
Theory of design	274
Partial Order Programming	274
Abstract interpretation	274

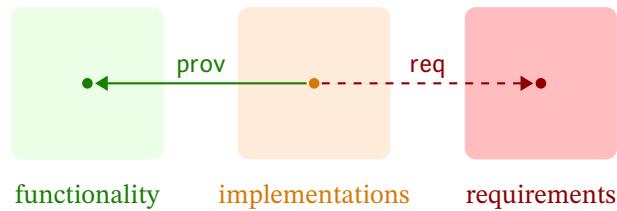


Figure 36.1.

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. 36.2).

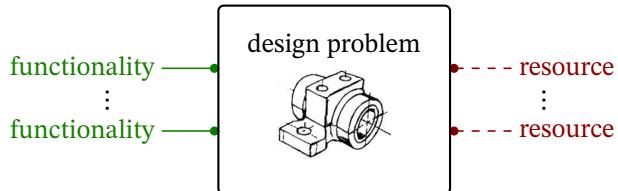


Figure 36.2.

This means that the functionality and resources spaces can be factorized in n_f and n_r components:

$$\mathbf{F} = \prod_{i=1}^{n_f} \pi_i \mathbf{F}_i, \quad \mathbf{R} = \prod_{j=1}^{n_r} \pi_j \mathbf{R}_j,$$

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 \mathbf{F} (respectively, \mathbf{R}) is equal to $\mathbf{1} = \{\langle \rangle\}$, the set containing one element, the empty tuple $\langle \rangle$.

These *co-design diagrams* are not to be confused with signal flow diagrams, in which the boxes represent oriented systems and the edges represent signals.

Example 36.2. We now want to revisit the leading example of ?? 25 with the newly introduced co-design perspective. Let's consider a list of electrical motors as in Table 36.1.

Table 36.1.: A simplified catalogue of motors.

Motor ID	Company	Torque [kg · cm]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	0.18	60.0	2.34	19.95
1206	SOYO	0.95	140.0	3.00	19.95
1207	SOYO	0.65	130.0	2.07	12.95
2267	SOYO	3.7	285.0	4.76	16.95
2279	Sanyo Denki	1.9	165.0	5.40	164.95
1478	SOYO	19.0	1,000	8.96	49.95
2299	Sanyo Denki	2.2	150.0	5.90	59.95

We can think of this as a catalogue of electric motors $\langle \mathbf{I}_{EM}, \text{prov}_{EM}, \text{req}_{EM} \rangle$. In particular, the set of implementations collects all the motor models, which we can specify using the motor IDs:

$$\mathbf{I}_{EM} = \{1204, 1206, 1207, 2267, 2279, 1478, 2299\}. \quad (36.2)$$

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 $\text{kg} \cdot \text{cm}$). Thus, we can identify

$$\mathbf{F} = \mathbb{R} \times \{\text{kg} \cdot \text{cm}\}, \quad \mathbf{R} = (\mathbb{R} \times \{\text{g}\}) \times (\mathbb{R} \times \{\text{W}\}) \times (\mathbb{R} \times \{\text{USD}\}),$$

by considering the units as discussed in Section 12.1. The correspondences are given by the details in Table 36.1. For instance, we have

$$\text{prov}_{\text{EM}}(1204) = 0.18 \text{ kg} \cdot \text{cm}, \quad \text{req}_{\text{EM}}(1204) = \langle 60 \text{ g}, 2.34 \text{ W}, 19.95 \text{ USD} \rangle. \quad (36.3)$$

Example 36.3 (Motor design). Suppose we need to choose a motor for a robot from a given set. The *functionality* of a motor could be parametrized by **torque** and **speed**. The *resources* to consider could include the **cost [USD]**, the **mass [g]**, the input **voltage [V]**, and the input **current [A]**. The map $\text{prov} : \mathbf{I} \rightarrow \mathbf{F}$ assigns to each motor its functionality, and the map $\text{req} : \mathbf{I} \rightarrow \mathbf{R}$ assigns to each motor the resources it needs (Fig. 36.3).

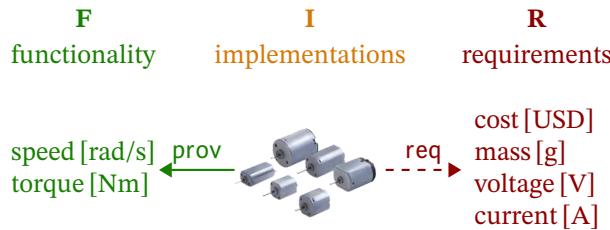


Figure 36.3.

Example 36.4 (Chassis design). Suppose we need to choose a chassis for a robot (Fig. 36.4). The implementation space \mathbf{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 [USD]** of the chassis; the total mass; and, for each motor to be placed in the chassis, the required **speed [rad/s]** and **torque [Nm]**.

Mechatronics

Many mechanisms can be readily modeled as relations between a provided functionality and required resources.

Example 36.5. The **functionality** of a DC motor (Fig. 36.5) is to provide a certain **speed** and **torque**, and the **resources** are **current** and **voltage**.

Example 36.6. A gearbox (Fig. 36.6) 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 $\gamma, 0 < \gamma < 1$, the constraints among those quantities are $\omega_i \geq r \omega_o$ and $\tau_i \omega_i \geq \gamma \tau_o \omega_o$.

Example 36.7. Propellers (Fig. 36.7) generate **thrust** given a certain **torque** and **speed**.



Figure 36.4.



Figure 36.5.

Example 36.8. A *crank-rocker* (Fig. 36.8) converts **rotational motion** into a **rocking motion**.

Geometrical constraints

Geometrical constraints are examples of constraints that are easily recognized as monotone, but possibly hard to write down in closed form.

Example 36.9 (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. 36.9). 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 [20]. 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.

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 36.10. (SLAM) One issue with particle-filter-based estimation procedures, such as the ones used in the popular GMapping [15] method, is that the



Figure 36.6.

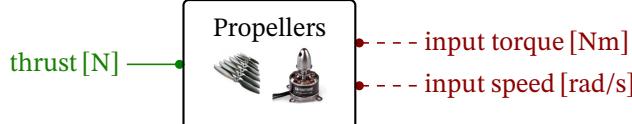


Figure 36.7.

filter might diverge if there aren't enough particles. Although the relation might be hard to characterize, there is a monotone relation between the **robustness** (1 - probability of failure), the **accuracy**, and the **number of particles** (Fig. 36.10).

Example 36.11. (Stereo reconstruction) Progressive reconstruction systems ([19]), 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. 36.11). A similar relation characterizes any anytime algorithms in other domains, such as robot motion planning.

Example 36.12. 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 [24, 37].

Communication

Example 36.13 (Transducers). Any type of "transducer" that bridges between different mediums can be modeled as a DP. For example, an access point (Fig. 36.12) provides the "**wireless access**" functionality, and requires that the infrastructure provides the "**Ethernet access**" resource.

Example 36.14 (Wireless link). The basic functionality of a wireless link is to provide a certain **bandwidth** (Fig. 36.13). Further refinements could include bounds on the latency or the probability that a packet drop is dropped. Given the established convention about the 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 $\mathbf{R} = \mathbb{R}^{[W]} \times \text{intervals}(\mathbb{R}^{[Hz]})$.

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.

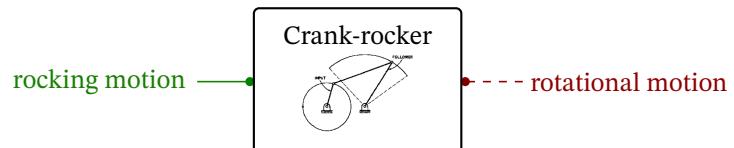


Figure 36.8.



Figure 36.9.

Example 36.15. 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. 36.14b).

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 [33, 18, Chapter 3], the application is represented as a graph of procedures that need to be allocated on a network of processors.

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

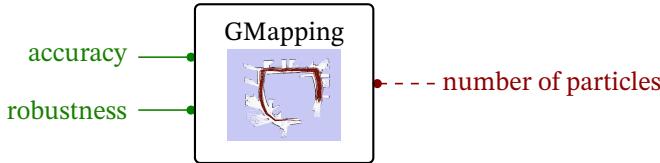


Figure 36.10.

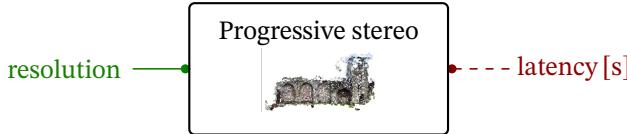


Figure 36.11.

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 36.16. Svorenova et al. [35] consider a joint sensor scheduling and control synthesis problem, in which a robot can decide to not perform sensing to save power, given performance objectives on the probability of reaching the target and the probability of collision. The method outputs a Pareto frontier of all possible operating points. This can be cast as a design problem with functionality equal to the **probability of reaching the target** and (the inverse of) the **collision probability**, and with resources equal to the **actuation power**, **sensing power**, and **sensor accuracy**.

Example 36.17. Nardi et al. [37] 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.

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. 36.20). Given a task, there is a minimal antichain in the resources poset that describes the possible trade-offs (for instance, compensating lousier sensors with more computation).



Figure 36.12.

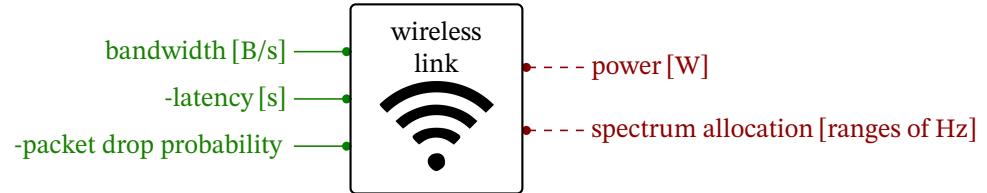


Figure 36.13.

The poset structure arises naturally: for example, in the *sensor lattice* [17], 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 [25] define a robot as a union of “robotic primitives”, where each primitive is an abstraction for a set of sensors, actuators, and control strategies that can be used together (for instance, a compass plus a contact sensor allow to “drive North until a wall is hit”). The effect of each primitive is modeled as an operator on the robot’s information space. It is possible to work out what are the minimal combinations of robotic primitives (minimal antichain) that are sufficient to perform a task (for instance, global localization), and describe a dominance relation (partial order) of primitives. Other works have focused on minimizing the complexity of the controller. Egerstedt [10] 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 [30] studies the relation between the **performance of a visual task**, and the **minimal representation** that is needed to perform that task.

Example 36.18 (Hoare logic).

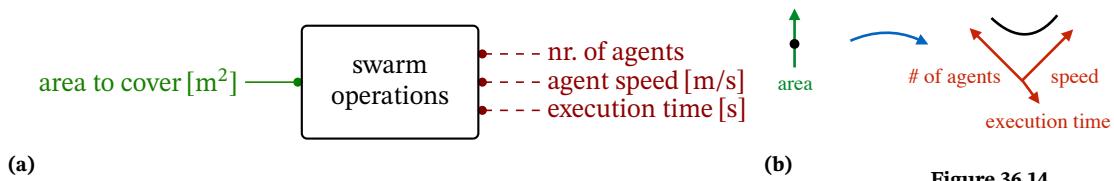


Figure 36.14.

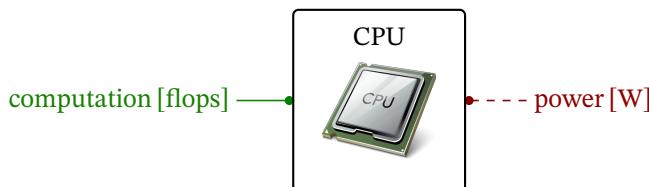


Figure 36.15.

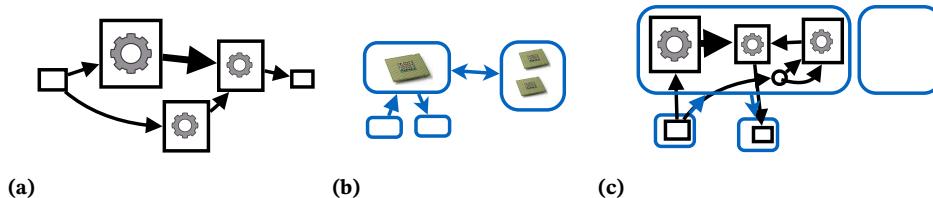


Figure 36.16.

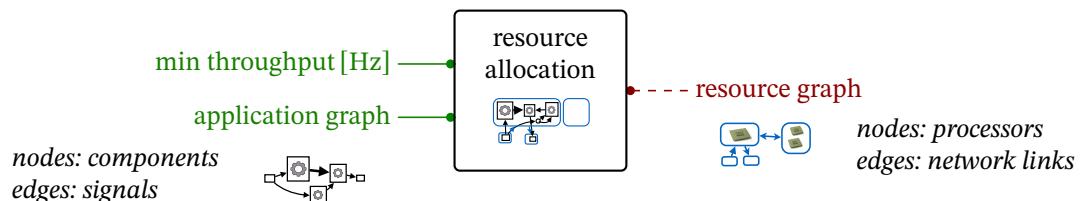


Figure 36.17.

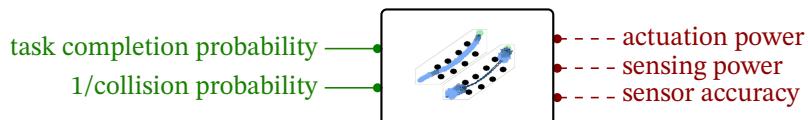


Figure 36.18.



Figure 36.19.

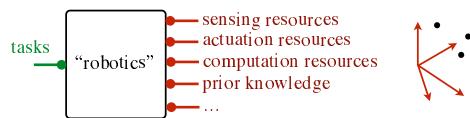


Figure 36.20.:

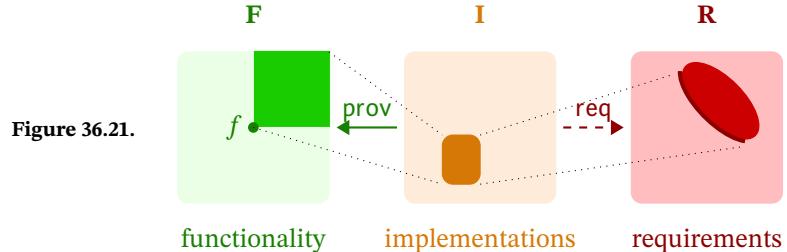
36.2. Queries

A DPI is a model to which we can associate a family of optimization problems. While in previous examples we covered the problem “feasibility”, we still miss FixFunMinReq, FixResMinFun, and FeasibleImp.

The first can be translated to “Given a lower bound on the functionality f , what are the implementations that have minimal resources usage?” (Fig. 36.21).

Problem (FixFunMinReq). Given $f \in \mathbf{F}$, find the implementations in \mathbf{I} that realize the functionality f (or higher) with minimal resources, or provide a proof that there are none:

$$\left\{ \begin{array}{ll} \text{using} & i \in \mathbf{I}, \\ \text{Min}_{\leq_{\mathbf{R}}} & r, \\ \text{s.t.} & r = \text{req}(i), \\ & f \leq_{\mathbf{F}} \text{prov}(i). \end{array} \right. \quad (36.4)$$

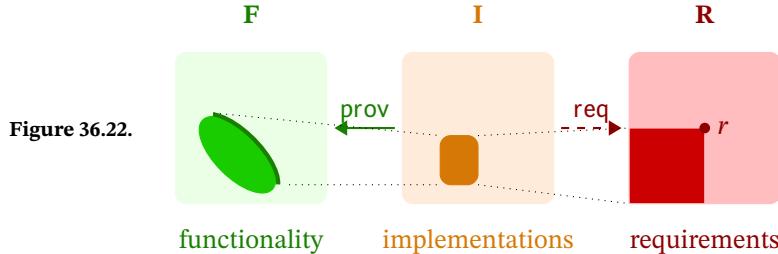


Remark 36.19 (Minimal vs least solutions). Note the use of “ $\text{Min}_{\leq_{\mathbf{R}}}$ ” in (36.4), which indicates the set of minimal (non-dominated) elements according to $\leq_{\mathbf{R}}$, rather than “ $\text{min}_{\leq_{\mathbf{R}}}$ ”, which would presume the existence of a least element. 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 $R \in \mathcal{AR}$. We will see that this formalization allows an elegant way to treat multi-objective optimization problems. The algorithm to be developed will directly solve for the set R , without resorting to techniques such as *scalarization*, and therefore is able to work with arbitrary posets, possibly discrete.

In an entirely symmetric fashion, we could fix an upper bound on the resources usage, and then maximize the functionality provided (Fig. 36.22). The formulation is entirely dual, in the sense that it is obtained from (36.4) by swapping Min with Max, \mathbf{F} with \mathbf{R} , and prov with req .

Problem (FixResMinFun). Given $r \in \mathbf{R}$, find the implementations in \mathbf{I} that requires r (or lower) and provide the maximal functionality, or provide a proof that there are none:

$$\left\{ \begin{array}{ll} \text{using} & i \in \mathbf{I}, \\ \text{Max}_{\leq_{\mathbf{F}}} & f, \\ \text{s.t.} & f = \text{prov}(i), \\ & r \geq_{\mathbf{R}} \text{req}(i). \end{array} \right. \quad (36.5)$$



Another type of query is “Given a lower bound on the functionality f and an upper bound on the costs f , what are the feasible implementations?

Problem (FeasibleImp). Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find the implementations in \mathbf{I} that requires r (or lower) and provide f (or higher)

$$\begin{cases} \text{using } i \in \mathbf{I}, \\ \text{s.t. } f \leq_{\mathbf{F}} \text{prov}(i), \\ \text{s.t. } \text{prov}(i) \leq_{\mathbf{R}} r, \end{cases} \quad (36.6)$$

Another variation is to find only whether there are feasible solutions or not.

Problem (Feasibility). Given $f \in \mathbf{F}$ and $r \in \mathbf{R}$, find if Section 36.2 is feasible.

36.3. The semi-category DPI

Graphically, one is allowed to connect only edges of different colors and of the same type. This interconnection is indicated with the symbol “ \leq ” in a rounded box (Fig. 36.23).

$$\boxed{\square} \xrightarrow[r_1]{\quad} \circledast \xleftarrow[f_2]{\quad} \boxed{\square} \quad \equiv \quad r_1 \leq f_2$$

Figure 36.23.

The semantics of the interconnection is that the second DPI provides the resources required by the first DPI. This is a partial order inequality constraint of the type $r_1 \leq f_2$.

Definition 36.20 (DPI composition). The series composition of two DPIs

$$\begin{aligned} \mathbf{d}_1 &= \langle \mathbf{F}_1, \mathbf{R}_1, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle, \\ \mathbf{d}_2 &= \langle \mathbf{F}_2, \mathbf{R}_2, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle, \end{aligned} \quad (36.7)$$

for which $\mathbf{F}_2 = \mathbf{R}_1$, is defined as

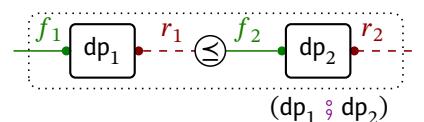
$$(\mathbf{d}_1 ; \mathbf{d}_2) := \langle \mathbf{F}_1, \mathbf{R}_2, \mathbf{I}, \text{prov}, \text{req} \rangle, \quad (36.8)$$

where:

$$\mathbf{I} = \{[i_1 ; i_2] \in (\mathbf{I}_1 ; \mathbf{I}_2) \mid \text{req}_1(i_1) \leq_{\mathbf{R}_1} \text{prov}_2(i_2)\}, \quad (36.9)$$

$$\begin{aligned} \text{prov} : [i_1 ; i_2] &\mapsto \text{prov}_1(i_1), \\ \text{req} : [i_1 ; i_2] &\mapsto \text{req}_2(i_2). \end{aligned} \quad (36.10)$$

The composition is graphically represented as in Section 36.3.



Lemma 36.21. Series composition is associative.

Proof. Consider

$$\begin{aligned} \text{dp}_1 &= \langle \mathbf{F}_1, \mathbf{R}_1, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle, \\ \text{dp}_2 &= \langle \mathbf{F}_2, \mathbf{R}_2, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle, \\ \text{dp}_3 &= \langle \mathbf{F}_3, \mathbf{R}_3, \mathbf{I}_3, \text{prov}_3, \text{req}_3 \rangle, \end{aligned} \quad (36.11)$$

for which $\mathbf{F}_2 = \mathbf{R}_1$ and $\mathbf{F}_3 = \mathbf{R}_2$. We want to show that

$$(\text{dp}_1 \circ \text{dp}_2) \circ \text{dp}_3 = \text{dp}_1 \circ (\text{dp}_2 \circ \text{dp}_3). \quad (36.12)$$

We know that the first part of the left term of (36.12) gives

$$(\text{dp}_1 \circ \text{dp}_2) = \langle \mathbf{F}_1, \mathbf{R}_2, \mathbf{I}_{1,2}, \text{prov}_{1,2}, \text{req}_{1,2} \rangle, \quad (36.13)$$

Therefore, the full left term of (36.12) reads

$$(\text{dp}_1 \circ \text{dp}_2) \circ \text{dp}_3 = \langle \mathbf{F}_1, \mathbf{R}_3, \mathbf{I}_{1,3}, \text{prov}_{1,3}, \text{req}_{1,3} \rangle, \quad (36.14)$$

with

$$\begin{aligned} \mathbf{I}_{1,3} &= \{[[i_1 ; i_2] ; i_3] \in ((\mathbf{I}_1 \circ \mathbf{I}_2) \circ \mathbf{I}_3) \mid \text{req}_{1,2}([i_1 ; i_2]) \leq_{\mathbf{R}_2} \text{prov}_3(i_3) \\ &\quad \wedge \text{req}_1(i_1) \leq_{\mathbf{R}_1} \text{prov}_2(i_2)\} \\ &= \{[i_1 ; i_2 ; i_3] \in (\mathbf{I}_1 \circ \mathbf{I}_2 \circ \mathbf{I}_3) \mid \text{req}_2(i_2) \leq_{\mathbf{R}_2} \text{prov}_3(i_3) \\ &\quad \wedge \text{req}_1(i_1) \leq_{\mathbf{R}_1} \text{prov}_2(i_2)\}. \end{aligned} \quad (36.15)$$

and

$$\begin{aligned} \text{prov}_{1,3} : [[i_1 ; i_2] ; i_3] &\mapsto \text{prov}_{1,2}([i_1 ; i_2]), \\ [i_1 ; i_2 ; i_3] &\mapsto \text{prov}_1(i_1), \\ \text{req}_{1,3} : [[i_1 ; i_2] ; i_3] &\mapsto \text{req}_3(i_3) \\ [i_1 ; i_2 ; i_3] &\mapsto \text{req}_3(i_3). \end{aligned} \quad (36.16)$$

By expanding the second part of the second term of (36.12), one has:

$$(\text{dp}_2 \circ \text{dp}_3) = \langle \mathbf{F}_2, \mathbf{R}_3, \mathbf{I}_{2,3}, \text{prov}_{2,3}, \text{req}_{2,3} \rangle, \quad (36.17)$$

Therefore, the full right term of (36.12) reads

$$\text{dp}_1 \circ (\text{dp}_2 \circ \text{dp}_3) = \langle \mathbf{F}_1, \mathbf{R}_3, \mathbf{I}'_{1,3}, \text{prov}'_{1,3}, \text{req}'_{1,3} \rangle, \quad (36.18)$$

with

$$\begin{aligned} \mathbf{I}'_{1,3} &= \{[i_1 ; [i_2 ; i_3]] \in (\mathbf{I}_1 \circ (\mathbf{I}_2 \circ \mathbf{I}_3)) \mid \text{req}_1(i_1) \leq_{\mathbf{R}_1} \text{prov}_{2,3}([i_2 ; i_3]) \\ &\quad \wedge \text{req}_2(i_2) \leq_{\mathbf{R}_2} \text{prov}_3(i_3)\} \\ &= \{[i_1 ; i_2 ; i_3] \in (\mathbf{I}_1 \circ \mathbf{I}_2 \circ \mathbf{I}_3) \mid \text{req}_1(i_1) \leq_{\mathbf{R}_1} \text{prov}_2(i_2) \\ &\quad \wedge \text{req}_2(i_2) \leq_{\mathbf{R}_2} \text{prov}_3(i_3)\} \end{aligned} \quad (36.19)$$

and

$$\begin{aligned} \text{prov}'_{1,3} : [\textcolor{brown}{i}_1 ; [\textcolor{brown}{i}_2 ; \textcolor{brown}{i}_3]] &\mapsto \text{prov}_1(\textcolor{brown}{i}_1) \\ &[\textcolor{brown}{i}_1 ; \textcolor{brown}{i}_2 ; \textcolor{brown}{i}_3] \mapsto \text{prov}_1(\textcolor{brown}{i}_1), \\ \text{req}'_{1,3} : [\textcolor{brown}{i}_1 ; [\textcolor{brown}{i}_2 ; \textcolor{brown}{i}_3]] &\mapsto \text{req}_{2,3}([\textcolor{brown}{i}_2 ; \textcolor{brown}{i}_3]) \\ &[\textcolor{brown}{i}_1 ; \textcolor{brown}{i}_2 ; \textcolor{brown}{i}_3] \mapsto \text{req}_3(\textcolor{brown}{i}_3) \end{aligned} \quad (36.20)$$

It is clear that $\mathbf{I}_{1,3} = \mathbf{I}'_{1,3}$, $\text{prov}_{1,3} = \text{prov}'_{1,3}$, and $\text{req}_{1,3} = \text{req}'_{1,3}$. This, together with (36.14) and (36.18) shows associativity. \square

These two properties are sufficient to conclude that there exists a semi-category of design problems.

Definition 36.22 (Semi-category **DPI**). There is a semi-category **DPI** where

- ▷ The objects are posets.
- ▷ The morphisms are **DPIs** $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \text{prov}, \text{req} \rangle$.
- ▷ Morphism composition is given by Def. 36.20.

Lemma 36.23. **DPI** is not a category, because we cannot find identities.

Proof. We prove this by contradiction. Suppose we can find a **DPI** that works as an identity for interconnection for any other **DPI**. Therefore, we have

$$\mathbf{I}_1 \circ \mathbf{I}_2 = \mathbf{I}_1. \quad (36.21)$$

This implies that \mathbf{I}_2 must be an empty list of sets, that is inhabited by only one element, the empty list of elements. Therefore, req_2 of the identity is necessarily a constant because there is an. \square

36.4. Co-design problems

A “co-design problem” will be defined as a multigraph of design problems.

Definition 36.24 (Co-design problem with implementation). A *Co-Design Problem with Implementation* (**CDPI**) is a tuple $\langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, where \mathbf{F} and \mathbf{R} are two posets, and $\langle \mathcal{V}, \mathcal{E} \rangle$ is a multigraph of **DPIs**. Each node $v \in \mathcal{V}$ is a **DPI** $v = \langle \mathbf{F}_v, \mathbf{R}_v, \mathbf{I}_v, \text{prov}_v, \text{req}_v \rangle$. An edge $e \in \mathcal{E}$ is a tuple $e = \langle \langle v_1, i_1 \rangle, \langle v_2, j_2 \rangle \rangle$, 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. 36.24).

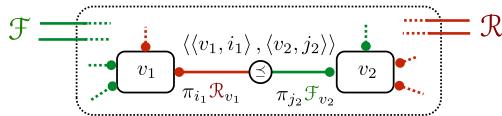


Figure 36.24.

A CDPI is equivalent to a **DPI** with an implementation space \mathbf{I} that is a subset of the product $\prod_{v \in \mathcal{V}} \mathbf{I}_v$, and contains only the tuples that satisfy the co-design constraints. An implementation tuple $i \in \prod_{v \in \mathcal{V}} \mathbf{I}_v$ belongs to \mathbf{I} iff it respects all functionality–resources constraints on the edges, in the sense that, for all edges $\langle \langle v_1, i_1 \rangle, \langle v_2, j_2 \rangle \rangle$ in \mathcal{E} , it holds that

$$\pi_{i_1} \text{req}_{v_1}(\pi_{v_1} i) \leq \pi_{j_2} \text{prov}_{v_2}(\pi_{v_2} i).$$

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 UF_v and 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: $\mathbf{F} = \prod_{v \in V} \prod_{j \in UF_v} \pi_j \mathbf{F}_v$ and $\mathbf{R} = \prod_{v \in V} \prod_{i \in UR_v} \pi_i \mathbf{R}_v$. The maps prov, req return the values of the unconnected functionality and resources:

$$\begin{aligned}\text{prov} : i &\mapsto \prod_{v \in V} \prod_{j \in UF_v} \pi_j \text{prov}_v(\pi_v i), \\ \text{req} : i &\mapsto \prod_{v \in V} \prod_{i \in UR_v} \pi_i \text{req}_v(\pi_v i).\end{aligned}$$

Example 36.25. The MCDP in Fig. 36.25 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. 36.26.

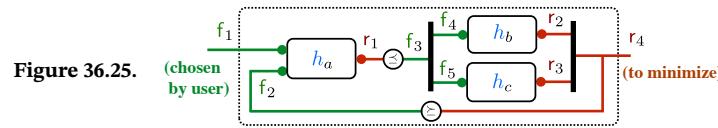


Figure 36.26. $f_1 \mapsto \left\{ \begin{array}{lll} \text{Min } r_4 & r_1 \in h_a(f_1, f_2) & r_1 \preceq f_3 \\ & r_2 \in h_b(f_4) & r_4 \preceq f_2 \\ & r_3 \in h_c(f_5) & r_4 = \langle r_2, r_3 \rangle \end{array} \right.$

Example 36.26. Consider the co-design of chassis (Example 36.4) plus motor (Example 36.3). 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. 36.27).

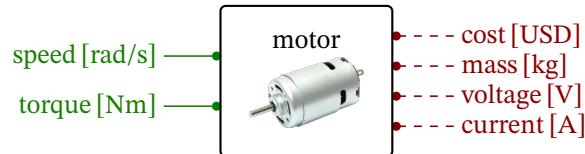


Figure 36.27.

For the chassis (Fig. 36.28), 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**.

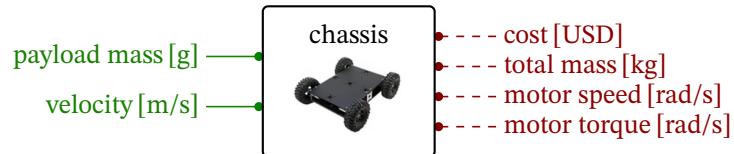


Figure 36.28.

The two design problem can be connected at the edges for torque and speed (Fig. 36.29). 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. 36.30).

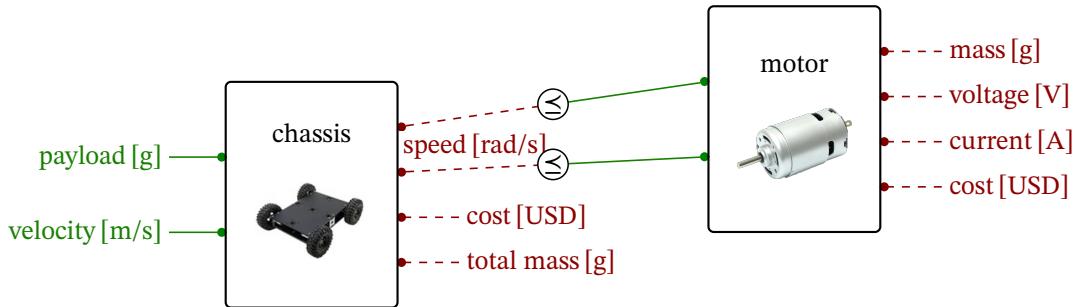


Figure 36.29.

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. 36.31).

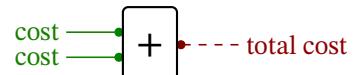


Figure 36.30.

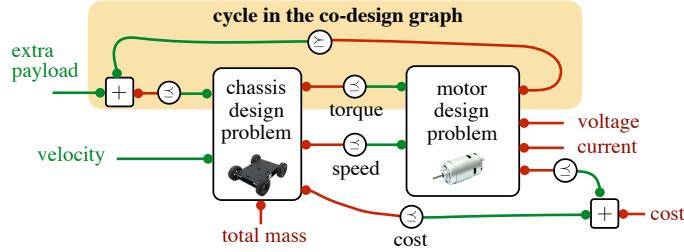
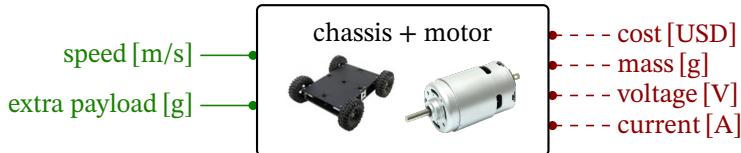


Figure 36.31.

This formalism makes it easy to abstract away the details in which we are not interested. Once a diagram like Fig. 36.31 is obtained, we can draw a box around it and consider the abstracted problem (??).



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. 36.33).

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. 36.34).

Relations such as **current × voltage ≤ power required** and **power × endurance ≤ energy required** can be modeled by a trivial “multiplication” DPI (Fig. 36.35).

We can connect these DPs to obtain a co-design problem with functionality **voltage**, **current**, **endurance** and resources **mass** and **cost** (Fig. 36.36).

Draw a box around the diagram, and call it “MCB+PSU”; then interconnect it with the “chassis+motor” diagram in Fig. 36.37.

We can further abstract away the diagram in Fig. 36.37 as a “mobility+power” CDPI, as in Fig. 36.38. 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 objec-

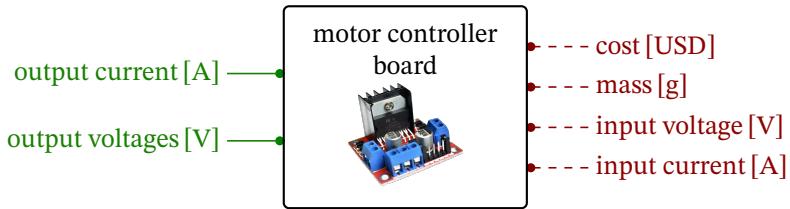


Figure 36.33.

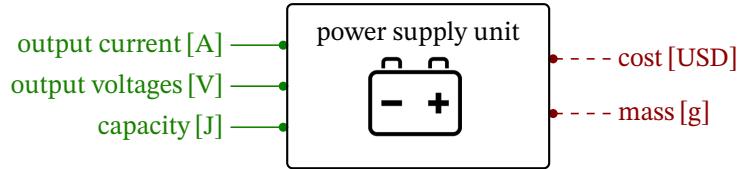


Figure 36.34.

tive. 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.

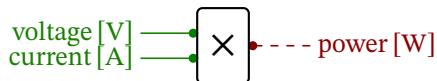


Figure 36.35.

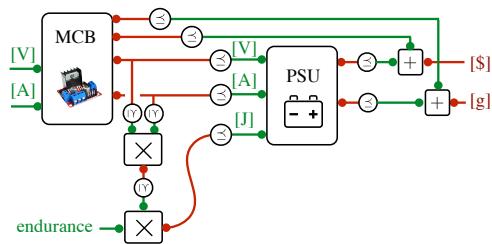


Figure 36.36.

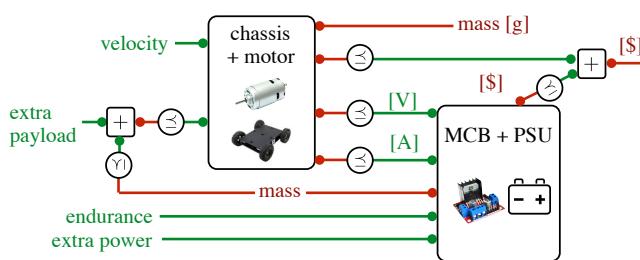


Figure 36.37.

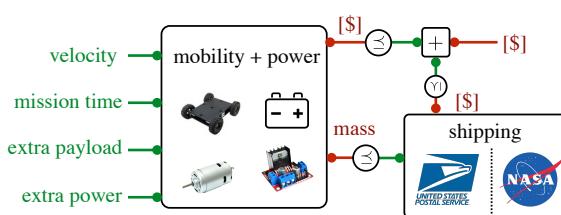


Figure 36.38.

36.5. Discussion of related work

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 [26, 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, (in other words, cycles in the co-design graph). Consider, as an example, Suh’s theory of *axiomatic design* [34], in which the first “axiom” is to keep the design requirements orthogonal (in other words, do not introduce cycles). This work shows that it is possible to deal elegantly with recursive constraints.

Partial Order Programming

In “Partial Order Programming” [27] 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 \mathbf{P} ($\mathbf{F}_v = \mathbf{R}_v = \mathbf{P}$); 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 ($\mathbf{F} = \mathbf{1}$).

In a MCDP, each DP is described by a Scott continuous map $h : \mathbf{F} \rightarrow \mathcal{AR}$ which maps one functionality to a minimal set of resources. By contrast, in a CMPOP an operator corresponds to a Scott continuous map $h : \mathbf{F} \rightarrow \mathbf{R}$. The consequence is that a CMPOP has a unique solution [27, Theorem 8], while an MCDP can have multiple minimal solutions (or none at all).

Abstract interpretation

The methods used from order theory are the same used in the field of *abstract interpretation* [7]. 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 (in other

words, a fixed point), and that is the smallest that contains the given initial state (in other words, a *least* fixed point). Reachability and other properties lead to considering systems of equation of the form

$$x_i = \varphi_i(x_1, \dots, x_i, \dots, x_n), \quad i = 1, \dots, n, \quad (36.22)$$

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 (36.22) does not matter. Asynchronous and "chaotic" iterations were proposed early [6] and are still object of investigation [2]. To speed up convergence, the so called "widening" and "narrowing" operators are used [5]. The ideas of chaotic iteration, widening, narrowing, are not immediately applicable to MCDPs, but it is a promising research direction.



37. Feasibility

37.1. Design problems as monotone maps

A DPI (Def. 36.1) 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”.

Recall how the problem Feasibility was defined in Section 36.2. Given a particular functionality f and resource r , we would like to know if they are feasible.

This is a function from $\mathbf{F} \times \mathbf{R}$ to \mathbf{Bool} :

$$f : \mathbf{F} \times \mathbf{R} \rightarrow \mathbf{Bool}. \quad (37.1)$$

The value $f(f^r)$ is the answer to the question “is the functionality f feasible with resources r ?” Due to how the problem is defined, we know that

1. If f is feasible with r , then any $f_2 \leq_{\mathbf{F}} f$ is feasible with r .
2. If f is feasible with r , then f is feasible with any $r_2 \geq_{\mathbf{R}} r$.

Therefore, we can conclude that f is monotone in the second argument r , and antitone in the first argument f .

37.1 Design problems as monotone maps	277
37.2 Querying design problems	280
37.3 Series composition of design problems	282
37.4 The category of design problems	285
37.5 DP Isomorphisms	286

Steinstossen is a sport in which the competitors need to throw a heavy stone as far away as possible. It was practiced among the alpine population since prehistoric times.

It is going to be convenient to have functions that are monotone, and not mixed monotone/antitone. Instead of considering a map from $\mathbf{F} \times \mathbf{R}$ to \mathbf{Bool} , we can turn things around and look at a map \mathbf{d} from $\mathbf{F}^{\text{op}} \times \mathbf{R}$ to \mathbf{Bool} , defined as $\mathbf{d}(\mathbf{x}, \mathbf{r}) = f(\mathbf{x}^*, \mathbf{r})$. Because we use \mathbf{F}^{op} rather than \mathbf{F} , the map \mathbf{d} is monotone.

The feasibility map \mathbf{d} has now forgotten everything about the implementations; however, it does contain all the information we need to solve co-design feasibility problems.

Therefore, we can define, DPs (design problems), now without implementation.

Definition 37.1 (Design Problem). A design problem (DP) is a tuple $\langle \mathbf{F}, \mathbf{R}, \mathbf{d} \rangle$, where \mathbf{F}, \mathbf{R} are posets and \mathbf{d} is a monotone map of the form

$$\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}.$$

We represent it by an arrow $\mathbf{d} : \mathbf{F} \rightarrow \mathbf{R}$.

Given a DPI $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}, \mathbf{prov}, \mathbf{req} \rangle$ it is always possible to obtain a DP

$$\begin{aligned} \mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} &\rightarrow_{\text{Pos}} \mathbf{Bool} \\ \langle \mathbf{f}^*, \mathbf{r} \rangle &\mapsto \exists \mathbf{i} \in \mathbf{I} : (\mathbf{f} \leq_{\mathbf{F}} \mathbf{prov}(\mathbf{i})) \wedge (\mathbf{req}(\mathbf{i}) \leq_{\mathbf{R}} \mathbf{r}). \end{aligned} \quad (37.2)$$

Evaluating the DP is the same as asking whether the set

$$\{ \mathbf{i} \in \mathbf{I} : (\mathbf{f} \leq_{\mathbf{F}} \mathbf{prov}(\mathbf{i})) \wedge (\mathbf{req}(\mathbf{i}) \leq_{\mathbf{R}} \mathbf{r}) \} \quad (37.3)$$

is empty or not.

Example 37.2. Recall Example 36.2, with the catalogue of electric motors in Table 37.1.

Table 37.1.: A simplified catalogue of motors.

Motor ID	Company	Torque [kg · cm]	Weight [g]	Max Power [W]	Cost [USD]
1204	SOYO	0.18	60.0	2.34	19.95
1206	SOYO	0.95	140.0	3.00	19.95
1207	SOYO	0.65	130.0	2.07	12.95
2267	SOYO	3.7	285.0	4.76	16.95
2279	Sanyo Denki	1.9	165.0	5.40	164.95
1478	SOYO	19.0	1,000	8.96	49.95
2299	Sanyo Denki	2.2	150.0	5.90	59.95

The catalogue induces a design problem \mathbf{d}_{EM} with diagrammatic form as in Fig. 37.1. In particular, we can query the design problem for combinations of **functionalities** and **resources**. For instance:

$$\mathbf{d} (0.2 \text{ kg} \cdot \text{cm}, \langle 50.0 \text{ g}, 2.0 \text{ W}, 15.0 \text{ USD} \rangle) = \perp, \quad (37.4)$$

since no listed model can provide $0.2 \text{ kg} \cdot \text{cm}$ torque by requiring the set of resources $\langle 50.0 \text{ g}, 2.0 \text{ W}, 15.0 \text{ USD} \rangle$ or less.

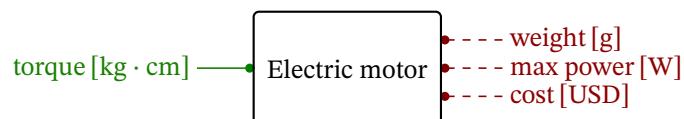


Figure 37.1.: Electric motor design problem.

Definition 37.3 (Feasible set of a design problem). We define the *feasible set* K_d of a design problem

$$d : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool} \quad (37.5)$$

as the subset of $\mathbf{F}^{\text{op}} \times \mathbf{R}$ for which d is the *indicator function*, that is

$$K_d = \{\langle f^*, r \rangle \in \mathbf{F}^{\text{op}} \times \mathbf{R} \mid d(f^*, r) = \top\}. \quad (37.6)$$

Remark 37.4. The set K_d is always an upper set (Def. 20.19). 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 perhaps 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. In any case, it is helpful to keep both of these perspectives in mind.

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, 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 $d : \mathbf{F} \rightarrow \mathbf{R}$ is represented as a box with functionality \mathbf{F} on the *left* and resources \mathbf{R} on the *right* (Fig. 37.2). As we did for DPIS, we

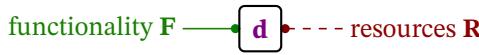


Figure 37.2.: Diagrammatic representation of a design problem.

will connect these diagrams.

Example 37.5. 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. 37.3.

Concretely, “engine” is represented as a monotone map

$$\text{engine} : \text{thrust}^{\text{op}} \times \text{fuel} \rightarrow_{\text{Pos}} \mathbf{Bool}. \quad (37.7)$$

Assuming that the posets **fuel**, **thrust**^{op} are finite, we can think of the “engine” design problem as a matrix, where each (i, j) -th entry is the answer to the question, “is the amount of thrust f_i feasible with the amount of fuel r_j ? ”:

		Fuel				
		$r_1 = 0$	r_2	r_3	...	r_m
Thrust ^{op}	$f_n^* = 0$	0	0	0		0
	f_{n-1}^*	0	0	0		1
	f_{n-2}^*	0	1	1		1
	f_1^*	1	1	1	..	1

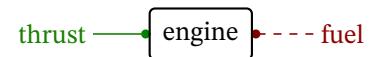
(37.8)


Figure 37.3.: Diagram of the engine design problem.

Suppose we have tested or are given the performance data of a few different

engines, as possible solutions to the “engine” design problem, each with a fixed optimal fuel-thrust value. To illustrate the monotonicity assumption, we can render the data of “engine” as a graph, as depicted in Fig. 37.4.

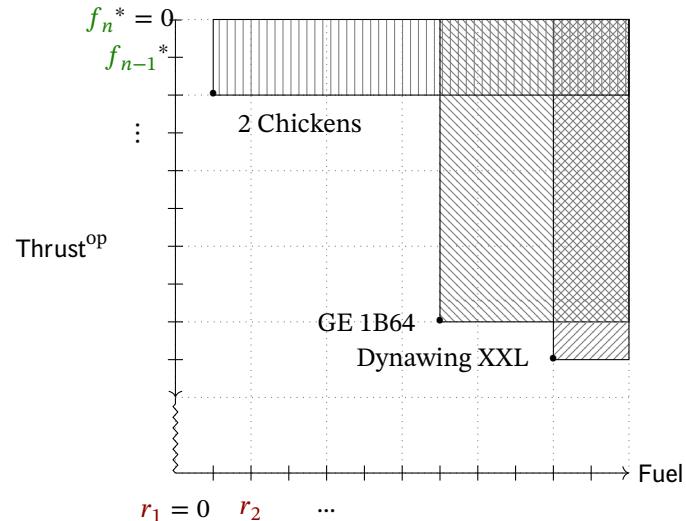


Figure 37.4.: Graphical representation of the possible solutions of the engine design problem.

Note that the shaded regions cover the feasible solution set. This feasible solution set is always an *upper set* (Def. 20.19) in $\text{thrust}^{\text{op}} \times \text{fuel}$, which is another way of characterizing the monotonicity of the design problem. The optimal solutions, indicated by dots, form an *antichain* of solutions. We will come back to antichains when discussing how to compute optimal solutions of design problems.

37.2. Querying design problems

(37.2) on the one hand, and (37.3) on the other hand, give to two perspectives on the mathematical definition of what we are calling a *design problem*. These two perspectives are analogous to something we already discussed in Remark 15.6, when talking about binary relations. There, we said that a binary relation from a set \mathbf{A} to a set \mathbf{B} is a subset $R \subseteq \mathbf{A} \times \mathbf{B}$, but that such a relation R can also, equivalently, be viewed as a function $\phi_R : \mathbf{A} \times \mathbf{B} \rightarrow \{\perp, \top\}$. The subset $R \subseteq \mathbf{A} \times \mathbf{B}$ corresponded to the set

$$\{\langle x, y \rangle \in \mathbf{A} \times \mathbf{B} \mid \phi_R(x, y) = \top\}. \quad (37.9)$$

To make the analogy with (37.6) more precise, note that $\mathbf{A}, \mathbf{B}, \{\perp, \top\}$, and $\phi_R : \mathbf{A} \times \mathbf{B} \rightarrow \{\perp, \top\}$ live in the category of *sets*, and that $\mathbf{F}, \mathbf{R}, \mathbf{Bool}$ and $\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ live in the category of *posets*.

In Remark 15.6, we also discussed two further ways to describe a relation $R \subseteq \mathbf{A} \times \mathbf{B}$: namely, we can transform the function $\phi_R : \mathbf{A} \times \mathbf{B} \rightarrow \{\perp, \top\}$ either into a function

$$\hat{\phi}_R : \mathbf{A} \rightarrow \mathcal{P}(\mathbf{B}), \quad \hat{\phi}_R(x) = \{y \in \mathbf{B} \mid \langle x, y \rangle \in R\}, \quad (37.10)$$

or a function

$$\check{\phi}_R : \mathbf{B} \rightarrow \mathcal{P}(\mathbf{A}), \quad \check{\phi}_R(y) = \{x \in \mathbf{A} \mid \langle x, y \rangle \in R\}. \quad (37.11)$$

There are analogous transformations for a design problem $\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$. Can you guess what they would be?

In order to use our “sets to posets” analogy and find an answer, it is useful to express the constructions we used in the setting of sets and relations entirely in terms of constructions from the category of sets, if possible. Then the strategy is to identify what are the analogous constructions in the category of posets, and this will allow us to make analogous definitions for design problems.

The functions $\hat{\phi}_{\mathbf{R}}$ and $\check{\phi}_{\mathbf{R}}$ above have powersets as their target objects. What is the analogue of the powerset operation in the category of posets?

The answer that we will use goes like this. Given a set \mathbf{A} , there is a 1-to-1 correspondence between subsets of \mathbf{A} and functions $\mathbf{A} \rightarrow \{\perp, \top\}$ (similar to above, a set corresponds here to its indicator function). Thus $\mathcal{P}(\mathbf{A})$ can be seen to correspond to $\text{Hom}_{\text{Set}}(\mathbf{A}; \{\perp, \top\})$. The latter is definitely an expression we can transfer, by analogy, to the category of posets, namely we can consider $\text{Hom}_{\text{Pos}}(\mathbf{P}; \mathbf{Bool})$. And from Graded exercise 3 we know that monotone maps $\mathbf{P} \rightarrow \mathbf{Bool}$ correspond to *upper* subsets of \mathbf{P} . So $\text{Hom}_{\text{Pos}}(\mathbf{P}; \mathbf{Bool})$ corresponds to set $\mathcal{U}(\mathbf{P})$ of upper subsets of \mathbf{P} (c.f. Section 20.5 for the definitions of upper and lower sets).

We now can write down the “poset” analogues of the functions $\hat{\phi}_{\mathbf{R}}$ and $\check{\phi}_{\mathbf{R}}$. Namely, given a design problem (37.5), we have associated functions

$$\hat{\mathbf{d}} : \mathbf{F}^{\text{op}} \rightarrow \mathcal{U}(\mathbf{R}) \quad (37.12)$$

and

$$\check{\mathbf{d}} : \mathbf{R} \rightarrow \mathcal{U}(\mathbf{F}^{\text{op}}). \quad (37.13)$$

However, we are not quite finished: are these monotone functions? Which poset structure can we choose on $\mathcal{U}(\mathbf{R})$ and $\mathcal{U}(\mathbf{F}^{\text{op}})$, respectively, so that $\hat{\mathbf{d}}$ and $\check{\mathbf{d}}$ are monotone?

Graded exercise 1 (CurryingDesignProblems). Let $\mathbf{d} : \mathbf{F}^{\text{op}} \times \mathbf{R} \rightarrow_{\text{Pos}} \mathbf{Bool}$ be a design problem. In this exercise we will show that (37.12) corresponds to a monotone function

$$\mathbf{F} \rightarrow (\mathcal{U}(\mathbf{R}), \supseteq), \quad (37.14)$$

and that (37.13) corresponds to a monotone function

$$\mathbf{R} \rightarrow (\mathcal{L}(\mathbf{F}), \subseteq). \quad (37.15)$$

Here $\mathcal{U}(\mathbf{R})$ denotes the set of upper sets of \mathbf{R} and $\mathcal{L}(\mathbf{F})$ denotes the set of lower sets of \mathbf{F} .

1. Show that $\hat{\mathbf{d}} : \mathbf{F}^{\text{op}} \rightarrow \mathcal{U}(\mathbf{R})$ and $\check{\mathbf{d}} : \mathbf{R} \rightarrow \mathcal{U}(\mathbf{F}^{\text{op}})$ are monotone maps when we consider $\mathcal{U}(\mathbf{R})$ and $\mathcal{U}(\mathbf{F}^{\text{op}})$ to have the partial order corresponding to the inclusion of subsets.
2. Show that the poset $(\mathcal{U}(\mathbf{F}^{\text{op}}), \subseteq)$ is isomorphic to the poset $(\mathcal{L}(\mathbf{F}), \subseteq)$.
3. Show that there is a 1-to-1 correspondence between monotone functions

$$\mathbf{F}^{\text{op}} \rightarrow (\mathcal{U}(\mathbf{R}), \subseteq) \quad (37.16)$$

and monotone functions

$$\mathbf{F} \rightarrow (\mathcal{U}(\mathbf{R}), \supseteq), \quad (37.17)$$

where in the latter poset, the order is given by the relation of “containment” (as opposed to “inclusion”).

4. Explain, in a few words, why the above steps prove the stated goal of this exercise.

37.3. 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 37.6 (Series composition). Let $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{g} : \mathbf{B} \rightarrow \mathbf{C}$ be design problems. We define their *series composition* $(\mathbf{f} ; \mathbf{g}) : \mathbf{A} \rightarrow \mathbf{C}$ as:

$$\begin{aligned} (\mathbf{f} ; \mathbf{g}) : \mathbf{A}^{\text{op}} \times \mathbf{C} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle \mathbf{a}^*, \mathbf{c} \rangle &\mapsto \bigvee_{b \in \mathbf{B}} \mathbf{f}(\mathbf{a}^*, b) \wedge \mathbf{g}(b^*, \mathbf{c}). \end{aligned} \quad (37.18)$$

Alternatively:

$$\begin{aligned} (\mathbf{f} ; \mathbf{g}) : \mathbf{A}^{\text{op}} \times \mathbf{C} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle \mathbf{a}^*, \mathbf{c} \rangle &\mapsto \bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in \mathbf{B}}} \mathbf{f}(\mathbf{a}^*, b_1) \wedge \mathbf{g}(b_2^*, \mathbf{c}). \end{aligned} \quad (37.19)$$

We represent series in the diagrammatic notation reported in Fig. 37.5.



Figure 37.5.: Diagrammatic representation of the series composition of design problems.

One can notice the “co-design constraint” \leq (already encountered in DPIs), which can be interpreted as follows. The **resource** required by a component is limited by the **functionality** produced by another component.

Remark 37.7. The series composition operations given in Eqs. (37.18) and (37.19) are equivalent.

First consider the direction (37.19) \implies (37.18). In order for

$$\bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in \mathbf{B}}} \mathbf{f}(\mathbf{a}^*, b_1) \wedge \mathbf{g}(b_2^*, \mathbf{c})$$

to be true, there should exist some $b_1 \leq b_2$ for which $\mathbf{f}(\mathbf{a}^*, b_1) \wedge \mathbf{g}(b_2^*, \mathbf{c})$ is true. However, due to the monotonicity of \mathbf{f} , $\mathbf{f}(\mathbf{a}^*, b_2) \wedge \mathbf{g}(b_2^*, \mathbf{c})$ and (37.18) must be true as well. On the other hand, if

$$\bigvee_{\substack{b_1 \leq b_2, b_1, b_2 \in \mathbf{B}}} \mathbf{f}(\mathbf{a}^*, b_1) \wedge \mathbf{g}(b_2^*, \mathbf{c})$$

is false, then due to the equality, it is false for any $b_1 = b_2$, and therefore all inner terms of (37.18) must be false as well.

The other direction, (37.18) \implies (37.19), can be shown in a similar way. If (37.18) is true, then there must exist a b' such that $\mathbf{f}(a^*, b') = \top$ and $\mathbf{g}(b'^*, c) = \top$. Then, the inner term in (37.19) is true for $b_1 = b_2 = b'$. If (37.18) is false, then there is no such b' for which both $\mathbf{f}(a^*, b')$ and $\mathbf{g}(b'^*, c)$ are true, but then due to the monotonicity of \mathbf{f} and \mathbf{g} they also cannot be true for any $b_1 \leq b_2 = b'$ or $b' = b_1 \leq b_2$. Hence, (37.19) must also be false.

Remark 37.8. At first sight, (37.19) might seem like a more verbose version of (37.18). However, assume that we have the means to obtain the minimal antichain of the feasible set of resources that provide \mathbf{A} for the first term:

$$\mathbf{B}_f = \text{Min}\{b_1 \in \mathbf{B} \mid \mathbf{f}(a^*, b_1) = \top\} \in \mathcal{A}\mathbf{B}.$$

This represents the minimal resources with which \mathbf{f} can provide \mathbf{A} . Assume further that we similarly have the means to obtain the maximal antichain of the feasible set of functionalities that \mathbf{c} provides for the second term

$$\mathbf{B}_g = \text{Max}\{b_2 \in \mathbf{B} \mid \mathbf{g}(b_2^*, c) = \top\} \in \mathcal{A}\mathbf{B},$$

which represents the maximal functionality that \mathbf{g} can provide given \mathbf{c} . Then, (37.19) implies that it suffices to only evaluate

$$\bigvee_{\substack{b_1 \leq b_2 \\ b_1 \in \mathbf{B}_f, b_2 \in \mathbf{B}_g}} \mathbf{f}(a^*, b_1) \wedge \mathbf{g}(b_2^*, c),$$

which can be much more efficient than iterating over all $b \in \mathbf{B}$.

Intended semantics The series composition $(\mathbf{f} ; \mathbf{g})$ judges a pair $\langle a^*, c \rangle$ as feasible if and only if there exists a $b \in \mathbf{B}$ such that $\mathbf{f}(a^*, b)$ and $\mathbf{g}(b^*, c)$ are feasible.[†]

Example 37.9. After the Bucket of Boom X100 blew upon re-entry, Jeb's Space-ship 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. 37.6).

Remark 37.10. 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,

[†] In (37.18) 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 $s(i_0) \leq \bigvee_{i \in I} s(i)$. 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

$$\bigvee_{i \in I} (b \wedge s(i)) = b \wedge \left(\bigvee_{i \in I} s(i) \right) \quad \text{and} \quad \bigvee_{(i,j) \in I \times J} (s(i) \wedge t(j)) = \left(\bigvee_{i \in I} s(i) \right) \wedge \left(\bigvee_{j \in J} t(j) \right).$$

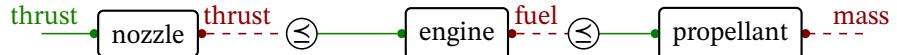


Figure 37.6.: Example of composition.

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 \mathbf{f} and \mathbf{g} , their series composition $(\mathbf{f} ; \mathbf{g})$ is in fact a design problem.

Lemma 37.11. Series composition as in (37.18) is monotone in a and c .

Proof. We need to show that $[(\mathbf{f} ; \mathbf{g})](\mathbf{a}^*, \mathbf{c})$ is monotone in \mathbf{a}^* and \mathbf{c} . Because \mathbf{f} represents a design problem, $\mathbf{f}(\mathbf{a}^*, \mathbf{b})$ is monotone in \mathbf{a}^* , and similarly $\mathbf{g}(\mathbf{b}^*, \mathbf{c})$ is monotone in \mathbf{c} . The conjunction “ \wedge ” is monotone in both variables, and likewise the “ \vee ” operation. \square

We can show two important properties for the “ \circ ” operation: associativity and unitality.

Lemma 37.12. The series composition operation as in (37.18) is associative:

$$(\mathbf{f} ; \mathbf{g}) ; \mathbf{h} = \mathbf{f} ; (\mathbf{g} ; \mathbf{h}). \quad (37.20)$$

Proof. Consider $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$, $\mathbf{g} : \mathbf{B} \rightarrow \mathbf{C}$, $\mathbf{h} : \mathbf{C} \rightarrow \mathbf{D}$. To show that the operation is associative, we can use distributivity and commutativity in **Bool**:

$$\begin{aligned} (((\mathbf{f} ; \mathbf{g}) ; \mathbf{h})(\mathbf{a}^*, \mathbf{d})) &= \bigvee_{c \in \mathbf{C}} \left(\bigvee_{b \in \mathbf{B}} \mathbf{f}(\mathbf{a}^*, \mathbf{b}) \wedge \mathbf{g}(\mathbf{b}^*, \mathbf{c}) \right) \wedge \mathbf{h}(\mathbf{c}^*, \mathbf{d}) \\ &= \bigvee_{c \in \mathbf{C}} \left(\bigvee_{b \in \mathbf{B}} \mathbf{f}(\mathbf{a}^*, \mathbf{b}) \wedge \mathbf{g}(\mathbf{b}^*, \mathbf{c}) \wedge \mathbf{h}(\mathbf{c}^*, \mathbf{d}) \right) \\ &= \bigvee_{b \in \mathbf{B}} \mathbf{f}(\mathbf{a}^*, \mathbf{b}) \wedge \left(\bigvee_{c \in \mathbf{C}} \mathbf{g}(\mathbf{b}^*, \mathbf{c}) \wedge \mathbf{h}(\mathbf{c}^*, \mathbf{d}) \right) \\ &= (\mathbf{f} ; (\mathbf{g} ; \mathbf{h}))(\mathbf{a}^*, \mathbf{d}). \end{aligned} \quad (37.21)$$

\square

Because of associativity, we can write $\mathbf{f} ; \mathbf{g} ; \mathbf{h}$ without ambiguity. Associativity of composition is represented as in Fig. 37.7.

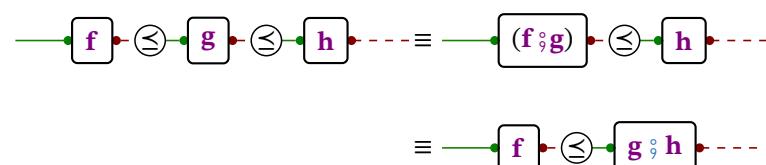


Figure 37.7.: Series composition is associative.

There exists an identity for the “ \circ ” operation. We define the identity $\text{id}_{\mathbf{A}} : \mathbf{A} \rightarrow \mathbf{A}$ as follows.

Definition 37.13 (Identity design problem). For any poset \mathbf{A} , the *identity design problem* $\text{id}_{\mathbf{A}} : \mathbf{A} \rightarrow_{\text{Pos}} \mathbf{Bool}$ is a monotone map

$$\begin{aligned} \text{id}_{\mathbf{A}} : \mathbf{A}^{\text{op}} \times \mathbf{A} &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle a_1^*, a_2 \rangle &\mapsto a_1 \leq_{\mathbf{A}} a_2. \end{aligned} \tag{37.22}$$

In the diagrammatic notation, we represent $\text{id}_{\mathbf{A}}$ as in Fig. 37.8.



Figure 37.8.: Diagrammatic representation of the identity design problem.

Lemma 37.14. The series composition operation as in (37.18) satisfies the left and right unit laws (Fig. 37.9).

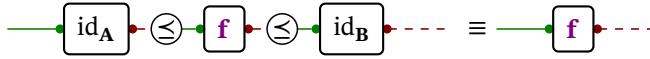


Figure 37.9.: Composition satisfies left and right unit laws.

Proof. Given $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$, we need to show:

$$\text{id}_{\mathbf{A}} ; \mathbf{f} = \mathbf{f} = \mathbf{f} ; \text{id}_{\mathbf{B}}.$$

In the following, we prove $\text{id}_{\mathbf{A}} ; \mathbf{f} = \mathbf{f}$. Proving $\mathbf{f} ; \text{id}_{\mathbf{B}} = \mathbf{f}$ is similar. Consider the poset \mathbf{Bool} . Since for $x, y \in \mathbf{Bool}$, $x \cong y \Rightarrow x = y$ (also referred to as skeletal [11]), we just need to show that $\mathbf{f} \leq \text{id}_{\mathbf{A}} ; \mathbf{f}$ and $\text{id}_{\mathbf{A}} ; \mathbf{f} \leq \mathbf{f}$. We have

$$\begin{aligned} \mathbf{f}(a^*, b) &= \top \wedge \mathbf{f}(a^*, b) \\ &\leq \text{id}_{\mathbf{A}}(a^*, a) \wedge \mathbf{f}(a^*, b) \\ &\leq \bigvee_{a' \in \mathbf{A}} \text{id}_{\mathbf{A}}(a^*, a') \wedge \mathbf{f}(a'^*, b) \\ &= (\text{id}_{\mathbf{A}} ; \mathbf{f})(a^*, b). \end{aligned}$$

For the other direction, we need to show that $\text{id}_{\mathbf{A}} ; \mathbf{f} \leq \mathbf{f}$:

$$\bigvee_{a' \in \mathbf{A}} \text{id}_{\mathbf{A}}(a^*, a') \wedge \mathbf{f}(a'^*, b) \leq \mathbf{f}(a^*, b).$$

This holds if and only if $\text{id}_{\mathbf{A}}(a^*, a') \wedge \mathbf{f}(a'^*, b) \leq \mathbf{f}(a^*, b)$ for some $a' \in \mathbf{A}$. 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 $\text{id}_{\mathbf{A}}(a^*, a') = \top$ and $\mathbf{f}(a'^*, b) = \top$. We know that $\text{id}_{\mathbf{A}}(a^*, a') = \top \Leftrightarrow \mathbf{A} \leq a'$, and hence $\mathbf{f}(a^*, b) = \top$. \square

37.4. The category of design problems

We will show that the class of all design problems forms a category, which we call \mathbf{DP} .

Definition 37.15 (Category of design problems). The *category of design problems*, \mathbf{DP} , consists of the following constituents:

1. *Objects:* The objects of \mathbf{DP} are posets.

2. *Morphisms*: The morphisms of \mathbf{DP} are design problems (Def. 37.1).
3. *Identity morphism*: The identity morphism $\text{Id}_A : A \leftrightarrow A$ is given by Def. 37.13.
4. *Composition operation*: Given two morphisms $f : A \leftrightarrow B$ and $g : B \leftrightarrow C$, their composition $f ; g : A \leftrightarrow C$ is given by Def. 37.6.

We have already shown that the composition operator “” is associative and unital, and that the composition of two design problems is a design problem (closure). Therefore, \mathbf{DP} is a category.

Remark 37.16. \mathbf{DP} is called **Feas** or **Prof_{Bool}** in [11].

37.5. DP Isomorphisms



38. Profunctors

In this chapter we introduce the notion of profunctors. We have already seen a special case of profunctors, the Boolean profunctors.

38.1 Profunctors	287
38.2 Hom Profunctor	288
38.3 Other examples of profunctors	289
38.4 The bicategory of profunctors	289
38.5 DPI as profunctors	289

38.1. Profunctors

We recall the definition of boolean profunctors:

Definition 38.1 (Boolean profunctors). Given two posets \mathbf{P} and \mathbf{Q} , a *boolean profunctor* from \mathbf{P} to \mathbf{Q} is a monotone function of the form

$$F : \mathbf{P}^{\text{op}} \times \mathbf{Q} \rightarrow_{\text{Pos}} \mathbf{Bool}. \quad (38.1)$$

This is also written as

$$F : \mathbf{P} \leftrightarrow \mathbf{Q}. \quad (38.2)$$

We are going to extend this to general profunctors:

- ▷ Instead of posets, we will have arbitrary categories.
- ▷ Instead of **Bool**, we will have **Set**.
- ▷ Instead of monotone functions, we will have functors.

Definition 38.2 (Profunctors). Given two categories \mathbf{C} and \mathbf{D} , a *profunctor* from \mathbf{C} to \mathbf{D} is a functor of the form

$$\mathbf{F} : \mathbf{C}^{\text{op}} \times \mathbf{D} \rightarrow_{\text{Cat}} \mathbf{Set}. \quad (38.3)$$

This is also written as

$$\mathbf{F} : \mathbf{C} \nrightarrow \mathbf{D}. \quad (38.4)$$

Definition 38.3 (Profunctor composition). Given two profunctors $\mathbf{F} : \mathbf{C} \nrightarrow \mathbf{D}$ and $\mathbf{G} : \mathbf{D} \nrightarrow \mathbf{E}$ we can define their composition $(\mathbf{F} ; \mathbf{G}) : \mathbf{C} \nrightarrow \mathbf{E}$ as follows:

$$\begin{aligned} (\mathbf{F} ; \mathbf{G})_{\text{ob}} &: \text{Ob}(\mathbf{C}^{\text{op}} \times \mathbf{E}) \rightarrow \text{Ob } \mathbf{Set}, \\ \langle \mathbf{C}^*, \mathbf{E} \rangle &\mapsto \coprod_{\mathbf{D} \in \text{Ob} \mathbf{D}} \mathbf{F}_{\text{ob}}(\mathbf{C}^*, \mathbf{D}) \times \mathbf{G}_{\text{ob}}(\mathbf{D}^*, \mathbf{E}) / \sim \end{aligned} \quad (38.5)$$

$$\begin{aligned} (\mathbf{F} ; \mathbf{G})_{\text{mor}} &: \text{Hom}_{(\mathbf{C}^{\text{op}} \times \mathbf{E})}((\mathbf{C}_1^*, \mathbf{E}_1); (\mathbf{C}_2^*, \mathbf{E}_2)) \rightarrow \text{Hom}_{\mathbf{Set}}((\mathbf{F} ; \mathbf{G})_{\text{ob}}(\mathbf{C}_1^*, \mathbf{E}_1); (\mathbf{F} ; \mathbf{G})_{\text{ob}}(\mathbf{C}_2^*, \mathbf{E}_2)) \\ \langle \alpha^*, \beta \rangle &\mapsto \begin{cases} (\mathbf{F} ; \mathbf{G})_{\text{ob}}(\mathbf{C}_1^*, \mathbf{E}_1) \rightarrow (\mathbf{F} ; \mathbf{G})_{\text{ob}}(\mathbf{C}_2^*, \mathbf{E}_2) \\ \langle s, t \rangle \mapsto \langle \mathbf{F}_{\text{mor}}(\langle \alpha, \text{Id}_{\mathbf{D}} \rangle)(s), \mathbf{G}_{\text{mor}}(\langle \text{Id}_{\mathbf{D}}^*, \beta \rangle)(t) \rangle \end{cases} \end{aligned} \quad (38.6)$$

In the formulas:

$$\alpha : \mathbf{C}_2 \rightarrow \mathbf{C}_1, \quad \beta : \mathbf{E}_1 \rightarrow \mathbf{E}_2, \quad (38.7)$$

and $\langle s, t \rangle$ is a pair of elements for which there exists a $\mathbf{D} \in \text{Ob} \mathbf{D}$ such that

$$s \in \mathbf{F}_{\text{ob}}(\mathbf{C}_1^*, \mathbf{D}), \quad t \in \mathbf{G}_{\text{ob}}(\mathbf{D}^*, \mathbf{E}). \quad (38.8)$$

Unfortunately the composition is not associative, hence profunctors do not form a category.

38.2. Hom Profunctor

In this section, we will see that given any category \mathbf{C} , its set of morphisms Hom can be seen as a profunctor of the form:

$$\text{Hom}_{\mathbf{C}} : \mathbf{C}^{\text{op}} \times \mathbf{C} \rightarrow_{\text{Cat}} \mathbf{Set} \quad (38.9)$$

First, we need to specify how this profunctor acts on objects and on morphisms. It maps any two objects $\mathbf{X}, \mathbf{Y} \in \mathbf{C}$ to their Hom -sets:

$$\text{Hom}_{\mathbf{C}}(\langle \mathbf{X}^*, \mathbf{Y} \rangle) = \text{Hom}_{\mathbf{C}}(\mathbf{X}; \mathbf{Y}) \quad (38.10)$$

The action on the morphisms is more involved. First of all, a morphism in $\mathbf{C}^{\text{op}} \times \mathbf{C}$ has signature

$$f : \langle \mathbf{X}^*, \mathbf{Y} \rangle \rightarrow \langle \mathbf{Z}^*, \mathbf{W} \rangle \quad (38.11)$$

In particular, f can be written as $\langle f_1^{\text{op}}, f_2 \rangle$ with $f_1^{\text{op}} : X^* \rightarrow Z^*$ and $f_2 : Y \rightarrow W$. Now, the profunctor maps each morphism $f : \langle X^*, Y \rangle \rightarrow \langle Z^*, W \rangle$ to a morphism

$$\text{Hom}_{\mathbf{C}}(f) : \text{Hom}_{\mathbf{C}}(X; Y) \rightarrow \text{Hom}_{\mathbf{C}}(Z; W). \quad (38.12)$$

More specifically, one has:

$$\begin{array}{c} f_1 : Z \rightarrow X \quad f_2 : Y \rightarrow W \quad g : X \rightarrow Y \\ \hline (f_1 ; g ; f_2) : Z \rightarrow W \end{array} \quad (38.13)$$

Therefore, the functor maps each morphism $f = \langle f_1^{\text{op}}, f_2 \rangle \in \text{Hom}_{\mathbf{C}}(X; Y)$ to the morphism in **Set** given by:

$$\begin{aligned} \text{Hom}_{\mathbf{C}}(f) : \text{Hom}_{\mathbf{C}}(X; Y) &\rightarrow \text{Hom}_{\mathbf{C}}(Z; W) \\ g &\mapsto (f_1 ; g ; f_2). \end{aligned} \quad (38.14)$$

We now need to check that this indeed forms a profunctor. First of all, we want to check

$$\text{Hom}_{\mathbf{C}}(f ; g) = \text{Hom}_{\mathbf{C}}(f) ; \text{Hom}_{\mathbf{C}}(g). \quad (38.15)$$

Let's consider any $f = \langle f_1^{\text{op}}, f_2 \rangle$ and $g = \langle g_1^{\text{op}}, g_2 \rangle$ in $\mathbf{C}^{\text{op}} \times \mathbf{C}$. We know that

$$\begin{aligned} f ; g &= \langle f_1^{\text{op}} ; g_1^{\text{op}}, f_2 ; g_2 \rangle \\ &= \langle (g_1 ; f_1)^{\text{op}}, f_2 ; g_2 \rangle. \end{aligned} \quad (38.16)$$

Therefore, one can write

$$\begin{aligned} \text{Hom}_{\mathbf{C}}(f ; g)(z) &= (g_1 ; f_1) ; z ; (f_2 ; g_2) \\ &= g_1 ; (f_1 ; z ; f_2) ; g_2 \\ &= (f_1 ; z ; f_2) ; \text{Hom}_{\mathbf{C}}(g) \\ &= (\text{Hom}_{\mathbf{C}}(f) ; \text{Hom}_{\mathbf{C}}(g))(z). \end{aligned} \quad (38.17)$$

38.3. Other examples of profunctors

38.4. The bicategory of profunctors

38.5. DPI as profunctors



39. Parallelism

In this chapter we add some structure to the definition of a poset, by introducing *monoidal posets* and *monoidal categories*.

39.1. Modeling parallelism

39.2. Monoidal categories

39.1 Modeling parallelism	291
39.2 Monoidal categories	291
39.3 DP is a monoidal category	298
DP is a symmetric monoidal category	300
39.4 Pre-monoidal categories	302
39.5 Monoidal functors	302
39.6 Dualizable objects	303

Definition 39.1 (Monoidal poset). A *monoidal structure* on a poset $\mathbf{P} = \langle \mathbf{P}, \leq_{\mathbf{P}} \rangle$ is specified by:

Constituents

1. A monotone map $\otimes : \mathbf{P} \times \mathbf{P} \rightarrow \mathbf{P}$, called the *monoidal product*.
Note that here we are implicitly assuming $\mathbf{P} \times \mathbf{P}$ as having the product order. In detail, monotonicity means that, for all $x_1, x_2, y_1, y_2 \in \mathbf{P}$:

$$x_1 \leq_{\mathbf{P}} y_1 \text{ and } x_2 \leq_{\mathbf{P}} y_2 \implies (x_1 \otimes x_2) \leq_{\mathbf{P}} (y_1 \otimes y_2).$$

2. An element $\mathbf{1} \in \mathbf{P}$, called the *monoidal unit*.

Raclette is a Swiss dish, based on heating cheese and scraping off the melted part. In Switzerland, raclette is typically served with potatoes, cornichons, pickled onions, and Fendant wine.

Conditions

1. Associativity: for all $x, y, z \in P$:

$$(x \otimes y) \otimes z = x(\otimes y \otimes z).$$

2. Left and right unitality: for all $x \in P$:

$$\mathbf{1} \otimes x = x \quad \text{and} \quad x \otimes \mathbf{1} = x.$$

A poset equipped with a monoidal structure is called a *monoidal poset*.

Definition 39.2 (Symmetric monoidal poset). A *symmetric monoidal poset* is a monoidal poset $\langle P, \leq_P, \otimes, \mathbf{1} \rangle$ such that

$$x \otimes y = y \otimes x$$

for all $x, y \in P$.

Example 39.3. 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 39.1 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 39.4. Someone proposes now to substite the monoidal unit in Example 39.3 with 1 and the monoidal product with multiplication “ \cdot ”. This does not form a monoidal poset anymore. To see a simple counterexample, consider the fact that $-5 \leq 0$ and $-4 \leq 3$. However, $(-5) \cdot (-4) \not\leq 0 \cdot 3$.

Example 39.5. 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, $x \wedge \top = x = \top \wedge x$.

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 10.2.

Definition 39.6 (Monoidal category). A *monoidal structure* on a category \mathbf{C} is specified by:

Constituents

1. A functor $\otimes : \mathbf{C} \times \mathbf{C} \rightarrow \mathbf{C}$, called the *monoidal product*.
2. An object $\mathbf{1} \in \text{Ob}_{\mathbf{C}}$, called the *monoidal unit*.
3. A natural isomorphism, called the *associator*, whose components are of the type

$$\text{as}_{X,Y,Z} : (X \otimes Y) \otimes Z \xrightarrow{\cong} X \otimes (Y \otimes Z) \quad X, Y, Z \in \text{Ob}_{\mathbf{C}}.$$

4. A natural isomorphism, called the *left unitor*, whose components are of the type

$$\text{lu}_X : \mathbf{1} \otimes X \xrightarrow{\cong} X \quad X \in \text{Ob}_{\mathbf{C}}.$$

5. A natural isomorphism, called the *right unitor*, whose components are of the type

$$\text{ru}_X : X \otimes \mathbf{1} \xrightarrow{\cong} X \quad X \in \text{Ob}_{\mathbf{C}}.$$

Conditions

For all $X, Y, Z, W \in \text{Ob}_{\mathbf{C}}$, the following diagrams must commute:

1. Triangle identities.

$$\begin{array}{ccc} (X \otimes \mathbf{1}) \otimes Y & \xrightarrow{\text{as}_{X,\mathbf{1},Y}} & X \otimes (\mathbf{1} \otimes Y) \\ \text{ru}_X \otimes \mathbf{1} \swarrow & & \downarrow \mathbf{1} \otimes \text{lu}_Y \\ X \otimes Y & & \end{array}$$

2. Pentagon identities.

$$\begin{array}{ccccc} & & (X \otimes Y) \otimes (Z \otimes W) & & \\ & \nearrow \text{as}_{X \otimes Y, Z, W} & & \searrow \text{as}_{X, Y, Z \otimes W} & \\ ((X \otimes Y) \otimes Z) \otimes W & & & & (X \otimes (Y \otimes (Z \otimes W))) \\ \downarrow \text{as}_{X, Y, Z} \otimes \text{Id}_W & & & & \uparrow \text{Id}_X \otimes \text{as}_{Y, Z, W} \\ (X \otimes (Y \otimes Z)) \otimes W & \xrightarrow{\text{as}_{X, Y \otimes Z, W}} & & & X \otimes ((Y \otimes Z) \otimes W) \end{array}$$

A category equipped with a monoidal structure is called a *monoidal category*. If the components of the associator, left unitor, and right unitor are all equalities, one calls the category *strict monoidal*.

Example 39.7. Let's digest the definition of monoidal category with an explanatory example. We consider the structure $\langle \mathbf{Set}, \times, \{\cdot\} \rangle$ 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 **Set** (it is a functor). Given $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\text{Set}}$, $\mathbf{A} \times \mathbf{B}$ is the cartesian product of sets, and given $f : \mathbf{A} \rightarrow \mathbf{A}'$, $g : \mathbf{B} \rightarrow \mathbf{B}'$, we have:

$$(f \times g) : \mathbf{A} \times \mathbf{B} \xrightarrow{\cong} \mathbf{A}' \times \mathbf{B}'$$

$$\langle a, b \rangle \mapsto \langle f(a), g(b) \rangle.$$

Furthermore, given any $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \text{Ob}_{\text{Set}}$, we specify the associator $\text{as}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$:

$$\text{as}_{\mathbf{A}, \mathbf{B}, \mathbf{C}} : (\mathbf{A} \times \mathbf{B}) \times \mathbf{C} \rightarrow \mathbf{A} \times (\mathbf{B} \times \mathbf{C})$$

$$\langle \langle a, b \rangle, c \rangle \mapsto \langle a, \langle b, c \rangle \rangle$$

This defines an isomorphism (I can go “back and forth”, by switching the tuple separation). We now need to check whether as is natural. We check this graphically, using the commutative diagram in Fig. 39.1.

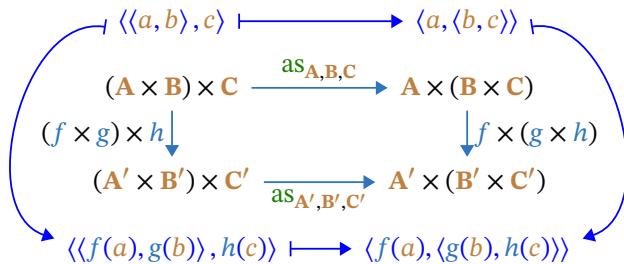


Figure 39.1.

Given an object $\mathbf{A} \in \text{Ob}_{\text{Set}}$, the unitor $\text{lu}_{\mathbf{A}}$ is given by:

$$\text{lu}_{\mathbf{A}} : \{\bullet\} \times \mathbf{A} \xrightarrow{\cong} \mathbf{A}$$

$$\langle \bullet, a \rangle \mapsto a.$$

Again, this defines an isomorphism. Consider a morphism $f : \mathbf{A} \rightarrow \mathbf{A}'$. We now prove naturality graphically (Fig. 39.2).

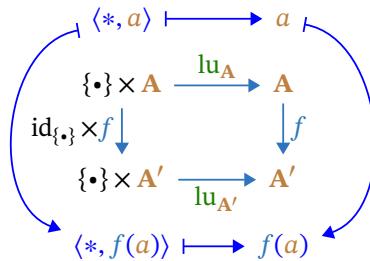


Figure 39.2.

Analogously, given an object $\mathbf{A} \in \text{Ob}_{\text{Set}}$, the unitor isomorphism $\text{ru}_{\mathbf{A}}$ is given by:

$$\text{ru}_{\mathbf{A}} : \mathbf{A} \times \{\bullet\} \xrightarrow{\cong} \mathbf{A}$$

$$\langle a, \bullet \rangle \mapsto a.$$

The proof for naturality is analogous to the one of $\text{lu}_{\mathbf{A}}$. We now need to check

whether the triangle and pentagon identities are satisfied. We start by the triangle. Given $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\text{Set}}$, the proof is displayed in Fig. 39.3.

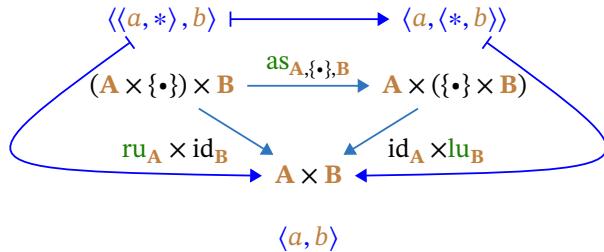


Figure 39.3.

We now prove the pentagon identity. Given $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \in \text{Ob}_{\text{Set}}$, the proof is reported in Fig. 39.4.

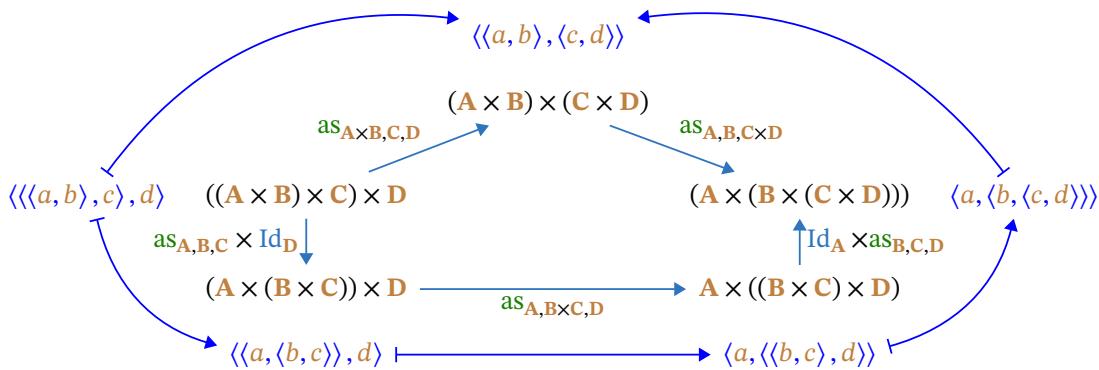


Figure 39.4.

Example 39.8. The category $\mathbf{Vect}_{\mathbb{R}}$ is can be equipped with a monoidal structure where the monoidal product is the tensor product of real vector spaces. It can also be equipped with a different monoidal structure where the monoidal product is the direct sum of real vector spaces.

Graded exercise 2 (`VectTensorMonStructure`). What are straightforward choices of monoidal unit, associator, and left/right unitors which, together with the tensor product as monoidal product, equip $\mathbf{Vect}_{\mathbb{R}}$ with a monoidal structure?

In this exercise, simply write down how you think each of these pieces of data would be defined – it is not asked that you prove that they do indeed form a monoidal structure (that would be much more involved).

Example 39.9. 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. 39.5). 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.

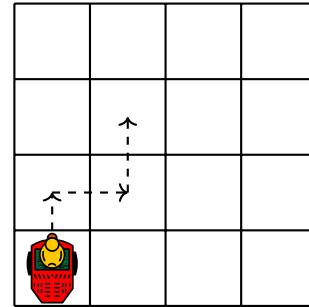


Figure 39.5.: Example of the robot category.

Definition 39.10 (Braided monoidal category). A *braided monoidal category* is a monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1} \rangle$, cf. Definition 39.6, equipped with a *braiding*, which is specified by

Constituents

1. A natural isomorphism br , called the braiding, whose components are of the type

$$\text{br}_{X,Y} : (X \otimes Y) \xrightarrow{\cong} (Y \otimes X), \quad X, Y \in \text{Ob}_{\mathbf{C}}. \quad (39.1)$$

Explicitly, this means that for any morphisms $f_1 : X_1 \rightarrow Y_1$ and $f_2 : X_2 \rightarrow Y_2$, the following diagram

$$\begin{array}{ccc} X_1 \otimes X_2 & \xrightarrow{f_1 \otimes f_2} & Y_1 \otimes Y_2 \\ \text{br}_{X_1, X_2} \downarrow & & \downarrow \text{br}_{Y_1, Y_2} \\ X_2 \otimes X_1 & \xrightarrow{f_2 \otimes f_1} & Y_2 \otimes Y_1 \end{array}$$

commutes.

Conditions

1. *Hexagon identities*: Given any objects $X, Y, Z \in \text{Ob}_{\mathbf{C}}$, the following diagrams must commute.

$$\begin{array}{ccccc} (X \otimes Y) \otimes Z & \xrightarrow{\text{br}_{X,Y} \otimes \text{Id}_Z} & (Y \otimes X) \otimes Z & \xrightarrow{\text{as}_{Y,X,Z}} & Y \otimes (X \otimes Z) \\ \text{as}_{X,Y,Z} \downarrow & & & & \downarrow \text{Id}_Y \otimes \text{br}_{X,Z} \\ X \otimes (Y \otimes Z) & \xrightarrow{\text{br}_{X,Y \otimes Z}} & (Y \otimes Z) \otimes X & \xrightarrow{\text{as}_{Y,Z,X}} & Y \otimes (Z \otimes X) \\ \\ X \otimes (Y \otimes Z) & \xrightarrow{\text{Id}_X \otimes \text{br}_{Y,Z}} & X \otimes (Z \otimes Y) & \xrightarrow{\text{as}_{Y,X,Z}^{-1}} & (X \otimes Z) \otimes Y \\ \text{as}_{X,Y,Z}^{-1} \downarrow & & & & \downarrow \text{br}_{X,Z} \otimes \text{Id}_Y \\ (X \otimes Y) \otimes Z & \xrightarrow{\text{br}_{X \otimes Y, Z}} & Z \otimes (X \otimes Y) & \xrightarrow{\text{as}_{Z,X,Y}^{-1}} & (Z \otimes X) \otimes Y \end{array}$$

Remark 39.11. If $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{br} \rangle$ is a braided monoidal category, one can show that the diagram

$$\begin{array}{ccc} \mathbf{1} \otimes X & \xrightarrow{\text{br}_{\mathbf{1}, X}} & X \otimes \mathbf{1} \\ \text{lu}_X \swarrow & & \searrow \text{ru}_X \\ X & & \end{array}$$

commutes for all $X \in \text{Ob}_{\mathbf{C}}$.

Definition 39.12 (Symmetric monoidal category). A *symmetric monoidal category* is a braided monoidal category $\langle \mathbf{C}, \otimes, \mathbf{1}, \text{br} \rangle$ for which the braiding satisfies the symmetry condition

$$\text{br}_{X,Y} \circ \text{br}_{Y,X} = \text{Id}_{X \otimes Y} \quad (39.2)$$

for all $X, Y \in \text{Ob}_{\mathbf{C}}$.

Remark 39.13. If br is a natural isomorphism such that it is a candidate to be a braiding on a given monoidal category, and if, additionally, it satisfies (39.1), then the two hexagon identities are equivalent, and so only one of them needs to be checked.

39.3. DP is a monoidal category

Example 39.14. 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. 39.6).

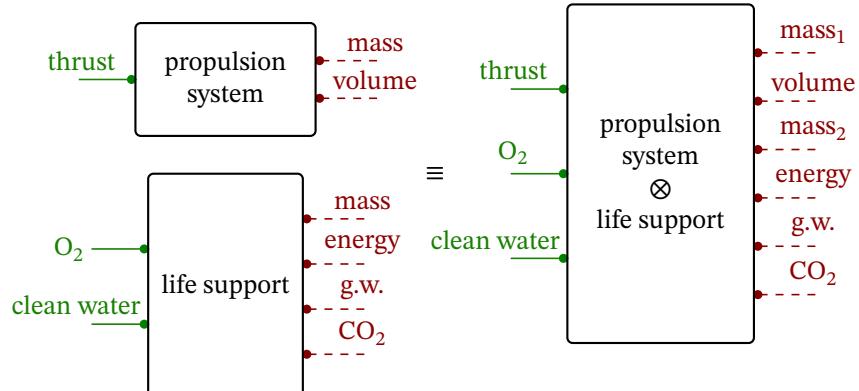


Figure 39.6.: Example of tensor of design problems.

In **DP**, putting two design problems in parallel corresponds to their *monoidal product*.

Definition 39.15 (Monoidal product in **DP**). Given two design problems $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{g} : \mathbf{C} \rightarrow \mathbf{D}$, their *monoidal product* $\mathbf{f} \otimes \mathbf{g} : \mathbf{A} \times \mathbf{C} \rightarrow \mathbf{B} \times \mathbf{D}$ is their conjunction:

$$\begin{aligned} \mathbf{f} \otimes \mathbf{g} : (\mathbf{A} \times \mathbf{C})^{\text{op}} \times (\mathbf{B} \times \mathbf{D}) &\rightarrow_{\text{Pos}} \mathbf{Bool}, \\ \langle \langle a, c \rangle^*, \langle b, d \rangle \rangle &\mapsto \mathbf{f}(a^*, b) \wedge \mathbf{g}(c^*, d). \end{aligned} \quad (39.3)$$

The diagrammatic representation of the monoidal product is reported in Fig. 39.7.

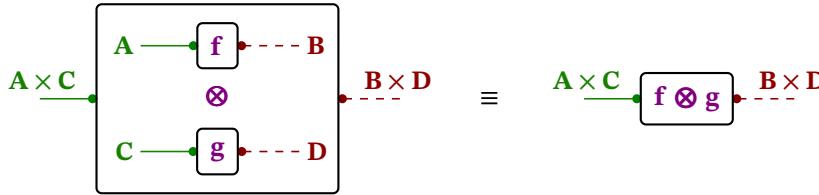


Figure 39.7.: Monoidal product of design problems.

Remark 39.16. For $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{C} \rightarrow \mathbf{D}$, the monoidal product

$$(f \otimes g)(\langle a^*, b^* \rangle, \langle c^*, d^* \rangle) \quad (39.4)$$

is true if both $f(a^*, b)$ and $g(c^*, d)$ are true, and false otherwise.

Lemma 39.17. The monoidal product \otimes is functorial (Definition 29.4) in \mathbf{DP} .

Proof. First, consider posets $\mathbf{A}, \mathbf{B} \in \text{Ob}_{\mathbf{DP}}$. We show that $\text{Id}_{\mathbf{A}} \otimes \text{Id}_{\mathbf{B}} = \text{Id}_{\mathbf{A} \times \mathbf{B}}$. It holds

$$\begin{aligned} (\text{Id}_{\mathbf{A}} \otimes \text{Id}_{\mathbf{B}})(\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle) &= \text{Id}_{\mathbf{A}}(a_1^*, a_2) \wedge \text{Id}_{\mathbf{B}}(b_1^*, b_2) \\ &= (a_1 \leq_{\mathbf{A}} a_2) \wedge (b_1 \leq_{\mathbf{B}} b_2) \\ &= \langle a_1, b_1 \rangle \leq_{\mathbf{A} \times \mathbf{B}} \langle a_2, b_2 \rangle \\ &= \text{Id}_{\mathbf{A} \times \mathbf{B}}(\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle). \end{aligned} \quad (39.5)$$

Furthermore, consider the design problems

$$f_1 : \mathbf{A}_1 \rightarrow \mathbf{B}_1, \quad f_2 : \mathbf{A}_2 \rightarrow \mathbf{B}_2, \quad g_1 : \mathbf{B}_1 \rightarrow \mathbf{C}_1, \quad g_2 : \mathbf{B}_2 \rightarrow \mathbf{C}_2.$$

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

$$\begin{aligned} \star (\langle a_1, a_2 \rangle^*, \langle c_1, c_2 \rangle) &= (f_1 \circ g_1)(a_1^*, c_1) \wedge (f_2 \circ g_2)(a_2^*, c_2) \\ &= \left(\bigvee_{b_1 \in \mathbf{B}_1} (f_1(a_1^*, b_1) \wedge g_1(b_1^*, c_1)) \right) \wedge \left(\bigvee_{b_2 \in \mathbf{B}_2} (f_2(a_2^*, b_2) \wedge g_2(b_2^*, c_2)) \right) \\ &= \bigvee_{b_1 \in \mathbf{B}_1} \bigvee_{b_2 \in \mathbf{B}_2} (f_1(a_1^*, b_1) \wedge g_1(b_1^*, c_1) \wedge f_2(a_2^*, b_2) \wedge g_2(b_2^*, c_2)) \\ &= \bigvee_{(b_1, b_2) \in \mathbf{B}_1 \times \mathbf{B}_2} (f_1(a_1^*, b_1) \wedge f_2(a_2^*, b_2) \wedge g_1(b_1^*, c_1) \wedge g_2(b_2^*, c_2)) \\ &= ((f_1 \otimes f_2) \circ (g_1 \otimes g_2))(\langle a_1, a_2 \rangle^*, \langle c_1, c_2 \rangle). \end{aligned} \quad (39.6)$$

Therefore, \otimes is functorial in \mathbf{DP} . \square

Lemma 39.18. $\langle \mathbf{DP}, \otimes, \{\bullet\} \rangle$ is a monoidal category.

Proof. To show that \mathbf{DP} is a monoidal category, we have to first identify the constituents presented in Definition 39.6. First, recall $\{\bullet\}$ to be singleton: this is the monoidal unit. In Lemma 39.17 we have shown that \otimes is a functor. Furthermore, we identify

- ▷ $\text{lu}_{\mathbf{A}} : \{\bullet\} \times \mathbf{A} \rightarrow \mathbf{A}$, for all $\mathbf{A} \in \text{Ob}_{\mathbf{DP}}$, is the left unit. This is given

by

$$\text{lu}_A(\langle 1, a_1 \rangle^*, a_2) := a_1 \leq_A a_2. \quad (39.7)$$

To prove that this is an isomorphism, we define its inverse $\text{lu}_A^{-1} : A \rightarrowtail \{\bullet\} \times A$ and show that $\text{lu}_A \circ \text{lu}_A^{-1} = \text{id}_{\{\bullet\} \times A}$ and $\text{lu}_A^{-1} \circ \text{lu}_A = \text{Id}_A$. One has

$$\begin{aligned} (\text{lu}_A^{-1} \circ \text{lu}_A)(\langle a_1 \rangle^*, a_2) &= \bigvee_{\langle 1, a \rangle \in \{\bullet\} \times A} \text{lu}_A^{-1}(a_1^*, \langle 1, a \rangle) \wedge \text{lu}_A(\langle 1, a \rangle^*, a_2) \\ &= \bigvee_{\langle 1, a \rangle \in \{\bullet\} \times A} (a_1 \leq a) \wedge a \leq a_2 \\ &= a_1 \leq a_2 \\ &= \text{id}_A(\langle a_1 \rangle^*, a_2). \end{aligned} \quad (39.8)$$

Similarly, one can show that $\text{lu}_A \circ \text{lu}_A^{-1} = \text{id}_{\{\bullet\} \times A}$.

▷ $\text{ru}_A : A \times \{\bullet\} \rightarrowtail A$, for all $A \in \text{Ob}_{\text{DP}}$, is the right unit. This is given by

$$\text{ru}(\langle a_1, 1 \rangle^*, a_2) := a_1 \leq_A a_2. \quad (39.9)$$

The proof that ru_A is an isomorphism is analogous to the one for lu_A .

▷ $\text{as}_{A,B,C} : (A \times B) \times C \rightarrowtail A \times (B \times C)$ for all $A, B, C \in \text{Ob}_{\text{DP}}$, is the associator. It is given by

$$\text{as}_{A,B,C}(\langle \langle a_1, b_1 \rangle, c_1 \rangle^*, \langle a_2, \langle b_2, c_2 \rangle \rangle) := (a_1 \leq_A a_2) \wedge (b_1 \leq_B b_2) \wedge (c_1 \leq_C c_2). \quad (39.10)$$

To prove that $\text{as}_{A,B,C}$ is an isomorphism, we define its inverse

$$\text{as}_{A,B,C}^{-1} : A \times (B \times C) \rightarrowtail (A \times B) \times C \quad (39.11)$$

and show $\text{as}_{A,B,C}^{-1} \circ \text{as}_{A,B,C} = \text{as}_{A,B,C} \circ \text{as}_{A,B,C}^{-1} = \text{Id}_{A \times B \times C}$. One has

$$\begin{aligned} &(\text{as}_{A,B,C}^{-1} \circ \text{as}_{A,B,C})(\langle a_1, \langle b_1, c_1 \rangle \rangle^*, \langle a_2, \langle b_2, c_2 \rangle \rangle) \\ &= \bigvee_{\langle \langle a, b \rangle, c \rangle \in (A \times B) \times C} \text{as}_{A,B,C}^{-1}(\langle \langle a_1, \langle b_1, c_1 \rangle \rangle^*, \langle \langle a, b \rangle, c \rangle) \wedge \\ &\quad \text{as}_{A,B,C}(\langle \langle a, b \rangle, c \rangle^*, \langle a_2, \langle b_2, c_2 \rangle \rangle) \\ &= \bigvee_{\langle \langle a, b \rangle, c \rangle \in (A \times B) \times C} ((a_1 \leq a) \wedge (b_1 \leq b) \wedge (c_1 \leq c)) \wedge \\ &\quad ((a \leq a_2) \wedge (b \leq b_2) \wedge (c \leq c_2)) \\ &= (a_1 \leq a_2) \wedge (b_1 \leq b_2) \wedge (c_1 \leq c_2) \\ &= \text{Id}_{A \times B \times C}(\langle a_1, b_1, c_1 \rangle^*, \langle a_2, b_2, c_2 \rangle). \end{aligned} \quad (39.12)$$

The proof for $\text{as}_{A,B,C} \circ \text{as}_{A,B,C}^{-1}$ is analogous.

Therefore, $\langle \text{DP}, \otimes, \{\bullet\} \rangle$ is a monoidal category. \square

DP is a symmetric monoidal category

Lemma 39.19. For any $A, B \in \text{Ob}_{\text{DP}}$, the design problem $\text{br}_{A,B} : A \times B \rightarrowtail B \times A$ given by

$$\text{br}_{A,B}(\langle a_1, b_1 \rangle^*, \langle b_2, a_2 \rangle) := (a_1 \leq_A a_2) \wedge (b_1 \leq_B b_2) \quad (39.13)$$

constitutes the braiding operation for a symmetric monoidal structure on $\langle \mathbf{DP}, \otimes, \{\bullet\} \rangle$. In other words, $\langle \mathbf{DP}, \otimes, \{\bullet\}, \text{br} \rangle$ is a symmetric monoidal category.

Proof. In this proof, given two elements a_1, a_2 of a poset \mathbf{A} , we denote for brevity $a_1 \leq_{\mathbf{A}} a_2$ by $a_1 \leq a_2$. To prove that $\text{br}_{\mathbf{A}, \mathbf{B}}$ is an isomorphism, we use Definition 39.6 and show $\text{br}_{\mathbf{A}, \mathbf{B}} \circ \text{br}_{\mathbf{B}, \mathbf{A}} = \text{id}_{\mathbf{A} \times \mathbf{B}}$. One has

$$\begin{aligned} & (\text{br}_{\mathbf{A}, \mathbf{B}} \circ \text{br}_{\mathbf{B}, \mathbf{A}}) (\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle) \\ &= \bigvee_{(b, a) \in \mathbf{B} \times \mathbf{A}} \text{br}_{\mathbf{A}, \mathbf{B}} (\langle a_1, b_1 \rangle^*, \langle b, a \rangle) \wedge \text{br}_{\mathbf{B}, \mathbf{A}} (\langle b, a \rangle^*, \langle a_2, b_2 \rangle) \\ &= ((a_1 \leq a) \wedge (b_1 \leq b)) \wedge ((a \leq a_2) \wedge (b \leq b_2)) \\ &= (a_1 \leq a_2) \wedge (b_1 \leq b_2) \\ &= \text{id}_{\mathbf{A} \times \mathbf{B}} (\langle a_1, b_1 \rangle^*, \langle a_2, b_2 \rangle). \end{aligned} \tag{39.14}$$

This also shows the second triangle identity: $\text{br}_{\mathbf{A}, \mathbf{B}}$ is its own identity. For naturality, let's consider two morphisms (design problems) $\mathbf{f}_1 : \mathbf{A}_1 \rightarrow \mathbf{B}_1, \mathbf{f}_2 : \mathbf{A}_2 \rightarrow \mathbf{B}_2$. For brevity, denote $\text{br}_{\mathbf{B}_1 \times \mathbf{B}_2, \mathbf{B}_2 \times \mathbf{B}_1}$ by $\text{br}_{\mathbf{B}}$ and $\text{br}_{\mathbf{A}_1 \times \mathbf{A}_2, \mathbf{A}_2 \times \mathbf{A}_1}$ by $\text{br}_{\mathbf{A}}$. One has

$$\begin{aligned} & ((\mathbf{f}_1 \otimes \mathbf{f}_2) \circ \text{br}_{\mathbf{B}}) (\langle a_1, a_2 \rangle^*, \langle b_2, b_1 \rangle) \\ &= \bigvee_{(b', b'') \in \mathbf{B}_1 \times \mathbf{B}_2} (\mathbf{f}_1 \otimes \mathbf{f}_2) (\langle a_1, a_2 \rangle^*, \langle b', b'' \rangle) \wedge \text{br}_{\mathbf{B}} (\langle b', b'' \rangle, \langle b_2, b_1 \rangle) \\ &= \bigvee_{(b', b'') \in \mathbf{B}_1 \times \mathbf{B}_2} (\mathbf{f}_1(a_1^*, b') \wedge \mathbf{f}_2(a_2^*, b'')) \wedge ((b' \leq b_1) \wedge (b'' \leq b_2)) \\ &= \mathbf{f}_1(a_1^*, b_1) \wedge \mathbf{f}_2(a_2^*, b_2), \end{aligned} \tag{39.15}$$

where the last step comes from the monotonicity of \mathbf{f}_1 and \mathbf{f}_2 . Similarly,

$$\begin{aligned} & (\text{br}_{\mathbf{A}} \circ (\mathbf{f}_2 \otimes \mathbf{f}_1)) (\langle a_1, a_2 \rangle^*, \langle b_2, b_1 \rangle) \\ &= \bigvee_{(a'', a') \in \mathbf{A}_2 \times \mathbf{B}_1} \text{br}_{\mathbf{A}} (\langle a_1, a_2 \rangle^*, \langle a'', a' \rangle) \wedge (\mathbf{f}_2 \otimes \mathbf{f}_1) (\langle a'', a' \rangle^*, \langle b_2, b_1 \rangle) \\ &= \bigvee_{(a'', a') \in \mathbf{A}_2 \times \mathbf{A}_1} ((a_1 \leq a'') \wedge (a_2 \leq a'')) \wedge (\mathbf{f}_2(a''^*, b_2) \wedge \mathbf{f}_1(a'^*, b_1)) \\ &= \mathbf{f}_2(a_1^*, b_1) \wedge \mathbf{f}_1(a_2^*, b_2). \end{aligned} \tag{39.16}$$

To show the first triangle identity, we write

$$\begin{aligned} & (\text{br}_{\{\bullet\} \times \mathbf{A}} \circ \text{ru}_{\mathbf{A}}) (\langle 1, a_1 \rangle^*, a_2) \\ &= \bigvee_{(a', 1) \in \mathbf{A} \times \{\bullet\}} \text{br}_{\{\bullet\} \times \mathbf{A}} (\langle 1, a_1 \rangle^*, \langle a', 1 \rangle) \wedge \text{ru}_{\mathbf{A}} (\langle a', 1 \rangle^*, a_2) \\ &= \bigvee_{(a', 1) \in \mathbf{A} \times \{\bullet\}} (1 \leq 1) \wedge (a_1 \leq a') \wedge (a' \leq a_2) \\ &= a_1 \leq a_2 \\ &= \text{lu}_{\mathbf{A}} (\langle 1, a_1 \rangle^*, a_2). \end{aligned} \tag{39.17}$$

The hexagon identities are more verbose. Consider $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \text{Ob}_{\mathbf{DP}}$. For brevity, we denote $\text{as}_{\mathbf{A}, \mathbf{B}, \mathbf{C}}$ by as , $\text{br}_{\mathbf{A}, \mathbf{B}} \otimes \text{id}_{\mathbf{C}}$ by br' , $\text{id}_{\mathbf{B}} \otimes \text{br}_{\mathbf{A}, \mathbf{C}}$ by br'' , $(\mathbf{B} \times$

$\mathbf{A} \times \mathbf{C}$ as \Diamond , and $\mathbf{B} \times (\mathbf{A} \times \mathbf{C})$ as Δ . Recall that

$$\begin{aligned}\text{br}'(\langle\langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) &= ((a_1 \leq a_2) \wedge (b_1 \leq b_2)) \wedge (c_1 \leq c_2) \\ &= (a_1 \leq a_2) \wedge (b_1 \leq b_2) \wedge (c_1 \leq c_2)\end{aligned}\quad (39.18)$$

One has

$$\begin{aligned}(\text{br}' ; \text{as})(\langle\langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) &= \bigvee_{\langle\langle b, a \rangle, c \rangle \in \Diamond} \text{br}'(\langle\langle a_1, b_1 \rangle, c_1 \rangle^*, \langle\langle b, a \rangle, c \rangle) \wedge \text{as}(\langle\langle b, a \rangle, c \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) \\ &= \bigvee_{\langle\langle b, a \rangle, c \rangle \in \Diamond} ((a_1 \leq a) \wedge (b_1 \leq b) \wedge (c_1 \leq c)) \wedge \\ &\quad ((b \leq b_2) \wedge (a \leq a_2) \wedge (c \leq c_2)) \\ &= (b_1 \leq b_2) \wedge (a_1 \leq a_2) \wedge (c_1 \leq c_2) \\ &= \underbrace{\text{as}(\langle\langle b_1, a_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle)}_{\star}.\end{aligned}\quad (39.19)$$

Furthermore, consider

$$\begin{aligned}(\star ; \text{br}'')(\langle\langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_3, \langle c_3, a_3 \rangle \rangle) &= \bigvee_{\langle b_2, \langle a_2, c_2 \rangle \rangle \in \Delta} \star(\langle\langle a_1, b_1 \rangle, c_1 \rangle^*, \langle b_2, \langle a_2, c_2 \rangle \rangle) \wedge \\ &\quad \text{br}''(\langle\langle b_2, \langle a_2, c_2 \rangle \rangle^*, \langle b_3, \langle c_3, a_3 \rangle \rangle) \\ &= \bigvee_{\langle b_2, \langle a_2, c_2 \rangle \rangle \in \Delta} ((b_1 \leq b_2) \wedge (a_1 \leq a_2) \wedge (c_1 \leq c_2)) \wedge \\ &\quad ((b_2 \leq b_3) \wedge (a_2 \leq a_3) \wedge (c_2 \leq c_3)) \\ &= (a_1 \leq a_3) \wedge (b_1 \leq b_3) \wedge (c_1 \leq c_3).\end{aligned}\quad (39.20)$$

It is easy to see that the other direction in the hexagon commutative diagram commutes. With this we have proved that br is a valid braiding operation and hence that $\langle \mathbf{DP}, \otimes, \{\bullet\}, \text{br} \rangle$ is a symmetric monoidal category. \square

39.4. Pre-monoidal categories

39.5. Monoidal functors

Definition 39.20 (Strong monoidal functor). Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}} \rangle$ and $\langle \mathbf{D}, \otimes_{\mathbf{D}}, \mathbf{1}_{\mathbf{D}} \rangle$ be two monoidal categories. A *strong monoidal functor* between \mathbf{C} and \mathbf{D} is given by:

1. A functor

$$\mathbf{F} : \mathbf{C} \rightarrow \mathbf{D}; \quad (39.21)$$

2. An isomorphism

$$\text{iso} : \mathbf{1}_{\mathbf{D}} \rightarrow \mathbf{F}(\mathbf{1}_{\mathbf{C}}); \quad (39.22)$$

3. A natural isomorphism μ

$$\mu_{X,Y} : F(X) \otimes_D F(Y) \rightarrow F(X \otimes_C Y), \quad \forall X, Y \in \mathbf{C}, \quad (39.23)$$

satisfying the following conditions:

- a) *Associativity*: For all objects $X, Y, Z \in \mathbf{C}$, the following diagram commutes.

$$\begin{array}{ccc}
 (F(X) \otimes_D F(Y)) \otimes_D F(Z) & \xrightarrow{\text{as}^D_{F(X), F(Y), F(Z)}} & F(X) \otimes_D (F(Y) \otimes_D F(Z)) \\
 \downarrow \mu_{X,Y} \otimes_D \text{Id}(F(Z)) & & \downarrow \text{Id}(F(X)) \otimes_D \mu_{Y,Z} \\
 F(X \otimes_C Y) \otimes_D F(Z) & & F(X) \otimes_D F(Y \otimes_C Z) \\
 \downarrow \mu_{X \otimes_D Y, Z} & & \downarrow \mu_{X,Y \otimes_D Z} \\
 F((X \otimes_C Y) \otimes_C Z) & \xrightarrow{F(\text{as}_X^C)} & F(X \otimes_C (Y \otimes_C Z))
 \end{array}$$

where as^C and as^D are called *associators*.

- b) *Unitality*: For all $X \in \mathbf{C}$, the following diagrams commute:

$$\begin{array}{ccc}
 1_D \otimes_D F(X) & \xrightarrow{\text{iso} \otimes_D \text{Id}_{F(X)}} & F(1_C) \otimes_D F(X) \\
 \downarrow \text{lu}_D & & \downarrow \mu_{1_C, X} \\
 F(X) & \xleftarrow{F(\text{lu}_X^C)} & F(1_C \otimes_C X) \\
 & & \\
 F(X) \otimes_D 1_D & \xrightarrow{\text{Id}_{F(X)} \otimes_D \text{iso}} & F(X) \otimes_D F(1_C) \\
 \downarrow \text{ru}_D & & \downarrow \mu_{X, 1_C} \\
 F(X) & \xleftarrow{F(\text{ru}_X^C)} & F(X \otimes_C 1_C)
 \end{array}$$

where lu^C and ru^C represent the left and right *unitors*.

39.6. Dualizable objects

There is a concept of “duality” for objects in monoidal category which we will introduce with an illustrative example. We have seen in [REF] that the category $\mathbf{C} = \mathbf{Vect}_{\mathbb{R}}$ of real vector spaces is symmetric monoidal, with tensor product as the monoidal product. Given a vector space V , its *linear dual* is the real vector space

$$V^* := \{\text{linear maps } V \rightarrow \mathbb{R}\} = \text{Hom}_{\mathbf{C}}(V; \mathbb{R}). \quad (39.24)$$

Recall from linear algebra the following fact about any vector space V :

$$V \simeq (V^*)^* \text{ if and only if } \dim V < \infty. \quad (39.25)$$

One might say that the finite-dimensional real vector spaces are characterizable

based on their behavior in this way with respect to the operation of taking the linear dual.

We will develop an alternative formulation of this fact, based on the notion of a *dualizable object*. This notion will make sense in the setting of any (symmetric) monoidal category, and we will see then, that (39.25) translates to the statement

$$V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}} \text{ is dualizable if and only if } \dim V < \infty. \quad (39.26)$$

Key protagonists in this reformulation are *evaluation* and *coevaluation* maps.

In the following, V denotes a *finite-dimensional* real vector space. The evaluation map ϵ_V associated to V is

$$\epsilon_V : V^* \otimes V \rightarrow \mathbb{R}, \langle l, v \rangle \mapsto l(v). \quad (39.27)$$

In other words, given $\langle l, v \rangle$, the map ϵ_V evaluates l at v .

The coevaluation map η_V associated to V is slightly trickier to describe. Let $\{e_1, \dots, e_n\}$ be a basis of V , and let $\{e_1^*, \dots, e_n^*\}$ be the corresponding dual basis of V^* . Then

$$\eta_V : \mathbb{R} \rightarrow V \otimes V^*, \lambda \mapsto \lambda \sum_{i=1}^n e_i \otimes e_i^*. \quad (39.28)$$

It turns out that this map is independent of the choice of basis. One way to think of this coevaluation map is to recall that $V \otimes V^* \simeq \text{Hom}(V, V)$. Under this identification, η_V maps the scalar λ to the linear endomorphism of V which is “multiplication by λ ”. (In terms of matrices, this is a diagonal matrix, with λ at every entry of the diagonal.)

Recall that as part of the monoidal structure on $\mathbf{Vect}_{\mathbb{R}}$ we have the left and right unitors

$$\text{lu}_V : \mathbf{1} \otimes V \xrightarrow{\cong} V \quad V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}} \quad (39.29)$$

$$\text{ru}_V : V \otimes \mathbf{1} \xrightarrow{\cong} V \quad V \in \text{Ob}_{\mathbf{Vect}_{\mathbb{R}}}. \quad (39.30)$$

The evalution and coevaluation maps defined above satisfy the following equations:

$$\text{lu}_V^{-1} ; (\eta_V \otimes \text{Id}_V) ; \text{as}_{V, V^*, V} ; (\text{Id}_V \otimes \epsilon_V) ; \text{ru}_V = \text{Id}_V \quad (39.31)$$

and

$$\text{ru}_{V^*}^{-1} ; (\text{Id}_{V^*} \otimes \eta_V) ; \text{as}_{V^*, V, V^*}^{-1} ; (\epsilon_V \otimes \text{Id}_{V^*}) ; \text{lu}_{V^*} = \text{Id}_{V^*} \quad (39.32)$$

Graded exercise 3 (`vectSnakeEquations`). Check (39.31) and (39.32) by direct calculation, where V is a finite-dimensional real vector space.

The equations (39.31) and (39.32) form the basis for the general notion of dualizability in a monoidal category.

Definition 39.21 (Dualizable object). Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}} \rangle$ be a monoidal category, and let $X \in \mathbf{Ob}_{\mathbf{C}}$. A *right dual object* of X is specified by:

Constituents

1. an object $X^{\vee} \in \mathbf{Ob}_{\mathbf{C}}$;
2. an evaluation map $\epsilon_X : X^{\vee} \otimes X \rightarrow \mathbf{1}$;
3. a coevaluation map $\eta_X : \mathbf{1} \rightarrow X \otimes X^{\vee}$;

Conditions

1. $\text{lu}_X^{-1} ; (\eta_X \otimes \text{Id}_X) ; \text{as}_{X, X^{\vee}, X} ; (\text{Id}_X \otimes \epsilon_X) ; \text{ru}_X = \text{Id}_X$;
2. $\text{ru}_{X^{\vee}}^{-1} ; (\text{Id}_{X^{\vee}} \otimes \eta_X) ; \text{as}_{X^{\vee}, X, X^{\vee}}^{-1} ; (\epsilon_X \otimes \text{Id}_{X^{\vee}}) ; \text{lu}_{X^{\vee}} = \text{Id}_{X^{\vee}}$.

Definition 39.22. An object X in a monoidal category is called *right dualizable* if there exists, in the category, a right dual object of X .

Remark 39.23. There is an analogous definition of *left dual object* and *left dualizability*. One can show that when the monoidal category in question is symmetric, then left dual objects can be seen as right dual objects, and vice versa. In this case, we then speak simply of *dualizability*.



40. Feedback

40.1 Feedback in design problems . . .	307
40.2 Feedback examples	307
40.3 Feedback in category theory . . .	307

40.1. Feedback in design problems

40.2. Feedback examples

40.3. Feedback in category theory

Definition 40.1 (Trace of an endomorphism). Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}}, \text{br} \rangle$ be a symmetric monoidal category. Let $X \in \text{Ob}_{\mathbf{C}}$ be dualizable and let $f \in \text{Hom}(X, X)$. The *trace* of f is the morphism $\text{Tr}(f) \in \text{Hom}(\mathbf{1}, \mathbf{1})$ defined by

$$\mathbf{1} \xrightarrow{\eta_X} X \otimes X^{\vee} \xrightarrow{f \otimes \text{id}_{X^{\vee}}} X \otimes X^{\vee} \xrightarrow{\text{br}} X^{\vee} \otimes X \xrightarrow{\epsilon_X} \mathbf{1} \quad (40.1)$$

Graded exercise 4 (LinearAlgebraTrace). Let \mathbf{C} be the category of finite-dimensional real vector spaces and \mathbb{R} -linear maps. We have seen that this category is symmetric monoidal when equipped with the usual tensor product as monoidal product. Furthermore, in Section 39.6 we saw that every object in this category is dualizable.

Fix a finite-dimensional real vector space V , and let $\{e_1, \dots, e_n\}$ be a basis of it. We saw that a choice of dual object for V is given by $V^* = \text{Hom}(V, \mathbb{R})$, together with the evaluation map

$$\epsilon_V : V^* \otimes V \rightarrow \mathbb{R}, \langle l, v \rangle \mapsto l(v) \quad (40.2)$$

and the co-evaluation map

$$\eta_V : \mathbb{R} \rightarrow V \otimes V^*, \lambda \mapsto \lambda \sum_{i=1}^n e_i \otimes e_i^*, \quad (40.3)$$

where $\{e_1^*, \dots, e_n^*\}$ is the basis dual to the one we chose for V .

Let $f : V \rightarrow V$ be a linear endomorphism – that is, $f \in \text{Hom}_{\mathbf{C}}(V, V)$. Compute the trace $\text{Tr}(f) \in \text{Hom}_{\mathbf{C}}(\mathbb{R}, \mathbb{R})$ of f according to Definition 40.1, and explain why it is the linear map “multiplication by the trace of f ”, where “trace” in this latter phrase is the usual notion that we know from linear algebra.

Definition 40.2 (Trace of a generalized endomorphism). Let $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}}, \text{br} \rangle$ be a symmetric monoidal category. Let $X \in \text{Ob}_{\mathbf{C}}$ be dualizable and let $f \in \text{Hom}_{\mathbf{C}}(Y \otimes X; Z \otimes X)$. The *trace over X* of f is the morphism $\text{Tr}_{Y,Z}^X(f) \in \text{Hom}(Y, Z)$ defined by

$$Y \xrightarrow{\text{id}_Y \otimes \eta_X} Y \otimes X \otimes X^{\vee} \xrightarrow{f \otimes \text{id}_{X^{\vee}}} Z \otimes X \otimes X^{\vee} \xrightarrow{\text{id}_Z \otimes \text{br}} Z \otimes X^{\vee} \otimes X \xrightarrow{\text{id}_Z \otimes \epsilon_X} Z \quad (40.4)$$

Definition 40.3 (Traced monoidal category). A symmetric monoidal category $\langle \mathbf{C}, \otimes_{\mathbf{C}}, \mathbf{1}_{\mathbf{C}}, \text{br} \rangle$ is said to be *traced* if it is equipped with a family of functions

$$\text{Tr}_{X,Y}^Z : \text{Hom}_{\mathbf{C}}(X \otimes_{\mathbf{C}} Z; Y \otimes_{\mathbf{C}} Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Y), \quad (40.5)$$

satisfying the following axioms:

1. *Vanishing*: For all morphisms $f : X \rightarrow Y$ in \mathbf{C} ,

$$\text{Tr}_{X,Y}^1(f) = f. \quad (40.6)$$

Furthermore, for all morphisms $f : X \otimes_{\mathbf{C}} Z \otimes_{\mathbf{C}} W \rightarrow Y \otimes_{\mathbf{C}} Z \otimes_{\mathbf{C}} W$

in \mathbf{C} :

$$\text{Tr}_{X,Y}^{Z \otimes_C W}(f) = \text{Tr}_{X,Y}^Z \left(\text{Tr}_{X \otimes_C Z, Y \otimes_C Z}^W(f) \right). \quad (40.7)$$

2. *Superposing*: For all morphisms $f : X \otimes_C Z \rightarrow Y \otimes_C Z$ in \mathbf{C} :

$$\text{Tr}_{V \otimes_C X, V \otimes_C Y}^Z(\text{Id}_V \otimes_C f) = \text{Id}_V \otimes_C \text{Tr}_{X,Y}^Z(f). \quad (40.8)$$

3. *Yanking*:

$$\text{Tr}_{Z,Z}^Z(\text{br}_{Z,Z}) = \text{Id}_Z. \quad (40.9)$$

Example 40.4. Consider a vehicle motor that weighs a certain amount but can also carry some weight (Fig. 40.1). In the diagram, we haven't added anything

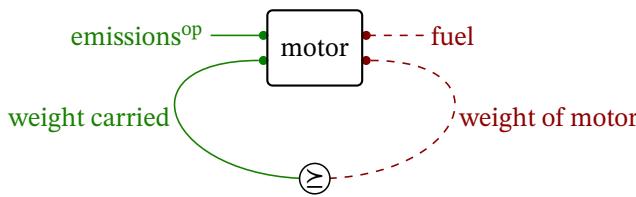


Figure 40.1.: Example of feedback in design problems.

to the weight of the 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. 40.2.

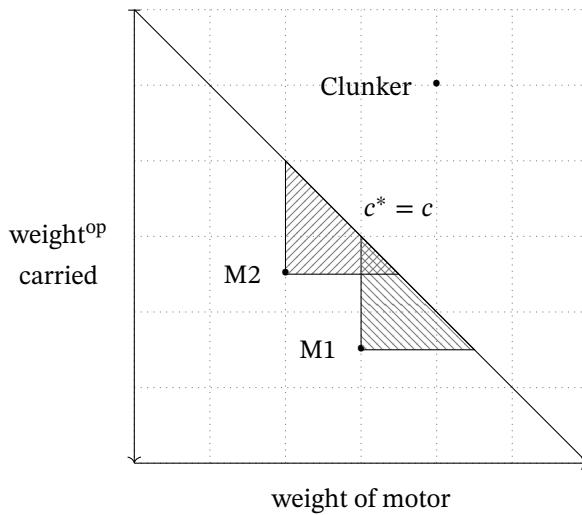


Figure 40.2.: The shaded area marks a portion of the feasible set of a traced design problem, ‘motor’. Note that it is not an upper set in the subspace “weight \times weight” of ‘motor’.

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. Suppose that we are given a design problem with a resource and a functionality of the same type \mathbf{C} (Fig. 40.3).

Can we “close the loop”, as in the diagram reported in Fig. 40.4?

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.

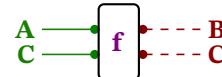


Figure 40.3.: Design problem with a resource and a functionality of the same type.

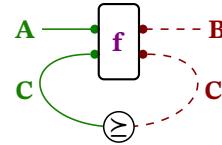


Figure 40.4.: Closing the loop in the design problem.

The following is the formal definition of the trace operation for design problems.

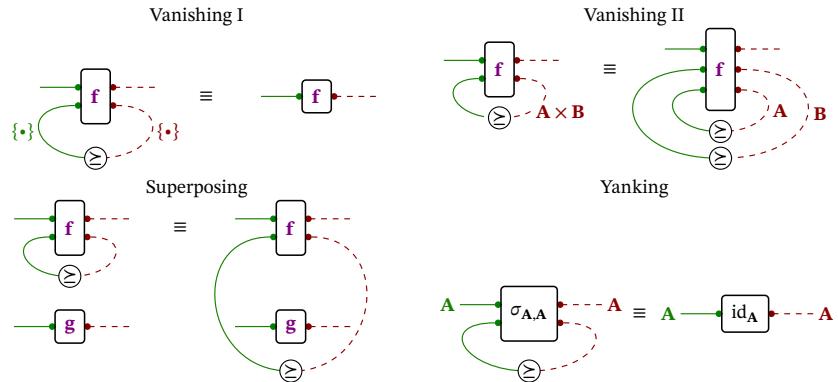
Definition 40.5 (Trace of a design problem). Given a design problem $\mathbf{f} : \mathbf{C} \times \mathbf{A} \rightarrow \mathbf{C} \times \mathbf{B}$, we can define its $\text{trace } \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}) : \mathbf{A} \rightarrow \mathbf{B}$ as follows:

$$\begin{aligned} \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}) : \mathbf{A}^{\text{op}} \times \mathbf{B} &\rightarrow_{\text{Pos Bool}} \\ \langle a^*, b \rangle &\mapsto \bigvee_{c \in \mathbf{C}} \mathbf{f}(\langle a^*, c \rangle, \langle b^*, c \rangle). \end{aligned} \quad (40.10)$$

Think of drawing a loop as a way of writing down the following requirement: Something that produces \mathbf{C} should not use up more of \mathbf{C} than it produces.

Trace axioms We will show that the loop operation $\text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}$ 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 ($\text{Tr } \mathbf{A} = \sum_i a_{ii}$), which defines the categorical trace in the (monoidal) category of vector spaces, as previously shown. 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, for instance [21]; these are equivalent to certain diagrammatic conditions [16], as in Table 40.1.

Table 40.1.: The trace axioms in diagrammatic form from [16] in **DP**.



Lemma 40.6. Trace as in Def. 40.5 satisfies the trace axioms. In other words, $(\mathbf{DP}, \otimes, \{\cdot\}, \sigma)$ is a traced monoidal category, with trace as in (40.10).

Proof. We have already shown that $\langle \mathbf{DP}, \otimes, \{\bullet\}, \sigma \rangle$ is a symmetric monoidal category (Lemma 39.19). We prove the trace axioms one by one, starting from vanishing ((40.6), (40.7)). Given any $\mathbf{A}, \mathbf{B} \in \mathbf{Ob}_{\mathbf{DP}}$ and $\mathbf{f} : \mathbf{A} \times \{\bullet\} \rightarrow \mathbf{B} \times \{\bullet\}$ in \mathbf{DP} , we have

$$\begin{aligned} \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\{\bullet\}}(\mathbf{f})(\mathbf{a}^*, \mathbf{b}) &= \bigvee_{c \in \{\bullet\}} \mathbf{f}(\langle \mathbf{a}, \mathbf{c} \rangle^*, \langle \mathbf{b}, \mathbf{c} \rangle) \\ &= \mathbf{f}(\langle \mathbf{a}, \mathbf{1} \rangle^*, \langle \mathbf{b}, \mathbf{1} \rangle) \\ &= \mathbf{f}(\mathbf{a}^*, \mathbf{b}). \end{aligned} \quad (40.11)$$

Furthermore, for any morphism $\mathbf{f} : \mathbf{A} \times \mathbf{X} \times \mathbf{Y} \rightarrow \mathbf{B} \times \mathbf{X} \times \mathbf{Y}$ in \mathbf{DP} , one has

$$\begin{aligned} \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{X} \times \mathbf{Y}}(\mathbf{f})(\mathbf{a}^*, \mathbf{b}) &= \bigvee_{\langle x, y \rangle \in \mathbf{X} \times \mathbf{Y}} \mathbf{f}(\langle \mathbf{a}, \mathbf{x}, \mathbf{y} \rangle^*, \langle \mathbf{b}, \mathbf{x}, \mathbf{y} \rangle) \\ &= \bigvee_{x \in \mathbf{X}} \left(\bigvee_{y \in \mathbf{Y}} \mathbf{f}(\langle \mathbf{a}, \mathbf{x}, \mathbf{y} \rangle^*, \langle \mathbf{b}, \mathbf{x}, \mathbf{y} \rangle) \right) \\ &= \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{X}} \left(\text{Tr}_{\mathbf{A} \times \mathbf{X}, \mathbf{B} \times \mathbf{X}}^{\mathbf{Y}}(\mathbf{f})(\langle \mathbf{a}, \mathbf{x} \rangle^*, \langle \mathbf{b}, \mathbf{x} \rangle) \right). \end{aligned} \quad (40.12)$$

For the superposing axiom ((40.8)), consider $\mathbf{f} : \mathbf{A} \times \mathbf{X} \rightarrow \mathbf{B} \times \mathbf{X}$ in \mathbf{DP} . One has

$$\begin{aligned} \text{Tr}_{\mathbf{C} \times \mathbf{A}, \mathbf{C} \times \mathbf{B}}^{\mathbf{X}}(\text{id}_{\mathbf{C}} \otimes \mathbf{f})(\langle \mathbf{c}_1, \mathbf{a} \rangle^*, \langle \mathbf{c}_2, \mathbf{b} \rangle) &= \bigvee_{x \in \mathbf{X}} \text{id}_{\mathbf{C}}(\mathbf{c}_1^*, \mathbf{c}_2) \wedge \mathbf{f}(\langle \mathbf{a}, \mathbf{x} \rangle^*, \langle \mathbf{b}, \mathbf{x} \rangle) \\ &= \text{id}_{\mathbf{C}}(\mathbf{c}_1^*, \mathbf{c}_2) \wedge \bigvee_{x \in \mathbf{X}} \mathbf{f}(\langle \mathbf{a}, \mathbf{x} \rangle^*, \langle \mathbf{b}, \mathbf{x} \rangle) \\ &= (\text{id}_{\mathbf{C}} \otimes \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{X}}(\mathbf{f}))(\langle \mathbf{c}_1, \mathbf{a} \rangle^*, \langle \mathbf{c}_2, \mathbf{b} \rangle). \end{aligned} \quad (40.13)$$

Finally, for yanking ((40.9)) consider $\sigma_{\mathbf{X}, \mathbf{X}}$. One has

$$\begin{aligned} \text{Tr}_{\mathbf{X}, \mathbf{X}}^{\mathbf{X}}(\sigma_{\mathbf{X}, \mathbf{X}})(\mathbf{x}_1^*, \mathbf{x}_2) &= \bigvee_{x \in \mathbf{X}} \sigma_{\mathbf{X}, \mathbf{X}}(\langle \mathbf{x}_1, \mathbf{x} \rangle^*, \langle \mathbf{x}, \mathbf{x}_2 \rangle) \\ &= \bigvee_{x \in \mathbf{X}} \mathbf{x}_1 \leq \mathbf{x}_2 \wedge \mathbf{x} \leq \mathbf{x} \\ &= \bigvee_{x \in \mathbf{X}} \mathbf{x}_1 \leq \mathbf{x}_2 \\ &= \text{id}_{\mathbf{X}}(\mathbf{x}_1^*, \mathbf{x}_2). \end{aligned} \quad (40.14)$$

□



41. Ordering design problems

41.1. 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 41.1 (Order on **DP**). Suppose that **A** and **B** are posets, and that $\mathbf{f}, \mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$ are design problems. We define the order as follows.

$$\frac{\mathbf{f} \leq_{\mathbf{DP}} \mathbf{g}}{\mathbf{f}(a^*, b) \leq_{\text{Bool}} \mathbf{g}(a^*, b), \quad \text{for all } a \in \mathbf{A}, b \in \mathbf{B}}$$

We diagrammatically represent the relation $\mathbf{f} \leq_{\mathbf{DP}} \mathbf{g}$ as in Fig. 41.1.

41.1 Ordering DPs	313
41.2 Restrictions and alternatives . .	314
Union of Design Problems	314
In DPI	315
Intersection of Design Problems	315
In DPI	316
41.3 Interaction with composition . .	316

Figure 41.1.: The design problem \mathbf{f} implies the design problem \mathbf{g} .

Remark 41.2. For any functionality-resource pair \mathbf{A}, \mathbf{B} , we denote by $T_{\mathbf{A}, \mathbf{B}}$ the design problem which is always feasible. We denote by $\perp_{\mathbf{A}, \mathbf{B}}$ the design problem which is never feasible, for any functionality-resource pair \mathbf{A}, \mathbf{B} .

Lemma 41.3. $\text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$ is a bounded lattice with union \vee as meet, intersection \wedge as join, least upper bound $T_{\mathbf{A}, \mathbf{B}}$ and greatest lower bound $\perp_{\mathbf{A}, \mathbf{B}}$.

Proof. First of all, we need to prove that $\text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$ is a poset. To prove this, we check the following:

- ▷ *Reflexivity*: Given $\mathbf{f} \in \text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$, $\mathbf{f} \leq_{\text{DP}} \mathbf{f}$ is always true.
- ▷ *Antisymmetry*: Given $\mathbf{f}, \mathbf{g} \in \text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$, if $\mathbf{f} \leq_{\text{DP}} \mathbf{g}$ and $\mathbf{g} \leq_{\text{DP}} \mathbf{f}$, then $\mathbf{f} = \mathbf{g}$.
- ▷ *Transitivity*: Given $\mathbf{f}, \mathbf{g}, \mathbf{h} \in \text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$, $\mathbf{f} \leq_{\text{DP}} \mathbf{g}$, and $\mathbf{g} \leq_{\text{DP}} \mathbf{h}$, then $\mathbf{f} \leq_{\text{DP}} \mathbf{h}$.

Therefore, Hom_{DP} is a poset. Furthermore, consider two design problems $\mathbf{f}, \mathbf{g} \in \text{Hom}_{\text{DP}}(\mathbf{A}; \mathbf{B})$. Their least upper bound (join) is $\mathbf{f} \wedge \mathbf{g}$, since it is the least design problem implying both \mathbf{f} and \mathbf{g} . Their greatest lower bound (meet), instead, is $\mathbf{f} \vee \mathbf{g}$, since it is the greatest design problem implied by both \mathbf{f} and \mathbf{g} . This proves that Hom_{DP} is a lattice. To prove that it is bounded, we identify the top element as $T_{\mathbf{A}, \mathbf{B}}$ (it implies all other design problems) and the bottom element as $\perp_{\mathbf{A}, \mathbf{B}}$ (it is implied by all the other design problems). \square

41.2. Restrictions and alternatives

Union of Design Problems

Let $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$ be design problems. We define the *union* $\mathbf{f} \vee \mathbf{g}$ to be the design problem which is feasible whenever *either* \mathbf{f} or \mathbf{g} is feasible. This models \mathbf{f} and \mathbf{g} as interchangeable technologies: either one can replace the other.

Definition 41.4(Union of design problems). Given two design problems $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$, their *union* $\mathbf{f} \vee \mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$ is defined by

$$\begin{aligned} (\mathbf{f} \vee \mathbf{g}) : \mathbf{A}^{\text{op}} \times \mathbf{B} &\rightarrow_{\text{Pos}} \text{Bool} \\ \langle \mathbf{a}^*, \mathbf{b} \rangle &\mapsto \mathbf{f}(\mathbf{a}^*, \mathbf{b}) \vee \mathbf{g}(\mathbf{a}^*, \mathbf{b}). \end{aligned} \tag{41.1}$$

The union of design problems is represented as in Fig. 41.2.

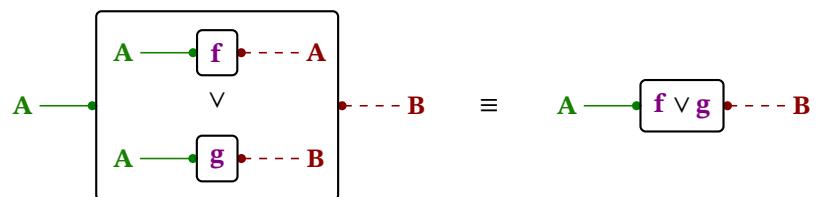
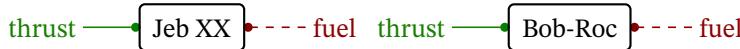


Figure 41.2.: Diagrammatic representation of the union of design problems.

Example 41.5. 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

**Figure 41.3.:** Example of two engine producers.

of performance benchmarks for their engines (Fig. 41.3). Back at NASA headquarters, Beau has uploaded Jeb and Bob's data in order to construct the design problem reported in Fig. 41.4.

**Figure 41.4.:** Example of the union of the engine design problems.

In DPI

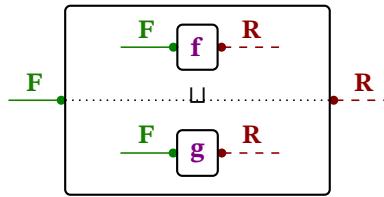
The union of two design problems is a design problem with the implementation space $\mathbf{I} = \mathbf{I}_1 \sqcup \mathbf{I}_2$, and it represents the exclusive choice between two possible alternative families of designs.

Definition 41.6 (Coproduct). Given two DPIs with same functionality and resources $\mathbf{f} = \langle \mathbf{F}, \mathbf{R}, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle$ and $\mathbf{g} = \langle \mathbf{F}, \mathbf{R}, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle$, define their co-product as

$$\mathbf{f} \sqcup \mathbf{g} := \langle \mathbf{F}, \mathbf{R}, \mathbf{I}_1 \sqcup \mathbf{I}_2, \text{prov}, \text{req} \rangle, \quad (41.2)$$

where

$$\begin{aligned} \text{prov} &: i \mapsto \begin{cases} \text{prov}_1(i), & \text{if } i \in \mathbf{I}_1, \\ \text{prov}_2(i), & \text{if } i \in \mathbf{I}_2, \end{cases} \\ \text{req} &: i \mapsto \begin{cases} \text{req}_1(i), & \text{if } i \in \mathbf{I}_1, \\ \text{req}_2(i), & \text{if } i \in \mathbf{I}_2. \end{cases} \end{aligned} \quad (41.3)$$

**Figure 41.5.**

Intersection of Design Problems

Given two design problems $\mathbf{f}, \mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$, we can define a design problem $\mathbf{f} \wedge \mathbf{g}$ that is feasible if only if \mathbf{f} and \mathbf{g} are both feasible. We call $\mathbf{f} \wedge \mathbf{g}$ the *intersection* of \mathbf{f} and \mathbf{g} . One interpretation of $\mathbf{f} \wedge \mathbf{g}$ is that \mathbf{f} and \mathbf{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 41.7 (Intersection of design problems). Given design problems $\mathbf{f} : \mathbf{A} \rightarrow \mathbf{B}$ and $\mathbf{g} : \mathbf{A} \rightarrow \mathbf{B}$, their *intersection* is denoted $(\mathbf{f} \wedge \mathbf{g}) : \mathbf{A} \rightarrow \mathbf{B}$, defined by:

$$\begin{aligned} (\mathbf{f} \wedge \mathbf{g}) : \mathbf{A}^{\text{op}} \times \mathbf{B} \rightarrow_{\text{Pos}} \mathbf{Bool} \\ \langle a^*, b \rangle \mapsto \mathbf{f}(a^*, b) \wedge \mathbf{g}(a^*, b). \end{aligned} \quad (41.4)$$

The intersection of design problems is represented as in Fig. 41.6.

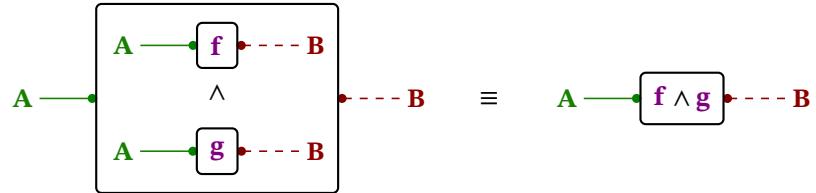


Figure 41.6.: Diagrammatic representation of the intersection of design problems.

We can directly generalize the intersection $f \wedge g$ by allowing f and g to have different domain and codomains, $f: A \rightarrow B$ and $g: C \rightarrow D$. We call this putting two design problems “in parallel”.

In DPI

41.3. Interaction with composition

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 (Def. 37.6), monoidal product (Def. 39.15), and trace (Def. 40.5) of design problems are order-preserving operations.

Lemma 41.8. Given $f_1, f_2 \in \text{Hom}_{\text{DP}}(A; B)$ and $g_1, g_2 \in \text{Hom}_{\text{DP}}(B; C)$ one has:

$$\frac{f_1 \leq_{\text{DP}} f_2 \quad g_1 \leq_{\text{DP}} g_2}{f_1 ; g_1 \leq_{\text{DP}} f_2 ; g_2} \quad (41.5)$$

In other words, series composition is order-preserving on **DP**.

Proof. We have

$$\begin{aligned} (f_1 ; g_1)(a^*, c) &= \bigvee_{b \in B} f_1(a^*, b) \wedge g_1(b^*, c) \\ &\leq_{\text{DP}} \bigvee_{b \in B} f_2(a^*, b) \wedge g_2(b^*, c) \\ &= (f_2 ; g_2)(a^*, c). \end{aligned} \quad (41.6)$$

Therefore \circ is order-preserving on **DP**. □

Lemma 41.9. Given $f_1, f_2 \in \text{Hom}_{\text{DP}}(A; B)$ and $g_1, g_2 \in \text{Hom}_{\text{DP}}(C; D)$, one has:

$$\frac{f_1 \leq_{\text{DP}} f_2 \quad g_1 \leq_{\text{DP}} g_2}{f_1 \otimes g_1 \leq_{\text{DP}} f_2 \otimes g_2} \quad (41.7)$$

In other words, the monoidal product preserves order on **DP**.

Proof. We have

$$\begin{aligned} (\mathbf{f}_1 \otimes \mathbf{g}_1)(\langle a, c \rangle^*, \langle b, d \rangle) &= \mathbf{f}_1(a^*, b) \wedge \mathbf{g}_1(c^*, d) \\ &\leq_{\text{DP}} \mathbf{f}_2(a^*, b) \wedge \mathbf{g}_2(c^*, d) \\ &= (\mathbf{f}_2 \otimes \mathbf{g}_2)(\langle a, c \rangle^*, \langle b, d \rangle) \end{aligned} \quad (41.8)$$

Therefore, \otimes is order-preserving on \mathbf{DP} . \square

Lemma 41.10. Given $\mathbf{f}_1, \mathbf{f}_2 \in \text{Hom}_{\mathbf{DP}}(\mathbf{C} \times \mathbf{A}; \mathbf{C} \times \mathbf{B})$, one has:

$$\frac{\mathbf{f}_1 \leq_{\text{DP}} \mathbf{f}_2}{\text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}_1) \leq_{\text{DP}} \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}_2)} \quad (41.9)$$

In other words, trace preserves order on \mathbf{DP} .

Proof. One has

$$\begin{aligned} \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}_1)(a^*, b) &= \bigvee_{c \in \mathbf{C}} \mathbf{f}_1(\langle c, a \rangle^*, \langle c, b \rangle) \\ &\leq_{\text{DP}} \bigvee_{c \in \mathbf{C}} \mathbf{f}_2(\langle c, a \rangle^*, \langle c, b \rangle) \\ &= \text{Tr}_{\mathbf{A}, \mathbf{B}}^{\mathbf{C}}(\mathbf{f}_2)(a^*, b). \end{aligned} \quad (41.10)$$

Therefore, Tr is order-preserving on \mathbf{DP} . \square

Lemma 41.11. Given $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4 \in \text{Hom}_{\mathbf{DP}}(\mathbf{A}; \mathbf{B})$ one has:

$$\frac{\mathbf{f}_1 \leq_{\text{DP}} \mathbf{f}_2 \quad \mathbf{f}_3 \leq_{\text{DP}} \mathbf{f}_4}{\mathbf{f}_1 \vee \mathbf{f}_3 \leq_{\text{DP}} \mathbf{f}_2 \vee \mathbf{f}_4}$$

Proof. For any $a \in \mathbf{A}, b \in \mathbf{B}$:

$$\begin{aligned} (\mathbf{f}_1 \vee \mathbf{f}_3)(a^*, b) &= \mathbf{f}_1(a^*, b) \vee \mathbf{f}_3(a^*, b) \\ &\leq_{\text{DP}} \mathbf{f}_2(a^*, b) \vee \mathbf{f}_3(a^*, b) \\ &\leq_{\text{DP}} \mathbf{f}_2(a^*, b) \vee \mathbf{f}_4(a^*, b). \end{aligned}$$

\square

Lemma 41.12. Given $\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3, \mathbf{f}_4 \in \text{Hom}_{\mathbf{DP}}(\mathbf{A}; \mathbf{B})$ one has:

$$\frac{\mathbf{f}_1 \leq_{\text{DP}} \mathbf{f}_2 \quad \mathbf{f}_3 \leq_{\text{DP}} \mathbf{f}_4}{\mathbf{f}_1 \wedge \mathbf{f}_3 \leq_{\text{DP}} \mathbf{f}_2 \wedge \mathbf{f}_4}$$

Proof. For any $a \in \mathbf{A}, b \in \mathbf{B}$:

$$\begin{aligned} (\mathbf{f}_1 \wedge \mathbf{f}_3)(a^*, b) &= \mathbf{f}_1(a^*, b) \wedge \mathbf{f}_3(a^*, b) \\ &\leq_{\text{DP}} \mathbf{f}_2(a^*, b) \wedge \mathbf{f}_3(a^*, b) \\ &\leq_{\text{DP}} \mathbf{f}_2(a^*, b) \wedge \mathbf{f}_4(a^*, b). \end{aligned}$$

\square



42. Constructing design problems

42.1 DPIs are spans	319
42.2 Companions and conjoint	320

42.1. DPIs are spans

Definition 42.1 (Span). Given a category **C**, a *span* from an object **X** to an object **Y** is a diagram of the form



where **Z** is some other object of **C**.

Example 42.2. Consider the category **Berg**, introduced in Section 14.2. An example of span in this category is reported in Fig. 42.1. Recall that Matterhorn Peak, Jungfrau Peak, and Pilatus Peak are objects of **Berg**, and the arrows are morphisms in **Berg** (paths from one location to the other).

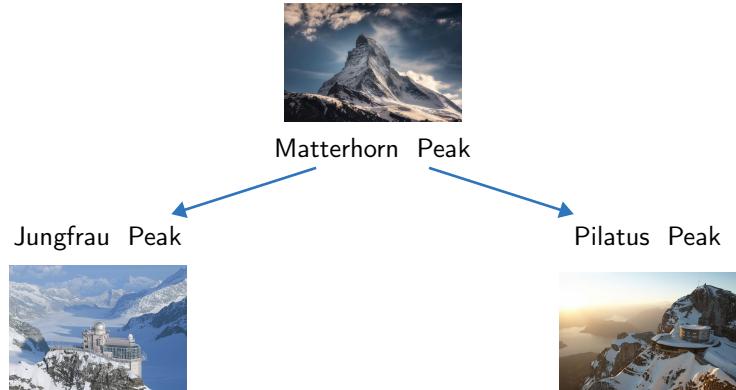


Figure 42.1.: Swiss peaks can be thought of as a span in **Berg**.

42.2. 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 42.3 (Companion and conjoint). Let \mathbf{P} and \mathbf{Q} be posets, and suppose that $f : \mathbf{P} \rightarrow_{\mathbf{Pos}} \mathbf{Q}$ is a monotone map. We define its *companion* in **DP**, denoted $\hat{f} : \mathbf{P} \rightarrowtail \mathbf{Q}$, and its *conjoint*, denoted $\check{f} : \mathbf{Q} \rightarrowtail \mathbf{P}$ as

$$\hat{f}(p^*, q) := f(p) \leq_{\mathbf{Q}} q \quad \text{and} \quad \check{f}(q^*, p) := q \leq_{\mathbf{P}} f(p). \quad (42.1)$$

Lemma 42.4. Both the companion and conjoint constructions from Def. 42.3 are functorial from **Pos** to **DP**: they preserve identities and composition.

Proof. We will show that the companion and conjoint are functors of the following forms:

$$\widehat{(\cdot)} : \mathbf{Pos} \rightarrow \mathbf{DP} \quad \text{and} \quad \check{(\cdot)} : \mathbf{Pos} \rightarrow \mathbf{DP}^{\text{op}}. \quad (42.2)$$

First, we see that they send the identity monotone map $f(p) = p$ to the unit $\text{Id}_{\mathbf{P}}$ for any poset \mathbf{P} , because

$$\begin{aligned} \widehat{\text{id}}(p_1^*, p_2) &= (p_1 \leq_{\mathbf{P}} p_2) \\ &= \check{\text{id}}(p_1^*, p_2). \end{aligned} \quad (42.3)$$

Now suppose that $f : \mathbf{P} \rightarrow_{\mathbf{Pos}} \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow_{\mathbf{Pos}} \mathbf{R}$ are given. We first show

that $\check{g} ; \check{f} = \widetilde{f ; g}$. For any $p \in \mathbf{P}$ and $r \in \mathbf{R}$, one has

$$\begin{aligned} (\check{g} ; \check{f})(p^*, r) &= \bigvee_{q \in Q} \check{g}(r^*, q) \wedge \check{f}(q^*, p) \\ &= \bigvee_{q \in Q} (r \leq_R g(q)) \wedge (q \le_Q f(p)) \\ &= r \leq_R g(f(p)) \\ &= (\widetilde{f ; g})(r^*, p). \end{aligned} \tag{42.4}$$

Similarly, we can prove that $\hat{f} ; \hat{g} = \widetilde{f ; g}$:

$$\begin{aligned} (\hat{f} ; \hat{g})(p^*, r) &= \bigvee_{q \in Q} \hat{f}(p^*, q) \wedge \hat{g}(q^*, r) \\ &= \bigvee_{q \in Q} (g(p) \leq_Q q) \wedge (g(q) \leq_R r) \\ &= g(f(p)) \leq_R r \\ &= (\widetilde{f ; g})(p^*, r). \end{aligned} \tag{42.5}$$

□

Example 42.5. The identity design problem $\text{id}_A : \mathbf{A} \rightarrowtail \mathbf{A}$ is the companion (and the conjoint) of the identity map $\text{id}'_A : A \rightarrow_{\mathbf{Pos}} A$. This is easy to check, as

$$\begin{aligned} \widehat{\text{id}}'_A(a_1^*, a_2) &= \text{id}'_A(a_1) \leq a_2 \\ &= a_1 \leq a_2 \\ &= \text{id}_A(a_1^*, a_2). \end{aligned} \tag{42.6}$$

Example 42.6. The coproduct injections ι_A, ι_B for design problems are the companions of the coproduct injections for the disjoint union.

Example 42.7. 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 \mathbf{A} , which can be thought of as a map $f : 1 \rightarrow \mathbf{A}$. By taking the companion of f one gets

$$\begin{aligned} \widehat{f} : 1 &\rightarrowtail \mathbf{A} \\ \langle 1, a \rangle &\mapsto f(1) \leq a. \end{aligned} \tag{42.7}$$

By taking the conjoint, one gets

$$\begin{aligned} \check{f} : \mathbf{A} &\rightarrowtail 1 \\ \langle a^*, 1 \rangle &\mapsto a \leq f(1). \end{aligned} \tag{42.8}$$

These two cases represent design problems with either *constant* resources or constant, functionalities, respectively.

43. Solutions to selected exercises

PART G.COMPUTATION



44. From math to implementation	327
45. Solving	329
46. Generalized computation	355
47. Uncertainty	369
48. Solutions to selected exercises	381
49. Enrichments	383
50. Operads	387



44. From math to implementation

44.1 From math to implementation . 327

44.1. From math to implementation

How can category theory help?

Looking for patterns.



45. Solving

45.1. Problem statement

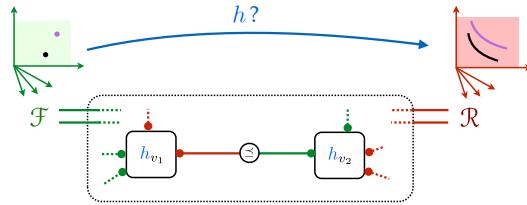
Given an arbitrary graph of design problems, and assuming we know how to solve each problem separately, we ask whether we can solve the *co-design* problem optimally.

Problem. Suppose that we are given a CDPI $\langle \mathbf{F}, \mathbf{R}, \langle \mathcal{V}, \mathcal{E} \rangle \rangle$, and that we can evaluate the map h_v for all $v \in \mathcal{V}$. Given a required functionality $f \in \mathbf{F}$, we wish to find the *minimal* resources in \mathbf{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 $h : \mathbf{F} \rightarrow \mathbf{AR}$ for the entire CDPI (Fig. 45.1).

45.1 Problem statement	329
Expressivity of MCDPs	330
Approach	331
45.2 Computational representation of DPIs	331
45.3 Composition operators for design problems	332
45.4 Monotonicity as compositional property	333
45.5 Monotonicity and fixed points .	336
45.6 Solution of MCDPs	338
Example: Optimizing over the natural numbers	339
Guarantees of Kleene ascent .	341
45.7 Extended Numerical Examples .	341
Language and interpreter/solver	341
Model of a battery	343
Competing battery technologies	343
Introducing other variations or objectives	343
From component to system co- design	345
45.8 Complexity of the solution . . .	350
Complexity of fixed point itera- tion	350
Relating complexity to the graph properties	350
Considering relations with infi- nite cardinality	351
45.9 Decomposition of MCDPs . . .	351

Figure 45.1.



The rest of the paper will provide a solution to Section 45.1, under the following assumptions:

1. The posets \mathbf{F}, \mathbf{R} are complete partial orders (Def. 45.15).
 2. The maps h is Scott continuous (Def. 45.17).

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.



Figure 45.2.: One feedback connection and a topologically continuous h are sufficient to induce a disconnected feasible set.

Example 45.1. Consider the CDPI in Fig. 45.2a. The minimal resources $M \subseteq \mathcal{AR}$ are the objectives of this optimization problem:

$$M := \begin{cases} \text{using } f, r \in \mathbf{F} = \mathbf{R}, \\ \text{Min}_{\leq} r, \\ r \in h(f), \\ r \leq f. \end{cases}$$

The **feasible set** $\Phi \subseteq \mathbf{F} \times \mathbf{R}$ is the set of functionality and resources that satisfy the constraints $r \in h(f)$ and $r \leq f$:

$$\Phi = \{ \langle f, r \rangle \in \mathbf{F} \times \mathbf{R} : (r \in h(f)) \wedge (r \leq f) \}. \quad (45.1)$$

The projection P of Φ to the functionality space is:

$$P = \{f \mid \langle f, r \rangle \in \Phi\}.$$

In the scalar case ($F = R = \langle \overline{\mathbb{R}}_{\geq 0}, \leq \rangle$), the map $h : F \rightarrow \mathcal{AR}$ is simply a map $h : \overline{\mathbb{R}}_{\geq 0} \rightarrow \overline{\mathbb{R}}_{\geq 0}$. The set P of feasible functionality is described by

$$P = \{f \in \overline{\mathbb{R}}_{\geq 0} : h(f) \leq f\}. \quad (45.2)$$

Fig. 45.2b 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.

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 45.3 defines these composition operators. Section 45.9 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.

45.2. 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.

Definition 45.2. 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 36.2, 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 $h(f) = \emptyset$.

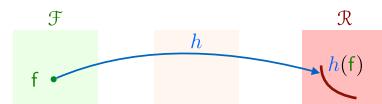


Figure 45.3.

Example 45.3. 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. 45.4). The antichains are depicted as continuous curves, but they could also be composed by a finite set of points.

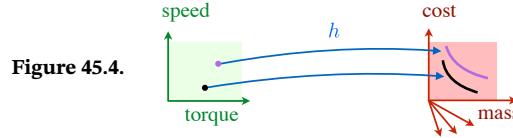


Figure 45.4.

By construction, h is monotone (Def. 22.1), 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 ???. 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.

45.3. Composition operators for design problems

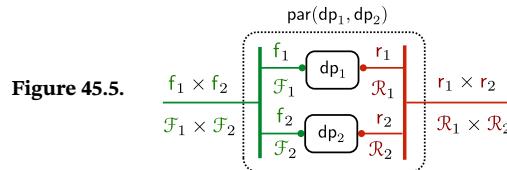
This section defines a handful of composition operators for design problems. Later, Section 45.9 will prove that any co-design problem can be described in terms of a subset of these operators.

Definition 45.4(par). The parallel composition of two DPIs $dp_1 = \langle \mathbf{F}_1, \mathbf{R}_1, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle$ and $dp_2 = \langle \mathbf{F}_2, \mathbf{R}_2, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle$ is

$$\text{par}(dp_1, dp_2) := \langle \mathbf{F}_1 \times \mathbf{F}_2, \mathbf{R}_1 \times \mathbf{R}_2, \mathbf{I}_1 \times \mathbf{I}_2, \text{prov}, \text{req} \rangle,$$

where:

$$\begin{aligned} \text{prov} &: \langle i_1, i_2 \rangle \mapsto \langle \text{prov}_1(i_1), \text{prov}_2(i_2) \rangle, \\ \text{req} &: \langle i_1, i_2 \rangle \mapsto \langle \text{req}_1(i_1), \text{req}_2(i_2) \rangle. \end{aligned} \quad (45.3)$$



Definition 45.5(loop). Suppose dp is a DPI with factored functionality space $\mathbf{F} \times \mathbf{R}$:

$$dp = \langle \mathbf{F} \times \mathbf{R}, \mathbf{R}, \mathbf{I}, \langle \text{prov}_1, \text{prov}_2 \rangle, \text{req} \rangle.$$

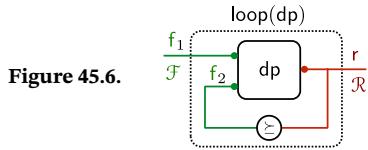
Then we can define the DPI $\text{loop}(dp)$ as

$$\text{loop}(dp) := \langle \mathbf{F}_1, \mathbf{R}, \mathbf{I}', \text{prov}_1, \text{req} \rangle,$$

where $\mathbf{I}' \subseteq \mathbf{I}$ limits the implementations to those that respect the additional constraint $\text{req}(i) \leq \text{prov}_2(i)$:

$$\mathbf{I}' = \{i \in \mathbf{I} : \text{req}(i) \leq \text{prov}_2(i)\}.$$

This is equivalent to “closing a loop” around dp with the constraint $f_2 \geq r$ (Fig. 45.6).



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. 45.7a, which can be rewritten in terms of loop, using the construction in Fig. 45.7b.

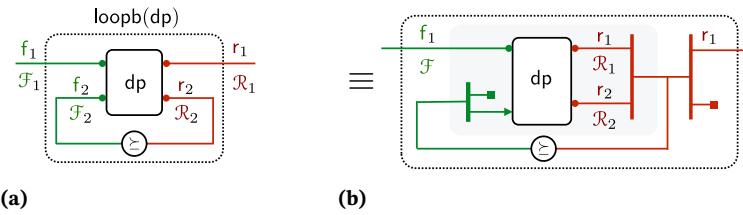


Figure 45.7.: A symmetric operator loopb can be defined in terms of loop.

45.4. Monotonicity as compositional property

The first main result of this paper is an invariance result.

Theorem 45.6. The class of MCDPs is closed with respect to interconnection.

Proof. Prop. 45.36 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 Def. 36.1 is preserved by each operator separately. This is done below in Prop. 45.7 and 45.9. \square

Proposition 45.7. If dp_1 and dp_2 are monotone (Def. 36.1), then also the composition $par(dp_1, dp_2)$ is monotone.

Proof. We need to refer to the definition of par in Def. 45.4 and check the conditions in Def. 36.1. 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.

From Def. 45.2 and Def. 45.4 we know h can be written as

$$\begin{aligned} h : F_1 \times F_2 &\rightarrow \mathcal{A}(R_1 \times R_2) \\ \langle f_1, f_2 \rangle &\mapsto \text{Min}\{\langle \text{req}_1(i_1), \text{req}_2(i_2) \rangle \mid \\ &\quad \leq_R \\ &\quad (\langle i_1, i_2 \rangle \in I_1 \times I_2) \\ &\quad \wedge (\langle f_1, f_2 \rangle \leq \langle \text{prov}_1(i_1), \text{prov}_2(i_2) \rangle)\}. \end{aligned}$$

All terms factorize in the two components, giving:

$$\begin{aligned} h : \langle f_1, f_2 \rangle &\mapsto \text{Min}_{R_1}\{\langle \text{req}_1(i_1) \rangle \mid (i \in I_1) \wedge (f_1 \leq \text{prov}_1(i_1))\} \\ &\quad \times \text{Min}_{R_2}\{\langle \text{req}_2(i_2) \rangle \mid (i \in I_2) \wedge (f_2 \leq \text{prov}_2(i_2))\}, \end{aligned}$$

which reduces to

$$\textcolor{blue}{h} : \langle \textcolor{green}{f}_1, \textcolor{green}{f}_2 \rangle \mapsto \textcolor{blue}{h}_1(\textcolor{green}{f}_1) \times \textcolor{blue}{h}_2(\textcolor{green}{f}_2). \quad (45.4)$$

The map $\textcolor{blue}{h}$ is Scott continuous iff $\textcolor{blue}{h}_1$ and $\textcolor{blue}{h}_2$ are [12, Lemma II.2.8]. \square

Proposition 45.8. If dp_1 and dp_2 are monotone (Def. 36.1), then also the composition series(dp_1, dp_2) is monotone.

Proof. From the definition of series (Def. 36.20), the semantics of the interconnection is captured by this problem:

$$\textcolor{blue}{h} : \textcolor{green}{f}_1 \mapsto \begin{cases} \text{using} & \textcolor{red}{r}_1, \textcolor{green}{f}_2 \in \mathbf{R}_1, \quad \textcolor{red}{r}_2 \in \mathbf{R}_2, \\ \text{Min}_{\leq_{\mathbf{R}_2}} & \textcolor{red}{r}_2, \\ \text{s.t.} & \textcolor{red}{r}_1 \in \textcolor{blue}{h}_1(\textcolor{green}{f}_1), \\ & \textcolor{red}{r}_1 \leq_{\mathbf{R}_1} \textcolor{green}{f}_2, \\ & \textcolor{red}{r}_2 \in \textcolor{blue}{h}_2(\textcolor{green}{f}_2). \end{cases} \quad (45.5)$$

The situation is described by Fig. 45.8. The point $\textcolor{green}{f}_1$ is fixed, and thus $\textcolor{blue}{h}(\textcolor{green}{f}_1)$ is a fixed antichain in \mathbf{R}_1 . For each point $\textcolor{red}{r}_1 \in \textcolor{blue}{h}(\textcolor{green}{f}_1)$, we can choose a $\textcolor{green}{f}_2 \geq \textcolor{red}{r}_1$. For each $\textcolor{green}{f}_2$, the antichain $\textcolor{blue}{h}_2(\textcolor{green}{f}_2)$ traces the solution in \mathbf{R}_2 , from which we can choose $\textcolor{red}{r}_2$.

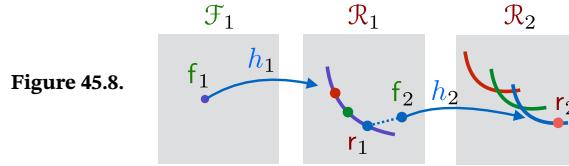


Figure 45.8.

Because $\textcolor{blue}{h}_2$ is monotone, $\textcolor{blue}{h}_2(\textcolor{green}{f}_2)$ is minimized when $\textcolor{green}{f}_2$ is minimized, hence we know that the constraint $\textcolor{red}{r}_1 \leq \textcolor{green}{f}_2$ will be tight. We can then conclude that the objective does not change introducing the constraint $\textcolor{red}{r}_1 = \textcolor{green}{f}_2$. The problem is reduced to:

$$\textcolor{blue}{h} : \textcolor{green}{f}_1 \mapsto \begin{cases} \text{using} & \textcolor{green}{f}_2 \in \mathbf{R}_1, \quad \textcolor{red}{r}_2 \in \mathbf{R}_2, \\ \text{Min}_{\leq_{\mathbf{R}_2}} & \textcolor{red}{r}_2, \\ \text{s.t.} & \textcolor{green}{f}_2 \in \textcolor{blue}{h}_1(\textcolor{green}{f}_1), \\ & \textcolor{red}{r}_2 \in \textcolor{blue}{h}_2(\textcolor{green}{f}_2). \end{cases} \quad (45.6)$$

Minimizing $\textcolor{red}{r}_2$ with the only constraint being $\textcolor{red}{r}_2 \in \textcolor{blue}{h}_2(\textcolor{green}{f}_2)$, and with $\textcolor{blue}{h}_2(\textcolor{green}{f}_2)$ being an antichain, the solutions are all and only $\textcolor{blue}{h}_2(\textcolor{green}{f}_2)$. Hence the problem is reduced to

$$\textcolor{blue}{h} : \textcolor{green}{f}_1 \mapsto \begin{cases} \text{using} & \textcolor{green}{f}_2 \in \mathbf{R}_1, \\ \text{Min}_{\leq_{\mathbf{R}_2}} & \textcolor{blue}{h}_2(\textcolor{green}{f}_2), \\ \text{s.t.} & \textcolor{green}{f}_2 \in \textcolor{blue}{h}_1(\textcolor{green}{f}_1). \end{cases} \quad (45.7)$$

The solution is simply

$$\textcolor{blue}{h} : \textcolor{green}{f}_1 \mapsto \text{Min}_{\leq_{\mathbf{R}_2}} \bigcup_{\textcolor{green}{f}_2 \in \textcolor{blue}{h}_1(\textcolor{green}{f}_1)} \textcolor{blue}{h}_2(\textcolor{green}{f}_2). \quad (45.8)$$

This map is Scott continuous because it is the composition of Scott continuous maps. \square

Proposition 45.9. If dp is monotone (Def. 36.1), so is $\text{loop}(dp)$.

Proof. The diagram in Fig. 45.6 implies that the map $h_{\text{loop}(dp)}$ can be described as:

$$h_{\text{loop}(dp)} : \mathbf{F}_1 \rightarrow \mathcal{AR}, \quad (45.9)$$

$$f_1 \mapsto \begin{cases} \text{using } r, f_2 \in R, \\ \text{Min}_{\leq_R} r, \\ \text{s.t. } r \in h_{dp}(f_1, f_2), \\ r \leq_R f_2. \end{cases} \quad (45.10)$$

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 (45.9) as

$$r \in h_{f_1}(f_2). \quad (45.11)$$

Let r be a feasible solution, but not necessarily minimal. Because of Lemma 45.10, the constraint (45.11) can be rewritten as

$$\{r\} = h_{f_1}(f_2) \cap \uparrow r. \quad (45.12)$$

Because $f_2 \geq r$, and h_{f_1} is Scott continuous, it follows that $h_{f_1}(f_2) \succeq_{\mathcal{AR}} h_{f_1}(r)$. Therefore, by Lemma 45.11, we have

$$\{r\} \succeq_{\mathcal{AR}} h_{f_1}(r) \cap \uparrow r. \quad (45.13)$$

This is a recursive condition that all feasible r must satisfy.

Let $\alpha \in \mathcal{AR}$ be an antichain of feasible resources, and let r be a generic element of R . Tautologically, rewrite α as the minimal elements of the union of the singletons containing its elements:

$$\alpha = \text{Min}_{\leq_R} \bigcup_{r \in \alpha} \{r\}. \quad (45.14)$$

Substituting (45.13) in (45.14) we obtain (cf Lemma 45.12)

$$\alpha \succeq_{\mathcal{AR}} \text{Min}_{\leq_R} \bigcup_{r \in \alpha} h_{f_1}(r) \cap \uparrow r. \quad (45.15)$$

[Converse: It is also true that if an antichain α satisfies (45.15) then all $r \in \alpha$ are feasible. The constraint (45.15) means that for any $r_0 \in \alpha$ on the left side, we can find a r_1 in the right side so that $r_0 \geq_R r_1$. The point r_1 needs to belong to one of the sets of which we take the union; say that it comes from $r_2 \in \alpha$, so that $r_1 \in h_{f_1}(r_2) \cap \uparrow r_2$. Summarizing:

$$\forall r_0 \in \alpha : \exists r_1 : (r_0 \geq_R r_1) \wedge (\exists r_2 \in \alpha : r_1 \in h_{f_1}(r_2) \cap \uparrow r_2). \quad (45.16)$$

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 α 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 (45.16) by using r_0 in place of r_1 and r_2 to obtain $\forall r_0 \in \alpha : r_0 \in h_{f_1}(r_0)$, which means that r_0 is a feasible resource.]

We have concluded that all antichains of feasible resources α satisfy (45.15), and conversely, if an antichain α satisfies (45.15), then it is an antichain of feasible resources.

Equation!(45.15) is a recursive constraint for α , of the kind

$$\Phi_{f_1}(\alpha) \leq_{\text{AR}} \alpha,$$

with the map Φ_{f_1} defined by

$$\begin{aligned} \Phi_{f_1} : \text{AR} &\rightarrow \text{AR}, \\ \alpha &\mapsto \underset{\leq_R}{\text{Min}} \bigcup_{r \in \alpha} h_{f_1}(r) \cap \uparrow r. \end{aligned} \tag{45.17}$$

If we want the *minimal* resources, we are looking for the *least* antichain:

$$\min_{\leq_{\text{AR}}} \{ \alpha \in \text{AR} : \Phi_{f_1}(\alpha) \leq_{\text{AR}} \alpha \},$$

which is equal to the *least fixed point* of Φ_{f_1} . Therefore, the map $h_{\text{loop(dp)}}$ can be written as

$$h_{\text{loop(dp)}} : f_1 \mapsto \text{lfp}(\Phi_{f_1}). \tag{45.18}$$

Lemma 45.13 shows that $\text{lfp}(\Phi_{f_1})$ is Scott continuous in f_1 . \square

Lemma 45.10. Let A be an antichain in P . Then

$$a \in A \quad \equiv \quad \{a\} = A \cap \uparrow a.$$

Lemma 45.11. For $A, B \in \text{AP}$, and $S \subseteq P$, $A \leq_{\text{AR}} B$ implies $A \cap S \leq_{\text{AR}} B \cap S$.

Lemma 45.12. For $A, B, C, D \in \text{AP}$, $A \leq_{\text{AR}} C$ and $B \leq_{\text{AR}} D$ implies $A \cup B \leq_{\text{AR}} C \cup D$.

Lemma 45.13. Let $f : \text{P} \times \text{Q} \rightarrow \text{Q}$ be Scott continuous. For each $x \in \text{P}$, define $f_x : y \mapsto f(x, y)$. Then $f^\dagger : x \mapsto \text{lfp}(f_x)$ is Scott continuous.

Proof. Davey and Priestly [9] leave this as Exercise 8.26. A proof is found in Gierz et al. [12, Exercise II-2.29]. \square

45.5. 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. [23]). It is used in embedded systems for defining the semantics of models of computation (see, e.g. [18]).

Definition 45.14 (Directed set). A set $S \subseteq \mathbf{P}$ 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 45.15 (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.

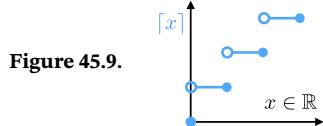
Example 45.16 (Completion of $\mathbb{R}_{\geq 0}$ to $\overline{\mathbb{R}}_{\geq 0}$). The set of real numbers \mathbb{R} is not a CPO, because it lacks a bottom. The nonnegative reals $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$ have a bottom $\perp = 0$, however, they are not a DCPO because some of their directed subsets do not have an upper bound. For example, take $\mathbb{R}_{\geq 0}$, which is a subset of $\mathbb{R}_{\geq 0}$. Then $\mathbb{R}_{\geq 0}$ is directed, because for each $a, b \in \mathbb{R}_{\geq 0}$, there exists $c = \max\{a, b\} \in \mathbb{R}_{\geq 0}$ for which $a \leq c$ and $b \leq c$. One way to make $(\mathbb{R}_{\geq 0}, \leq)$ a CPO is by adding an artificial top element \top , by defining $\overline{\mathbb{R}}_{\geq 0} \triangleq \mathbb{R}_{\geq 0} \cup \{\top\}$, and extending the partial order \leq so that $a \leq \top$ for all $a \in \mathbb{R}_{\geq 0}$.

A property of maps that will be important are monotonicity and the stronger property of Scott continuity.

Definition 45.17 (Scott continuity). A map $f : \mathbf{P} \rightarrow \mathbf{Q}$ between DCPOs is *Scott continuous* iff for each directed subset $D \subseteq \mathbf{P}$, the image $f(D)$ is directed, and $f(\sup D) = \sup f(D)$.

Remark 45.18. Scott continuity implies monotonicity.

Remark 45.19. Scott continuity does not imply topological continuity. A map from the CPO $(\overline{\mathbb{R}}_{\geq 0}, \leq)$ 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. 45.9).



A *fixed point* of $f : \mathbf{P} \rightarrow \mathbf{P}$ is a point x such that $f(x) = x$.

Definition 45.20. A *least fixed point* of $f : \mathbf{P} \rightarrow \mathbf{P}$ is the minimum (if it exists) of the set of fixed points of f :

$$\text{lfp}(f) := \min_{\leq} \{x \in \mathbf{P} : f(x) = x\}. \quad (45.19)$$

The equality in (45.19) 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 45.21 ([9, CPO Fixpoint Theorem II, 8.22]). If \mathbf{P} is a CPO and $f : \mathbf{P} \rightarrow \mathbf{P}$ 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 45.22 (Kleene's fixed-point theorem [9, CPO fixpoint theorem I, 8.15]). Assume \mathbf{P} is a CPO, and $f : \mathbf{P} \rightarrow \mathbf{P}$ is Scott continuous. Then the least fixed point of f is the supremum of the Kleene ascent chain

$$\perp \leq f(\perp) \leq f(f(\perp)) \leq \dots \leq f^{(n)}(\perp) \leq \dots.$$

45.6. 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 45.23. 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 ??.

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 Prop. 45.7 and 45.9.
The operators $\odot, \otimes, \dagger, \oslash$ are defined in Def. 45.24. \square

Definition 45.24 (Series operator \odot). For two maps $h_1 : \mathbf{F}_1 \rightarrow \mathcal{AR}_1$ and $h_2 : \mathbf{F}_2 \rightarrow \mathcal{AR}_2$, if $\mathbf{R}_1 = \mathbf{F}_2$, define

$$h_1 \odot h_2 : \mathbf{F}_1 \rightarrow \mathcal{AR}_2, \\ f_1 \mapsto \text{Min}_{\leq_{\mathbf{R}_2}} \bigcup_{s \in h_1(f_1)} h_2(s).$$

Definition 45.25 (Parallel operator \otimes). For two maps $h_1 : \mathbf{F}_1 \rightarrow \mathcal{AR}_1$ and $h_2 : \mathbf{F}_2 \rightarrow \mathcal{AR}_2$, define

$$h_1 \otimes h_2 : (\mathbf{F}_1 \times \mathbf{F}_2) \rightarrow \mathcal{A}(\mathbf{R}_1 \times \mathbf{R}_2), \\ \langle f_1, f_2 \rangle \mapsto h_1(f_1) \times h_2(f_2), \quad (45.20)$$

where \times is the product of two antichains.

Definition 45.26 (Feedback operator \dagger). For $h : \mathbf{F}_1 \times \mathbf{R} \rightarrow \mathcal{AR}$, define

$$h^\dagger : \mathbf{F}_1 \rightarrow \mathcal{AR}, \\ f_1 \mapsto \text{lfp}(\Psi_{f_1}^h), \quad (45.21)$$

where $\Psi_{f_1}^h$ is defined as

$$\Psi_{f_1}^h : \mathcal{AR} \rightarrow \mathcal{AR}, \\ R \mapsto \text{Min}_{\leq_{\mathbf{R}}} \bigcup_{r \in R} h(f_1, r) \cap \uparrow r. \quad (45.22)$$

Definition 45.27 (Coproduct operator \otimes). For $h_1, h_2 : \mathbf{F} \rightarrow \mathcal{AR}$, define

$$\begin{aligned} h_1 \otimes h_2 &: \mathbf{F} \rightarrow \mathcal{AR}, \\ f &\mapsto \text{Min}_{\leq_{\mathbf{R}}} (h_1(f) \cup h_2(f)). \end{aligned}$$

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 $c \in \mathbb{N}$:

$$\begin{cases} \text{Min}_{\leq_{\overline{\mathbb{N}} \times \overline{\mathbb{N}}}} \langle x, y \rangle, \\ \text{s.t.} \quad x + y \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c. \end{cases} \quad (45.23)$$

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. 45.10a.

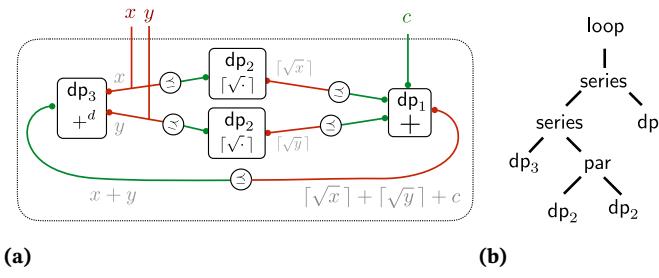


Figure 45.10.: Co-design diagram equivalent to (45.23) and its tree representation.

The diagram contains three primitive DPs: dp_1 , dp_2 (used twice), and dp_3 . Their h maps are:

$$\begin{aligned} h_1 : \overline{\mathbb{N}} \times \overline{\mathbb{N}} \times \overline{\mathbb{N}} &\rightarrow \mathcal{A}\overline{\mathbb{N}}, \\ \langle f_1, f_2, f_3 \rangle &\mapsto \{f_1 + f_2 + f_3\}, \\ h_2 : \overline{\mathbb{N}} &\rightarrow \mathcal{A}\overline{\mathbb{N}}, \\ f &\mapsto \{\lceil \sqrt{f} \rceil\}, \\ h_3 : \overline{\mathbb{N}} &\rightarrow \mathcal{A}(\overline{\mathbb{N}} \times \overline{\mathbb{N}}), \\ f &\mapsto \{(a, b) \in \overline{\mathbb{N}} \times \overline{\mathbb{N}} : a + b = f\}. \end{aligned}$$

The tree decomposition (Fig. 45.10b) corresponds to the expression

$$dp = \text{loop}(\text{series}(\text{par}(dp_2, dp_2), \text{series}(dp_1, dp_3))). \quad (45.24)$$

Consulting ??, from (45.24) one obtains an expression for h :

$$h = ((h_2 \otimes h_2) \odot h_1 \odot h_3)^\dagger. \quad (45.25)$$

This problem is small enough that we can write down an explicit expression for h . By substituting in (45.25) the definitions given in Defs. 45.24 and 45.26, we obtain

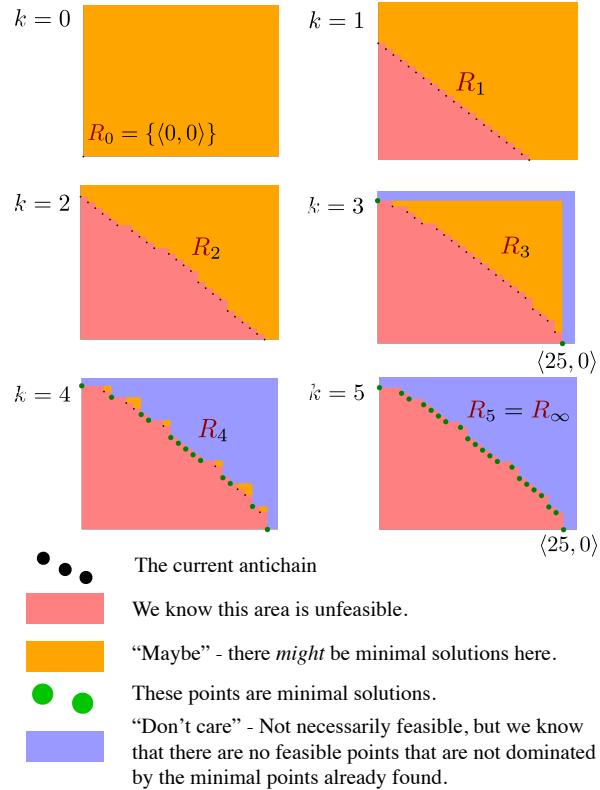


Figure 45.11: Kleene ascent to solve the problem (45.23) for $c = 20$. The sequence converges in five steps to $\alpha_5 = \alpha_\infty$.

that evaluating $h(c)$ means finding the least fixed point of a map Ψ_c :

$$h : c \mapsto \text{lfp}(\Psi_c).$$

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 (45.22) as follows:

$$\Psi_c : \alpha \mapsto \text{Min} \bigcup_{(x,y) \in \alpha} \uparrow \langle x, y \rangle \cap \quad (45.26)$$

$$\cap \left\{ \langle a, b \rangle \in \mathbb{N}^2 : (a + b \geq \lceil \sqrt{x} \rceil + \lceil \sqrt{y} \rceil + c) \right\}. \quad (45.27)$$

Kleene’s algorithm is the iteration $\alpha_{k+1} = \Psi_c(\alpha_k)$ starting from $\alpha_0 = \perp_{\mathcal{A}(\bar{\mathbb{N}} \times \bar{\mathbb{N}})} = \{\langle 0, 0 \rangle\}$.

For $c = 0$, the sequence converges immediately:

$$\alpha_0 = \{\langle \mathbf{0}, \mathbf{0} \rangle\} = h(0).$$

For $c = 1$, the sequence converges at the second step:

$$\begin{aligned} \alpha_0 &= \{\langle 0, 0 \rangle\}, \\ \alpha_1 &= \{\langle \mathbf{0}, \mathbf{1} \rangle, \langle \mathbf{1}, \mathbf{0} \rangle\} = h(1). \end{aligned}$$

For $c = 2$, the sequence converges at the fourth step; however, some solutions (in

bold) converge sooner:

$$\begin{aligned}\alpha_0 &= \{\langle 0, 0 \rangle\}, \\ \alpha_1 &= \{\langle 0, 2 \rangle, \langle 1, 1 \rangle, \langle 2, 0 \rangle\}, \\ \alpha_2 &= \{\langle \mathbf{0}, \mathbf{4} \rangle, \langle 2, 2 \rangle, \langle \mathbf{4}, \mathbf{0} \rangle\}, \\ \alpha_3 &= \{\langle \mathbf{0}, \mathbf{4} \rangle, \langle 3, 3 \rangle, \langle \mathbf{4}, \mathbf{0} \rangle\} = h(2).\end{aligned}$$

The next values in the sequence are:

$$\begin{aligned}h(3) &= \{\langle \mathbf{0}, \mathbf{6} \rangle, \langle 3, 4 \rangle, \langle \mathbf{4}, \mathbf{3} \rangle, \langle \mathbf{6}, \mathbf{0} \rangle\}, \\ h(4) &= \{\langle \mathbf{0}, 7 \rangle, \langle 3, 6 \rangle, \langle \mathbf{4}, \mathbf{4} \rangle, \langle 6, 3 \rangle, \langle 7, \mathbf{0} \rangle\}.\end{aligned}$$

Fig. 45.11 shows the sequence for $c = 20$.

Guarantees of Kleene ascent

Solving an MCDP with cycles reduces to computing a Kleene ascent sequence α_k . At each instant k we have some additional guarantees.

For any finite k , the resources “below” α_k (the set $\mathbf{R} \setminus \uparrow \alpha_k$) are infeasible. (In Fig. 45.11, those are colored in red.)

If the iteration converges to a non-empty antichain α_∞ , the antichain α_∞ divides \mathbf{R} in two. Below the antichain, all resources are infeasible. However, above the antichain (purple area), it is not necessarily true that all points are feasible, because there might be holes in the feasible set, as in Example 45.1. Note that this method does not compute the entire feasible set, but rather only the *minimal elements* of the feasible set, which might be much easier to compute.

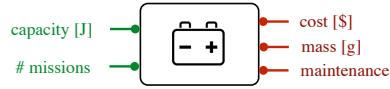
Finally, if the sequence converges to the empty set, it means that there are no solutions. The sequence α_k can be considered a certificate of infeasibility.

45.7. 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.

Language and interpreter/solver

MCDPL is a modeling language to describe MCDPs and their compositions. It is inspired by CVX and “disciplined convex programming” [14]. MCDPL is even more disciplined than CVX; for example, multiplying by a negative number is a *syntax* error. The figures are generated by PyMCDP, an interpreter and solver for MCDPs, which implements the techniques described in this paper. The software and a manual are available at <http://mcdp.mit.edu>.



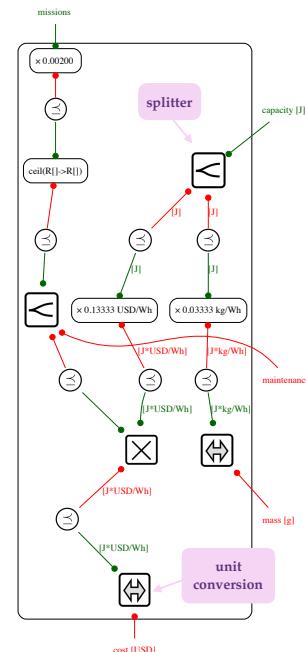
(a) Interface of battery design problem.

```

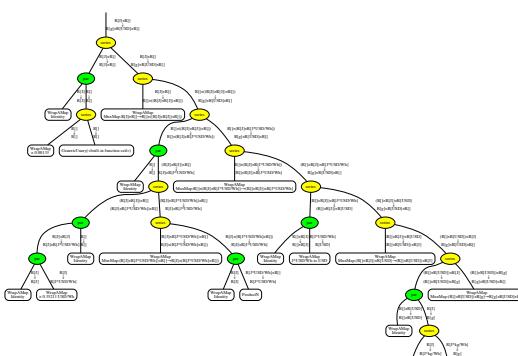
1 mcdp {
2   provides capacity [J]
3   provides missions [R]
4
5   requires mass [g]
6   requires cost [$]
7
8   # Number of replacements
9   requires maintenance [R]
10
11  # Battery properties
12  specific_energy = 30 Wh/kg
13  specific_cost = 7.50 Wh/$
14  cycles = 500 []
15
16  # Constraint between 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 }

```

(b) MCDPL code equivalent to equations Eqs. (45.28) to (45.30).



(c) Co-design diagram generated by PyMCDP from code in panel (b).



(d) Tree representation using par/series of diagram in panel (c).

Figure 45.12.: Panel (c) shows the co-design diagram generated from the code in (b). Panel (d) shows a tree representation (series, parallel) for the diagram. The edges show the types of functionality and resources. The leaves are labeled with the Python class used internally by the interpreter P_MCDP.

Model of a battery

The choice of a battery can be modeled as a DPI (Fig. 45.12a) with functionalities **capacity [J]** and **number of missions** and with resources **mass [kg]**, **cost [\$]** and “**maintenance**”, defined as the number of times that the battery needs to be replaced over the lifetime of the robot.

Each battery technology is described by the three parameters specific energy, specific cost, and lifetime (number of cycles):

$$\rho := \text{specific energy [Wh/kg]},$$

$$\alpha := \text{specific cost [Wh/$]},$$

$$c := \text{battery lifetime [\# of cycles]}.$$

The relation between functionality and resources is described by three nonlinear monotone constraints:

$$\text{mass} \geq \text{capacity}/\rho, \quad (45.28)$$

$$\text{maintenance} \geq \lceil \text{missions}/c \rceil, \quad (45.29)$$

$$\text{cost} \geq \lceil \text{missions}/c \rceil (\text{capacity}/\alpha). \quad (45.30)$$

Fig. 45.12b shows the MCDPL code that describes the model corresponding to Eqs. (45.28) to (45.30). The diagram in Fig. 45.12c is automatically generated from the code. Fig. 45.12d shows a tree representation of the diagram using the series/par operators.

Competing battery technologies

The parameters for the battery technologies used in this example are shown in Section 45.7.

<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. 45.12b by plugging in the specific values in lines 12–14.

Given the different models, we can define their co-product (Fig. 45.13a) using the MCDPL code in Fig. 45.13b.

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

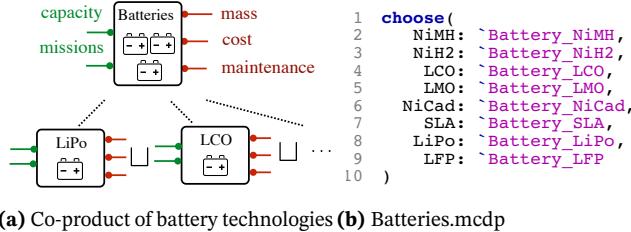


Figure 45.13.: The co-product of design problems describes the choices among different technologies. The MCDPL keyword for the co-product is “choose”.

a family of multi-objective optimization problems, of the type “Given **capacity** and **missions**, minimize $\langle \text{cost}, \text{mass}, \text{maintenance} \rangle$ ”. We can further extend the class of optimization problems by introducing other hard constraints and by choosing which resource to prioritize. This can be done by composition of design problems; that is, by creating a larger DP that contains the original DP as a subproblem, and contains some additional degenerate DPs that realize the desired semantics.

For example, suppose that we would like to find the optimal solution(s) such that: 1) The mass does not exceed 3 kg; 2) The mass is minimized as a primary objective, while cost/maintenance are secondary objectives.

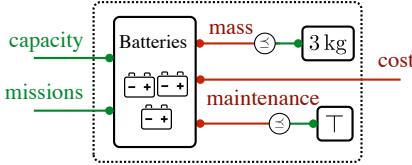
This semantics can be described by the co-design diagram in Fig. 45.14a, which contains two new symbols. The DP labeled “3 kg” implements the semantics of hard constraints. It has one functionality ($F = \mathbb{R}_{\geq 0}$) and zero resources ($R = \overline{\text{kg}}$). The poset $\mathbf{1} = \{\langle \rangle\}$ has exactly two antichains: \emptyset and $\{\langle \rangle\}$. These represent “infeasible” and “feasible”, respectively. The DP is described by the map

$$\begin{aligned}
 h : \overline{\mathbb{R}_{\geq 0}} &\rightarrow \mathcal{A}\mathbf{1}, \\
 f &\mapsto \begin{cases} \{\langle \rangle\}, & \text{if } f \leq 3 \text{ kg,} \\ \emptyset, & \text{if } f > 3 \text{ kg.} \end{cases}
 \end{aligned}$$

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. 45.14b. Note the intuitive interface: the user can directly write “mass required by battery ≤ 3 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. 45.17 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. 45.17f 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. 45.17d shows the case where the objective is to minimize the **cost**, which, defined by (45.30), 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. 45.17h) or $\langle \text{mass}, \text{cost}, \text{maintenance} \rangle$ (Fig. 45.17h), there



(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 45.14.: 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.

are multiple non-dominated solutions.

From component to system co-design

The rest of the section reuses the battery DP into a larger co-design problem that considers the co-design of actuation together with energetics for a drone (Fig. 45.15a). 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. 45.15a) are parametrized by **endurance**, **number of missions**, **extra power** to be supplied, and **payload**. We model “actuation” as a design problem with functionality **lift [N]** and resources **cost**, **mass** and **power**, and we assume that power is a quadratic function of lift (Fig. 45.16). Any other monotone map could be used.

Figure 45.16.

The co-design constraints that combine energetics and actuation are

$$\text{battery capacity} \geq \text{total power} \times \text{endurance}, \quad (45.31)$$

$$\text{total power} = \text{actuation power} + \text{extra power},$$

$$\text{weight} = \text{total mass} \times \text{gravity},$$

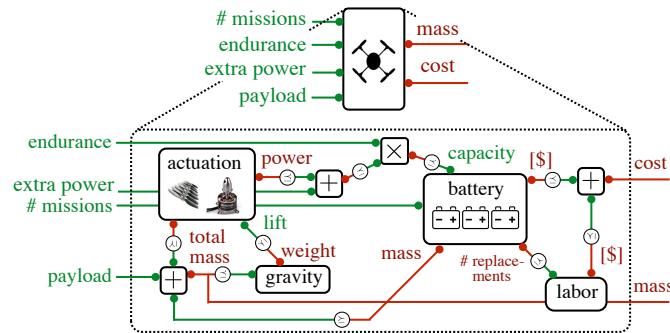
$$\text{actuation lift} \geq \text{weight},$$

$$\text{labor cost} = \text{cost per replacement} \times \text{battery maintenance},$$

$$\text{total cost} = \text{battery cost} + \text{actuation cost} + \text{labor cost},$$

$$\text{total mass} = \text{battery mass} + \text{actuation mass} + \text{payload}. \quad (45.32)$$

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. 45.15b; it refers to the previously defined models for “batteries” and “actuation”.



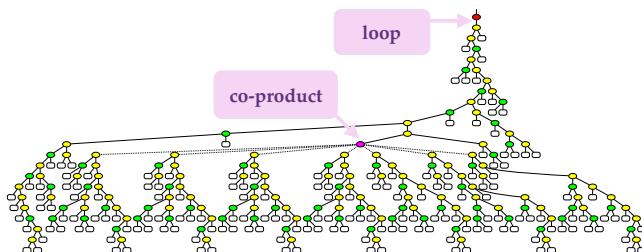
(a) Co-design diagram corresponding to Eqs. (45.31) to (45.32).

```

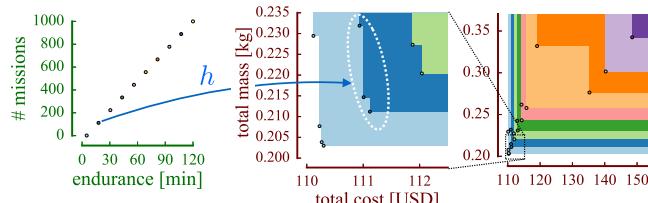
1 mcdp {
2   provides endurance [s]
3   provides extra_payload [kg]
4   provides extra_power [W]
5   provides num_missions [R]
6   provides velocity [m/s]
7
8   requires total_cost [$]
9   requires total_mass [g]
10
11   battery = instance `Batteries
12   actuation = instance `Actuation
13
14   total_power = 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 }

```

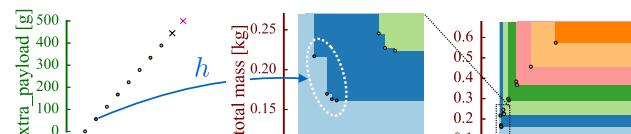
(b) MCDPL code for Eqs. (45.31) to (45.32). The “instance” statements refer to previously defined models for batteries (Fig. 45.13b) and actuation (not shown).



(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.



(d) Relation between endurance and number of missions and cost and mass.



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. 45.15a, we know that there is a monotone relation between any pair of functionality and resources, such as **payload** and **cost**, or **endurance** and **mass**, even without knowing exactly what are the models for battery and actuation.

When fully expanded, the co-design graph (too large to display) contains 110 nodes and 110 edges. It is possible to remove all cycles by removing only one edge (**e.g.**, the **energy** \leq **capacity** constraint), so the design complexity (Def. 45.32) is equal to $\text{width}(\mathbb{R}_{\geq 0}) = 1$. The tree representation is shown in Fig. 45.15c. 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 (45.24) for the previous example. The least fixed point sequence converges to 64 bits machine precision in 50–100 iterations.

To visualize the multidimensional relation

$$\mathbf{h} : \overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0}^s \times \overline{\mathbb{R}}_{\geq 0}^w \times \overline{\mathbb{R}}_{\geq 0}^g \rightarrow \mathcal{A}(\overline{\mathbb{R}}_{\geq 0}^{\text{kg}} \times \overline{\mathbb{R}}_{\geq 0}^{\text{USD}}),$$

we need to project across 2D slices. Fig. 45.15d shows the relation when the functionality varies in a chain in the space **endurance/missions**, and Fig. 45.15e shows the results for a chain in the space **endurance/payload**.

Finally, Fig. 45.18 shows the optimal choices of battery technologies in the **endurance/missions** space, when one wants to minimize **mass**, **cost**, or $\langle \text{mass}, \text{cost} \rangle$. The decision boundaries are completely different from those in Fig. 45.17. 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 45.17.: 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. 45.13). 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 **(capacity, cost, maintenance)**. The shape of the symbols shows how many minimal solutions exists for a particular value of the functionality **(capacity, missions)**. 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.

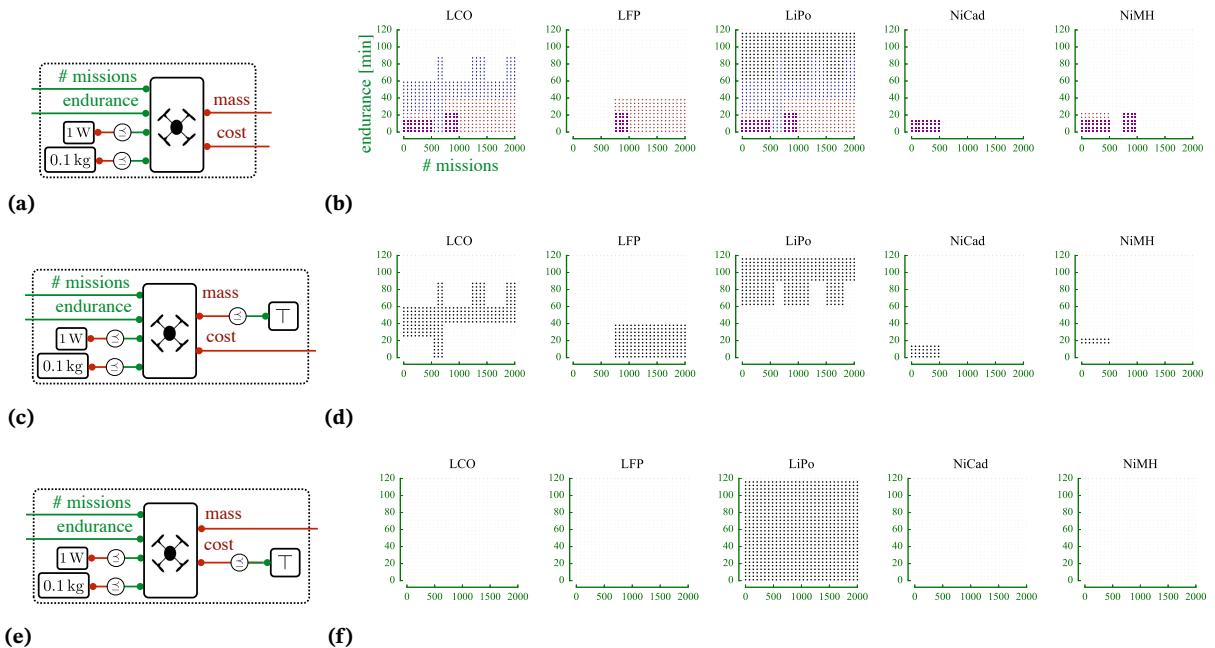


Figure 45.18: This figure shows the decision boundaries for the different values of battery technologies for the integrated actuation-energetics model described in Fig. 45.15. Please see the caption of Fig. 45.17 for an explanation of the symbols. Notice how in most cases the decision boundaries are different than those in Fig. 45.17: this is an example in which one component cannot be optimized by itself without taking into account the rest of the system.

45.8. Complexity of the solution

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{AR} . This allows to give worst-case bounds on the number of iterations.

Proposition 45.28. 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	$O(\text{height}(\mathcal{AR}_0))$
total computation	$O(\text{width}(\mathbf{R}_0) \times \text{height}(\mathcal{AR}_0) \times c)$

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{AR}_0 , and we are constructing an ascending chain, so it can take at most $\text{height}(\mathcal{AR}_0)$ 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 45.29. Consider solving $\text{dp} = \text{loop}(\text{dp}_0)$ with dp_0 defined by $\text{h}_0 : \langle \langle \rangle, x \mapsto x + 1$ with $x \in \overline{\mathbb{N}}$. Then the least fixed point equation is equivalent to solving $\min\{x : \Psi(x) \leq x\}$ with $\Psi : x \mapsto x + 1$. The iteration $R_{k+1} = \Psi(R_k)$ converges to T in $\text{height}(\overline{\mathbb{N}}) = \aleph_0$ steps.

Remark 45.30. 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 45.31 (Invariance to re-parameterization). All the results given in this paper are invariant to any order-preserving re-parameterization of all the variables involved.

Relating complexity to the graph properties

Prop. 45.28 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 45.9, we need to find an arc feedback set (AFS) of the graph $\langle \mathcal{V}, \mathcal{E} \rangle$. 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 Prop. 45.28.

Of the three performance measures in Prop. 45.28, 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 45.32. Given a graph $(\mathcal{V}, \mathcal{E})$ and a labeling of each edge $e \in \mathcal{E}$ with a poset \mathbf{R}_e , the *design complexity* $\text{DC}((\mathcal{V}, \mathcal{E}))$ is defined as

$$\text{DC}((\mathcal{V}, \mathcal{E})) = \min_{F \text{ is an AFS}} \text{width}\left(\prod_{e \in F} \mathbf{R}_e\right). \quad (45.33)$$

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.

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 45.33. Suppose that the platform to be designed must travel a distance d [m], and we need to choose the endurance T [s] and the velocity v [m/s]. The relation among the quantities is $d \leq T v$. This is a design problem described by the map

$$\begin{aligned} h : \overline{\mathbb{R}}_{\geq 0} &\rightarrow \mathcal{A}\overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0}, \\ d &\mapsto \{(T, v) \in \overline{\mathbb{R}}_{\geq 0} \times \overline{\mathbb{R}}_{\geq 0} : d = T v\}. \end{aligned}$$

For each value of d , there is a continuum of solutions.

One approach to solving this problem would be to discretize the functionality \mathbf{F} and the resources \mathbf{R} by sampling and/or coarsening. However, sampling and coarsening makes it hard to maintain completeness and consistency.

One effective approach, outside of the scope of this paper, that allows to use finite computation is to *approximate the design problem itself*, rather than the spaces \mathbf{F}, \mathbf{R} , which are left as possibly infinite. The basic idea is that an infinite antichain can be bounded from above and above by two antichains that have a finite number of points. This idea leads to an algorithm that, given a prescribed computation budget, can compute an inner and outer approximation to the solution antichain [3].

45.9. 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).

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 45.34. Two DPs $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}_1, \text{prov}_1, \text{req}_1 \rangle$ and $\langle \mathbf{F}, \mathbf{R}, \mathbf{I}_2, \text{prov}_2, \text{req}_2 \rangle$ are *equivalent* if there exists a map $\varphi : \mathbf{I}_2 \rightarrow \mathbf{I}_1$ such that $\text{prov}_2 = \text{prov}_1 \circ \varphi$ and $\text{req}_2 = \text{req}_1 \circ \varphi$.

Plumbing

We also need to define “trivial DPs”, 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 DP. The implementation space of a trivial DP is a copy of the functionality space and there is a 1-to-1 correspondence between functionality and implementation.

Definition 45.35 (Trivial DPs). Given a map $f : \mathbf{F} \rightarrow \mathbf{R}$, we can lift it to define a trivial DP $\text{Triv}(f) = \langle \mathbf{F}, \mathbf{R}, \mathbf{F}, \text{Id}_{\mathbf{F}}, f \rangle$, where $\text{Id}_{\mathbf{F}}$ is the identity on \mathbf{F} .

Proposition 45.36. 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 DPs. 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 [13]). For example, the CDPI represented in Fig. 45.21a has a minimal AFS that contains the edge $c \rightarrow a$ (Fig. 45.21b).

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 DPs. 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. 45.19).

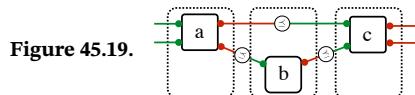
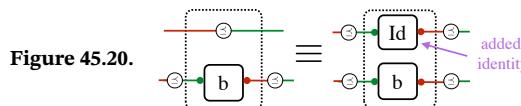


Figure 45.19.

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. 45.20).



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. 45.22a). In this example, the tree representation (Fig. 45.22b) 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. 45.23a can be rewritten as Fig. 45.23b. This construction is analogous to the construction used for the analysis of process networks [18] (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

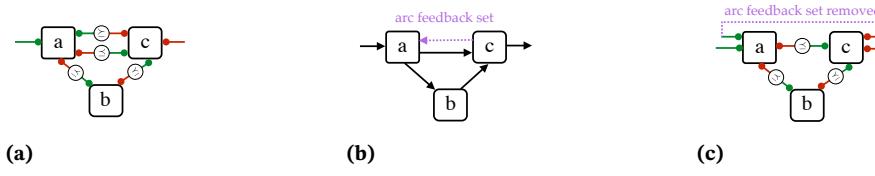


Figure 45.21.: An example co-design diagram with three nodes $V = \{a, b, c\}$, in which a minimal arc feedback set is $\{c \rightarrow a\}$.

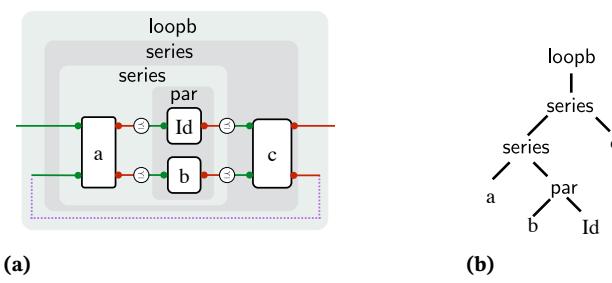


Figure 45.22.: Tree representation for the co-design diagram in Fig. 45.21a.

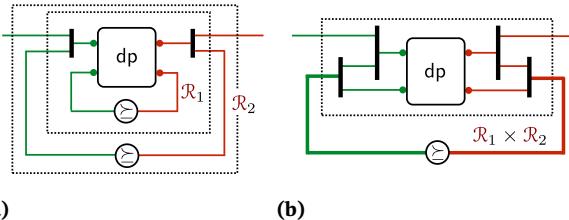


Figure 45.23.: If there are nested loops in a co-design diagram, they can be rewritten as one loop, by taking the product of the edges.

45.10. Related work



46. Generalized computation

The definition of monads is very powerful because it is very abstract and can fit many possible scenarios. Before getting to the formal definition, it is useful to build up the intuition using several examples.

46.1. Generalized objects and operations

Monads are a type of algebraic structure that is well-suited to represent generalized objects and operations. We begin by giving several examples.

Modeling nondeterministic uncertainty

For the engineer, one intuitive scenario where generalization is necessary is handling uncertainty. We have seen the category **Set** of sets and functions between sets. A function

$$f : X \rightarrow Y \tag{46.1}$$

between sets is “deterministic”, in the sense that, given as input an element of X , it always produces the same output in Y .

Suppose now we want to deal with nondeterministic functions: functions that return for each value any element of a certain subset. We can model nondeter-

46.1 Generalized objects and operations	355
Modeling nondeterministic uncertainty	355
Modeling interval uncertainty	357
Upper sets	359
Keeping track of resource usage	359
46.2 Monads	360
46.3 The Kleisli construction	364
46.4 Algebras of a monad	364
46.5 Monads, computer science definition	368

ministic functions from X to Y as functions of the type

$$\textcolor{blue}{f} : \textcolor{red}{X} \rightarrow \textcolor{violet}{P}Y. \quad (46.2)$$

Note that this is a generalization, in the sense that any deterministic function is a special nondeterministic function. For example, the function

$$\begin{aligned} \alpha : \mathbb{N} &\longrightarrow \mathbb{N}, \\ x &\longmapsto x^2, \end{aligned} \quad (46.3)$$

can be rewritten as the function

$$\begin{aligned} \alpha' : \mathbb{N} &\longrightarrow \textcolor{violet}{P}\mathbb{N}, \\ x &\longmapsto \{x^2\}, \end{aligned} \quad (46.4)$$

which maps each element to a singleton set.

Need: mapping a regular morphism into a generalized morphism.

Once we have these generalized functions, we really want them to form a category, so that they can compose. To do this, we need extra information: additional structure. So far we know the rules of composition for functions:

$$\begin{array}{c} \textcolor{blue}{f} : \textcolor{red}{X} \rightarrow \textcolor{red}{Y} \qquad \textcolor{blue}{g} : \textcolor{red}{Y} \rightarrow \textcolor{red}{Z} \\ \hline (\textcolor{blue}{f} ; \textcolor{blue}{g}) : \textcolor{red}{X} \longrightarrow \textcolor{red}{Z} \\ x \longmapsto \textcolor{blue}{g}(\textcolor{blue}{f}(x)), \end{array} \quad (46.5)$$

but this does not help for the generalized functions. How can we compose nondeterministic function? What to fill in the space below?

$$\begin{array}{c} \textcolor{blue}{f} : \textcolor{red}{X} \rightarrow \textcolor{violet}{P}Y \qquad \textcolor{blue}{g} : \textcolor{red}{Y} \rightarrow \textcolor{violet}{P}Z \\ \hline (\textcolor{blue}{f} ; \textcolor{blue}{g}) : \textcolor{red}{X} \longrightarrow \textcolor{violet}{P}Z \\ x \longmapsto ? \end{array} \quad (46.6)$$

There is one natural way to define this operation. Did you notice the following?

Lemma 46.1. Functions from X to $\textcolor{violet}{P}Y$ are exactly the relations from X to Y .

| *Proof.*

□

Therefore, we expect that nondeterministic functions compose like relations. If we have $\textcolor{blue}{f} : \textcolor{red}{X} \rightarrow \textcolor{violet}{P}Y$ and $\textcolor{blue}{g} : \textcolor{red}{Y} \rightarrow \textcolor{violet}{P}Z$ then the composite nondeterministic function is

$$\begin{aligned} (\textcolor{blue}{f} ; \textcolor{blue}{g}) : \textcolor{red}{X} &\longrightarrow \textcolor{violet}{P}Z \\ x &\longmapsto \bigcup_{y \in \textcolor{blue}{f}(x)} \textcolor{blue}{g}(y) \end{aligned} \quad (46.7)$$

One can convince oneself that this is correct by looking at Fig. 46.1.

We should also check that this composition is associative; however, this comes

Figure 46.1.

automatically from the fact that we already know that **Rel** is a category.

To summarize:

- ▷ We wanted to extend **Set** from functions $X \rightarrow Y$ to nondeterministic functions of type $X \rightarrow \mathcal{P}Y$.
- ▷ To do this, we needed three things:
 1. The particular choice of \mathcal{P} as what maps a set to another set.
 2. A way to lift a function of type $X \rightarrow Y$ to a function of type $X \rightarrow \mathcal{P}Y$, so that we can say that nondeterministic functions are a generalization of deterministic functions. This lifting operation is a family of functions of type

$$\text{lift}_{X,Y} : (X \rightarrow Y) \rightarrow (X \rightarrow \mathcal{P}Y). \quad (46.8)$$

- 3. A way to define composition, through a map, traditionally called “fish”, of type

$$\text{fish}_{X,Y,Z} : (X \rightarrow \mathcal{P}Y) \times (Y \rightarrow \mathcal{P}Z) \rightarrow (X \rightarrow \mathcal{P}Z) \quad (46.9)$$

- ▷ And we needed these pieces to satisfy the conditions:

1. *fish* is associative. This ensures the generalized functions form a semi-category.
2. The composition of the lifted functions are the lifting of the composition:

$$\frac{f : X \rightarrow \mathcal{P}Y \quad g : Y \rightarrow \mathcal{P}Z}{\text{lift}_{X,Y}(f) ; \text{lift}_{Y,Z}(g) = \text{lift}_{X,Z}(f ; g)} \quad (46.10)$$

This ensures that inside the generalized functions, the composition of regular functions continues to work as it should.

Modeling interval uncertainty

We continue to build intuition considering another type of uncertainty.

We have seen the category **Pos** of posets and monotone functions between posets (Definition 22.10).

In this category it is easy to propagate uncertainty if the uncertain sets are represented by intervals.

Recall that for a poset **P**, we can define the poset of intervals **Arr P** and **Arr P**.

Analogously to the previous case, here we want to generalize from monotone functions to nondeterministic monotone functions, where the uncertainty is represented by intervals.

We use the following notation for intervals:

- ▷ The interval

$$\{x : a \leq x \leq b\} \quad (46.11)$$

is denoted

$$[a, b] \quad (46.12)$$

- ▷ **L**· and **U**· extract the lower and upper bound from the interval, so that we

have

$$\textcolor{brown}{L}[a, b] = a \quad (46.13)$$

$$\textcolor{brown}{U}[a, b] = b \quad (46.14)$$

$$\begin{aligned} \textcolor{violet}{L}_P : \text{Arr } P &\longrightarrow_{\text{Pos}} P \\ [a, b] &\longmapsto a \end{aligned} \quad (46.15)$$

$$\begin{aligned} \textcolor{violet}{U}_P : \text{Arr } P^{\text{op}} &\longrightarrow_{\text{Pos}} P \\ [a, b] &\longmapsto b \end{aligned} \quad (46.16)$$

For example, one such function is

$$\begin{aligned} \textcolor{teal}{f} : \mathbb{R} &\longrightarrow \text{Arr } \mathbb{R}, \\ x &\longmapsto [x - 1, x + 1]. \end{aligned} \quad (46.17)$$

Because an interval is defined by two values, a function that returns an interval is a pair of functions, whose results are constrained to be ordered. In the example above, we always have $x - 1 \leq x + 1$.

We now retrace the steps of the previous example.

First, we need to ensure that we can see regular monotone functions as special cases of interval functions. For example, we want a function

$$\begin{aligned} \alpha : \langle \mathbb{N}, \leq \rangle &\longrightarrow \langle \mathbb{N}, \leq \rangle \\ x &\longmapsto x^2, \end{aligned} \quad (46.18)$$

can be rewritten as the function

$$\begin{aligned} \alpha' : \langle \mathbb{N}, \leq \rangle &\longrightarrow \text{Arr } \langle \mathbb{N}, \leq \rangle, \\ x &\longmapsto [x^2, x^2] \end{aligned} \quad (46.19)$$

Generically, this is the definition of the *lift* function

$$\begin{aligned} \text{lift}_{X,Y} : (\textcolor{red}{X} \rightarrow_{\text{Pos}} Y) &\longrightarrow (\textcolor{red}{X} \rightarrow_{\text{Pos}} \text{Arr } Y) \\ \textcolor{teal}{f} &\longmapsto \begin{cases} \text{lift}_{X,Y} f : X \longrightarrow \text{Arr } Y \\ x \longmapsto [\textcolor{blue}{f}(x), \textcolor{teal}{f}(x)] \end{cases} \end{aligned} \quad (46.20)$$

What is the "fish" function? Note that an interval-valued monotone map is also a special relation. Therefore, we want that composition continues to work in the same manner.

Therefore, we obtain for the *fish* operation:

$$\begin{array}{c} \underline{\mathfrak{f} : X \rightarrow \text{Arr } Y \quad g : Y \rightarrow \text{Arr } Z} \\ (\mathfrak{f} ; g) : X \longrightarrow \text{Arr } Z \\ x \mapsto [\mathbf{L}g(\mathbf{L}\mathfrak{f}(x)), \mathbf{U}g(\mathbf{U}\mathfrak{f}(x))] \end{array} \quad (46.21)$$

Upper sets

We have seen that to solve DPs we encountered functions of type

$$f : \mathbf{F} \rightarrow_{\mathbf{Pos}} \mathbf{U}\mathbf{R} \quad (46.22)$$

We can see this as another example of generalization from a functions

$$f : X \rightarrow_{\mathbf{Pos}} Y \quad (46.23)$$

to functions

$$f : X \rightarrow_{\mathbf{Pos}} \mathbf{U}Y \quad (46.24)$$

Once again we see these as particular types of relations.

The *lift* operation is

$$\begin{array}{c} lift_{X,Y} : (X \rightarrow_{\mathbf{Pos}} Y) \longrightarrow (X \rightarrow_{\mathbf{Pos}} \mathbf{U}Y) \\ \mathfrak{f} \mapsto \left\{ \begin{array}{l} lift_{X,Y} f : X \longrightarrow \mathbf{U}Y \\ \quad x \mapsto \uparrow \mathfrak{f}(x) \end{array} \right. \end{array} \quad (46.25)$$

The *fish* operation is

$$\begin{array}{c} \underline{\mathfrak{f} : X \rightarrow \mathbf{U}Y \quad g : Y \rightarrow \mathbf{U}Z} \\ (\mathfrak{f} ; g) : X \longrightarrow \mathbf{U}Z \\ x \mapsto \bigcup_{y \in \mathfrak{f}(x)} g(y) \end{array} \quad (46.26)$$

Note that the expression is the same as in (46.7) - only we are guaranteed to obtain upper sets.

Keeping track of resource usage

As another example, we consider the case where we want to attach additional information to a category.

Suppose we want to consider not functions, but *procedures* which have resource consumption. A function is a mathematical entity - a procedure is a program that *implements* a function. Suppose we want to model execution time - and, that execution time might depend on the input of the procedure.

In this case, we would like to extend a function

$$\mathfrak{f} : X \rightarrow Y \quad (46.27)$$

into a procedure

$$f : X \rightarrow (Y \times \mathbb{R}_{\geq 0}) \quad (46.28)$$

which gives both the result as well as the execution time.

We consider the ideal functions to be procedures that have zero execution time.

We can define the *lift* map as follows:

$$\begin{aligned} lift_{X,Y} : (X \rightarrow Y) &\longrightarrow (X \rightarrow Y \times \mathbb{R}_{\geq 0}) \\ f &\mapsto \begin{cases} lift_{X,Y} f : X \longrightarrow Y \times \mathbb{R}_{\geq 0} \\ x \mapsto \langle x, 0 \rangle \end{cases} \end{aligned} \quad (46.29)$$

As for the fish function we have

$$\begin{array}{c} \underline{f : X \rightarrow (Y \times \mathbb{R}_{\geq 0}) \quad g : Y \rightarrow (Z \times \mathbb{R}_{\geq 0})} \\ (f ; g) : X \longrightarrow (Z \times \mathbb{R}_{\geq 0}) \\ x \mapsto \langle g_1(f_1(x)), g_2(f_1(x)) + f_2(x) \rangle \end{array} \quad (46.30)$$

Note that we can recover the naked function by taking the first component of the tuple.

Generalization with monoid

This can be further generalized from the real numbers to an arbitrary monoid

$$\mathbf{M} = \langle M, \circ_M, \text{id}_M \rangle \quad (46.31)$$

structure.

$$\begin{aligned} lift_{X,Y} : (X \rightarrow Y) &\longrightarrow (X \rightarrow Y \times M) \\ f &\mapsto \begin{cases} lift_{X,Y} f : X \longrightarrow Y \times M \\ x \mapsto \langle x, \text{id}_M \rangle \end{cases} \end{aligned} \quad (46.32)$$

As for the fish function we have

$$\begin{array}{c} \underline{f : X \rightarrow (Y \times M) \quad g : Y \rightarrow (Z \times M)} \\ (f ; g) : X \longrightarrow (Z \times M) \\ x \mapsto \langle g_1(f_1(x)), g_2(f_1(x)) \circ_M f_2(x) \rangle \end{array} \quad (46.33)$$

46.2. Monads

In the previous section we considered the operation of taking the powerset of a set. This was used to think about generalized sets and generalized functions: given a set A , we thought of its powerset $\mathcal{P}A$ as a set whose elements – subsets of A – are “generalized elements of A ”. Furthermore we saw that it was possible

to generalize a function $f : \mathbf{A} \rightarrow \mathbf{B}$ to a function $\hat{f} : \mathbf{A} \rightarrow \mathcal{P}\mathbf{B}$, and that such generalized functions could be composed. In this section we will see that this construction is possible because the operation of “taking the powerset” has a special structure to it: it is an example of a *monad*.

Before introducing the the general definition, let us approach this concept by listing some of the properties that make the “powerset operation” a monad.

Firstly, note that the “powerset operation” can be viewed as functor $\mathbf{Set} \rightarrow \mathbf{Set}$. For each object – that is, for each set \mathbf{A} – it produces a new set, $\mathcal{P}\mathbf{A}$. For each function $f : \mathbf{A} \rightarrow \mathbf{B}$ we define $\mathcal{P}f : \mathcal{P}\mathbf{A} \rightarrow \mathcal{P}\mathbf{B}$ by

$$\mathbf{S} \mapsto \bigcup_{x \in \mathbf{S}} f(x). \quad (46.34)$$

In other words, $\mathcal{P}f$ maps a subset $\mathbf{S} \subseteq \mathbf{A}$ to its image under f , which is a subset of \mathbf{B} .

Graded exercise 1 (`PowersetImageFunctor`). Prove that “taking the powerset” defines a functor $\mathcal{P} : \mathbf{Set} \rightarrow \mathbf{Set}$ if we define its component on morphisms as in (46.34).

Second, as noted in the previous section, each element $x \in \mathbf{A}$ can be viewed as a “generalized element” $\{x\} \in \mathcal{P}\mathbf{A}$. So we have, for each set, a function $\text{un}_{\mathbf{A}} : \mathbf{A} \rightarrow \mathcal{P}\mathbf{A}, x \mapsto \{x\}$. Furthermore, these maps are “coherent” with morphisms in \mathbf{Set} in the sense that ... In other words, the diagram ... commutes. This means precisely that the maps ... are the components of a natural transformation

Third, consider the question of how to compose generalized morphisms – that is, how to define the mape *fish*... We saw that...

Definition 46.2 (Monad). Let \mathbf{C} be a category. A *monad* on \mathbf{C} is specified by:

Constituents

1. A functor $M : \mathbf{C} \rightarrow \mathbf{C}$;
2. A natural transformation $\text{mu} : M \circ M \Rightarrow M$, called the *composition* or *multiplication*;
3. A natural transformation $\text{un} : \text{Id}_{\mathbf{C}} \Rightarrow M$, called the *unit*.

Conditions

1. *Associativity*: the diagram

$$\begin{array}{ccc} M ; M ; M & \xrightarrow{M\text{mu}} & M ; M \\ \Downarrow \text{mu}M & & \Downarrow \text{mu} \\ M ; M & \xrightarrow{\text{mu}} & M \end{array}$$

must commute.

2. *Left and right unitality*: the diagrams

$$\begin{array}{ccc} M & \xrightarrow{\text{un}M} & M ; M \\ \text{Id}_M \searrow & \Downarrow \text{mu} & \Downarrow \text{mu} \\ M & & M \end{array}$$

must commute.

Remark 46.3. In terms of components, the unitality conditions state that for every object $X \in \text{Ob}_C$, the diagrams

$$\begin{array}{ccc} M(X) & \xrightarrow{\text{un}_M X} & (M ; M)(X) \\ \searrow \text{Id}_M & \downarrow \mu_X & \downarrow \mu_X \\ & M(X) & \end{array} \quad \begin{array}{ccc} M(X) & \xrightarrow{M \text{un}_X} & (M ; M)(X) \\ \searrow \text{Id}_M & \downarrow \mu_X & \downarrow \mu_X \\ & M(X) & \end{array}$$

commute. And the associativity condition states that for every object $X \in \text{Ob}_C$,

$$\begin{array}{ccc} (M ; M ; M)(X) & \xrightarrow{M \mu_X} & (M ; M)(X) \\ \downarrow \mu_{M(X)} & & \downarrow \mu_X \\ (M ; M)(X) & \xrightarrow{\mu_X} & M(X) \end{array}$$

commutes.

Graded exercise 2 (PowersetMonad). The aim of this exercise is to prove in full detail that the powerset functor $\mathcal{P} : \text{Set} \rightarrow \text{Set}$ from Graded exercise 1 is a monad, when equipped with the following *unit* and *multiplication*. We define $\text{un} : \text{Id}_{\text{Set}} \Rightarrow \mathcal{P}$ and $\mu : \mathcal{P} ; \mathcal{P} \Rightarrow \mathcal{P}$ in terms of components: given an object $A \in \text{Set}$, let

$$\mu_A : (\mathcal{P} ; \mathcal{P})(A) \rightarrow \mathcal{P}(A), D \mapsto \bigcup_{S \in D} S \quad (46.35)$$

and

$$\text{un}_A : A \rightarrow \mathcal{P}(A), x \mapsto \{x\}. \quad (46.36)$$

To show that $(\mathcal{P}, \mu, \text{un})$ is a monad,

1. prove that un , as defined in components in (46.36), is a natural transformation;
2. prove that μ , as defined in components in (46.35), is a natural transformation;
3. prove that un and μ satisfy the associativity condition and the left and right unitality conditions given in Definition 46.2. For this, work in components, as in Remark 46.3.

Exercise 29.

See solution on page 381.

$$\begin{aligned} \mathcal{P}f : \mathcal{P}X &\longrightarrow \mathcal{P}Y \\ S &\longmapsto \bigcup_{x \in S} f(x) \end{aligned} \quad (46.37)$$

$$\begin{aligned} \text{un}_X : X &\longrightarrow \mathcal{P}X \\ x &\longmapsto \{x\} \end{aligned} \tag{46.38}$$

$$\begin{aligned} \text{un}_{\mathcal{P}X} : \mathcal{P}X &\longrightarrow \mathcal{P}(\mathcal{P}X) \\ \{a, b, c\} &\longmapsto \{\{a, b, c\}\} \end{aligned} \tag{46.39}$$

$$\begin{aligned} \mathcal{P}(\text{un}_X) : \mathcal{P}X &\longrightarrow \mathcal{P}(\mathcal{P}X) \\ \{a, b, c\} &\longmapsto \{\{a\}, \{b\}, \{c\}\} \end{aligned} \tag{46.40}$$

$$\begin{aligned} \text{mu}_X : \mathcal{P}(\mathcal{P}X) &\longrightarrow \mathcal{P}X \\ \{S_1, S_2, \dots\} &\longmapsto \bigcup_i S_i \end{aligned} \tag{46.41}$$

$$\begin{aligned} \mathcal{P}\text{mu}_X : \mathcal{P}(\mathcal{P}(\mathcal{P}X)) &\longrightarrow \mathcal{P}(\mathcal{P}X) \\ \{\{\{a\}, \{b\}\}, \{\{c\}, \{d\}\}\} &\longmapsto \{\{a, b\}, \{c, d\}\} \end{aligned} \tag{46.42}$$

$$\begin{aligned} \text{mu}_{\mathcal{P}X} : \mathcal{P}(\mathcal{P}(\mathcal{P}X)) &\longrightarrow \mathcal{P}(\mathcal{P}X) \\ \{\{\{a\}, \{b\}\}, \{\{c\}, \{d\}\}\} &\longmapsto \{\{a\}, \{b\}, \{c\}, \{d\}\} \end{aligned} \tag{46.43}$$

$$\begin{aligned} \text{Arr } f : \text{Arr } X &\longrightarrow \text{Arr } Y \\ [a, b] &\longmapsto [\textcolor{blue}{f}(a), \textcolor{blue}{f}(b)] \end{aligned} \tag{46.44}$$

$$\begin{aligned} \text{un}_X : X &\longrightarrow \text{Arr } X \\ x &\longmapsto [x, x] \end{aligned} \tag{46.45}$$

$$\begin{aligned} \text{un}_{\text{Arr } X} : \text{Arr } X &\longrightarrow \text{Arr}(\text{Arr } X) \\ [a, b] &\longmapsto [[a, b], [a, b]] \end{aligned} \tag{46.46}$$

$$\begin{aligned} \text{Arr}(\text{un}_X) : \text{Arr } X &\longrightarrow \text{Arr}(\text{Arr } X) \\ [a, b] &\longmapsto [[a, a], [b, b]] \end{aligned} \tag{46.47}$$

$$\begin{aligned} \text{mu}_X : \text{Arr}(\text{Arr } X) &\longrightarrow \text{Arr } X \\ [[\alpha, \beta], [\gamma, \delta]] &\longmapsto [\min(\alpha, \gamma), \max(\beta, \delta)] = [\alpha, \delta] \end{aligned} \tag{46.48}$$

$$\begin{aligned} \text{Arr mu}_X : \text{Arr}(\text{Arr}(\text{Arr } X)) &\longrightarrow \text{Arr}(\text{Arr } X) \\ [[[[\alpha_1, \beta_1], [\gamma_1, \delta_1]], [[\alpha_2, \beta_2], [\gamma_2, \delta_2]]]] &\longmapsto [[[\alpha_1, \delta_1], [\alpha_2, \delta_2]]] \end{aligned} \tag{46.49}$$

$$\begin{aligned} \text{mu}_{\mathbf{Arr}X} : \mathbf{Arr}(\mathbf{Arr}(\mathbf{Arr}X)) &\longrightarrow \mathbf{Arr}(\mathbf{Arr}X) \\ [[[\alpha_1, \beta_1], [\gamma_1, \delta_1]], [[\alpha_2, \beta_2], [\gamma_2, \delta_2]]] &\longmapsto [[\alpha_1, \beta_1], [\gamma_2, \delta_2]] \end{aligned} \tag{46.50}$$

46.3. The Kleisli construction

We return now to the discussion from the opening section of this chapter, in order to spell out further the relationship to monads.

Definition 46.4 (Kleisli morphisms). Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} , and let $X, Y \in \text{Ob}_{\mathbf{C}}$. A *Kleisli morphism* $X \rightarrow Y$ is morphism of \mathbf{C} of the form $X \rightarrow MY$.

Definition 46.5 (Kleisli composition). Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} , let $X, Y, Z \in \text{Ob}_{\mathbf{C}}$, and let $f : X \rightarrow MY$ and $g : Y \rightarrow MZ$ be morphisms in \mathbf{C} (so, they are Kleisli morphisms). Their *Kleisli composition* is the morphism in \mathbf{C} given by the composition

$$X \xrightarrow{f} M(Y) \xrightarrow{Mg} (M ; M)(Z) \xrightarrow{\text{mu}_Z} M(Z). \tag{46.51}$$

Definition 46.6 (Kleisli category). Let $\langle M, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} . The *Kleisli category* \mathbf{C}_M of the monad M is specified by:

1. *Objects*: $\text{Ob}(\mathbf{C}_M) := \text{Ob}(\mathbf{C})$;
2. *Morphisms*: $\text{Hom}_{\mathbf{C}_M}(X, Y) := \text{Hom}_{\mathbf{C}}(X, M(Y))$;
3. *Identities*: $\text{Id}_X := \text{un}_X$;
4. *Composition*: Kleisli composition.

46.4. Algebras of a monad

In the context of Kleisli morphisms, we developed the intuition that monads can be used to encode “generalized objects” and “generalized morphisms”. In this section we will introduce a different intuition for monads: that they can be used to provide coherent way to encode “formal expressions”, together with a way to “evaluate” or “compute” such expressions.

Let us explain what we mean using an example. Given a set A , say

$$A = \{\text{apple}, \text{banana}, \text{carrot}\}, \tag{46.52}$$

let's define a certain type of "formal expressions", using elements of \mathbf{A} and using a "formal composition symbol", which we choose to be " $*$ ". Now, the "formal expressions" we'll consider are all finite expressions which have a form such as

$$\text{banana} * \text{banana} * \text{apple}, \quad (46.53)$$

or

$$\text{apple} * \text{banana} * \text{apple} * \text{apple} * \text{banana} * \text{apple}, \quad (46.54)$$

and so on. These expressions are "formal" (or "purely symbolic") in the sense that, a priori, the symbol " $*$ " does not have any *meaning* beyond simply being a symbol, a "marking". After all, so far, \mathbf{A} is just a set, and we did not assume that it comes equipped with any sort of "multiplication operation", for instance. In the following we will discuss a way of giving such formal expressions a "computational meaning" by specifying a way to evaluate them.

Before we come to this however, let us introduce a notation to explicitly distinguish when we are thinking about apple as an element of \mathbf{A} , or apple as a "formal expression". For the latter situation we'll write " $[\text{apple}]$ ". In other words, the square brackets indicate that $[\text{apple}]$ is a formal expression. And we'll say that formal expressions can be combined, using $*$, to larger formal expressions. So, following this convention, $[\text{apple}] * [\text{banana}]$ is also a formal expression. And in particular, instead of (46.54), we'll write

$$[\text{apple}] * [\text{banana}] * [\text{apple}] * [\text{apple}] * [\text{banana}] * [\text{apple}]. \quad (46.55)$$

In an expression such as (46.55) we'll think of the components $[\text{apple}]$ and $[\text{banana}]$ as if they are "letters" (but we won't count " $*$ " as a letter) and we'll think of the whole expression (46.55) like a "word". For any such word, we'll say its *length* is the number of letters it is built from. So, for instance, the word in (46.55) has a length of 6. In our notion of formal expression, we'll choose to include a unique special word of length 0, which we call the "empty formal expression" and denote by " $[]$ ".

Using square brackets we can also build "formal expressions of formal expressions" or "second-order formal expressions". For example, given the formal expression $[\text{apple}] * [\text{banana}]$, we can turn it into a second-order formal expression by putting brackets around it:

$$[[\text{apple}] * [\text{banana}]]. \quad (46.56)$$

And given another second-order formal expression, say $[[\text{banana}]] * [[\text{banana}]] * [\text{apple}]$, we can "compose" it with the one in (46.56) like so:

$$[[\text{apple}] * [\text{banana}]] * [[\text{banana}]] * [[\text{banana}]] * [\text{apple}]. \quad (46.57)$$

The notion of length will also apply to second-order expressions. For instance, the second-order expression (46.57) has length 2, and is composed of one first-order expression of length 2 and one first-order expression of length 3.

This whole game can continue ad infinitum: we define third-order formal expressions to be those with three-layers of square brackets, fourth-order formal expressions have four layers of brackets, and so on. In the following, we'll probably only ever consider up to three layers.

We started our story just with the set $\mathbf{A} = \{\text{apple}, \text{banana}, \text{carrot}\}$. However we can do the same construction – using " $*$ " and building formal expressions of any order –

with any set. In fact, we can define a functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ which maps any set \mathbf{A} to the set $F\mathbf{A}$ whose elements are all finite first-order formal expressions built from elements of \mathbf{A} . We also include this to mean the empty formal expression “[]”. What might this functor do on the level of morphisms? For concreteness, let $\mathbf{A} = \{\text{apple}, \text{banana}, \text{carrot}\}$ and $\mathbf{B} = \{\text{cup}, \text{mug}\}$. Given a function $f : \mathbf{A} \rightarrow \mathbf{B}$, we define

$$F(f) : F(\mathbf{A}) \rightarrow F(\mathbf{B}) \quad (46.58)$$

to act on (first-order) expressions like so

$$F(f)([\text{apple}] * [\text{banana}]) = [f(\text{apple})] * [f(\text{banana})]. \quad (46.59)$$

It turns out that this functor $F : \mathbf{Set} \rightarrow \mathbf{Set}$ can be made into a monad!

Let us explain how the unit and multiplication for this monad are defined. For any set \mathbf{A} , the component of the unit at \mathbf{A} is

$$\mathbf{un}_{\mathbf{A}} : \mathbf{A} \rightarrow F\mathbf{A}, x \mapsto [x]. \quad (46.60)$$

The multiplication is a bit more of a mouthful. It's component at \mathbf{A} ,

$$\mathbf{mu}_{\mathbf{A}} : (F \circledast F)\mathbf{A} \rightarrow F\mathbf{A}, \quad (46.61)$$

is the function which takes a second-order formal expression

$$[[x_{11}] * [x_{12}] * \dots * [x_{1k_1}]] * \dots * [[x_{n1}] * [x_{n2}] * \dots * [x_{nk_n}]] \quad (46.62)$$

and “collapses” it to the first-order expression

$$[x_{11}] * [x_{12}] * \dots * [x_{1k_1}] * \dots * [x_{n1}] * [x_{n2}] * \dots * [x_{nk_n}]. \quad (46.63)$$

In other words, this operation simply “removes the outer brackets” from a second-order formal expression.

Graded exercise 3 (ListMonad). Let $F : \mathbf{Set} \rightarrow \mathbf{Set}$ be the functor above that sends any set \mathbf{A} to the set of first-order formal expressions of the form

$$[x_1] * [x_2] * \dots * [x_n] \quad x_i \in \mathbf{A}, n \in \mathbb{Z}_{\geq 0},$$

and let $\mathbf{mu}_{\mathbf{A}}$ and $\mathbf{un}_{\mathbf{A}}$ be defined as above.

Show that:

1. the components $\mathbf{mu}_{\mathbf{A}}$ define a natural transformation $\mathbf{mu} : F \circledast F \Rightarrow F$;
2. the components $\mathbf{un}_{\mathbf{A}}$ define a natural transformation $\mathbf{un} : \mathbf{Id}_{\mathbf{Set}} \rightarrow F$;
3. \mathbf{mu} and \mathbf{un} satisfy the conditions for $\langle F, \mathbf{mu}, \mathbf{un} \rangle$ to be a monad.

Now let's finally talk about giving formal expressions a computational meaning: a way to *evaluate* them. The way we will do this is to define a notion of “evaluation map” $a : F\mathbf{A} \rightarrow \mathbf{A}$ which we will interpret as a way of specifying how any formal expression – an element of $F\mathbf{A}$ – should be evaluated (or: computed) to an element of \mathbf{A} . We will require such evaluation maps to additionally satisfy two coherence conditions, and the resulting mathematical structure will be what is

called an *algebra* of the monad \mathbf{F} .

Definition 46.7 (Algebra of a monad). Let $\langle \mathbf{M}, \text{mu}, \text{un} \rangle$ be a monad on a category \mathbf{C} . An algebra of \mathbf{M} (also called an \mathbf{M} -algebra) is specified by:

Constituents

1. an object X of \mathbf{C} ;
2. a morphism $a : \mathbf{M}(X) \rightarrow X$ of \mathbf{C} .

Conditions

1. *Composition*: the diagram

$$\begin{array}{ccc} (\mathbf{M} ; \mathbf{M})(X) & \xrightarrow{\mathbf{M}a} & \mathbf{M}(X) \\ \text{mu}_X \downarrow & & \downarrow a \\ \mathbf{M}(X) & \xrightarrow{a} & X \end{array}$$

commutes.

2. *Unit*: the diagram

$$\begin{array}{ccc} X & \xrightarrow{\text{un}_X} & \mathbf{M}(X) \\ & \searrow \text{Id} & \downarrow a \\ & & X \end{array}$$

commutes.

Definition 46.8 (\mathbf{M} -algebra morphism). Let $\langle \mathbf{M}, \text{un}, \text{mu} \rangle$ be a monad on a category \mathbf{C} , and let $\langle X_1, a_1 \rangle$ and $\langle X_2, a_2 \rangle$ be algebras of \mathbf{M} . A morphism $\langle X_1, a_1 \rangle \rightarrow \langle X_2, a_2 \rangle$ of \mathbf{M} -algebras is specified by:

Constituents

1. A morphism $f : X_1 \rightarrow X_2$ in \mathbf{C} .

Conditions

1. The diagram

$$\begin{array}{ccc} \mathbf{M}(X_1) & \xrightarrow{\mathbf{M}f} & \mathbf{M}(X_2) \\ a_1 \downarrow & & \downarrow a_2 \\ X_1 & \xrightarrow{f} & X_2 \end{array}$$

commutes.

Definition 46.9 (Category of \mathbf{M} -algebras). Let $\langle \mathbf{M}, \text{un}, \text{mu} \rangle$ be a monad on a category \mathbf{C} . The *category of \mathbf{M} -algebras* $\mathbf{C}^{\mathbf{M}}$ of the monad \mathbf{M} is specified by:

1. *Objects*: \mathcal{M} -algebras;
2. *Morphisms*: \mathcal{M} -algebra morphisms;
3. *Identities*: given an \mathcal{M} -algebra $\langle \mathcal{X}, a \rangle$, its identity morphism is $\text{Id}_{\mathcal{X}}$;
4. *Composition*: is induced by the composition of morphisms in \mathbf{C} .

46.5. Monads, computer science definition

Definition 46.10 (Monad in functional programming). A monad

$$\langle \text{return}, \text{join}, \text{fmap}, \text{bind}, \text{fish}, \text{lift} \rangle \quad (46.64)$$

is a set of operations with the following signature:

$$\begin{aligned} \text{return} &: \mathcal{X} \rightarrow \mathcal{M}\mathcal{X} \\ \text{lift} &: (\mathcal{X} \rightarrow \mathcal{Y}) \rightarrow (\mathcal{X} \rightarrow \mathcal{M}\mathcal{Y}) \\ \text{fish} &: (\mathcal{X} \rightarrow \mathcal{M}\mathcal{Y}) \rightarrow (\mathcal{Y} \rightarrow \mathcal{M}\mathcal{Z}) \rightarrow (\mathcal{X} \rightarrow \mathcal{M}\mathcal{Z}) \\ \text{join} &: \mathcal{M}\mathcal{M}\mathcal{X} \rightarrow \mathcal{M}\mathcal{X} \\ \text{fmap} &: (\mathcal{X} \rightarrow \mathcal{Y}) \rightarrow (\mathcal{M}\mathcal{X} \rightarrow \mathcal{M}\mathcal{Y}) \\ \text{bind} &: \mathcal{M}\mathcal{X} \rightarrow (\mathcal{X} \rightarrow \mathcal{M}\mathcal{Y}) \rightarrow \mathcal{M}\mathcal{Y} \end{aligned}$$

These maps satisfy the equivalent axioms of unitality and associativity:

- ▷ *return* is a left identity for *bind*;
- ▷ *return* is a right identity for *bind*;
- ▷ *bind* is associative.



47. Uncertainty

47.1 Introduction	369
47.2 UPos and LPos categories	369
47.3 From DP to UPos and LPos	373
47.4 \mathcal{L} and \mathcal{U} monads	373

47.1. Introduction

47.2. UPos and LPos categories

Definition 47.1 (Category **UPos**). The category **UPos** consists of:

1. *Objects*: objects are posets;
2. *Morphisms*: given objects $X, Y \in \text{Ob}_{\text{UPos}}$, morphisms from X to Y are monotone maps of the form $f : X \rightarrow \mathcal{U}Y$.
3. *Composition of morphisms*: Given morphisms $f : X \rightarrow \mathcal{U}Y$ $g : Y \rightarrow \mathcal{U}Z$, their composition is given as

$$f \circ g : X \rightarrow \mathcal{U}Z \\ x \mapsto \bigcup_{y \in f(x)} g(y); \quad (47.1)$$

4. *Identity morphism*: given an object $X \in \text{Ob}_{\text{UPos}}$, the identity morphism is given by the application of the upper closure operator: $\text{Id}_X(x) := \uparrow\{x\}$.

Remark 47.2. Note that the composition of morphisms in this category corre-

sponds to the generalization of the series operator for boolean profunctors.

Analogously, we can define the **LPos** category.

Definition 47.3 (Category **LPos**). The category **LPos** consists of:

1. *Objects*: objects are posets;
2. *Morphisms*: given objects $X, Y \in \text{Ob}_{\text{LPos}}$, morphisms from X to Y are monotone maps of the form $f : X \rightarrow \mathcal{L}Y$.
3. *Composition of morphisms*: Given morphisms $f : X \rightarrow \mathcal{L}Y, g : Y \rightarrow \mathcal{L}Z$, their composition is given by

$$\begin{aligned} f ; g : X &\rightarrow \mathcal{U}Z \\ x &\mapsto \bigcup_{y \in f(x)} g(y); \end{aligned} \tag{47.2}$$

4. *Identity morphism*: given an object $X \in \text{Ob}_{\text{LPos}}$, the identity morphism is given by the application of the lower closure operator: $\text{Id}_X(x) := \downarrow\{x\}$.

We now show that **UPos** and **LPos** are indeed categories.

Lemma 47.4. **UPos** and **LPos** are categories.

Proof. We prove that **UPos** is a category. The proof for **LPos** is analogous. In the following, we show unitality and associativity.

Unitality Given $f : X \rightarrow \mathcal{U}Y$, we have:

$$\begin{aligned} (f ; \text{Id}_Y)(x) &= \bigcup_{y \in f(x)} \text{Id}_Y(y) \\ &= \bigcup_{y \in f(x)} \uparrow\{y\} \\ &= \bigcup_{y \in f(x)} \{y' \in Y : y \leq_Y y'\} \end{aligned}$$

We know that $f(x)$ is an upperset:

$$\begin{aligned} f(x) &= \bigcup_{y \in f(x)} \{y\} \\ &= \bigcup_{y \in f(x)} \{y' \in Y : y \leq_Y y'\}. \end{aligned}$$

Therefore, $(f ; \text{Id}_Y)(x) = f(x)$ for all $x \in X$. Similarly, we have:

$$\begin{aligned} (\text{Id}_X ; f)(x) &= \bigcup_{x' \in \text{Id}_X(x)} f(x') \\ &= \bigcup_{x' \in \uparrow\{x\}} f(x') \\ &= f(x), \end{aligned}$$

where the last equality holds since f is a monotone function and $f(x') \subseteq f(x)$ for all $x' \in \uparrow\{x\}$.

Associativity Let's consider three morphisms $f : X \rightarrow \mathcal{U}Y$, $g : Y \rightarrow \mathcal{U}Z$, and $h : Z \rightarrow \mathcal{U}W$. We have:

$$\begin{aligned} ((f ; g) ; h)(x) &= \bigcup_{z \in (\bigcup_{y \in f(x)} g(y))} h(z) \\ &= \bigcup_{y \in f(x)} \bigcup_{z \in g(y)} h(z) \\ &= (f ; (g ; h))(x). \end{aligned}$$

Therefore, **UPos** is a category. \square

Definition 47.5 (Equivalence of categories). An *equivalence* between two categories **C** and **D** is given by:

1. A pair of functors F, G of the form:

$$\mathbf{C} \xleftrightarrow[F]{G} \mathbf{D} \quad (47.3)$$

2. natural isomorphisms $F ; G \cong \text{Id}_{\mathbf{C}}$ and $G ; F \cong \text{Id}_{\mathbf{D}}$.

We can show that **UPos** and **LPos** are equivalent.

Lemma 47.6. **UPos** and **LPos** are equivalent: there exists a pair of functors

$$\begin{array}{l} \swarrow : \mathbf{UPos} \rightarrow \mathbf{LPos}, \\ \nearrow : \mathbf{LPos} \rightarrow \mathbf{UPos}, \end{array} \quad (47.4)$$

such that $\swarrow ; \nearrow = \text{Id}_{\mathbf{UPos}}$ and $\nearrow ; \swarrow = \text{Id}_{\mathbf{LPos}}$, where $\text{Id}_{\mathbf{UPos}}$ and $\text{Id}_{\mathbf{LPos}}$ are the identity functors on **UPos** and **LPos**, respectively.

Proof. To prove this, we need to define the needed functors and to show that they satisfy the listed properties. We choose the functors to be the ones that map a poset **P** in a category to its opposite version **P**^{op} in another category. Given a morphism $f : X \rightarrow \mathcal{U}Y$ in **UPos**, we have:

$$\begin{aligned} \swarrow(f) : X^{\text{op}} &\rightarrow \mathcal{L}(Y^{\text{op}}) \\ x &\mapsto f(x). \end{aligned}$$

Given a morphism $g : X \rightarrow \mathcal{L}Y$ in **LPos**, we have:

$$\begin{aligned} \nearrow(g) : X^{\text{op}} &\rightarrow \mathcal{U}(Y^{\text{op}}) \\ x &\mapsto g(x). \end{aligned}$$

\swarrow and \nearrow are functors

- ▷ *Preservation of identities:* Given $X \in \text{Ob}_{\mathbf{UPos}}$, we have:

$$\begin{aligned} \swarrow(\text{Id}_X) &= \uparrow_X\{x\} \\ &= \downarrow_{X^{\text{op}}}\{x\} \\ &= \text{Id}_{X^{\text{op}}}, \end{aligned}$$

where Id_X is an identity morphism in **UPos**, and $\text{Id}_{X^{\text{op}}}$ is an identity

morphism in **LPos**. Similarly, given $X \in \text{Ob}_{\text{LPos}}$ we have:

$$\begin{aligned}\nearrow(\text{Id}_X) &= \downarrow_X\{\textcolor{red}{x}\} \\ &= \uparrow_{X^{\text{op}}}\{\textcolor{red}{x}\} \\ &= \text{Id}_{X^{\text{op}}}.\end{aligned}$$

▷ *Preservation of composition:* This can be easily seen as follows. Given any $f \in \text{Hom}_{\text{UPos}}(X; Y)$, $g \in \text{Hom}_{\text{UPos}}(Y; Z)$:

$$\begin{aligned}\nearrow(f \circ g) &= f \circ g \\ &= \nearrow(f) \circ \nearrow(g).\end{aligned}$$

Similarly, given any $f \in \text{Hom}_{\text{LPos}}(X; Y)$, $g \in \text{Hom}_{\text{LPos}}(Y; Z)$:

$$\begin{aligned}\nearrow(f \circ g) &= f \circ g \\ &= \nearrow(f) \circ \nearrow(g).\end{aligned}$$

Compositions return identity functors We want to show that by composing the two functors we obtain the identity functors in **UPos** and **LPos**, respectively. Clearly, comosing the two functors returns the identity on the objects, since for any poset P , one has $(P^{\text{op}})^{\text{op}} = P$. The functors act on morphisms by “flipping the context”, and “flipping” twice is the “same” as not flipping. □

We can show that **UPos** and **LPos** are monoidal categories.

Lemma 47.7. **UPos** is a monoidal category with the following additional structure:

1. *Tensor product*: On objects, the tensor product corresponds to the product of posets. Given two morphisms $f : X \rightarrow \mathcal{U}Y$ and $g : Z \rightarrow \mathcal{U}W$, we have:

$$\begin{aligned}f \circ g : X \times Z &\rightarrow \mathcal{U}(Y \times W) \\ \langle \textcolor{red}{x}, z \rangle &\mapsto f(x) \times g(z).\end{aligned}\tag{47.5}$$

Note that the Cartesian product of upper sets is an upper set.

2. *Unit*: The unit is the identity poset.
3. *Left unitor*: The left unitor is given by the pair of morphisms

$$\begin{aligned}\text{lu}_P : \{\bullet\} \circ X &\rightarrow \mathcal{U}X \\ \langle \bullet, x \rangle &\mapsto \uparrow\{\textcolor{red}{x}\},\end{aligned}\tag{47.6}$$

and

$$\begin{aligned}\text{lu}_P^{-1} : X &\rightarrow \mathcal{U}(\{\bullet\} \circ X) \\ x &\mapsto \{\bullet\} \times \uparrow\{x\}.\end{aligned}\tag{47.7}$$

4. *Right unitor*: The right unitor is given by the pair of morphisms

$$\begin{aligned}\text{ru}_P : X \circ \{\bullet\} &\rightarrow \mathcal{U}X \\ \langle x, \bullet \rangle &\mapsto \uparrow\{x\},\end{aligned}\tag{47.8}$$

and

$$\begin{aligned} \text{ru}_P^{-1} : X &\rightarrow \mathcal{U}(X \setminus \{\bullet\}) \\ x &\mapsto \uparrow\{x\} \times \{\bullet\}. \end{aligned} \quad (47.9)$$

5. *Associator*: The associator is given by the pair of morphisms:

$$\begin{aligned} \text{as}_{XY,Z} : (X \setminus Y) \setminus Z &\rightarrow \mathcal{U}X \times (\mathcal{U}Y \times \mathcal{U}Z) \\ \langle\langle x, y \rangle, z \rangle &\mapsto \uparrow\{x\} \times (\uparrow\{y\} \times \uparrow\{z\}), \end{aligned} \quad (47.10)$$

and

$$\begin{aligned} \text{as}_{X,YZ} : X \setminus (Y \setminus Z) &\rightarrow (\mathcal{U}X \times \mathcal{U}Y) \times \mathcal{U}Z \\ \langle x, \langle y, z \rangle \rangle &\mapsto (\uparrow\{x\} \times \uparrow\{y\}) \times \uparrow\{z\}. \end{aligned} \quad (47.11)$$

| Proof.

□

47.3. From DP to UPos and LPos

Lemma 47.8. There is a functor $\Pi_f : \mathbf{DP} \rightarrow \mathbf{UPos}$ which maps:

1. An object (poset) in **DP** to the same object (poset) in **UPos**.
2. A morphism $d \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$ to the morphism $h_d \in \text{Hom}_{\mathbf{UPos}}(\mathbf{F}; \mathbf{R})$, where:

$$\begin{aligned} h_d : \mathbf{F}^{\text{op}} &\rightarrow_{\mathbf{Pos}} \langle \mathcal{U}\mathbf{R}, \subseteq \rangle \\ f^* &\mapsto \{r \in \mathbf{R} \mid d(f^*, r) = \top\}. \end{aligned} \quad (47.12)$$

Lemma 47.9. There is a contravariant functor $\Pi_p : \mathbf{DP} \rightarrow \mathbf{LPos}$ which maps:

1. An object (poset) of **DP** to the same object (poset) in **LPos**.
2. A morphism $dp \in \text{Hom}_{\mathbf{DP}}(\mathbf{F}; \mathbf{R})$ to the morphism $g \in \text{Hom}_{\mathbf{LPos}}(\mathbf{R}; \mathbf{F})$, where:

$$\begin{aligned} g : \mathbf{R} &\rightarrow \mathcal{L}\mathbf{F} \\ r &\mapsto \{f \in \mathbf{F} \mid dp(f, r) = \top\}. \end{aligned}$$

47.4. \mathcal{L} and \mathcal{U} monads

In this section we propose another example of monads related to posets and upper/lower sets. We start by defining the \mathcal{U} endofunctor.

Definition 47.10 (\mathcal{U} endofunctor). The \mathcal{U} endofunctor has the form $\mathcal{U} : \mathbf{Pos} \rightarrow \mathbf{Pos}$ and acts on objects and morphisms as follows:

1. *On objects*: Given a poset $P \in \mathbf{Ob}_{\mathbf{Pos}}$, \mathcal{U} maps P to its upper set[†].
2. *On morphisms*: Given posets P, Q , and a monotone map $f : P \rightarrow Q$, the \mathcal{U} endofunctor acts as:

$$\begin{aligned} \mathcal{U}(f) : \mathcal{U}P &\rightarrow \mathcal{U}Q \\ P' &\mapsto \uparrow \left(\bigcup_{p \in P'} \{f(p)\} \right). \end{aligned} \quad (47.13)$$

We now want to prove that the \mathcal{U} endofunctor is an endofunctor, and the proof requires the following two facts.

[†] Recall that in Lemma 20.39 we proved that the upper set is itself an object of **Pos**.

Lemma 47.11. Given posets \mathbf{P}, \mathbf{Q} , a monotone map $f : \mathbf{P} \rightarrow \mathbf{Q}$, and a family of singleton sets $\{S_i\}_{i \in I}$, with $S_i = \{s_i\}$, $s_i \in \mathbf{P}$, the following equality holds:

$$\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right) = \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (47.14)$$

Proof. We first want to show that:

$$\underbrace{\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right)}_{\star} \subseteq \underbrace{\uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right)}_{\diamond}. \quad (47.15)$$

Let's take a

$$q \in \uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right). \quad (47.16)$$

If we have such a q , it means that there exists a

$$q' \in \bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \quad (47.17)$$

such that $q' \leq_Q q$, and hence there is a $p' \in \uparrow \bigcup_{i \in I} S_i$ such that $q' = f(p')$. Consequently, there must exist an $i' \in I$ such that $s_{i'} \leq_P p'$. The monotonicity of f implies:

$$f(s_{i'}) \leq_P f(p') = q' \leq_Q q. \quad (47.18)$$

We know that $s_{i'} \in \diamond$ and any $q^* \in \mathbf{Q}$ satisfying $f(s_{i'}) \leq_Q q^*$ belongs to $\uparrow \diamond$. Therefore, $\star \subseteq \uparrow \diamond$, which proves the validity of (47.15).

We now want to show that:

$$\uparrow \left(\bigcup_{p \in \uparrow \bigcup_{i \in I} S_i} \{f(p)\} \right) \supseteq \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right). \quad (47.19)$$

By now taking a

$$q \in \uparrow \left(\bigcup_{i \in I} \{f(s_i)\} \right), \quad (47.20)$$

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 $q^* \leq_Q f(s_{i'})$ must be in $\uparrow \diamond$, meaning that $q \in \star$, and proving the validity of (47.19).

The validity of (47.15) and (47.19) implies (47.14). \square

Lemma 47.12. Given posets \mathbf{P}, \mathbf{Q} and a monotone map $f : \mathbf{P} \rightarrow \mathbf{Q}$, we have:

$$\uparrow \left(\bigcup_{p' \in \uparrow \{p\}} \{f(p')\} \right) = \uparrow \{f(p)\}. \quad (47.21)$$

Proof. The proof follows from Lemma 47.11, by considering a family of singleton sets consisting solely of the set $\{p\}$. \square

We can now show that the \mathcal{U} endofunctor is indeed a functor.

Lemma 47.13. The \mathcal{U} endofunctor is indeed a functor.

Proof. \mathcal{U} has a valid form and given a poset \mathbf{P} , maps $\text{Id}_{\mathbf{P}}$ to $\text{Id}_{\mathcal{U}\mathbf{P}}$. We now need to show that \mathcal{U} fulfills morphism composition. Consider maps $f : \mathbf{P} \rightarrow \mathbf{Q}$ and $g : \mathbf{Q} \rightarrow \mathbf{R}$. We have:

$$\begin{aligned} \mathcal{U}(f \circ g) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}' &\mapsto \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{g(f(p))\} \right). \end{aligned} \quad (47.22)$$

On the other hand, we have:

$$\begin{aligned} \mathcal{U}(f) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}' &\mapsto \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{f(p)\} \right), \end{aligned} \quad (47.23)$$

and

$$\mathcal{U}(g) : \mathbf{Pos} \rightarrow \mathbf{Pos} \quad (47.24)$$

$$\mathbf{Q}' \mapsto \uparrow \left(\bigcup_{q \in \mathbf{Q}'} \{g(q)\} \right),$$

leading to

$$\begin{aligned} \mathcal{U}(f) \circ \mathcal{U}(g) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{Q}' &\mapsto \uparrow \left(\bigcup_{q \in \uparrow \bigcup_{p \in \mathbf{Q}'} \{f(p)\}} \{g(q)\} \right) \\ &\mapsto \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{g(f(p))\} \right). \end{aligned} \quad (47.25)$$

(Lemma 47.11)

Since (47.22) and (47.25) are equivalent, \mathcal{U} is a functor. \square

Having proven that \mathcal{U} is a valid functor, we are now ready to define the \mathcal{U} monad.

Definition 47.14 (\mathcal{U} monad). The \mathcal{U} monad on \mathbf{Pos} consists of:

1. The \mathcal{U} endofunctor (Def. 47.10).
2. The unit natural transformation $\text{un}_{\mathcal{U}} : \text{Id}_{\mathbf{Pos}} \Rightarrow \mathcal{U}$, which associates to every object $\mathbf{P} \in \mathbf{Ob}_{\mathbf{Pos}}$ a morphisms in \mathbf{Pos} given by:

$$\begin{aligned} \text{un}_{\mathcal{U}}^{\mathbf{P}} : \mathbf{P} &\rightarrow \mathcal{U}\mathbf{P} \\ p &\mapsto \uparrow \{p\}. \end{aligned} \quad (47.26)$$

3. The compositional natural transformation $\text{mu}_{\mathcal{U}} : \mathcal{U} \circ \mathcal{U} \Rightarrow \mathcal{U}$, which associates to every $\mathbf{P} \in \mathbf{Ob}_{\mathbf{Pos}}$ the morphism in \mathbf{Pos} given by:

$$\begin{aligned} \text{mu}_{\mathcal{U}}^{\mathbf{P}} : \mathcal{U}(\mathcal{U}\mathbf{P}) &\rightarrow \mathcal{U}\mathbf{P} \\ \mathbf{P}'' &\mapsto \bigcup_{\mathbf{P}' \in \mathbf{P}''} \mathbf{P}'. \end{aligned} \quad (47.27)$$

Before showing that this indeed defines a monad, we list some results which will be instrumental later.

Lemma 47.15. Given a poset \mathbf{P} and a family of upper sets of \mathbf{P} , denoted $\{\mathbf{P}_i\}_{i \in \mathbf{I}}$ (with index set \mathbf{I}), their union is also an upper set.

Lemma 47.16. Given a set \mathbf{A} and a family of subsets of \mathbf{A} , denoted $\{\mathbf{A}_i\}_{i \in \mathbf{I}}$ (with index set \mathbf{I}), one has:

$$\bigcup_{i \in \mathbf{I}} \bigcup_{i \in \mathbf{I}} \{\mathbf{A}_i\} = \bigcup_{i \in \mathbf{I}} \{\mathbf{A}_i\}. \quad (47.28)$$

Lemma 47.17. The \mathcal{U} monad is indeed a monad.

Proof. To show that \mathcal{U} is indeed a monad, we need to show the following:

1. $\text{un}_{\mathcal{U}}$ is a natural transformation;
2. $\text{mu}_{\mathcal{U}}$ is a natural transformation;
3. left unitality holds;
4. right unitality holds;
5. associativity holds;

We prove them in order.

1) $\text{un}_{\mathcal{U}}$ is a natural transformation: We need to show that for any $f \in \mathbf{Hom}_{\mathbf{Pos}}(\mathbf{P}; \mathbf{Q})$, we have:

$$\text{Id}_{\mathbf{Pos}}(f) \circ \text{un}_{\mathcal{U}}^{\mathbf{Q}} = \text{un}_{\mathcal{U}}^{\mathbf{P}} \circ \mathcal{U}(f). \quad (47.29)$$

By expanding the left-hand side, we obtain:

$$[\text{Id}_{\mathbf{Pos}}(f) \circ \text{un}_{\mathcal{U}}^{\mathbf{Q}}](\mathbf{p}) = \uparrow \{f(\mathbf{p})\}. \quad (47.30)$$

By expanding the right-hand side, we get:

$$\begin{aligned} \text{un}_{\mathcal{U}}^{\mathbf{P}} : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{p} &\mapsto \uparrow \{\mathbf{p}\}. \end{aligned} \quad (\text{Lemma 47.12}) \quad (47.31)$$

and

$$\begin{aligned} \mathcal{U}(f) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}' &\mapsto \uparrow \bigcup_{\mathbf{p}' \in \mathbf{P}'} \{f(\mathbf{p}')\}, \end{aligned} \quad (47.32)$$

and hence

$$\begin{aligned} [\text{un}_{\mathcal{U}}^{\mathbf{P}} \circ \mathcal{U}(f)](\mathbf{p}) &= \uparrow \left(\bigcup_{\mathbf{p}' \in \uparrow \{\mathbf{p}\}} \{f(\mathbf{p}')\} \right) \\ &= \uparrow \{f(\mathbf{p})\}. \end{aligned} \quad (47.33)$$

2) $\text{mu}_{\mathcal{U}}$ is a natural transformation: We want to show that for every $f \in$

$\text{Hom}_{\mathbf{Pos}}(\mathbf{P}; \mathbf{Q})$, one has:

$$\mathbf{U}(\mathbf{U}(f)) ; \mathbf{mu}_{\mathbf{U}}^{\mathbf{Q}} = \mathbf{mu}_{\mathbf{U}}^{\mathbf{P}} ; \mathbf{U}(f). \quad (47.34)$$

Let's start with the left-hand side. We first look at $\mathbf{U}(\mathbf{U}(f))$:

$$\begin{aligned} (\mathbf{U}(\mathbf{U}(f))) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}'' &\mapsto \uparrow \left(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \left\{ \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{f(p)\} \right) \right\} \right). \end{aligned} \quad (47.35)$$

Furthermore, one has:

$$\begin{aligned} \mathbf{U}(\mathbf{U}(f)) ; \mathbf{mu}_{\mathbf{U}}^{\mathbf{Q}} : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}'' &\mapsto \bigcup_{\mathbf{Q}' \in \uparrow(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \{\uparrow(\bigcup_{p \in \mathbf{P}'} \{f(p)\})\})} \mathbf{Q}' \\ &\mapsto \bigcup \uparrow \left(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \left\{ \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{f(p)\} \right) \right\} \right) \\ &\mapsto \bigcup \left(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \left\{ \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{f(p)\} \right) \right\} \right) \quad (\text{Lemma 47.15}) \\ &\mapsto \bigcup_{\mathbf{P}' \in \mathbf{P}''} \left\{ \uparrow \left(\bigcup_{p \in \mathbf{P}'} \{f(p)\} \right) \right\} \quad (\text{Lemma 47.16}) \\ &\mapsto \uparrow \left(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \bigcup_{p \in \mathbf{P}'} \{f(p)\} \right), \end{aligned} \quad (47.36)$$

where we used the fact that the union of upper sets is the upper closure of the union of sets. By looking at the right-hand side, one has:

$$\begin{aligned} (\mathbf{mu}_{\mathbf{U}}^{\mathbf{P}} ; \mathbf{U}(f)) : \mathbf{Pos} &\rightarrow \mathbf{Pos} \\ \mathbf{P}'' &\mapsto \uparrow \left(\bigcup_{p \in \bigcup_{\mathbf{P}' \in \mathbf{P}''} \mathbf{P}'} \{f(p)\} \right) \\ &\mapsto \uparrow \left(\bigcup_{\mathbf{P}' \in \mathbf{P}''} \bigcup_{p \in \mathbf{P}'} \{f(p)\} \right), \end{aligned} \quad (47.37)$$

proving that $\mathbf{mu}_{\mathbf{U}}$ is a valid natural transformation.

Left unitality holds: We want to show that given a poset $\mathbf{P} \in \mathbf{Ob}_{\mathbf{Pos}}$, one has:

$$\mathbf{un}_{\mathbf{U}}^{\mathbf{U}(\mathbf{P})} ; \mathbf{mu}_{\mathbf{U}}^{\mathbf{P}} = \mathbf{Id}_{\mathbf{Pos}}^{\mathbf{U}(\mathbf{P})}. \quad (47.38)$$

One has:

$$\begin{aligned} \text{un}_{\mathcal{U}}^{U(\mathbf{P})} ; \text{mu}_{\mathcal{U}}^{\mathbf{P}} : \mathcal{U}\mathcal{U} &\rightarrow \mathcal{U}\mathcal{U} \\ \mathbf{P}' &\mapsto \bigcup_{\mathbf{P}'' \in \uparrow\{\mathbf{P}'\}} \mathbf{P}'' \\ &\mapsto \mathbf{P}', \end{aligned} \quad (47.39)$$

because the upper sets of an upper set \mathbf{P}' are subsets of \mathbf{P}' (and hence their union is \mathbf{P}').

Right unitality holds: We want to show that given a poset $\mathbf{P} \in \text{Ob}_{\text{Pos}}$, one has:

$$U(\text{un}_{\mathcal{U}}^{\mathbf{U}}) ; \text{mu}_{\mathcal{U}}^{\mathbf{P}} = \text{Id}_{\text{UPos}}^{U(\mathbf{P})}. \quad (47.40)$$

One has:

$$\begin{aligned} U(\text{un}_{\mathcal{U}}^{\mathbf{U}}) ; \text{mu}_{\mathcal{U}}^{\mathbf{P}} : \mathcal{U}\mathbf{P} &\rightarrow \mathcal{U}\mathbf{P} \\ \mathbf{P}' &\mapsto \bigcup_{\mathbf{P}'' \in \uparrow(\bigcup_{p \in \mathbf{P}'} \uparrow\{p\})} \mathbf{P}'' \\ &\mapsto \bigcup \uparrow \left(\bigcup_{p \in \mathbf{P}'} \uparrow\{p\} \right) \\ &\mapsto \uparrow \bigcup_{p \in \mathbf{P}'} \uparrow\{p\} \\ &\mapsto \uparrow \bigcup_{p \in \mathbf{P}'} \uparrow\{p\} \\ &\mapsto \uparrow \mathbf{P}' \\ &\mapsto \mathbf{P}', \end{aligned}$$

where we use the facts that the union of upper sets is an upper set, the union of an upper sets is the upper closure of the union of sets, and that the upper closure of an upper set returns the upper set.

Associativity holds: We want to show that given a poset $\mathbf{P} \in \text{Ob}_{\text{Pos}}$, one has:

$$U(\text{mu}_{\mathcal{U}}^{\mathbf{P}}) ; \text{mu}_{\mathcal{U}}^{\mathbf{P}} = \text{mu}_{\mathcal{U}}^{U(\mathbf{P})} ; \text{mu}_{\mathcal{U}}^{\mathbf{P}}. \quad (47.41)$$

We start by exploding the left-hand side of the equation. First, one has:

$$U(\text{mu}_{\mathcal{U}}^{\mathbf{P}})(\mathbf{R}) = \uparrow \left(\bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}' \right).$$

Therefore, we have:

$$\begin{aligned}
 (\textcolor{violet}{U}(\text{mu}_{\mathcal{U}}^{\mathbf{P}}) ; \text{mu}_{\mathcal{U}}^{\mathbf{P}})(\mathbf{R}) &= \bigcup_{\mathbf{S} \in \uparrow(\bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}')} \mathbf{S} \\
 &= \bigcup \uparrow \left(\bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}' \right) \\
 &= \uparrow \bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}' \\
 &= \uparrow \bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}' \\
 &= \bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}'
 \end{aligned}$$

Starting from right-hand side, instead, one has:

$$\begin{aligned}
 (\text{mu}_{\mathcal{U}}^{U(\mathbf{P})} ; \text{mu}_{\mathcal{U}}^{\mathbf{P}})(\mathbf{R}) &= \bigcup_{\mathbf{P}' \in \bigcup_{\mathbf{Q} \in \mathbf{R}} \mathbf{Q}} \mathbf{P}' \\
 &= \bigcup_{\mathbf{Q} \in \mathbf{R}} \bigcup_{\mathbf{P}' \in \mathbf{Q}} \mathbf{P}'.
 \end{aligned}$$

□

Lemma 47.18. **UPos** is the Kleisli category of \mathcal{U} .

48. Solutions to selected exercises

Solution of Exercise 29.



49. Enrichments

49.1. Enriched categories

49.1 Enriched categories 383

Definition 49.1 (Enriched category). Let $\langle \mathbf{V}, \otimes, \mathbf{1}, \text{as}, \text{lu}, \text{ru} \rangle$ be a monoidal category. A category \mathbf{C} enriched in \mathbf{V} is composed of:

1. The set of objects $\text{Ob}_{\mathbf{C}}$;
2. For all $X, Y \in \text{Ob}_{\mathbf{C}}$, an object $\text{Hom}_{\mathbf{C}}(X; Y)$, called the *hom-object* from X to Y .
3. For all $X, Y, Z \in \text{Ob}_{\mathbf{C}}$, there exists a morphism $\circ_{X,Y,Z}$ in \mathbf{V} :

$$\circ_{X,Y,Z} : \text{Hom}_{\mathbf{C}}(X; Y) \otimes \text{Hom}_{\mathbf{C}}(Y; Z) \rightarrow \text{Hom}_{\mathbf{C}}(X; Z). \quad (49.1)$$

This is called *composition morphism*.

4. For each $X \in \text{Ob}_{\mathbf{C}}$, a morphism $\text{Id}_X : \mathbf{1} \rightarrow \text{Hom}_{\mathbf{C}}(X; X)$, called *identity element*.

Furthermore, for any $X, Y, Z, W \in \text{Ob}_{\mathbf{C}}$, the following diagrams must commute.

$$\begin{array}{ccc}
 \text{Hom}_{\mathbf{C}}(X; Y) \otimes (\text{Hom}_{\mathbf{C}}(Y; Z) \otimes \text{Hom}_{\mathbf{C}}(Z; W)) & \xrightarrow{\text{as}} & (\text{Hom}_{\mathbf{C}}(X; Y) \otimes \text{Hom}_{\mathbf{C}}(Y; Z)) \otimes \text{Hom}_{\mathbf{C}}(Z; W) \\
 \text{Id}_{\mathbf{C}} \otimes m_{Y,Z,W} \downarrow & & \downarrow m_{X,Y,Z} \otimes \text{Id}_{\mathbf{C}} \\
 \text{Hom}_{\mathbf{C}}(X; Y) \otimes \text{Hom}_{\mathbf{C}}(Y; W) & \xrightarrow{m_{X,Y,W}} & \text{Hom}_{\mathbf{C}}(X; W) \xleftarrow{m_{X,Z,W}} \text{Hom}_{\mathbf{C}}(X; Z) \otimes (\text{Hom}_{\mathbf{C}}(Z; W)) \\
 \\
 \text{Hom}_{\mathbf{C}}(X; Y) \otimes \text{Hom}_{\mathbf{C}}(Y; Y) & \xrightarrow{m_{X,Y,Y}} & \text{Hom}_{\mathbf{C}}(X; Y) \xleftarrow{m_{X,X,Y}} \text{Hom}_{\mathbf{C}}(X; X) \otimes \text{Hom}_{\mathbf{C}}(X; Y) \\
 \text{Id}_{\text{Hom}_{\mathbf{C}}(X; Y)} \otimes j_Y \uparrow & \xrightarrow{\text{ru}} & \uparrow j_X \otimes \text{Id}_{\text{Hom}_{\mathbf{C}}(X; Y)} \\
 \text{Hom}_{\mathbf{C}}(X; Y) \otimes \mathbf{1} & & \mathbf{1} \otimes \text{Hom}_{\mathbf{C}}(X; Y) \xleftarrow{\text{lu}}
 \end{array}$$

Definition 49.2. Let \mathbf{A} and \mathbf{B} be categories enriched in a symmetric monoidal category \mathbf{V} . Their *product* is a \mathbf{V} -enriched category $\mathbf{A} \times \mathbf{B}$ with:

1. $\text{Ob}_{\mathbf{A} \times \mathbf{B}} := \text{Ob}_{\mathbf{A}} \times \text{Ob}_{\mathbf{B}}$;
2. $\text{Hom}_{\mathbf{A} \times \mathbf{B}}(\langle X, Y \rangle; \langle X', Y' \rangle) := \text{Hom}_{\mathbf{A}}(X; X') \otimes \text{Hom}_{\mathbf{B}}(Y; Y')$, for two objects $\langle X, Y \rangle$ and $\langle X', Y' \rangle$ in $\text{Ob}_{\mathbf{A} \times \mathbf{B}}$.

Graded exercise 4 (`DPIsEnrichedInPos`). Show that the category of design problems (objects are posets and morphisms are **Bool**-profunctors) can be viewed as a category enriched in the monoidal category of posets.

The monoidal product $P_1 \times P_2$ of posets is defined straightforwardly as the product poset. The monoidal unit in this category is any choice of 1-element poset.

Graded exercise 5 (`ProductOfEnrichedCats`). Let \mathbf{V} be a symmetric monoidal category, and let \mathbf{C} and \mathbf{D} be \mathbf{V} -enriched categories. Write down a definition of the cartesian product $\mathbf{C} \times \mathbf{D}$ of \mathbf{C} and \mathbf{D} such that $\mathbf{C} \times \mathbf{D}$ is again a \mathbf{V} -enriched category, and provide a proof that this is indeed true for your definition!

Is it needed that \mathbf{V} be *symmetric* monoidal? If yes, where is symmetry needed?

Definition 49.3 (Enriched functor). Given two categories \mathbf{C} and \mathbf{D} enriched in the same monoidal category \mathbf{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 \mathbf{C} to objects of \mathbf{D} .
2. For each X, Y in $\text{Ob}_{\mathbf{C}}$, there exists a morphism in \mathbf{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 \mathbf{C} and the unit in \mathbf{V} in the obvious ways:

$$\begin{array}{ccc}
 \text{Hom}_{\mathbf{C}}(X; Y) \otimes \text{Hom}_{\mathbf{C}}(Y; Z) & \xrightarrow{m_{X,Y,Z}} & \text{Hom}_{\mathbf{C}}(X; Z) \\
 F_{X,Y} \otimes F_{Y,Z} \downarrow & & \downarrow F_{X,Z} \\
 \text{Hom}_{\mathbf{D}}(F(Y); F(Z)) \otimes \text{Hom}_{\mathbf{D}}(F(X); F(Y)) & \xrightarrow{m_{F(X), F(Y), F(Z)}} & \text{Hom}_{\mathbf{D}}(F(X); F(Z))
 \end{array}$$

and

$$\begin{array}{ccc}
 \mathbf{1} & \searrow j_{F(X)} & \\
 \downarrow j_X & & \\
 \text{Hom}_{\mathbf{C}}(X; X) & \xrightarrow{F_{X,X}} & \text{Hom}_{\mathbf{D}}(F(X); F(X))
 \end{array}$$

where \otimes and $\mathbf{1}$ are the monoidal product and monoidal unit in \mathbf{V} .



50. Operads

Definition 50.1 (Operad). An *operad* is defined by:

Constituents

1. *Objects*: A collection $\text{Ob}_{\mathcal{O}}$;
2. *Morphisms*: Let $n \in \mathbb{N}$. For each finite string $[X_1, \dots, X_n]$ of objects and each object Y , one specifies a set $\text{Hom}_{\mathcal{O}}([X_1, \dots, X_n]; Y)$, elements of which are morphisms $[X_1, \dots, X_n] \rightarrow Y$;
3. *Identity morphisms*: For each object X , a morphism $\text{Id}_X \in \text{Hom}_{\mathcal{O}}([X]; Y)$;
4. *Composition operations*:

$$\text{Hom}_{\mathcal{O}}([X_1^1, \dots, X_{n_1}^1]; Y_1) \times \dots \times \text{Hom}_{\mathcal{O}}([X_1^m, \dots, X_{n_m}^m]; Y_m) \times \text{Hom}_{\mathcal{O}}([Y_1, \dots, Y_m]; Z) \xrightarrow{\quad} \text{Hom}_{\mathcal{O}}([X_1^1, \dots, X_{n_m}^m]; Z) \\ \langle f_1, \dots, f_m, g \rangle \mapsto [f_1, \dots, f_m] \circ g. \quad (50.1)$$

Conditions

1. *Associativity*:

$$[[f_1^1, \dots, f_{n_1}^1] \circ g_1, [f_1^2, \dots, f_{n_2}^2] \circ g_2, \dots, [f_1^m, \dots, f_{n_m}^m] \circ g_m] \circ h = [f_1^1, \dots, f_{n_m}^m] \circ ([g_1, \dots, g_m] \circ h). \quad (50.2)$$

2. *Unitality:*

$$[\text{Id}_{\textcolor{red}{X}_1}, \dots, \text{Id}_{\textcolor{red}{X}_n}] ; f = f = f ; \text{Id}_{\textcolor{red}{Y}}, \quad \forall f : [\textcolor{red}{X}_1, \dots, \textcolor{red}{X}_n] \rightarrow \textcolor{red}{Y}. \quad (50.3)$$

Remark 50.2. For non-strict monoidal categories it is also possible to define an associated operad similarly. There are in fact various ways of going about this. One way is to choose a convention about bracketing: for instance one might set

$$\text{Hom}_{\mathcal{O}_C}([\textcolor{red}{X}_1, \dots, \textcolor{red}{X}_n]; \textcolor{red}{Y}) = \text{Hom}_C((\dots(\textcolor{red}{X}_1 \otimes \textcolor{red}{X}_2) \dots \otimes \textcolor{red}{X}_n); \textcolor{red}{Y})$$

by putting all brackets in the left-most manner. Another approach is to invoke a well-known theorem (called the “strictification theorem” for monoidal categories) that says that every monoidal category is monoidally equivalent to a strict monoidal category. Then, one can assume that one is working with the “strict version” of a given monoidal category. We leave out discussing these technical details any further here. When using operads associated with monoidal categories we will sometimes simply act “as if” our monoidal categories are strict, even when they are not.

Example 50.3. Let $\mathbf{C} = \mathbf{Set}$ be the category of sets and functions, and view it as equipped with the monoidal structure induced by the cartesian product. This category is *not* strict monoidal as is, however, in light of Remark 50.2, we can still define an associated operad. As mentioned there, we will sweep the details under the rug for the time-being and act as if the category \mathbf{Set} were indeed strict monoidal. The associated operad that we obtain from the monoidal category \mathbf{Set} will also be called \mathbf{Set} .

Definition 50.4. Let \mathcal{O}, \mathcal{P} be operads. A *functor* between operads $F : \mathcal{O} \rightarrow \mathcal{P}$ is composed of:

1. A function $F_{\text{ob}} : \text{Ob}_{\mathcal{O}} \rightarrow \text{Ob}_{\mathcal{P}}$;
2. A function $F_{\text{mor}} : \text{Hom}_{\mathcal{O}}([\textcolor{red}{X}_1, \dots, \textcolor{red}{X}_n]; \textcolor{red}{Y}) \rightarrow \text{Hom}_{\mathcal{P}}([F_{\text{ob}}(\textcolor{red}{X}_1), \dots, F_{\text{ob}}(\textcolor{red}{X}_n)]; F_{\text{ob}}(\textcolor{red}{Y}))$.

These constituents must satisfy conditions which encode compatibility with the composition operations and with identity morphisms; these conditions are analogous to the ones in the definition of a functor between categories.

Definition 50.5. Let \mathcal{O} be an operad. An *algebra* for \mathcal{O} is a functor of operads $\mathcal{O} \rightarrow \mathbf{Set}$.

Graded exercise 6 (MonoidsAsAlgebras). Let \mathcal{O} be the following operad. It has just a single object, which we call \star . For each natural number $n \in \mathbb{N}$ (including 0), we let $[n]$ denote the list consisting of n copies of the object \star (so $[0]$ in this case is a synonym for the empty list). With this notation, we define $\text{Hom}_{\mathcal{O}}([n]; 1)$ to be the 1-element set $\{\star\}$ for each $n \in \mathbb{N}$. The intuition is that, for each n , $\text{Hom}_{\mathcal{O}}([n]; 1)$ represents a single n -ary operation.

In this exercise, the task is to describe what an algebra for this operad is! Try to prove as many of the statements that you make as you can.

Graded exercise 7 ([MonoidActionsAsAlgebras](#)). For the duration of this exercise, we fix a monoid $\textcolor{violet}{M}$. We define an \mathcal{O} as follows. It has a single object, which we call \star . For each natural number $n \in \mathbb{N}$ (including 0), let $[n]$ denote the list consisting of n copies of the object \star . We define $\text{Hom}_{\mathcal{O}}([n]; 1)$ to be the set underlying $\textcolor{violet}{M}$ if $n = 1$, and we set $\text{Hom}_{\mathcal{O}}([n]; 1) = \emptyset$ if $n \neq 1$.

Can you describe what an algebra for this operad is? Please prove your statements as best you can.

BACK MATTER



Solutions to selected exercises	393
Nomenclature	395
Tables and maps	407
Example exams	415

Solutions to selected exercises

Nomenclature

Sets

Generic sets and elements

A, B, C	Generic names for sets.
C	
S, T	Generic names for subsets.
<i>a</i>	
<i>b</i>	
<i>c</i>	
<i>d</i>	
<i>x, y, z</i>	Generic names for elements of sets.
<i>f, g, h</i>	Generic names for maps between sets.
cod	
dom	

Well-known sets.

C	Complex numbers
R	Real numbers
N	Natural numbers: 0, 1, 2, ...
Z	Integers: 0, 1, -1, 2, -2, ...
Q	Rational numbers
R_{>0}	Positive real numbers
R_{≥0}	Non-negative real numbers
̄R_{≥0}	Completion of non-negative real numbers.
•	element of the singleton
{•}	singleton set

Constructors

P_A	Power set of A .	→ Example 20.12	162
----------------------	-------------------------	-----------------	-----

Operations

A × B	Cartesian product of two sets.	→ Def. 24.1	187
A + B	Disjoint union of two sets.	→ Def. 24.22	195
⟨1, a⟩, ⟨1, b⟩	Decorated elements of disjoint union	→ Def. 24.22	195
l₁, l₂	Injections into A + B .	→ Def. 24.22	195

Well-known functions

Id_A	Identity map on A		
[x]	Rounding of <i>x</i> to the next integer		
ceil			
floor		→ Example 22.5	178
rtntte	Round to nearest, ties to even	→ Example 22.5	178

Linear Algebra

1
Tr
Det
<i>U</i>

V
 W
 U_1
 U_2
 U_3
 V_1
 V_2
 V_3
 W_1
 W_2
 W_3

Relations

R, S	Generic relation names.
R^\top	Transpose of a relation R .
A^\top	Transpose of a relation R .

Posets

Generic poset names

P, Q, R	Generic posets	\rightarrow Def. 20.2	159
P, Q, R	Underlying set for the posets	\rightarrow Def. 20.2	159
p, q, r	Generic elements of posets		
x			
y			
z			
A			
B			
C			
D			
a			
b			
c			
d			
A			
B			
C			
D			
a			
b			
c			
d			
A			
B			
C			
D			
X			
Y			
\leq_p			
$\mathcal{U}P$			
$\mathcal{L}P$			

Int

<i>Operations on sets</i>			
$\text{Min}_{\leq_P} S$	Minimal elements of the subset S .	$\rightarrow \text{Def. 20.29}$	167
$\text{Max}_{\leq_P} S$	Maximal elements of the subset S .	$\rightarrow \text{Def. 20.30}$	167
Inf			
Sup			
L			
U			
\uparrow_S	Upper closure of S .	$\rightarrow \text{Def. 20.21}$	165
\downarrow_S	Lower closure of S .	$\rightarrow \text{Def. 20.26}$	166
<i>Operations on elements</i>			
$x \vee y$	Join of two elements x, y	$\rightarrow \text{Def. 20.35}$	169
$x \wedge y$	Meet of two elements x, y	$\rightarrow \text{Def. 20.35}$	169
<i>Constructors</i>			
$\mathcal{A}\mathbf{P}$	Antichains of \mathbf{P} .	$\rightarrow \text{Def. 20.14}$	163
$\mathcal{L}\mathbf{P}$	Lower sets of \mathbf{P} .	$\rightarrow \text{Def. 20.20}$	164
$\mathcal{U}\mathbf{P}$	Upper sets of \mathbf{P} .	$\rightarrow \text{Def. 20.19}$	164
$\mathcal{W}\mathbf{P}$	Downward-closed upper sets of \mathbf{P} .	$\rightarrow \text{Def. 20.32}$	168
<i>Symbols</i>			
\leq_P	Order relation associated to the poset \mathbf{P}		
$\top_{\mathbf{P}}$	Top of poset \mathbf{P}	$\rightarrow \text{Remark 20.36}$	169
$\perp_{\mathbf{P}}$	Bottom of poset \mathbf{P}	$\rightarrow \text{Remark 20.36}$	169
<i>Attributes</i>			
$\text{width}(\mathbf{P})$	Width of the poset \mathbf{P} .	$\rightarrow \text{Def. 20.33}$	168
$\text{height}(\mathbf{P})$	Height of the poset \mathbf{P} .	$\rightarrow \text{Def. 20.34}$	168
<i>Domain theory</i>			
lfp	Least fixed point		
CPO	Complete partial order	$\rightarrow \text{Def. 45.15}$	337
DCPO	Directed-complete partial order	$\rightarrow \text{Def. 45.15}$	337

Categories

<i>Basic</i>			
$b \circ a$	“ b after a ”		
$a ; b$	“ a then b ”		
$f ; g$	Composition of morphisms		
$F ; G$	Composition of functors		
$\alpha ; \beta$	Composition of natural transformations		
$\text{Ob}_{\mathbf{A}}$	Objects of the category \mathbf{A} .	$\rightarrow \text{Definition 10.2}$	99
Id_X	Identity morphism for the object X	$\rightarrow \text{Definition 10.2}$	99
$\text{Hom}_{\mathbf{A}}(X; Y)$	Hom-set between X and Y .	$\rightarrow \text{Definition 10.2}$	99
<i>Generic names</i>			
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D} \dots$	Symbols for categories		
\mathbf{V}	Symbol for enrichment categories.		
X			
X, Y, Z, W	generic objects		
Y			
Z			
\mathcal{O}			

\mathcal{P}				
f, g, h, i	Generic morphisms		→Definition 10.2	99
Π_f				
Π_r				
\swarrow				
\searrow				
F_{mor}				
F_{ob}				
G_{mor}				
G_{ob}				
F, G, H, I	Generic functors		→Definition 29.4	218
I				
F_1			→Definition 29.4	218
F_2			→Definition 29.4	218
F_3			→Definition 29.4	218
Id_A	Identity functor for category A			
Id_F	Identity natural transformation for functor F			
$\alpha, \beta, \gamma, \delta$	Generic natural transformations		→Definition 32.1	234
	<i>Monads</i>			
M, N	Generic monads.		→Definition 46.2	361
un	Monad unit		→Definition 46.2	361
mu	Monad identity		→Definition 46.2	361
L	lower-set endofunctor			
\mathcal{L}	lower-set monad			
U	upper-set endofunctor		→Def. 47.10	373
\mathcal{U}	upper-set monad		→Def. 47.14	375
	<i>Companion/conjoints</i>			
colim				
Col				
	<i>Operations</i>			
\times	Product in a category		→Definition 24.8	190
$+$	Co-product in a category		→Definition 24.29	197
	<i>Constructors</i>			
$\text{Tw}(A)$	Twisted arrow construction on category A .		→Definition 27.3	211
Arr	Arrow category			
op				
between				
	<i>Semigroups</i>			
S	Generic semigroup names			
S, T, U	Generic semigroup names.			
T	Generic semigroup			
U	Generic semigroup			
$\textcolor{brown}{T}$	Generic semigroup			
$\textcolor{brown}{U}$	Generic semigroup			
\log				
x	Generic semigroup element			
y	Generic semigroup element			
z	Generic semigroup element			

s	Generic semigroup elements.		
F, G	Generic semigroup morphisms.		
<i>Monoids</i>			
id	identity for monoid	\rightarrow Definition 6.12	69
M, N	Generic monoid names	\rightarrow Definition 6.12	69
M, N	underlying set	\rightarrow Definition 6.12	69
x, y, z	Generic monoid elements	\rightarrow Definition 6.12	69
y	Generic monoid element	\rightarrow Definition 6.12	69
z	Generic monoid element	\rightarrow Definition 6.12	69
m, n	Generic monoid elements	\rightarrow Definition 6.12	69
\circ	Monoid operation	\rightarrow Definition 6.12	69
<i>Groups</i>			
id	identity for group	\rightarrow Definition 6.23	73
G, H	Generic group names	\rightarrow Definition 6.23	73
G, H	underlying set	\rightarrow Definition 6.12	69
x	Generic group element		
y	Generic group element		
z	Generic group element		
g, n	Generic group elements	\rightarrow Definition 6.12	69
\circ	Group operation	\rightarrow Definition 6.23	73
<i>Monoidal categories</i>			
\otimes_A	Monoidal operation for category A .	\rightarrow Definition 39.6	293
1	Identity object for monoidal operation	\rightarrow Definition 39.6	293
inv	group inverse		
\det	group inverse		
lu	Left unit	\rightarrow Definition 39.6	293
ru	Right unit	\rightarrow Definition 39.6	293
as	Associator	\rightarrow Definition 39.6	293
br	Braiding	\rightarrow Definition 39.6	293
μ	Isomorphism for strong monoidal functor	\rightarrow Definition 39.20	302
iso	Isomorphism for strong monoidal functor	\rightarrow Definition 39.20	302
ϵ	evaluation map for dualizable objects		
η	coevaluation map for dualizable objects		
<i>Adjunctions</i>			
L	left adjunct functor	\rightarrow Definition 33.7	241
R	right adjunct functor	\rightarrow Definition 33.7	241
$L \dashv R$	L and R are adjoint functors.	\rightarrow Definition 33.7	241
τ			
un	Unit		
co	Co-unit	\rightarrow Definition 33.9	242
<i>Traced monoidal categories</i>			
Tr	Trace operator	\rightarrow Definition 40.3	308
Conw	Conway operator		
par			
Unc			
\mathcal{U}			
<i>Named categories</i>			
\mathbf{DP}	Category of design problems	\rightarrow Def. 37.15	285

UDP			
DPI			
Draw	Category of drawings	→Def. 30.4	224
Bool	Booleans (: see how?)		
Cat	Category of small categories	→Definition 31.1	230
Vect	Category of vector spaces	→Example 32.3	235
FinVect	Category of finite-dimensional vector spaces	→??	??
Rel	Category of sets and relations	→Definition 15.4	123
FinSet	Category of finite sets and functions	→Example 30.2	223
Set	Category of sets and functions	→Definition 16.5	135
Prof			
Pos	Category of posets and monotone maps	→Definition 22.10	180
UPos			
LPos			
InjSet	Category of sets and injective functions	→Example 30.6	225
Grph		→Def. 18.1	147
<i>g</i>			
<i>src</i>			
<i>tgt</i>			
V			
<i>v</i>			
<i>w</i>			
A			
<i>a</i>			
<i>a</i>			
<i>b</i>			
LTI			
Free	free construction		
Feas			
<hr/>			
stuff missing			
<i>ready</i>			
<i>draft</i>			
<i>missing</i>			
<hr/>			
Misc			
<i>P</i>			
<i>Q</i>			
<i>R</i>			
<hr/>			
Tuples			
<i>⟨⟩</i>	zero-size tuple		
<hr/>			
Booleans			
<i>⊤</i>			
<i>⊥</i>			
<i>∧</i>			
<i>∨</i>			
<hr/>			
Arrows			
<i>→</i>	Set arrow		
<i>↗</i>	Morphism arrow		
<i>⤔</i>	Functors arrow		

\rightarrow	Functors arrow, longer		
\Rightarrow	Natural transformation arrow		
\Rrightarrow	Natural transformation arrow, longer		
\Lleftarrow	Inverted natural transformation arrow		
\cong			
\approx			
\cong			
\approx			
\rightarrowtail	profunctor arrow	\rightarrow Definition 38.2	288
\rightarrow_{Pos}			
\rightarrow_{Cat}			
\sim			
\rightarrow			
\Rightarrow	Implies		
$\mathbf{A} \hookrightarrow \mathbf{B}$	\mathbf{A} embeds in \mathbf{B} .		
DP			
	<i>Formalization</i>		
f	A generic functionality in \mathbf{F} .	\rightarrow Def. 36.1	257
r	A generic cost in \mathbf{R} .	\rightarrow Def. 36.1	257
i	A generic implementation in in \mathbf{I} .	\rightarrow Def. 36.1	257
\mathbf{F}	Functionality space		
\mathbf{A}	Functionality space		
\mathbf{B}	Functionality space		
\mathbf{C}	Functionality space		
\mathbf{A}			
\mathbf{B}		\rightarrow Def. 36.1	257
\mathbf{R}	Requirements space		
\mathbf{A}	Requirements space		
\mathbf{B}	Requirements space		
\mathbf{C}	Requirements space		
\mathbf{D}	Requirements space	\rightarrow Def. 36.1	257
\mathbf{I}	Implementation space	\rightarrow Def. 36.1	257
$\text{prov} : \mathbf{I} \rightarrow \mathbf{F}$	functionality of an implementation	\rightarrow Def. 36.1	257
$\text{req} : \mathbf{I} \rightarrow \mathbf{R}$	requirements of an implementation	\rightarrow Def. 36.1	257
	<i>Computational representation</i>		
φ		\rightarrow ??	??
h		\rightarrow Def. 45.2	331
h_L			
h_U			
H			
	<i>DP</i>		
\mathbf{d}	generic design problem as a profunctor		
\mathbf{f}	generic design problem as a profunctor		
\mathbf{g}	generic design problem as a profunctor		
$(\mathbf{f} ; \mathbf{g})$			
\mathbf{h}			
\mathbf{dp}			
series			
par			
loop			
loopb			

Terms

Φ

φ

\mathcal{A}

\mathbf{T}

v

ops

\times

\dagger

\odot

\otimes

\oslash

\sqcup

UId

vdc

Queries in DP

Feasibility

→Section 36.2 267

FeasibleImp

→Section 36.2 267

FixFunMinReq

→Section 36.2 266

FixResMinFun

→Section 36.2 266

Original paper

\mathcal{V}

v

v_1

v_2

i

j

i_1

j_2

n_f

n_r

$n_{f,v}$

$n_{r,v}$

\mathcal{E}

UF XXX: not a good choice

UR

Uncertainty paper

floor

ceil

u_a

u_b

L

U

UDP

\preceq_{UDP}

DP

Currencies

USD

USD

SGD

SGD

CHF

CHF

EUR

EUR

฿

Generic currency

Symbols used in particular chapters

?? 25

ε

F

Δx

k

E

?? 10

Curr

Currency category

→Def. 12.2

109

?? 14

Berg

The category of Swiss mountains

→Def. 14.2

120

BergAma

→Section 30.4

226

BergLazy

→Section 30.4

226

Intermodal

Car

Flight

Board

FCO

ZRH

↗

↗ **Alitalia N**

↗ **Alitalia S**

↗ **Ewings N**

?? 18

Database

?? 28

Plans

→Example 29.9

220

Epluribus

⌚

~~

PropertyParams

•

•

⌚

~~

PropertyParams

sprout

young

dead

mature

old

Morse code

•	Morse dot
—	Morse dash
s_1	Silence between dots and dashes
s_3	Silence between letters
s_7	Silence between words
■	Beep of ℓ
■■	Beep of 3ℓ
■■■	Silence of ℓ
■■■■	Silence of 3ℓ
■■■■■	Silence of 7ℓ

ASCII example

A
ASCII
ASCII

Processes

A
U
Y
X
dyn
ro
End
Aut

To categorize

Constituents

Conditions

1 not sure if category or set
:= “defined as”
Lax
Triv
Vert
d

Deprecated

id

Frequently misspelled words

morphism
morphisms
Morphism
Morphisms
morse
M
morse

Matrix groups

O(n) Orthogonal group
SO(n) Special orthogonal group

$\mathrm{GL}(n)$	General linear group
$\mathrm{SL}(n)$	Special linear group
$\mathrm{E}(n)$	Euclidean group
$\mathrm{SE}(n)$	Special euclidean group
$\mathrm{SO}(2)$	
$\mathrm{SO}(3)$	
$\mathrm{SE}(2)$	
$\mathrm{SE}(3)$	
act	
Covact	
Contravact	
apply	
start	
\circ	
:	
Moo	
Mor	
VF	
EB	
DS	
φ	
γ	
<i>fish</i>	
<i>lift</i>	
<i>join</i>	
<i>bind</i>	
<i>fmap</i>	
<i>return</i>	
a	
α	
β	
C	
C^*	
D	
D^*	
E	
$(F; G)$	
$(F; G)_{\mathrm{ob}}$	
$(F; G)_{\mathrm{mor}}$	
s	
t	
Tup	

Tables and maps

Bibliography

- [1] Erik K Antonsson and Jonathan Cagan. *Formal engineering design synthesis*. Cambridge University Press, 2005.
- [2] 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).
- [3] Andrea Censi. «Handling Uncertainty in Monotone Co-Design Problems». In: *cs.RO* abs/1609.03103 (2016). URL: <https://arxiv.org/abs/1609.03103>.
- [4] Edgar F Codd. «A relational model of data for large shared data banks». In: *Software pioneers*. Springer, 2002, pp. 263–294.
- [5] 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.csl.2010.09.001](https://doi.org/10.1016/j.csl.2010.09.001).
- [6] 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.
- [7] 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).
- [8] P. Cuffe and A. Keane. «Visualizing the Electrical Structure of Power Systems». In: *IEEE Systems Journal* 11.3 (2017), pp. 1810–1821. DOI: [10.1109/JST.2015.2427994](https://doi.org/10.1109/JST.2015.2427994).
- [9] 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).
- [10] 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).
- [11] Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press, 2019.
- [12] G. Gierz et al. *Continuous Lattices and Domains*. Cambridge University Press, 2003. DOI: [10.1017/cbo9780511542725](https://doi.org/10.1017/cbo9780511542725).
- [13] 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.
- [14] 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.

- [15] 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).
- [16] 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.
- [17] 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).
- [18] 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.
- [19] 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).
- [20] 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.
- [21] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 2013.
- [22] 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>.
- [23] 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).
- [24] 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.
- [25] 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.
- [26] Gerhard Pahl et al. *Engineering Design: A Systematic Approach*. 3rd. Springer, Jan. 2007. ISBN: 1846283183.
- [27] 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.
- [28] 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).
- [29] Herbert A. Simon. *The Sciences of the Artificial* (3rd Ed.) Cambridge, MA, USA: MIT Press, 1996. ISBN: 0262691914.
- [30] 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>.
- [31] David I Spivak. «Categorical databases». In: *Presented at Kensho* (2019).

- [32] David I Spivak and Robert E Kent. «Ologs: a categorical framework for knowledge representation». In: *PloS one* 7.1 (2012), e24274.
- [33] 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).
- [34] N.P. Suh. *Axiomatic Design: Advances and Applications*. Oxford University Press, 2001. ISBN: 9780195134667.
- [35] Maria Svorenova et al. *Resource-Performance Trade-off Analysis for Mobile Robots*. 2016. eprint: [arXiv: 1609.04888](https://arxiv.org/abs/1609.04888).
- [36] 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).
- [37] 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**, 162
- Cat**, 233
- Draw**, 224
- DP**, 285
- FinSet**, 223
- InjSet**, 225
- Pos**, 180
- Rel**, 123
- Set**, 135
- EnumerableSet**, 5
- FiniteMapRepresentation**, 56
- FiniteMap**, 56
- FiniteMonoidRepresentation**, 72
- FiniteMonoid**, 71
- FiniteSemigroupConstruct**, 67
- FiniteSemigroupRepresentation**, 68
- FiniteSemigroup**, 67
- FiniteSetProperties**, 55
- FiniteSetRepresentation**, 6
- FiniteSet**, 6
- FreeSemigroup**, 67
- MakeSetProduct**, 57
- MakeSetUnion**, 58
- Mapping**, 55
- Monoid**, 71
- Semigroup**, 67
- SetProduct**, 56
- SetUnion**, 58
- Setoid**, 4
- adjunction, 241, 242
- antitone Galois connection, 238
- antitone map, 177
- binary relation, 123
- Braided monoidal category, 297
- cartesian product, 187
- category, 99
- category of design problems, 285
- coproduct, 197
- design problem, 278
- design problem with implementation, 257
- endofunctor, 219
- endorelation, 127
- Enriched functor, 384
- equivalence relation, 127
- function, 133
- Functor, 218
- graph, 147
- group, 73
- Hasse diagram, 159
- identity design problem, 285
- initial object, 207
- Injective function, 225
- inverse, 204
- isomorphism, 204
- magma, 61
- monoid, 69
- Monoidal category, 293
- Monoidal poset, 291
- monotone Galois connection, 237
- monotone map, 177
- natural isomorphism, 234
- natural transformation, 234
- operad, 387
- opposite category, 209
- partition, 128
- permutations, 206
- poset, 159
- product, 190
- profunctor, 288
- relation, 133
- Semi-functor, 218

semi-functor, 218
semicategory, 98
semigroup, 62
span, 319
Strong monoidal functor, 302

subcategory, 223
Symmetric monoidal category, 298
Symmetric monoidal poset, 292
terminal object, 207



Example exams

This chapter contains some examples of the style of the ETH class exam.

Both sample exams are thought to last 90 minutes and to be open book (all notes allowed).

Example 1: Uncertain Machines

Consider the category of Moore machines acting on signals as described in the course. Assuming the input/output sets are also ordered sets (posets), construct the Kleisli category corresponding to the interval monad \mathcal{U} . That is, the signals are closed intervals of values. Call this category **UMoore**.

1. (30%) Is **UMoore** a monoidal category?
 2. (20%) Is **UMoore** a traced monoidal category?
 3. (50%) Would the answers be the same if asked about the More category?
-

Example 2: Machines with resources consumption

Consider the category of Moore machines. We want to be more precise about resource consumption and want to define an extension of Moore machines in which each machine also has associated a certain time $T_1 \geq 0$ to run the dynamic function `dyn` and a certain time $T_2 \geq 0$ for running the readout function `ro`. We call these resource-Moore machines (**RMoore**).

1. (30%) Formalize the **RMoore** category giving formulas for identities, compositions, and proof of associativity.
2. (70%) Suppose that you want to design the software for a robot. You are given the wiring diagram of the architecture, in which you have to plug in specific **RMoore** machines to implement the algorithmic functionality. Assume that the wiring diagram does not contain any loop - only series and parallel composition, and that there is only one input (robot observations) and one output (robot commands). For each hole in the diagram, you are given a set of 1 or more **RMoore** machines that can implement the functionality. Assume that the computer on which to run everything has $N \geq 1$ processors. Think of each processor as a “lane” in which the operations of each machine are cars that must run sequentially. Formalize the design problem in the category **DPI** that corresponds to choosing the best combination of machines and the best assignment to processors. Include as a resource the number of processors and as functionality the throughput of the system (how many commands are generated per second).