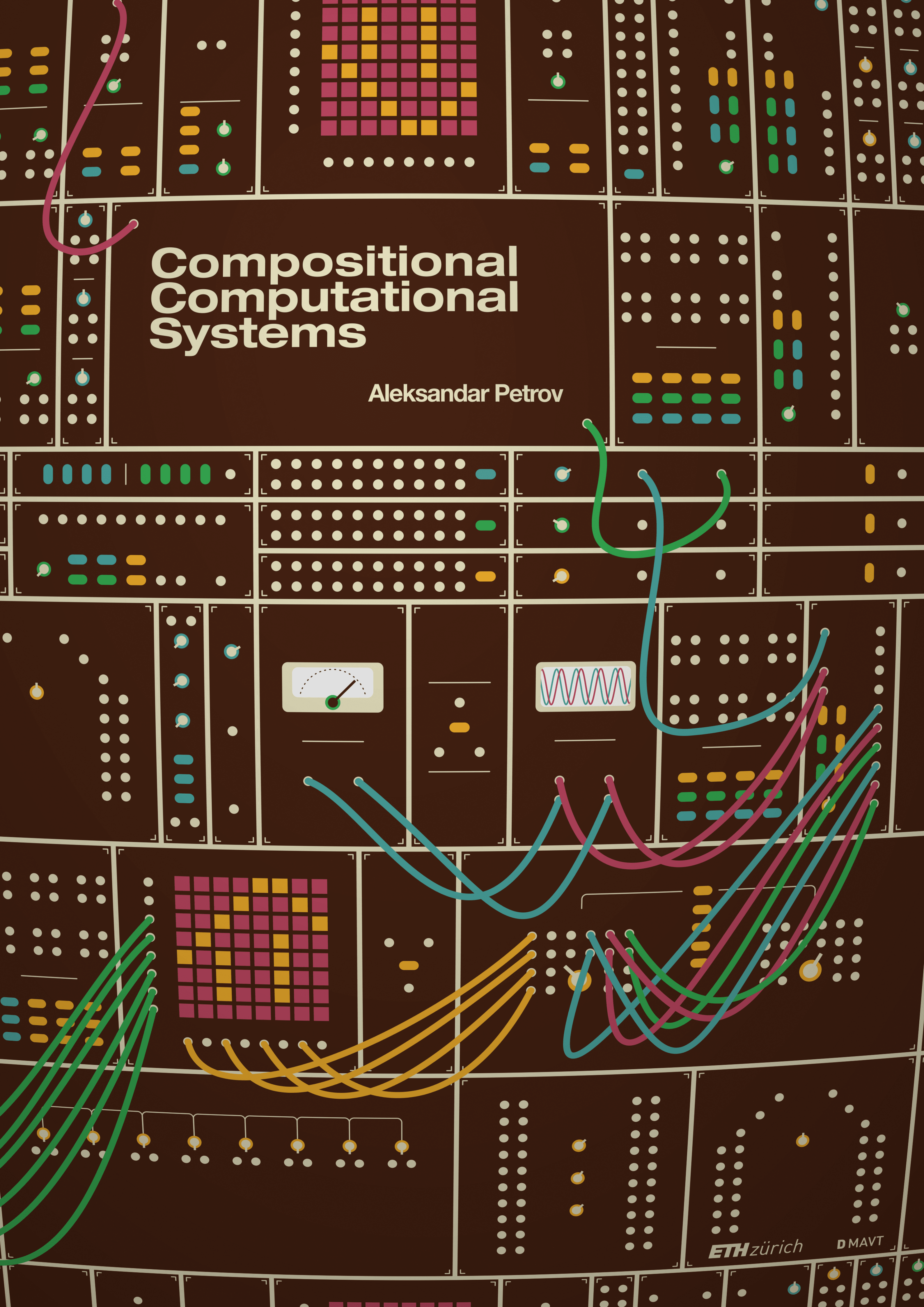
Compositional Computational Systems

Compositional Computational Systems

Aleksandar Petrov



ETH zürich D MAVT

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology Zürich

Master's Thesis

Compositional Computational Systems

Aleksandar Petrov

Supervision
Gioele Zardini
Dr. Andrea Censi
Prof. Dr. Emilio Frazzoli

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

October 2020
Zürich

Contents

Chapter 1

Introduction

If we are to summarize the topic of this thesis succinctly, perhaps the best way would be to say that it deals with problems, their solutions, and the relationships between the two. We are surrounded by countless problems in our everyday lives, many of which we can't escape from solving. Hence, we all frequently find ourselves face to face with the task of solving problems. However, given its importance, how many of us have stopped to think about or study the process of problem-solving? This thesis is an exercise in investigating exactly this question: what is problem-solving and how to do it in the best possible way.

We are surrounded by thousands of problems and very few solutions. But why is problem-solving so difficult? Well, for once, problems are often not specified well-enough. Making a "fair" legal system or a "safe" autonomous car are examples of problems that cannot be solved until one clarifies what "fair" and "safe" even mean. But even if a problem is unambiguously defined, and is solvable, it does not mean that we are done. Often we have to pick one among a collection of different solutions, many of which are not dominated by another. This is usually a big pain when planning a trip or a holiday. There might be dozens of different ways to get to your destination, each with different costs, taking different time, with different modes of transportation, and with many different characteristics. Picking the "right" option can be very stressful and often takes quite a toll on the holiday-planner. But there's some light in the tunnel. Even the hardest problems become much easier if we find a way to reduce them to problems that we already know how to solve. And we use this technique a lot! We can reduce the problem of filling in our tax return to the problem of hiring an accountant, which can be much easier.

Or we can reduce the problem of evaluating a neural network to the problem of matrix multiplication, which we know how to do very efficiently. Hence, this philosophy of problem reduction indeed seems to be quite handy for any problem-solver.

Often when one struggles with solving or understanding something, the most important question to ask is if they are solving the *right* problem. Or it could be that the problem is ambiguously defined or defined in inconsistent ways. Sometimes, especially when multiple people are involved, this can also be a result of some sort of miscommunication. It often happens that one party has one thing in mind, but the other interprets it completely differently. While neither side is really at fault here, there are strategies to help in this kind of situation.

This is where formal systems come into play. Using formalisms allow us to define and handle various objects and concepts unambiguously and consistently. Hence, we can also communicate while being sure that both parties interpret the problem in the same way: the only way. It is not surprising that mathematicians are perhaps the most agreeable trope of the science and engineering world: they tend to define their problems and objectives in formal and unambiguous ways which ensures that everyone is on the same page. Of course, formalism won't protect us from defining the *wrong problem*, but that's a point for a whole different discussion.

We started by saying that we stumble upon many problems in our daily lives. And we mentioned that problems can be badly defined, might have many different possible solutions, and can be reducible to other problems. The main philosophy behind this thesis is that these three characteristic properties of problems can be used to formalize problem-solving. Or, we could also say that formalizing problem-solving implies that the resulting problems would be well-specified, supporting multiple solutions, and reducible to one another. Whether it is these characteristic properties or the formalization that gives rise to the other, and if any at all, is one of the questions that we will *not* answer. But one can imagine that we might be also observing a self-referential strange loop biting its own tail. However, regardless of which comes first, and if any, it is this intimate relationship that we are ultimately curious about.

The current work, despite conceived with a very different mission, is, in fact, a study on the meta-problem of problem-solving. "Meta" because in some way we tried to study the common nature of all problems. But this is also a beautifully self-referential problem because the problem of problem-solving is itself a problem and hence should be subjected to the very conclusions it tries to reach. While this perhaps sounds like some mumbo-jumbo right now, we will show it formally later in the text.

Putting this philosophical pondering aside, the official objective of this work is to formalize problem-solving so that we can reason about it in a systematic, and automated way. We want to work in a formal setting because, as we mentioned above, that is a way to make sure we agree what problem we are interested in and what we mean by "solving" it. We are interested in systematic reasoning because this would allow us to reason fast and efficiently. And as a nice bonus of a process being systematic is that it might be well-suited for automation with a computer or via some other mechanical means.

We tried to keep this work as general and widely applicable as possible. However, the task of solving the problem of defining formalisms for defining problems and for evaluating their solutions is in itself not well-defined. No one provided us with a formal definition of a "problem" or a "solution" on the outset. Hence, the definitions and the resulting theory that you will see in this work are deeply ingrained with our assumptions of what problems and solutions are and with our beliefs of what problem-solving should look like. Hence, it is only fair that we make these biases explicitly known to the reader.

First, we are deeply convinced that problem-solving has an intrinsically compositional nature. Problems rarely appear in a complete vacuum. A single problem can often act as a component of more complex problems and can be itself composed of other problems. For example, the problem of making a salad has as subproblems washing the vegetables and cutting them. The very same problem can also be a component of the problem of preparing a three-course meal for a family. When we use the word "component", we can mean many different things. For example, we can use the answer to one problem as the statement of the next one. The washed vegetables can act as the starting point of their cutting. We can even use the answer of a problem as its own statement, or part of it, creating a loop in the process. Or, we might be interested in finding a steady solution to this loop. Or, we might be solving two problems simultaneously. Or, we might be handling multiple problems and their solutions and then comparing them in various ways. All these different possible operations we will refer collectively to as "compositions".

Everyone who was involved in this work is an engineer. And it is commonly believed that engineers are supposed to be lazy. Though in practice they tend to be unreasonably zealous at making the contraptions that allow them to be lazy. This work is a very good example of such a device. Hence, due to our engineering laziness, we conceived this thesis hoping that it will help us outsource the actual problem solving, or even the solving of problem-solving itself to a computer, a robot, or another computational device. Precisely because we want to use computing devices we talk of procedures for solving problems which are backed by concrete sequences of steps, algorithms, some computer code, or even actual executables.

Finally, through the course of the study of the various ways one can define problems and solutions, we reached the very liberal conclusion that there is no single correct way to do it. One might want to include the computational resources needed for solving a problem within the particular formalism. Or might be interested in the rich ways in which problems interact. Or in a more probabilistic setting. Therefore, instead of simply saying, “well, it depends”, we chose to take a look at the common properties of any such formalism and of how they relate to one another.

This also explains the title of this thesis: Compositional Computational Systems. “Compositional” because problems compose to create other problems. “Computational” because we want a computer to do the job instead of us. And “Systems”, in the plural, because one can construct a family of such systems with some common properties and structure while keeping a very rich variety of the concrete implementation and applications.

This thesis is divided into two parts. Part I introduces the theory of compositional computational systems. Then we use Part II to discuss the mathematical theory of co-design and the place it has in the compositional computational systems framework. The particular choice of co-design is both because of our personal interests and because it a structurally very rich theory and hence serves as a good extended example. Nevertheless, Part II is not required to understand the compositional computational systems aspect of this work.

We start in Chapter 2 by reviewing some of the most fundamental concepts of the branches of mathematics that will be used throughout this work. We review some basic notions from order theory such as posets and feasibility relations. Then, the focus moves to category theory and the definitions of categories, functors, natural transformations, and their various variations. Following that, we introduce notation for defining anonymous functions which will make some of the proofs much easier to read and understand. This is a non-standard notation, so we recommend that even the knowledgeable reader takes a quick look at it. Finally, we review some concepts from type theory which will be useful for defining normed types, as well as problems and procedures in Chapter 3 and Chapter 4.

Chapters 3 and 4 deal with a specific instance of a compositional computational system that we call **Lagado**. We start with a concrete system as it can be quite helpful for understanding the basic principles and philosophy of representing problems, procedures, and solutions. Then, Chapters 5 and 6 generalize these ideas to an abstract setting where one can make various choices for defining problems, procedures, and solutions, each one of which giving rise to a different compositional computational system.

More precisely, in Chapter 3 we introduce the concept of normed types. A normed type extends the notion of a type with a set of sizes and a function that assigns a size to each term of the type. This will come especially handy when we later introduce procedures which are computational processes consuming some resources to perform the computation, where the amount of the required resources depends on the size of the concrete normed term.

Then, in Chapter 4 we provide our first formal definition of a problem and a solution. We will define problems as binary relations between two normed types, one dubbed a *statement type*, and another called *answer type*. Every instance of a problem identifies with a particular statement, while the answer would be what we want to receive when the instance is solved. We allow for this relationship to be a binary relation because then we can model both problems with multiple correct answers per statement and problems that might have no correct answers for some statements. The collection of problems is then shown to give rise to a categorical structure in the form of the **Prob** category. Once we have cleared up a definition for what problems are, we move to solutions. A solution to a given problem will be simply defined as a procedure between the same normed types as the ones of the problem, with the procedure returning a correct answer for any possible statement of the given problem. Finally, we show how both problems and solutions can be represented in a category, which we call **Lagado**, and how **Lagado** can be considered to have a structure similar to a twisted category. To the best of our knowledge, everything that is introduced in Chapters 3 and 4, is a novel contribution of this thesis.

The reader will likely notice that Chapters 3 and 4 are build up with the desire to put everything on a category-theoretical foundation. This might seem arbitrary but it is not an end in itself. The first half of Chapter 5 (Sections 5.1 to 5.4) is an apology of this choice. There, we start by defining a few basic assumptions that intuitively carry the meaning of "compositionality" and we show that these assumptions result in a semicategorical structure. While the presentation of this argument is original, the basic ideas go back to the seminal paper on category theory by Eilenberg and MacLane (1945). We do though contribute a new way of relating the morphisms of two (semi)categories which we call *kinded functions*. We then go on to show how these kinded functions generalize many other constructions that bridge two containers of mathematical objects, i.e. categories, sets, groups. Examples of such kinded functions are procedures (as defined in Chapter 4), binary relations (so problems), functions, functors, bifunctors, monoidal functors, etc.

The fact that Chapter 5 shows that a semicategory meets our assumptions for compositionality and that kinded functions are a good abstraction of various relations between such semicategories is the reason why we base our definition of a compositional computational system, the topic of Chapter 6, on them. In Chapter 6, we provide new and more general definitions for problems and solutions: kinded functions between two semicategories. As long as the kinds of these kinded functions are rigs, one can construct a system (called **Laputa**), with a structure very similar to the one of **Lagado**, and that contains the new generalized definitions of problems and solutions. Depending on the particular choice of objects, rigs for the kinds of the problems, and the solutions, as well as the conditions which make a procedure a solution for a given problem, we can define various such compositional computational systems, each one with slightly different structure and properties. To illustrate this, we provide examples for a system with functorial problems and procedures, and one with probabilistic problems and procedures. Chapter 6 generalizes the theory that we earlier developed in Chapters 3 and 4), and hence consists of novel results. This would mark the end of Part I and would conclude the development of the theory of compositional computational systems.

Then, in Part II we take the theory of co-design, first developed by Censi (2016), and we show how it fits in the family of compositional computational systems. We start this extended example in Chapter 7 by providing a few more definitions of order theory and introducing various types of design problems: the building blocks of co-design. Then, Chapter 8 focuses on the computational aspect of co-design, i.e. how to compute the solutions of co-design problems. The approach to solving these problems as well as the procedures themselves are closely following the ones introduced by Censi (2016). However, these solutions rely on a duality between upper sets of posets and the antichains of their minimal elements. To the best of the authors' knowledge, a characterization of the conditions under which such duality indeed holds has not been performed yet, so we propose one in Chapter 8. When combined with the necessary conditions for solving loop problems via least fixpoints (again a method due Censi (2016)), we obtain a concise definition of well-behaved design problems, which are the ones for which the computational techniques of co-design are indeed guaranteed to work. This too is a novel contribution.

Chapter 9 then focuses on categorical structures and connections in co-design that have not been explored before. While we base the study of these structures on the well-known properties of the category of design problems which were first studied by Censi et al. (2020), our main contribution is extending them to the categories of answers of design problems and showing that the functors which we use to relate the various problem formulations and their answers preserve their rich structure. Concretely, we show that these relationships are monoidal functors that preserve traces and the locally-posetal structure of hom-sets. Unless whenever otherwise mentioned, this chapter too contains original material.

Finally, Chapter 10 puts the two parts of the thesis together. It shows how co-design fits in the bigger framework of compositional computational theories. We first show that every design problem is a problem in at least one compositional computational system. Then, we move up a level of abstraction and we show that the whole collection of design problems (the category of design problems) can itself be considered as problem statements in a different compositional computational system. Then we show how the functors developed in Chapter 9 act as problems and procedures in this new system. As these functors preserve tensor products, traces, joins, meets, etc, we provide a yet another compositional computational system that possesses such structure. Finally, the focus is back on the computational question and we show how to add resource-awareness to the procedures in this new system, similarly to how it was earlier done for **Lagado**.

The process of exploring and studying the concepts and ideas that eventually made the body of this thesis was extremely rewarding. It allowed us to find some fundamental and profound properties and connections in the meta-problem of problem-solving. We are confident that this experience not only produced this work but will also inform our future problem-solving endeavours. We hope that the reader can also experience at least part of our excitement and that they will find as much use of these results and ideas as we did.

Part I

Compositional problem-solving

Chapter 2

Mathematical preliminaries

We start this thesis by reviewing some of the mathematical concepts that will be frequently used throughout this document. Contrary to how one should start any piece of writing, we will keep this chapter formal and rather dry. The subjects covered are simply so rich in depth and scope that nothing we can introduce here would give them justice. Nevertheless, we will provide pointers to literature that discusses these topics in sufficient depth.

2.1 Posets

Posets, short for *partially ordered sets*, are one of the building blocks of order theory and we will be frequently using them throughout this text. The interested reader can find a more thorough treatment by Davey and Priestley (2002), while Fong and Spivak (2019) give a broader study of the relations of posets with some of the other concepts used here.

Definition 2.1 (Binary operation)**.** A binary operation on a set N is a function

$$N \times N \to N.$$

Definition 2.2 (Poset)**.** A *poset* (partially ordered set) is a tuple $(P, \leq_P)$ where P is a set and $\leq_P$ is a binary relation on P which for all $a, b, c \in P$ has the following properties:

i. *Reflexivity:* $a \leq_P a$;
ii. *Transitivity:* if $a \leq_P b$ and $b \leq_P c$, then $a \leq_P c$;
iii. *Antisymmetry:* if $a \leq_P b$ and $b \leq_P a$, then $a = b$.

In words we will refer to $a \leq b$ as *'a precedes b'* or *'b reduces to a'*. We might sometimes omit the subscript of the relation when it is clear from the context (as in the previous sentence). We will often denote by P both the poset and its set of elements. It should be clear from the context which one we are referring to.

Definition 2.3 (`Bool` poset)**.** The poset `Bool` is defined as $(\{\mathtt{T}, \mathtt{F}\}, \leq)$ with only one non-reflexive relation: $\mathtt{F} \leq \mathtt{T}$.

Definition 2.4 (Identity poset)**.** We define a special poset I with a single element ι and with a single relation $\iota \leq \iota$.

Definition 2.5 (Cartesian product)**.** For two sets A and B, their Cartesian product, denoted $A \times B$, is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$, that is

$$A \times B := \{(a, b) \mid a \in A \wedge b \in B\}.$$

Definition 2.6 (Product poset)**.** Given posets $\langle P, \leq_P \rangle$ and $\langle Q, \leq_Q \rangle$ we can define the *product poset* $\langle P \times Q, \leq \rangle$ on the Cartesian product $P \times Q$ by requiring $\langle p, q \rangle \leq \langle p', q' \rangle$ if and only if (iff) $p \leq_P p'$ and $q \leq_Q q'$.

Definition 2.7 (Opposite poset)**.** Given a poset $P = \langle P, \leq \rangle$ we define the *opposite poset* $P^{\mathrm{op}} = \langle P, \leq_{\mathrm{op}} \rangle$ as the poset which has the same elements but with $p \leq_{\mathrm{op}} q$ iff $q \leq p$.

Definition 2.8 (Monoidal poset)**.** A *monoidal structure* on a poset $\langle P, \leq \rangle$ consists of:

(i) An element $I \in P$, called *monoidal unit*, and
(ii) a function $\otimes \colon P \times P \to P$, called the *monoidal product*. Note that we write

$$\otimes(p_1, p_2) = p_1 \otimes p_2, \; p_1, p_2 \in P.$$

The constituents must satisfy the following properties:

(a) *Monotonicity*: For all $p_1, p_2, q_1, q_2 \in P$, if $p_1 \leq q_1$ and $p_2 \leq q_2$, then $p_1 \otimes p_2 \leq q_1 \otimes q_2$.
(b) *Unitality*: For all $p \in P$, $I \otimes p = p$ and $p \otimes I = p$.
(c) *Associativity*: For all $p, q, r \in P$, $(p \otimes q) \otimes r = p \otimes (q \otimes r)$.

A poset equipped with a monoidal structure $\langle P, \leq, I, \otimes \rangle$ is called a *monoidal poset*.

Remark 2.9. If the monoidal poset further satisfies that for any $p_1, p_2 \in P$ we have $p_1 \otimes p_2 = p_2 \otimes p_1$, then we call it a *symmetric monoidal poset*.

Lemma 2.10. *The product poset $P \times Q$ of two monoidal posets $\langle P, \leq_P, I_P, \otimes_P \rangle$ and $\langle Q, \leq_Q, I_Q, \otimes_Q \rangle$ is also monoidal with:*

(i) Monoidal unit: $I_P \times I_Q$,
(ii) Monoidal product:

$$\begin{aligned}\otimes_{P\times Q}\colon \quad & (P\times Q)\times(P\times Q)\to(P\times Q),\\ & \langle\langle p_1, q_1\rangle, \langle p_2, q_2\rangle\rangle \mapsto \langle p_1 \otimes_P p_2,\ q_1 \otimes_Q q_2\rangle .\end{aligned}$$

Proof. All required properties follow directly from the definitions of monoidal poset (Definition 2.8) and product poset (Definition 2.6). □

Definition 2.11 (Monotone function)**.** A *monotone function* between posets $\langle P, \leq_P\rangle$ and $\langle Q, \leq_Q\rangle$ is a function $f\colon P \to Q$ such that, for all elements $x, y \in P$, if $x \leq_P y$ then $f(x) \leq_Q f(y)$.

Definition 2.12 (Upper set)**.** Given a poset $\langle P, \leq_P\rangle$, a subset $S \subseteq P$ is called *upper set* iff $x \in S$ and $x \leq_P y$ implies that $y \in S$. The set of all upper sets of P will be denoted by $\mathsf{U}P$.

Definition 2.13 (Lower set)**.** Given a poset $(P, \leq_P)$, a subset $S \subseteq P$ is called *lower set* iff $x \in S$ and $y \leq_P x$ implies that $y \in S$. The set of all lower sets of P will be denoted by $\mathsf{L}P$.

Definition 2.14 (Feasibility relation)**.** Let $P = \langle P, \leq_P\rangle$ and $Q = \langle Q, \leq_Q\rangle$ be posets. A *feasibility relation* for P given Q is a monotone function

$$\Phi\colon P^{\mathrm{op}} \times Q \to \mathtt{Bool}.$$

This is denoted by $\Phi\colon P \nrightarrow Q$. For $p \in P$ and $q \in Q$, if $\Phi(p, q) = \mathtt{T}$ we say *p can be obtained given q*.

Remark 2.15. A feasibility relation is a `Bool`-profunctor.

The feasibility relations admit a plethora of composition operations which are studied in detail by Censi et al. (2020). We will review the most important such operations in Chapter 7. Taking posets as objects and feasibility relations as morphisms one obtains the compact closed category **DP** (Censi et al., 2020).

2.2 Category theory

Category theory is the workhorse of this thesis. Hence, almost anything we do in the next chapters will be built up on top of its main concepts. This section should be treated only as review of these concepts. Should one wish to learn more about them, we recommend the *Seven Sketches in Compositionality* book by Fong and Spivak (2019) as an introductory text, the classic *Categories for the Working Mathematician* by Lane (1998) as a more general reference, and ncatlab.org as an all-around resource.

Definition 2.16 (Category)**.** To specify a category $\mathbf{C}$:

i. one specifies a collection $\mathsf{Ob}(\mathbf{C})$, elements of which are called *objects*;
ii. for every two objects $c, d \in \mathsf{Ob}(\mathbf{C})$, one specifies a set $\mathrm{Hom}_{\mathbf{C}}(c, d)$, the elements of which are called *morphisms* from c to d; any morphism $k \in \mathrm{Hom}_{\mathbf{C}}(c, d)$ can also be denoted as $k : c \to d$. The collection of all morphisms in the category is denoted by $\mathrm{Hom}_{\mathbf{C}}$;
iii. for every object $c \in \mathsf{Ob}(\mathbf{C})$, one specifies a morphism id_c called the *identity morphism* on c;
iv. for every three objects $c, d, e \in \mathsf{Ob}(\mathbf{C})$ and two morphisms $k \in \mathrm{Hom}_{\mathbf{C}}(c, d)$ and $l \in \mathrm{Hom}_{\mathbf{C}}(d, e)$, one specifies a morphism $k \mathbin{⨟} l \in \mathrm{Hom}_{\mathbf{C}}(c, e)$, called the *composite of k and l*.

The morphisms should satisfy the following two properties:

i. *Unitality:* for any morphism $k : c \to d$, composing with the identities at c or d doesn't do anything:

$$\mathrm{id}_c \mathbin{⨟} k = k = k \mathbin{⨟} \mathrm{id}_d;$$

ii. *Associativity:* for any three morphisms $k : c_0 \to c_1$, $l : c_1 \to c_2$, and $m : c_2 \to c_3$ we have

$$(k \mathbin{⨟} l) \mathbin{⨟} m = k \mathbin{⨟} (l \mathbin{⨟} m) = k \mathbin{⨟} l \mathbin{⨟} m.$$

Note that the existence of a morphisms only specifies that a relation between two objects exist, but does not define what this relation is. For example, there might be two different morphisms $k, l : c \to d$ with $k \neq l$.

Lemma 2.17. *Every poset (and in fact, also every preorder) is a category.*

Proof. The elements of the poset are the objects of the category and anytime $x \leq y$ in the poset, we have a morphism from x to y in the category. Due to the transitivity property of the posets, it follows that a poset is category with at most one element in any of its hom-sets. It is trivial to check that the structure of a poset meets all the requirements for a category. □

Example 2.18. As every poset is a category, that means that we can also have a `Bool` category defined in the exact same way as in Definition 2.3.

Definition 2.19 (Product category)**.** If $\mathbf{C}$ and $\mathbf{D}$ are two categories, then the product category $\mathbf{C} \times \mathbf{D}$ is constructed as following:

i. the collection of objects $\mathsf{Ob}(\mathbf{C} \times \mathbf{D})$ is the Cartesian product of $\mathsf{Ob}(\mathbf{C})$ and $\mathsf{Ob}(\mathbf{D})$, i.e. all pairs of objects from the two categories;
ii. for every two objects $\langle a, b\rangle, \langle c, d\rangle \in \mathsf{Ob}(\mathbf{C} \times \mathbf{D})$, the morphisms between them $\mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle a, b\rangle, \langle c, d\rangle)$ are the Cartesian product of $\mathrm{Hom}_{\mathbf{C}}(a, c)$ and $\mathrm{Hom}_{\mathbf{D}}(b, d)$, i.e. all pairs of morphisms (k, l), where $k : a \to c$ is a morphism in $\mathbf{C}$ and $l : b \to d$ is a morphism in $\mathbf{D}$;

iii. identity morphisms are defined as above: $\mathrm{id}_{\langle a,b\rangle} = (\mathrm{id}_a, \mathrm{id}_b)$;
iv. for every three objects $\langle a,b\rangle, \langle c,d\rangle, \langle e,f\rangle \in \mathsf{Ob}(\mathbf{C}\times\mathbf{D})$ and morphisms $\langle k,l\rangle \in \mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle a,b\rangle, \langle c,d\rangle)$ and $\langle m,n\rangle \in \mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle c,d\rangle, \langle e,f\rangle)$, the composite $\langle k,l\rangle \fatsemi \langle m,n\rangle$ is $\langle k \fatsemi m, l \fatsemi n\rangle$.

Lemma 2.20. *A product category is a category.*

Proof. If $\mathbf{C}$ and $\mathbf{D}$ are two categories, then Definition 2.19 is sufficient to define the required objects, morphisms, identity morphisms, and composites. Hence, the only thing left is to show that the so-constructed morphisms satisfy the unitality and associativity properties.

For any morphism $\langle k,l\rangle : \langle a,b\rangle \to \langle c,d\rangle$ we have

$$\mathrm{id}_{\langle a,b\rangle} \fatsemi \langle k,l\rangle = \langle \mathrm{id}_a \fatsemi k, \mathrm{id}_b \fatsemi l\rangle = \langle k,l\rangle .$$

The same holds for $\langle k,l\rangle \fatsemi \mathrm{id}_{\langle c,d\rangle}$.

For any three morphisms

$$\langle k,l\rangle \in \mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle a,b\rangle, \langle c,d\rangle),$$
$$\langle m,n\rangle \in \mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle c,d\rangle, \langle e,f\rangle),$$
$$\langle p,q\rangle \in \mathrm{Hom}_{\mathbf{C}\times\mathbf{D}}(\langle e,f\rangle, \langle g,h\rangle),$$

we have:

$$\begin{aligned}((k,l) \fatsemi (m,n)) \fatsemi (p,q) &= (k \fatsemi m, l \fatsemi n) \fatsemi (p,q) \\ &= ((k \fatsemi m) \fatsemi p, (l \fatsemi n) \fatsemi q) \\ &= (k \fatsemi (m \fatsemi p), l \fatsemi (n \fatsemi q)) \\ &= (k \fatsemi l) \fatsemi (m \fatsemi p, n \fatsemi q) \\ &= (k \fatsemi l) \fatsemi ((m,n) \fatsemi (p,q)),\end{aligned}$$

due to the associativity of the morphisms in $\mathbf{C}$ and $\mathbf{D}$.

Therefore, the morphisms of $\mathbf{C}\times\mathbf{D}$ satisfy the unitality and associativity properties, and $\mathbf{C}\times\mathbf{D}$ is a category. □

Definition 2.21 (Subcategory)**.** Given a category $\mathbf{C}$, a *subcategory* $\mathbf{D}$ of $\mathbf{C}$ has:
i. objects being a subcollection of $\mathsf{Ob}(\mathbf{C})$;
ii. morphisms being a subcollection of the morphisms of $\mathbf{C}$,

such that:
i. if the morphism $k : c \to d$ is in $\mathbf{D}$, then $c, d \in \mathsf{Ob}(\mathbf{D})$;
ii. if $k : c \to d$ and $l : d \to e$ are in $\mathbf{D}$, then their composite $k \fatsemi l$ is also in $\mathbf{D}$;
iii. if $c \in \mathsf{Ob}(\mathbf{D})$, then the identity morphism $\mathrm{id}_c^{\mathbf{C}}$ is also in $\mathbf{D}$.

Definition 2.22 (Functor)**.** Let $\mathbf{C}$ and $\mathbf{D}$ be categories. To specify a *functor from* $\mathbf{C}$ *to* $\mathbf{D}$, denoted $F : \mathbf{C} \to \mathbf{D}$:

i. for every object $c \in \mathsf{Ob}(\mathbf{C})$, one specifies an object $F(c) \in \mathsf{Ob}(\mathbf{D})$;
ii. for every morphism $k : c \longrightarrow d$ in $\mathbf{C}$, one specifies a morphism in $\mathbf{D}$:

$$F(k) : F(c) \longrightarrow F(d).$$

Furthermore, the following two properties need to be satisfied:
i. for every object $c \in \mathsf{Ob}(\mathbf{C})$, we have that $F(\mathrm{id}_c) = \mathrm{id}_{F(c)}$;
ii. for every three objects $c, d, e \in \mathsf{Ob}(\mathbf{C})$ and for every two morphisms $k \in \mathrm{Hom}_{\mathbf{C}}(c, d)$ and $l \in \mathrm{Hom}_{\mathbf{C}}(d, e)$, the equation

$$F(k \mathbin{⨟} l) = F(k) \mathbin{⨟} F(l)$$

holds in $\mathbf{D}$.

Remark 2.23. One can think of a functor as a map between two categories which respects their structures.

Definition 2.24 (Bifunctor)**.** A *bifunctor* or *functor of two variables* is a functor whose domain is the product of two categories. For the three categories $\mathbf{C}_1$, $\mathbf{C}_2$, and $\mathbf{D}$, a bifunctor B from $\mathbf{C}_1$ and $\mathbf{C}_2$ to $\mathbf{D}$ is a functor $B : \mathbf{C}_1 \times \mathbf{C}_2 \longrightarrow \mathbf{D}$.

Definition 2.25 (Covariant and contravariant functor)**.** A functor F is called *covariant* if it preserves the directions of the morphisms, i.e. for every morphism $f \colon A \longrightarrow B$, one has $F(f) \colon F(A) \longrightarrow F(B)$. A functor is called *contravariant* if it *reverses* the directions of the morphisms, i.e. for every morphism $f \colon A \longrightarrow B$, one has $F(f) \colon F(B) \longrightarrow F(A)$. This can also be seen as a functor from the opposite category $\mathbf{A}^{\mathrm{op}}$ to $\mathbf{B}$.

Definition 2.26 (Isomorphism)**.** An *isomorphism* is a morphism $k : c \longrightarrow d$ in a category $\mathbf{C}$ such that there exists another morphism $l : d \longrightarrow c$ satisfying $k \mathbin{⨟} l = \mathrm{id}_c$ and $l \mathbin{⨟} k = \mathrm{id}_d$. The two morphisms k and l are called *inverses* of each other.

Definition 2.27 (Isomorphic objects)**.** If two objects $c, d \in \mathsf{Ob}(\mathbf{C})$ have a pair of isomorphisms between them, then the two objects are called *isomorphic objects*.

Remark 2.28. Given two isomorphic objects in a category, they always behave in the same way when composed with other morphisms. The reason is that if one of the objects has an outgoing morphism, then the other object can also use it by pre-composing with the respective component of the isomorphism. The same, of course, holds for the incoming morphisms as well. Hence such objects are also referred to as *the same up to an isomorphism*.

Definition 2.29 (Symmetric monoidal category)**.** A symmetric monoidal category is a category $\mathbf{C}$ equipped with a symmetric monoidal structure. A symmetric monoidal structure consists of:

i. a bifunctor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, called *monoidal product;*
ii. an object $I \in \mathsf{Ob}(\mathbf{C})$, called *monoidal unit,*

which have to satisfy the following isomorphisms:

i. $\lambda_c : I \otimes c \to c$, and its inverse $\lambda_c^{-1} : c \to I \otimes c$, for every $c \in \mathsf{Ob}(\mathbf{C})$;
ii. $\rho_c : c \otimes I \to c$, and its inverse $\rho_c^{-1} : c \to c \otimes I$, for every $c \in \mathsf{Ob}(\mathbf{C})$;
iii. $\alpha_{c,d,e} : (c \otimes d) \otimes e \to c \otimes (d \otimes e)$, and its inverse $\alpha^{-1}_{c,d,e} : c \otimes (d \otimes e) \to (c \otimes d) \otimes e$, for every $c, d, e \in \mathsf{Ob}(\mathbf{C})$;
iv. $\sigma_{c,d} : c \otimes d \to d \otimes c$, for every $c, d \in \mathsf{Ob}(\mathbf{C})$, such that $\sigma_{c,d} \fatsemi \sigma_{d,c} = \mathrm{id}_{c \otimes d}$.

If only condition iv. does not hold, then the resulting category is called a *monoidal category.*

Remark 2.30. It is easy to see that the same way that a category generalizes a poset, a symmetric monoidal category generalizes a symmetric monoidal poset. In other words, any symmetric monoidal poset is a symmetric monoidal category.

Definition 2.31 (Strong monoidal functor)**.** Given two monoidal categories $\langle \mathbf{C}, \otimes_{\mathbf{C}}, I_{\mathbf{C}} \rangle$ and $\langle \mathbf{D}, \otimes_{\mathbf{D}}, I_{\mathbf{D}} \rangle$, a *strong monoidal functor* from $\mathbf{C}$ to $\mathbf{D}$ consists of

i. a functor $F \colon \mathbf{C} \to \mathbf{D}$;
ii. a natural isomorphism $\mu_{A,B}$ for every pair A, B of elements in $\mathbf{C}$, such that:
$$\mu_{A,B} \colon F(A) \otimes_{\mathbf{D}} F(B) \to F(A \otimes_{\mathbf{C}} B);$$
iii. an isomorphism $\epsilon : I_{\mathbf{D}} \to F(I_{\mathbf{C}})$,

satisfying the following properties:

i. *Associativity*: for all $A, B, C \in \mathbf{C}$ the following diagram commutes:

$$\begin{tikzcd}
(F(A) \otimes_{\mathbf{D}} F(B)) \otimes_{\mathbf{D}} F(C) \arrow[r, "\alpha^{\mathbf{D}}_{F(A)F(B),F(C)}"] \arrow[d, "\mu_{A,B} \otimes_{\mathbf{D}} \mathrm{id}^{\mathbf{D}}_{F(C)}"] & F(A) \otimes_{\mathbf{D}} (F(B) \otimes_{\mathbf{D}} F(C)) \arrow[d, "\mathrm{id}^{\mathbf{D}}_{F(A)} \otimes_{\mathbf{D}} \mu_{B,C}"] \\
F(A \otimes_{\mathbf{C}} B) \otimes_{\mathbf{D}} F(C) \arrow[d, "\mu_{(A \otimes_{\mathbf{C}} B),C}"] & A \otimes_{\mathbf{D}} F(B \otimes_{\mathbf{C}} F(C)) \arrow[d, "\mu_{(A \otimes_{\mathbf{C}} B),C}"] \\
F((A \otimes_{\mathbf{C}} B) \otimes_{\mathbf{C}} C) \arrow[r, "F(\alpha^{\mathbf{C}}_{AB,C})"] & F(A \otimes_{\mathbf{C}} (B \otimes_{\mathbf{C}} C)),
\end{tikzcd} \tag{2.1}$$

where $\alpha^{\mathbf{C}}$ and $\alpha^{\mathbf{D}}$ are associator morphisms in $\mathbf{C}$ and $\mathbf{D}$, and $\mathrm{id}^{\mathbf{C}}$ and $\mathrm{id}^{\mathbf{D}}$ are their identity morphisms;

ii. *Unitality*: for all $A \in \mathbf{C}$ the following diagrams commute:

$$\begin{tikzcd}
I_{\mathbf{D}} \otimes_{\mathbf{D}} F(A) \arrow[r, "\epsilon \otimes_{\mathbf{D}} \mathrm{id}^{\mathbf{D}}_{F(A)}"] \arrow[d, "\lambda^{\mathbf{D}}_{F(A)}"] & F(I_{\mathbf{C}}) \otimes_{\mathbf{D}} F(A) \arrow[d, "\mu_{I_{\mathbf{C}},A}"] \\
F(A) & F(I_{\mathbf{C}} \otimes_{\mathbf{C}} A) \arrow[l, "F(\lambda^{\mathbf{C}}_A)"']
\end{tikzcd} \tag{2.2}$$

and

$$\begin{tikzcd}
F(A) \otimes_{\mathbf{D}} I_{\mathbf{D}} \arrow[r, "\mathrm{id}^{\mathbf{D}}_{F(A)} \otimes_{\mathbf{D}} \epsilon"] \arrow[d, "\rho^{\mathbf{D}}_{F(A)}"] & F(A) \otimes_{\mathbf{D}} F(I_{\mathbf{C}}) \arrow[d, "\mu_{A,I_{\mathbf{C}}}"] \\
F(A) & F(A \otimes_{\mathbf{C}} I_{\mathbf{C}}), \arrow[l, "F(\rho^{\mathbf{C}}_{A})"]
\end{tikzcd} \tag{2.3}$$

where $\lambda^{\mathbf{C}}$, $\lambda^{\mathbf{D}}$, $\rho^{\mathbf{C}}$, and $\rho^{\mathbf{D}}$ are the left and right unitors of $\mathbf{C}$ and $\mathbf{D}$.

Definition 2.32 (Compact closed category)**.** Let $\mathbf{C}$ be a category, $(\mathbf{C}, I, \otimes)$ be a symmetric monoidal structure on it, and $c \in \mathsf{Ob}(\mathbf{C})$ an object. A *dual for c* is an object $c^* \in \mathsf{Ob}(\mathbf{C})$ and the following morphisms:

i. *unit for c:* $\eta_c : I \to c^* \otimes c$;
ii. *counit for c:* $\epsilon_c : c \otimes c^* \to I$,

satisfying the following two equations:

i. $\rho_c^{-1} \fatsemi (\mathrm{id}_c \otimes \eta_c) \fatsemi \alpha^{-1}_{c,c^*,c} \fatsemi (\epsilon_c \otimes \mathrm{id}_c) \fatsemi \lambda_c = \mathrm{id}_c$;
ii. $\lambda^{-1}_{c^*} \fatsemi (\eta_c \otimes \mathrm{id}_{c^*}) \fatsemi \alpha_{c,c^*,c} \fatsemi (\mathrm{id}_{c^*} \otimes \epsilon_c) \fatsemi \rho_{c^*} = \mathrm{id}_{c^*}$.

Definition 2.33 (Small category)**.** A category $\mathbf{C}$ is a *small* category if both $\mathsf{Ob}(\mathbf{C})$ and $\mathrm{Hom}_{\mathbf{C}}(A, B)$ are sets, for all $A, B \in \mathsf{Ob}(\mathbf{C})$.

Definition 2.34 (Semicategory)**.** To specify a *semicategory* $\mathbf{C}^*$ (also known as *semigroupoid* or a *precategory*):

(i) one specifies a collection $\mathsf{Ob}(\mathbf{C}^*)$, elements of which are called *objects*;
(ii) for every two objects $c, d \in \mathsf{Ob}(\mathbf{C}^*)$, one specifies a set $\mathrm{Hom}_{\mathbf{C}^*}(c, d)$, the elements of which are called *morphisms* from c to d; a morphism $k \in \mathrm{Hom}_{\mathbf{C}^*}(c, d)$ can also be denoted as $k : c \to d$;
(iii) for every three objects $c, d, e \in \mathsf{Ob}(\mathbf{C}^*)$ and morphisms $k \in \mathrm{Hom}_{\mathbf{C}^*}(c, d)$ and $l \in \mathrm{Hom}_{\mathbf{C}^*}(d, e)$, one specifies a morphism $k \fatsemi l \in \mathrm{Hom}_{\mathbf{C}^*}(c, e)$, called the *composite of k and l*.

The morphisms should have the property that for any three morphisms $k : c_0 \to c_1$, $l : c_1 \to c_2$, and $m : c_2 \to c_3$ it holds that:

$$(k \fatsemi l) \fatsemi m = k \fatsemi (l \fatsemi m) = k \fatsemi l \fatsemi m.$$

Remark 2.35. A semicategory is simply a category without requiring identity morphisms and the identity property on the morphisms. It is clear that any category is also a semicategory.

Definition 2.36 (Semifunctor)**.** Let $\mathbf{C}^*$ and $\mathbf{D}^*$ be semicategories. To specify a *semifunctor from $\mathbf{C}^*$ to $\mathbf{D}^*$*, denoted $F : \mathbf{C}^* \to \mathbf{D}^*$:

i. for every object $c \in \mathsf{Ob}(\mathbf{C}^*)$, one specifies an object $F(c) \in \mathsf{Ob}(\mathbf{D}^*)$;
ii. for every morphism $k : c \to d$ in $\mathbf{C}^*$, one specifies a morphism in $\mathbf{D}^*$:

$$F(k) : F(c) \to F(d).$$

Furthermore, for every three objects $c, d, e \in \mathsf{Ob}(\mathbf{C}^*)$ and for every two morphisms $k \in \mathrm{Hom}_{\mathbf{C}^*}(c, d)$ and $l \in \mathrm{Hom}_{\mathbf{C}^*}(d, e)$, the equation

$$F(k \fatsemi l) = F(k) \fatsemi F(l)$$

must hold in $\mathbf{D}^*$.

Definition 2.37 (The **Cat** category)**.** The **Cat** category is constructed as following:

i. objects are all small categories (Definition 2.33);
ii. morphisms are all functors between small categories;
iii. identity morphisms are identity functors;
iv. morphism composition is functor composition.

Definition 2.38 (The **SemiCat** category)**.** The **SemiCat** category is constructed as following:

i. objects are all small semicategories (Definition 2.34);
ii. morphisms are all semifunctors (Definition 2.36);
iii. identity morphisms are identity semifunctors;
iv. morphism composition is semifunctor composition.

Definition 2.39 (The **Set** category)**.** The **Set** category is constructed as following:

i. objects are all sets;
ii. morphisms from a set A to a set B are all functions $f : A \to B$;
iii. identity morphisms are identity functions;
iv. morphism composition is function composition.

Definition 2.40 (The **Rel** category)**.** The **Rel** category is constructed as following:

i. objects are all sets;
ii. morphisms from a set A to a set B are all binary relations $R \subseteq A \times B$;
iii. identity morphisms are identity binary relations;
iv. morphism composition is the composition of binary relations.

Definition 2.41 (Twisted category)**.** Given a category $\mathbf{C}$, the *twisted category on* $\mathbf{C}$, denoted by $\mathbf{Tw}(\mathbf{C})$ and also called *category of factorizations*, has

i. Objects which are morphisms in $\mathbf{C}$;
ii. Morphisms between two objects $f : A \to B$ and $g : C \to D$ which are pairs of morphisms $\langle p : C \to A, q : B \to D \rangle$, such that the following diagram commutes:

$$\begin{tikzcd} A \arrow[d, "f"] & C \arrow[l, "p"'] \arrow[d, "g"] \\ B \arrow[r, "q"] & D \end{tikzcd}$$

in other words morphisms p and q such that $g = p \mathbin{;} f \mathbin{;} q$;

iii. Identity morphism for $f : A \to B$ is the pair $\langle \mathrm{id}^{\mathbf{C}}_A, \mathrm{id}^{\mathbf{C}}_B \rangle$ of identity morphisms in $\mathbf{C}$;

iv. The composition of two morphisms

$$\langle h : E \to C, i : D \to F \rangle$$

and

$$\langle p : C \to A, q : B \to D \rangle$$

is

$$\langle h, i \rangle \mathbin{;} \langle p, q \rangle = \langle h \mathbin{;}_{\mathbf{C}} p,\ q \mathbin{;}_{\mathbf{C}} i \rangle .$$

Definition 2.42 (Natural transformation)**.** Given two categories $\mathbf{C}$ and $\mathbf{D}$, and functors $F, G \colon \mathbf{C} \to \mathbf{D}$, a *natural transformation* $\alpha \colon F \Rightarrow G$ between them is an assignment of a morphism $\alpha_x \colon F(x) \to G(x)$ (called *component of* α *at* x) in $\mathbf{D}$ for every object $x \in \mathsf{Ob}(\mathbf{C})$ such that the following diagram commutes

$$\begin{tikzcd}
F(x) \arrow[r, "F(f)"] \arrow[d, "\alpha_x"] & F(y) \arrow[d, "\alpha_y"] \\
G(x) \arrow[r, "G(f)"] & G(y)
\end{tikzcd}$$

for any morphism $f \in \mathrm{Hom}_{\mathbf{C}}(x, y)$, i.e. $F(f) \mathbin{;} \alpha_y = \alpha_x \mathbin{;} G(f)$.

Definition 2.43 (Natural isomorphism)**.** A *natural isomorphism* $\alpha : F \Rightarrow G$ between two functors $F, G \colon \mathbf{C} \to \mathbf{D}$ is a natural transformation such that each of its components $\alpha_x \colon F(x) \to G(x)$, for all $x \in \mathsf{Ob}(\mathbf{C})$, is an isomorphism in $\mathbf{D}$.

2.3 Anonymous functions

In order to keep the presentation succinct we will make a frequent use of anonymous functions. Here we introduce the notation that will be employed when dealing with them.

When we wish to denote a function of one variable $f : X \to Y$ in anonymous function notation we will write

$$f := [x \mapsto f(x)] .$$

Note that the choice of the symbol x is completely arbitrary and can be changed:

$$f := [x \mapsto f(x)] = [\xi \mapsto f(\xi)] .$$

We will sometimes explicitly specify the domain and range of the anonymous function by adding them on top of the map symbol:

$$f := \left[x \xmapsto{X \to Y} f(x)\right] = \left[\xi \xmapsto{X \to Y} f(\xi)\right].$$

We denote the application of an anonymous function to an argument by adding the argument in parenthesis after the function, i.e.

$$f(x') = \left[\xi \xmapsto{X \to Y} f(\xi)\right](x'),\ x' \in X.$$

Example 2.44 (Identity function)**.** The identity function can be expressed as the anonymous function $[x \mapsto x]$. Furthermore, clearly we also have $[x \mapsto x]\,(x') = x'$.

Analogously, we can have anonymous functions of multiple parameters, e.g.

$$\left[x, y \xmapsto{\mathbb{R}^2 \to \mathbb{R}} x^y\right].$$

Example 2.45 (Quadratic function)**.** The quadratic function can be expressed as the anonymous function

$$\left[x \xmapsto{\mathbb{R} \to \mathbb{R}} ax^2 + bx + c\right].$$

Note how $a, b, c \in \mathbb{R}$ are not defined. We say that x is *bound* in this anonymous function, while a, b, and c are *free variables*.

Definition 2.46 (Composition of anonymous functions)**.** Given two anonymous functions

$$\left[x \xmapsto{X \to Y} f(x)\right]$$

and

$$\left[y \xmapsto{Y \to Z} g(y)\right],$$

such that the variable x is not a free variable in the second function, we can compose them by applying one to the other:

$$\left[y \xmapsto{Y \to Z} g(y)\right]\left(\left[x \xmapsto{X \to Y} f(x)\right]\right) = \left[x \xmapsto{X \to Z} g(f(x))\right].$$

Remark 2.47. In the case that x *is* a free variable in the second anonymous function, we would have to first rewrite

$$\left[x \xmapsto{X \to Y} f(x)\right]$$

into
$$\left[x' \xmapsto{X\to Y} f(x')\right]$$
with x' not a free variable in the second function.

Example 2.48. To illustrate why the substitution above is important, take the quadratic and identity functions as before with signatures as:
$$\left[x \xmapsto{\mathbb{R}\to\mathbb{R}} ax^2 + bx + c\right] \quad \text{and} \quad [a \mapsto a] .$$
If we do not perform substitution composing the two expressions will leave us with:
$$\left[x \mapsto ax^2 + bx + c\right]([a \mapsto a]) = \left[a \mapsto a^3 + ba + c\right] ,$$
which is clearly not the same as the desired result:
$$\begin{aligned}\left[x \mapsto ax^2 + bx + c\right]([a \mapsto a]) &= \left[x \mapsto ax^2 + bx + c\right]([a' \mapsto a']) \\ &= \left[a' \mapsto aa'^2 + ba' + c\right] \\ &= \left[x \mapsto ax^2 + bx + c\right] .\end{aligned}$$

We will also make a frequent use of the following two functions which extract an element of a tuple:

Definition 2.49 (`fst` and snd)**.** For any two sets X and Y, we define the following two functions:
$$\begin{aligned}\texttt{fst} &= \left[x, y \xmapsto{X\times Y\to X} x\right] , \\ \texttt{snd} &= \left[x, y \xmapsto{X\times Y\to Y} y\right] .\end{aligned}$$

Definition 2.50 (Product of functions)**.** Given two functions $f : X \to Y$ and $g : Z \to W$, the product function $f \times g$ is defined as:
$$\begin{aligned}f \times g \colon X \times Z &\to Y \times W, \\ \langle x, z\rangle &\mapsto \left\langle f(x), g(z)\right\rangle .\end{aligned}$$

2.4 Type theory

In this section we will do a quick and dirty review of some of the basic concepts of type theory that will be used for the rest of the thesis. The readers interested in more detailed and rigorous treatment can refer to (Pierce, 2002; The Univalent Foundations Program, 2013), and for a study of type theory's connections with category theory, (Crole, 1994). The review here follows the book by Crole.

Definition 2.51 (Type system)**.** A *type system* is a formal system which consists of:

i. a collection of *types*;
ii. a collection of *function symbols*, each associated with a natural number called an *arity*;
iii. a *sorting* for each function symbol f which is a list of $a + 1$ types $([\alpha_1, \ldots, \alpha_a, \alpha])$ and will be written

$$f : \alpha_1, \ldots, \alpha_a \to \alpha,$$

where a is the arity of f.

If the arity of a function symbol k is 0, we will denote it by $k : \alpha$, and will say that *k is a constant function symbol of type α*.

Definition 2.52 (Terms of a type system)**.** The terms of a type system is the collection of:

i. *variable terms*: a countably infinite set of variables $(x, y, \ldots)$;
ii. *constant terms*: all constant function symbols of the type system $(k, l, \ldots)$;
iii. *function terms*: all expressions $f(M_1, \ldots, M_a)$, where f is any function symbol with any arity $a > 0$ and each of $M_1, \ldots, M_a$ is a term of the type system.

Remark 2.53. Note that it is the recursive formulation in the last point in the above definition that provides the expressivity of a type system.

Definition 2.54 (Free variables of terms)**.** Every term of a type system has a set of *free variables*, which is defined inductively as:

i. $\mathsf{FV}(x) = \{x\}$, where x is a variable term;
ii. $\mathsf{FV}(k) = \varnothing$, where k is a constant term;
iii. $\mathsf{FV}(f(M_1, \ldots, M_a)) = \bigcup_{i=1}^{a} \mathsf{FV}(M_i)$, where $f(M_1, \ldots, M_a)$ is a function term.

Remark 2.55. When working with types it is important to be careful not to reuse the same symbol for different terms. That is why, sometimes, when substituting terms, one needs to change the variable names, in order to keep the semantics of the combined term as expected. This is also called *α-conversion*. In this work, we will implicitly take care of such conversions, but we will nevertheless provide formal explanation of the substitution process.

Definition 2.56 (Term substitution)**.** A variable x in a term M can be *substituted* with another term N, an operation denoted by $M[N/x]$ inductively as follows:

i. if M is the same variable x, then

$$M[N/x] = x[N/x] = N;$$

ii. if M is another variable y, then

$$M[N/x] = y[N/x] = y;$$

iii. if M is a constant term, then

$$M[N/x] = k[M/x] = k;$$

iv. if M is a function term $f(M_1, \ldots, M_a)$ of non-zero arity a, then

$$M[N/x] = f(M_1, \ldots, M_a)[N/x] = f(M_1[N/x], \ldots, M_a[N/x]).$$

Furthermore, if $\mathbf{x} = [x_1, \ldots, x_n]$ is any list of n distinct variables, and if $\mathbf{N} = [N_1, \ldots, N_n]$ is a list of n terms, then the *simultaneous substitution* of the variables $\mathbf{x}$ with the terms $\mathbf{N}$, denoted $M[\mathbf{N}/\mathbf{x}]$ or $M[N_1/x_1, \ldots, N_n/x_n]$, is recursively defined as:

$$M[N_1/x_1, \ldots, N_n/x_n] = M[z_1/x_1][N_2/x_2, \ldots, N_n/x_n][N_1/z_1],$$

or more explicitly:

$$M[N_1/x_1, \ldots, N_n/x_n] = M[z_1/x_1] \ldots [z_n/x_n][N_n/z_n] \ldots [N_1/z_1].$$

In both places, the $z_1, \ldots, z_n$ are distinct variables, also distinct from $x_1, \ldots, x_n$, such that

$$z_i \notin \mathsf{FV}(M) \cup \bigcup_{i=1}^{n} \mathsf{FV}(N_i).$$

Remark 2.57. The above definition of simultaneous substitution is obtained via successive applications of point 7.6 from (Curry, 1952). Crole (1994) has an alternative but equivalent definition:

$$M[N_1/x_1, N_2/x_2] = M[N_1[z/x_2]/x_1][N_2/x_2][x_2/z].$$

Remark 2.58. The problem mentioned in Remark 2.55 can now be formally posed. Take M, N, N' to be any terms in a type system, and x, y, z be distinct variables. Then the term $M[N/x, N'/y]$ is not in general syntactically equivalent to $M[N/x][N'/y]$. This is presented as Exercise 3.2.6 in (Crole, 1994).

Definition 2.59 (Typing context)**.** A *typing context* Γ is a sequence of terms and their types, denoted by

$$\Gamma := [x_1 : \alpha_1, \ldots, x_n : \alpha_n].$$

We say that the *variable* x_1 *appears in* Γ *with type* α_1. A typing context can also be empty.

Definition 2.60 (Context concatenation)**.** Contexts can be combined by list concatenation, denoted Γ, Γ' or $\Gamma, x : \alpha, \Gamma'$.

Example 2.61. An example of a typing context is

$$\Gamma_a = \left[x \colon T_1,\ y \colon T_2\right].$$

A typing context can be extended via concatenation:

$$\Gamma_b \coloneqq \Gamma_a,\ z \colon T_3 = \left[x \colon T_1,\ y \colon T_2,\ z \colon T_3\right].$$

Definition 2.62 (Proved terms)**.** Given a context Γ, we write $\Gamma \vdash M \colon \alpha$ to denote a *proved term*. Proved terms are generated inductively by the following rules:

i. $\Gamma, x : \alpha, \Gamma' \vdash x : \alpha$ is a proved term, where Γ and Γ' are typing contexts, x is a variable, and α is a type;
ii. $\Gamma \vdash k : \alpha$, where Γ is a typing context and k is a constant term of a constant function symbol of type α;
iii. if $\Gamma \vdash M_1 : \alpha_1, \dots,$ and $\Gamma \vdash M_a : \alpha_a$, and f is a function symbol with arity a and sorting $f : \alpha_1, \dots, \alpha_a \longrightarrow \alpha$, then:

$$\Gamma \vdash f(M_1, \dots, M_a) : \alpha.$$

Whenever $\Gamma \vdash M : \alpha$ is a proved term, we will say that *the term M has type α in the context of* Γ.

Remark 2.63. Due to the recursive definition of proved terms, given a proved term $\Gamma \vdash M : \alpha$, we know that the context Γ implies that M can be only of type α, and is also a *well-formed* term, which intuitively means that it is formed with every single one of its constituent terms also being proved terms. This is formally presented as Proposition 3.2.12 and Exercise 3.2.15.1 in (Crole, 1994). Throughout the rest of this thesis we will only consider proved terms. So whenever we refer to *term*, we mean a *proved term*.

Definition 2.64 (Unit type)**.** The *unit type I* has the single term ι. Alternatively, there is only one term with the unit type I and this term is denoted by ι.

Definition 2.65 (Product type)**.** Given two types α_1 and α_2, we construct the *product type* $\alpha_1 \times \alpha_2$. Furthermore, if in some context Γ we have a term $\Gamma \vdash x_1 \colon \alpha_1$ and a term $\Gamma \vdash x_2 \colon \alpha_2$, the term $\langle x_1, x_2 \rangle$ has a type $\alpha_1 \times \alpha_2$ in the context of Γ, i.e,

$$\Gamma \vdash \langle x_1, x_2 \rangle : \alpha_1 \times \alpha_2.$$

Definition 2.66 (Function type)**.** Given two types α_1 and α_2, we construct the *function type* $\alpha_1 \longrightarrow \alpha_2$ with domain α_1 and codomain α_2. Given some context Γ, a term f of the type $\alpha_1 \longrightarrow \alpha_2$ (i.e. $\Gamma \vdash f \colon \alpha_1 \longrightarrow \alpha_2$) and a term $\Gamma \vdash x \colon \alpha_1$, there is a term $\Gamma \vdash f(x) \colon \alpha_2$ called *value of f at a*.

Remark 2.67. Function types are *not* the same thing as functions. Functions $f : A \longrightarrow \alpha_2$ *have type* $A \longrightarrow \alpha_2$. Function types are also *not* function symbols or function terms.

Definition 2.68 (Association bijection)**.** Given three types A, B, and C we define the following two *association* functions:

$$\begin{aligned} \alpha_{AB,C} \colon (A \times B) \times C &\longrightarrow A \times (B \times C), \\ \langle \langle a, b \rangle, c \rangle &\mapsto \langle a, \langle b, c \rangle \rangle ; \end{aligned}$$

$$\begin{aligned} \alpha_{A,BC} \colon A \times (B \times C) &\longrightarrow (A \times B) \times C, \\ \langle a, \langle b, c \rangle \rangle &\mapsto \langle \langle a, b \rangle, c \rangle . \end{aligned}$$

These functions are each other's inverse, hence they form a *bijection*. The exact same association bijection can be defined for A, B, and C being sets. We abuse the notation $\alpha_{AB,C}$ and $\alpha_{A,BC}$ by using them for both types and sets.

Chapter 3

Normed types and procedures

We start by introducing some of the building blocks of this work: *normed types* and *procedures*. Normed types are simply types whose terms are endowed with some notion of size. We are interested in the “size” variation between different terms because later some of these terms will be used to represent “easy” problems while others will be used to represent “difficult” problems. This hierarchy of “difficultness” will be represented by an order on the size of terms. Procedures, on the other hand, will be our representation of *computational* processes. A procedure connects two normed types and quantifies the resources necessary to convert a term of one of the types to a term of the other type. It does this while taking into account the resources needed for this operation, which in turn depend on the size of the terms.

3.1 Normed types

As mentioned above, normed types are nothing but types endowed with a notion of size. We leave the way of measuring a given type open. After all, types which represent different things have different notions of size. Some types can even have multiple different sensible choices for assigning sizes, as we will see in Example 3.6. We do, however, ask for the size to have a notion of order, meaning that we can ascertain whether one term is smaller, larger, equal, or incomparable to any other term. This means that we ask that the sizes of a normed type form a poset.

Definition 3.1 (Normed type)**.** A *normed type* is a triplet

$$\mathbf{A} := \langle A, |A|, s\colon A \to |A| \rangle \,,$$

where A is a type, $|A|$ is a poset called *sizes of* **A**, and s is a map from the terms of A to the sizes $|A|$. The elements of A will be also called *terms of* **A**.

Remark 3.2. Note that we somewhat abuse the label "normed" in the above definition. The sizes that we endow **A** with by no means constitute a proper norm. That is, they do not satisfy a triangle inequality and are not absolutely scalable.

Example 3.3 (Lists)**.** Given a type A, we define the type $\mathsf{List}[A]$, such that there is a term:

$$\langle\rangle : \mathsf{List}[A],$$

and a term $\langle a, l\rangle$:

$$[a\colon A,\ l\colon \mathsf{List}[A]] \vdash \langle a, l\rangle : \mathsf{List}[A].$$

A typical notion of size of a list is its length which has its values in $\mathbb{N}_0$. We can then recursively define a function

$$\begin{aligned} s_{\mathsf{List}} : \mathsf{List} &\longrightarrow \mathbb{N}_0, \\ \langle\rangle &\longmapsto 0, \\ \langle a, l\rangle &\longmapsto 1 + s_{\mathsf{List}}(l). \end{aligned}$$

Note that s_{List} has the same definition regardless of the underlying type A. Now, the triplet $\mathbf{List}[A] := \langle \mathsf{List}[A], \mathbb{N}_0, s_{\mathsf{List}}\rangle$ is a normed type.

Remark 3.4. In the above example, we have allowed ourselves to abuse the functional notation a bit. Strictly speaking, a type is not a set. If A is a type and B is a set, then

$$f : A \longrightarrow B$$

does not make much sense. Hence, a function from a type to a set is not well-defined. Nevertheless, we will pretend that the collection of terms of a given type forms a set and such functions would be associating every terms of the type A with a single element from the set B.

Remark 3.5. Technically, List is a generic type and an example of parametric polymorphism in type theory. The interested reader can find more details in (Crole, 1994, Chapter 5) or in (Pierce, 2002, Part V). We will not consider this in further detail and will take any $\mathsf{List}[A]$ to be an ordinary type, as introduced in Definition 2.51.

Example 3.6 (Directed unweighted graphs)**.** Throughout this work, we will be making a frequent use of some running examples using graphs. These will be done under the assumption that the reader is familiar with the basic terms in graph theory. Otherwise, *Graphs, Networks and Algorithms* by Jungnickel (2013) is a good reference.

We denote by $\mathbb{N}$ the type of natural numbers and by $\mathbb{B}$ the boolean type which has only two terms: `T` and `F`. Then, for any $n : \mathbb{N}$, we define the type

$$\vec{G}[n] := \mathbb{B}^{n^2},$$

where the exponentiation of the type is simply consecutive application of the type product operation $\times$. The key observation here is that when

$$(n : \mathbb{N}) \vdash g : \vec{G}[n],$$

then g represents the adjacency matrix of a directed graph with n nodes.

We can define several different size functions on the terms with type $\vec{G}[n]$:

i. the number of nodes:

$$\begin{aligned} s_{\vec{G}}^{\text{nodes}} \colon \vec{G}[n] &\to \mathbb{N}, \\ g &\mapsto n; \end{aligned}$$

ii. the number of edges:

$$\begin{aligned} s_{\vec{G}}^{\text{edges}} \colon \vec{G}[n] &\to \mathbb{N}_0, \\ g &\mapsto \sum_{i=1}^{n^2} \mathbb{1}_{g_i}, \end{aligned}$$

where the indicator function $\mathbb{1}_{g_i}$ is 1 if the i-th component of g is `T` and 0 otherwise;

iii. the number of nodes and edges:

$$\begin{aligned} s_{\vec{G}}^{\text{n+e}} \colon \vec{G}[n] &\to \mathbb{N} \times \mathbb{N}_0, \\ g &\mapsto \left\langle n, \sum_{i=1}^{n^2} \mathbb{1}_{g_i} \right\rangle. \end{aligned}$$

iv. the number of cycles:

$$\begin{aligned} s_{\vec{G}}^{\text{cycles}} \colon \vec{G}[n] &\to \mathbb{N}_0, \\ g &\mapsto \mathsf{cycles}(g), \end{aligned}$$

where cycles is a function that counts the number of cycles in the graph.

Hence, for any $n : \mathbb{N}$ we can define the following three distinct normed types, all of which have the same underlying type but different ways to measure their size:

$$\vec{\mathbf{G}}^{\text{nodes}}[n] := \left\langle \vec{G}[n], \mathbb{N}, s_{\vec{G}}^{\text{nodes}} \right\rangle,$$

$$\begin{aligned}\vec{\mathbf{G}}^{\text{edges}}[n] &:= \left\langle \vec{G}[n],\, \mathbb{N}_0,\, s^{\text{edges}}_{\vec{G}} \right\rangle,\\ \vec{\mathbf{G}}^{\text{n+e}}[n] &:= \left\langle \vec{G}[n],\, \mathbb{N}\times\mathbb{N}_0,\, s^{\text{n+e}}_{\vec{G}} \right\rangle,\\ \vec{\mathbf{G}}^{\text{cycles}}[n] &:= \left\langle \vec{G}[n],\, \mathbb{N}_0,\, s^{\text{cycles}}_{\vec{G}} \right\rangle.\end{aligned}$$

Remark 3.7. Here, $\vec{G}[n]$ is parameterized by the *terms* of the type $\mathbb{N}$, hence it is a *dependent type*. However, just like with the generic types above, we will ignore this detail in the rest of this work and will treat it as any other type.

Example 3.8 (Other types of graphs)**.** Similarly to the previous example we can also define the type of undirected graphs with n nodes as

$$\bar{G}[n] := \mathbb{B}^{\sum_{i=1}^{n} n},$$

where the exponent comes from the fact that an undirected graph can be represented as a triangular matrix.

Both directed and undirected weighted graph types can be constructed by just changing the underlying type from $\mathbb{B}$ to $\mathbb{R}$. Even more, we can distinguish between positively weighted graphs, arbitrarily weighted graphs, acyclic graphs and graphs with only positive cycles:

i. $\vec{G}[n] := \mathbb{B}^{n^2}$: directed unweighted graphs;
ii. $\bar{G}[n] := \mathbb{B}^{\sum_{i=1}^{n} n}$: undirected unweighted graphs;
iii. $\vec{G}_{\geq 0}[n] := \mathbb{R}^{n^2}_{\geq 0}$: directed, positively-weighted graphs;
iv. $\vec{G}_{\oslash}[n] := \{g \in \mathbb{R}^{n^2} \mid \mathsf{cycles}(g) = 0\}$: directed weighted acyclic graphs;
v. $\vec{G}_{\oplus}[n] := \{g \in \mathbb{R}^{n^2} \mid g \text{ has no negative cycles}\}$: directed graphs with no negative cycles.

For each one of the above types, we can define a normed type, in the exact same way as we did for $\vec{G}[n]$ in Example 3.6. One only needs to replace the indicator function for s^{edges} to $\mathbb{1}_{g_i \neq 0}$ for the weighted graphs.

Remark 3.9. In the above example, once again we abuse type theory. The set comprehension notation clearly has no place in type theory. As we try to keep the presentation light, we will allows ourselves to do that. Note, however, that from a type theoretical perspective, $\vec{G}_{\oslash}[n]$ and $\vec{G}_{\oplus}[n]$ are subtypes of $\mathbb{R}^{n^2}$.

Definition 3.10 (The **NTypes** category)**.** We define a category **NTypes**:

i. The objects of **NTypes** are normed types (Definition 3.1).
ii. A morphism between two objects $\mathbf{A} := \langle A, |A|, s_A\rangle$ and $\mathbf{B} := \langle B, |B|, s_B\rangle$ is a tuple $\langle f \colon A \to B, h \colon |A| \to |B|\rangle$, where:
    a) $f \colon A \to B$ is a term of the function type $A \to B$,
    b) $g \colon |A| \to |B|$ is a monotone function (Definition 2.11).
iii. The identity morphism for $\mathbf{A}$ is $\mathrm{id}_{\mathbf{A}} := \langle \mathrm{id}_A, \mathrm{id}_{|A|}\rangle$.

iv. The composition of morphisms is the composition of the constituent functions.

Remark 3.11. The **NTypes** category is what brings life into the normed types. A morphism in **NTypes** tell us how we can convert the term of one type to the term of another type, while at the same time, also handling the (possible) change in size.

Lemma 3.12 (**NTypes** is a monoidal category)**.** *The* **NTypes** *category is a monoidal category when considering the following additional structure:*

- *Tensor product $\otimes$, such that given two objects*

$$\mathbf{A} := \langle A, |A|, s_A \rangle$$

and

$$\mathbf{B} := \langle B, |B|, s_B \rangle$$

we have

$$\mathbf{A} \otimes \mathbf{B} := \langle A \times B, |A| \times |B|, s_A \times s_B \rangle ,$$

with the first product being the product on types (Definition 2.65), the second being Cartesian product (Definition 2.5), and the third being the product on functions (Definition 2.50). The result of applying $\otimes$ to two morphisms $\langle f, g \rangle$ from $\mathbf{A}$ to $\mathbf{B}$ and $\langle h, k \rangle$ from $\mathbf{C}$ to $\mathbf{D}$ is:

$$\langle f, g \rangle \otimes \langle h, k \rangle := \langle f \times h,\ g \times k \rangle .$$

- *Unit object being $\mathbf{I} = \langle I, |I|, s_I \rangle$, where I is the unit type (Definition 2.64), $|I| = \{1\}$ and $s_I(\iota) = 1$.*
- *Left unitor being the pair of morphisms*

$$\begin{aligned} \lambda_{\mathbf{A}} &\colon \mathbf{I} \otimes \mathbf{A} \to \mathbf{A}, & \lambda_{\mathbf{A}} &:= \langle \mathtt{snd},\ \mathtt{snd} \rangle ; \\ \lambda_{\mathbf{A}}^{-1} &\colon \mathbf{A} \to \mathbf{I} \otimes \mathbf{A}, & \lambda_{\mathbf{A}}^{-1} &:= \langle [a \mapsto \langle \iota, a \rangle],\ [a \mapsto \langle 1, a \rangle] \rangle . \end{aligned}$$

- *Right unitor being the pair of morphisms*

$$\begin{aligned} \rho_{\mathbf{A}} &\colon \mathbf{A} \otimes \mathbf{I} \to \mathbf{A}, & \rho_{\mathbf{A}} &:= \langle \mathtt{fst},\ \mathtt{fst} \rangle ; \\ \rho_{\mathbf{A}}^{-1} &\colon \mathbf{A} \to \mathbf{A} \otimes \mathbf{I}, & \rho_{\mathbf{A}}^{-1} &:= \langle [a \mapsto \langle a, \iota \rangle],\ [a \mapsto \langle a, 1 \rangle] \rangle . \end{aligned}$$

- *Associator being the pair of morphisms*

$$\begin{aligned} \alpha_{\mathbf{AB,C}} &\colon (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} \to \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}), & \alpha_{\mathbf{AB,C}} &:= \left\langle \alpha_{AB,C}, \alpha_{|A||B|,|C|} \right\rangle ; \\ \alpha_{\mathbf{A,BC}} &\colon \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}) \to (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C}, & \alpha_{\mathbf{A,BC}} &:= \left\langle \alpha_{A,BC}, \alpha_{|A|,|B||C|} \right\rangle , \end{aligned}$$

where the α association bijections were defined in Definition 2.68.

Proof. In order to show that **NTypes** is a monoidal category, we need to show that the triangle and pentagon diagrams commute.

The triangle identity

$$\begin{tikzcd}
(\mathbf{A}\otimes\mathbf{I})\otimes\mathbf{B} \arrow[rr, "\alpha_{\mathbf{AI,B}}"] \arrow[dr, "\rho_{\mathbf{A}}\otimes\mathrm{id}_{\mathbf{B}}"'] & & \mathbf{A}\otimes(\mathbf{I}\otimes\mathbf{B}) \arrow[dl, "\mathrm{id}_{\mathbf{A}}\otimes\lambda_{\mathbf{B}}"] \\
& \mathbf{A}\otimes\mathbf{B} &
\end{tikzcd} \tag{3.1}$$

is equivalent to the equation:

$$\alpha_{\mathbf{AI,B}} \fatsemi (\mathrm{id}_{\mathbf{A}} \otimes \lambda_{\mathbf{B}}) = \rho_{\mathbf{A}} \otimes \mathrm{id}_{\mathbf{B}}\,.$$

The left-hand side evaluates to

$$\begin{aligned}
&\alpha_{\mathbf{AI,B}} \fatsemi (\mathrm{id}_{\mathbf{A}} \otimes \lambda_{\mathbf{B}}) \\
&= \left\langle \alpha_{AI,B}, \alpha_{|A||I|,|B|} \right\rangle \fatsemi \\
&\quad \left\langle [\langle a, \langle \iota, b\rangle\rangle \mapsto \langle \mathrm{id}_A(a), \mathsf{snd}(\iota, b)\rangle],\ \left[\langle a, \langle 1, b\rangle\rangle \mapsto \left\langle \mathrm{id}_{|A|}(a), \mathsf{snd}(1, b)\right\rangle\right]\right\rangle \\
&= \left\langle \alpha_{AI,B}, \alpha_{|A||I|,|B|} \right\rangle \fatsemi \langle [\langle a, \langle \iota, b\rangle\rangle \mapsto \langle a, b\rangle],\ [\langle a, \langle 1, b\rangle\rangle \mapsto \langle a, b\rangle]\rangle \\
&= \langle [\langle a, \langle \iota, b\rangle\rangle \mapsto \langle a, b\rangle]\,([\langle\langle a', \iota'\rangle, b'\rangle \mapsto \langle a', \langle \iota', b'\rangle\rangle]), \\
&\quad [\langle a, \langle 1, b\rangle\rangle \mapsto \langle a, b\rangle]\,([\langle\langle a', 1'\rangle, b'\rangle \mapsto \langle a', \langle 1', b'\rangle\rangle])\rangle \\
&= \langle [\langle\langle a', \iota'\rangle, b'\rangle \mapsto \langle a', b'\rangle],\ [\langle\langle a', 1'\rangle, b'\rangle \mapsto \langle a', b'\rangle]\rangle\,.
\end{aligned}$$

The right-hand side evaluates to

$$\begin{aligned}
\rho_{\mathbf{A}} \otimes \mathrm{id}_{\mathbf{B}} &= \langle \texttt{fst},\ \texttt{fst}\rangle \otimes \left\langle \mathrm{id}_B, \mathrm{id}_{|B|}\right\rangle \\
&= \langle [\langle a, \iota\rangle \mapsto a],\ [\langle a, 1\rangle \mapsto a]\rangle \otimes \langle [b \mapsto b], [b \mapsto b]\rangle \\
&= \langle [\langle\langle a, \iota\rangle, b\rangle \mapsto \langle a, b\rangle],\ [\langle\langle a, 1\rangle, b\rangle \mapsto \langle a, b\rangle]\rangle\,.
\end{aligned}$$

The two expressions are equivalent, hence the triangle identity holds.

The pentagon identity

$$\begin{tikzcd}
& (\mathbf{A}\otimes\mathbf{B})\otimes(\mathbf{C}\otimes\mathbf{D}) \arrow[dr, "\alpha_{\mathbf{(AB),CD}}"] & \\
((\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C})\otimes\mathbf{D} \arrow[ur, "\alpha_{\mathbf{(AB)C,D}}"] \arrow[d, "\alpha_{\mathbf{AB,C}}\otimes\mathrm{id}_{\mathbf{D}}"] & & \mathbf{A}\otimes(\mathbf{B}\otimes(\mathbf{C}\otimes\mathbf{D})) \\
(\mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}))\otimes\mathbf{D} \arrow[rr, "\alpha_{\mathbf{A(BC),D}}"] & & \mathbf{A}\otimes((\mathbf{B}\otimes\mathbf{C})\otimes\mathbf{D}) \arrow[u, "\mathrm{id}_{\mathbf{A}}\otimes\alpha_{\mathbf{BC,D}}"']
\end{tikzcd} \tag{3.2}$$

is equivalent to the equation

$$\alpha_{\mathbf{(AB)C,D}} \fatsemi \alpha_{\mathbf{AB,(CD)}} = \left(\alpha_{\mathbf{AB,C}} \otimes \mathrm{id}_{\mathbf{D}}\right) \fatsemi \alpha_{\mathbf{A(BC),D}} \fatsemi \left(\mathrm{id}_{\mathbf{A}} \otimes \alpha_{\mathbf{BC,D}}\right).$$

The left-hand side evaluates to

$$\begin{aligned}
&\alpha_{\mathbf{(AB)C,D}} \mathbin{⨟} \alpha_{\mathbf{AB,(CD)}}\\
&= \left\langle \alpha_{(AB)C,D}, \alpha_{(|A||B|)|C|,|D|} \right\rangle \mathbin{⨟} \left\langle \alpha_{AB,(CD)}, \alpha_{|A||B|,(|C||D|)} \right\rangle\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,b\rangle ,\langle c,d\rangle\rangle],\ [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,b\rangle ,\langle c,d\rangle\rangle]\rangle \mathbin{⨟}\\
&\quad \langle [\langle\langle a,b\rangle ,\langle c,d\rangle\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle],\ [\langle\langle a,b\rangle ,\langle c,d\rangle\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle]\rangle\\
&= \langle [\langle\langle a,b\rangle ,\langle c,d\rangle\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle]\ ([\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,b\rangle ,\langle c,d\rangle\rangle]),\\
&\quad\ [\langle\langle a,b\rangle ,\langle c,d\rangle\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle]\ ([\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,b\rangle ,\langle c,d\rangle\rangle])\rangle\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle],\ [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle]\rangle .
\end{aligned}$$

And the right-hand side evaluates to

$$\begin{aligned}
&\left(\alpha_{\mathbf{AB,C}} \otimes \mathrm{id}_{\mathbf{D}}\right) \mathbin{⨟} \alpha_{\mathbf{A(BC),D}} \mathbin{⨟} \left(\mathrm{id}_{\mathbf{A}} \otimes \alpha_{\mathbf{BC,D}}\right)\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle \alpha_{AB,C}\ \langle\langle\langle a,b\rangle ,c\rangle\rangle ,\mathrm{id}_D(d)\rangle],\\
&\quad \left[\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \left\langle \alpha_{|A||B|,|C|}\ \langle\langle\langle a,b\rangle ,c\rangle\rangle ,\mathrm{id}_{|D|}(d)\right\rangle\right]\rangle \mathbin{⨟}\\
&\quad \alpha_{\mathbf{A(BC),D}} \mathbin{⨟} \left(\mathrm{id}_{\mathbf{A}} \otimes \alpha_{\mathbf{BC,D}}\right)\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,\langle b,c\rangle\rangle ,d\rangle],\ [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle\langle a,\langle b,c\rangle\rangle ,d\rangle]\rangle \mathbin{⨟}\\
&\quad \alpha_{\mathbf{A(BC),D}} \mathbin{⨟} \left(\mathrm{id}_{\mathbf{A}} \otimes \alpha_{\mathbf{BC,D}}\right)\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle\langle b,c\rangle ,d\rangle\rangle],\ [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle\langle b,c\rangle ,d\rangle\rangle]\rangle \mathbin{⨟}\\
&\quad \left(\mathrm{id}_{\mathbf{A}} \otimes \alpha_{\mathbf{BC,D}}\right)\\
&= \langle [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle],\ [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto \langle a,\langle b,\langle c,d\rangle\rangle\rangle]\rangle .
\end{aligned}$$

The two expressions are equivalent, hence the pentagon identity also holds and **NTypes** is indeed a monoidal category. □

The **NTypes** category can be thought of "types with something extra". However, we can also drop the "extra" should we wish to do so. Let's first define **Types**, the category without the "extras".

Definition 3.13 (The **Types** category)**.** The **Types** category has types for objects types and function types for hom-sets. In other words, a morphism from type A to type B is a term of the function type $A \to B$. Morphisms composition is defined by function composition. Associativity and identity trivially follow.

Remark 3.14. The **Types** category is also monoidal due to us having well-defined product on types (Definition 2.65) and product of functions (Definition 2.50).

We can formalize "dropping" the extra size information as a forgetful functor:

Definition 3.15 (The И functor)**.** The forgetful functor И: **NTypes** → **Types** maps normed types to their types and morphisms to the respective terms of the function types. In other words:

$$\begin{aligned} \text{И}\colon \qquad & \mathbf{NTypes} \to \mathbf{Types}, \\ & \langle A, |A|, s_A \rangle \mapsto A, \\ & \big\langle f : A \to B, g : |A| \to |B| \big\rangle \mapsto f, \end{aligned}$$

where the first mapping is for objects and the second one for morphisms.

3.2 Procedures

A procedure represents a computational process with a given input type and output type. Procedures do not model such processes explicitly. Rather, they just represent the input type, output type, and the resources needed for computation, without specifying the exact algorithm or implementation. Nevertheless, a procedure is expected to be backed by an actual implementation. One exception to this are the identity procedures which simply return the input, something so trivial that an algorithm or implementation is not even necessary.

We can extend the morphisms in **NTypes** in order to also account for the computation resources needed to execute the mapping represented by the first element of the morphism. We will model the computational resources as the elements of a monoidal poset R. The fact that R has monoidal structure allows tracking the accumulation of resources as we compose procedures.

The resources needed to execute a procedure depend on the complexity of the input. We connect the input size with the computational effort: input terms with small sizes need less resource, input terms with bigger sizes need more resources. Furthermore, we want procedures to also estimate the size of the output. Then, we can know the computational effort of composed procedures *before* we even start computing. This proves quite useful for selecting the most efficient procedure for solving a given problem.

Definition 3.16 (The **Proc**(R) category)**.** Given a monoidal poset $\langle R, \leq_R, 0, + \rangle$ that we interpret as *resources*, we define a category **Proc**(R):

i. The objects of **Proc**(R) are normed types (Definition 3.1).
ii. A morphism (also called *procedure*) between two objects

$$\mathbf{A} := \langle A, |A|, s_A \colon A \to |A| \rangle$$

and

$$\mathbf{B} := \langle B, |B|, s_B \colon B \to |B| \rangle$$

is a triplet

$$\left\langle p\colon A \to B, \tau_p\colon |A| \to |B|, \rho_p\colon |A| \nrightarrow R \right\rangle,$$

where:

- $p\colon A \to B$ is a term of the function type $A \to B$ called *map of the procedure*,
- $\tau_p\colon |A| \to |B|$ is a monotonic function called *translation* which maps sizes in **A** to sizes in **B**;
- $\rho_p\colon |A| \nrightarrow R$ is a feasibility relation (Definition 2.14).

iii. The composition of a morphism

$$\mathbf{p} \coloneqq \left\langle p\colon A \to B, \tau_p\colon |A| \to |B|, \rho_p\colon |A| \nrightarrow R \right\rangle$$

from $\mathbf{A} \coloneqq \langle A, |A|, s_A : A \to |A| \rangle$ to $\mathbf{B} \coloneqq \langle B, |B|, s_B\colon B \to |B| \rangle$ with a morphism

$$\mathbf{q} \coloneqq \left\langle q : B \to C, \tau_q : |B| \to |C|, \rho_q\colon |B| \nrightarrow R \right\rangle$$

from **B** to $\mathbf{C} \coloneqq \langle C, |C|, s_C\colon C \to |C| \rangle$ is given by

$$\mathbf{p} \mathbin{⨟} \mathbf{q} \coloneqq \langle p \mathbin{⨟} q, \tau_p \mathbin{⨟} \tau_q, \rho_p \boxplus_{\tau_p} \rho_q \rangle,$$

where $\rho_p \boxplus_{\tau_p} \rho_q$ is defined as:

$$\begin{aligned} \rho_p \boxplus_{\tau_p} \rho_q\colon |A|^{\mathrm{op}} \times R &\to \texttt{Bool}, \\ \langle a, r \rangle &\mapsto \bigvee_{r_1, r_2 \in R} \rho_p(a, r_1) \wedge \rho_q(\tau_p(a), r_2) \wedge (r_1 + r_2 \leq_R r). \end{aligned}$$

The binary operation $\rho_p \boxplus_{\tau_p} \rho_q$ is called *series sum of ρ_p and ρ_q (with translation τ_p)* and its result is also a feasibility relation (Lemma 3.18). Procedure composition is associative as function composition and series sum are both associative. The second is due Lemma 3.19.

iv. The identity procedure for the normed type **A** is

$$\mathrm{id}_{\mathbf{A}} \coloneqq \langle \mathrm{id}_A, \mathrm{id}_{|A|}, \rho_A^{\mathrm{id}} \rangle,$$

where ρ_A^{id} is the identity feasibility relation

$$\begin{aligned} \rho_A^{\mathrm{id}}\colon |A|^{\mathrm{op}} \times R &\to \texttt{Bool}, \\ \langle a, r \rangle &\mapsto \texttt{T}. \end{aligned}$$

Remark 3.17. Applying $\boxplus_\tau$ makes sense only when τ is a map between the domains of the two operands and when the codomains of the operands are the same monoidal poset, as is always the case in the above definition.

Lemma 3.18. *For any two feasibility relations $\rho_p : |A| \nrightarrow R$ and $\rho_q : |B| \nrightarrow R$, and a translation $\tau_p : |A| \longrightarrow |B|$, their series sum $\rho_p \boxplus_{\tau_p} \rho_q$ is a feasibility relation.*

Proof. It is sufficient to show that whenever $(\rho_p \boxplus_{\tau_p} \rho_q)(a, r) = \mathtt{T}$, then for all $a' \in A$ and $r' \in R$ such that $a' \leq_A a$ and $r \leq_R r'$ we also have

$$(\rho_p \boxplus_{\tau_p} \rho_q)(a', r') = \mathtt{T}.$$

If $(\rho_p \boxplus_{\tau_p} \rho_q)(a, r) = \mathtt{T}$, then there exist $r_1, r_2 \in R$ such that

$$\rho_p(a, r_1) \wedge \rho_q(\tau_p(a), r_2) \wedge (r_1 + r_2 \leq_R r) = \mathtt{T}.$$

This trivially also holds if we replace r with $r' \geq_R r$. Due to the monotonicity of τ_p and the fact that ρ_p and ρ_q are also feasibility relations, it also holds if we replace a with $a' \leq_A a$. □

Lemma 3.19. *Series sum with translation is associative. In other words, given three procedures*

$$\begin{aligned}
\mathbf{p} &= \left\langle p\colon A \longrightarrow B, \tau_p\colon |A| \longrightarrow |B|, \rho_p\colon |A| \nrightarrow R \right\rangle \quad \textit{from } \mathbf{A} \textit{ to } \mathbf{B},\\
\mathbf{q} &= \left\langle q\colon B \longrightarrow C, \tau_q\colon |B| \longrightarrow |C|, \rho_q\colon |B| \nrightarrow R \right\rangle \quad \textit{from } \mathbf{B} \textit{ to } \mathbf{C},\\
\mathbf{r} &= \left\langle r\colon C \longrightarrow D, \tau_r\colon |C| \longrightarrow |D|, \rho_r\colon |C| \nrightarrow R \right\rangle \quad \textit{from } \mathbf{C} \textit{ to } \mathbf{D},
\end{aligned}$$

then $(\rho_p \boxplus_{\tau_p} \rho_q) \boxplus_{(\tau_p ⨾ \tau_q)} \rho_r = \rho_p \boxplus_{\tau_p} (\rho_q \boxplus_{\tau_q} \rho_r)$.

Proof. Writing out the left side we get:

$$\begin{aligned}
&\Big((\rho_p \boxplus_{\tau_p} \rho_q) \boxplus_{(\tau_p ⨾ \tau_q)} \rho_r\Big)(a,r)\\
&= \bigvee_{r_1,r_2 \in R} (\rho_p \boxplus_{\tau_p} \rho_q)(a,r_1) \wedge \rho_r((\tau_p ⨾ \tau_q)(a),r_2) \wedge (r_1+r_2 \leq r)\\
&= \bigvee_{r_1,r_2 \in R} \left(\bigvee_{r_1',r_2' \in R} \rho_p(a,r_1') \wedge \rho_q(\tau_p(a),r_2') \wedge (r_1'+r_2' \leq r_1) \right) \wedge \rho_r((\tau_p ⨾ \tau_q)(a),r_2) \wedge (r_1+r_2 \leq r)\\
&= \bigvee_{r_1,r_2,r_1',r_2' \in R} \rho_p(a,r_1') \wedge \rho_q(\tau_p(a),r_2') \wedge (r_1'+r_2' \leq r_1) \wedge \rho_r((\tau_p ⨾ \tau_q)(a),r_2) \wedge (r_1+r_2 \leq r)\\
&= \bigvee_{r_2,r_1',r_2' \in R} \rho_p(a,r_1') \wedge \rho_q(\tau_p(a),r_2') \wedge \rho_r((\tau_p ⨾ \tau_q)(a),r_2) \wedge (r_1'+r_2'+r_2 \leq r).
\end{aligned}$$

The right hand-side evaluates to the same expression, hence the equality holds. □

Lemma 3.20. *Series sum with the identity feasibility relation and identity translation keeps the other operand unchanged, i.e.*

$$\rho_A^{\mathrm{id}} \boxplus_{\mathrm{id}_{|A|}} \rho_p = \rho_p \quad \textit{and} \quad \rho_p \boxplus_{\mathrm{id}_{|A|}} \rho_A^{\mathrm{id}} = \rho_p.$$

where ρ_p is a feasibility relation $\rho_p\colon |A| \nrightarrow R$.

Proof. Writing out the left side of the first equation we get:

$$\begin{aligned}\left(\rho^{\mathrm{id}}_A \boxplus_{\mathrm{id}_{|A|}} \rho_p\right)(a,r) &= \bigvee_{r_1,r_2\in R} \rho^{\mathrm{id}}_A(a,r_1) \wedge \rho_p(\mathrm{id}_{|A|}(a),r_2) \wedge (r_1+r_2 \leq_R r)\\ &= \bigvee_{r_1,r_2\in R} \rho^{\mathrm{id}}_A(a,r_1) \wedge \rho_p(a,r_2) \wedge (r_1+r_2 \leq_R r).\end{aligned}$$

As by definition $\rho^{\mathrm{id}}_A(a,r_1)$ is $\mathtt{T}$ for any r_1, it is $\mathtt{T}$ for $r_1 = 0$. However, then we have

$$\begin{aligned}\left(\rho^{\mathrm{id}}_A \boxplus_{\mathrm{id}_{|A|}} \rho_p\right)(a,r) &= \bigvee_{r_2\in R} \rho^{\mathrm{id}}_A(a,0) \wedge \rho_p(a,r_2) \wedge (0+r_2 \leq_R r)\\ &= \bigvee_{r_2\in R} \rho_p(a,r_2) \wedge (r_2 \leq_R r)\\ &= \rho_p(a,r),\end{aligned}$$

where the last equality is due the monotonicity of ρ_p. Showing that the second equality in the lemma holds can be done in the same way. □

Lemma 3.21. *For any monoidal poset R, the* **Proc**(R) *category is a monoidal category when considering the following additional structure:*

i. Tensor product $\otimes$, such that given two objects $\mathbf{A} := \langle A, |A|, s_A\rangle$ and $\mathbf{B} := \langle B, |B|, s_B\rangle$ we have

$$\mathbf{A} \otimes \mathbf{B} := \langle A \times B, |A| \times |B|, s_A \times s_B\rangle\,,$$

the same as the tensor product on the objects of **NTypes** *(Lemma 3.12). The result of applying $\otimes$ on two morphisms $\langle p, \tau_p, \rho_p\rangle$ from $\mathbf{A}$ to $\mathbf{B}$ and $\langle q, \tau_q, \rho_q\rangle$ from $\mathbf{C}$ to $\mathbf{D}$ is:*

$$\langle p, \tau_p, \rho_p\rangle \otimes \langle q, \tau_q, \rho_q\rangle := \langle p \times q,\ \tau_p \times \tau_q,\ \rho_p \boxtimes \rho_q\rangle\,,$$

where the parallel sum operator $\boxtimes$ is defined as

$$\begin{aligned}\rho_p \boxtimes \rho_q \colon (|A| \times |C|)^{\mathrm{op}} \times R &\to \mathtt{Bool}\\ \langle (a,c), r\rangle &\mapsto \bigvee_{r_1,r_2\in R} \rho_p(a,r_1) \wedge \rho_q(c,r_2) \wedge (r_1+r_2 \leq_R r).\end{aligned}$$

The product morphism is a valid morphism in **Proc**(R) *due to $\rho_p \boxtimes \rho_q$ being a valid feasibility relation (Lemma 3.23).*

ii. Unit object being $\mathbf{I} = \langle I, |I|, s_I\rangle$, where I is the unit type (Definition 2.64), $|I| = \{1\}$ and $s_I(\iota) = 1$.

iii. Left unitor being the pair of morphisms

$$\begin{aligned}\lambda_{\mathbf{A}} &\colon \mathbf{I} \otimes \mathbf{A} \to \mathbf{A}, & \lambda_{\mathbf{A}} &:= \left\langle \mathsf{snd},\ \mathsf{snd},\ \rho^{\mathrm{id}}_{I\times A}\right\rangle;\\ \lambda^{-1}_{\mathbf{A}} &\colon \mathbf{A} \to \mathbf{I} \otimes \mathbf{A}, & \lambda^{-1}_{\mathbf{A}} &:= \left\langle [a \mapsto \langle \iota, a\rangle],\ [a \mapsto \langle 1, a\rangle],\ \rho^{\mathrm{id}}_A\right\rangle.\end{aligned}$$

iv. Right unitor being the pair of morphisms

$$\rho_{\mathbf{A}} \colon \mathbf{A} \otimes \mathbf{I} \to \mathbf{A}, \quad \rho_{\mathbf{A}} := \left\langle \texttt{fst},\ \texttt{fst}\ \rho^{\mathrm{id}}_{A\times I} \right\rangle;$$
$$\rho^{-1}_{\mathbf{A}} \colon \mathbf{A} \to \mathbf{A} \otimes \mathbf{I}, \quad \rho^{-1}_{\mathbf{A}} := \left\langle [a \mapsto \langle a, \iota\rangle],\ [a \mapsto \langle a, 1\rangle],\ \rho^{\mathrm{id}}_{A} \right\rangle.$$

v. Associator being the pair of morphisms

$$\alpha_{\mathbf{AB,C}} \colon (\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C} \to \mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}), \quad \alpha_{\mathbf{AB,C}} := \left\langle \alpha_{AB,C},\ \alpha_{|A||B|,|C|},\ \rho^{\mathrm{id}}_{(A\times B)\times C} \right\rangle;$$
$$\alpha_{\mathbf{A,BC}} \colon \mathbf{A}\otimes(\mathbf{B}\otimes\mathbf{C}) \to (\mathbf{A}\otimes\mathbf{B})\otimes\mathbf{C}, \quad \alpha_{\mathbf{A,BC}} := \left\langle \alpha_{A,BC},\ \alpha_{|A|,|B||C|},\ \rho^{\mathrm{id}}_{A\times(B\times C)} \right\rangle.$$

Remark 3.22. As before, applying $\boxtimes$ is defined only when the two operands are feasibility relations with codomains being the same monoidal poset, which is always the case in **Proc**(R).

Lemma 3.23. *For any two feasibility relations $\rho_p : |A| \nrightarrow R$ and $\rho_q : |B| \nrightarrow R$, their parallel sum $\rho_p \boxtimes \rho_q \colon |A| \times |B| \nrightarrow R$ is a feasibility relation.*

Proof. It is sufficient to show that whenever $(\rho_p \boxtimes \rho_q)(\langle a, b\rangle, r) = \mathtt{T}$, then for all $a' \in A$, $b' \in B$ and $r' \in R$ such that $a' \leq_A a$, $b' \leq_B b$ and $r \leq_R r'$ we also have $(\rho_p \boxtimes \rho_q)(\langle a', b'\rangle, r') = \mathtt{T}$. If $(\rho_p \boxtimes \rho_q)(\langle a, b\rangle, r) = \mathtt{T}$, then there exist $r_1, r_2 \in R$ such that

$$\rho_p(a, r_1) \wedge \rho_q(b, r_2) \wedge (r_1 + r_2 \leq_R r) = \mathtt{T}.$$

This trivially also holds if we replace r with $r' \geq_R r$. Due to the fact that ρ_p and ρ_q are also feasibility relations, it also holds if we replace a with $a' \leq_A a$ and b with $b' \leq_B b$. □

Lemma 3.24. *Parallel sum is associative.*

Proof of Lemma 3.21. Analogously to the proof of Lemma 3.12 we need to show that the triangle and pentagon diagrams commute. Note that the first two elements of the morphisms in **Proc**(R) are exactly the same as the elements of the morphisms in **NTypes**. Also, the composition of morphisms is defined element-wise. Hence, it suffices to show only that the last elements (the boolean profunctors representing the feasibility relation) result in commuting triangle and pentagon diagrams.

This is trivial to show because all of the terms that appear in these diagrams are ρ^{id} which are $\mathtt{T}$ for any input pair and $\left(\rho^{\mathrm{id}}_A \boxtimes \rho^{\mathrm{id}}_B\right)((a, b), r) = \mathtt{T}$ and any A, B, $a \in A$, $b \in B$, $r \in R$. This can be directly concluded from Item i. and from R being a monoidal poset. □

Remark 3.25. The composition of procedures in **Proc**(R) cannot be done in some cases which at first one might expect to be possible. Consider two procedures

$\mathbf{p} = \langle p : A \to B, \tau_p : |A| \to |B|, \rho_p \rangle$ and $\mathbf{q} = \langle q : B \to C, \tau_q : |B|' \to |C|, \rho_q \rangle$. One can compose their maps p and q without a problem. However, we cannot compose the translations because the codomain $|B|$ of τ_p and the domain $|B|'$ of τ_q differ. On the other hand though, for any $a \in A$ we know the value of $p(a)$ after computing the first function, we can use the $s_{B'}$ size function of the origin of $\mathbf{q}$ to compute the size of the instance in $|B|'$. That would be simply $s_{B'}(p(a))$. So why shouldn't we be able to do the composition?

Recall that the purpose of the translation composition is to be able to obtain an estimate of the resources needed for the composite computation *before* we perform the computation. In the above example, even though we *eventually* can obtain the size of the input for the second procedure, we can do that *only after* we finish the computation of the first procedure. That is why the composition of $\mathbf{p}$ and $\mathbf{q}$ is invalid.

As mentioned before, the objects of **NTypes** and **Proc**(R) are the same. Also, if we take a morphism $\mathbf{p} = \langle p, \tau_p, \rho_p \rangle$ from A to B, ignoring the third element ρ_p (the feasibility relation), we get a morphism in **NTypes**. Therefore, we can define a forgetful functor Ц: **Proc**$(R) \to$ **NTypes** that forgets how much resources each procedure requires while keeping everything else. It collapses all procedures that evaluate the same function with the same translation into a single morphism in **NTypes**.

Definition 3.26 (The Ц functor)**.** The forgetful functor Ц: **Proc**$(R) \to$ **NTypes** maps every object A in Ob(**Proc**(R)) to A in Ob(**NTypes**) and every morphism $\mathbf{p} = \langle p, \tau_p, \rho_p \rangle$ to the morphism $\langle p, \tau_p \rangle$. In other words:

$$\begin{aligned} \text{Ц}: \quad & \mathbf{Proc}(R) \to \mathbf{NTypes}, \\ & \langle A, |A|, s_A \rangle \mapsto \langle A, |A|, s_A \rangle, \\ & \langle p : AtoB, \tau_p : |A| \to |B|, \rho_p : |A| \nrightarrow R \rangle \mapsto \langle p : A \to B, \tau_p : |A| \nrightarrow R \rangle, \end{aligned}$$

where the first mapping is for objects and the second one for morphisms.

Remark 3.27. One can compose the Ц and И functors to obtain a forgetful functor that only preserves the types and function types:

$$\text{Ц} \mathbin{⨾} \text{И}: \mathbf{Proc}(R) \to \mathbf{Types}.$$

Example 3.28 (Matrix multiplication)**.** Let's consider one of the most ubiquitous problem in computer science: matrix multiplication. Take the two monoidal posets $T = \langle \mathbb{R}, \leq_{\mathbb{R}}, 0, + \rangle$, which we will refer to as *time* (e.g. in seconds), and $F = \langle \mathbb{R}, \leq_{\mathbb{R}}, 0, + \rangle$, or *flops* (floating operations per second). We denote by R their product $T \times F$, which by Lemma 2.10 is also a monoidal product.

There is a natural trade-off that we would expect for procedures to have when considering the resources R. If one increases the number of flops a computer performs, the computation time would generally decrease. Conversely, if we increase the computation time, one can do with a computer with lower performance in terms of flops.

However, it is also possible that some algorithms dominate others over some inputs for all values of T and F. This is the situation we'd like to study.

Let's consider three matrix multiplication algorithms for matrices of size n:

(i) The naïve algorithm, which takes $2\,n^3$ operations;

(ii) The algorithm by Strassen (1969, Fact 2), which can compute the product with at most $4.7\,n^{2.807355}$ operations;

(iii) The algorithm by Coppersmith and Winograd (1990), which takes about $C\,n^{2.375477}$ operations , where C is an extremely large constant.

Clearly, the Strassen algorithm dominates the naïve one for large n and for small n the naïve approach has better performance. Without going into details, although the Coppersmith-Winograd algorithm is asymptotically the best, C is so large that it cannot be used for any practical problem (Robinson, 2005). Hence the answer to the question which algorithm is "the best" is "it depends". In this case, it depends on the size of the input matrices.

The respective numbers of operations can be achieved via various time-flops trade-offs. Hence we can define the following feasibility relations:

$$\begin{aligned}\rho^{\text{naïve}}_{\text{MM}}\colon \mathbb{N}^{\text{op}} \times R &\longrightarrow \texttt{Bool},\\ \langle n,(t,f)\rangle &\mapsto \begin{cases}\texttt{T} & \text{if } 2\,n^3 \leq_{\mathbb{R}} tf,\\ \texttt{F} & \text{otherwise;}\end{cases}\end{aligned}$$

$$\begin{aligned}\rho^{\text{Strassen}}_{\text{MM}}\colon \mathbb{N}^{\text{op}} \times R &\longrightarrow \texttt{Bool},\\ \langle n,(t,f)\rangle &\mapsto \begin{cases}\texttt{T} & \text{if } 4.7\,n^{2.807355} \leq_{\mathbb{R}} tf,\\ \texttt{F} & \text{otherwise;}\end{cases}\end{aligned}$$

$$\begin{aligned}\rho^{\text{CW}}_{\text{MM}}\colon \mathbb{N}^{\text{op}} \times R &\longrightarrow \texttt{Bool},\\ \langle n,(t,f)\rangle &\mapsto \begin{cases}\texttt{T} & \text{if } C\,n^{2.375477} \leq_{\mathbb{R}} tf,\\ \texttt{F} & \text{otherwise.}\end{cases}\end{aligned}$$

These can be then three procedures in $\mathbf{Proc}(R)$, each being a morphism from the normed type

$$\langle (\text{SM} \times \text{SM})',\ \mathbb{N},\ S_{\text{SM}\times\text{SM}} : \text{SM} \times \text{SM} \to \mathbb{N}\rangle$$

to the normed type

$$\langle \text{SM},\ \mathbb{N},\ S_{\text{SM}} : \text{SM} \to \mathbb{N}\rangle\ ,$$

where SM is the type of all square matrices, (SM × SM)′ are the pairs of square matrices of equal size, and S_{SM} maps a square matrix to its order. The procedures are then:

$$\begin{aligned} \mathbf{p}^{\text{naïve}}_{\mathrm{MM}} &= \left\langle [\langle A, B\rangle \mapsto AB]\,,\ [\langle a, b\rangle \mapsto a]\,,\ \rho^{\text{naïve}}_{\mathrm{MM}} \right\rangle, \\ \mathbf{p}^{\text{Strassen}}_{\mathrm{MM}} &= \left\langle [\langle A, B\rangle \mapsto AB]\,,\ [\langle a, b\rangle \mapsto a]\,,\ \rho^{\text{Strassen}}_{\mathrm{MM}} \right\rangle, \\ \mathbf{p}^{\text{CW}}_{\mathrm{MM}} &= \left\langle [\langle A, B\rangle \mapsto AB]\,,\ [\langle a, b\rangle \mapsto a]\,,\ \rho^{\text{CW}}_{\mathrm{MM}} \right\rangle. \end{aligned}$$

The three procedures are identical, except for the last elements. Therefore, Ц collapses them onto the same morphism in **NTypes**:

$$Ц\left(\mathbf{p}^{\text{naïve}}_{\mathrm{MM}}\right) = Ц\left(\mathbf{p}^{\text{Strassen}}_{\mathrm{MM}}\right) = Ц\left(\mathbf{p}^{\text{CW}}_{\mathrm{MM}}\right) = \langle [\langle A, B\rangle \mapsto AB]\,,\ [\langle a, b\rangle \mapsto a]\rangle\,.$$

Example 3.29 (Distance product)**.** Given two $n \times n$ matrices A and B with entries respectively a_{ij} and b_{ij}, their distance product $C = A \star B$ is

$$c_{ij} = \min\left\{a_{ik} + b_{kj} \mid k = \{1..n\}\right\}.$$

The distance product is usually applied to the adjacency matrix of a graph. If A is the adjacency matrix of $G = (V, E)$, then $A \star A = A^{\star 2}$ is a matrix whose ij-th entry is the length of shortest path of exactly two legs from i to j. Hence, if $|V| = n$, then $A^{\star} = \min\{A, A^{\star 2}, \dots, A^{\star n}\}$, where the minimization is applied entry-wise, is the matrix containing the all-pairs shortest paths.

It has been shown by Romani (1980) that the complexity of the all-pairs-shortest paths problem is the same as ordinary matrix multiplication (at least for positive integer weights, and for simplicity we will assume that our adjacency matrices can be approximated by positive integer matrices). Without getting into too much details, let's assume that the number of operations needed to obtain all shortest paths and their lengths are $C(2n^2 + M(n))$, where C is a constant and $M(n)$ is the number of operations needed for ordinary matrix multiplication of two matrices of size n (Watanabe, 1981). We will denote the matrix of shortest paths corresponding to the lengths in $A^{\star}$ by $\Sigma^{\star}(A)$.

Therefore, we can define feasibility relations similarly to Example 3.28 which results in the following three procedures for computing $(A^{\star}, \Sigma^{\star}(A))$ for a square matrix A:

$$\begin{aligned} \mathbf{p}^{\text{naïve}}_{\star} &= \left\langle \left[A \mapsto \left\langle A^{\star}, \Sigma^{\star}(A)\right\rangle\right]\,,\ [a \mapsto a]\,,\ \rho^{\text{naïve}}_{\star} \right\rangle, \\ \mathbf{p}^{\text{Strassen}}_{\star} &= \left\langle \left[A \mapsto \left\langle A^{\star}, \Sigma^{\star}(A)\right\rangle\right]\,,\ [a \mapsto a]\,,\ \rho^{\text{Strassen}}_{\star} \right\rangle, \\ \mathbf{p}^{\text{CW}}_{\star} &= \left\langle \left[A \mapsto \left\langle A^{\star}, \Sigma^{\star}(A)\right\rangle\right]\,,\ [a \mapsto a]\,,\ \rho^{\text{CW}}_{\star} \right\rangle. \end{aligned}$$

These procedures are morphisms from the normed type

$$\langle \mathrm{SM},\ \mathbb{N},\ S_{\mathrm{SM}} : \mathrm{SM} \to \mathbb{N}\rangle$$

to the normed type

$$\langle \mathrm{SM} \times \mathrm{PM},\ \mathbb{N},\ S_{\mathrm{SM}\times\mathrm{PM}} : \mathrm{SM} \times \mathrm{PM} \to \mathbb{N} \rangle\ ,$$

where SM is the type of all square matrices, PM is the type of all square matrices with entries paths, and $S_{\mathrm{SM}\times\mathrm{PM}} : \mathrm{SM} \times \mathrm{PM} \to \mathbb{N}$ maps a square matrix and a path matrix to their order. Again, Ц collapses these three procedures onto the same morphism in **NTypes**:

$$Ц\left(\mathbf{p}_{\star}^{\text{naïve}}\right) = Ц\left(\mathbf{p}_{\star}^{\text{Strassen}}\right) = Ц\left(\mathbf{p}_{\star}^{\text{CW}}\right) = \left\langle \left[A \mapsto \left\langle A^{\star}, \Sigma^{\star}(A)\right\rangle\right],\ [a \mapsto a]\right\rangle .$$

Chapter 4

Relationships between problems and solutions

One of the main goals of this thesis is to formally study problems and solutions. However, in order to do that we need to first explicitly define what we mean by “problems“ and “solutions”. We start this chapter by looking at how the notion of a problem can be formalized and we will show that problems can be represented as the category **Prob**. In the following chapters we will also see how this notion can be further generalized and abstracted in order to encompass an even larger array of applications.

We then turn our attention to the notion of “solution”. The procedures which were introduced in Section 3.2 will play a key role. Procedures will be the substance that *can* solve a problem. However, our intuition calls that not every procedure can solve a given problem. Hence, some procedures would be solutions for the problem, and some not. We will look in the formal conditions in Section 4.2.

Next, we introduce **Lagado**: our first example of a compositional computational system. It is a category which combines the notions of problems, procedures, solutions, as well as the notion of problem reduction under one roof.

Finally, we show that the ways that problems and procedures interact in **Lagado** resemble the structure of a twisted category. However, there are some key differences, so we define *heteromorphic twisted arrow categories* which generalize twisted categories. Finally we show that **Lagado** is a prime example of a heteromorphic twisted arrow category.

Throughout the chapter the running example of shortest path search on various types of graphs will be further developed and its connections to some of the other problems that we introduced before will be illuminated.

4.1 Problems

Intuitively when we think of problems we think of something that is in need for an answer but is currently lacking one. What we sometimes miss to realize is that many problems can have statements which are parametrized. For example, the class of problems of summing two real numbers is very similar, regardless of the particular choice of the two numbers and the regardless of the answers being different. It is this type of structure that we attempt to capture in our formal definition of a problem.

Definition 4.1 (Problem)**.** A *problem* with a *statement type* being a normed type $\mathbf{T} = \langle T, |T|, s_T \rangle$ and an *answer type* being a normed type $\mathbf{A} = \langle A, |A|, s_A \rangle$ is a relation between T and A, that is a function $\pi\colon T \times A \to \texttt{Bool}$. We will denote such a problem by $\pi\colon \mathbf{T} \to \mathbf{A}$. The problem π can also be identified with the set

$$\Re_\pi := \{\langle t, a \rangle \mid t \in T, a \in A, \pi(t, a) = \texttt{T}\} \subseteq T \times A,$$

which is the more customary way of denoting a binary relation.

Example 4.2 (Shortest path problems)**.** Let's see how some of the classical flavours of the shortest path problem fit in this framework. First, recall from Example 3.6 and Example 3.8 that we can represent various types of graphs with matrices.

We can represent a path in a graph of type $G[n]$ (any of the types in Example 3.8 works) with a list of integers between 1 and n for each node in the path. We'd also like to keep some information about the length of each segment in the path. We can do that by combining every node in the path with the weight of the edge departing from it (and taking a 0 weight for the last node). In the case of an unweighted graph, we'd just assign 1 to all nodes but the last one. Hence, a path type could look like this:

$$\mathsf{P}[n] := \mathsf{List}\left[\mathbb{N}_1^n \times \mathbb{R}\right], \ \forall n \in \mathbb{N},\ \mathbb{N}_l^u := \{n \in \mathbb{N} \mid l \leq n \leq u\}.$$

And $P[n]$ can be converted into the normed type $\mathbf{P}[n]$ by using the construction in Example 3.3. We also define the function $\mathsf{length}(p)$ which computes the length of a path recursively as:

$$\begin{aligned} \mathsf{length}\colon \left\langle p\colon P[n] \right\rangle &\to \mathbb{R}, \\ \langle\rangle &\mapsto 0, \\ \left\langle \langle n, l \rangle, p' \right\rangle &\mapsto l + \mathsf{length}(p'). \end{aligned}$$

Additionally, when we specify a shortest path problem we also need to specify the origin and target nodes. This can again be done by the index of the node in the adjacency matrix. Hence, we define the type $\mathsf{OT}[n] := \mathbb{N}_1^n \times \mathbb{N}_1^n$, and its normed version $\mathbf{OT}[n] := \langle \mathsf{OT}[n], |I|, s_I \rangle$, where s_I maps any element of $\mathsf{OT}[n]$ to 1, the only element of $|I|$.

Now, we can define shortest path problem formulations for the various graph types in Example 3.8. We will only consider the normed graph types with size the number of nodes and edges. Therefore, we can define the following problems:

- $\vec{\sigma}[n]$, the *shortest path problem on a directed unweighted graph of size n*, has a statement being the type $\vec{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n]$, an answer being the type $\mathbf{P}[n]$, and $\vec{\sigma}[n]((g,(o,t)),p) = \mathtt{T}$ iff p is a valid path in g and $\mathsf{length}(p)$ is the minimal length of any path in g from o to t;
- $\bar{\sigma}[n]$, the *shortest path problem on an undirected unweighted graph of size n*, has a statement type $\vec{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n]$, an answer type $\mathbf{P}[n]$, and $\bar{\sigma}[n]((g,(o,t)),p) = \mathtt{T}$ iff p is a valid path in g and $\mathsf{length}(p)$ is the minimal length of any path in g from o to t;
- $\vec{\sigma}_{\geq 0}[n]$, the *shortest path problem on a directed positively-weighted graph of size n*, has a statement type $\vec{\mathbf{G}}_{\geq 0}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n]$, an answer type $\mathbf{P}[n]$, and $\vec{\sigma}_{\geq 0}[n]((g,(o,t)),p) = \mathtt{T}$ iff p is a valid path in g and $\mathsf{length}(p)$ is the minimal length of any path in g from o to t;
- $\vec{\sigma}_{\oslash}[n]$, the *shortest path problem on a directed weighted acyclic graph of size n*, has a statement type $\vec{\mathbf{G}}_{\oslash}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n]$, an answer type $\mathbf{P}[n]$, and $\vec{\sigma}_{\oslash}[n]((g,(o,t)),p) = \mathtt{T}$ iff p is a valid path in g and $\mathsf{length}(p)$ is the minimal length of any path in g from o to t;
- $\vec{\sigma}_{\oplus}[n]$, the *shortest path problem on a directed weighted graph of size n with no negative cycles*, has a statement type $\vec{\mathbf{G}}_{\oplus}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n]$, an answer type $\mathbf{P}[n]$, and $\vec{\sigma}_{\oplus}[n]((g,(o,t)),p) = \mathtt{T}$ iff p is a valid path in g and $\mathsf{length}(p)$ is the minimal length of any path in g from o to t.

These problems are also good examples of problems which can have multiple solutions. In a graph there might be more than one path that has the minimum length. Also, if the graph contains two disconnected components, then there might be also problems with no solutions. That is why the relations $\Re_\sigma$, which these problems define are not necessarily functions. In this setting, we are also implicitly assuming that we are satisfied with *any* of the shortest paths, should there be more than one, and we do not prefer one over another.

Definition 4.3 (The **Prob** category)**.** The problem category **Prob** has:

i. objects which are normed types;
ii. morphisms $\mathbf{A} \to \mathbf{B}$ which are problems with a statement type $\mathbf{A}$ and an answer type $\mathbf{B}$;

iii. composition of two problems $\pi_1 : \mathbf{A} \to \mathbf{B}$ and $\pi_2 : \mathbf{B} \to \mathbf{C}$ defined as:

$$(\pi_1 \mathbin{⨾} \pi_2)(a, c) \coloneqq \bigvee_{b \in B} \pi_1(a, b) \wedge \pi_2(b, c).$$

iv. identity morphism defined as:

$$\begin{aligned} \mathrm{id}_{\mathbf{A}} : A \times A &\to \mathtt{Bool} \\ \langle a_1, a_2 \rangle &\mapsto \begin{cases} \mathtt{T} & \text{if } a_1 = a_2, \\ \mathtt{F} & \text{otherwise.} \end{cases} \end{aligned}$$

Definition 4.4 (The Б functor)**.** Assuming that the terms of the type of any normed type in **Prob** form a set, we define the forgetful functor Б : **Prob** $\to$ **Rel** to map every object $\mathbf{A} = \langle A, |A|, s_A \rangle$ in $\mathsf{Ob}(\textbf{Prob})$ to the set A of its term in **Rel** (Definition 2.40) and every morphism $\pi : A \times B \to \mathtt{Bool}$ to the relation $\Re_\pi$.

4.2 Solutions

In the introduction of this chapter we hinted at solutions being some sort of procedures. This is because we want computable means of obtaining answers to particular statements of a problem.

However, not every procedure is a solution to a given problem. There should be some consistency between the statement-answer map defined by the problem and between the input-output map of the procedure. Here we provide one formalization of this "consistency" that we find particularly intuitive. However, this notion can be further abstracted and that would be a major theme in some of the following chapters.

Definition 4.5 (Solution)**.** We say that a procedure $\mathbf{p} = \langle p, \tau_p, \rho_p \rangle$ from the normed type $\mathbf{A}$ to the normed type $\mathbf{B}$ *solves* a problem $\pi : \mathbf{A} \to \mathbf{B}$ iff

$$\pi(a, p(a)) = \mathtt{T}, \forall a : A.$$

We also say that *the procedure* **p** *is a solution of the problem* π.

Example 4.6 (Shortest path solutions)**.** Let's consider two of the most popular shortest path search algorithms: the one due to Dijkstra (1959), and the one proposed independently by Shimbel (1954), Bellman (1958), Ford (1956), and Moore (1959), commonly called the *Bellman-Ford* algorithm. The Dijkstra algorithm, with small modifications, can be applied to the undirected ($\bar{\sigma}[n]$) and the directed acyclic ($\vec{\sigma}_\oslash[n]$) problems. The Bellman-Ford algorithm, again with a bit of customization, can be used to solve any of the problems in Example 4.2. While the Bellman-Ford algorithm is more versatile, it also has worse computational complexity. The average complexity of the Dijkstra algorithm is

$$\mathcal{O}\left(|E| + |V| \log \frac{|E|}{|V|} \log |V|\right)$$

and of the Bellman-Ford algorithm is

$$\mathcal{O}(|V||E|),$$

where $|V|$ and $|E|$ are the number of vertices and edges respectively (Mehlhorn and Sanders, 2008, Ch. 10.4), (Cormen et al., 2009, Ch. 24.1).

We can therefore define the set of procedures for every pair of shortest path problem and algorithm that can solve it:

- $\vec{\mathbf{s}}^{\mathrm{D}}[n] = \left\langle \vec{f}^{\mathrm{D}}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{D}} \right\rangle$, the solution of a shortest path problem on a directed unweighted graph of size n with the Dijkstra algorithm;
- $\vec{\mathbf{s}}^{\mathrm{BF}}[n] = \left\langle \vec{f}^{\mathrm{BF}}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{BF}} \right\rangle$, the solution of a shortest path problem on a directed unweighted graph of size n with the Bellman-Ford algorithm;
- $\bar{\mathbf{s}}^{\mathrm{D}}[n] = \langle \bar{f}^{\mathrm{D}}[n] \colon \bar{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{D}} \rangle$, the solution of a shortest path problem on an undirected unweighted graph of size n with the Dijkstra algorithm;
- $\bar{\mathbf{s}}^{\mathrm{BF}}[n] = \langle \bar{f}^{\mathrm{BF}}[n] \colon \bar{\mathbf{G}}^{\mathrm{n+e}}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n]\ \rho^{\mathrm{BF}} \rangle$, the solution of a shortest path problem on an undirected unweighted graph of size n with the Bellman-Ford algorithm;
- $\vec{\mathbf{s}}^{\mathrm{D}}_{\geq 0}[n] = \left\langle \vec{f}^{\mathrm{D}}_{\geq 0}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}_{\geq 0}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{D}} \right\rangle$, the solution of a shortest path problem on a directed positively-weighted graph of size n with the Dijkstra algorithm;
- $\vec{\mathbf{s}}^{\mathrm{BF}}_{\geq 0}[n] = \left\langle \vec{f}^{\mathrm{BF}}_{\geq 0}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}_{\geq 0}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{BF}} \right\rangle$, the solution of a shortest path problem on a directed positively-weighted graph of size n with the Bellman-Ford algorithm;
- $\vec{\mathbf{s}}^{\mathrm{BF}}_{\oslash}[n] = \left\langle \vec{f}^{\mathrm{BF}}_{\oslash}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}_{\oslash}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{BF}} \right\rangle$, the solution of a shortest path problem on a directed weighted acyclic graph of size n with the Bellman-Ford algorithm;
- $\vec{\mathbf{s}}^{\mathrm{BF}}_{\oplus}[n] = \left\langle \vec{f}^{\mathrm{BF}}_{\oplus}[n] \colon \vec{\mathbf{G}}^{\mathrm{n+e}}_{\oplus}[n] \otimes \mathbf{OT}[n] \to \mathbf{P}[n],\ \tau_{\mathbf{s}}[n],\ \rho^{\mathrm{BF}} \right\rangle$, the solution of a shortest path problem on a directed weighted graph of size n with no negative cycles with the Bellman-Ford algorithm.

We define the translation function used in all the shortest path procedures as:

$$\tau_{\mathbf{s}}[n] \coloneqq \left[\left\langle g, \langle o, t \rangle \right\rangle \mapsto n\right].$$

We made the choice that regardless of the particular problem statement, we will always keep the size of the output as n, the number of nodes. This is the longest possible shortest path in any graph. However, chances are that given a fixed graph with n nodes the shortest path in it won't be of length n. The problem is that we cannot know that *before* we solve the problem, and we choose to have the worst case scenario. Nevertheless, one can also fix 0 or any other natural number they see fit.

Let's also take a look at the resources needed. Our measure of resources will again be time and flops, as in Example 3.28. In the current setting we unfortunately do not know the precise amount of floating point operations needed to obtain an answer, we only have the asymptotic complexity. Hence, we will only use the asymptotic complexity, even though it might be vastly inaccurate for small graphs:

$$\begin{aligned}
\rho^{\mathrm{D}}\colon \quad & (\mathbb{N}\times\mathbb{N}_0)\times|I| \to T\times F,\\
& \big\langle\langle\langle n,e\rangle,\iota\rangle,\langle t,f\rangle\big\rangle \mapsto C^{\mathrm{D}}\left(e+n\log\frac{e}{n}\log n\right);\\
\rho^{\mathrm{BF}}\colon \quad & (\mathbb{N}\times\mathbb{N}_0)\times|I| \to T\times F,\\
& \big\langle\langle\langle n,e\rangle,\iota\rangle,\langle t,f\rangle\big\rangle \mapsto C^{\mathrm{BF}}nv,
\end{aligned}$$

where C^D and C^{BF} are some (carefully chosen) constants.

4.3 Lagado

The **Lagado** category is one of the most important constructions in this thesis. It intuitively wraps the notion of problems (as its objects) and the notion of procedures (as components of its morphisms).

A morphism in **Lagado** from one problem to another represents the reduction of the first problem to the second. This is done by a pair of procedures. The first performs some computation that transforms the original problem statement into a statement for the second problem. The second procedure performs computation that transforms the answer of the second problem back to the answer type of the original problem. The only thing missing is the means to solve the second problem. But we will see that solving a problem as defined above actually corresponds to morphisms to an identity problem in the setting of **Lagado**.

Furthermore, **Lagado** also maintains the notion of resources needed for computation that it inherits from the **Proc** category (Definition 3.16).

Definition 4.7 (The **Lagado**(R)[1] category). Assuming we are given a monoidal poset $\langle R, \leq_R, 0, +\rangle$ that we interpret as *resources*, we define a category **Lagado**(R) such that:

i. objects are problems, i.e. the elements of the hom-sets of **Prob**;
ii. morphisms from a problem $\pi_1 \colon \mathbf{P}_1 \to \mathbf{S}_1$ to a problem $\pi_2 \colon \mathbf{P}_2 \to \mathbf{S}_2$ are pairs of procedures $\langle \mathbf{p}, \mathbf{s}\rangle$, such that $\mathbf{p} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathbf{P}_1, \mathbf{P}_2)$, $\mathbf{s} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathbf{S}_2, \mathbf{S}_1)$, and for any $\mathbf{h} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathbf{P}_2, \mathbf{S}_2)$ which is a solution of π_2 it holds that $\mathbf{p} \fatsemi \mathbf{h} \fatsemi \mathbf{s} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathbf{P}_1, \mathbf{S}_1)$ is a solution of π_1;
iii. the identity morphism for a problem $\pi \colon \mathbf{P} \to \mathbf{S}$ is the pair of procedures $\mathrm{id}_\pi^{\mathbf{Lagado}(R)} = \left\langle \mathrm{id}_{\mathbf{P}}^{\mathbf{Proc}}, \mathrm{id}_{\mathbf{S}}^{\mathbf{Proc}} \right\rangle$ (the superscript indicates the category in which this is an identity morphism);
iv. given three problems $\pi_1 \colon \mathbf{P}_1 \to \mathbf{S}_1$, $\pi_2 \colon \mathbf{P}_2 \to \mathbf{S}_2$, $\pi_3 \colon \mathbf{P}_3 \to \mathbf{S}_3$ and two morphisms $\langle \mathbf{p_a}, \mathbf{s_a}\rangle$ from π_1 to π_2 and $\langle \mathbf{p_b}, \mathbf{s_b}\rangle$ from π_2 to π_3, the composition of the morphisms is:

$$\langle \mathbf{p_a}, \mathbf{s_a}\rangle \fatsemi \langle \mathbf{p_b}, \mathbf{s_b}\rangle \coloneqq \left\langle \mathbf{p_a} \fatsemi \mathbf{p_b}, \mathbf{s_b} \fatsemi \mathbf{s_a} \right\rangle .$$

Lemma 4.8. Lagado(R) *is indeed a category.*

Proof. We need to demonstrate that the identity and associativity properties hold.

Identity is straight-forward to show. Take two objects $\pi_1 \colon \mathbf{P}_1 \to \mathbf{S}_1$ and $\pi_2 \colon \mathbf{P}_2 \to \mathbf{S}_2$, together with a morphism between them $\langle \mathbf{p}, \mathbf{s}\rangle$, with

$$\begin{aligned} \mathbf{p} &= \left\langle p \colon P_1 \to P_2,\ \tau_p \colon |P_1| \to |P_2|,\ \rho_p \colon |P_1| \twoheadrightarrow R \right\rangle , \\ \mathbf{s} &= \left\langle s \colon S_2 \to S_1,\ \tau_s \colon |S_2| \to |S_1|,\ \rho_s \colon |S_2| \twoheadrightarrow R \right\rangle . \end{aligned}$$

Then,

$$\begin{aligned} &\mathrm{id}_{\pi_1}^{\mathbf{Lagado}(R)} \fatsemi \langle \mathbf{p}, \mathbf{s}\rangle \\ &= \left\langle \mathrm{id}_{\mathbf{P}_1}^{\mathbf{Proc}}, \mathrm{id}_{\mathbf{S}_1}^{\mathbf{Proc}} \right\rangle \fatsemi \langle \mathbf{p}, \mathbf{s}\rangle \\ &= \left\langle \mathrm{id}_{\mathbf{P}_1}^{\mathbf{Proc}} \fatsemi \mathbf{p},\ \mathbf{s} \fatsemi \mathrm{id}_{\mathbf{S}_1}^{\mathbf{Proc}} \right\rangle \\ &= \left\langle \left\langle \mathrm{id}_{P_1}, \mathrm{id}_{|P_1|}, \rho_{P_1}^{\mathrm{id}} \right\rangle \fatsemi \left\langle p, \tau_p, \rho_p \right\rangle,\ \left\langle s, \tau_s, \rho_s \right\rangle \fatsemi \left\langle \mathrm{id}_{S_1}, \mathrm{id}_{|S_1|}, \rho_{S_1}^{\mathrm{id}} \right\rangle \right\rangle \\ &= \left\langle \left\langle p, \tau_p, \rho_p \right\rangle,\ \left\langle s, \tau_s, \rho_s \right\rangle \right\rangle \\ &= \langle \mathbf{p}, \mathbf{s}\rangle , \end{aligned}$$

[1]Named after Lagado, the capital of the island Balnibarbi in *Gulliver's Travels* by Jonathan Swift (1726). The Academy of Projectors in Lagado has built The Engine which resembles a modern computer and is believed to be the earliest mention of such a device (Weiss, 1985).

by using the properties of function composition with identity and Lemma 3.20. And analogously:

$$\begin{aligned}
&\langle \mathbf{p}, \mathbf{s} \rangle \mathbin{⨾} \mathrm{id}^{\mathbf{Lagado}(R)}_{\pi_2} \\
&= \langle \mathbf{p}, \mathbf{s} \rangle \mathbin{⨾} \left\langle \mathrm{id}^{\mathbf{Proc}}_{\mathbf{P}_2}, \mathrm{id}^{\mathbf{Proc}}_{\mathbf{S}_2} \right\rangle \\
&= \left\langle \mathbf{p} \mathbin{⨾} \mathrm{id}^{\mathbf{Proc}}_{\mathbf{P}_2},\ \mathrm{id}^{\mathbf{Proc}}_{\mathbf{S}_2} \mathbin{⨾} \mathbf{s} \right\rangle \\
&= \left\langle \left\langle p, \tau_p, \rho_p \right\rangle \mathbin{⨾} \left\langle \mathrm{id}_{P_2}, \mathrm{id}_{|P_2|}, \rho^{\mathrm{id}}_{P_2} \right\rangle,\ \left\langle \mathrm{id}_{S_2}, \mathrm{id}_{|S_2|}, \rho^{\mathrm{id}}_{S_2} \right\rangle \mathbin{⨾} \left\langle s, \tau_s, \rho_s \right\rangle \right\rangle \\
&= \left\langle \left\langle p, \tau_p, \rho_p \right\rangle,\ \left\langle s, \tau_s, \rho_s \right\rangle \right\rangle \\
&= \langle \mathbf{p}, \mathbf{s} \rangle .
\end{aligned}$$

Associativity follows directly from the fact that we compose the morphisms element-wise, and due to function composition and series sum both being associative as shown in Lemma 3.19. □

Remark 4.9. The definition of the **Lagado**(R) category shows us how we can "convert" one problem into another, while accounting for the cost of the conversion, i.e. the cost of executing the procedures that convert a statement for one problem into one for another and the solution of the second problem into a solution of the original problem. This definition is broad enough to encompass various different settings. To provide some intuition, consider the following few (non-exhaustive) cases:

- We can convert a problem instance into another representation almost for free and the new representation allows us to perform the operation much more efficiently. For example, imagine that you can rewrite your problem from iterating over lists to matrix multiplications. In many cases, hardware is optimized to calculate matrix multiplications much more efficiently, hence that can result in less resources used. After the computation is done you might want to again change the representation of your data to the original structure, which would correspond to the second procedure in the morphism pair.
- Maybe your problem can be solved by three sub-procedures. Imagine we have the problem of calculating the Euclidean distance between two vectors $x, y \in \mathbb{R}^n$ after they have been linearly transformed by the matrix $A \in \mathbb{R}^{m \times n}$, i.e $\|Ax - Ay\|_2$. Then we can have a pair of procedures $\langle \mathbf{p}, \mathbf{s} \rangle$ which is a morphism from our original problem to the problem of computing the difference of two vectors. Here, $\mathbf{p}$ is the procedure that takes two vectors and a matrix, applies the matrix to each vector and returns the resultant pair of vectors. Then, $\mathbf{s}$ is the procedure that takes a vector and computes its Euclidean norm. The gap between the two, i.e. computing the difference between the vectors, is to be performed by any solution of the problem of computing the difference of two vectors,

which problem is the target of our morphism. This is a simple example in which the procedures not only change the shape of the input and the output but also perform partial computation towards solving the problem.

- The first point above only changed the representation of the data, the second also performed some computation. It is only natural to ask, is it possible for a morphism to perform the *full* computation, i.e. to be a solution for the problem. And naturally, the answer is *yes* and the next lemma shows how this can be done.

Lemma 4.10. *Given a problem $\pi\colon \mathsf{P} \to \mathsf{S} \in \mathrm{Ob}(\mathbf{Lagado}(R))$, and a solution $\mathbf{p}$ of it (as per Definition 4.5), then there are two morphisms in $\mathbf{Lagado}(R)$ from π to the identity problems on P and S:*

$$\begin{tikzcd}
 & \mathrm{id}^{\mathbf{Prob}}_{\mathsf{P}}\colon \mathsf{P} \to \mathsf{P} \\
\pi\colon \mathsf{P} \to \mathsf{S} \arrow[ur, "\langle \mathrm{id}^{\mathbf{Proc}}_{\mathsf{P}}{,}\mathbf{p} \rangle"] \arrow[dr, "\langle \mathbf{p}{,}\mathrm{id}^{\mathbf{Proc}}_{\mathsf{S}} \rangle"'] & \\
 & \mathrm{id}^{\mathbf{Prob}}_{\mathsf{S}}\colon \mathsf{S} \to \mathsf{S}
\end{tikzcd} \tag{4.1}$$

Remark 4.11. It is easy to see how the diagram in Equation (4.1) represents the solution $\mathbf{p}$ of the problem π. One doesn't really need a "solution" for an identity problem: the problem statement instance is the solution instance. Therefore, we can intuitively think of "solving a problem" and "reducing a problem to an identity problem" as being the exact same thing. This is formally formulated in Definition 4.12 and the connection between the solving a problem and reducing it to identity is shown in Lemma 4.13.

Note also that every solution results in two morphisms. However, in practice it makes little difference which one we pick. As the resources required by an identity procedure can be assumed negligible, in both cases the real cost comes only from the resources required by $\mathbf{p}$.

Definition 4.12 (Solution of a problem in $\mathbf{Lagado}(R)$)**.** Given any problem $\pi\colon \mathsf{P} \to \mathsf{S} \in \mathrm{Ob}(\mathbf{Lagado}(R))$, any morphism $\langle \mathbf{p}, \mathbf{s} \rangle$ from π to $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{Q}}$ for any $\mathsf{Q} \in \mathbf{NTypes}$ is called *a solution of π in* $\mathbf{Lagado}(R)$.

Lemma 4.13. *Given a problem $\pi\colon \mathsf{P} \to \mathsf{S} \in \mathrm{Ob}(\mathbf{Lagado}(R))$ and a morphism $\langle \mathbf{p}, \mathbf{s} \rangle$ that is a solution of π in $\mathbf{Lagado}(R)$, it holds that $\mathbf{p} \mathbin{⨟_{\mathbf{Proc}}} \mathbf{s}$ is a solution of π as per Definition 4.5.*

Proof. For the procedure $\mathbf{a} = \mathbf{p} \mathbin{⨟_{\mathbf{Proc}}} \mathbf{s}$ to be a solution of the problem $\pi\colon \mathsf{P} \to \mathsf{S}$ we need that the origin and target normed types of $\mathbf{a}$ to be respectively P and

S and that for any $p\colon P$ we have $\pi(p, a(p)) = \mathtt{T}$. From Definition 4.7 we know that $\mathbf{p}\colon \mathsf{P} \longrightarrow \mathsf{Q}$ and $\mathbf{s}\colon \mathsf{Q} \longrightarrow \mathsf{S}$. Then, the first condition holds because the origin of $\mathbf{a}$ is the origin of $\mathbf{p}$ and its target is the target of $\mathbf{s}$ (due to the definition of composition of procedures in Definition 3.16).

For the second condition, we know that $\mathbf{p} \mathbin{⨾_{\mathbf{Proc}}} \mathbf{q} \mathbin{⨾_{\mathbf{Proc}}} \mathbf{s}$ must be a solution of π if $\mathbf{q}$ is a solution of $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{Q}}$. Given that the identity procedure $\mathrm{id}^{\mathbf{Proc}}_{\mathsf{Q}}$ is a solution of $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{Q}}$ and that $\mathbf{p} \mathbin{⨾_{\mathbf{Proc}}} \mathrm{id}^{\mathbf{Proc}}_{\mathsf{Q}} = \mathbf{p}$, we have that $\mathbf{a} = \mathbf{p} \mathbin{⨾_{\mathbf{Proc}}} \mathbf{s}$ is a solution of the problem π, hence $\pi(p, a(p)) = \mathtt{T}$. □

People often find themselves in situations where they need to solve more than one problem, hence it makes sense for the **Lagado**(R) category to allow us to solve multiple problems at the same time. In fact, that is enabled by defining a monoidal product on it:

Lemma 4.14 (**Lagado**(R) is a monoidal category)**.** *For any monoidal poset R, the* **Lagado**(R) *category is a monoidal category when considering the following structure:*

- *Tensor product $\|$ (read parallel), such that given two problems $\pi_1\colon \mathsf{P}_1 \longrightarrow \mathsf{S}_1$ and $\pi_2\colon \mathsf{P}_2 \longrightarrow \mathsf{S}_2$ we have*

$$\pi_1 \parallel \pi_2\colon \mathsf{P}_1 \otimes_{\mathbf{NTypes}} \mathsf{P}_2 \longrightarrow \mathsf{S}_1 \otimes_{\mathbf{NTypes}} \mathsf{S}_2,$$

that is the relation between $P_1 \times P_2$ and $S_1 \times S_2$:

$$\begin{aligned}\pi_1 \parallel \pi_2\colon\ & (P_1 \times P_2) \times (S_1 \times S_2) \longrightarrow \mathtt{Bool},\\ & \big\langle \langle p_1, p_2 \rangle, \langle s_1, s_2 \rangle \big\rangle \mapsto \pi_1(p_1, s_1) \wedge \pi_2(p_2, s_2).\end{aligned}$$

Take two morphisms $\langle \mathbf{p}', \mathbf{s}' \rangle$ and $\langle \mathbf{p}'', \mathbf{s}'' \rangle$, such that

$$\begin{aligned}\mathbf{p}' &\in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{P}'_1, \mathsf{P}'_2),\\ \mathbf{s}' &\in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{S}'_2, \mathsf{S}'_1),\\ \mathbf{p}'' &\in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{P}''_1, \mathsf{P}''_2),\\ \mathbf{s}'' &\in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{S}''_2, \mathsf{S}''_1).\end{aligned}$$

The result of applying $\|$ to them is:

$$\langle \mathbf{p}', \mathbf{s}' \rangle \parallel \langle \mathbf{p}'', \mathbf{s}'' \rangle := \left\langle \mathbf{p}' \otimes_{\mathbf{Proc}(R)} \mathbf{p}'',\ \mathbf{s}' \otimes_{\mathbf{Proc}(R)} \mathbf{s}'' \right\rangle.$$

- *Unit problem is the identity problem on the identity normed type $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{I}}$, where I is as used in Lemmas 3.12 and 3.21 and the identity problem is the identity morphism in* **Prob** *(Definition 4.3).*
- *Given a problem $\pi : \mathsf{P} \longrightarrow \mathsf{S}$, its left unitor is the pair of morphisms:*

$$\begin{aligned}\lambda^{\mathbf{Lagado}(R)}_{\pi}\colon \mathrm{id}^{\mathbf{Prob}}_{\mathsf{I}} \parallel \pi \longrightarrow \pi, &\quad \lambda^{\mathbf{Lagado}(R)}_{\pi} := \left\langle \lambda^{\mathbf{Proc}(R)}_{\mathsf{P}},\ \lambda^{-1\ \mathbf{Proc}(R)}_{\mathsf{S}} \right\rangle;\\ \lambda^{-1\ \mathbf{Lagado}(R)}_{\pi}\colon \pi \longrightarrow \mathrm{id}^{\mathbf{Prob}}_{\mathsf{I}} \parallel \pi, &\quad \lambda^{-1\ \mathbf{Lagado}(R)}_{\pi} := \left\langle \rho^{-1\ \mathbf{Proc}(R)}_{\mathsf{P}},\ \rho^{\mathbf{Proc}(R)}_{\mathsf{S}} \right\rangle.\end{aligned}$$

- *Right unitor is defined analogously.*
- *Given three problems $\pi_1 : \mathsf{P}_1 \longrightarrow \mathsf{S}_1$, $\pi_2 : \mathsf{P}_2 \longrightarrow \mathsf{S}_2$, $\pi_3 : \mathsf{P}_3 \longrightarrow \mathsf{S}_3$, associator being the pair of morphisms*

$$\alpha^{\mathbf{Lagado}(R)}_{\pi_1\pi_2,\pi_3} : (\pi_1\|\pi_2)\|\pi_3 \to \pi_1\|(\pi_2\|\pi_3), \quad \alpha^{\mathbf{Lagado}(R)}_{\pi_1\pi_2,\pi_3} := \left\langle \boldsymbol{\alpha}^{\mathbf{Proc}(R)}_{\mathsf{P}_1\mathsf{P}_2,\mathsf{P}_3}, \boldsymbol{\alpha}^{\mathbf{Proc}(R)}_{\mathsf{S}_1,\mathsf{S}_2\mathsf{S}_3} \right\rangle;$$
$$\alpha^{\mathbf{Lagado}(R)}_{\pi_1,\pi_2\pi_3} : \pi_1\|(\pi_2\|\pi_3) \to (\pi_1\|\pi_2)\|\pi_3, \quad \alpha^{\mathbf{Lagado}(R)}_{\pi_1,\pi_2\pi_3} := \left\langle \boldsymbol{\alpha}^{\mathbf{Proc}(R)}_{\mathsf{P}_1,\mathsf{P}_2\mathsf{P}_3}, \boldsymbol{\alpha}^{\mathbf{Proc}(R)}_{\mathsf{S}_1\mathsf{S}_2,\mathsf{S}_3} \right\rangle.$$

Example 4.15 (Shortest path problems in **Lagado**(R))**.** Some of the five shortest path problems we defined in Example 4.2 can be easily represented as other shortest path problems:

- every undirected graph can be converted into a directed graph by simply adding two directed edges for every undirected edge, which is the first procedure in the morphism:

$$\left\langle \mathbf{c}^{\bar\sigma[n]}_{\vec\sigma[n]}, \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle;$$

- an undirected graph can be represented as a directed graph with edge weights 1 if there is an edge in the original graph and ∞ otherwise, hence we also have the following morphism:

$$\left\langle \mathbf{c}^{\bar\sigma[n]}_{\vec\sigma_{\geq 0}[n]}, \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle;$$

- if a graph has positive weights, it cannot have a negative cycle, hence we also have a morphism

$$\left\langle \mathbf{c}^{\vec\sigma_{\geq 0}[n]}_{\vec\sigma_{\oplus}[n]}, \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle;$$

- and trivially, if a graph has no cycles, it has no negative cycles too, hence there's also the morphism

$$\left\langle \mathbf{c}^{\vec\sigma_{\oslash}[n]}_{\vec\sigma_{\oplus}[n]}, \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle.$$

Recall that in Example 3.29 we defined procedures for computing all shortest paths in a graph defined by an adjacency matrix. **Lagado**(R) can be used to show how a shortest path problem can be converted into a distance product problem. To keep the example simple, we will again ignore the fine print about infinite precision and the weights being positive integers.

First, we define the distance product problem δ to have statement type

$$\langle \mathrm{SM}, \mathbb{N}, S_{\mathrm{SM}} : \mathrm{SM} \to \mathbb{N} \rangle$$

$$\begin{tikzcd}[column sep=huge, row sep=huge]
 & & \mathrm{id}^{\mathbf{Prob}}_{\mathsf{SM}\otimes\mathsf{PM}} \parallel \mathrm{id}^{\mathbf{Prob}}_{\mathsf{OT}[n]} & & \\
 & & \delta \parallel \mathrm{id}^{\mathbf{Prob}}_{\mathsf{OT}[n]} \arrow[u, bend left=60, "\left\langle \mathbf{p}^{\text{naïve}}_{\star} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathrm{id}^{\mathbf{Proc}}_{\mathsf{SM}\otimes\mathsf{PM}} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]} \right\rangle"] \arrow[u, "\left\langle \mathbf{p}^{\text{Strassen}}_{\star} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathrm{id}^{\mathbf{Proc}}_{\mathsf{SM}\otimes\mathsf{PM}} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]} \right\rangle"] \arrow[u, bend right=60, "\left\langle \mathbf{p}^{\text{CW}}_{\star} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathrm{id}^{\mathbf{Proc}}_{\mathsf{SM}\otimes\mathsf{PM}} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]} \right\rangle"'] & & \\
\vec{\sigma}_{\oslash}[n] \arrow[rr, phantom] \arrow[r, "\left\langle \mathbf{c}^{\vec{\sigma}_{\oslash}[n]}_{\vec{\sigma}_{\oplus}[n]},\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] \arrow[urr, "\left\langle \mathbf{a}^{G}_{\vec{\sigma}_{\oslash}[n]} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathbf{k}^{G} \right\rangle"] \arrow[drr, "\left\langle \vec{\mathbf{s}}^{\text{BF}}_{\oslash}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"'] & \vec{\sigma}_{\oplus}[n] \arrow[ur, "\left\langle \mathbf{a}^{G}_{\vec{\sigma}_{\oplus}[n]} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathbf{k}^{G} \right\rangle"] \arrow[dr, "\left\langle \vec{\mathbf{s}}^{\text{BF}}_{\oplus}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] & \vec{\sigma}_{\geq 0}[n] \arrow[l, "\left\langle \mathbf{c}^{\vec{\sigma}_{\geq 0}[n]}_{\vec{\sigma}_{\oplus}[n]},\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"'] \arrow[u, "\left\langle \mathbf{a}^{G}_{\vec{\sigma}_{\geq 0}[n]} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathbf{k}^{G} \right\rangle"] \arrow[d, bend right, "\left\langle \vec{\mathbf{s}}^{\text{D}}_{\geq 0}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"'] \arrow[d, bend left, "\left\langle \vec{\mathbf{s}}^{\text{BF}}_{\geq 0}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] & \vec{\sigma}[n] \arrow[l, "\left\langle \mathbf{c}^{\vec{\sigma}[n]}_{\vec{\sigma}_{\geq 0}[n]},\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"'] \arrow[ul, "\left\langle \mathbf{a}^{G}_{\vec{\sigma}[n]} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathbf{k}^{G} \right\rangle"'] \arrow[dl, "\left\langle \vec{\mathbf{s}}^{\text{D}}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] \arrow[dl, bend left, "\left\langle \vec{\mathbf{s}}^{\text{BF}}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] & \bar{\sigma}[n] \arrow[l, "\left\langle \mathbf{c}^{\bar{\sigma}[n]}_{\vec{\sigma}[n]},\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"'] \arrow[ull, "\left\langle \mathbf{a}^{G}_{\bar{\sigma}[n]} \parallel \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{OT}[n]}\,,\ \mathbf{k}^{G} \right\rangle"'] \arrow[dll, "\left\langle \bar{\mathbf{s}}^{\text{D}}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] \arrow[dll, bend left, "\left\langle \bar{\mathbf{s}}^{\text{BF}}[n],\ \mathrm{id}^{\mathbf{Proc}(R)}_{\mathsf{P}[n]} \right\rangle"] \\
 & & \mathrm{id}^{\mathbf{Prob}}_{\mathsf{P}[n]} & &
\end{tikzcd}$$

Figure 4.1: The subcategory of **Lagado**(R) that contains the problems and procedures discussed in Example 4.15.

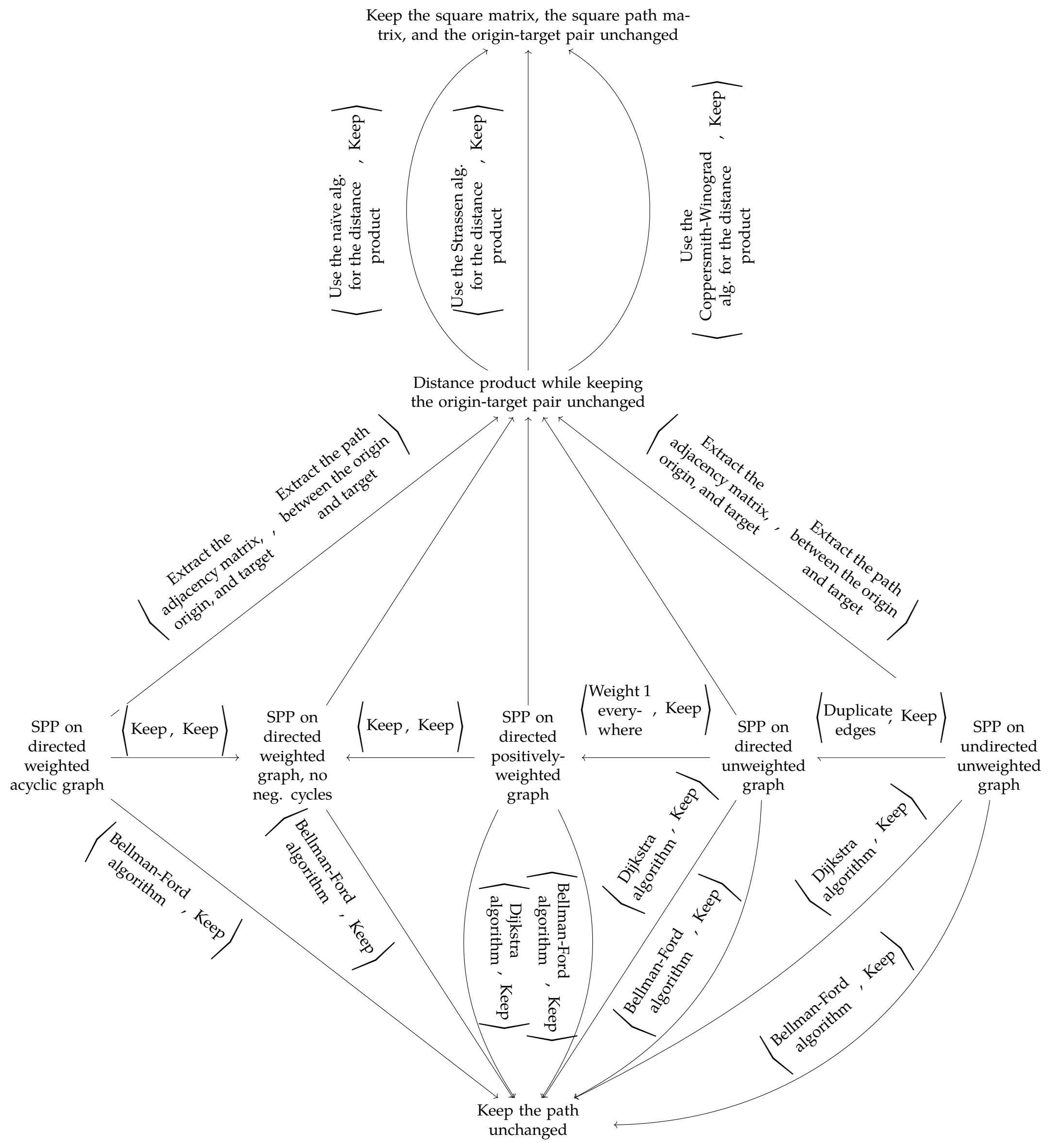

Keep the square matrix, the square path matrix, and the origin-target pair unchanged
⟨Use the naïve alg. for the distance product, Keep⟩
⟨Use the Strassen alg. for the distance product, Keep⟩
⟨Use the Coppersmith-Winograd alg. for the distance product, Keep⟩
Distance product while keeping the origin-target pair unchanged
⟨Extract the adjacency matrix, origin, and target, Extract the path between the origin and target⟩
⟨Extract the adjacency matrix, origin, and target, Extract the path between the origin and target⟩
SPP on directed weighted acyclic graph
⟨Keep, Keep⟩
SPP on directed weighted graph, no neg. cycles
⟨Keep, Keep⟩
SPP on directed positively-weighted graph
⟨Weight 1 every-where, Keep⟩
SPP on directed unweighted graph
⟨Duplicate edges, Keep⟩
SPP on undirected unweighted graph
⟨Bellman-Ford algorithm, Keep⟩
⟨Bellman-Ford algorithm, Keep⟩
⟨Dijkstra algorithm, Keep⟩
⟨Bellman-Ford algorithm, Keep⟩
⟨Dijkstra algorithm, Keep⟩
⟨Bellman-Ford algorithm, Keep⟩
⟨Dijkstra algorithm, Keep⟩
⟨Bellman-Ford algorithm, Keep⟩
Keep the path unchanged

Figure 4.2: The subcategory of **Lagado**(R) that contains the problems and procedures discussed in Example 4.15 represented with intuitive descriptions. The objects and morphisms correspond to the ones in Figure 4.1. “SPP” refers to “shortest path problem” and “Keep” is used for identity procedures that simply return their statement as an answer.

and solution type

$$\langle \mathrm{SM} \times \mathrm{PM},\ \mathbb{N},\ S_{\mathrm{SM}\times\mathrm{PM}} : \mathrm{SM} \times \mathrm{PM} \rightarrow \mathbb{N} \rangle$$

and $\delta(A, (A^\star, \Sigma^\star)) = \mathtt{T}$ iff $A^\star_{ij}$ is the length of the shortest path from i to j in the matrix generated by the adjacency matrix A and $\Sigma^\star_{ij}$ is one of the paths from i to j in that matrix with length $A^\star_{ij}$. From Example 3.29 we know that there are three procedures that can be used to solve δ: $\mathbf{p}^{\text{naïve}}_\star$, $\mathbf{p}^{\text{Strassen}}_\star$, and $\mathbf{p}^{\text{CW}}_\star$.

Now, assume that there is a procedure that "extracts" the adjacency matrix from the graph type at no cost and a translating function that maps to the number of vertices of the graph. We will denote these procedures by $\mathbf{a}^G$ and will index them by their problem type. Similarly, we define the procedure $\mathbf{k}^G$ which, again at no cost, extracts a path, given an adjacency matrix (and corresponding paths), together with an origin and target vertex. In other words, if we have the matrix of all-pairs shortest paths $\Sigma^\star$, and a pair of vertices $\langle i, j \rangle$ it returns the path $\Sigma_{i,j}$ which is the shortest path from i to j.

All the resulting problems (objects in **Lagado**(R)) and procedure pairs (morphisms in **Lagado**(R)) are illustrated in Figure 4.1 using the notation developed so far. Figure 4.2 shows the exact same setting but with more intuitive labels.

The wide row has the five shortest path problems that we are considering in this example. The procedures between them either keep the structure of the graphs unchanged, or simply add two directed edges for each undirected edge or weigh every edge with a constant weight 1.0. The answers don't need to be changed because all shortest path problems have the same answer type, hence the second procedure in the morphisms between them is simply an identity procedure.

All of these problems can be directly solved with the Bellman-Ford algorithm and some can also be solved with Dijkstra's algorithm, which is illustrated with morphisms to $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{P}[n]}$. Again, once the problem is solved and a path is obtained as an answer, no further processing is needed so the second procedure in these morphisms is also just an identity.

As mentioned above, all five shortest path problems can be represented as an all-pairs shortest path problem. That is because if one wants a single shortest path it can be trivially obtained from the all-pairs answer. However, when one moves from a single path to all-pairs they also lose the information on the origin and target nodes because origin and target mean nothing in the all-pairs problem because all nodes are both origins and targets. Therefore, in order to maintain this knowledge, we add the identity problem $\mathrm{id}^{\mathbf{Prob}}_{\mathsf{OT}[n]}$ in parallel to it. This identity problem simply holds the origin and target nodes. Then, once we have an all-pairs solution, then $\mathbf{k}^G$ uses the information stored in this identity problem to know which entry of the answer path matrix it should extract.

Finally, the all-pairs shortest path δ can be solved using any of the three procedures introduced in Example 3.29. These are represented by the topmost three morphisms in the figures.

4.4 Lagado as a heteromorphic twisted category

The careful observer probably noticed that **Lagado**(R) looks very much like a twisted category but where objects and morphisms come from different categories. That is indeed the case and we aim to demonstrate it in this section by introducing *heteromorphic twisted categories* which are a generalization of the concept of a twisted category (Definition 2.41).

Definition 4.16 (Heteromorphic twisted category)**.** Consider two categories $\mathbf{C}$ and $\mathbf{D}$ which have the same objects, i.e. $\mathsf{Ob}(\mathbf{C}) = \mathsf{Ob}(\mathbf{D})$. Consider also that we have a family of functions $\Box$ such that for every $A, B, C, D \in \mathsf{Ob}(\mathbf{C})$ we have the function:

$$ {}^{A}_{B}\Box^{C}_{D} \colon \mathrm{Hom}_{\mathbf{D}}(C,A) \times \mathrm{Hom}_{\mathbf{C}}(A,B) \times \mathrm{Hom}_{\mathbf{D}}(B,D) \times \mathrm{Hom}_{\mathbf{C}}(C,D) \to \mathtt{Bool}, $$

satisfying the following two requirements:

(i) for all $A, B \in \mathsf{Ob}(\mathbf{C})$ and all $f \in \mathrm{Hom}_{\mathbf{C}}(A,B)$ it holds that

$$ {}^{A}_{B}\Box^{A}_{B}\left(\mathrm{id}^{\mathbf{D}}_{A}, f, \mathrm{id}^{\mathbf{D}}_{B}, f\right) = \mathtt{T}, $$

where $\mathrm{id}^{\mathbf{D}}$ are the identity morphisms in $\mathbf{D}$;

(ii) for all $A, B, C, D, E, F \in \mathsf{Ob}(\mathbf{C})$ it holds that

$$ {}^{A}_{B}\Box^{C}_{D}(p,f,q,g) \ \wedge\ {}^{C}_{D}\Box^{E}_{F}(r,g,s,h) = \mathtt{T} \implies {}^{A}_{B}\Box^{E}_{F}(r \mathbin{⨾_{\mathbf{D}}} p, f, g \mathbin{⨾_{\mathbf{D}}} s, h) = \mathtt{T}. $$

Then we define the $\mathbf{HTw}(\mathbf{C}, \mathbf{D}, \Box)$ category having:

i. objects which are morphisms in $\mathbf{C}$;

ii. morphisms from the object $g : C \to D$ to the object $f : A \to B$ are pairs of morphisms $\langle p, q \rangle$, where $p \in \mathrm{Hom}_{\mathbf{D}}(C, A)$ and $q \in \mathrm{Hom}_{\mathbf{D}}(B, D)$, such that

$$ {}^{A}_{B}\Box^{C}_{D}(p,f,q,g) = \mathtt{T}; $$

iii. identity morphism for the object $f : A \to B$ is $\langle \mathrm{id}^{\mathbf{D}}_{A}, \mathrm{id}^{\mathbf{D}}_{B} \rangle$;

iv. morphism composition for any three objects $f\colon A \to B$, $g\colon C \to D$, $h\colon E \to F$, and two morphisms $\langle p, q \rangle : g \to f$, $\langle r, s \rangle : h \to g$, their composition is defined as:

$$ \langle r, s \rangle ⨾ \langle p, q \rangle = \langle r \mathbin{⨾_{\mathbf{D}}} p,\ q \mathbin{⨾_{\mathbf{D}}} s \rangle . $$

Remark 4.17. In the above definition we defined ${}^{A}_{B}\Box^{C}_{D}$ to be a function with a domain being the Cartesian product of hom-sets. Strictly speaking that is true only if the hom-sets are actually sets (rather than classes), which means that **C** and **D** would have to be small categories. However, we will ignore such technicalities and will assume that the above definition is always valid.

Lemma 4.18. *For any two categories* **C** *and* **D** *with* $\mathsf{Ob}(\mathbf{C}) = \mathsf{Ob}(\mathbf{D})$*, together with a family of functions* $\Box$ *that satisfies the two requirements in Definition 4.16, it holds that* $\mathbf{HTw}(\mathbf{C}, \mathbf{D}, \Box)$ *is indeed a category.*

Proof. We need to show that the requirements for the $\Box$ family imply that all identity morphisms and all compositions of morphisms exist, as well as that the associativity and identity axioms hold. Condition (i) ensures that the identity morphisms are included in $\mathbf{HTw}(\mathbf{C}, \mathbf{D}, \Box)$. Condition (ii) ensures that all composition are also included in $\mathbf{HTw}(\mathbf{C}, \mathbf{D}, \Box)$.

Let's demonstrate that the identity axiom holds. Take two objects $g : C \to D$ and $f : A \to B$, together with the morphism between them

$$\left\langle p : C \to A, q : B \to D \right\rangle .$$

Then, then due to the identity axiom holding for morphisms in **D** we have:

$$\left\langle p, q \right\rangle ⨟ \mathrm{id}_f = \left\langle p, q \right\rangle ⨟ \left\langle \mathrm{id}^{\mathbf{D}}_A, \mathrm{id}^{\mathbf{D}}_B \right\rangle = \left\langle p \;⨟_{\mathbf{D}}\; \mathrm{id}^{\mathbf{D}}_A,\; \mathrm{id}^{\mathbf{D}}_B \;⨟_{\mathbf{D}}\; q \right\rangle = \left\langle p, q \right\rangle ,$$

and

$$\mathrm{id}_g ⨟ \left\langle p, q \right\rangle = \left\langle \mathrm{id}^{\mathbf{D}}_C, \mathrm{id}^{\mathbf{D}}_D \right\rangle \left\langle p, q \right\rangle = \left\langle \mathrm{id}^{\mathbf{D}}_C \;⨟_{\mathbf{D}}\; p,\; q \;⨟_{\mathbf{D}}\; \mathrm{id}^{\mathbf{D}}_D \right\rangle = \left\langle p, q \right\rangle .$$

Finally, let's also show that the associativity axiom holds. Take three morphisms $\left\langle p, q \right\rangle : g \to f$, $\langle r, s \rangle : h \to g$, $\langle t, u \rangle : i \to h$. Then, due to the associativity of the morphisms in **D** we have:

$$\begin{aligned}
(\langle t, u \rangle ⨟ \langle r, s \rangle) ⨟ \left\langle p, q \right\rangle &= \left\langle t ⨟ r, s ⨟ u \right\rangle ⨟ \left\langle p, q \right\rangle \\
&= \left\langle (t ⨟ r) ⨟ p, q ⨟ (s ⨟ u) \right\rangle \\
&= \left\langle t ⨟ (r ⨟ p), (q ⨟ s) ⨟ u \right\rangle \\
&= \langle t, u \rangle ⨟ \left\langle r ⨟ p, q ⨟ s \right\rangle \\
&= \langle t, u \rangle ⨟ \left(\langle r, s \rangle ⨟ \left\langle p, q \right\rangle \right) .
\end{aligned}$$

□

Remark 4.19. The definition of the heteromorphic twisted category is almost the same as the one for a twisted category. There are two main differences, however. First, the direction of the arrows is flipped. In a twisted category we have a morphism $f \to g$ if g factorizes through f, but in a heteromorphic twisted category a morphism $f \longrightarrow g$ implies that f factorizes through g. The change is introduced for convenience down the road and the definition can easily be flipped, should one wish to do so.

The other difference is the introduction of the $\square$ functions. This is necessary because once we get morphisms from two different categories, we can no longer talk about building commutative squares with them, or even about composing such morphisms. Hence, the purpose of the $\square$ functions is to act as a "glue" and to generalize the notion of commutativity to morphisms from different (heteromorphic) categories.

Remark 4.20. The heteromorphic twisted category **HTw** generalizes the twisted category **Tw**. In particular, if we take:

$$
{}^{A}_{B}\square^{C}_{D} \colon \mathrm{Hom}_{\mathbf{C}}(C,A) \times \mathrm{Hom}_{\mathbf{C}}(A,B) \times \mathrm{Hom}_{\mathbf{C}}(B,D) \times \mathrm{Hom}_{\mathbf{C}}(C,D) \longrightarrow \mathtt{Bool},
$$
$$
\langle p, f, q, g \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } p \,\mathbin{⨟}_{\mathbf{C}}\, f \,\mathbin{⨟}_{\mathbf{C}}\, q = g, \\ \mathtt{F} & \text{otherwise,} \end{cases}
$$

then $\mathbf{HTw}(\mathbf{C}, \mathbf{C}, \square) = (\mathbf{Tw}(\mathbf{C}))^{\mathrm{op}}$.

Lemma 4.21. *Take a family of functions* $\square$ *such that for any* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D} \in \mathrm{Ob}(\mathbf{NTypes})$ *we have:*

$$
{}^{\mathsf{A}}_{\mathsf{B}}\square^{\mathsf{C}}_{\mathsf{D}} \colon \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{C},\mathsf{A}) \times \mathrm{Hom}_{\mathbf{Prob}}(\mathsf{A},\mathsf{B}) \times \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{B},\mathsf{D}) \times \mathrm{Hom}_{\mathbf{Prob}}(\mathsf{C},\mathsf{D}) \longrightarrow \mathtt{Bool},
$$
$$
\langle \mathbf{p}, f, \mathbf{q}, g \rangle \mapsto \begin{cases} \mathtt{T} & \textit{if } \forall \mathbf{h} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{A},\mathsf{B}),\ \textit{s.t. } f(a, h(a)) = \mathtt{T}, \forall a \in A \\ & \textit{it holds that} \\ & g\left(c,\ (p \mathbin{⨟} h \mathbin{⨟} q)(c)\right) = \mathtt{T}, \forall c \in C, \\ \mathtt{F} & \textit{otherwise.} \end{cases}
$$

Then, $\mathbf{Lagado}(R) = \mathbf{HTw}(\mathbf{Prob}, \mathbf{Proc}(R), \square)$*.*

Proof. Both the objects of $\mathbf{Lagado}(R)$ and $\mathbf{HTw}(\mathbf{Prob}, \mathbf{Proc}(R), \square)$ are morphisms in **Prob**, hence the two categories are the same object-wise. So it's only left to show that if a morphism is in $\mathbf{Lagado}(R)$ then it also must be in $\mathbf{HTw}(\mathbf{Prob}, \mathbf{Proc}(R), \square)$, and if a morphism is in $\mathbf{HTw}(\mathbf{Prob}, \mathbf{Proc}(R), \square)$ then it also is in $\mathbf{Lagado}(R)$.

A morphism in $\mathbf{Lagado}(R)$ from a problem $\pi_1 \colon \mathsf{P}_1 \longrightarrow \mathsf{S}_1$ to a problem $\pi_2 \colon \mathsf{P}_2 \longrightarrow \mathsf{S}_2$ is a pair of procedures $\langle \mathbf{p}, \mathbf{s} \rangle$, such that $\mathbf{p} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{P}_1, \mathsf{P}_2)$, $\mathbf{s} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{S}_2, \mathsf{S}_1)$, and for any $\mathbf{h} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{P}_2, \mathsf{S}_2)$ which is a solution of π_2 it holds that $\mathbf{p} \mathbin{⨟} \mathbf{h} \mathbin{⨟} \mathbf{s} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{P}_1, \mathsf{S}_1)$ is a solution of π_1. For any $\mathbf{h}$ which is a solution of π_2 we have $\pi_2(p_2, h(p_2)) = \mathtt{T}, \forall p_2 \in P_2$. and of course we also have that $\pi_1(p_1, (p \mathbin{⨟} h \mathbin{⨟} q)(p_1)) = \mathtt{T}, \forall p_1 \in P_1$. But then that also means that

$$
{}^{\mathsf{P}_2}_{\mathsf{S}_2}\square^{\mathsf{P}_1}_{\mathsf{S}_1}(\mathbf{p}, \pi_2, \mathbf{s}, \pi_1) = \mathtt{T}.
$$

And from Definition 4.16 we know that if the above is true then the pair $\langle \mathbf{p}, \mathbf{q} \rangle$ is a morphism from π_1 to π_2 in $\mathbf{HTw}(\mathbf{C}, \mathbf{D}, \Box)$.

Showing that the opposite also holds can be done in an analogous way. □

Chapter 5

Compositional systems and relations

In the previous chapters we developed the theory of compositional problems in a factual-definition way by simply saying "this is that and this and that happen to work out just fine". For example, from the onset we started defining everything in terms of various concepts of category theory but we never justified *why* this is a good modelling framework. The main goal of this chapter is to serve as a defense for this choice.

We will do that by defining a number of assumptions which seem reasonable when one aspires to work with compositional theories and will show that these assumptions result in the structures used in the previous chapters. In fact, we will show that they result in something more general: semicategories and kinded functions. Then, finally, we will show how many other concepts we used in the previous chapters are instances of semicategories and kinded functions. We do all that in order to prepare for Chapter 6 where we will use them to generalize the problems, procedures and solutions that we saw in Chapter 4.

In order to keep an open mind, we will introduce things with new names and will only use concepts previously introduced in this thesis only after they have been shown to be equivalent to the concepts here.

5.1 Compositional systems

First, let's address the "compositionality" question. We have never defined what we mean by compositional so far. Intuitively, "compositional" means that as long as some things respect the same interfaces they are interchangeable. To be more concrete, let's call these things *components*. Then, our first assumption would be:

Assumption 5.1. Anything that *composes in* a component is a component. Anything that a component *composes in* is a component.

We will illustrate a component C_1 composing in a component C_2 like this:

$$\underline{\quad C_1 \quad} \circledcirc \underline{\quad C_2 \quad}$$

We said we'd like to have components sharing the same interface to be interchangeable. If one component composes in another, then there is an interface between them. Hence, any other component sharing this interface should be able to replace one of the original components. Then, it is also natural to make the following assumption.

Assumption 5.2. If component C_1 *composes in* component C_2, then any component C_L which composes in C_2 also composes in any component C_R that C_1 composes in.

Hence, visually:

$$\begin{array}{ccc}
\underline{\quad C_1 \quad} \circledcirc \underline{\quad C_2 \quad} & & \\
\underline{\quad C_L \quad} \circledcirc \underline{\quad C_2 \quad} & \Longrightarrow & \underline{\quad C_L \quad} \circledcirc \underline{\quad C_R \quad} \\
\underline{\quad C_1 \quad} \circledcirc \underline{\quad C_R \quad} & &
\end{array}$$

We will also refer to the statement "C_1 composes in C_2" as the *tandem of* C_1 *and* C_2 and will denote it by $T(C_1, C_2)$. Note that the composition as defined above is not symmetric, i.e. C_1 composing in C_2 does not imply C_2 composing in C_1. The precises meaning of *component* and *composes in* we leave free, we only constrain them to relate as in the assumption.

It is also reasonable to consider that whether composing two components affects their ability to compose to other components. For example consider having two pipes, each with two holes, one on each end. These holes can have different diameters but we require that one of the holes of the first pipe is the same as one of the holes of the other pipe, so that we can connect them. Once we connect them we again have a pipe with two holes which so happen to have the diameters of the holes we did not connect and to which we can connect other pipes. This example illustrates why we believe that it is reasonable to make the next assumption.

Assumption 5.3. If component C_1 composes in component C_2 and component C_2 composes in component C_3, then the tandem of C_1 and C_2 composes in C_3 and C_1 composes in the tandem of C_2 and C_3.

From Assumption 5.1 and Assumption 5.3 we can conclude that the tandem of two components is a component itself, hence:

$$\begin{array}{c} \underset{\rule{3em}{0.4pt}}{C_1} \circledcirc \underset{\rule{3em}{0.4pt}}{C_2} \\[1em] \underset{\rule{3em}{0.4pt}}{C_2} \circledcirc \underset{\rule{3em}{0.4pt}}{C_3} \end{array} \Longrightarrow \begin{array}{c} \underset{\rule{3em}{0.4pt}}{T(C_1,C_2)} \circledcirc \underset{\rule{3em}{0.4pt}}{C_3} \\[1em] \underset{\rule{3em}{0.4pt}}{C_1} \circledcirc \underset{\rule{3em}{0.4pt}}{T(C_2,C_3)} \end{array} \tag{5.1}$$

Note that we speak of *the* tandem even though we never specifically constrained that there is only one. However, it is easy to see that as far as the composability properties outlined above are concerned, any tandem would behave identically in this system. Hence, should there be more than one, it does not matter which one we pick.

We will call a collection of such components together with a collection of pairs of components where one composes in the other a *compositional system*. If we want to write this more precisely, take $\mathcal{C}$ to be a set of components, $T : \mathcal{C} \times \mathcal{C} \hookrightarrow \mathcal{C}$ is a partial function, and $\mathcal{P} \subseteq \mathcal{S}$ to be a set of pairs of components, such that if $\langle C_1, C_2\rangle, \langle C_2, C_3\rangle \in \mathcal{P}$, then $T(C_1, C_2)$ and $T(C_2, C_3)$ are defined (i.e. $\mathcal{P}$ is the domain of definition of T) and $\langle T(C_1, C_2), C_3\rangle \in \mathcal{P}$ and $\langle C_1, T(C_2, C_3)\rangle \in \mathcal{P}$. Then $\langle \mathcal{C}, T, \mathcal{P}\rangle$ is a compositional system.

Remark 5.4. It is perfectly possible to construct compositional systems in which no two components can be composed. The simplest example is a compositional system with only one component which doesn't compose in itself. Or a compositional system with two components neither of which composes into itself or the other. Or a compositional system with N such components. While it is rather counter-intuitive that we use the adjective *compositional* for systems in which no components can be composed, this will allows us to develop a more general theory later on.

5.2 Relationships between compositional systems

Let's take a step back. In this thesis, we are addressing the problem of compositional problems and solutions. So let's say we have two compositional systems called **Problems** and **Solutions**. If we call them with these names, it would give them some semantic meaning and expectations. However, these can be any arbitrary compositional systems. So, to keep our discussion as general as possible, let's rename them **Alice** and **Bob**.

Now, we would like some way of studying the relations between the components of **Alice** and the components of **Bob**. Of course, the first question we should ask is *what is a relation between components of two compositional systems*. The most basic kind of a relation is one that is constant: regardless of what it is, it is the same for any pair of components of **Alice** and **Bob**. Clearly, this is as good as nothing because if everything relates in the same way there's not much to study about this relationship. Perhaps, instead we can distinguish two levels of relating which we can call *Compatible* and *Incompatible*. Now, we can be speaking of a component A_1 of **Alice** being compatible with a component B_1 of **Bob**. Of course, we can extend the two levels to any arbitrary amount. We will call these *kinds* and will denote the kind of two components A, B, respectively from **Alice** and **Bob**, by $K(A, B)$. Let's formulate this as a new assumption:

Assumption 5.5. A *kinded relation* of kind set **K** between two compositional systems **Alice** and **Bob** assigns every pair of components of **Alice** and **Bob** an element of the kind set **K**.

This might be sufficient at first sight but recall that we can also compose components of **Alice** and **Bob** which results in new components which in turn should again be mapped to elements of **K**. If we do this arbitrarily we might be left in an interesting situation. For example, imagine we map $\langle A_1, B_1\rangle$ to $\triangleleft$, $\langle A_2, B_2\rangle$ to $\triangleright$, and that A_1 composes in A_2 and B_1 composes in B_2, while their tandems happen to map to $\square$ as in the equation bellow. Then, one is left wondering whether $\triangleleft$, $\triangleright$, and $\square$ should be completely independent or whether we should require some relationship between them as well.

$$\begin{array}{rcccl}
\textbf{Alice}: & \underline{\;A_1\;} & \underline{\;A_2\;} & \underline{\;A_1\;} \circledcirc \underline{\;A_2\;} & (T(A_1, A_2)) \\
\textbf{Bob}: & \underline{\;B_1\;} & \underline{\;B_2\;} & \underline{\;B_1\;} \circledcirc \underline{\;B_2\;} & (T(B_1, B_2)) \\
\mathbf{K}: & \triangleleft & \triangleright & \square &
\end{array} \tag{5.2}$$

For an even more convincing illustration for the need for structure in **K** consider the following case where we assume that A_1 composes into A_2 which in turn composes into A_3, and similarly for the Bs:

$$\begin{array}{rccccc}
\textbf{Alice}: & \underline{\;A_1\;} & \underline{\;A_2\;} & \underline{\;A_3\;} & \underline{T(A_1,A_2)} & \underline{T(A_2,A_3)} \\
\textbf{Bob}: & \underline{\;B_1\;} & \underline{\;B_2\;} & \underline{\;B_3\;} & \underline{T(B_1,B_2)} & \underline{T(B_2,B_3)} \\
\mathbf{K}: & \triangleleft & \triangle & \triangleright & \boxplus & \boxminus \\
\textbf{Alice}: & \underline{T(T(A_1,A_2),A_3)} & \underline{T(T(A_1,A_2),A_3)} & \underline{T(A_1,T(A_2,A_3))} & \underline{T(A_1,T(A_2,A_3))} & \\
\textbf{Bob}: & \underline{T(T(B_1,B_2),B_3)} & \underline{T(B_1,T(B_2,B_3))} & \underline{T(T(B_1,B_2),B_3)} & \underline{T(B_1,T(B_2,B_3))} & \\
\mathbf{K}: & \clubsuit & \diamondsuit & \spadesuit & \heartsuit &
\end{array} \tag{5.3}$$

In the second row we are trying to assess the kind of compositions of the exact same things (sameness will be formalized in Assumption 5.7).

Recall the pipe example. Whether we first connect pipes 1 and 2 and then pipe 3 or we connect pipe 1 to the previously connected pipes 2 and 3 should not change the kind of the relation between the final tandem and a component in another compositional system. Hence, it is also reasonable to identify the two compositions (i.e. to enforce associativity). We can, however, do this at the level of the compositional system or at the level of the relation between two compositional systems. In the compositional system that'd be done by asking for $T(A_1, T(A_2, A_3))$ to be the same component as $T(T(A_1, A_2), A_3)$. If we are to require it at the level of the relation between **Alice** and **Bob**, we'd be asking for the four pairs in the bottom row of Equation (5.3) to have the same kind, i.e:

$$\begin{array}{rcccc} \textbf{Alice}: & \underline{T(T(A_1,A_2),A_3)} & \underline{T(T(A_1,A_2),A_3)} & \underline{T(A_1,T(A_2,A_3))} & \underline{T(A_1,T(A_2,A_3))} \\ \textbf{Bob}: & \underline{T(T(B_1,B_2),B_3)} & \underline{T(B_1,T(B_2,B_3))} & \underline{T(T(B_1,B_2),B_3)} & \underline{T(B_1,T(B_2,B_3))} \\ \textbf{K}: & \spadesuit & \spadesuit & \spadesuit & \spadesuit \end{array} \tag{5.4}$$

The first situation, of course, implies the second one. Furthermore, following the example with the tubes, as well as other real-world systems, our intuition hints at associativity being an *intrinsic* property of the compositional system itself, rather than an effect that only arises whenever it is being related to another system. Connecting pipes 1, 2 and 3 results in the same final pipe regardless of whether it is being in relation to another system or simply by itself. Hence, we add the following assumption:

Assumption 5.6. If component C_1 composes in component C_2 which composes in component C_3 then the tandem of the tandem of C_1 and C_2 with C_3 is the same component as the tandem of C_1 with the tandem of C_2 and C_3. Or,

$$T(T(C_1, C_2), C_3) = T(C_1, T(C_2, C_3)).$$

The above assumption itself depends on the assumption that we have a concept of "sameness" of components in a compositional system. To make sure that this is indeed the case, let's formalize it.

Assumption 5.7. Given any two components C_1 and C_2, they are either *the same* or they are *distinct*. If C_1 and C_2 are the same then any component that composes in C_1 composes in C_2 and vice versa. Furthermore, if C_1 and C_2 are the same then C_1 composes in any component that C_2 composes in and vice versa.

Thanks to Assumption 5.6 it means that we do not need to specify the order of composition, hence all four of the tandems in Equation (5.4) can be represented as:

$$\begin{array}{rccccc}
\textbf{Alice}: & \frac{A_1}{} & \circledcirc & \frac{A_2}{} & \circledcirc & \frac{A_3}{} \\
\textbf{Bob}: & \frac{B_1}{} & \circledcirc & \frac{B_2}{} & \circledcirc & \frac{B_3}{} \\
\textbf{K}: & & & \spadesuit & &
\end{array}$$

We still haven't solved the original problem, though. Recall that we were concerned that $\triangleleft$, $\triangleright$ and $\square$ in Equation (5.2) are not at all related. But once again, why do we care about it? After all, we solved the associativity problem already! Let's see another example that should illustrate the nature of our concern:

$$\begin{array}{rcccccccccc}
\textbf{Alice}: & \frac{A_1}{} & \frac{A_2}{} & \frac{A_1}{} & \circledcirc & \frac{A_2}{} & \frac{A_2'}{} & \frac{A_1}{} & \circledcirc & \frac{A_2'}{} \\
\textbf{Bob}: & \frac{B_1}{} & \frac{B_2}{} & \frac{B_1}{} & \circledcirc & \frac{B_2}{} & \frac{B_2'}{} & \frac{B_1}{} & \circledcirc & \frac{B_2'}{} \\
\textbf{K}: & \triangleleft & \triangleright & & \square & & \triangleright & & \blacksquare &
\end{array} \tag{5.5}$$

Of course we assume that everything that's composed above is actually composable. Now we get that the pairs $\langle A_2, B_2\rangle$ and $\left\langle A_2', B_2'\right\rangle$ are of the same kind but if the pair $\langle A_1, B_1\rangle$ is composed into each of them we obtain two different kinds of tandems. That doesn't feel very compositional.

So, essentially we want that $T(A_1, A_2)$ and $T(B_1, B_2)$ are mapped to the same kind as $T(A_1, A_2')$ and $T(B_1, B_2')$, or that in Equation (5.5) we have $\square$ and $\blacksquare$ being the same kind. In other words, if we replace a pair of components with another pair which maps to the same kind, then the kind of the tandem should not change. Let's put it as an assumption:

Assumption 5.8. If

$$\begin{aligned}
K(A_1, B_1) &= \triangleleft, \\
K(A_2, B_2) &= \triangleright, \\
K(A_1', B_1') &= \triangleleft, \\
K(A_2', B_2') &= \triangleright, \\
K(T(A_1, A_2), T(B_1, B_2)) &= \square,
\end{aligned}$$

then

$$K(T(A_1, A_2'), T(B_1, B_2')) = \square \ \wedge \ K(T(A_1', A_2), T(B_1', B_2)) = \square,$$

where A_1, A_1', A_2, A_2' are components in **Alice** and B_1, B_1', B_2, B_2' are components in **Bob**.

Visually that looks like:

$$\begin{array}{rccccccccccccccccc}
\textbf{Alice}: & \frac{A_1}{} & \frac{A_2}{} & \frac{A_1'}{} & \frac{A_2'}{} & \frac{A_1}{} & \circledcirc & \frac{A_2}{} & & \frac{A_1'}{} & \circledcirc & \frac{A_2}{} & \frac{A_1}{} & \circledcirc & \frac{A_2'}{} & & \frac{A_1'}{} & \circledcirc & \frac{A_2'}{} \\
\textbf{Bob}: & \frac{B_1}{} & \frac{B_2}{} & \frac{B_1'}{} & \frac{B_2'}{} & \frac{B_1}{} & \circledcirc & \frac{B_2}{} & \Longrightarrow & \frac{B_1'}{} & \circledcirc & \frac{B_2}{} & \frac{B_1}{} & \circledcirc & \frac{B_2'}{} & \Longrightarrow & \frac{B_1'}{} & \circledcirc & \frac{B_2'}{} \\
\textbf{K}: & \triangleleft & \triangleright & \triangleleft & \triangleright & & \square & & & & \square & & & \square & & & & \square &
\end{array}$$

The second implication follows from applying the assumption twice.

5.3 The kinded nature

Assumption 5.8 prevents the tandem of similarly kinded components to be differently kinded, but it still leave space for some interesting effects. In this section we will try to see what these effects are and what should the structure of a kind be.

The main observation here is that Assumption 5.8 requires that *both* the interfaces and the kinds of the components match. Hence, it does not prevent that a pair of components kinded $\lhd$ and $\rhd$ composes to a kind $\square$ for some components and to a kind $\blacksquare$ for other components, as long as the constituent components cannot be composed. While the explanation can be a bit convoluted, an example can make the situation clear. The above statement essentially means that we are completely fine with the following situation:

$$\begin{array}{rccccccc}
\textbf{Alice}: & \underline{A_1} & \underline{A_2} & \underline{A_1} \circledcirc \underline{A_2} & \underline{A_x} & \underline{A_y} & \underline{A_x} \circledcirc \underline{A_y} \\
\textbf{Bob}: & \underline{B_1} & \underline{B_2} & \underline{B_1} \circledcirc \underline{B_2} & \underline{B_x} & \underline{B_y} & \underline{B_x} \circledcirc \underline{B_y} \\
\textbf{K}: & \lhd & \rhd & \square & \lhd & \rhd & \blacksquare
\end{array} \tag{5.6}$$

where A_1 *cannot* be composed in A_y, A_x *cannot* be composed in A_2, B_1 *cannot* be composed in B_y, and B_x *cannot* be composed in B_2. We could ask that such situations do not occur, thus identifying the pair $\langle \lhd, \rhd \rangle$ with $\square$. However, we could also choose not to do it, because the above situation does not hurt us as long as we don't try to compose the pairs that we just said we cannot. And Assumption 5.8 ensures that this is indeed the case.

Even if we compose the two distinct tandems from the above equation into the same pair, we still can map the composition of the components with the same kinds to different kinds. Here's an example of this situation:

$$\begin{array}{rccccccc}
\textbf{Alice}: & \underline{A_1} & \underline{A_2} & \underline{A_1} \circledcirc \underline{A_2} & \underline{A_x} & \underline{A_y} & \underline{A_x} \circledcirc \underline{A_y} & \underline{A_R} \\
\textbf{Bob}: & \underline{B_1} & \underline{B_2} & \underline{B_1} \circledcirc \underline{B_2} & \underline{B_x} & \underline{B_y} & \underline{B_x} \circledcirc \underline{B_y} & \underline{B_R} \\
\textbf{K}: & \lhd & \rhd & \square & \lhd & \rhd & \blacksquare & \star
\end{array}$$

$$\begin{array}{rcccc}
\textbf{Alice}: & \underline{A_y} \circledcirc \underline{A_R} & \underline{A_2} \circledcirc \underline{A_R} & \underline{A_x} \circledcirc \underline{A_y} \circledcirc \underline{A_R} & \underline{A_1} \circledcirc \underline{A_2} \circledcirc \underline{A_R} \\
\textbf{Bob}: & \underline{B_y} \circledcirc \underline{B_R} & \underline{B_2} \circledcirc \underline{B_R} & \underline{B_x} \circledcirc \underline{B_y} \circledcirc \underline{B_R} & \underline{B_1} \circledcirc \underline{B_2} \circledcirc \underline{B_R} \\
\textbf{K}: & \times & \times & \vee & \wedge
\end{array}$$

where we have the same *non*-composability conditions as above. First, note that the first two tandems on the bottom row must map to the same kind.

That is directly due Assumption 5.8 as $K(A_y, B_y) = K(A_2, B_2)$ and both A_y and A_2 compose into A_R, and as B_y and B_2 compose into B_R. However, this does not prevent us from having the second two tandems on the second row being different. Assumption 5.8 does not apply here because A_1 *cannot* be composed in A_y, A_x *cannot* be composed in A_2, B_1 *cannot* be composed in B_y, and B_x *cannot* be composed in B_2.

We have observed that pairs of components together with a specific kind map induce some sort of structure on the kinds. For example, above, we saw that "combining" $\lhd$ and $\rhd$ "can result" in $\square$ (but can also result in something else). In fact we saw all the following combination mappings:

(i) $\langle \lhd, \rhd \rangle \mapsto \square$,
(ii) $\langle \lhd, \rhd \rangle \mapsto \blacksquare$,
(iii) $\langle \blacksquare, \star \rangle \mapsto \vee$,
(iv) $\langle \square, \star \rangle \mapsto \wedge$,
(v) $\langle \rhd, \star \rangle \mapsto \times$,
(vi) $\langle \lhd, \times \rangle \mapsto \vee$,
(vii) $\langle \lhd, \times \rangle \mapsto \wedge$.

Instead of simply thinking of $\mathbf{K}$ as a set with this structure being induced by a specific pair of compositional systems and a particular map K, we can think of what is the structure on $\mathbf{K}$ that makes it possible to be a kind. This structure can be considered as a homogeneous relation of degree 3 over the set of kinds $\mathbf{K}$, i.e. $R \subseteq \mathbf{K} \times \mathbf{K} \times \mathbf{K}$. Then the above list of combinations can be rewritten as:

(i) $\langle \lhd, \rhd, \square \rangle \in R$,
(ii) $\langle \lhd, \rhd, \blacksquare \rangle \in R$,
(iii) $\langle \blacksquare, \star, \vee \rangle \in R$,
(iv) $\langle \square, \star, \wedge \rangle \in R$,
(v) $\langle \rhd, \star, \times \rangle \in R$,
(vi) $\langle \lhd, \times, \vee \rangle \in R$,
(vii) $\langle \lhd, \times, \wedge \rangle \in R$.

Note that such a relation over the kinds is the most general construction that can be used to describe a relationship between two compositional systems. However, depending on the precise nature of this relationship, one might wish to restrict it further. We will soon see that that is indeed the case when one wishes to characterize the problem relationship between the compositional systems of statements and answers and the solution relationship between these two compositional systems.

5.4 The categorical connection

In the previous sections we developed several assumptions that resulted in specific constraints for our compositional systems and the possible kinded relations between them. Here we will see the connections between these constraints and well-known categorical concepts.

First, we claim that Assumption 5.1, Assumption 5.2, Assumption 5.3, and Assumption 5.6 imply that a compositional system is a semicategory (Definition 2.34).

Lemma 5.9. *A compositional system satisfying Assumption 5.1, Assumption 5.2, Assumption 5.3, and Assumption 5.6 is a semicategory.*

Proof. First, let's see what is the connection between the components of the compositional system and the structure of the semicategory. Then, we will show that composition is well-defined and associative, as required in Definition 2.34.

Take $\mathbf{M} = \langle \mathcal{C}, T, \mathcal{P} \rangle$ to be our compositional system. Recall that $\mathcal{C}$ is the set of components, $T : \mathcal{C} \times \mathcal{C} \hookrightarrow \mathcal{C}$ is a partial function that identifies some pairs of components with another component which we called their tandem, and that $\mathcal{P}$ is the domain of definition of T which has the property that

$$\langle C_1, C_2 \rangle \in \mathcal{P} \wedge \langle C_2, C_3 \rangle \in \mathcal{P} \implies \langle T(C_1, C_2), C_3 \rangle \in \mathcal{P} \wedge \langle T(C_1, C_2), C_3 \rangle \in \mathcal{P}.$$

Now, the components $\mathcal{C}$ are the morphisms in a semicategory $\mathbf{M}^*$. We will define the objects implicitly, based on the compositional structure of $\mathbf{M}$. Our construction is based on the following rules for the target of the morphism corresponding to a component $A \in \mathcal{C}$:

T.i If A composes into another component B, then the target of the morphism corresponding to A is the origin of the morphism corresponding to B.

T.ii Otherwise, the target of the morphism corresponding to A is an object that is the origin of no morphism.

And the following rules for the origin of the morphism corresponding to A:

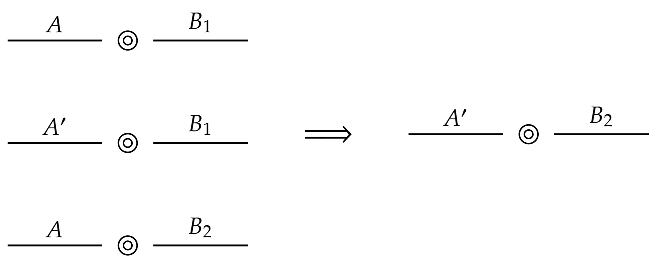
O.i If another component B composes into A, then the origin of the morphism corresponding to A is the target of the morphism corresponding to B.

O.ii Otherwise, the origin of the morphism corresponding to A is an object that is the target of no morphism.

Let's show that this is a consistent definition, meaning that every morphism in $\mathbf{M}^*$ has exactly one origin and one target object. Take a component $A \in \mathcal{C}$. A must have at least one target because in both T.i and T.ii it is assigned a target and a case not covered by these two cannot exist. The same holds for the origin of A.

Now, let's show by contradiction that A cannot have two or more targets. If it has two targets, then there must be two distinct components $B_1, B_2 \in \mathcal{C}$ that A composes into. Then, due to O.i we know that unless there is a component A' composing into B_1 which doesn't compose into B_2, the origins of B_1 and

B_2 would be the same. However, it is impossible to have a component A' composing into B_1 which doesn't compose into B_2 due to Assumption 5.2:



Hence, the morphism corresponding to A has exactly one target. The argument that it also has exactly one origin is symmetric.

Next, the tandem of components and the composition of morphisms in a semicategory are exactly the same thing. Furthermore, Assumption 5.3 guarantees that the composition of morphisms in $\mathbf{M}^*$ is transitive.

Finally, the associativity of $\mathbf{M}^*$ directly follows from Assumption 5.6. □

Remark 5.10. In fact, our original setting of compositional systems that focused on component compositionality rather than on objects is much more aligned with the historical view on categories. In their seminal paper, Eilenberg and MacLane, 1945 defined a category purely by the means of its morphisms, while objects were delegated solely to be induced by the identity morphisms.

The natural question that follows is how do kinded relationships as in Assumption 5.5 fit in the categorical perspective. We defined kinded relationships to map pairs of components of two distinct composite systems **Alice** and **Bob** to elements of a set **K**. We just showed that there are semicategories **Alice*** and **Bob*** whose morphisms are the component of **Alice** and **Bob**. Hence, pairs of components of **Alice** and **Bob** are pairs of morphisms of **Alice*** and **Bob***. Therefore, a kinded relation can be represented as a function from the product of the sets of morphisms (assuming that the two categories are small) to the set of kinds. We call such functions *kinded functions*.

Definition 5.11 (Kinded function)**.** Given two semicategories $\mathbf{C}^*$ and $\mathbf{D}^*$, and a set $\mathbf{K}$, a *kinded function between* $\mathbf{C}^*$ *and* $\mathbf{D}^*$ *of kind* $\mathbf{K}$, denoted

$$\mathbf{C}^* \xrightarrow{\mathbf{K}} \mathbf{D}^*$$

is a function

$$\bigcup_{A,B \in \mathsf{Ob}(\mathbf{C}^*)} \mathrm{Hom}_{\mathbf{C}^*}(A,B) \times \bigcup_{A,B \in \mathsf{Ob}(\mathbf{D}^*)} \mathrm{Hom}_{\mathbf{D}^*}(A,B) \to \mathbf{K},$$

where the unions are over all the objects in the respective semicategories.

Remark 5.12. In Definition 5.11 we implicitly assume that taking the union over these hom-sets is indeed possible. However, to be strict, that is only possible if **T**, **A**, and **C** are small categories. We will nevertheless ignore such subtleties and will allow ourselves to continue abusing the notation in this way.

Lemma 5.13. *Given two compositional systems* **Alice** *and* **Bob** *and a kinded relation of set* ***K*** *between them, one can construct a kinded function:*

$$R\colon \mathbf{Alice}^* \xrightarrow{K} \mathbf{Bob}^*,$$

or if expanded:

$$R : \bigcup_{A,B\in \mathsf{Ob}(\mathbf{Alice}^*)} \mathrm{Hom}_{\mathbf{Alice}^*}(A,B) \times \bigcup_{A,B\in \mathsf{Ob}(\mathbf{Bob}^*)} \mathrm{Hom}_{\mathbf{Bob}^*}(A,B) \to \boldsymbol{K},$$

such that if A and B are components of respectively **Alice** *and* **Bob** *and α and β are their respective morphisms in* $\mathbf{Alice}^*$ *and* $\mathbf{Bob}^*$*, then $K(A,B) = R(\alpha,\beta)$. Here* $\mathbf{Alice}^*$ *and* $\mathbf{Bob}^*$ *are the semicategorical representations of* **Alice** *and* **Bob**.

5.5 Categorical representations of some compositional systems

In the previous sections we built up to the conclusion that compositional systems can be represented as semicategories and that the relationships between them can be represented as kinded functions. We did that somewhat handwavy by establishing several assumptions that *seemed* to sound right. However, nothing gives us any guarantees that the above conclusion is at all useful. In order to clear up any remaining doubt we will provide a number of examples of collections of components with various degrees of "compositionality", as well as relations between them, and will show that all of them can be represented as semicategories and kinded functions.

The least compositional case is when nothing is composable (as discussed in Remark 5.4). This degenerate setting would correspond to a good old-fashioned set.

Lemma 5.14. *Given a set S, the semicategory* $\mathbf{S}^*$ *with* $\mathsf{Ob}(\mathbf{S}^*) = S$ *and*

$$\mathrm{Hom}_{\mathbf{S}^*}(S_1,S_2) = \begin{cases} \varnothing & \textit{if } S_1 \textit{ is distinct from } S_2, \\ \{S_1\} & \textit{if } S_1 \textit{ is the same as } S_2, \end{cases}$$

for all $S_1, S_2 \in \mathsf{Ob}(\mathbf{S}^*)$ *is indeed a semicategory.*

Proof. First of all, we are working in a setting where the elements of S are components of a compositional system, hence they satisfy Assumption 5.7 and the notions of *sameness* and *distinctness* are well-defined. Second, as there are no morphisms between distinct objects in $\mathbf{S}^*$, there are no compositions, hence nothing to check for associativity. Therefore, it trivially holds that $\mathbf{S}^*$ is indeed a semicategory. □

Remark 5.15. We require that the components of the compositional system become morphisms in the respective semicategory. That is why in the above setting we have each component be not only an object in $\mathbf{S}^*$ but also a morphism.

Remark 5.16. A (semi)category where there are no morphisms between any two distinct objects is usually called a *discrete category*.

Of course, we can treat the terms of a type in the exact same way. Hence, they can also be represented as a discrete semicategory.

Lemma 5.17. *Given a type T, the semicategory $\mathbf{T}^*$ with $\mathsf{Ob}(\mathbf{T}^*)$ being the terms of T and*

$$\mathrm{Hom}_{\mathbf{T}^*}(T_1, T_2) = \begin{cases} \varnothing & \text{if } T_1 \text{ is distinct from } T_2, \\ \{T_1\} & \text{if } T_1 \text{ is the same as } T_2, \end{cases}$$

for all $T_1, T_2 \in \mathsf{Ob}(\mathbf{T}^*)$ is indeed a semicategory.

As a normed type can be represented as a collection of terms of a type and their respective sizes, it is a yet another example of a discrete semicategory.

Lemma 5.18. *Given a normed type $\mathbf{A} := \langle A, |A|, s : A \longrightarrow |A| \rangle$, the semicategory $\mathbf{A}^*$ with $\mathsf{Ob}(\mathbf{A}^*)$ being pairs $\langle a, s(a) \rangle$, with a a term of A and*

$$\mathrm{Hom}_{\mathbf{A}^*}\left(\langle a_1, s(a_1) \rangle, \langle a_2, s(a_2) \rangle\right) = \begin{cases} \varnothing & \text{if } a_1 \text{ is distinct from } a_2, \\ \{\langle a_1, s(a_1) \rangle\} & \text{if } a_1 \text{ is the same as } a_2, \end{cases}$$

for all $a_1 \colon T, a_2 \colon T$ is indeed a semicategory.

So far we considered only compositional systems with no compositionality. However, if we move away from the discrete semicategories to a more general one, in its morphisms we can find the very notion of compositionality we were seeking earlier in the chapter. Of course, needless to say, any semicategory can extremely trivially be represented as a semicategory. But, more interestingly, *any category* can be represented as a semicategory as mentioned in Remark 2.35. This is simply due to the definition of a semicategory being a relaxation of the definition of a category.

$$\begin{tikzcd}
 & & \text{Semicategories} & & \\
\text{Normed types} \arrow[urr] & \text{Types} \arrow[ur] & & & \text{Categories} \arrow[ull] \\
 & & & \text{Sets} \arrow[uul] \arrow[ur] & \text{Symmetric monoidal categories} \arrow[u] \\
 & & & & \text{Compact closed categories} \arrow[u]
\end{tikzcd}$$

Figure 5.1: Hierarchy of compositional systems, i.e. the Ω_S poset.

As any category is a semicategory, also categories with even more structure such as (symmetric) monoidal categories (Definition 2.29) and compact closed categories (Definition 2.32) also represent compositional systems.

In fact, the property of one of these compositional systems being representable in another system can be thought of a poset Ω_S, where $A \leq_{\Omega_S} B$ if any compositional system of the collection A can be represented as a compositional system of the collection B. This is illustrated in Figure 5.1.

Remark 5.19. Upon seeing Figure 5.1 some readers might be eager to say that **Set** (Definition 2.39) is a category, hence "Sets" should be under "Categories". However, in the context of compositional systems we are considering a "set" to be a collection of components, which become the *morphisms* in the representative semicategory. If we instead consider the category **Set**, whose morphisms are *functions*, then the morphisms in the representative semicategory would also be functions. While both settings are certainly valid, they are very different.

5.6 Categorical representations of some kinded relations

In the previous section we demonstrated that a number of widely used compositional systems can be represented as semicategories, hence defending our call for categorical representations for compositional systems from Section 5.4. Here, we continue this argumentation by also illustrating how various types of relationships between such compositional systems can be represented as kinded functions (Lemma 5.13).

Let's first consider compositional systems that were originally sets. One type of a relationship between sets is a function. Assume we have two sets, S and P, and their semicategorical representations $\mathbf{S}^*$ and $\mathbf{P}^*$ due to Lemma 5.14. Take any function $f : S \longrightarrow P$. Then the question is, what should the corresponding kinded function R_f be in order to respect Lemma 5.13.

But then, first, what should the kind $\mathbf{K}$ of this kinded relation be? Clearly, as for any $s \in S$, and any $p \in P$, $f(s)$ is either p or isn't p, the kind set of this relation should have two elements. Then, let's pick $\mathbf{K} = \{\mathtt{T}, \mathtt{F}\}$.

Now, according to Lemma 5.13 the signature of R_f is:

$$R_f : \bigcup_{s,s' \in \mathsf{Ob}(\mathbf{S}^*)} \mathrm{Hom}_{\mathbf{S}^*}(s, s') \times \bigcup_{p,p' \in \mathsf{Ob}(\mathbf{P}^*)} \mathrm{Hom}_{\mathbf{P}^*}(p, p') \longrightarrow \{\mathtt{T}, \mathtt{F}\}.$$

But from Lemma 5.14 we know that these homsets are non-empty only when $s = s'$ or $p = p'$, hence we define R_f as:

$$R_f : \bigcup_{s \in \mathsf{Ob}(\mathbf{S}^*)} \mathrm{Hom}_{\mathbf{S}^*}(s, s) \times \bigcup_{p \in \mathsf{Ob}(\mathbf{P}^*)} \mathrm{Hom}_{\mathbf{P}^*}(p, p) \longrightarrow \{\mathtt{T}, \mathtt{F}\},$$
$$\langle s, p \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } f(s) = p, \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

Hence, we've proven the following lemma:

Lemma 5.20. *For any two sets S and P, and any function $f : S \longrightarrow P$, there exists a kinded function R_f of kind $\{\mathtt{T}, \mathtt{F}\}$ from $\mathbf{S}^*$ to $\mathbf{P}^*$ that satisfies Lemma 5.13.*

Another type of a relation between sets is a binary relation. We already encountered that in the definition of problems in **Lagado** (Definition 4.1). Take again the two sets S and P and their semicategorical representations $\mathbf{S}^*$ and $\mathbf{P}^*$. Now take any binary relation $G \subseteq S \times P$. We can construct a kinded function that represents it:

$$R_G : \bigcup_{s \in \mathsf{Ob}(\mathbf{S}^*)} \mathrm{Hom}_{\mathbf{S}^*}(s, s) \times \bigcup_{p \in \mathsf{Ob}(\mathbf{P}^*)} \mathrm{Hom}_{\mathbf{P}^*}(p, p) \longrightarrow \{\mathtt{T}, \mathtt{F}\},$$
$$\langle s, p \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } \langle s, p \rangle \in G, \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

Hence, we've also proven the following lemma:

Lemma 5.21. *For any two sets S and P, and any binary relation $G \subseteq S \times P$, there exists a kinded function R_G of kind $\{\mathtt{T}, \mathtt{F}\}$ from $\mathbf{S}^*$ to $\mathbf{P}^*$ that satisfies Lemma 5.13.*

Speaking of **Lagado**, we also made use of procedures there (Definition 3.16). A procedure connected two normed types, which we already know to be representable as semicategories (Lemma 5.18). A procedure from a normed type

$$\mathsf{A} \coloneqq \langle A, |A|, s_A \colon A \to |A| \rangle$$

to a normed type

$$\mathsf{B} \coloneqq \langle B, |B|, s_B \colon B \to |B| \rangle$$

is a triplet

$$\mathbf{p} \coloneqq \left\langle p \colon A \to B, \tau_p \colon |A| \to |B|, \rho_p \colon |A| \nrightarrow R \right\rangle,$$

where $p \colon A \to B$ is a term of the function type $A \to B$, $\tau_p \colon |A| \to |B|$ is a monotonic function, and $\rho_p \colon |A| \nrightarrow R$ is a feasibility relation (Definition 2.14), with R a monoidal poset.

Let's see how $\mathbf{p}$ can be represented as a kinded function $R_\mathbf{p}$. First, the morphisms of the categorical representations $\mathbf{A}^*$ and $\mathbf{B}^*$ of the two normed types are pairs $\langle a, s_A(a) \rangle$ and $\langle b, s_B(b) \rangle$, as already established in Lemma 5.18. Hence, given two such terms, we need to evaluate if the procedure $\mathbf{p}$ can produce the second from the first one while obtaining its correct size. Furthermore, this has to be done while keeping track of the resources which enable such conversion. If the conversion is possible, then the resources enabling it form an upper set (Definition 2.12) in R, something which follows from the monotonicity of ρ_p. Following this intuition, we can define $R_\mathbf{p}$ as:

$$\begin{aligned} R_\mathbf{p} \ :\ H_A \ \times\ H_B &\to \{\mathtt{F}\} \cup \mathsf{U}R, \\ \langle \langle a, s_A(a) \rangle, \langle b, s_B(b) \rangle \rangle &\mapsto \begin{cases} \{ r \mid \rho_p(s_A(a), r) = \mathtt{T} \} & \text{if } p(a) = b \text{ and} \\ & \tau_p(s_A(a)) = s_B(b), \\ \mathtt{F} & \text{otherwise}, \end{cases} \end{aligned}$$

with

$$H_A = \bigcup_{a \colon A} \mathrm{Hom}_{\mathbf{A}^*}(\langle a, s_A(a) \rangle, \langle a, s_A(a) \rangle),$$
$$H_B = \bigcup_{b \colon B} \mathrm{Hom}_{\mathbf{B}^*}(\langle b, s_B(b) \rangle, \langle b, s_B(b) \rangle).$$

Note that the kinds of $R_\mathbf{p}$ consist of $\mathtt{F}$ if the relation doesn't hold, and of the upper sets of resources (R) that enable it, in the cases when the relation holds. This is the proof of the following lemma:

Lemma 5.22. *For any two objects* $\mathsf{A}, \mathsf{B} \in \mathrm{Ob}(\mathbf{Proc}(R))$ *and any procedure*

$$\mathbf{p} \in \mathrm{Hom}_{\mathbf{Proc}(R)}(\mathsf{A}, \mathsf{B}),$$

there exists a kinded function $R_\mathbf{p}$ *of kind* $\{\mathtt{F}\} \cup \mathsf{U}R$ *from* $\mathbf{A}^*$ *to* $\mathbf{B}^*$ *that satisfies Lemma 5.13.*

So far we looked only at relations between compositional theories that are represented as discrete semicategories. Let's see what happens when we consider something with more structure, for example categories. The fundamental relation between two categories is a functor.

We will take a look at functors in a bit, but let's first see what happens if our relation is described by a bifunctor to a third category. In a nutshell, every bifunctor corresponds to a kinded function. Then we will show that every functor can be represented as a bifunctor, so then also every functor corresponds to a kinded function too.

Take the bifunctor

$$\Pi : \mathbf{T} \times \mathbf{A} \to \mathbf{C},$$

where **T**, **A** and **C** are all categories. The morphisms of the category **C** will become the kinds of the kinded relation. Then we need to define a kinded function with the following signature:

$$R_\Pi : \bigcup_{t,t' \in \mathrm{Ob}(\mathbf{T})} \mathrm{Hom}_{\mathbf{T}}(t,t') \times \bigcup_{a,a' \in \mathrm{Ob}(\mathbf{A})} \mathrm{Hom}_{\mathbf{A}}(a,a') \to \bigcup_{c,c' \in \mathrm{Ob}(\mathbf{C})} \mathrm{Hom}_{\mathbf{C}}(c,c'). \tag{5.7}$$

However, the fact that Π is a functor means that its effect on morphisms is exactly a function with this signature. Hence, we need not explicitly define a kinded function representation for such a bifunctor: the bifunctor itself can be thought of as a kinded function!

Remark 5.23. In fact, a bifunctor is much stricter than the kinded function. While the kinded function was defined to be as general as possible, the definition of a bifunctor requires that the endpoints of the morphisms are consistently mapped to the endpoints of the maps of the morphisms, as well as that the compositions are mapped to the compositions of the mapped components.

Lemma 5.24. *Every bifunctor*

$$\Pi : \mathbf{T} \times \mathbf{A} \to \mathbf{C}$$

induces a kinded function R_Π *from* **T** *to* **A** *with its kind being the collection of morphisms of* **C**.

Now, let's see what happens with functors. We want to show that every functor can be represented as a bifunctor.

Definition 5.25 (The **And** category)**.** The **And** category consists of two objects T and F with the following morphisms between them:

$$\begin{tikzcd}
\mathtt{T} \arrow[loop above, "\mathtt{t}"] \arrow[loop left, "\mathtt{f}"] \arrow[r, bend left, "\mathtt{f}"] & \mathtt{F} \arrow[l, bend left, "\mathtt{f}"] \arrow[loop right, "\mathtt{f}"]
\end{tikzcd}.$$

The identity morphism of `T` is `t` and of `F` is `f`. Furthermore, we identify the following compositions:

(i) $\mathtt{t} \fatsemi \mathtt{t} = \mathtt{t}$,
(ii) $\mathtt{t} \fatsemi \mathtt{f} = \mathtt{f}$,
(iii) $\mathtt{f} \fatsemi \mathtt{t} = \mathtt{f}$,
(iv) $\mathtt{f} \fatsemi \mathtt{f} = \mathtt{f}$.

Remark 5.26. The name of the **And** category comes from the fact that only a pair of `t` morphisms identifies with `t` and from the fact that the only `t` morphism is the identity on `T`. The use of this structure will become clear shortly.

Lemma 5.27. *Given two categories* **C** *and* **D** *and a functor* $F : \mathbf{C} \to \mathbf{D}$ *between them, the map* $\hat{F}$ *defined as*

$$\begin{aligned} \hat{F} \colon \mathbf{C} \times \mathbf{D} &\to \mathbf{And} \\ \langle C, D \rangle &\mapsto \begin{cases} \mathtt{T} & \textit{if } F(C) = D, \\ \mathtt{F} & \textit{otherwise}, \end{cases} \\ \left\langle \mu, \eta \right\rangle &\mapsto \begin{cases} \mathtt{t} & \textit{if } F(\mu) = \eta, \\ \mathtt{f} & \textit{otherwise}, \end{cases} \end{aligned}$$

is a functor. In the above definition, C and D are objects and μ and η morphisms in respectively **C** *and* **D**.

Proof. First we need to show that the above mapping respects the endpoints of the morphisms. Consider any $\mu : C_1 \to C_2$ in **C** and $\eta : D_1 \to D_2$ in **D**. Note that regardless of where $\hat{F}$ sends μ and η, there is always a `f` morphism between $\hat{F}(C_1, D_1)$ and $\hat{F}(C_2, D_2)$. This is not the case with `t`, so we need to show that whenever $F(\mu) = \eta$, it follows that $F(C_1) = D_1$ and $F(C_2) = D_2$. This, however, must hold as F itself is a functor, hence $\hat{F}$ respects the morphisms' endpoints.

Next, we also need to show that $\hat{F}$ respects identities. From the functoriality of F we know that if $F(C) = D$, then $F\left(\mathrm{id}_C^{\mathbf{C}}\right) = \mathrm{id}_D^{\mathbf{D}}$, hence $\hat{F}\left(\mathrm{id}_C^{\mathbf{C}}, \mathrm{id}_D^{\mathbf{D}}\right) = \mathtt{t}$. If $F(C) = D$, hence also $\hat{F}(C, D) = \mathtt{T}$. And as `t` is the identity of `T`, it respects the identity requirement. In the other case, i.e. if $F(C) \neq D$, $F\left(\mathrm{id}_C^{\mathbf{C}}\right) \neq \mathrm{id}_D^{\mathbf{D}}$, hence $\hat{F}\left(\mathrm{id}_C^{\mathbf{C}}, \mathrm{id}_D^{\mathbf{D}}\right) = \mathtt{f}$. If $F(C) \neq D$, hence also $\hat{F}(C, D) = \mathtt{F}$ and `f` is its identity. □

Remark 5.28. The main implication of the above lemma is that any functor F between the categories **C** and **D** can be represented as a bifunctor $\hat{F}$ between their product category $\mathbf{C} \times \mathbf{D}$ and the category **And**. Hence, a bifunctor $\mathbf{C} \times \mathbf{D} \to \mathbf{And}$ is a generalization of a functor $\mathbf{C} \to \mathbf{D}$. If one is to make an analogy to the world of functions, a function relates to a binary relation in the

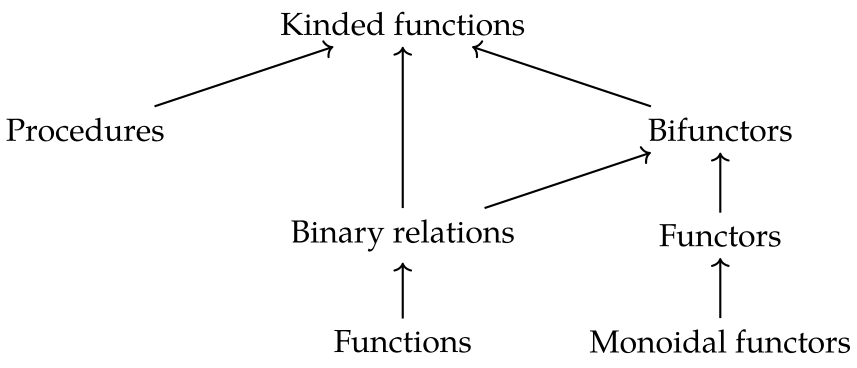


Figure 5.2: Hierarchy of compositional relations, i.e. the Ω_R poset.

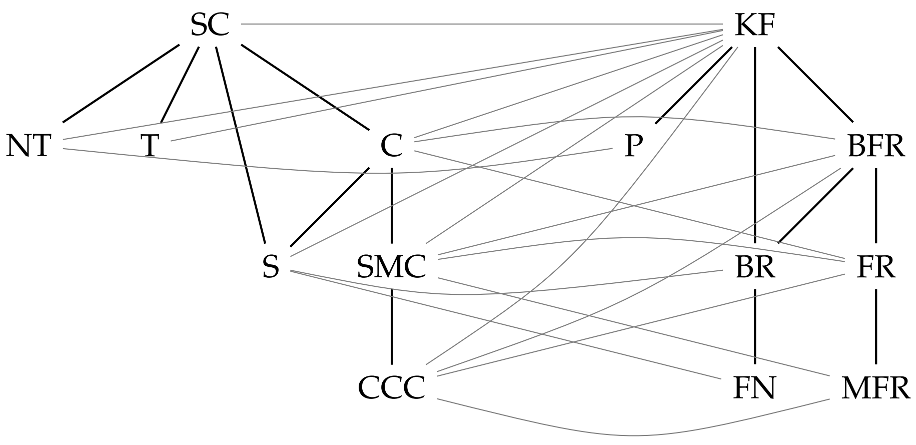


Figure 5.3: Valid relationships between the hierarchies of compositional systems and compositional relations. The acronyms correspond to the collections in Figures 5.1 and 5.2.

same way a functor relates to a bifunctor to **And**. In fact, any binary relation can be represented as a bifunctor to **And**, as the next example shows.

We just showed (Lemma 5.27 and Remark 5.28) that every functor $F : \mathbf{C} \to \mathbf{D}$ can be represented as a bifunctor $\hat{F} : \mathbf{C} \times \mathbf{D} \to \mathbf{And}$. Before, we also showed in Lemma 5.24 that any bifunctor can be represented as a kinded function. Therefore, any functor can then too. The following lemma naturally follows:

Lemma 5.29. *Every functor*

$$F : \mathbf{C} \to \mathbf{D}$$

induces a kinded function R_F from **C** *to* **D** *with its kind being the set* $\{\mathtt{t},\mathtt{f}\}$.

Remark 5.30. The kind of the kinded relation in Lemma 5.29 is simply the collection of morphisms of the **And** category.

Remark 5.31. Actually, we can also show that bifunctors generalize binary relations too! Take any binary relation $R \subseteq C \times D$ between two sets C and D. We can define **C** and **D** to be the discrete categories whose objects are the elements of respectively C and D. Then, the functor $\hat{R} : \mathbf{C} \times \mathbf{D} \to \mathbf{And}$ maps the pair $\langle c, d \rangle$ to `T` if $\langle c, d \rangle \in R$ and to `F` otherwise. As there are only identity morphisms in **C** and **D** they map to the corresponding identity morphisms in **And**. Hence, the bifunctor to **And** can also be thought of as a generalization of a binary relation.

Just in the same way that we showed that some compositional systems include others, it is also true that some relations between compositional systems are generalized by others. Hence, we can create a hierarchy of compositional relations. This again can be represented as a poset Ω_R where $A \leq_{\Omega_R} B$ means that any relation of the collection A can be represented as a relation of the collection B. We illustrate this in Figure 5.2.

Remark 5.32. Not only do compositional systems and relations form two posets, but one can also define a feasibility relation (Definition 2.14) between them. In other words, we can define the monotone function

$$\Omega \colon \Omega_S^{\text{op}} \times \Omega_R \to \texttt{Bool},$$
$$\langle S, R \rangle \mapsto \begin{cases} \texttt{T} & \text{if a relation from the collection } R \text{ can be defined} \\ & \text{for any two compositional systems from } S, \\ \texttt{F} & \text{otherwise.} \end{cases}$$

To see that this is indeed a feasibility relation, we have illustrated the valid pairs in Figure 5.3.

Chapter 6

Compositional computational systems

In the previous chapter we went on a quest for the minimal (and hence most general) way to represent compositional systems and the relationships between them. The result was that compositional systems which fall in the boundaries of the set of assumptions we developed can be modelled as semicategories and the relationships between them as kinded functions. Then, when we looked for examples of such systems and relations, we saw that all of the concepts that we used earlier in this thesis to construct **Lagado** showed up as particular instances of semicategories and kinded functions. But if **Lagado**, a concept that we will call a *compositional computational system*, is build up of particular instances of compositional systems and relations, then is it possible to generalize it by using semicategories and kinded functions as the building elements? That leaves us with a question of creating the minimal (and hence most general) system of problems and solutions, i.e. the "ultimate" generalization of **Lagado**. And this is what we devote this chapter to.

Most readers probably already see how the generalization of a compositional computation system would look like. For the rest, let's spoil it. We already concluded that semicategories and kinded relations should be our fundamental building blocks. Therefore, we will define our problems and solutions as kinded relations between semicategories. Then, this induces categorical structures on problems and solutions, which allow us to define a heteromorphic twisted category on them, just as we did with **Prob**, **Proc**(R) and **Lagado**(R) (recall Lemma 4.21). The resulting category is our answer to the question of generality above, and we will call it **Laputa**.

We will reuse the names of a lot of the previously defined concepts. In the text, it would be clear from the context which one we are using. For the mathematical concepts, in order to distinguish them, we will be putting a small circle over the last letter of the generalization concept.

6.1 Generalized problems and procedures

As we mentioned above, we generalize the definition of a problem (Definition 4.1) to a new one:

Partial Definition 6.1 (Problem)**.** A *problem* Π with a *statement* semicategory $\mathbf{T}^*$, *answer* category $\mathbf{A}^*$, and *kind* the set N is a kinded function of kind N from $\mathbf{T}^*$ to $\mathbf{A}^*$:

$$\Pi\colon \mathbf{T}^* \xrightarrow{N} \mathbf{A}^*.$$

Partial Definition 6.2 (Procedure)**.** A *procedure* Δ with a *statement* semicategory $\mathbf{T}^*$, *answer* category $\mathbf{A}^*$, and *kind* the set M is a kinded function of kind M from $\mathbf{T}^*$ to $\mathbf{A}^*$:

$$\Delta\colon \mathbf{T}^* \xrightarrow{M} \mathbf{A}^*.$$

Remark 6.3. We label the above two definitions *partial* because there is structure on N and M that we require but haven't established yet. This stems from our desire for problems and procedures that are composable. We will soon take a look at this.

Remark 6.4. Apart from that, Definitions 6.1 and 6.2 are exactly the same. They only differ in the name of the kind set but as this is arbitrary, it is not a real difference. Hence, in this generalized setting problems and procedures are modelled in the exact same way. Note however, that we give them different semantic meaning. While we do consider a problem to be simply a map of sorts between statements and answers, we do expect that a procedure has a computational backing, i.e. some algorithm, code, or program that takes an input a statement and produces an answer. Furthermore, we allow different kind sets for the two settings. For the problems, the kind stresses their (potentially) compositional structure. For the procedures, the kind can be containing information of the quality of the approximation or the computational resources needed.

We said above that Definitions 6.1 and 6.2 are *partial* definitions because we need to be stricter in how we pick N and M. Let's see exactly what we meant there.

First, we want to be considering *compositional* problems. Hence let's consider three semicategories and two problems (kinded functions) between them.

$$\mathbf{T_1}^* \xrightarrow{\Pi_A} \mathbf{T_2}^* \xrightarrow{\Pi_B} \mathbf{T_3}^* \tag{6.1}$$

$$\begin{aligned}
\Pi_A \colon & \bigcup_{A,B \in \mathrm{Ob}(\mathbf{T_1}^*)} \mathrm{Hom}_{\mathbf{T_1}^*}(A,B) \times \bigcup_{A,B \in \mathrm{Ob}(\mathbf{T_2}^*)} \mathrm{Hom}_{\mathbf{T_2}^*}(A,B) \to N \\
\Pi_B \colon & \bigcup_{A,B \in \mathrm{Ob}(\mathbf{T_2}^*)} \mathrm{Hom}_{\mathbf{T_2}^*}(A,B) \times \bigcup_{A,B \in \mathrm{Ob}(\mathbf{T_3}^*)} \mathrm{Hom}_{\mathbf{T_3}^*}(A,B) \to N
\end{aligned}$$

If we are to compose the two problems, we should end up with a problem

$$\Pi_{AB} \colon \mathbf{T_1}^* \xrightarrow{N} \mathbf{T_3}^*.$$

This problem takes a morphism t_1 from $\mathbf{T_1}^*$ and a morphism t_3 from $\mathbf{T_3}^*$ and returns an element of N that characterizes the problem. If we are building this from Π_A and Π_B this constrains the domain of Π_A and the range of Π_B but not their interaction at $\mathbf{T_3}^*$. Hence, we need to "integrate out" the $\mathbf{T_2}^*$.

Let's first fix a single morphism t_2' from $\mathbf{T_2}^*$. Then we can obtain $\Pi_A(t_1, t_2') = n_A \in N$ and $\Pi_B(t_2', t_3) = n_B \in N$. However, in order to obtain a single element of N which characterizes the composition of Π_A and Π_B we need a way of "combining" n_A and n_B. As this must hold for any $n_A, n_B \in N$ it can be thought of as a binary operation. We will call it *multiplication* and denote it by $\times$. Now, we can combine n_A and n_B as $n_A \times n_B \in N$.

Imagine we have three problems instead:

$$\mathbf{T_1}^* \xrightarrow{\Pi_A} \mathbf{T_2}^* \xrightarrow{\Pi_B} \mathbf{T_3}^* \xrightarrow{\Pi_C} \mathbf{T_4}^* \ , \tag{6.2}$$

and we have fixed morphisms t_2' from $\mathbf{T_2}^*$ and t_3' from $\mathbf{T_3}^*$. Then, for any t_1 from $\mathbf{T_1}^*$ and t_4 from $\mathbf{T_4}^*$ we can write the multiplication of the respective kinds in two ways:

$$\begin{gathered}
\left(\Pi(t_1, t_2') \times \Pi(t_2', t_3')\right) \times \Pi(t_3', t_4), \\
\Pi(t_1, t_2') \times \left(\Pi(t_2', t_3') \times \Pi(t_3', t_4)\right).
\end{gathered}$$

However, it doesn't make sense that the order of multiplication of the kinds would affect the resulting kind of the composition. Hence we would like to have:

$$\begin{aligned}
\left(\Pi(t_1, t_2') \times \Pi(t_2', t_3')\right) \times \Pi(t_3', t_4) &= \Pi(t_1, t_2') \times \left(\Pi(t_2', t_3') \times \Pi(t_3', t_4)\right) \\
&= \Pi(t_1, t_2') \times \Pi(t_2', t_3') \times \Pi(t_3', t_4).
\end{aligned}$$

Therefore, the multiplication operation $\times$ over N should be associative.

Now, let's get back to composing the two problems in Equation (6.1). Recall that so far we had the morphism at $\mathbf{T_2}^*$ fixed as some particular value. To obtain Π_{AB} we need to evaluate what happens with the value of

$$\Pi_A(t_1, t_2') \times \Pi_B(t_2', t_3)$$

over the whole range of morphisms t_2' in $\mathbf{T_2}^*$. To this end, we can define a new binary operation over N, called *addition*, which will be denoted by + (or $\sum$).

Now, we can define the composition of problems (or any kinded functions) as:

$$\Pi_{AB}(t_1, t_3) = \left(\Pi_A \fatsemi \Pi_B\right)(t_1, t_3) = \sum_{t_2 \in H_2} \Pi_A(t_1, t_2) \times \Pi_B(t_2, t_3),$$

where

$$H_2 = \bigcup_{A, B \in \mathsf{Ob}(\mathbf{T_2}^*)} \mathrm{Hom}_{\mathbf{T_2}^*}(A, B).$$

It of course makes sense for the addition on N to also be associative. Furthermore, we don't want to have the value of $\Pi(t_1, t_3)$ depending on the order of going through the morphisms in H_2. To ensure that we will also require that the addition operation on N is commutative.

Now that we have defined how the composition should look like, let's take another look at the example where we compose three problems (Equation (6.2)). Then we would get:

$$\begin{aligned} \Pi_{ABC} \colon H_1 \times H_4 &\longrightarrow N \\ \langle t_1, t_4 \rangle &\mapsto \sum_{t_2 \in H_2} \left(\Pi_A(t_1, t_2) \times \sum_{t_3 \in H_3} \Pi_B(t_2, t_3) \times \Pi_C(t_3, t_4) \right), \end{aligned} \tag{6.3}$$

where H_1, H_2, H_3 and H_4 as defined analogously to above. However, we would like the kind of the composition to not change if we instead consider summation over the multiplication of three kinds (recall that the multiplication was defined to be associative). I.e. we want that the term in Equation (6.3) to be equal to:

$$\sum_{t_2 \in H_2} \sum_{t_3 \in H_3} \Pi_A(t_1, t_2) \times \Pi_B(t_2, t_3) \times \Pi_C(t_3, t_4).$$

This is equivalent to asking that the multiplication operator $\times$ distributes over the addition operator +.

Recall that in **Lagado** we also had a notion of identity problem which actually represented a solution. That was because both the statement and answer types of such a problem are the same and the map between them is the identity function. Hence the statement and the answer are always equal, making it a problem that is very trivial to solve. We would like to extend this concept to the general setting we are constructing. So, we will also need identity problems.

An identity problem should have the property that when composed with another problem, it doesn't change it. Let's consider the following setting:

$$\begin{tikzcd} \mathbf{T_1}^* \arrow[r, "\Pi_A"] & \mathbf{T_2}^* \arrow[loop right, "\Pi_2^{\mathrm{id}}"] \end{tikzcd} . \tag{6.4}$$

We ask that:

$$\sum_{t_2' \in H_2} \Pi_A(t_1, t_2') \times \Pi_2^{\mathrm{id}}(t_2', t_2) = \Pi_A(t_1, t_2), \quad \forall t_1 \in H_1, \forall t_2 \in H_2.$$

The left-hand side of the above equation can be rewritten as:

$$\Pi_A(t_1, t_2) \times \Pi_2^{\mathrm{id}}(t_2, t_2) \; + \sum_{t_2' \in H_2 \setminus \{t_2\}} \Pi_A(t_1, t_2') \times \Pi_2^{\mathrm{id}}(t_2', t_2).$$

The desired result would be achieved if $\Pi_2^{\mathrm{id}}(t_2, t_2)$ evaluates to a multiplicative identity on N and

$$\sum_{t_2' \in H_2 \setminus \{t_2\}} \Pi_A(t_1, t_2') \times \Pi_2^{\mathrm{id}}(t_2', t_2)$$

evaluates to an additive identity. The second essentially means that multiplication by the additive identity should annihilate any value (i.e. should result to the additive identity itself).

Let's summarize all the properties that we established to be desirable in the kind N:

i. N is equipped with a binary operation $\times : N \times N \longrightarrow N$.
ii. The binary operation $\times$ is associative.
iii. The binary operation $\times$ has an identity element.
iv. N is equipped with a binary operation $+ : N \times N \longrightarrow N$.
v. The binary operation $+$ is associative.
vi. The binary operation $+$ is commutative.
vii. The binary operation $+$ has an identity element.
viii. The binary operation $\times$ distributes over the binary operation $+$.

It just so happens that this is the definition of a *rig*.

Definition 6.5 (Rig). A *rig*, also called *semiring*, is a set R equipped with two binary operations (Definition 2.1), addition ($+$) and multiplication ($\times$) such that:

i. The addition operation:
   - is associative, i.e.

$$(a + b) + c = a + (b + c), \quad \forall a, b, c \in R;$$

- is commutative, i.e.

$$a + b = b + a, \quad \forall a, b \in R;$$

- has an identity element denoted 0, i.e.

$$0 + a = a + 0 = a, \quad \forall a \in R;$$

ii. The multiplication operation:
- is associative, i.e.

$$(a \times b) \times c = a \times (b \times c) \quad \forall a, b, c \in R;$$

- has an identity element denoted 1, i.e.

$$1 \times a = a \times 1 = a, \quad \forall a \in R;$$

iii. The multiplication operation distributes over the addition operation, i.e. for all $a, b, c \in R$ it holds that

$$a \times (b + c) = (a \times b) + (a \times c)$$
$$(a + b) \times c = (a \times c) + (b \times c);$$

iv. The addition identity 0 is the annihilating element of R with respect to the multiplication operation:

$$0 \times a = a \times 0 = 0, \quad \forall a \in R.$$

A rig can be denoted by the tuple $\langle R, +, 0, \times, 1 \rangle$.

Clearly, everything we asked for N leads to it being a rig. Moreover, as we mentioned in Remark 6.4, apart from the symbol we use for the kind the two partial definitions for problems and procedures (Definitions 6.1 and 6.2) are exactly the same. Hence, if we wish to have identities and the same compositional properties on the procedures, we also need the structure of a rig on M.

With this we can now give the complete definitions of problems and procedures, as well as formal definitions of composition and identity.

Definition 6.6 (Problem)**.** Given a rig

$$N := \langle N, +_N, 0_N, \times_N, 1_N \rangle\,,$$

a *problem* Π with a *statement* semicategory $\mathbf{T}^*$, *answer* semicategory $\mathbf{A}^*$, and *kind* N is a kinded function of kind N from $\mathbf{T}^*$ to $\mathbf{A}^*$:

$$\Pi\colon \mathbf{T}^* \xrightarrow{N} \mathbf{A}^*.$$

Definition 6.7 (Procedure)**.** Given a rig

$$M := \langle M, +_M, 0_M, \times_M, 1_M \rangle,$$

a *procedure* Δ with a *statement* semicategory $\mathbf{T}^*$, *answer* semicategory $\mathbf{A}^*$, and *kind* M is a kinded function of kind M from $\mathbf{T}^*$ to $\mathbf{A}^*$:

$$\Delta\colon \mathbf{T}^* \xrightarrow{M} \mathbf{A}^*.$$

Definition 6.8 (Problem composition)**.** Given a rig

$$N := \langle N, +_N, 0_N, \times_N, 1_N \rangle,$$

and two problems:

$$\Pi_A\colon \mathbf{T_1}^* \xrightarrow{N} \mathbf{T_2}^* \quad \text{and} \quad \Pi_B\colon \mathbf{T_2}^* \xrightarrow{N} \mathbf{T_3}^*,$$

we define the composition $\Pi_A \mathbin{⨟} \Pi_B$ as:

$$\begin{aligned} \Pi_A \mathbin{⨟} \Pi_B\colon H_1 \times H_3 &\longrightarrow N. \\ \langle t_1, t_3 \rangle &\mapsto \sum_{t_2 \in H_2} \Pi_A(t_1, t_2) \times_N \Pi_B(t_2, t_3), \end{aligned}$$

with $\sum$ being the repeated application of $+_N$, and H_1, H_2, H_3 being the collections of morphisms of respectively $\mathbf{T_1}^*$, $\mathbf{T_2}^*$, $\mathbf{T_3}^*$.

Definition 6.9 (Procedure composition)**.** Analogous to Definition 6.8.

Definition 6.10 (Identity problem)**.** Given a rig

$$N := \langle N, +_N, 0_N, \times_N, 1_N \rangle,$$

and a semicategory $\mathbf{T}^*$, we define the identity problem $\Pi^{\mathrm{id}}_{\mathbf{T}^*}: \mathbf{T}^* \xrightarrow{N} \mathbf{T}^*$ on $\mathbf{T}^*$ as

$$\begin{aligned} \Pi^{\mathrm{id}}_{\mathbf{T}^*}\colon H \times H &\longrightarrow N, \\ \langle t, t' \rangle &\mapsto \begin{cases} 1_N & \text{if } t = t', \\ 0_N & \text{otherwise,} \end{cases} \end{aligned}$$

with H being the collection of morphisms of $\mathbf{T}^*$.

Definition 6.11 (Identity procedure)**.** Analogous to Definition 6.10.

Of course, we need to show that the identity problem indeed acts as an identity:

Lemma 6.12. *Given a rig*

$$N := \langle N, +_N, 0_N, \times_N, 1_N \rangle ,$$

and a problem:

$$\Pi \colon \mathbf{T_1}^* \xrightarrow{N} \mathbf{T_2}^*,$$

it holds that

$$\Pi \mathbin{⨟} \Pi^{\mathrm{id}}_{\mathbf{T_2}^*} = \Pi \quad \textit{and} \quad \Pi^{\mathrm{id}}_{\mathbf{T_1}^*} \mathbin{⨟} \Pi = \Pi.$$

Proof. The left-hand side of the first equation is:

$$\begin{aligned}
&\left(\Pi \mathbin{⨟} \Pi^{\mathrm{id}}_{\mathbf{T_2}^*}\right)(t_1, t_2)\\
&= \sum_{t_2' \in H_2} \Pi(t_1, t_2') \times_N \Pi^{\mathrm{id}}_{\mathbf{T_2}^*}(t_2', t_2)\\
&= \left(\Pi(t_1, t_2) \times_N \Pi^{\mathrm{id}}_{\mathbf{T_2}^*}(t_2, t_2)\right) +_N \sum_{t_2' \in H_2 \setminus \{t_2\}} \Pi(t_1, t_2') \times_N \Pi^{\mathrm{id}}_{\mathbf{T_2}^*}(t_2', t_2)\\
&= (\Pi(t_1, t_2) \times_N 1_N) +_N \sum_{t_2' \in H_2 \setminus \{t_2\}} \Pi(t_1, t_2') \times_N 0_N\\
&= \Pi(t_1, t_2) +_N \sum_{t_2' \in H_2 \setminus \{t_2\}} 0_N\\
&= \Pi(t_1, t_2) +_N 0_N\\
&= \Pi(t_1, t_2),
\end{aligned}$$

which is the right-hand side. Hence, the first equation holds. Above we used the properties of the additive and multiplicative identities of a rig, as well as the fact that the additive identity 0_N is the annihilating element of N with respect to $\times_N$ (Definition 6.5). The second equation can be shown in an analogous way. □

And the case for the identity procedure is exactly the same.

Lemma 6.13. *Given a rig*

$$M := \langle M, +_M, 0_M, \times_M, 1_M \rangle ,$$

and a procedure:

$$\Delta \colon \mathbf{T_1}^* \xrightarrow{M} \mathbf{T_2}^*,$$

it holds that

$$\Delta \mathbin{⨟} \Delta^{\mathrm{id}}_{\mathbf{T_2}^*} = \Delta \quad \textit{and} \quad \Delta^{\mathrm{id}}_{\mathbf{T_1}^*} \mathbin{⨟} \Delta = \Delta.$$

6.2 Laputa

We just showed that problems and procedures are kinded functions between semicategories with kinds having the structure of rigs. Furthermore, we showed how this structure implies that we can compose these kinded functions and that they also have identities. We have seen a mathematical structure before that consists of objects, things between them, means of composing these things, and identities: a category. Indeed, collections of problems and procedures can be thought of as two categories.

Definition 6.14 (The $\mathbf{Pro\mathring{b}}(\mathbf{O}^*, N)$ category)**.** Given a rig

$$N = \langle N, +_N, 0_N, \times_N, 1_N \rangle ,$$

and a subcategory $\mathbf{O}^*$ (Definition 2.21) of **SemiCat** (Definition 2.38), the $\mathbf{Pro\mathring{b}}(\mathbf{O}^*, N)$ category has:

i. objects which are the objects of $\mathbf{O}^*$;
ii. morphisms $\mathbf{A}^* \to \mathbf{B}^*$ which are problems with statement semicategory $\mathbf{A}^*$, answer semicategory $\mathbf{B}^*$, and kind N;
iii. composition of two morphisms defined by problem composition (Definition 6.6);
iv. identity morphisms being the identity problems (Definition 6.10).

Definition 6.15 (The $\mathbf{Pro\mathring{c}}(\mathbf{O}^*, M)$ category)**.** Given a rig

$$M = \langle M, +_M, 0_M, \times_M, 1_M \rangle ,$$

and a subcategory $\mathbf{O}^*$ of **SemiCat**, the $\mathbf{Pro\mathring{c}}(\mathbf{O}^*, M)$ category has:

i. objects which are the objects of $\mathbf{O}^*$;
ii. morphisms $\mathbf{A}^* \to \mathbf{B}^*$ which are procedures with statement semicategory $\mathbf{A}^*$, answer category $\mathbf{B}^*$, and kind M;
iii. composition of two morphisms defined by procedure composition (Definition 6.9);
iv. identity morphisms being the identity procedures (Definition 6.11).

In the **Lagado** setting we also had a notion of *solution* (Definition 4.5). We labeled a given procedure as a *solution* of a problem if it always produced a correct answer. In the generalized setting we're considering here we are moving away from "correctness" and instead allow to label problems and procedures with kinds. Hence, we also need to generalize the notion of a solution. In fact, we will abstract the decision of whether a certain procedure is a solution to a problem in a higher-order function

$$\Psi_{\mathbf{T}^*, \mathbf{A}^*} \colon \left(\mathbf{T}^* \xrightarrow{N} \mathbf{A}^* \right) \times \left(\mathbf{T}^* \xrightarrow{M} \mathbf{A}^* \right) \to \mathtt{Bool}.$$

We will call Ψ a *solution judgement map*.

Definition 6.16 (Solution)**.** Given two rigs N and M, and function Ψ indexed by two semicategories:

$$\Psi_{\mathbf{T}^*,\mathbf{A}^*}\colon \left(\mathbf{T}^* \xrightarrow{N} \mathbf{A}^*\right) \times \left(\mathbf{T}^* \xrightarrow{M} \mathbf{A}^*\right) \to \mathtt{Bool},$$

a *solution* of a problem $\Pi : \mathbf{T}^* \xrightarrow{N} \mathbf{A}^*$ is a procedure $\Delta : \mathbf{T}^* \xrightarrow{M} \mathbf{A}^*$, such that

$$\Psi_{\mathbf{T}^*,\mathbf{A}^*}(\Pi, \Delta) = \mathtt{T}.$$

Now that we have cleared up that problems and procedures form categories, and that we have means of assessing when a procedure is a solution to a problem, we are ready to define the category **Laputa** which represents our general compositional computational system.

Definition 6.17 (The **Laputa**[1] Category)**.** Take a subcategory $\mathbf{O}^*$ of **SemiCat**, two rigs N and M, a solution judgement map Ψ, and the family of functions $\square$ such that for any four semicategories $\mathbf{A}^*, \mathbf{B}^*, \mathbf{C}^*, \mathbf{D}^* \in \mathsf{Ob}(\mathbf{O}^*)$ we have:

$$\begin{aligned} {}^{\mathbf{A}^*}_{\mathbf{B}^*}\square^{\mathbf{C}^*}_{\mathbf{D}^*} &: \mathring{\mathbf{C}}(\mathbf{C}^*, \mathbf{A}^*) \times \mathring{\mathbf{B}}(\mathbf{A}^*, \mathbf{B}^*) \times \mathring{\mathbf{C}}(\mathbf{B}^*, \mathbf{D}^*) \times \mathring{\mathbf{B}}(\mathbf{C}^*, \mathbf{D}^*) \to \mathtt{Bool}, \\ \left\langle \Delta_p, \Pi_f, \Delta_q, \Pi_g \right\rangle &\mapsto \begin{cases} \mathtt{T} & \text{if } \forall \Delta_h \in \mathrm{Hom}_{\mathbf{Pro}\mathring{\mathbf{c}}(M)}(\mathbf{A}^*, \mathbf{B}^*), \\ & \text{s.t. } \Psi_{\mathbf{A}^*,\mathbf{B}^*}\left(\Pi_f, \Delta_h\right) = \mathtt{T} \\ & \text{it holds that } \Psi_{\mathbf{C}^*,\mathbf{D}^*}\left(\Pi_g, \Delta_p \fatsemi \Delta_h \fatsemi \Delta_q\right) = \mathtt{T}, \\ \mathtt{F} & \text{otherwise,} \end{cases} \end{aligned}$$

where $\mathring{\mathbf{C}} = \mathrm{Hom}_{\mathbf{Pro}\mathring{\mathbf{c}}(M)}$ and $\mathring{\mathbf{B}} = \mathrm{Hom}_{\mathbf{Pro}\mathring{\mathbf{b}}(N)}$. Then, the $\mathbf{Laputa}(\mathbf{O}^*, N, M, \Psi)$ category is the heteromorphic twisted category

$$\mathbf{HTw}(\mathbf{Pro}\mathring{\mathbf{b}}(\mathbf{O}^*, N), \mathbf{Pro}\mathring{\mathbf{c}}(\mathbf{O}^*, M), \square).$$

Heteromorphic twisted categories were introduced in Definition 4.16.

Remark 6.18. As for any selection of $\mathbf{O}^*$, N, M, and Ψ the resulting

$$\mathbf{L}_1 = \mathbf{Laputa}(\mathbf{O}^*, N, M, \Psi)$$

is a category, this **Laputa** category can be a problem statement or an answer in another **Laputa** category that has as objects categories. In other words $\mathbf{L}_1$ can be a problem statement or an answer in any $\mathbf{L}_2 = \mathbf{Laputa}(\mathbf{Cat}, \cdot, \cdot, \cdot)$ or any $\mathbf{L}_3 = \mathbf{Laputa}(\mathbf{SemiCat}, \cdot, \cdot, \cdot)$ category. It is even more curious that as $\mathbf{L}_2$ and $\mathbf{L}_3$ are themselves categories, they can also be problems statements and answers in themselves. This self-referentiality allows some of the compositional computational systems, such as any such $\mathbf{L}_2$ or $\mathbf{L}_3$ to have an "internal model" of themselves and to reason about their own structure. Note that in such a case a problem statement or an answer would correspond to a pair of procedures of the same category.

[1] Named after Laputa, the island floating above the island of Balnibarbi in *Gulliver's Travels* by Jonathan Swift (1726). In the novel, Balnibarbi is subjugated to the king of Laputa.

6.3 Lagado and Laputa

We deliberately constructed **Laputa**$(\mathbf{O}^*, N, M, \Psi)$ to be the most general category for handling generalizations of problems, procedures and solutions. But then, if **Laputa** is so general, then our original problem-solution category **Lagado** should be nothing but a specific instance of it. In this section we aim to show that this is indeed the case. All we need to do is to show that given a monoidal poset R, there is a choice of $\mathbf{O}^{*\mathbf{Lagado}}$, rigs $N^{\mathbf{Lagado}}$ and $M^{\mathbf{Lagado}}$, and a solution judgement map $\Psi^{\mathbf{Lagado}}$, for which **Lagado**(R) is equivalent to

$$\mathbf{Laputa}(\mathbf{O}^{*\mathbf{Lagado}}, N^{\mathbf{Lagado}}, M^{\mathbf{Lagado}}, \Psi^{\mathbf{Lagado}}).$$

What should $\mathbf{O}^{*\mathbf{Lagado}}$ be then? Well, while all normed types can be considered as semicategories (Lemma 5.18), not all semicategories are normed types. Hence, we will take $\mathbf{O}^{*\mathbf{Lagado}}$ to be the subcategory of semicategories which represent normed types.

Now, what should the rig N be so that $\mathring{\mathbf{Prob}}(\mathbf{O}^{*\mathbf{Lagado}}, N)$ is **Prob**. A problem is nothing but a binary relation (Definition 4.1) and we already showed that binary relations are kinded functions with kind `Bool` = $\{$`T`$,$ `F`$\}$. Now, we only need to add the additive and multiplicative structure on `Bool` that results in the composition and identities from Definition 4.3.

Recall that problem composition was defined in Definition 4.3 as:

$$\bigvee_{b \in B} \pi_1(a, b) \wedge \pi_2(b, c).$$

And problem composition was defined in Definition 6.8 as:

$$\sum_{t_2 \in H_2} \Pi_A(t_1, t_2) \times_N \Pi_B(t_2, t_3).$$

Hence, the addition operation should be $\vee$ and the multiplication operation should be $\wedge$. As an additive identity we need an element $X \in \{\mathtt{T}, \mathtt{F}\}$ such that:

$$X \vee \mathtt{T} = \mathtt{T},$$
$$X \vee \mathtt{F} = \mathtt{F},$$

hence, X must be `F`. Similarly, as a multiplicative identity we need an element $X \in \{\mathtt{T}, \mathtt{F}\}$ such that:

$$X \wedge \mathtt{T} = \mathtt{T},$$
$$X \wedge \mathtt{F} = \mathtt{F},$$

hence, X must be `T`. And it is now also easy to check that the additive identity is the annihilating element of the multiplication operation:

$$\mathtt{F} \wedge \mathtt{T} = \mathtt{F},$$

$$\mathtt{F} \land \mathtt{F} = \mathtt{F}.$$

Therefore, we have $N^{\textbf{Lagado}} := \langle \mathtt{Bool}, \lor, \mathtt{F}, \land, \mathtt{T} \rangle$.

Now, let's see what the rig M should be so that $\mathbf{Pro\mathring{c}}(\mathbf{O}^{*\textbf{Lagado}}, M)$ is $\mathbf{Proc}(R)$ for some monoidal poset $R = \langle R, \leq_R, 0_R, +_R \rangle$. We already showed how procedures (Definition 3.16) can be represented as semicategories (Lemma 5.22). Recall that the kind of the corresponding kinded function is $\{\mathtt{F}\} \cup \mathsf{U}R$. Hence we also pick $\{\mathtt{F}\} \cup \mathsf{U}R$ as our set underlying the $M^{\textbf{Lagado}}$ rig.

Take three normed types $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$. Each one of these has a corresponding category $\mathring{\mathbf{A}}$, $\mathring{\mathbf{B}}$, $\mathring{\mathbf{C}}$, with both objects and morphisms being pairs $\langle a, s_a \rangle$, where $s_a = s_A(a)$ is the size of the term a. Take also two procedures between them:

$$\Delta_1 \colon \mathring{\mathbf{A}} \xrightarrow{M^{\textbf{Lagado}}} \mathring{\mathbf{B}},$$
$$\Delta_2 \colon \mathring{\mathbf{B}} \xrightarrow{M^{\textbf{Lagado}}} \mathring{\mathbf{C}}.$$

Then, their composite is:

$$\begin{aligned} &\Delta_1 ⨟ \Delta_2 \colon H_{\mathring{\mathbf{A}}} \times H_{\mathring{\mathbf{C}}} \to \{\mathtt{F}\} \cup \mathsf{U}R, \\ &\quad \langle \langle a, s_a \rangle, \langle c, s_c \rangle \rangle \mapsto \sum_{b \colon B} \Delta_1 (\langle a, s_a \rangle, \langle b, s_B(b) \rangle) \times_M \Delta_2 (\langle b, s_B(b) \rangle, \langle c, s_C(c) \rangle). \end{aligned}$$

We achieve this by defining $\times_M$ as:

$$\begin{aligned} \times_M \colon \quad M \times M &\to M, \\ \langle \mathtt{F}, \mathtt{F} \rangle &\mapsto \mathtt{F}, \\ \langle R', \mathtt{F} \rangle &\mapsto \mathtt{F}, \\ \langle \mathtt{F}, R' \rangle &\mapsto \mathtt{F}, \\ \langle R_1, R_2 \rangle &\mapsto \{ r_1 +_R r_2 \mid r_1 \in R_1, r_2 \in R_2 \}, \end{aligned}$$

for all $R', R_1, R_2 \in \mathsf{U}R$, and with multiplicative identity $1_M = R$. And by defining $+_M$ as:

$$\begin{aligned} +_M \colon \quad M \times M &\to M, \\ \langle \mathtt{F}, \mathtt{F} \rangle &\mapsto \mathtt{F}, \\ \langle R', \mathtt{F} \rangle &\mapsto R', \\ \langle \mathtt{F}, R' \rangle &\mapsto R', \\ \langle R_1, R_2 \rangle &\mapsto \{ r_1 +_R r_2 \mid r_1 \in R_1, r_2 \in R_2 \}, \end{aligned}$$

for all $R', R_1, R_2 \in \mathsf{U}R$, and with additive identity $0_M = \mathtt{F}$. One can easily verify that procedure composition and identity procedures following from the above definitions are exactly the same as the ones in Definition 3.16. Also, the additive identity F is indeed the annihilating element with respect to the multiplication operation. Therefore, we have

$$M^{\mathbf{Lagado}} := \langle \{\mathtt{F}\} \cup \mathsf{U}R, +_M, \mathtt{F}, \times_M, R \rangle .$$

The only thing left is to provide an appropriate solution judgement map $\Psi^{\mathbf{Lagado}}$. We already know that the resultant family of functions $\Box^{\mathbf{Lagado}}$ should be the same as the one in Lemma 4.21. Hence, we get that $\Psi^{\mathbf{Lagado}}$ should be:

$$\Psi^{\mathbf{Lagado}}_{\mathsf{T},\mathsf{A}} : \left(\mathsf{T} \xrightarrow{N^{\mathbf{Lagado}}} \mathsf{A}\right) \times \left(\mathsf{T} \xrightarrow{M^{\mathbf{Lagado}}} \mathsf{A}\right) \to \mathtt{Bool},$$

$$\langle \Pi, \Delta \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } \Delta(t,a) \neq \mathtt{F} \implies \Pi(t, \Delta(t,a)) = \mathtt{T}, \\ & \forall t : T, \forall a : A, \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

Therefore,

$$\mathbf{Laputa}\left(\mathbf{O}^{*\mathbf{Lagado}}, N^{\mathbf{Lagado}}, M^{\mathbf{Lagado}}, \Psi^{\mathbf{Lagado}}\right)$$

is indeed the category **Lagado**(R) as defined before in Definition 4.7.

6.4 Functorial problems and procedures

The semicategories that form the objects of $\mathbf{O}^{*\mathbf{Lagado}}$ are discrete semicategories. That means that there's not much interaction between their elements. Let's then see an example where the relation is functorial. That means that we have compositional properties not only between problems and between procedures but also within the statements and answers themselves.

To be more concrete, we will be asking that our statements and answers form categories and that problems and procedures are functors between them. We already saw in Lemma 5.27 that we can represent any functor $\Pi : \mathbf{C} \to \mathbf{D}$ as a bifunctor $\mathbf{C} \times \mathbf{D} \to \mathbf{And}$. As part of this realization we also concluded that it can then be represented as a kinded function with kind being the set $\{\mathtt{t}, \mathtt{f}\}$ (Lemma 5.29).

Note that as $\mathbf{C} \times \mathbf{D} \to \mathbf{And}$ generalizes also binary relations, we get even more structure. The functor Π maps an object C from $\mathbf{C}$ to only one and exactly one object in $\mathbf{D}$. The same holds for morphisms too. However, $\mathbf{C} \times \mathbf{D} \to \mathbf{And}$ can map C to any number of objects in $\mathbf{D}$ (or to no object too). We will use that property to handle problems that can have varying number of solutions.

Remark 6.19. It is immediately obvious how also any problem $\pi : \mathsf{T} \to \mathsf{A}$ as per Definition 4.1 can be represented in the alternative form $\mathbf{T} \times \mathbf{A} \to \mathbf{And}$. The category $\mathbf{T}$ will have an object $\langle t, s_T(t) \rangle$, a pair of element and its size, for every element t of the normed type **T**. The category **A** is similarly defined. Then $\Pi\left(\langle t, s_T(t)\rangle, \langle a, s_A(a)\rangle\right) = \mathtt{T}$ iff $\pi(t,a) = \mathtt{T}$. Hence, the definition of problems in this section encompasses the original one that we saw in the context of **Prob** and **Lagado**(R).

So how would this compositional computational system look like? First, let's call it **LAnd**. We want to deal with categories only, so then we are interested only in the objects of **Cat**, the category of (small) categories (Definition 2.37). It is trivial to see that **Cat** is indeed a subcategory of **SemiCat**. As we take both problems and procedures to be functors (or bifunctors to **And**), we know that the sets underlying our two rigs N and M should both be $\{\mathtt{t},\mathtt{f}\}$. Note that despite the difference in capitalization, we give the same semantic meaning to $\{\mathtt{t},\mathtt{f}\}$ and $\{\mathtt{T},\mathtt{F}\} = \mathtt{Bool}$. Hence, the rig $N^{\mathbf{Lagado}}$ also can apply here. It also carries the meaning that we wish: to work as functor composition. Therefore, we define both the problem and procedure rigs to be:

$$B \coloneqq \langle \{\mathtt{t},\mathtt{f}\}, \vee, \mathtt{F}, \wedge, \mathtt{T}\rangle\,.$$

And we can define a simple solution judgement map for any two categories **T** and **A**:

$$\begin{aligned}\Psi^{\mathbf{LAnd}}_{\mathbf{T},\mathbf{A}} \colon \left(\mathbf{T} \xrightarrow{B} \mathbf{A}\right) \times \left(\mathbf{T} \xrightarrow{B} \mathbf{A}\right) &\to \mathtt{Bool}, \\ \langle \Pi, \Delta\rangle \mapsto \bigwedge_{T\in\mathsf{Ob}(\mathbf{T})} \bigwedge_{A\in\mathsf{Ob}(\mathbf{A})} &\Delta(T,A) \implies \Pi(T,A).\end{aligned} \tag{6.5}$$

This behavior $\Psi^{\mathbf{LAnd}}$ is to simply check whether every time that the procedure says that a statement and an answer match, they also match according to the map of the problem.

Now we can define the **LAnd** category as:

$$\mathbf{LAnd} \coloneqq \mathbf{Laputa}\left(\mathbf{Cat}, B, B, \Psi^{\mathbf{And}}\right).$$

Let's see an example of a problem in this setting.

Example 6.20 (Linear systems)**.** As an illustration of a problem with more structure, let's consider solving systems of linear equations, that is problems of the type:

$$Ax = a,$$

where $A \in \mathbb{R}^{k\times l}$, $a \in \mathbb{R}^k$ are given, and we solve for $x \in \mathbb{R}^l$. More precisely, we'll consider a slight twist of this problem, namely, given a matrix A, find a function that maps every a to an x such that $Ax = a$. Note that depending on A, such a function might not exist (the inconsistent case), or might not be unique (the indeterminate case).

Our statement category will be the category of matrices **M** defined as follows. The objects of **M** are the (positive) natural numbers. A morphism from an object k to an object l is then a matrix $A \in \mathbb{R}^{k \times l}$. The identity morphism for the object k is the identity matrix I_k. Morphism composition in **M** is simply matrix multiplication.

The answer category **N** also has as objects the (positive) natural numbers. A morphism from an object k to an object l in **N** is a function $f : \mathbb{R}^k \to \mathbb{R}^l$. Composing two morphisms $f : \mathbb{R}^k \to \mathbb{R}^l$ from k to l and $g : \mathbb{R}^l \to \mathbb{R}^m$ from l to m is the function composition $f \fatsemi g$ from k to m. Identity morphism for the object k is the identity function $\mathrm{id}_{\mathbb{R}^k}$.

Now we need to provide the problem functor $\Pi^L : \mathbf{M} \times \mathbf{N} \to \mathbf{And}$. Π^L maps a pair of integers to `T` only if they are the same. Otherwise they are sent to `F`. Π^L maps a pair of morphisms, one matrix and one function, to `T` only if the function provides an answer to the problem for any input. Formally:

$$
\begin{aligned}
\Pi^L : \quad & \mathbf{M} \times \mathbf{N} \to \mathbf{And}, \\
& \langle m, n \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } m = n, \\ \mathtt{F} & \text{otherwise}, \end{cases} \\
& \big\langle A : k \to l,\ f : m \to n \big\rangle \mapsto \begin{cases} \mathtt{t} & \text{if } k = m,\ l = n, \\ & Af(z) = z,\ \forall z \in \mathbb{R}^m, \\ \mathtt{f} & \text{otherwise.} \end{cases}
\end{aligned}
$$

In order for this to be a valid functor definition, it must respect morphism endpoints, as well as the identity and composition requirements.

The endpoints are respected because a pair of a matrix and a function gets mapped to `t` only if their dimensions match, which is also the requirement for mapping their endpoints to `T`. Validating the identity requirement follows a similar procedure as in the proof of Lemma 5.27. The composition requirement states that for any $A \in \mathbb{R}^{k \times l}$ and $B \in \mathbb{R}^{l \times m}$, and every $f : k' \to l'$ and $g : l' \to m'$ it must hold that:

$$
\Pi^L \left(\big\langle A, f \big\rangle \fatsemi_{\mathbf{M} \times \mathbf{N}} \big\langle B, g \big\rangle \right) = \Pi^L(A, f) \fatsemi_{\mathbf{And}} \Pi^L(B, g). \tag{6.6}
$$

The right-hand side is `t` only if f is a solution for A and g is a solution for B, i.e. $k = k', l = l', m = m'$, and

$$
Af(u) = u,\ \forall u \in \mathbb{R}^k \quad \text{and} \quad Bg(v) = v,\ \forall v \in \mathbb{R}^l.
$$

But that also implies that

$$
ABg(f(u)) = Af(u) = u,\ \forall u \in \mathbb{R}^k,
$$

hence $f \fatsemi g$ is an answer for AB, and the left-hand side of Equation (6.6) must also hold. One can also show that the reverse also holds, i.e. the composition of answers implies that the two functions should be answers by themselves.

For a given problem statement (i.e. a matrix $A \in \mathbb{R}^{k \times l}$), there might be none, one, or an infinite amount of answers. In other words:

- if A is an inconsistent matrix, there's no morphism f in $\mathrm{Hom}_{\mathbf{N}}(k, l)$ such that $\Pi^L(A, f) = \mathtt{t}$;
- if A is an indeterminate matrix, then there are an infinite amount of morphisms $f \in \mathrm{Hom}_{\mathbf{N}}(k, l)$ such that $\Pi^L(A, f) = \mathtt{t}$;
- otherwise, there's exactly one morphism f in $\mathrm{Hom}_{\mathbf{N}}(k, l)$ such that $\Pi^L(A, f) = \mathtt{t}$.

Remark 6.21. Equation (6.6) shows why we are interested in functorial relationships between statement and answer categories. When we have such a functorial relationship it naturally follows that the kind of a composition of a statements and answers is the composition of the kinds of the individual pairs.

Example 6.22 (Solving linear systems via Moore-Penrose pseudoinverses). One common way to solve a system of linear equations $Ax = a$ is via the Moore-Penrose pseudoinverse A^+. A very useful feature of this result is that it is well-behaved also for indeterminate and the inconsistent cases. Given a matrix $A \in \mathbb{R}^{k \times l}$ and a vector $a \in \mathbb{R}^k$, the Moore-Penrose pseudoinverse A^+ provides all solutions via the equation

$$x = A^+ a + (I - A^+ A) w,$$

where w can be any vector in $\mathbb{R}^l$. If the system has a single answer, then $I - A^+ A$ evaluates to a zero matrix, hence $x = A^+ a$. If the system has multiple answers, then they can all be obtained with a suitable choice for w. Finally, if the system is inconsistent, then $AA^+ a \neq a$, so this can be used as a test.

Therefore, we can now define a procedure Δ^{MP} in **LAnd** that solves the Π^L problem outlined above.

$$\begin{aligned}
\Delta^{MP}: \quad & \mathbf{M} \times \mathbf{N} \to \mathbf{And}, \\
& \langle m, n \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } m = n, \\ \mathtt{F} & \text{otherwise,} \end{cases} \\
& \big\langle A : k \to l,\ f : m \to n \big\rangle \mapsto \begin{cases} \mathtt{t} & \text{if } k = m,\ l = n, \text{ and} \\ & \forall z \in \mathbb{R}^m \text{ exists a } w \in \mathbb{R}^l \text{ s.t.} \\ & f(z) = A^+ z + (I - A^+ A) w, \\ \mathtt{f} & \text{otherwise.} \end{cases}
\end{aligned}$$

We can directly see that Δ^{MP} is indeed a solution of Π^L, i.e. we have

$$\Psi^{\mathbf{LAnd}}_{\mathbf{M},\mathbf{N}}(\Pi^L, \Delta^{MP}) = \mathtt{T}.$$

That is because if we have $f(z) = A^+z + (I - A^+A)w$, for any z then f is indeed the right answer as per the definition of Π^L.

Example 6.23 (Linear systems (extended))**.** Let's revisit the problem definition in Example 6.20. We mentioned that a linear system represented by a matrix A can have no solutions, a single one, or an infinite amount of solutions. Note that when the matrix has no solutions, then no matter what function we pair it with, it will never map to the $\mathtt{t}$ morphism in **And**. Hence, if a matrix together with some function map to $\mathtt{t}$, the corresponding linear system would have either one or an infinite amount of solutions. More concretely, if A is invertible, it'd be of the first case, otherwise, it'd be of the second case. And furthermore, note that the product of two matrices A and B is invertible only if both of them are. This hints at a compositionality property.

Let's modify **And** a bit:

$$\begin{tikzcd}
\mathtt{T} \arrow[loop left, "\mathtt{f}"] \arrow[loop above, "\mathtt{t}^1"] \arrow[loop below, "\mathtt{t}^\infty"] \arrow[rr, bend left, "\mathtt{f}"] & & \mathbf{F} \arrow[loop right, "\mathtt{f}"] \arrow[ll, bend left, "\mathtt{f}"]
\end{tikzcd}.$$

We call this category $\mathbf{And}^\#$ and we identify $\mathtt{t}^1 \mathbin{⨟} \mathtt{t}^1$ with $\mathtt{t}^1$ and all other compositions between $\mathtt{t}^1$ and $\mathtt{t}^\infty$ with $\mathtt{t}^\infty$. Now, we can have a problem $\Pi^{L\#} : \mathbf{M} \times \mathbf{N} \to \mathbf{And}^\#$ that maps single-solution and multi-solution problems to different morphisms in $\mathbf{And}^\#$ while respecting the compositionality of this property.

Note that $\Pi^{L\#}$ is not a problem in **LAnd**, but rather can be one in a **Laputa** category where the kinds for the problems are the morphisms of $\mathbf{And}^\#$:

$$\mathbf{LAnd}^\# := \mathbf{Laputa}\left(\mathbf{Cat}, B^\#, B, \Psi^{\mathbf{And}^\#}\right),$$

with $B^\#$ and $\Psi^{\mathbf{And}^\#}$ appropriately defined.

6.5 Probabilistically correct solutions

So far we have been working in a deterministic setting. However sometimes we can solve a problem only probabilistically, i.e. we will get an "answer" but whether this answer is correct or not is something that we cannot be 100% sure.

One example of that are inference problems. When dealing with such problems, one usually has a dataset sampled from a (partially) unknown distribution and they need to infer the distribution from the data. Because the sample is a random variable itself, the inference about the distribution is also random. Hence, conclusions about it can be only made in a probabilistic setting. We'll illustrate this in Examples 6.24 and 6.25.

Another setting where we speak of having a solution only probabilistically is the case of randomized algorithms. We will show how verifying that a function solves a linear system (the problem in Example 6.20) can be solved with a randomized algorithm.

The computational computational system that we will be using will be very similar to the **LAnd** system in the previous section. Again, statements and answers will form categories. However, now instead of having kind $\{\mathtt{t}, \mathtt{f}\}$ we will use probabilities instead.

So we define the category of probabilities **Pr**. It has a single object $\circ$, and morphisms indexed by the reals in the range $[0, 1]$. The identity morphism on this object is 0, and the composition of two morphisms r_1, r_2 is

$$r_1 \mathbin{⨾} r_2 := \min(1, r_1 + r_2).$$

We still need to define a rig on the morphisms of this category. The objects of this rig would be simply the reals in the range $[0, 1]$. When we compose problems, we would like the resulting probability to be the probability that both statement-answer mappings are correct, hence, assuming that they are independent, that would result in the product of their respective probabilities. And as we integrate over the various terms for the inner object, we would like to pick the pair that results in the highest probability. Hence, we define our probability rig as:

$$\mathrm{Pr} := \langle \{x \in \mathbb{R} \mid x \in [0, 1]\}, \max, 0, \times, 1\rangle .$$

One can easily check that with this definition composition, identities, and the annihilation properties follow.

Finally, we also need a solution judgement map. Let's choose the following semantics for something being a "solution": a procedure is a solution to a problem if the probability that the procedure assigns to the correctness of a pair of a statement and an answer is at least as high as the probability assigned by the problem. Hence, we want to err on the safe side. Put more formally:

$$\begin{aligned} \Psi^{\mathbf{LPr}}_{\mathbf{T},\mathbf{A}} \colon & \left(\mathbf{T} \xrightarrow{\mathrm{Pr}} \mathbf{A}\right) \times \left(\mathbf{T} \xrightarrow{\mathrm{Pr}} \mathbf{A}\right) \to \mathtt{Bool}, \\ & \qquad \langle \Pi, \Delta \rangle \mapsto \bigwedge_{T \in \mathsf{Ob}(\mathbf{T})} \bigwedge_{A \in \mathsf{Ob}(\mathbf{A})} \Delta(T, A) \geq \Pi(T, A). \end{aligned}$$

Therefore, the resulting compositional computation system can be defined as:

$$\mathbf{LPr} := \mathbf{Laputa}\left(\mathbf{Cat}, \mathrm{Pr}, \mathrm{Pr}, \Psi^{\mathbf{LPr}}\right).$$

In order to illustrate the utility of **LPr**, let's take a look at a concrete example of a problem in it.

Example 6.24 (Location model of the normal distribution)**.** Consider the problem of estimating the mean of a normally distributed variable. Take X to be normally distributed with an unknown mean μ and known variance σ^2, i.e. $X \sim \mathcal{N}(\mu, \sigma^2)$. We also assume a Bayesian setting, hence we take as a prior that μ is also normally distributed with known mean and variance, i.e. $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$. The problem then is, given n identically and independently distributed (iid) samples $x_1, \ldots, x_n$ taken from X, and an interval $[l, u]$ in $\mathbb{R}$, find the probability that μ is in this interval, i.e. $\mathbb{P}(\mu \in [l, u] \mid x_1, \ldots, x_n)$.

Our statement category $\mathbf{T}$ is a discrete category, with objects finite sets of real numbers. The only morphisms are the identity morphisms, hence morphism composition is not possible. Hence, an element $T \in \mathsf{Ob}(\mathbf{T})$ is a dataset with $n = |T|$ samples.

The answer category $\mathbf{A}$ is a preorder (a `Bool`-enriched category, or a category with at most one morphism between objects) with objects the real numbers and with morphism $i : l \to u,\ l, u \in \mathsf{Ob}(\mathbf{A})$ designating the $[l, u]$ interval in $\mathbb{R}$. Note that we require that the intervals are always valid, i.e. $l \leq u$. The identity morphism on object x is the interval $[x, x]$. Morphism composition is simply interval concatenation, i.e. given any $i : l \to u$, $i' : u \to u'$, we have that $i \mathbin{⨾} i' : l \to u'$ is the interval $[l, u']$.

Now, the problem functor $\Pi^N : \mathbf{T} \times \mathbf{A} \to \mathbf{Pr}$ maps (the identity morphisms on) a dataset T sampled iid from X and an interval $[l, u]$ to the probability that the unknown μ is in the interval:

$$\begin{aligned} \Pi^N : \qquad \mathbf{T} \times \mathbf{A} &\to \mathbf{Pr}, \\ \langle T, x \rangle &\mapsto \circ, \\ \langle \mathrm{id}_T, [l, u] \rangle &\mapsto \mathbb{P}\left(\mu \in [l, u] \mid T\right). \end{aligned}$$

The compositionally of this system is due to the fact that the probability of μ falling in an interval $[l, u]$ which contains m equals the sum of the probabilities that it falls in either $[l, m]$ or in $[m, u]$.

We can also provide a procedure for solving the Π^N problem.

Example 6.25 (Solving the location model)**.** It is a well-known fact that the posterior distribution of μ given a dataset $T = \{x_1, \ldots, x_n\}$ is also normal and is distributed as

$$\mu \mid x_1, \dots, x_n \sim \mathcal{N}\Bigg(\underbrace{\frac{\sigma_0^2}{\sigma_0^2 + \frac{\sigma^2}{n}} \frac{1}{n} \sum_{i=1}^{n} x_i + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \mu_0}_{\mu^*(T)},\ \underbrace{\frac{1}{\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}}}_{\sigma^*(T)} \Bigg).$$

Therefore the probability that $\mu \in [l, u]$ is:

$$\mathbb{P}\left(\mu \in [l, u] \mid T\right) = \Phi\left(\frac{u - \mu^*(T)}{\sigma^*(T)}\right) - \Phi\left(\frac{l - \mu^*(T)}{\sigma^*(T)}\right), \tag{6.7}$$

where Φ is the cumulative distribution function of the standard normal distribution.

Hence, we have the procedure

$$\begin{aligned} \Delta^N : \qquad & \mathbf{T} \times \mathbf{A} \to \mathbf{Pr}, \\ & \langle T, x \rangle \mapsto \circ, \\ & \langle \mathrm{id}_T, [l, u] \rangle \mapsto \Phi\left(\frac{u - \mu^*(T)}{\sigma^*(T)}\right) - \Phi\left(\frac{l - \mu^*(T)}{\sigma^*(T)}\right). \end{aligned}$$

We can immediately see from Equation (6.7) that Δ^N always return the same morphism in Pr as Π^N. Hence, we must have $\Psi^{\mathbf{LPr}}_{\mathbf{T},\mathbf{A}}\left(\Pi^N, \Delta^N\right) = \mathtt{T}$, meaning that the procedure Δ^N is indeed a solution for the problem Π^N.

Let's also take a look at a probabilistic algorithm. Recall Example 6.22. The definition of the procedure contains a statement like "for every $z \in \mathbb{R}^m$, there exists a $w \in \mathbb{R}^l$". However, an algorithm that checks a statement for all elements of an uncountably infinite set would never terminate. Hence, that is not a very good procedure. We will now look into a randomized algorithm that can check whether a function f provides solutions for the linear system A with various degrees of certainty.

Example 6.26 (Randomized verification of linear system answers)**.** If we don't have any additional information about the functions that constitute the morphisms of **N**, e.g. that they are linear, then there can be no deterministic procedure that can verify that a function always provides the correct answer to a system A. To see why this is the case, assume that there's a function f that is indeed the correct answer, i.e. $Af(z) = z$, $\forall z \in \mathbb{R}^m$, where $A \in \mathbb{R}^{m\times n}$. Now pick any $z' \in \mathbb{R}^m$ and a $y \in \mathbb{R}^n$ such that $y \neq f(z')$. Then the function

$$f'(z) = \begin{cases} f(z) & \text{if } z \neq z', \\ y & \text{otherwise} \end{cases}$$

provides the correct answer everywhere but on z'. A procedure can only find this if it checks for all elements of $\mathbb{R}^m$ which can be an infinitely long process.

We can, however, sample many random vectors from $\mathbb{R}^m$. If $Af(z) = z$ holds on all of them, then it is very likely that f is a correct answer. Exactly how likely is a question that we can answer with the Hoeffding's inequality (Hoeffding, 1963). The Hoeffding's inequality states that if $X_1, \ldots, X_n$ are iid random variables with support on the interval $[0, 1]$, then:

$$\mathbb{P}\left(\mathbb{E}[X] - \frac{1}{n}(X_1 + \ldots + X_n) \geq t\right) \leq e^{-2nt^2}, \quad \forall t \geq 0.$$

Take an iid sample $V = \{v_1, \ldots, v_n\}$ of vectors from $\mathbb{R}^m$. We said that we can be never completely right so let's say that we are happy if the function f fails to produce the right answer in at most η of the cases. For example, for 1 in 1000 failure rate, that'd be $\eta = 0.001$.

We can consider the indicator variable

$$b(z) := \mathbb{1}_{[Af(z) \neq z]}$$

to be a Bernoulli random variable. Let's also assume that for all $v_i \in V$ it holds that $b(v_i) = 0$. After all, if that's not the case we know that f is not a correct answer. As all random variables $b(v_i)$ have support only on 0 and 1, we can apply Hoeffding's inequality:

$$\mathbb{P}\left(\mathbb{E}_Z[Af(Z) \neq Z] - \frac{1}{n}(b(v_1) + \ldots + b(v_n)) \geq t\right) \leq e^{-2nt^2}, \quad \forall t \geq 0,$$

where $\mathbb{E}_Z[Af(Z) \neq Z]$ is the probability that f is not A's answer for some z. And as all $b(v_i)$ are 0, this becomes

$$\mathbb{P}\left(\mathbb{E}_Z[Af(Z) \neq Z] \geq t\right) \leq e^{-2nt^2}, \quad \forall t \geq 0.$$

We said above that we are willing to tolerate at most η failures, so:

$$\mathbb{P}\left(\mathbb{E}_Z[Af(Z) \neq Z] \geq \eta\right) \leq e^{-2n\eta^2}.$$

Hence, using the above result we can define the following procedure which evaluates the probability that a function f is the correct answer to a linear system problem with statement A:

$$\begin{aligned}\Delta^S\colon \quad & \mathbf{M} \times (\mathbf{N} \times \mathbb{N}_{\geq 1}) \to \mathbf{Pr}, \\ & \langle m, \langle n, s \rangle\rangle \mapsto \circ \\ & \left\langle A, \left\langle f, s \right\rangle\right\rangle \mapsto \min\left\{1 - \max\{b(v_1), \ldots, b(v_s)\}, 1 - e^{-2s\eta^2}\right\},\end{aligned}$$

where $\mathbb{N}_{\geq 1}$ is the discrete category whose elements are the positive natural numbers, and the $\{v_1, \ldots, v_s\}$ is an iid sampled dataset of size s. It is assumed that η is a constant that is determined beforehand.

For example, if we take $\eta = 0.001$ as above and we take $s = 1,000,000$ samples, all of which resulting in $b(v_i) = 0$, then we have the probability of f providing the correct answer in at least 99.9% of all possible cases to be

$$1 - \exp(-2 \times 1,000,000 \times 0.001^2) = 0.8647,$$

so about 86.5%. However, if even for a single v_i we have $b(v_i) = 1$, then Δ^S would map to 0%.

Part II

Co-design and compositional computation

Chapter 7

The mathematical theory of co-design

This part of the thesis will be mainly devoted to the theory of co-design and how it fits in the compositional computational systems developed in Part I. The theory of co-design is particularly rich in compositional structure which makes it a very good choice for illustrating the main concepts developed in this work. Furthermore, co-design is of significant utility for designing and optimizing real-world systems, hence showing connections with it further illustrates the theory of compositional computational problem-solving.

We start this extended example by first introducing the mathematical theory of co-design. This chapter will review just the fundamental definitions and results and might be rather dry to the reader who encounters co-design for the first time. For those interested to learn more, we strongly recommend (Censi et al., 2020). Historically, co-design, as we use it here, was first proposed by Censi (2016) as means for designing complex physical systems with multiple design objectives. It can also be extended to work with uncertainties (Censi, 2017). Nevertheless, its applications are much broader than this and it sits on an especially rich compositional and category-theoretical foundation. In this and the following chapters we will explore that and provide some new useful results as well as connections with some compositional computational systems.

7.1 A bit more order theory

We first introduced some notions of order theory in Section 2.1 because they were necessary for the development of the compositional computational theories in the first part of this thesis. Co-design is solidly grounded in order theory. Therefore, we need to also arm ourselves with the notions of upper and lower closures, joins, meets, lattices, as well as with some properties of posets.

Definition 7.1 (Upper closure)**.** The operator $\uparrow$ maps subsets S of a poset $\langle P, \leq_P \rangle$ to the smallest upper sets that contain them:

$$\begin{aligned}\uparrow \colon \mathcal{P}P &\to \mathsf{U}P,\\ S &\mapsto \{y \in P \colon \exists x \in S \colon x \leq_P y\}.\end{aligned}$$

Lemma 7.2. *Given a poset $\langle P, \leq_P \rangle$, $\mathsf{U}P$ is a poset with order given by*

$$A \leq_{\mathsf{U}P} B \;\coloneqq\; A \supseteq B.$$

There is a top element in $\mathsf{U}P$ and it is the empty set. There is also a bottom element: P itself.

We can similarly define the dual concept of lower closure:

Definition 7.3 (Lower closure)**.** The operator $\downarrow$ maps subsets S of a poset $(P, \leq_P)$ to the smallest lower sets that contain them:

$$\begin{aligned}\downarrow \colon \mathcal{P}P &\to \mathsf{L}P,\\ S &\mapsto \{y \in P \colon \exists x \in S \colon y \leq_P x\}.\end{aligned}$$

Lemma 7.4. *Given a poset $\langle P, \leq_P \rangle$, $\mathsf{L}P$ is a poset with order given by*

$$A \leq_{\mathsf{L}P} B \;\coloneqq\; A \subseteq B.$$

There is a top element in $\mathsf{L}P$ and it is P itself. There is also a bottom element: the empty set $\varnothing$.

Definition 7.5 (Join)**.** Given a poset $P = (P, \leq_P)$ and a subset of its elements $S \subseteq P$, then the *join* $\bigsqcup S$ of this subset is an element $x \in P$ such that $s \leq_P x$, for all $s \in S$, and for any $x' \in P$, such that $s \leq_P x'$, $\forall s \in S$, we have $x \leq x'$. In other words, $\bigsqcup S$ is the least element that reduce to all elements of S. If S has only two elements, i.e. $S = \{a, b\}$, we will also denote their join by $a \sqcup b$. The join of a subset S *does not always exist*.

Example 7.6. Take the real numbers and their typical order $\leq$. Then the join of any two $a, b \in \mathbb{R}$ is

$$a \sqcup b = \max\{a, b\}.$$

The join of the subset $S_1 = \{x \in \mathbb{R}\colon 0 \leq x \leq 1\}$ is $\bigsqcup S_1 = 1$. The join of the subset $S_2 = \{x \in \mathbb{R}\colon 0 < x < 1\}$ is also $\bigsqcup S_2 = 1$, even though $1 \notin S_2$.

Example 7.7. Take the set

$$P = \{(\vartriangle, x)\colon x \in \mathbb{R},\ x < 0\} \cup \{(\vartriangleleft, x)\colon x \in \mathbb{R},\ x \geq 0\} \cup \{(\vartriangleright, x)\colon x \in \mathbb{R},\ x \geq 0\},$$

together with the relation $\leq_P$ such that:

i. $(\vartriangle, a) \leq_P (\vartriangle, b) \iff a \leq b,$
ii. $(\vartriangleleft, a) \leq_P (\vartriangleleft, b) \iff a \leq b,$
iii. $(\vartriangleright, a) \leq_P (\vartriangleright, b) \iff a \leq b,$
iv. $(\vartriangle, a) \leq_P (\vartriangleleft, 0),\ \forall a \in \mathbb{R},\ a < 0\,,$
v. $(\vartriangle, a) \leq_P (\vartriangleright, 0),\ \forall a \in \mathbb{R},\ a < 0.$

Then $\bigsqcup\{(\vartriangle, x) \in \mathbb{R}\colon x < 0\}$ does not exist because, both $(\vartriangleleft, 0)$ and $(\vartriangleright, 0)$ are upper bounds for the subset but neither reduces to the other. If we also had $(\vartriangleleft, 0) \leq_P (\vartriangleright, 0)$ and $(\vartriangleright, 0) \leq_P (\vartriangleleft, 0)$, then both of them would be joins for this subset, and would be equal due to the antisymmetry property of posets.

Definition 7.8 (Meet)**.** Given a poset $P = (P, \leq_P)$ and a subset of its elements $S \subseteq P$, then the *meet* $\bigsqcap S$ of this subset is an element $x \in P$ such that $x \leq_P s$, for all $s \in S$, and for any $x \in P$, such that $x' \leq_P s,\ \forall s \in S$, we have $x' \leq x$.

Remark 7.9. The meet is a dual of the join (i.e. it is the join in the opposite poset), hence all the claims for the join hold equally for the meet.

Definition 7.10 (Lattice)**.** A *lattice* is a poset $\langle P, \leq \rangle$ with some additional properties:

i. Given two points $p, q \in P$, it is always possible to define their join $a \sqcup b$;
ii. Given two points $p, q \in P$, it is always possible to define their meet $a \sqcap b$.

Definition 7.11 (Bounded lattices)**.** If there is a least upper bound for the entire lattice A, it is called the *top* ($\top$). If the greatest lower bound exists it is called the *bottom* ($\bot$). If both a top and a bottom exist, we call the lattice *bounded*, and denote it by $\langle A, \leq, \sqcup, \sqcap, \bot, \top \rangle$.

Lemma 7.12. *Given a poset R, $\mathsf{U}R$ is a bounded lattice (Definition 7.10) with*

$$\langle \mathsf{U}R, \leq_{\mathsf{U}R}, \bot_{\mathsf{U}R}, \top_{\mathsf{U}R}, \sqcup_{\mathsf{U}R}, \sqcap_{\mathsf{U}R} \rangle = \langle \mathsf{U}R, \supseteq, R, \emptyset, \cap, \cup \rangle\,.$$

Proof. Can be found in (Censi et al., 2020). □

Lemma 7.13. *Given a poset F, $\mathsf{L}F$ is a bounded lattice (Definition 7.10) with*

$$\langle \mathsf{L}F, \leq_{\mathsf{L}F}, \bot_{\mathsf{L}F}, \top_{\mathsf{L}F}, \sqcup_{\mathsf{L}F}, \sqcap_{\mathsf{L}F} \rangle = \langle \mathsf{L}F, \subseteq, \emptyset, F, \cup, \cap \rangle\,.$$

Proof. Can be found in (Censi et al., 2020). □

Lemma 7.14. *Given a poset $\langle P, \leq_P \rangle$, we can identify every element $p \in P$ by an upper set in $\mathsf{U}P$, by taking its upper closure. Furthermore, there exists a partial function $\kappa_P : \mathsf{U}P \hookrightarrow P$ such that*

$$P' = \uparrow\{p\} \iff \kappa_P(P') = p.$$

Proof. We give an explicit definition for κ_P:

$$\begin{aligned} \kappa_P \colon \mathsf{U}P &\hookrightarrow P, \\ u &\mapsto \begin{cases} p' & \text{if exists } p' \in u \text{ such that } p' \leq_P u',\ \forall u' \in u, \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned}$$

Note that if such p' exists it must be unique due to the antisymmetry property of posets. Furthermore, we can see from Definition 7.1 that for any p it holds that $\kappa_P(\uparrow\{p\}) = p$. Now we need to also show that $\uparrow\{p\}$ is the only upper set in $\mathsf{U}P$ which κ_P maps to p. Assume there is another upper set $v \in \mathsf{U}P$ such that $p \in v, p \leq_P v', \forall v' \in v$. As $\uparrow\{p\}$ is the smallest such upper set, v must contain at least one additional element x, for which it must hold that $x \leq_P p$, $p \leq_P x$ or they are not comparable. The first case is in contradiction with the definition of κ_P, the second case implies that $x \in \uparrow\{p\}$ and v cannot be distinct, which is another contradiction, and the last again contradicts the definition of κ_P. Hence, v must equal $\uparrow\{p\}$. □

Lemma 7.15. *Given a family $P' = \{P_i\}_{i \in I}$ of upper sets of the poset $\langle P, \leq_P \rangle$ indexed by a set I, their union $\bigcup_{i \in I} P_i$ is also an upper set.*

Lemma 7.16. *Given a poset A, and any element $a \in A$, it holds that:*

$$\uparrow_A\{a\} = \downarrow_{A^{\mathrm{op}}}\{a\}.$$

7.2 Design problems

From a mathematical point of view, a "design problem" is simply a different name for a feasibility relation. We first saw the notion of feasibility relation in Definition 2.14. Then, feasibility relations were used to keep track of the resources required for computing procedures (Definition 3.16). However, design problems (as we will call them from now on) are much richer in structure and are the building blocks of co-design.

From a semantic point of view, design problems illustrate trade-offs in decision-making. If we have two posets F and R and a feasibility relation (or, interchangeably, a design problem) with *functionality* F and *resource* R is written

$$d : F \nrightarrow R.$$

We call the elements of F a *functionality* because they usually refer to something we want, e.g. money in the right currency, a new bike, universal healthcare, or a well-functioning legal system. Each element of F corresponds to a different specific instance of this class of desired things, and the order on P acts to designate which instances are more preferable. For example, getting 100 CHF is commonly believed to be better than getting 10 CHF and having impartial judges but limited access to legal help, while not perfect, is perhaps better than having biased judges *and* limited access to legal help. Both of these things can be expressed as order relations on the functionality poset F.

But, as people say, "there is no free lunch". Hence, the elements of the poset on the other end of the design problem, the resource R, correspond to the "price" of getting what we want. It can represent other currencies, or debt, prices, tax burden, or political risk. And again we can have our preferences ordered.

Recall that we ask for feasibility relations to be monotone. When we translate this to the interpretation we use above this results in the following philosophy:

> *If it is possible to get feasibility f by giving up resource r, then it is also possible to get it by giving up "more" than r, e.g. $r' \geq_R r$. Similarly, if it is possible to get feasibility f by giving up resource r, then it is also possible to get "less" than f, e.g. $f' \leq_F f$ with the same resource r.*

At first this might seem a little bit constraining and the curious reader might be ready to argue that many things in the real world are not monotone. If that's the case, we'd like to urge them to look for an example that cannot be represented as or converted to a monotone problem. It is surprisingly difficult.

The intuition and applications are much broader and Censi et al. (2020) do a great job at explaining them so we recommend the reader to refer to their paper. We will instead focus on the mathematical aspects of co-design.

Definition 7.17 (Composite design problem)**.** Given three posets P, Q and R, and two design problems (feasibility relations) $\Phi : P \nrightarrow Q$ and $\Psi : Q \nrightarrow R$, one can define a *(then-)composite design problem for P given R* as

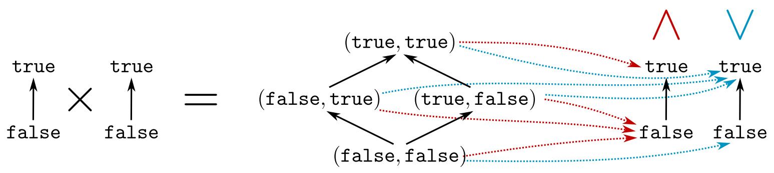
$$\begin{aligned} \Phi \fatsemi \Psi \colon P^{\mathrm{op}} \times R &\to \texttt{Bool}, \\ \langle p, r \rangle &\mapsto \bigvee_{q \in Q} \Phi(p, q) \wedge \Psi(q, r), \end{aligned}$$

where $q \in Q$ refers to the set of objects of the poset Q.

An intuition for the composite feasibility relation is that p can be obtained given r if there is at least one $q \in Q$ which can be obtained given r and which in turn can be used to obtain p.

Lemma 7.18. *The composite design problem is indeed a design problem (feasibility relation), i.e. composition preserves monotonicity.*

Proof. As Φ and Ψ are monotone maps to `Bool` we only need to show that $\vee$ and $\wedge$ preserve monotonicity. We can directly see that from how they map the product poset `Bool` $\times$ `Bool` to `Bool`:



□

Lemma 7.19. *The composite design problem is associative, i.e. for any four posets P, Q, R, S and three feasibility relations $\Phi_{PQ} : P \nrightarrow Q$, $\Phi_{QR} : Q \nrightarrow R$, and $\Phi_{RS} : R \nrightarrow S$ we have:*

$$(\Phi_{PQ} \mathbin{⨾} \Phi_{QR}) \mathbin{⨾} \Phi_{RS} = \Phi_{PQ} \mathbin{⨾} (\Phi_{QR} \mathbin{⨾} \Phi_{RS}).$$

Proof. Due to the properties of $\vee$ and $\wedge$ we have:

$$\begin{aligned}(\Phi_{PQ} \mathbin{⨾} \Phi_{QR}) \mathbin{⨾} \Phi_{RS} &= \bigvee_{r \in R} \left[\left(\bigvee_{q \in Q} \Phi_{PQ}(p,q) \wedge \Phi_{QR}(q,r) \right) \wedge \Phi_{RS}(r,s) \right] \\ &= \bigvee_{r \in R} \bigvee_{q \in Q} \left[\Phi_{PQ}(p,q) \wedge \Phi_{QR}(q,r) \wedge \Phi_{RS}(r,s) \right].\end{aligned}$$

The same can be shown for $\Phi_{PQ} \mathbin{⨾} (\Phi_{QR} \mathbin{⨾} \Phi_{RS})$, hence the composition of design problems is associative. □

Definition 7.20 (Identity design problem). For any poset $P = (P, \leq_P)$ one can define the identity design problem $\Phi^{\text{id}}_P : P \nrightarrow P$ as

$$\Phi^{\text{id}}_P(p, p') = \begin{cases} \mathtt{T} & \text{if } p \leq_P p', \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

Lemma 7.21. *The identity design problem is indeed a design problem.*

Proof. In order to show that the identity design problem is a design problem we need to show that it is a monotone map to `Bool`. In particular, that if for some $p, p', q, q' \in P$ we have that

$$\langle p, p' \rangle \leq_{(P^{\text{op}} \times P)} \langle q, q' \rangle,$$

then $\Phi^{\mathrm{id}}_P(p,p') \leq_{\mathtt{Bool}} \Phi^{\mathrm{id}}_P(q,q')$.

From Definition 2.6 and Definition 2.7 we know that:

$$\begin{aligned}\langle p,p'\rangle \leq_{(P^{\mathrm{op}}\times P)} \langle q,q'\rangle &\iff p \leq_{P^{\mathrm{op}}} q \ \wedge\ p' \leq_P q' \\ &\iff q \leq_P p \ \wedge\ p' \leq_P q'.\end{aligned}$$

If $p \leq_P p'$ then by the transitivity property of posets $q \leq_P q'$. But then also $\Phi^{\mathrm{id}}_P(p,p') = \mathtt{T}$ and $\Phi^{\mathrm{id}}_P(q,q') = \mathtt{T}$, so $\Phi^{\mathrm{id}}_P(p,p') \leq_{\mathtt{Bool}} \Phi^{\mathrm{id}}_P(q,q')$. If $p \nleq_P p'$, then $\Phi^{\mathrm{id}}_P(p,p') = \mathtt{F}$ and for any value of $\Phi^{\mathrm{id}}_P(q,q')$ we have $\Phi^{\mathrm{id}}_P(p,p') \leq_{\mathtt{Bool}} \Phi^{\mathrm{id}}_P(q,q')$. □

Definition 7.22 (Product design problem)**.** Given 4 posets P, Q, R, S and two design problems $\Phi : P \nrightarrow Q$ and $\Psi : R \nrightarrow S$, we define the *product design problem* for the product poset $P \times R$ given the product poset $Q \times S$ as

$$\begin{aligned}\Phi \times \Psi : (P\times R)^{\mathrm{op}} \times (Q \times S) &\to \mathtt{Bool}, \\ \langle\langle p,r\rangle, \langle q,s\rangle\rangle &\mapsto \Phi(p,q) \wedge \Psi(r,s).\end{aligned}$$

Lemma 7.23. *The product design problem is indeed a design problem.*

Proof. In order to show that the product design problem is a design problem we need to show that it is a monotone map to `Bool`. Consider some $p, p' \in P$, $q, q' \in Q$, $r, r' \in R$, $s, s' \in S$ such that

$$(\Phi \times \Psi)\left(\langle p,r\rangle, \langle q,s\rangle\right) = \mathtt{T},$$

as well as $\langle p',r'\rangle \leq_{P\times R} \langle p,r\rangle$, and $\langle q,s\rangle \leq_{Q\times S} \langle q',s'\rangle$. From the product poset definition (Definition 2.6) we have:

$$\langle p',r'\rangle \leq_{P\times R} \langle p,r\rangle \implies p' \leq_P p \ \wedge\ r' \leq_R r,$$

$$\langle q,s\rangle \leq_{Q\times S} \langle q',s'\rangle \implies q \leq_Q q' \ \wedge\ s \leq_R s'.$$

Also from the product design problem definition:

$$(\Phi \times \Psi)\left(\langle p,r\rangle, \langle q,s\rangle\right) = \mathtt{T} \implies \Phi(p,q) = \mathtt{T} \ \wedge\ \Psi(r,s) = \mathtt{T}.$$

And due to the monotonicity of Φ and Ψ we have:

$$\left.\begin{aligned}\Phi(p,q) = \mathtt{T} \\ p' \leq_P p \\ q \leq_Q q'\end{aligned}\right\} \implies \Phi(p',q') = \mathtt{T},$$

$$\left.\begin{aligned}\Psi(r,s) = \mathtt{T} \\ r' \leq_R r \\ s \leq_S s'\end{aligned}\right\} \implies \Psi(r',s') = \mathtt{T}$$

Therefore, $(\Phi \times \Psi)\left(\langle p',r'\rangle, \langle q',s'\rangle\right)$ is also $\mathtt{T}$. □

Definition 7.24 (Sum design problem)**.** Given 2 posets P, Q and two design problems $\Phi : P \nrightarrow Q$ and $\Psi : P \nrightarrow Q$, we define the *sum design problem* for the two as:

$$\begin{aligned} \Phi \vee \Psi : P^{\mathrm{op}} \times Q &\to \mathtt{Bool}, \\ \langle p, q \rangle &\mapsto \Phi(p,q) \vee \Psi(p,q). \end{aligned}$$

Lemma 7.25. *The sum design problem is indeed a design problem.*

Proof. In order to show that the sum design problem is a design problem we need to show that it is a monotone map to `Bool`. Consider some $p, p' \in P$, $q, q' \in Q$ such that $(\Phi \vee \Psi)(p,q) = \mathtt{T}$, $p' \leq_P p$, and $q \leq_Q q'$. Note from Definition 7.24 that the sum relation is true if at least one of its two terms are true. And due to the monotonicity of Φ and Ψ, if one of them is `T` for (p,q), it must be `T` for (p', q'). Hence if $(\Phi \vee \Psi)(p,q) = \mathtt{T}$, then it must hold that $(\Phi \vee \Psi)(p', q') = \mathtt{T}$ as well. □

Definition 7.26 (Intersection design problem)**.** Given 2 posets P, Q and two design problems $\Phi : P \nrightarrow Q$ and $\Psi : P \nrightarrow Q$, we define the *intersection design problem* for the two as:

$$\begin{aligned} \Phi \wedge \Psi : P^{\mathrm{op}} \times Q &\to \mathtt{Bool}, \\ \langle p, q \rangle &\mapsto \Phi(p,q) \wedge \Psi(p,q). \end{aligned}$$

Lemma 7.27. *The intersection design problem is indeed a design problem.*

Proof. Analogous to the proof of Lemma 7.25 □

Chapter 8

Solving design problems

So far we have discussed design problems (feasibility relations) and the posets on which they operate. We have seen various ways to combine them and to build more complex problems from simple ones. However, we have not mentioned how one can *solve* a design problem. This section will focus on that. The parts of this section which describe the means for representing the solutions of design problems and solving them follow the treatment from Censi (2016). To the author's best knowledge, the characterization of the properties needed for a design problem to be solvable with these techniques is a novel contribution of this work.

8.1 Representation of a solution

First, we need to address the question of what does it even mean to *solve a design problem*. Intuitively, we mean finding the "smallest" resource that makes a given functionality in which we are interested feasible. However, as we deal with posets, there might also be multiple non-comparable resources that make our target functionality feasible. Therefore, we will need some more tools in order to make this definition specific.

Definition 8.1 (Chain)**.** A *chain* in a poset $P = (P, \leq_P)$ is a subset $C \subseteq P$ such that for any $a, b \in C$ we have either $a \leq_P b$ or $b \leq_P a$.

Definition 8.2 (Antichain)**.** An *antichain* in a poset $P = (P, \leq_P)$ is a subset $A \subseteq P$ such that for any $a, b \in A$ it holds that if $a \leq_P b$ then $a = b$. The set of all antichains in P will be denoted by $\mathcal{A}P$.

A chain is a subset of the poset on which we have total order (also called linear order). An antichain is the opposite: none of the elements of an antichain are comparable.

Definition 8.3 (Minimum elements of a set)**.** The *minimum elements map* of a poset $P = (P, \leq_P)$ is the function

$$\begin{aligned}\mathrm{Min}\colon \mathscr{P}P &\to \mathscr{A}P \\ S &\mapsto \{x \in S \mid (y \in S) \wedge (y \leq_P x) \implies (x = y)\}.\end{aligned}$$

Min maps every subset of P to the antichain that contains all elements in the subset that do not reduce to any other element of the subset.

Note that for a set S endowed with a relation $\leq_S$ there's a difference between the minimum element $\min S$, should such exist and the minimum elements $\mathrm{Min}\, S$. The first exists only if there is an element $s \in S$ such that $s \leq p, \forall p \in S$. If, however, there is an element $s' \in S$ which is not comparable with s, then there would be no minimum element in S. In this case we can still have (an antichain of) *minimum elements*, which is a generalization of the minimum element.

Lemma 8.4. *$\mathscr{A}P$ is a poset with order given by*

$$A \leq_{\mathscr{A}P} B := \uparrow A \supseteq \uparrow B.$$

Proof. This is a poset because $A \subseteq B \;\wedge\; B \subseteq A \implies A = B$. There is a top element in $\mathscr{A}P$ and it is the empty set. There is a bottom element only if P itself has a bottom element. If that is the case, then the bottom of $\mathscr{A}P$ is the bottom of P. □

All the computational results in this section make use of an intuitive duality between upper sets and the minimum elements. The idea is that each upper set can be compactly represented with its set of minimum elements and that (most) operations performed on upper sets can be performed more efficiently on their corresponding antichains of minimum elements. However, there are some subtleties when dealing with the minimum elements of a set that have counter-intuitive results, as the following example illustrates.

Example 8.5. Consider the (total) order on the reals $(\mathbb{R}, \leq)$ and the subset $S = \{x \in \mathbb{R} \mid x > 3.0\}$. One might expect that $\mathrm{Min}\, S = \{3.0\}$ but $3.0 \notin S$, hence that cannot be true. In fact, as for every $x \in S$ we have at least one $y \neq x$, such that $y \leq x$, it is true that for no $x \in S$ it holds that

$$(y \in S) \wedge (y \leq x) \implies (x = y).$$

Therefore, $\mathrm{Min}\, S = \varnothing$. Furthermore $\uparrow \mathrm{Min}\, S = \varnothing$.

For reasons that will become clear later in this section this setting is highly undesirable. In fact, we would like to make use of the following relation for every upper set $U \in \mathsf{U}P$:

$$U = \uparrow \mathrm{Min}\, U. \tag{8.1}$$

So, the natural question to ask and study is *what are the conditions for Equation* (8.1) *to hold*. Theorem 8.7 addresses this question.

Remark 8.6. Using the minimum antichain as a representation for an upper set is an idea going back to the original paper by Censi (2016) that introduced the mathematical theory of co-design. However, Censi did not consider the conditions under which Equation (8.1) holds true. To the best of the author's knowledge, no subsequent publications have addressed this question. Theorem 8.7 proposes an answer to it and is one of the main results of this section. When combined with the conditions necessary for solving loop problems with the Kleene Fixpoint Theorem (Theorem 8.33), we end up at the definition of well-behaved design problems (Definition 8.39). This definition clarifies a fundamental but previously unaddressed theoretical aspect of the theory of co-design: which design problems admit the solution techniques forming the basis of computational co-design.

Theorem 8.7 (Equivalence between upper sets and their minimum elements)**.** *Sufficient conditions for Equation* (8.1) *to hold for an upper set $U \in \mathsf{U}P$ are:*

i. U is a finite set, or

ii. every chain in U has a lower bound, or

iii. for every element a of a chain in U which doesn't have a lower bound it holds that $\{x \in U \mid x \leq a\} \cap \mathrm{Min}\, U \neq \varnothing$.

The third condition is also a necessary condition.

Before we prove it, we need the definition for a lower bound of a chain and the axiom of choice.

Definition 8.8 (Lower bound of a chain)**.** Given a poset $P = (P, \leq_P)$ and a subset $S \subseteq P$, the lower bound of S (under the induced order from P) is an element of P such that $\forall s \in S,\ p \leq s$.

Example 8.9. Take again the (total) order on the reals $(\mathbb{R}, \leq)$ and the subset

$$S = \{x \in \mathbb{R} \mid x > 3.0\}.$$

First, note that S is a chain as it has a total order. Any $y \leq 3.0$ is a lower bound of the chain S, hence a chain can have more than one lower bound. Consider now the subset

$$S' = \{x \in \mathbb{R} \mid x < 3.0\}.$$

This is also a chain but has no lower bound.

Definition 8.10 (Choice function)**.** A *choice function* is a function f that is defined on some collection X of nonempty sets and assigns to each set S in that collection some element $f(S)$ of S.

We need a choice function in order to be able to "take an element from the set S". One can construct such a function for a finite collection X by simply stating which element should be picked out from every set. Things get complicated when the collection X has an infinite amount of sets. Then, such a construction cannot be created and there is no way to show that it even exists under the usual axioms of set theory (e.g. Zermelo-Fraenkel axiomatization). That is why we need the Axiom of choice:

Axiom 8.11 (Axiom of choice)**.** Given a non-empty family $X = \{X_i\}_{i\in I}$ of non-empty sets indexed by the elements of I, there exists a choice function for X, that is a function

$$f : I \to \bigcup_{i\in I} A_i,$$

such that for all $i \in I$, it holds that $f(i) \in A_i$.

Armed with the above definitions and the Axiom of choice, we can now prove Theorem 8.7.

Proof of Theorem 8.7. Given a subset $S \subseteq P$ and $s \in S$ we can define the set of all the elements of S strictly preceding s as:

$$S_s = \{x \in S \mid x \leq_P s \wedge x \neq s\}.$$

We will call S_s the *strictly preceding set of s in S*. Hence, we can rewrite the definition of Min as:

$$S \mapsto \{x \in S \mid S_x = \varnothing\}.$$

Using this we can rewrite the expression in Equation (8.1) as:

$$\begin{aligned} U &= \uparrow \operatorname{Min} U \\ &= \{u \in P \mid \exists m \in \operatorname{Min} U, \text{ s.t. } m \leq_P u\} \\ &= \{u \in U \mid \exists m \in \operatorname{Min} U, \text{ s.t. } m \leq_P u\} \qquad (8.2) \\ &= \{u \in U \mid \exists m \in \{x \in U \mid U_x = \varnothing\}, \text{ s.t. } m \leq_P u\} \\ &= \{u \in U \mid \exists m \in U, \text{ s.t. } U_m = \varnothing \wedge m \leq_P u\}. \qquad (8.3) \end{aligned}$$

Equation (8.2) follows from the fact that for no $p \in P/U$ it can hold that $m \leq_P u$ for some $m \in \operatorname{Min} U \subseteq U$. Therefore, proof of equality in Equation (8.3) boils down to showing that for every $u \in U$ there is an m for which the condition on the right-hand-side holds.

We will start with showing that (ii) is a sufficient condition. For any chain $C \subseteq U$ we then have a lower bound $L(C)$. Then it must hold that $C_{L(C)} = \emptyset$, otherwise $L(C)$ wouldn't be a lower bound.

To prove that (ii) holds we will show that for every $u \in U$ we can build a sequence by recursively picking elements from the strictly preceding set of the last element of the sequence until we reach a $u' \in U$, such that $U_{u'} = \emptyset$. Note that by the Axiom of choice we can have a choice function that selects an element from every non-empty U_u:

$$I = \{u \in U \mid U_u \neq \emptyset\}$$

$$f : I \to U, \text{ s.t. } \forall i \in I,\ f(i) \in U_u.$$

For an $u_1 \in U$, if $U_{u_1} = \emptyset$, then the right-hand-side of Equation (8.3) holds trivially and $u_1 \in \uparrow \operatorname{Min} U$. Otherwise, if $U_{u_1} \neq \emptyset$, then we can use f to select another element $u_2 = f(u_1) \in U_{u_1}$. Again, if $U_{u_2} = \emptyset$, then the right-hand-side of Equation (8.3) holds for $m = u_2$. If $U_{u_2} \neq \emptyset$, then we can continue building the sequence. This sequence is strictly decreasing, as we always pick the next element from the strictly decreasing set of the previous element. Hence, it is a chain, and as every chain has a lower bound, the sequence must terminate, and the final element would be one that satisfies the right-hand-side of Equation (8.3) and $u_1 \in \uparrow \operatorname{Min} U$. This holds for any $u_1 \in U$, so the equality of the two sides of Equation (8.3) directly follows.

Now, let's show that (i) implies (ii), hence it is also a sufficient condition. We will do that by contradiction. Assume U is finite, but there is some chain starting at $a \in U$ which doesn't have a lower bound. Hence, $\nexists b \in U_a,\ U_b = \emptyset$, i.e. $\forall b \in U_a,\ U_b \neq \emptyset$. Assume again that we have a choice function f from every $U_b,\ b \in U_a$. These subsets are all non-empty, so its existence is guaranteed (and as $U_b \subset U_a \subset U$ and $\mathcal{P}U$ are all finite, we don't even need the Axiom of choice). So we can build a sequence (which is also a chain) as in the previous paragraph. However, we will never reach the terminal condition, so this sequence would be infinite. Furthermore, as it is a strictly decreasing sequence, its elements must be all unique. Hence we are trying to select an infinite amount of unique elements from a finite set, which is impossible. Therefore, every chain in a finite set must have a lower bound.

Let's prove the third necessary condition. Note that $\{x \in U \mid x \leq a\} \cap \operatorname{Min} U \neq \emptyset$ holds for every $a \in U$, such that all the chains that contain it have a lower bound (following from (ii)). Hence, condition (iii) states that for all elements $a \in U$, it holds that $\{x \in U \mid x \leq a\} \cap \operatorname{Min} U \neq \emptyset$. But:

$$\{x \in U \mid x \leq a\} \cap \operatorname{Min} U \neq \emptyset \ \equiv\ \exists m \in U, \text{ s.t. } U_m = \emptyset,\ m \leq a, \tag{8.4}$$

which is the right-hand-condition of Equation (8.3). Therefore, the condition holds for all elements of U and the equality of the sets holds.

Finally, we have to confirm that (iii) is also a necessary condition. In particular, we have to show that if there is an element $a \in U$, such that $\{x \in U \mid x \leq a\} \cap \operatorname{Min} U = \varnothing$, then $a \notin \uparrow \operatorname{Min} U$. From Equation (8.4) we know that the condition in Equation (8.3) would not be satisfied for this a, hence $a \notin \uparrow \operatorname{Min} U$. This concludes the proof. □

Example 8.12. Let's now see where Example 8.5 fails. We saw that

$$S = \{x \in \mathbb{R} \mid x > 3.0\}$$

does not satisfy Equation (8.1). Now we can see why. The set is itself a chain with no lower bound. Furthermore, as $\operatorname{Min} S = \varnothing$, condition (iii) in Theorem 8.7 is violated. Therefore, one should either restrict themselves to upper sets of $\mathbb{R}$ defined by non-strict inequalities, i.e.

$$S' = \{x \in \mathbb{R} \mid x \geq 3.0\},$$

or would need to extend $\mathbb{R}$ with special elements that serve as minimums: $\underline{\mathbb{R}} = \mathbb{R} \cup \{L_x \mid x \in \mathbb{R}\}$ such that

$$L_x < y, \ \forall y \in \{y' \in \mathbb{R} \mid x \leq y'\}, \forall x \in \mathbb{R}.$$

Now we can better describe what we mean by *solving a design problem*. Given a design problem $D\colon F \nrightarrow R$, and a functionality $f \in F$ which we wish to obtain, we define the *feasible set of resources for* f as the set

$$\mathfrak{F}_f = \{r \in R \mid D(f, r) = \mathtt{T}\}.$$

Lemma 8.13. *For any $f \in F$ the set $\mathfrak{F}_f$ is an upper set of F.*

Proof. By the requirement that the design problem D is monotonic we know that if for some $r' \in R$ we have that $r' \in \mathfrak{F}_f$, then any $r \in R$, such that $r' \leq_R r$, must also be in $\mathfrak{F}_f$. □

Remark 8.14. For the rest of this document, we will *always* assume that the feasible sets of resources satisfy condition (iii) of Theorem 8.7. This is a critical requirement, as all the computational results would depend on the duality in Equation (8.1).

The feasible set of resources for our desired functionality u is in a sense what we are looking for. However, typically most of the elements of the set would have very little practical value. In fact, it makes more sense to instead only consider the minimum resources needed. Due to the monotonicity, we know that any resource that reduces to a minimum resource must also be in the feasibility set. Furthermore, thanks to Theorem 8.7 we know that we can always recover the upper set from its minimum elements, as long as it satisfies the necessary condition in the theorem. The minimum resources are in fact the antichain $\operatorname{Min} \mathfrak{F}_f$.

Definition 8.15 (Solution map for a design problem)**.** Given a design problem $D\colon F \nrightarrow R$, its solution map h_D is defined as:

$$\begin{aligned} h_D\colon F &\to \mathcal{A}R \\ f &\mapsto \operatorname{Min}\{r \in R \mid D(f,r) = \mathtt{T}\}. \end{aligned}$$

The map h maps each functionality f to the set of minimal resources that can realize it. We will from now on assume that each design problem D comes with its solution map h_D. The rest of this section will study what happens with the solution maps under the various composition operations we defined before. In fact, we will show that the solution map for a problem composed from subproblems with known solution maps, can be represented (and calculated) as a function of the maps of the subproblems.

As the solution map is a representation of a feasibility relation, it is only natural to expect that it also has some monotonicity properties. These are formalized in the following lemma.

Lemma 8.16 (Monotonicity of the solution map)**.** *Given a design problem*

$$D\colon F \nrightarrow R,$$

and $f \in F$, then for any $f' \in F$, $f' \leq_F f$, it holds that $h_D(f') \leq_{\mathcal{A}R} h_D(f)$.

Proof. Directly from the definition of a solution map we have that:

$$\begin{aligned} h_D(f) &= \operatorname{Min}\{r \in R \mid D(f,r) = \mathtt{T}\}, \\ h_D(f') &= \operatorname{Min}\{r \in R \mid D(f',r) = \mathtt{T}\}. \end{aligned}$$

Due to the monotonicity of D we know that if $D(f,r) = \mathtt{T}$ for some r, then $D(f',r) = \mathtt{T}$. Hence,

$$\{r \in R \mid D(f,r) = \mathtt{T}\} \subseteq \{r \in R \mid D(f',r) = \mathtt{T}\}.$$

Note that the two sides of the above equations are feasible sets of resources and therefore, due to Lemma 8.13 they are upper sets. Hence, assuming Remark 8.14, Equation (8.1) holds for both of them:

$$\uparrow\operatorname{Min}\{r \in R \mid D(f,r) = \mathtt{T}\} \subseteq \uparrow\operatorname{Min}\{r \in R \mid D(f',r) = \mathtt{T}\}.$$

It follows from the definition of the order on antichains (Lemma 8.4) that the following must also hold true:

$$h_D(f') \leq_{\mathcal{A}R} h_D(f).$$

□

Remark 8.17. It is important to mention that co-design also has the concept of *dual problems*. That is, finding the maximal antichain of functionalities that can be achieved with a given resource. The whole treatment is symmetric and one would need a dual solution map. We will show in Chapter 9 that any dual problem can be represented as a problem in the original setting we presented here. Therefore, we will not concern ourselves with the solution techniques for dual problems.

8.2 Composition of solution maps

Now, we need to show what every single one of the feasibility relation composition operations that we defined in Section 7.2 does with the solution maps. Again, these composition operations would be correct only if the duality between minimum elements and upper sets holds (Remark 8.14).

Lemma 8.18 (Solution map of the composite feasibility relation). *Given two feasibility relations $\Phi : P \to Q$ and $\Psi : Q \to R$ with solution maps $h_\Phi\colon P \to \mathcal{A}Q$ and $h_\Psi\colon Q \to \mathcal{A}R$ respectively, the solution map $h_\Phi \mathbin{⨾} h_\Psi$ for the composite feasibility relation $\Phi \mathbin{⨾} \Psi$ is:*

$$\begin{aligned} h_\Phi \mathbin{⨾} h_\Psi\colon P &\to \mathcal{A}R \\ p &\mapsto \operatorname*{Min}_{\leq_R} \bigcup_{s \in h_\Phi(p)} h_\Psi(s). \end{aligned}$$

Lemma 8.19 (Solution map of the identity feasibility relation). *Given an identity feasibility relation $\Phi^{\mathrm{id}}_P : P \to P$, its solution map is:*

$$\begin{aligned} h_{\Phi^{\mathrm{id}}_P}\colon P &\to \mathcal{A}P \\ p &\mapsto \{p\}. \end{aligned}$$

Lemma 8.20 (Solution map of the product feasibility relation). *Given two feasibility relations $\Phi : P \to Q$ and $\Psi : R \to S$ with solution maps $h_\Phi\colon P \to \mathcal{A}Q$ and $h_\Psi\colon R \to \mathcal{A}S$ respectively, the solution map $h_\Phi \times h_\Psi$ for the composite feasibility relation $\Phi \times \Psi$ is:*

$$\begin{aligned} h_\Phi \times h_\Psi\colon P \times R &\to \mathcal{A}(Q \times S) \\ (p, r) &\mapsto h_\Phi(p) \times h_\Psi(r). \end{aligned}$$

Lemma 8.21 (Solution map of the sum feasibility relation). *Given two feasibility relations $\Phi : P \to Q$ and $\Psi : P \to Q$ with solution maps $h_\Phi\colon P \to \mathcal{A}Q$ and $h_\Psi\colon P \to \mathcal{A}Q$ respectively, the solution map $h_\Phi \vee h_\Psi$ for the composite feasibility relation $\Phi \vee \Psi$ is:*

$$h_\Phi \vee h_\Psi\colon P \to \mathcal{A}Q$$

$$p \mapsto \operatorname*{Min}_{\leq_Q} \left(h_\Phi(p) \cup h_\Psi(p) \right).$$

Lemma 8.22 (Solution map of the intersection feasibility relation)**.** *Given two feasibility relations $\Phi : P \to Q$ and $\Psi : P \to Q$ with solution maps $h_\Phi \colon P \to \mathcal{A}Q$ and $h_\Psi \colon P \to \mathcal{A}Q$ respectively, the solution map $h_\Phi \wedge h_\Psi$ for the composite feasibility relation $\Phi \wedge \Psi$ is:*

$$\begin{aligned} h_\Phi \wedge h_\Psi \colon P &\to \mathcal{A}Q \\ p &\mapsto \operatorname*{Min}_{\leq_Q} \left((\uparrow h_\Phi(p)) \cap (\uparrow h_\Psi(p)) \right), \end{aligned}$$

as long as

$$\uparrow \operatorname*{Min}_{\leq_Q} \left((\uparrow h_\Phi(p)) \cap (\uparrow h_\Psi(p)) \right) = (\uparrow h_\Phi(p)) \cap (\uparrow h_\Psi(p)).$$

Lemma 8.23. *Let $P = (P, \leq_P)$ be a poset in which all joins exist and $\{A_i\}$ be a family of non-empty elements of $\mathcal{A}P$. Then it holds that*

$$\operatorname*{Min}_{\leq_P} \left(\bigcap_{i \in I} \uparrow A_i \right) = \operatorname*{Min}_{\leq_P} \left\{ \bigsqcup_{i \in I} a_i \mid a_i \in A_i,\ \forall i \in I \right\}.$$

Proof. First, let's show that any $x \in \operatorname{Min}_{\leq_P} (\bigcap_{i \in I} \uparrow A_i)$ is also in the right-hand side of the equation. For this x it holds that $x \in \uparrow A_i, \forall i \in I$, and if there is an $x' \in \uparrow A_j,\ \forall j \in I$, then $x' = x$. As $x \in \uparrow A_i$ we also know that for all i there exists a $x_i^* \in A_i$ such that $x_i^* \leq_P x$. Now take

$$X = \bigsqcup_{i \in I} x_i^* \in \left\{ \bigsqcup_{i \in I} a_i \mid a_i \in A_i,\ \forall i \in I \right\}.$$

Due to the properties of the join, we know that $x_i^* \leq_P X,\ \forall i \in I$, hence $X \in A_i,\ \forall i \in I$. But we said above that if such an x' exists, then it should equal x. Thus

$$x = X = \bigsqcup_{i \in I} x_i^* \in \left\{ \bigsqcup_{i \in I} a_i \mid a_i \in A_i,\ \forall i \in I \right\}.$$

Now we only need to show that it is a minimal element of this set. Assume by contradiction that

$$x \notin \operatorname*{Min}_{\leq_P} \left\{ \bigsqcup_{i \in I} a_i \mid a_i \in A_i,\ \forall i \in I \right\}.$$

Then there must be a collection $\{b_i \mid b_i \in A_i\}$ such that $\bigsqcup_{i \in I} b_i <_P x$. The join $\bigsqcup_{i \in I} b_i$ should be in $\bigcap_{i \in I} \uparrow A_i$, but then x cannot be in $\operatorname{Min}_{\leq_P} (\bigcap_{i \in I} \uparrow A_i)$, which is our contradiction. Therefore:

$$\operatorname*{Min}_{\leq_P} \left(\bigcap_{i \in I} \uparrow A_i \right) \subseteq \operatorname*{Min}_{\leq_P} \left\{ \bigsqcup_{i \in I} a_i \mid a_i \in A_i,\ \forall i \in I \right\}.$$

Now, let's show that any $x \in \mathrm{Min}_{\leq_P} \{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\}$ is also in $\mathrm{Min}_{\leq_P} (\bigcap_{i\in I} \uparrow A_i)$. We need to show two things: first, that $x \in \bigcap_{i\in I} \uparrow A_i$, and second that there is no $y <_P x$ also in $\bigcap_{i\in I} \uparrow A_i$. We know that we can represent x as $\bigsqcup_{i\in I} x_i^*$, where $x_i^* \in A_i$. Then:

$$x_i^* \leq_P x,\ \forall i \in I \implies x \in \uparrow A_i,\ \forall i \in I \implies x \in \bigcap_{i\in I} \uparrow A_i.$$

Now, for the second requirement, assume, by contradiction, that a lesser element y exists. Then $y \in \uparrow A_i$ and exists a $y_i^* \in A_i$, s.t. $y_i^* \leq_P y$, for all $i \in I$. It follows that

$$\bigsqcup_{i\in I} y_i^* \in \left\{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\right\} \leq_P y \leq_P x.$$

But then, either $\bigsqcup_{i\in I} y_i^* = x$ or $x \notin \mathrm{Min}_{\leq_P} \{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\}$. This is a contradiction. Therefore:

$$\underset{\leq_P}{\mathrm{Min}} \left(\bigcap_{i\in I} \uparrow A_i\right) \supseteq \underset{\leq_P}{\mathrm{Min}} \left\{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\right\},$$

and the two sides of the equation in the theorem are indeed equal. □

Lemma 8.24 (Solution map of the intersection feasibility relation (alternative))**.** *Given two feasibility relations $\Phi : P \to Q$ and $\Psi : P \to Q$ with solution maps $h_\Phi : P \to \mathcal{A}Q$ and $h_\Psi : P \to \mathcal{A}Q$ respectively, the solution map $h_\Phi \wedge h_\Psi$ for the composite feasibility relation $\Phi \wedge \Psi$ is:*

$$\begin{aligned} h_\Phi \wedge h_\Psi &: P \to \mathcal{A}Q \\ & p \mapsto \underset{\leq_Q}{\mathrm{Min}} \{q_1 \sqcup q_2 \mid q_1 \in h_\Phi(p),\ q_2 \in h_\Psi(p)\}, \end{aligned}$$

if the join $q_1 \sqcup q_2$ (Definition 7.5) exists for all $q_1, q_2 \in Q$ and

$$\uparrow \underset{\leq_Q}{\mathrm{Min}} \left((\uparrow h_\Phi(p)) \cap (\uparrow h_\Psi(p))\right) = (\uparrow h_\Phi(p)) \cap (\uparrow h_\Psi(p)).$$

Proof. Follows directly from Lemma 8.23.

□

8.3 Fixpoints and feedback

The one type of composition that we have not yet provided a solution map for is the feedback loop. Feedback loops can be used when a design problem has the same poset as a resource and as a functionality. Then we can feed the provided functionality as the resource, making it at least partially self-sufficient. This is a very powerful tool so we refer to curious reader to (Censi, 2016; Censi et al., 2020) where feedback in design problems is discussed in a great detail.

Given a design problem $\Phi\colon F \times R \nrightarrow R$ and its solution map h_Φ, we define the loop(Φ) problem which feeds the resource as part of the functionality as:

$$\begin{aligned}\Phi_{\text{loop}}\colon \quad & F^{\text{op}} \times R \to \texttt{Bool},\\ \langle f, r\rangle \mapsto & \begin{cases}\texttt{T} & \text{if } \exists r' \leq_R r,\ \Phi((f, r'), r) = \texttt{T},\\ \texttt{F} & \text{otherwise.}\end{cases}\end{aligned}$$

Computing the solution map $h_{\Phi_{\text{loop}}}$ however, is more complicated as it must be the solution of

$$\begin{aligned}h_{\Phi_{\text{loop}}}\colon & F \to R\\ f \mapsto & \begin{cases}\text{Using} & r, r' \in R,\\ \text{Min}_{\leq_R} & r,\\ \text{s.t.} & r \in h_\Phi(f, r'),\\ & r \leq_R r'.\end{cases}\end{aligned}$$

We can fix the f in h_Φ in order to obtain a new design problem that has as a resource and as a functionality only R. We call this h_Φ^f :

$$\begin{aligned}h_\Phi^f\colon & R \to \mathcal{A}R,\\ & r' \mapsto h_\Phi(f, r').\end{aligned}$$

Lemma 8.25. *Let A be an antichain in the poset P. Then*

$$a \in A \iff \{a\} = A \cap \uparrow a.$$

Proof. First, let's show the $\implies$ direction. If $a \in A$, then we have

$$A \cap \uparrow a = A \cap \{p \in P \mid a \leq_P p\}.$$

But, as A is an antichain, the only element $a' \in A$ such that $a \leq_P a'$ is a itself. Therefore, the intersection can only contain a.

The $\impliedby$ direction is immediately obvious: if a is in an intersection of A with another set, then $a \in A$. □

By using h_Φ^f and Lemma 8.25, we know that for every feasible $r, r' \in R$ according to Section 8.3 the following must hold:

$$\{r\} = h_\Phi^f(r') \cap \uparrow r. \tag{8.5}$$

But we also know that $r \leq_R r'$, and hence by Lemma 8.16 we have that $h^f_\Phi(r) \leq_{\mathcal{A}R} h^f_\Phi(r')$. We can use this result, together with the following lemma to get rid of r'.

Lemma 8.26. *Let $A, B \in \mathcal{A}P$ and $S \subseteq P$, where P is the poset $(P, \leq_P)$. Then, $A \leq_{\mathcal{A}P} B$ implies $A \cap S \leq_{\mathcal{A}P} B \cap S$.*

Proof. Take an element $b \in \uparrow(B \cap S)$, then $b \in B \wedge b \in S$. It is trivial to see that $S \subseteq \uparrow(A \cap S)$. Hence, $b \in \uparrow(A \cap S)$. As this holds for any $b \in \uparrow(B \cap S)$, it holds that $\uparrow(A \cap S) \supseteq \uparrow(B \cap S)$. □

Now we can remove the r' term from Equation (8.5):

$$h^f_\Phi(r) \cap \uparrow r \leq_{\mathcal{A}R} h^f_\Phi(r') \cap \uparrow r = \{r\}. \tag{8.6}$$

Any feasible r should satisfy this recursive condition. Here by *feasible* we mean an r that satisfies the conditions in Section 8.3 and one that is not reducible to any other r^* that also satisfies the conditions. In other words, any r in the solution antichain of Section 8.3 must satisfy Equation (8.6).

Instead of considering the elements of the antichain separately, we would like to instead obtain a condition for the whole antichain. Call the antichain of solutions $R \in \mathcal{A}R$. Then we can tautologically rewrite it as:

$$R = \operatorname{Min} \bigcup_{r \in R} \{r\}. \tag{8.7}$$

Lemma 8.27. *Let $A, B, C, D \in \mathcal{A}P$ and $A \leq_{\mathcal{A}P} C, B \leq_{\mathcal{A}P} D$, where P is the poset $(P, \leq_P)$. Then, $A \cup B \leq_{\mathcal{A}P} C \cup D$.*

Proof. Note that $A \leq_{\mathcal{A}P} C \implies \uparrow A \supseteq \uparrow C$, and $B \leq_{\mathcal{A}P} D \implies \uparrow B \supseteq \uparrow D$. If $e \in \uparrow(C \cup D)$, then there should exist an $e' \in C \cup D$, such that $e \leq_P e'$. Then there are three possible cases: $e' \in C \subseteq \uparrow A$, $e' \in D \subseteq \uparrow B$, or e' is in both. Hence we have that $e \in \uparrow A$ and/or $e \in \uparrow B$. But $\uparrow A \subseteq \uparrow(A \cup B)$, because for every $a \in \uparrow A$, an $a' \in A$, such that $a' \leq_P a$ should exist. But this a' is also in $A \cup B$ and hence the same a is also in $\uparrow(A \cup B)$. Similarly, $\uparrow B \subseteq \uparrow(A \cup B)$. Therefore, $e \in \uparrow(A \cup B)$, and that holds for any $e \in \uparrow(C \cup D)$. □

Combining Equations (8.6) and (8.7) and using Lemma 8.27, we get:

$$\operatorname{Min} \bigcup_{r \in R} h^f_\Phi(r) \cap \uparrow r \leq_{\mathcal{A}R} \operatorname{Min} \bigcup_{r \in R} \{r\} = R. \tag{8.8}$$

One can verify that all chains of feasible resources R satisfy Equation (8.8) and if an antichain R satisfies Equation (8.8), then it must be an antichain of feasible resources. We can rewrite the constraint in Equation (8.8) as:

$$\Phi_f(R) \leq_{\mathcal{A}R} R, \tag{8.9}$$

where Φ_f is defined as:

$$\begin{aligned}\Phi_f \colon \mathcal{A}R &\to \mathcal{A}R,\\ R &\mapsto \mathrm{Min} \bigcup_{r \in R} h_\Phi^f(r) \cap \uparrow r \leq_{\mathcal{A}R} .\end{aligned}$$

Therefore, the optimization problem in Section 8.3 can be rewritten as the problem of finding the minimum antichain in $\mathcal{A}R$ that satisfies Equation (8.9). This is the *least fixpoint* of Φ_f. *Fixpoint* refers to the fact that $\Phi_f(R) \leq_{\mathcal{A}R} R$, where we don't require equality (this condition is also referred to as *pre-fixpoint*). *Least* refers to the fact that we are interested in the antichain that precedes all antichains that are fixed points. The final form of Section 8.3 is thus:

$$h_{\Phi_{\text{loop}}} \colon f \mapsto \mathrm{lfp}(\Phi_f).$$

The least fixpoint of Φ_f does not exist in general. However, it exists if $\mathcal{A}R$ is a *pointed directed-complete partial order* and if Φ_f is *Scott-continuous*. If that is the case, then the *Kleene Fixpoint Theorem* states that Φ_f has a least fixpoint and provides a way for computing it.

Definition 8.28 (Directed subset)**.** Given a poset $P = (P, \leq_P)$, a subset of its elements $S \subseteq P$ is called a *directed subset* if for any $a, b \in S$ exists a $c \in S$ such that $a \leq c$ and $b \leq c$.

Definition 8.29 (Directed-complete poset)**.** A poset $P = (P, \leq_P)$ is a *directed-complete poset* if each of its directed subsets has a join (Definition 7.5).

Definition 8.30 (Pointed directed-complete poset)**.** A poset $P = (P, \leq_P)$ is a *pointed directed-complete poset* if it is directed-complete and also has a bottom element.

Definition 8.31 (Scott-continuous function)**.** Given two posets $P = (P, \leq_P)$ and $Q = (Q, \leq_Q)$, a function $f \colon P \to Q$ is called *Scott-continuous* if it preserves the joins of all directed subsets. In other words, for any directed subset $D \subseteq P$, the image $f(D)$ is also directed and furthermore

$$f(\sqcup D) = \sqcup\{f(d) \mid d \in D\}.$$

Lemma 8.32. *If a function $f \colon P \to Q$ is Scott-continuous, then it is also monotonic.*

Proof. Take two elements from $a, b \in P$ such that $a \leq_P b$. Then, due to the Scott-continuity of f we have that

$$f(\sqcup\{a, b\}) = f(b) = \sqcup\{f(a), f(b)\}.$$

But then, due to the definition of the join, it must hold that $f(a) \leq_P f(b)$, hence f is monotonic. □

Theorem 8.33 (Kleene Fixpoint Theorem). *Take any pointed directed-complete partial order (poset) $P = (P, \leq_P)$ and a Scott-continuous endofunction $f : P \to P$. Then f has a least fixpoint which is the join of the ascending Kleene chain of f:*

$$\bot \leq_P f(\bot) \leq_P f^2(\bot) \leq_P \ldots .$$

Hence,

$$\mathrm{lfp}(f) = \sqcup\{f^n(\bot) \mid n \in \mathbb{N}\}.$$

Proof. First, we have to show that indeed $f^n(\bot) \leq_P f^{n+1}(\bot), \forall n \in \mathbb{N}_0$. As $\bot$ precedes all elements of P, it holds that $\bot = f^0(\bot) \leq f^1(\bot)$. Now assume that $f^{n-1}(\bot) \leq_P f^n(\bot)$ for some n. Due to Lemma 8.32 it holds that $f(f^{n-1}(\bot)) \leq_P f(f^n(\bot))$. Hence, by induction, $f^n(\bot) \leq_P f^{n+1}(\bot), \forall n \in \mathbb{N}_0$.

Denote the chain $\{f^n(\bot) \mid n \in \mathbb{N}\}$ by K. As every chain is a directed subset, and as P is directed-complete, $k = \sqcup K$ exists. Note that for any non-empty $S \subseteq P$, it holds that $\sqcup S = \sqcup(S \cup \{\bot\})$. Hence, $\sqcup K = \sqcup\{f(k') \mid k' \in K\} = k$. Due to the Scott-continuity of f, we know that

$$f(k) = f(\sqcup K) = f(\sqcup\{f(k') \mid k' \in K\}) = \sqcup\{f(k') \mid k' \in K\} = k.$$

Therefore, k is a fixpoint.

Now, we only need to show it is the least fixpoint. Assume that there is another fixpoint $k^* \in P$. Let's show that $k \leq_P k^*$. First of all, all elements of K precede k^*. Trivially, $f^0(\bot) = \bot \leq_P k^*$. If $f^n(\bot) \leq_P k^*$, due to the monotonicity of f it holds that $f^{n+1}(\bot) \leq_P f(k^*) \leq_P k^*$, where the last equality is due to k^* being a fixpoint. Then by induction, all elements of K precede k^*. If all the elements of K precede k^*, then their join also precedes it, hence $k = \sqcup K \leq_P k^*$. □

When we solve a loop problem, the poset on which we are computing the least fixpoint is the partial order (poset) of antichains on the resources. Hence we would have to show that for a given R, $\mathcal{A}R$ is a pointed directed-complete partial order. Luckily, it so happens that *any* (useful) antichain poset is a pointed directed-complete partial order.

Theorem 8.34. *Take any poset $P = (P, \leq_P)$ for which the join $\sqcup Q$ exists for any subset $Q \subseteq P$, and a subset of its antichains $S \subseteq \mathcal{A}P$ such that*

$$\bigcap_{T \in S'} \uparrow T = \uparrow \mathrm{Min}\left(\bigcap_{T \in S'} \uparrow T\right), \ \forall S' \in \mathcal{P}S.$$

In other words, Equation (8.1) (i.e. the conditions in Theorem 8.7) hold for the union of every subset of the upper sets of the antichains in S. Then, the poset $S = (S, \leq_{\mathcal{A}P})$ is a pointed directed-complete partial order.

Proof. First, S is a subset of $\mathcal{A}P$, and the antichains order induces posetal structure (Lemma 8.4, hence S is a poset. It is pointed, because the bottom element of $\mathcal{A}P$, which is P itself, is trivially in S. Take any subset of elements of S. We will refer to the elements of this subset as A_i, indexed by the set I. We argue that their join exists, and that it is

$$J = \operatorname{Min}\left(\bigcap_{i\in I} \uparrow A_i\right) = \operatorname{Min}\left\{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\right\}. \tag{8.10}$$

The above equality is due Lemma 8.23. The join exists because we require that the joins exist for all elements of P and because Min is defined for all subsets of P.

We need to show that $J \in S$. Take any potentially empty subset S' of the elements of S. Then combining J with the condition in the theorem we get:

$$\uparrow J \cap \left(\bigcap_{T\in S'} \uparrow T\right) = \left(\uparrow \operatorname{Min}\left(\bigcap_{i\in I} \uparrow A_i\right)\right) \cap \left(\bigcap_{T\in S'} \uparrow T\right).$$

As $\{A_i\} \subseteq S$, the condition should also hold for it:

$$\begin{aligned}\uparrow J \cap \left(\bigcap_{T\in S'} \uparrow T\right) &= \left(\bigcap_{i\in I} \uparrow A_i\right) \cap \left(\bigcap_{T\in S'} \uparrow T\right)\\ &= \bigcap_{T\in S'\cup\{A_i\}} \uparrow T\\ &= \uparrow \operatorname{Min}\left(\bigcap_{T\in S'\cup\{A_i\}} \uparrow T\right).\end{aligned}$$

As this holds for any J constructed from elements of S and for any subset S' of S, $J \in S$.

Now, let's also show that J is indeed the join of $\{A_i\}$. For it to be the join, in needs to reduce to every A_i and to be the least element of S with this property. From Equation (8.10) we know that for every $x \in J$ there exist $x_i \in A_i$ such that $x = \bigsqcup_{i\in I} x_i$. Then, $x_i \in \uparrow A_i,\ \forall i \in I$. But as $x_i \leq_P x$ it also holds that $x \in \uparrow A_i,\ \forall i \in I$. Hence, $\uparrow A_i \supseteq J,\ \forall i \in I$, or alternatively, $A_i \leq_{\mathcal{A}P} J,\ \forall i \in I$.

To show that it is the least antichain with this property, assume by contradiction that there is a J' such that $A_i \leq_{\mathcal{A}P} J',\ \forall i \in I$ and $J' <_{\mathcal{A}P} J$. Take any $y \in \uparrow J'$, and any $y_j \in A_j$ such that $y_j \leq_P y$, and such y_j exists for all $j \in I$ as $\uparrow J' \subseteq \uparrow A_j$. Then we have $\bigsqcup_{j\in I} y_j \leq_P y$, but also $\bigsqcup_{j\in I} y_j \in \{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\}$. For any $x \in \operatorname{Min}\{\bigsqcup_{i\in I} a_i \mid a_i \in A_i,\ \forall i \in I\} = J$ it then holds that $x \leq_P \bigsqcup_{j\in I} y_j \leq_P y$. This holds for any $x \in J$ and $y \in J'$, hence $\uparrow J \supseteq \uparrow J'$ and $J \leq_{\mathcal{A}P} J'$. □

The essence of Theorem 8.34 is that, as long as one restricts themselves to the subset of antichains satisfying the condition of the theorem, one can safely apply Theorem 8.33.

The condition of Theorem 8.34 seems quite restrictive at first. However, it is also a necessity if one wants to have the duality between minimal elements and intersections of upper sets to hold. For finite sets, for example, the condition holds trivially.

Remark 8.35. In the proof of Theorem 8.34 we never restricted that $\{A_i\}$ is directed. One can see that it always is. Hence, the subset S is in fact a pointed *join-semilattice*, or a partially ordered set that has a join for any nonempty set, and also has a bottom element.

Remark 8.36. The identity feasibility relation is Scott-continuous. Furthermore, if Φ and Ψ are Scott-continuous, then so are $\Phi \mathbin{⨟} \Psi$, $\Phi \times \Psi$, $\Phi \vee \Psi$, $\Phi \wedge \Psi$, and Φ_{loop}, as long as these exist. This is discussed by Censi (2016), and the proofs can be found in (Gierz et al., 2003).

8.4 Well-behaved design problems

In this chapter we looked at a number of different conditions that allow us to perform composition operations on design problems. In practice, however, we don't want to keep track of the properties of each design problem and to wonder whether if we compose it it will still be well-behaved. Therefore, let's combine all the conditions that allow us to have an isomorphism between minimal elements and upper sets, to take intersections of antichains representing upper sets, and to be able to apply Kleene's theorem. It is important to see that these are not properties of the poset alone, but are more so of the feasible sets of a given design problem that has it as a resource. Hence, we will call any design problem that has these properties a *well-behaved* one.

Definition 8.37 (Well-behaved upper sets)**.** Given a poset $P = (P, \leq_P)$ and its set of upper sets $\mathsf{U}P$, we call a subset of the upper sets $\mathsf{U}^\star P \subseteq \mathsf{U}P$ the *set of well-behaved upper sets* if for any subset $P' \subseteq \mathsf{U}^\star P$ it holds that

$$\bigcap_{p \in P'} p = \uparrow \mathrm{Min}\left(\bigcap_{p \in P'} p \right),$$

which is equivalent to conditions in Theorem 8.7 for the intersection of the upper sets, and if furthermore for any other subset $\mathfrak{U}'$ with the same property, it holds that $\mathfrak{U}' \subseteq \mathsf{U}^\star P$. In other words, $\mathsf{U}^\star P$ is the universal set with this property.

Definition 8.38 (Primally well-behaved design problem). A design problem $D : F \nrightarrow R$, with a solution map

$$h_D(f) = \mathrm{Min}\{r \in R \mid D(f,r) = \mathtt{T}\},$$

is *primally* well-behaved if:

i. for any subset $R' \subseteq R$ there is a join element $\sqcup R' \in R$;
ii. all feasible sets of resources are well-behaved upper sets, i.e.

$$\{r \in R \mid D(f,r) = \mathtt{T}\} \in \mathsf{U}^{\star}R, \ \forall f \in F;$$

iii. the solution map h_D is Scott-continuous.

We mentioned before that in co-design there are also dual problems. While we haven't provided formal definition of the dual problem and the means to solve it, it suffices to mention that it behaves symmetrically to the primal problem. It will be discussed in further detail in Chapter 9. In order for the dual problem to also be well-defined we can extend the definition to:

Definition 8.39 (Well-behaved design problems). A design problem $D : F \nrightarrow R$, with a primal solution map

$$h_D(f) = \mathrm{Min}\{r \in R \mid D(f,r) = \mathtt{T}\}$$

and a dual solution map

$$h'_D(r) = \mathrm{Max}\{f \in F \mid D(f,r) = \mathtt{T}\},$$

is well-behaved if:

i. for any subset $R' \subseteq R$ there is a join element $\sqcup R' \in R$;
ii. for any subset $F' \subseteq F$ there is a meet element $\sqcap F' \in F$;
iii. all feasible sets of resources for the primal problem are well-behaved upper sets, i.e.

$$\{r \in R \mid D(f,r) = \mathtt{T}\} \in \mathsf{U}^{\star}R, \ \forall f \in F;$$

iv. all feasible sets of functionalities for the dual problem are well-behaved upper sets of the F^{op} poset (well-behaved lower sets of F), i.e.

$$\{f \in F \mid D(f,r) = \mathtt{T}\} \in \mathsf{U}^{\star}F^{\mathrm{op}}, \ \forall r \in R;$$

v. the primal solution map h_D is Scott-continuous;
vi. the dual solution map h'_D is Scott-continuous in the inverse posets F^{op} and R^{op}.

As long as a design problem is well-behaved, we can identify every upper set with an antichain and Equation (8.1) holds. We can also take the intersection of design problems (Lemmas 8.22 and 8.24). And we can also close feedback loops (Theorem 8.33), because thanks to Theorem 8.34 we know that the antichains for the resource upper sets for this feasibility relation form a pointed directed-complete partial order.

Chapter 9

Compositional properties of co-design

In Section 7.2 we showed that co-design problems can be composed in various ways. Then, in Chapter 8 we saw how each composition of design problems can be solved via composing their solution maps. In the current chapter, we will show that this compositionality not only happens to be a very nice property but also has a strong structural representation in the way of categorical constructions.

We are going to see how co-design problems and their solutions form categories and the relationships between them are not only functorial, but also have much stronger properties. This will be the main argumentation behind using co-design as an example of a collection of problems and solutions in compositional computational systems that are rich in structure, something that will be the main focus of the next chapter.

9.1 The categories of upper and lower sets

We will show how co-design problems form a category, a result originally by Censi et al. (2020). The feasible sets for the primal design problems form upper sets, which can also be thought of as objects in a category. Similarly, the feasible sets for the dual design problems form lower sets with their own category.

Definition 9.1 (The **DP** category)**.** The *category of design problems*, **DP**, consists of:

i. objects which are posets;
ii. morphisms between two objects $A, B \in \mathsf{Ob}(\mathbf{DP})$ are design problems $d : A \nrightarrow B$ (formally defined as feasibility relations, Definition 2.14);
iii. composition of two morphisms $f\colon A \nrightarrow B$ and $g\colon B \nrightarrow C$, denoted by $(f \mathbin{⨟} g)\colon A \nrightarrow C$, given by Definition 7.17;
iv. identity morphism $\mathrm{id}_A\colon A \nrightarrow A$, given by Lemma 7.21.

Lemma 9.2. **DP** *is indeed a category.*

Proof. We already showed that the $⨟$ operator is associative (Lemma 7.19) and that the composition of two design problems is a design problem (Lemma 7.18). It is also easy to see from the definition (Definition 7.20) of the identity design problem that $⨟$ is also unital. □

Definition 9.3 (The **UPos** category)**.** The **UPos** category has:
i. objects which are posets;
ii. morphisms between two objects $A, B \in \mathsf{Ob}(\mathbf{UPos})$ are monotone maps of the form $f\colon A \to \mathsf{U}B$ (recall Definition 2.12);
iii. composition of two morphisms $\alpha : A \to \mathsf{U}B$ and $\beta : B \to \mathsf{U}C$ defined as the "fish" operator:

$$\begin{aligned}\alpha \ltimes \beta\colon A &\to \mathsf{U}C,\\ a &\mapsto \bigcup_{b \in \alpha(a)} \beta(b);\end{aligned}$$

iv. identity morphism for an object A: $\mathrm{id}_A(a) := \uparrow\{a\}$ (Definition 7.1).

Lemma 9.4. **UPos** *is indeed a category.*

Proof. First, let's show unitality. Given $\alpha\colon A \to \mathsf{U}B$, we have:

$$\begin{aligned}(\alpha \ltimes \mathrm{id}_B)(a) &= \bigcup_{b \in \alpha(a)} \mathrm{id}_B(b)\\ &= \bigcup_{b \in \alpha(a)} \uparrow\{b\}\\ &= \bigcup_{b \in \alpha(a)} \{b' \in B\colon b \leq_B b'\}.\end{aligned}$$

We can rewrite $\alpha(a)$ as:

$$\begin{aligned}\alpha(a) &= \bigcup_{b \in \alpha(a)} \{b\}\\ &= \bigcup_{b \in \alpha(a)} \{b' \in B\colon b \leq_B b'\},\end{aligned}$$

where the second equality is due to $\alpha(a)$ being an upper set. Thus,

$$(\alpha \ltimes \mathrm{id}_B)(a) = \alpha(a).$$

Similarly,

$$\begin{aligned}(\mathrm{id}_A \ltimes \alpha)(a) &= \bigcup_{a' \in \mathrm{id}_A(a)} \alpha(a') \\ &= \bigcup_{a' \in \uparrow\{a\}} \alpha(a') \\ &= \alpha(a),\end{aligned}$$

with the last equality due to α being monotonic and due to the definition of the order on upper sets (Lemma 7.2), resulting in $\alpha(a') \subseteq \alpha(a),\ \forall a' \in \uparrow\{a\}$.

We also need to show that the fish operator is associative. Take three morphisms $\alpha : A \to \mathsf{U}B$, $\beta : B \to \mathsf{U}C$, and $\gamma : C \to \mathsf{U}D$, then:

$$\begin{aligned}\big((\alpha \ltimes \beta) \ltimes \gamma\big)(a) &= \bigcup_{c \in \left(\bigcup_{b \in \alpha(a)} \beta(b)\right)} \\ \gamma(c) &= \bigcup_{b \in \alpha(a)} \bigcup_{c \in \beta(b)} \gamma(c),\end{aligned}$$

and

$$\big(\alpha \ltimes (\beta \ltimes \gamma)\big)(a) = \bigcup_{b \in \alpha(a)} \bigcup_{c \in \beta(b)} \gamma(c),$$

demonstrating associativity.

Finally, we need to make sure that the output of $\ltimes$ is also an upper set, which trivially follows from the definition of the operator. □

Definition 9.5 (The **LPos** category)**.** The **LPos** category has:

i. objects which are posets;
ii. morphisms between two objects $A, B \in \mathsf{Ob}(\mathbf{LPos})$ are monotone maps of the form $f : A \to \mathsf{L}B$;
iii. composition of two morphisms $\alpha : A \to \mathsf{L}B$ and $\beta : B \to \mathsf{L}C$ is the fish operator:

$$\begin{aligned}\alpha \ltimes \beta \colon A &\to \mathsf{L}C, \\ a &\mapsto \bigcup_{b \in \alpha(a)} \beta(b);\end{aligned}$$

iv. identity morphism for an object A: $\mathrm{id}_A(a) = \downarrow\{a\}$ (Definition 7.3).

Lemma 9.6. LPos *is indeed a category.*

Proof. Analogous to the proof of Lemma 9.4. □

We will now show that the **UPos** and **LPos** categories are equivalent.

Definition 9.7 (Equivalence of categories)**.** An *equivalence* between two categories **C** and **D** is:

i. a pair of functors F, G:

$$\begin{tikzcd} \mathbf{C} \arrow[r, bend right, "F"'] & \mathbf{D}; \arrow[l, bend right, "G"'] \end{tikzcd}$$

ii. natural isomorphisms (Definition 2.43)

$$F \fatsemi G \cong \mathrm{id}_{\mathbf{C}},$$

and

$$G \fatsemi F \cong \mathrm{id}_{\mathbf{D}}\,.$$

Two categories are called *equivalent* if there exists an equivalence between them.

This definition of equivalence of categories is due Lane (1998, Sec. IV.4).

Remark 9.8. Note that $\mathbf{LPos}^{\mathrm{op}} \neq \mathbf{UPos}$, because the morphisms are different, just flipping the direction does not suffice. The next lemma, makes this remark more precise.

Lemma 9.9 (**UPos** and **LPos** are equivalent)**.** *There is a pair of functors*

$$\swarrow : \mathbf{UPos} \to \mathbf{LPos}$$

and

$$\nearrow : \mathbf{LPos} \to \mathbf{UPos}$$

such that $\swarrow \fatsemi \nearrow = \mathrm{id}_{\mathbf{UPos}}$ *and* $\nearrow \fatsemi \swarrow = \mathrm{id}_{\mathbf{LPos}}$*, where* $\mathrm{id}_{\mathbf{UPos}}$ *and* $\mathrm{id}_{\mathbf{LPos}}$ *are the identity functors on* **UPos** *and* **LPos***, respectively. In other words,* $\swarrow$ *and* $\nearrow$ *are inverses of each other and* **UPos** *and* **LPos** *are equivalent categories (Definition 9.7). We call* $\swarrow$ lower flip *and* $\nearrow$ upper flip.

Proof. We prove the lemma by giving explicit definitions for the $\swarrow$ and $\nearrow$ functors. Both **UPos** and **LPos** have the same objects, which are posets. We know that for every poset P, there's the opposite poset P^{op} (Definition 2.7). Therefore, we choose our two functors such that they map the poset P in the origin category to the poset P^{op} in the target category.

Given a morphism $\alpha : A \to \mathsf{U}B$ in **UPos**, we have:

$$\begin{aligned} \swarrow(\alpha) \colon A^{\mathrm{op}} &\to \mathsf{L}(B^{\mathrm{op}}), \\ a &\mapsto \alpha(a). \end{aligned}$$

Similarly, given a morphism $\beta : A \longrightarrow \mathsf{L}B$ in **LPos**, we have:

$$\begin{aligned} \nearrow(\beta) \colon A^{\mathrm{op}} &\longrightarrow \mathsf{U}(B^{\mathrm{op}}), \\ a &\longmapsto \beta(a). \end{aligned}$$

Hence, the functors maintain the subset to which they map, but by flipping the order they change the context of this subset from upper set to a lower set and back.

For $\swarrow$ and $\nearrow$ to be functors, they need to preserve compositions and identities. Composition is preserved because the functors keep the sets unchanged:

$$\swarrow(\alpha \ltimes \beta) = \alpha \ltimes \beta = \swarrow(\alpha) \ltimes \swarrow(\beta), \quad \alpha \in \mathrm{Hom}_{\mathbf{UPos}}(A, B),\ \beta \in \mathrm{Hom}_{\mathbf{UPos}}(B, C),$$

$$\nearrow(\alpha \ltimes \beta) = \alpha \ltimes \beta = \nearrow(\alpha) \ltimes \nearrow(\beta), \quad \alpha \in \mathrm{Hom}_{\mathbf{LPos}}(A, B),\ \beta \in \mathrm{Hom}_{\mathbf{LPos}}(B, C).$$

Preservation of identities is also trivial:

$$\swarrow(\mathrm{id}_A) = \uparrow_A\{a\} = \downarrow_{A^{\mathrm{op}}}\{a\} = \mathrm{id}_{A^{\mathrm{op}}}, \quad A \in \mathsf{Ob}(\mathbf{UPos}),$$

where id_A is an identity morphism in **UPos**, $\mathrm{id}_{A^{\mathrm{op}}}$ is an identity morphism in **LPos**, and the second equality is due Lemma 7.16. Similarly:

$$\nearrow(\mathrm{id}_A) = \downarrow_A\{a\} = \uparrow_{A^{\mathrm{op}}}\{a\} = \mathrm{id}_{A^{\mathrm{op}}}, \quad A \in \mathsf{Ob}(\mathbf{LPos}).$$

Finally, we need to show that composing the two functors results in the identity functor. The composition of $\swarrow$ and $\nearrow$ is an identity on the objects because for any poset A, it holds that $(A^{\mathrm{op}})^{\mathrm{op}} = A$. And again, as the functors do not change the subset of elements of the poset but only flip its context, the composition of the functors is also an identity on the morphisms. □

Remark 9.10. In the statement of Lemma 9.9 we set that $\swarrow \fatsemi \nearrow = \mathrm{id}_{\mathbf{UPos}}$ and $\nearrow \fatsemi \swarrow = \mathrm{id}_{\mathbf{LPos}}$, hence that the composition of these functors results in the identity functor. This is a stronger condition than the one in Definition 9.7 which asks only for natural isomorphisms. This stronger condition is often referred to as an *isomorphism of categories* (Lane, 1998).

Remark 9.11. As $\swarrow$ and $\nearrow$ are a pair of functors which is an equivalence of the categories **UPos** and **LPos**, it also holds that the $\swarrow$ and $\nearrow$ functors are *full and faithful* (Lane, 1998, Section IV.4, Theorem 1). We can also see that directly from the definitions of the two functors. Their effect on hom-sets is bijective because they do not alter the monotone maps, but simply flip the orders in their domain and range, which is a trivially invertible operation.

Finally, let's formalize the notion of duality of problems.

Definition 9.12 (Dual design problem)**.** Given a design problem

$$d : A^{\mathrm{op}} \times B \to \mathtt{Bool}$$

in $\mathrm{Hom}_{\mathbf{DP}}(A, B)$, we define the dual design problem $d' \in \mathrm{Hom}_{\mathbf{DP}}(B^{\mathrm{op}}, A^{\mathrm{op}})$ as:

$$\begin{aligned} d' \colon B \times A^{\mathrm{op}} &\to \mathtt{Bool}, \\ \langle b, a \rangle &\mapsto d(b, a). \end{aligned}$$

As mentioned before, the duality of design problems is a concept due Censi (2016) and Censi et al. (2020).

Definition 9.13 (The $\multimapbothvert$ functor)**.** The $\multimapbothvert$ functor is an involute contravariant (Definition 2.25) endo-functor on **DP** that maps each poset P to its opposite P^{op} and each design problem (morphism) to its dual.

Lemma 9.14. *Take a design problem $d : A \nrightarrow B$. For all $a \in A$, it holds that the upper set $\{b \in B \mid d(a, b) = \mathtt{T}\}$ in the poset $(B, \leq_B)$ and the lower set $\{b \in B \mid d'(b, a) = \mathtt{T}\}$ in the poset $(B, \leq_B^{\mathrm{op}})$ have the same elements. Furthermore*

$$\multimapbothvert(\multimapbothvert(d)) = d,$$

hence taking duals is an involution.

Proof. Directly from the definition we see that $d(a, b) = d'(b, a)$ for all $a \in A$ and $b \in B$, hence the two sets are the same. □

9.2 Co-design problems and Lagado

Recall from Section 7.2 and (Censi et al., 2020) that a design problem in **DP** is a feasibility relation between two posets. We can have two different formulations for this problem. The primal formulation fixes a functionality and asks for the upper set of resources that can provide it. The dual formulation (Definition 9.12) fixes a resource and asks for the lower set of functionalities that can be obtained with it. And one can convert a primary problem to a dual problem via the $\multimapbothvert$ functor (Definition 9.13).

We will now study how these two different formulations (which correspond to different problems in the sense of Definitions 4.1 and 6.14) are included in the **Lagado** category. This is meant to serve as a simple example to motivate why we are interested in co-design. We will then study the deeper compositional computational properties of co-design in greater detail in Chapter 10.

Consider the type

$$\mathsf{DP} = \bigcup_{A,B} \mathrm{Hom}_{\mathbf{DP}}(A, B),$$

which gives rise to the normed type **DP**. We can also define the type of upper set functions

$$\mathsf{UPos} = \bigcup_{A,B} \mathrm{Hom}_{\mathbf{UPos}}(A, B),$$

each element of which is a function from a poset to the upper sets of a poset. By endowing it with a size, we obtain the normed type of upper set functions **UPos**. Similarly, we also have the normed type of lower set functions **LPos**.

Now, the primal problem is a morphism in **Prob** (and an object in **Lagado**):

$$\pi_f^{\mathrm{DP}} \colon \mathbf{DP} \times \mathbf{UPos} \longrightarrow \mathtt{Bool},$$
$$\langle d, u \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } u \colon F(d) \longrightarrow \mathsf{U}(R(d)) \text{ and for all } f \in F(d) \text{ it holds:} \\ & u(f) = \{r \mid r \in R(d),\ d(f, r) = \mathtt{T}\}, \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

In the above definition, the problem statement d is a co-design problem, $R(d)$ is its resource space and $F(d)$ is its functionality space. As the solution u is a morphism from $F(d)$ to $R(d)$ in **UPos**, it is a function from the elements of the functionality space of the design problem to the upper sets of its resource space. The condition in the above definition ensures that the only pair of d and u for which the problem relation holds is the pair where the function u is the one that maps a functionality to the resource upper set that can provide it.

In an analogous way there's an object in **Lagado** for the dual problem:

$$\pi_r^{\mathrm{DP}} \colon \mathbf{DP} \times \mathbf{LPos} \longrightarrow \mathtt{Bool},$$
$$\langle d, l \rangle \mapsto \begin{cases} \mathtt{T} & \text{if } l \colon R(d) \longrightarrow \mathsf{L}(F(d)) \text{ and for all } r \in R(d) \text{ it holds:} \\ & l(r) = \{f \mid f \in F(d),\ d(f, r) = \mathtt{T}\}, \\ \mathtt{F} & \text{otherwise.} \end{cases}$$

This problem has a problem statement d, again a co-design problem, but the solution l is now a function from the elements of the resource space of d to the lower sets of feasibilities that d can provide with a given resource.

The π_r^{DP} problem is the dual of π_f^{DP} because of a fundamental relationship between the two problems. One can always reduce one to the other. To see this, let's first see how we can flip the resource and feasibility spaces of a design problem. Lemma 9.14 immediately shows us that if we have a problem in π_f^{DP} which has as a solution a function that returns an upper set of resources for every feasibility f, then there is a problem in π_r^{DP} which has as a solution a function that returns a lower set of feasibilities for every resource f, such that the sets returned by the two functions are the same. To make this more precise, we provide the explicit definitions of these conversion procedures:

$$
\begin{aligned}
c^{\mathrm{DP}}_{\mathrm{dual}} \colon \mathbf{DP} &\longrightarrow \mathbf{DP}, \\
d &\longmapsto ⫯(d).
\end{aligned}
$$

Note that after performing the conversion, the result is no longer of the correct type. If originally we were looking for an upper set of R, now we get a lower set of R^{op} as $c^{\mathrm{DP}}_{\mathrm{dual}}$ maps the design problem $d : F^{\mathrm{op}} \times R \longrightarrow \mathtt{Bool}$ to the design problem $d' : R \times F^{\mathrm{op}} \longrightarrow \mathtt{Bool}$. We already saw the same problem in the proof of Lemma 9.9, where the ${\swarrow\!\circ}$ and ${\circ\!\nearrow}$ functors maintain the subsets but flip the order of the target. We will abuse the notation here, and will assume that there are two procedures, also denoted by ${\swarrow\!\circ}$ and ${\circ\!\nearrow}$ which apply this operation on our results in **Lagado**. Hence, we get the following problems and pairs of procedures in **Lagado**:

$$
\begin{tikzcd}
\pi^{\mathrm{DP}}_{f} \arrow[r, shift left, "\langle \mathbf{c}^{\mathrm{DP}}_{\mathrm{dual}},\ {\circ\!\nearrow} \rangle"] & \pi^{\mathrm{DP}}_{r} \arrow[l, shift left, "\langle \mathbf{c}^{\mathrm{DP}}_{\mathrm{dual}},\ {\swarrow\!\circ} \rangle"]
\end{tikzcd}. \tag{9.1}
$$

9.3 Functorial relationships

As we mentioned, π^{DP}_{f} and π^{DP}_{r} are functions from the space of design problems to respectively the spaces of upper sets and lower sets of posets. As the domain and ranges of these functions also happen to be the hom-sets of **DP**, **UPos**, and **LPos**, these functions also have functorial properties. We refer to the functorial avatars of π^{DP}_{f} and π^{DP}_{r} as Π^{DP}_{f} and Π^{DP}_{r}.

Definition 9.15 (The Π^{DP}_{f} functor.)**.** The Π^{DP}_{f} is a *covariant functor* (Definition 2.25) from **DP** to **UPos** which maps:

i. a poset in **DP** to the same poset in **UPos**;
ii. a morphism d in $\mathrm{Hom}_{\mathbf{DP}}(A, B)$ to the morphism u in $\mathrm{Hom}_{\mathbf{UPos}}(A, B)$, where u is the monotone function

$$
\begin{aligned}
u \colon A &\longrightarrow \mathsf{U}B, \\
a &\longmapsto \{b \in B \mid d(a, b) = \mathtt{T}\}.
\end{aligned}
$$

Remark 9.16. Recall that for any design problem $d : A \nrightarrow B$ and any $a \in A$ it holds that $\{b \in B \mid d(a, b) = \mathtt{T}\}$ is an upper set (Lemma 8.13). One can similarly show that for any $b \in B$ it holds that $\{a \in A \mid d(a, b) = \mathtt{T}\}$ is a lower set.

Lemma 9.17. *Π^{DP}_{f} is indeed a functor.*

Proof. Directly from Definition 9.15 we see that Π_f^{DP} respects the sources and targets of the morphisms. To check that it also preserves identities we need to recall what the identity design problem is. The identity design problem on the poset A is:

$$\begin{aligned} \mathrm{id}_A^{\mathrm{DP}} \colon A^{\mathrm{op}} \times A &\to \mathtt{Bool}, \\ \langle a, a' \rangle &\mapsto \begin{cases} \mathtt{T} & \text{if } a \leq_A a', \\ \mathtt{F} & \text{otherwise}, \end{cases} \end{aligned}$$

as in Definition 7.20. The functor takes this to the u function in $\mathrm{Hom}_{\mathbf{UPos}}(A, A)$:

$$\begin{aligned} u \colon A &\to \mathsf{U}A, \\ a &\mapsto \{a' \in A \mid a \leq_A a'\}, \end{aligned}$$

which is exactly $\uparrow_A \{a\}$ as in the definition of the identity in Definition 9.3.

Finally, we need to show that Π_f^{DP} respects the composition of morphisms, in other words that

$$\Pi_f^{\mathrm{DP}}(d_1 \mathbin{⨟} d_2) = \Pi_f^{\mathrm{DP}}(d_1) \ltimes \Pi_f^{\mathrm{DP}}(d_2),$$

where $d_1 : A \twoheadrightarrow B$ and $d_2 : B \twoheadrightarrow C$ are two design problems.

The left-hand side is a monotone function from A to C, hence a morphism in **UPos**. Writing it out (referring to Definition 7.17) we get:

$$\begin{aligned} \Pi_f^{\mathrm{DP}}\left(d_1 \mathbin{⨟_{\mathbf{DP}}} d_2\right) &= \left[a \mapsto \Pi_f^{\mathrm{DP}}\left([\langle a, b \rangle \mapsto \textstyle\bigvee_{b \in B} d_1(a, b) \wedge d_2(b, c)]\right)\right] \\ &= \left[a \mapsto \{c \in C \mid \textstyle\bigvee_{b \in B} d_1(a, b) \wedge d_2(b, c) = \mathtt{T}\}\right]. \end{aligned}$$

Writing out the right-hand side we obtain:

$$\begin{aligned} &\Pi_f^{\mathrm{DP}}(d_1) \ltimes \Pi_f^{\mathrm{DP}}(d_2) \\ &= [a \mapsto \{b \in B \mid d_1(a, b) = \mathtt{T}\}] \ltimes [b \mapsto \{c \in C \mid d_2(b, c) = \mathtt{T}\}] \\ &= \left[a \mapsto \bigcup_{b \in \{b \in B \mid d_1(a,b) = \mathtt{T}\}} \{c \in C \mid d_2(b, c) = \mathtt{T}\}\right] \\ &= [a \mapsto \{c \in C \mid b \in B,\ d_1(a, b) = \mathtt{T},\ d_2(b, c) = \mathtt{T}\}] \\ &= [a \mapsto \{c \in C \mid \textstyle\bigvee_{b \in B} d_1(a, b) \wedge d_2(b, c) = \mathtt{T}\}], \end{aligned}$$

and hence the two sides are equal. Therefore, Π_f^{DP} is indeed a functor. □

In an analogous way we can also define the Π_r^{DP} functor:

Definition 9.18 (The Π_r^{DP} functor.)**.** The Π_r^{DP} functor is a *contravariant functor* (Definition 2.25) from **DP** to **LPos** which maps:

i. a poset in **DP** to the same poset in **LPos**;
ii. a morphism d in $\mathrm{Hom}_{\mathbf{DP}}(A, B)$ to the morphism l in $\mathrm{Hom}_{\mathbf{LPos}}(B, A)$, where l is the monotone function

$$\begin{aligned} l\colon B &\to \mathsf{L}A, \\ b &\mapsto \{a \in A \mid d(a, b) = \mathtt{T}\}. \end{aligned}$$

Lemma 9.19. Π_r^{DP} *is a functor.*

Proof. Analogous to the proof of Lemma 9.17. □

Now, we have the following functors between our three categories **DP**, **LPos**, and **UPos**:

$$\begin{tikzcd} & & \mathbf{UPos} \arrow[dd, bend left, "\swarrow"] \\ \mathbf{DP} \arrow[loop left, "⚲"] \arrow[urr, "\Pi_f^{DP}"] \arrow[drr, "\Pi_r^{DP}"'] & & \\ & & \mathbf{LPos} \arrow[uu, bend left, "\nearrow"] \end{tikzcd} \tag{9.2}$$

Note that this diagram *does not* commute! However, this only raises the question whether we can still identify some of the compositions. To address this, we offer the following lemma.

Lemma 9.20. $⚲ \fatsemi \Pi_f^{DP} \fatsemi \swarrow = \Pi_r^{DP}$ *and* $⚲ \fatsemi \Pi_r^{DP} \fatsemi \nearrow = \Pi_f^{DP}$.

Proof. Object-wise, the Π_f^{DP} and Π_r^{DP} map the posets in **DP** to the same preoreders in **UPos** and **LPos** respectively. $⚲$, $\swarrow$ and $\nearrow$, however, map each poset in their orgin to its opposite poset in the target. In both of the equations in the body of the lemma we have two functors of the second kind, which cancel each other, hence the effects of left-hand and right-hand side of the equations are the same on the object.

Let's verify that this also holds for the morphisms. First, we will address the $⚲ \fatsemi \Pi_f^{\mathrm{DP}} \fatsemi \swarrow = \Pi_r^{\mathrm{DP}}$ equation. Take a design problem

$$d\colon A \twoheadrightarrow B \in \mathrm{Hom}_{\mathbf{DP}}(A, B).$$

Taking its dual we obtain $⚲(d) = d' \in \mathrm{Hom}_{\mathbf{DP}}(B^{\mathrm{op}}, A^{\mathrm{op}})$, with

$$\begin{aligned} d'\colon B \times A^{\mathrm{op}} &\to \mathtt{Bool}, \\ \langle b, a\rangle &\mapsto d(a, b). \end{aligned} \tag{9.3}$$

Applying the Π_f^{DP} functor we obtain $\Pi_f^{\mathrm{DP}}(d') = u$, with

$$u\colon B^{\mathrm{op}} \to \mathsf{U}(A^{\mathrm{op}}),$$

$$b \mapsto \{a \in A \mid d'(b,a) = \mathtt{T}\} = \{a \in A \mid d(a,b) = \mathtt{T}\},$$

where the equality is due to the definition of a dual design problem (Definition 9.12). Finally, applying the last functor, we obtain $\swarrow(u) = l$, with

$$\begin{aligned} l \colon B &\longrightarrow \mathsf{L}A, \\ b &\longmapsto \{a \in A \mid d(a,b) = \mathtt{T}\}. \end{aligned}$$

And directly from Definition 9.18 we see that, $\Pi_r^{\mathrm{DP}}(d) = l$, and hence the first equality holds.

Now let's take a look at $⧟ \fatsemi \Pi_r^{\mathrm{DP}} \fatsemi \nearrow\ = \Pi_f^{\mathrm{DP}}$. We take the same design problem d with the same dual $⧟(d) = d'$ (Equation (9.3)). Applying the Π_r^{DP} functor we obtain $\Pi_r^{\mathrm{DP}}(d') = l$, with

$$\begin{aligned} l \colon A^{\mathrm{op}} &\longrightarrow \mathsf{L}(B^{\mathrm{op}}), \\ a &\longmapsto \{b \in B \mid d'(b,a) = \mathtt{T}\} = \{b \in B \mid d(a,b) = \mathtt{T}\}. \end{aligned}$$

Finally, applying the last functor, we obtain $\nearrow(l) = u$, with

$$\begin{aligned} u \colon A &\longrightarrow \mathsf{U}B, \\ a &\longmapsto \{b \in B \mid d(a,b) = \mathtt{T}\}. \end{aligned}$$

And directly from Definition 9.15 we see that, $\Pi_f^{\mathrm{DP}}(d) = u$, and hence the second equality also holds. □

Remark 9.21. One can now see that Equation (9.1) and Equation (9.2) look very similar. In both cases we have a way of translating the primal problem into a dual problem and the solution of the dual problem into a solution for the original primal problem. And of course, we can also do the opposite: translating the dual problem into a primal problem and the solution of the primal problem into a solution for the original dual problem.

Lemma 9.22. *It holds that* $⧟ \fatsemi \Pi_f^{DP} = \Pi_r^{DP} \fatsemi \nearrow$ *and* $⧟ \fatsemi \Pi_r^{DP} = \Pi_f^{DP} \fatsemi \swarrow$.

Proof. Follows directly from Lemma 9.20 and Lemma 9.9. □

9.4 Monoidal properties

The **DP** category is a symmetric monoidal category (Censi et al., 2020). The monoidal product of the objects in **DP** (which are posets) is simply the product poset (Definition 2.6). The monoidal product of two design problems (morphisms in **DP**) is simply their then-composition (Definition 7.17). We will now show that **UPos** and **LPos** are also symmetric monoidal. And not only that, but also the functors that we defined in the previous section preserve the tensor products of these categories.

Lemma 9.23. **UPos** *is a monoidal category when considering the following additional structure:*

i. Tensor product $\otimes$, with the operation on objects being the poset product as defined in Definition 2.6, and operation on two morphisms $f\colon A \to \mathsf{U}B$ and $g\colon C \to \mathsf{U}D$

$$\begin{aligned} f \otimes g\colon A \times C &\to \mathsf{U}(B \times D), \\ \langle a, c\rangle &\mapsto f(a) \times g(c), \end{aligned}$$

as the Cartesian product of upper sets is also an upper set.

ii. Unit object being the identity poset I (Definition 2.4).

iii. Left unitor being the pair of morphisms

$$\begin{aligned} \lambda_A\colon I \otimes A &\to \mathsf{U}A, \\ \langle \iota, a\rangle &\mapsto \uparrow\{a\}; \end{aligned}$$

and

$$\begin{aligned} \lambda_A^{-1}\colon A &\to \mathsf{U}(I \otimes A), \\ a &\mapsto \{\iota\} \times \uparrow\{a\}. \end{aligned}$$

iv. Right unitor being the pair of morphisms

$$\begin{aligned} \rho_A\colon A \otimes I &\to \mathsf{U}A, \\ \langle a, \iota\rangle &\mapsto \uparrow\{a\}; \end{aligned}$$

and

$$\begin{aligned} \rho_A^{-1}\colon A &\to \mathsf{U}(A \otimes I), \\ a &\mapsto \uparrow\{a\} \times \{\iota\}. \end{aligned}$$

v. Associator being the pair of morphisms

$$\begin{aligned} \alpha_{AB,C}\colon \quad (A \otimes B) \otimes C &\to \mathsf{U}A \times (\mathsf{U}B \times \mathsf{U}C), \\ \langle\langle a, b\rangle, c\rangle &\mapsto \uparrow\{a\} \times (\uparrow\{b\} \times \uparrow\{c\}); \end{aligned}$$

and

$$\begin{aligned} \alpha_{A,BC}\colon \quad A \otimes (B \otimes C) &\to (\mathsf{U}A \times \mathsf{U}B) \times \mathsf{U}C, \\ \langle a, \langle b, c\rangle\rangle &\mapsto (\uparrow\{a\} \times \uparrow\{b\}) \times \uparrow\{c\}. \end{aligned}$$

Proof. In order to show that **UPos** is a monoidal category, we need to show that the triangle and pentagon diagrams commute. The triangle identity (see Equation (3.1)) is equivalent to:

$$\alpha_{AI,B} \ltimes (\mathrm{id}_A \otimes \lambda_B) = \rho_A \otimes \mathrm{id}_B \,.$$

We have the left-hand side equal to:

$$\alpha_{AI,B} \ltimes (\mathrm{id}_A \otimes \lambda_B)$$

$$
\begin{aligned}
&= [\langle\langle a,\iota\rangle,b\rangle \mapsto (\uparrow\{a\}\times(\uparrow\{\iota\}\times\uparrow\{b\}))] \ltimes ([a\mapsto\uparrow\{a\}]\otimes[(\iota,b)\mapsto\uparrow\{b\}])\\
&= [\langle\langle a,\iota\rangle,b\rangle \mapsto (\uparrow\{a\}\times(\{\iota\}\times\uparrow\{b\}))] \ltimes [\langle a,\langle\iota,b\rangle\rangle\mapsto(\uparrow\{a\}\times\uparrow\{b\})]\\
&= \left[\langle\langle a,\iota\rangle,b\rangle \mapsto \bigcup_{\langle a',\langle\iota,b'\rangle\rangle\in\uparrow\{a\}\times(\{\iota\}\times\uparrow\{b\})} (\uparrow\{a'\}\times\uparrow\{b'\})\right]\\
&= [\langle\langle a,\iota\rangle,b\rangle \mapsto (\uparrow\{a\}\times\uparrow\{b\})]\,.
\end{aligned}
$$

And the right-hand side is equal to:

$$
\begin{aligned}
\rho_A\otimes\mathrm{id}_B &= [\langle a,\iota\rangle\mapsto\uparrow\{a\}]\otimes[b\mapsto\uparrow\{b\}]\\
&= [\langle\langle a,\iota\rangle,b\rangle\mapsto(\uparrow\{a\}\times\uparrow\{b\})]\,.
\end{aligned}
$$

Therefore, the triangle equality holds.

The pentagon inequality (see Equation (3.2)) is equivalent to:

$$
\alpha_{(AB)C,D}\ltimes\alpha_{AB,(CD)} = (\alpha_{AB,C}\otimes\mathrm{id}_D)\ltimes\alpha_{A(BC),D}\ltimes(\mathrm{id}_A\otimes\alpha_{BC,D})\,.
$$

The left hand-side evaluates to:

$$
\begin{aligned}
&\alpha_{(AB)C,D}\ltimes\alpha_{AB,(CD)}\\
&= [\langle\langle\langle a,b\rangle,c\rangle,d\rangle\mapsto((\uparrow\{a\}\times\uparrow\{b\})\times(\uparrow\{c\}\times\uparrow\{d\}))]\ltimes\\
&\quad [\langle\langle a,b\rangle,\langle c,d\rangle\rangle\mapsto(\uparrow\{a\}\times(\uparrow\{b\}\times(\uparrow\{c\}\times\uparrow\{d\})))]\\
&= \left[\langle\langle\langle a,b\rangle,c\rangle,d\rangle\mapsto \bigcup_{\substack{\langle\langle a',b'\rangle,\langle c',d'\rangle\rangle\in\\(\uparrow\{a\}\times\uparrow\{b\})\times(\uparrow\{c\}\times\uparrow\{d\})}} (\uparrow\{a'\}\times(\uparrow\{b'\}\times(\uparrow\{c'\}\times\uparrow\{d'\})))\right]\\
&= [\langle\langle\langle a,b\rangle,c\rangle,d\rangle\mapsto(\uparrow\{a\}\times(\uparrow\{b\}\times(\uparrow\{c\}\times\uparrow\{d\})))]\,.
\end{aligned}
$$

The two monoidal products on the right-hand side evaluate to:

$$
\begin{aligned}
\alpha_{AB,C}\otimes\mathrm{id}_D &= [\langle\langle\langle a,b\rangle,c\rangle,d\rangle\mapsto((\uparrow\{a\}\times(\uparrow\{b\}\times\uparrow\{c\}))\times\uparrow\{d\})]\,,\\
\mathrm{id}_A\otimes\alpha_{BC,D} &= [\langle a,\langle\langle b,c\rangle,d\rangle\rangle\mapsto(\uparrow\{a\}\times(\uparrow\{b\}\times(\uparrow\{c\}\times\uparrow\{d\})))]\,.
\end{aligned}
$$

The first two terms of the right-hand side evaluate to:

$$
\begin{aligned}
&(\alpha_{AB,C}\otimes\mathrm{id}_D)\ltimes\alpha_{A(BC),D}\\
&= \left[(((a,b),c),d)\mapsto \bigcup_{\substack{((a',(b',c')),d')\in\\(\uparrow\{a\}\times(\uparrow\{b\}\times\uparrow\{c\}))\times\uparrow\{d\}}} (\uparrow\{a'\}\times((\uparrow\{b'\}\times\uparrow\{c'\})\times\uparrow\{d'\}))\right]\\
&= [(((a,b),c),d)\mapsto(\uparrow\{a\}\times((\uparrow\{b\}\times\uparrow\{c\})\times\uparrow\{d\}))]\,.
\end{aligned}
$$

Hence, the right-hand side of the pentagon equality is:

$$\begin{aligned}
&(\alpha_{AB,C} \otimes \mathrm{id}_D) \ltimes \alpha_{A(BC),D} \ltimes (\mathrm{id}_A \otimes \alpha_{BC,D}) \\
&= [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto (\uparrow\{a\} \times ((\uparrow\{b\} \times \uparrow\{c\}) \times \uparrow\{d\}))] \ltimes \\
&\quad [\langle a,\langle\langle b,c\rangle ,d\rangle\rangle \mapsto (\uparrow\{a\} \times (\uparrow\{b\} \times (\uparrow\{c\} \times \uparrow\{d\})))] \\
&= [\langle\langle\langle a,b\rangle ,c\rangle ,d\rangle \mapsto (\uparrow\{a\} \times (\uparrow\{b\} \times (\uparrow\{c\} \times \uparrow\{d\})))],
\end{aligned}$$

hence, the pentagon equality also holds and **UPos** is a monoidal category. □

Lemma 9.24. **UPos** *is a symmetric monoidal category when considering the braiding isomorphism*

$$\begin{aligned}
\sigma_{A,B} \colon A \otimes B &\to B \otimes A, \\
\langle a,b\rangle &\mapsto \uparrow\{b\} \times \uparrow\{a\},
\end{aligned}$$

defined for all $A,B \in \mathsf{Ob}(\mathbf{UPos})$*, As σ is involute, it is also the map of the inverse morphism.*

Proof. To prove that **UPos** is a symmetric monoidal category we need to show that the braiding σ makes the unit coherence, associativity coherence, and the inverse law diagrams commute. The unit coherence diagram is:

$$\begin{tikzcd}
A \otimes I \arrow[rr, "\sigma_{A,I}"] \arrow[dr, "\rho_A"] & & I \otimes A \arrow[dl, "\lambda_A"] \\
& A &
\end{tikzcd}$$

The coherence diagram commutes because $\sigma_{A,I}$ just flips the operands in the Cartesian product, while ρ_A and λ just extract the non-identity component out if it. The associativity coherence diagram is:

$$\begin{tikzcd}
(A \otimes B) \otimes C \arrow[r, "\sigma_{A,B}\otimes \mathrm{id}_C"] \arrow[d, "\alpha_{AB,C}"] & (B \otimes A) \otimes C \arrow[d, "\alpha_{BA,C}"] \\
A \otimes (B \otimes C) \arrow[d, "\sigma_{A,B\otimes C}"] & B \otimes (A \otimes C) \arrow[d, "\mathrm{id}_B \otimes \sigma_{A,C}"] \\
(B \otimes C) \otimes A \arrow[r, "\alpha_{BC,A}"] & B \otimes (C \otimes A)
\end{tikzcd}$$

The associativity coherence diagram again commutes because σ just flips the positions of the elements in the Cartesian product and α simply alters the groupings.

Finally, the inverse law diagram is:

$$\begin{tikzcd}
A \otimes B \arrow[rr, equal, "\mathrm{id}_{A\otimes B}"] \arrow[dr, "\sigma_{A,B}"] & & A \otimes B \\
& B \otimes A \arrow[ur, "\sigma_{B,A}"] &
\end{tikzcd}$$

The inverse law diagram commutes because σ is involute and hence applying it twice result in the identity morphism. □

Lemma 9.25. **LPos** *is a symmetric monoidal category when considering the following additional structure:*

i. *Tensor product $\otimes$, with operation on objects being the poset product as defined in Definition 2.6, and operation on two morphisms $f\colon A \longrightarrow \mathsf{L}B$ and $g\colon C \longrightarrow \mathsf{L}D$*

$$\begin{aligned} f \otimes g\colon A \times C &\longrightarrow \mathsf{L}(B \times D), \\ \langle a, c\rangle &\mapsto f(a) \times g(c), \end{aligned}$$

as the Cartesian product of lower sets is also a lower set.

ii. *Unit object being the identity poset I (Definition 2.4).*

iii. *Left unitor being the pair of morphisms*

$$\begin{aligned} \lambda_A\colon I \otimes A &\longrightarrow \mathsf{L}A, \\ \langle \iota, a\rangle &\mapsto {\downarrow}\{a\}; \end{aligned}$$

and

$$\begin{aligned} \lambda_A^{-1}\colon \quad A &\longrightarrow \mathsf{L}(I \otimes A), \\ \langle a\rangle &\mapsto \{\iota\} \times {\downarrow}\{a\}. \end{aligned}$$

iv. *Right unitor being the pair of morphisms*

$$\begin{aligned} \rho_A\colon A \otimes I &\longrightarrow \mathsf{L}A, \\ \langle a, \iota\rangle &\mapsto {\downarrow}\{a\}; \end{aligned}$$

and

$$\begin{aligned} \rho_A^{-1}\colon \quad A &\longrightarrow \mathsf{L}(A \otimes I), \\ \langle a\rangle &\mapsto {\downarrow}\{a\} \times \{\iota\}. \end{aligned}$$

v. *Associator being the pair of morphisms*

$$\begin{aligned} \alpha_{AB,C}\colon (A \otimes B) \otimes C &\longrightarrow \mathsf{L}A \times (\mathsf{L}B \times \mathsf{L}C), \\ \langle\langle a, b\rangle, c\rangle &\mapsto {\downarrow}\{a\} \times ({\downarrow}\{b\} \times {\downarrow}\{c\}); \end{aligned}$$

and

$$\begin{aligned} \alpha_{A,BC}\colon A \otimes (B \otimes C) &\longrightarrow (\mathsf{L}A \times \mathsf{L}B) \times \mathsf{L}C, \\ \langle a, \langle b, c\rangle\rangle &\mapsto ({\downarrow}\{a\} \times {\downarrow}\{b\}) \times {\downarrow}\{c\}. \end{aligned}$$

vi. *Braiding being the involute isomorphism*

$$\begin{aligned} \sigma_{A,B}\colon A \otimes B &\longrightarrow B \otimes A, \\ \langle a, b\rangle &\mapsto {\downarrow}\{b\} \times {\downarrow}\{a\}. \end{aligned}$$

Proof. Analogous to the proof of Lemmas 9.23 and 9.24. □

In this section we aim to show that the functors in Equation (9.2) respect this monoidal structure.

Lemma 9.26. *The ⧟ functor is a strong monoidal functor (Definition 2.31).*

Proof. First, notice that ⧟ is an endofunctor with origin and target **DP**. Take two arbitrary posets A, B. Then, we have

$$\begin{aligned} \text{⧟}(A) \otimes_{\mathbf{DP}} \text{⧟}(B) &= A^{\mathrm{op}} \otimes_{\mathbf{DP}} B^{\mathrm{op}} \\ &= A^{\mathrm{op}} \times B^{\mathrm{op}} \\ &= (A \times B)^{\mathrm{op}} \\ &= (A \otimes_{\mathbf{DP}} B)^{\mathrm{op}} \\ &= \text{⧟}(A \otimes_{\mathbf{DP}} B)\,, \end{aligned}$$

where $\times$ is the poset product (Definition 2.6). The associativity diagram holds because it is the same as the associativity diagram for **DP**, and as **DP** is monoidal (Censi et al., 2020), Equation (2.1) must also hold. The same argument holds for the left and right unitality diagrams (Equations (2.2) and (2.3)). □

Lemma 9.27. *The $\swarrow$ and $\nearrow$ functors are strong monoidal functors.*

Proof. We will prove only that $\swarrow$ is strong monoidal, the case for $\nearrow$ is analogous. First, let's define the natural isomorphism μ for $\swarrow$. Given some $A, B \in \mathsf{Ob}(\mathbf{UPos})$, from Definition 2.31 we know it should identify $\swarrow(A) \otimes_{\mathbf{LPos}} \swarrow(B)$ with $\swarrow(A \otimes_{\mathbf{UPos}} B)$, where $\otimes_{\mathbf{LPos}}$ and $\otimes_{\mathbf{UPos}}$ are the monoidal products defined in Lemmas 9.23 and 9.25. This identification is simply equality as:

$$\begin{aligned} \swarrow(A) \otimes_{\mathbf{LPos}} \swarrow(B) &= A^{\mathrm{op}} \otimes_{\mathbf{LPos}} B^{\mathrm{op}} \\ &= A^{\mathrm{op}} \times B^{\mathrm{op}} \\ &= (A \times B)^{\mathrm{op}} \\ &= (A \otimes_{\mathbf{UPos}} B)^{\mathrm{op}} \\ &= \swarrow(A \otimes_{\mathbf{UPos}} B). \end{aligned}$$

The isomorphism ϵ must map the identity poset I in **UPos** to the identity poset I in **LPos** and back.

The associativity diagram (Equation (2.1)) for the $\swarrow$ functor is:

$$\begin{tikzcd}
\big(\swarrow(A) \otimes_{\mathbf{LPos}} \swarrow(B)\big) \otimes_{\mathbf{LPos}} \swarrow(C) \arrow[r, "\alpha^{\mathbf{LPos}}_{\swarrow(A)\swarrow(B),\swarrow(C)}"] \arrow[d, "\mu_{A,B} \otimes_{\mathbf{LPos}} \mathrm{id}^{\mathbf{LPos}}_{\swarrow(C)}"] & \swarrow(A) \otimes_{\mathbf{LPos}} \big(\swarrow(B) \otimes_{\mathbf{LPos}} \swarrow(C)\big) \arrow[d, "\mathrm{id}^{\mathbf{LPos}}_{\swarrow(A)} \otimes_{\mathbf{LPos}} \mu_{B,C}"] \\
\swarrow(A \otimes_{\mathbf{UPos}} B) \otimes_{\mathbf{LPos}} \swarrow(C) \arrow[d, "\mu_{(A \otimes_{\mathbf{UPos}} B),C}"] & A \otimes_{\mathbf{LPos}} \swarrow(B \otimes_{\mathbf{UPos}} \swarrow(C)) \arrow[d, "\mu_{(A \otimes_{\mathbf{UPos}} B),C}"] \\
\swarrow((A \otimes_{\mathbf{UPos}} B) \otimes_{\mathbf{UPos}} C) \arrow[r, "\swarrow(\alpha^{\mathbf{UPos}}_{AB,C})"] & \swarrow(A \otimes_{\mathbf{UPos}} (B \otimes_{\mathbf{UPos}} C)),
\end{tikzcd}$$

The vertical arrows are isomorphisms due to μ being defined with an equality above. The horizontal arrows simply change the groupings (see Lemmas 9.23 and 9.25). Therefore, the diagram commutes.

The unitality diagrams for the $\swarrow$ functor are:

$$\begin{tikzcd}
I \otimes_{\mathbf{LPos}} \swarrow(A) \arrow[r, "\epsilon \otimes_{\mathbf{LPos}} \mathrm{id}^{\mathbf{LPos}}_{\swarrow(A)}"] \arrow[d, "\lambda^{\mathbf{LPos}}_{\swarrow(A)}"] & \swarrow(I) \otimes_{\mathbf{LPos}} \swarrow(A) \arrow[d, "\mu_{I,A}"] \\
\swarrow(A) & \swarrow(I \otimes_{\mathbf{UPos}} A) \arrow[l, "\swarrow(\lambda^{\mathbf{UPos}}_{A})"]
\end{tikzcd}$$

and

$$\begin{tikzcd}
\swarrow(A) \otimes_{\mathbf{LPos}} I \arrow[r, "\mathrm{id}^{\mathbf{LPos}}_{\swarrow(A)} \otimes_{\mathbf{LPos}} \epsilon"] \arrow[d, "\rho^{\mathbf{LPos}}_{\swarrow(A)}"] & \swarrow(A) \otimes_{\mathbf{LPos}} \swarrow(I) \arrow[d, "\mu_{A,I}"] \\
\swarrow(A) & \swarrow(A \otimes_{\mathbf{UPos}} I), \arrow[l, "\swarrow(\rho^{\mathbf{UPos}}_{A})"]
\end{tikzcd}$$

which also obviously commute. □

Lemma 9.28. *The Π^{DP}_f functor is a strong monoidal functor.*

Proof. Let's take a look at the natural isomorphism μ. From the definition of Π^{DP}_f (Definition 9.15) we know that given two posets A, B:

$$\begin{aligned}
\Pi^{\mathrm{DP}}_f(A) \otimes_{\mathbf{UPos}} \Pi^{\mathrm{DP}}_f(B) &= A \otimes_{\mathbf{UPos}} B \\
&= A \times B \\
&= A \otimes_{\mathbf{DP}} B \\
&= \Pi^{\mathrm{DP}}_f(A \otimes_{\mathbf{DP}} B).
\end{aligned}$$

Furthermore, Π^{DP}_f maps the identity poset to itself. Hence, showing associativity and unitality is equivalent to showing that these properties hold on a category with poset objects, something we know holds, as **DP** is monoidal. □

Lemma 9.29. *The Π_r^{DP} functor is a strong monoidal functor.*

Proof. From Lemma 9.20 we know that $\Pi_r^{\mathrm{DP}} = \updownarrow \fatsemi \Pi_f^{\mathrm{DP}} \fatsemi \swarrow$. We showed that the three functors on the right side are strong monoidal and as the composition of strong monoidal functors is also strong monoidal (morphism composition in the **MonCat** category (Lane, 1998, Ch. VII)), Π_r^{DP} is too a strong monoidal functor. ◻

9.5 Locally lattical structure

In this section we study some more properties of the structure of design problems. The main observation is that the one can define an order on the collection of design problems with functionality A and resource B. Intuitively, one can imagine that there are feasibility relations which provide better trade-offs than some other feasibility relations. More formally the hom-sets of **DP** have lattice structure: one can take joins and meets of them, which are also called respectively the *intersection* and *sum* design problems. This observation was first made by Censi et al. (2020). We will extend the results there by showing that this structure also exist in **UPos** and **LPos** and is respected by the functors in Equation (9.2).

Lemma 9.30. *Given any two posets $A, B \in \mathsf{Ob}(\mathbf{DP})$, $\mathrm{Hom}_{\mathbf{DP}}(A,B)$ has a bounded lattice structure (Definition 7.11) with:*

i. *Order: for any $d_1, d_2 \in \mathrm{Hom}_{\mathbf{DP}}(A,B)$ we have*

$$d_1 \preceq_{\mathbf{DP}(A,B)} d_2$$

if and only if

$$d_2(a,b) = \mathtt{T} \implies d_1(a,b) = \mathtt{T}, \quad \forall a \in A,\ \forall b \in B;$$

ii. *Joins: intersection design problems (Definition 7.26);*

iii. *Meets sum design problems (Definition 7.24);*

iv. *Top: the design problem which is always unfeasible:*

$$\begin{aligned} \top_{\mathbf{DP}(A,B)} \colon A^{\mathrm{op}} \times B &\to \mathtt{Bool}, \\ \langle a, b \rangle &\mapsto \mathtt{F}; \end{aligned}$$

v. *Bottom: the design problem which is always feasible:*

$$\begin{aligned} \bot_{\mathbf{DP}(A,B)} \colon A^{\mathrm{op}} \times B &\to \mathtt{Bool}, \\ \langle a, b \rangle &\mapsto \mathtt{T}. \end{aligned}$$

Proof. See (Censi et al., 2020). ◻

Lemma 9.31. *Given any two posets $A, B \in \mathsf{Ob}(\mathbf{UPos})$, $\mathrm{Hom}_{\mathbf{UPos}}(A, B)$ has a bounded lattice structure with:*

i. Order: for any two monotone maps $u_1, u_2 \colon A \to \mathsf{U}B$ we have

$$u_1 \leq_{\mathbf{UPos}(A,B)} u_2$$

if and only if

$$u_1(a) \supseteq u_2(a), \quad \forall a \in A;$$

ii. Joins: for any two monotone maps $u_1, u_2 \colon A \to \mathsf{U}B$, we have

$$(u_1 \sqcup u_2)(a) = u_1(a) \cap u_2(a), \quad \forall a \in A;$$

iii. Meets: for any two monotone maps $u_1, u_2 \colon A \to \mathsf{U}B$, we have

$$(u_1 \sqcap u_2)(a) = u_1(a) \cup u_2(a), \quad \forall a \in A;$$

iv. Top: the map $\top_{\mathbf{UPos}(A,B)} : a \mapsto \emptyset$;

v. Bottom: the map $\bot_{\mathbf{UPos}(A,B)} : a \mapsto B$.

Lemma 9.32. *Given any two posets $A, B \in \mathsf{Ob}(\mathbf{LPos})$, $\mathrm{Hom}_{\mathbf{LPos}}(A, B)$ has a bounded lattice structure with:*

i. Order: for any two monotone maps $l_1, l_2 \colon A \to \mathsf{L}B$ we have

$$l_1 \leq_{\mathbf{LPos}(A,B)} l_2$$

if and only if

$$l_1(a) \supseteq l_2(a), \quad \forall a \in A;$$

ii. Joins: for any two monotone maps $l_1, l_2 \colon A \to \mathsf{L}B$, we have

$$(l_1 \sqcup l_2)(a) = l_1(a) \cap l_2(a), \quad \forall a \in A;$$

iii. Meets: for any two monotone maps $l_1, l_2 \colon A \to \mathsf{L}B$, we have

$$(l_1 \sqcap l_2)(a) = l_1(a) \cup l_2(a), \quad \forall a \in A;$$

iv. Top: the map $\top_{\mathbf{LPos}(A,B)} : a \mapsto \emptyset$;

v. Bottom: the map $\bot_{\mathbf{LPos}(A,B)} : a \mapsto B$.

Lemma 9.33. *The ⫯ functor preserves the bounded lattice structure of the hom sets of* $\mathbf{DP}$*. In other words, for any $A, B \in \mathsf{Ob}(\mathbf{DP})$ and any $d_1, d_2 \in \mathrm{Hom}_{\mathbf{DP}}(A, B)$:*

i. if $d_1 \leq_{\mathbf{DP}(A,B)} d_2$, then $\text{⫯}(d_1) \leq_{\mathbf{DP}(B^{\mathrm{op}},A^{\mathrm{op}})} \text{⫯}(d_2)$;

ii. $\text{⫯}(d_1 \wedge d_2) = \text{⫯}(d_1) \sqcup \text{⫯}(d_2)$;

iii. $\text{⫯}(d_1 \vee d_2) = \text{⫯}(d_1) \sqcap \text{⫯}(d_2)$;

iv. $\text{⫯}(\top_{\mathbf{DP}(A,B)}) = \top_{\mathbf{DP}(B^{\mathrm{op}},A^{\mathrm{op}})}$;

v. $\text{⫯}(\bot_{\mathbf{DP}(A,B)}) = \bot_{\mathbf{DP}(B^{\mathrm{op}},A^{\mathrm{op}})}$.

Lemma 9.34. *The $\swarrow^{\circ}$ functor preserves the bounded lattice structure of the hom sets of* **UPos** *when it maps them to* **LPos**. *In other words, for any $A, B \in \mathsf{Ob}(\mathbf{UPos})$ and any $u_1, u_2 \in \mathrm{Hom}_{\mathbf{UPos}}(A, B)$:*

i. *if* $u_1 \leq_{\mathbf{UPos}(A,B)} u_2$, *then* $\swarrow^{\circ}(u_1) \leq_{\mathbf{LPos}(A^{\mathrm{op}},B^{\mathrm{op}})} \swarrow^{\circ}(u_2)$;
ii. $\swarrow^{\circ}(u_1 \sqcup u_2) = \swarrow^{\circ}(u_1) \sqcup \swarrow^{\circ}(u_2)$;
iii. $\swarrow^{\circ}(u_1 \sqcap u_2) = \swarrow^{\circ}(u_1) \sqcap \swarrow^{\circ}(u_2)$;
iv. $\swarrow^{\circ}(\top_{\mathbf{UPos}(A,B)}) = \top_{\mathbf{LPos}(A^{\mathrm{op}},B^{\mathrm{op}})}$;
v. $\swarrow^{\circ}(\bot_{\mathbf{UPos}(A,B)}) = \bot_{\mathbf{LPos}(A^{\mathrm{op}},B^{\mathrm{op}})}$.

Lemma 9.35. *The ${}_{\circ}\!\nearrow$ functor preserves the bounded lattice structure of the hom sets of* **LPos** *when it maps them to* **UPos**. *In other words, for any $A, B \in \mathsf{Ob}(\mathbf{LPos})$ and any $u_1, u_2 \in \mathrm{Hom}_{\mathbf{LPos}}(A, B)$:*

i. *if* $u_1 \leq_{\mathbf{UPos}(A,B)} u_2$, *then* ${}_{\circ}\!\nearrow(u_1) \leq_{\mathbf{LPos}(A^{\mathrm{op}},B^{\mathrm{op}})} {}_{\circ}\!\nearrow(u_2)$;
ii. ${}_{\circ}\!\nearrow(u_1 \sqcup u_2) = {}_{\circ}\!\nearrow(u_1) \sqcup {}_{\circ}\!\nearrow(u_2)$;
iii. ${}_{\circ}\!\nearrow(u_1 \sqcap u_2) = {}_{\circ}\!\nearrow(u_1) \sqcap {}_{\circ}\!\nearrow(u_2)$;
iv. ${}_{\circ}\!\nearrow(\top_{\mathbf{LPos}(A,B)}) = \top_{\mathbf{UPos}(A^{\mathrm{op}},B^{\mathrm{op}})}$;
v. ${}_{\circ}\!\nearrow(\bot_{\mathbf{LPos}(A,B)}) = \bot_{\mathbf{UPos}(A^{\mathrm{op}},B^{\mathrm{op}})}$.

Lemma 9.36. *The Π_f^{DP} functor preserves the bounded lattice structure of the design problems (hom sets) of* **DP** *when it maps them to* **UPos**. *In other words, for any $A, B \in \mathsf{Ob}(\mathbf{DP})$ and any $d_1, d_2 \in \mathrm{Hom}_{\mathbf{DP}}(A, B)$:*

i. *if* $d_1 \leq_{\mathbf{DP}(A,B)} d_2$, *then* $\Pi_f^{DP}(d_1) \leq_{\mathbf{UPos}(A,B)} \Pi_f^{DP}(d_2)$;
ii. $\Pi_f^{DP}(d_1 \wedge d_2) = \Pi_f^{DP}(d_1) \sqcup \Pi_f^{DP}(d_2)$;
iii. $\Pi_f^{DP}(d_1 \vee d_2) = \Pi_f^{DP}(d_1) \sqcap \Pi_f^{DP}(d_2)$;
iv. $\Pi_f^{DP}(\bot_{\mathbf{DP}(A,B)}) = \top_{\mathbf{UPos}(A,B)}$;
v. $\Pi_f^{DP}(\top_{\mathbf{DP}(A,B)}) = \bot_{\mathbf{UPos}(A,B)}$.

Lemma 9.37. *The Π_r^{DP} functor preserves the bounded lattice structure of the design problems (hom sets) of* **DP** *when it maps them to* **LPos**. *In other words, for any $A, B \in \mathsf{Ob}(\mathbf{DP})$ and any $d_1, d_2 \in \mathrm{Hom}_{\mathbf{DP}}(A, B)$:*

i. *if* $d_1 \leq_{\mathbf{DP}(A,B)} d_2$, *then* $\Pi_r^{DP}(d_1) \leq_{\mathbf{LPos}(B,A)} \Pi_r^{DP}(d_2)$;
ii. $\Pi_r^{DP}(d_1 \wedge d_2) = \Pi_r^{DP}(d_1) \sqcup \Pi_r^{DP}(d_2)$;
iii. $\Pi_r^{DP}(d_1 \vee d_2) = \Pi_r^{DP}(d_1) \sqcap \Pi_r^{DP}(d_2)$;
iv. $\Pi_r^{DP}(\top_{\mathbf{DP}(A,B)}) = \top_{\mathbf{LPos}(B,A)}$;
v. $\Pi_r^{DP}(\bot_{\mathbf{DP}(A,B)}) = \bot_{\mathbf{LPos}(B,A)}$.

9.6 Trace and feedback

The **DP** category is also a traced monoidal category (Definition 9.38) and the proof of that can be found in (Censi et al., 2020). This means that there's the notion of "feedback" of design problems in **DP**. In particular, if we have a design problem $d \in \mathrm{Hom}_{\mathbf{DP}}(A \times X, B \times X)$, i.e. a design problem with feasibilities in $A \times X$ and resources in $B \times X$, we can use the component of the feasibility in X as a resource, and thus closing the loop. This, of course, requires that the provided component of the feasibility precedes the required resource (Definition 9.39 introduces this formally). By "closing the loop" we can obtain a new problem in $\mathrm{Hom}_{\mathbf{DP}}(A, B)$ as the X component of the feasibility and resource spaces are abstracted out.

Definition 9.38 (Traced monoidal category)**.** Take the symmetric monoidal category $\langle \mathbf{C}, \otimes, I, \sigma \rangle$ with $\otimes$ being the monoidal product, I the unit object, σ the braiding (or symmetry) isomorphism. Then, $\mathbf{C}$ is *traced* if for any three objects $A, B, X \in \mathsf{Ob}(\mathbf{C})$ we have a function

$$\mathrm{Tr}^X_{A,B} \colon \mathrm{Hom}_{\mathbf{C}}(A \otimes X, B \otimes X) \to \mathrm{Hom}_{\mathbf{C}}(A, B),$$

satisfying the following axioms:

i. *Vanishing*:

$$\mathrm{Tr}^I_{A,B}(f) = f,$$

for all $f \in \mathrm{Hom}_{\mathbf{C}}(A, B)$, and

$$\mathrm{Tr}^{X\otimes Y}_{A,B}(f) = \mathrm{Tr}^X_{A,B}\left(\mathrm{Tr}^Y_{A\otimes X, B\otimes X}(f)\right),$$

for all $f \in \mathrm{Hom}_{\mathbf{C}}(A \otimes X \otimes Y, B \otimes X \otimes Y)$;

ii. *Superposing*

$$\mathrm{Tr}^X_{C\otimes A, C\otimes B}(\mathrm{id}_C \otimes f) = \mathrm{id}_C \otimes \mathrm{Tr}^X_{A,B}(f),$$

for all $f \in \mathrm{Hom}_{\mathbf{C}}(A \otimes X, B \otimes X)$;

iii. *Yanking*:

$$\mathrm{Tr}^X_{X,X}(\sigma_{X,X}) = \mathrm{id}_X .$$

Definition 9.39 (Trace of design problems)**.** Given a design problem

$$d \colon A \times X \twoheadrightarrow B \times X,$$

its *trace* $\mathrm{Tr}_{\mathbf{DP}}(d) \colon A \twoheadrightarrow B$ is defined as:

$$\begin{aligned} \mathrm{Tr}_{\mathbf{DP}}(d) \colon A^{\mathrm{op}} \times B &\to \mathtt{Bool}, \\ \langle a, b \rangle &\mapsto \bigvee_{x \in X} d(\langle a, x \rangle, \langle b, x \rangle). \end{aligned}$$

Lemma 9.40. *The $\langle \mathbf{DP}, \otimes, I, \sigma \rangle$ category is a traced monoidal category with trace defined as in Definition 9.39.*

Proof. See (Censi et al., 2020). □

As we can take the trace of problems in **DP** it is only natural to ask what happens with these traced problems when we "feed" them to the ⧟, ⤡, ⤢, Π_f^{DP}, and Π_r^{DP} functors. Hence, just as in the previous sections, we will show that these functors preserve the trace.

Lemma 9.41. *The* **UPos** *category is a traced monoidal category with trace defined for any posets $A, B, X \in \mathsf{Ob}(\mathbf{UPos})$ as*

$$\begin{aligned}\mathrm{Tr}^X_{A,B}\colon\ &(A \otimes_{\mathbf{UPos}} X \to \mathsf{U}(B \otimes_{\mathbf{UPos}} X)) \to (A \to \mathsf{U}B)\,,\\ &u \mapsto [a \mapsto \{b \in B \mid (\textstyle\bigvee_{x\in X}((b,x) \in u(a,x))) = \mathtt{T}\}]\,.\end{aligned}$$

We will also denote the trace for morphisms in **UPos** *by* $\mathrm{Tr}_{\mathbf{UPos}}$.

Proof. First, note that **UPos** is a symmetric monoidal category due Lemma 9.24. Now let's verify that the output of $\mathrm{Tr}^X_{A,B}$ is indeed a monotone function mapping to upper sets in B. Monotonicity requires that for any

$$u : A \otimes_{\mathbf{UPos}} X \to \mathsf{U}(B \otimes_{\mathbf{UPos}} X),$$

and any $a \leq_A a' \in A$ it holds that:

$$\mathrm{Tr}_{\mathbf{UPos}}(u)(a) \leq_{\mathsf{U}B} \mathrm{Tr}_{\mathbf{UPos}}(u)(a') \qquad \Longleftrightarrow \qquad \mathrm{Tr}_{\mathbf{UPos}}(u)(a) \supseteq \mathrm{Tr}_{\mathbf{UPos}}(u)(a').$$

Due to the monotonicity of u, for any $b \in B$ and $x \in X$ we know that

$$(b,x) \in u(a',x) \qquad \Longrightarrow \qquad (b,x) \in u(a,x).$$

Therefore, if $b \in \mathrm{Tr}_{\mathbf{UPos}}(u)(a')$, then $b \in \mathrm{Tr}_{\mathbf{UPos}}(u)(a)$, which is exactly the same as the right condition in Section 9.6, hence the output of $\mathrm{Tr}^X_{A,B}$ is indeed a monotone function. Again, due to the monotonicity of u we have that for any $b \leq_B b' \in B$, $a \in A$, and $x \in X$:

$$(b,x) \in u(a,x) \qquad \Longrightarrow \qquad (b',x) \in u(a,x).$$

Therefore, $\mathrm{Tr}_{\mathbf{UPos}}(u)(a)$ is indeed an upper set for all $a \in A$.

Next, we need to prove the axioms of traced monoidal categories (Definition 9.38). Given an $f\colon A \to \mathsf{U}B$, the first vanishing axiom is equivalent to

$$f = \left[a \mapsto \left\{b \in B \mid (b,\iota) \in (f \otimes \mathrm{id}_I)(a,\iota)\right\}\right],$$

which is trivially true.

Given an $f : A \otimes X \otimes Y \to B \otimes X \otimes Y$, the left-hand side of the second vanishing axiom is:

$$\mathrm{Tr}^{X\otimes Y}_{A,B}(f) = \left[(a) \mapsto \left\{b \in B \mid \left(\textstyle\bigvee_{(x,y)\in X\otimes Y}((b,x,y) \in f(a,x,y))\right) = \mathtt{T}\right\}\right].$$

We also have:

$$\mathrm{Tr}^{Y}_{A\otimes X,B\otimes X}(f)=\Big[(a,x)\mapsto\Big\{(b,x)\in B\otimes X\,|\,\Big(\bigvee_{y\in Y}\big((b,x,y)\in f(a,x,y)\big)\Big)=\mathtt{T}\Big\}\Big].$$

Hence, the right-hand side of the second vanishing axiom is:

$$\begin{aligned}&\mathrm{Tr}^{X}_{A,B}(\mathrm{Tr}^{Y}_{A\otimes X,B\otimes X}(f))\\ &=\Big[a\mapsto\Big\{b\in B\,|\,\Big(\bigvee_{x\in X}\Big((b,x)\in\mathrm{Tr}^{Y}_{A\otimes X,B\otimes X}(f)(a,x)\Big)\Big)=\mathtt{T}\Big\}\Big]\\ &=\Big[a\mapsto\Big\{b\in B\,|\,\bigvee_{x\in X}(b,x)\in\Big\{(b',x')\in B\otimes X\,|\,\Big(\bigvee_{y'\in Y}\big((b',x',y')\in f(a,x',y')\big)\Big)=\mathtt{T}\Big\}\Big\}\Big]\\ &=\Big[a\mapsto\Big\{b\in B\,|\,\bigvee_{x\in X}\Big(\bigvee_{y\in Y}\big((b,x,y)\in f(a,x,y)\big)\Big)=\mathtt{T}\Big\}\Big],\end{aligned}$$

which is equivalent to the left-hand side, hence the second vanishing axiom also holds.

Given an $f\colon A\otimes X\to B\otimes X$, the right-hand side of the superposing axiom equals:

$$\begin{aligned}&\mathrm{Tr}^{X}_{C\otimes A,C\otimes B}(\mathrm{id}^{\mathbf{UPos}}_{C}\otimes f)\\ &=\Big[(c,a)\mapsto\Big\{(c,b)\in C\otimes B\,|\,\Big(\bigvee_{x\in X}\Big((c,b,x)\in\Big(\mathrm{id}^{\mathbf{UPos}}_{C}\otimes f\Big)(c,a,x)\Big)\Big)=\mathtt{T}\Big\}\Big]\\ &=\Big[(c,a)\mapsto\Big\{(c,b)\in C\otimes B\,|\,\Big(\bigvee_{x\in X}\Big(c\in\mathrm{id}^{\mathbf{UPos}}_{C}(c)\ \wedge\ (b,x)\in f(a,x)\Big)\Big)=\mathtt{T}\Big\}\Big]\\ &=\big[(c,a)\mapsto\big\{(c,b)\in C\otimes B\,|\,\big(\bigvee_{x\in X}\big(c\in\uparrow\{c\}\ \wedge\ (b,x)\in f(a,x)\big)\big)=\mathtt{T}\big\}\big]\\ &=\big[(c,a)\mapsto\big\{(c,b)\in\uparrow\{c\}\otimes B\,|\,\big(\bigvee_{x\in X}\big((b,x)\in f(a,x)\big)\big)=\mathtt{T}\big\}\big]\\ &=\big[(c,a)\mapsto\big(\uparrow\{c\}\times\big\{b\in B\,|\,\big(\bigvee_{x\in X}\big((b,x)\in f(a,x)\big)\big)=\mathtt{T}\big\}\big)\big].\end{aligned}$$

At the same time, the right hand-side of the superposing axiom is:

$$\begin{aligned}&\mathrm{id}^{\mathbf{UPos}}_{C}\otimes\mathrm{Tr}^{X}_{A,B}(f)\\ &=[c\mapsto\uparrow\{c\}]\ \otimes\ [a\mapsto\{b\in B\,|\,(\bigvee_{x\in X}((b,x)\in u(a,x)))=\mathtt{T}\}]\\ &=[(c,a)\mapsto(\uparrow\{c\}\times\{b\in B\,|\,(\bigvee_{x\in X}((b,x)\in u(a,x)))=\mathtt{T}\})],\end{aligned}$$

and hence the superposing axiom also holds.

Finally, we can also show that the yanking axiom holds:

$$\begin{aligned}\mathrm{Tr}^{X}_{X,X}(\sigma_{X,X})&=[x\mapsto\{x'\in X\,|\,(\bigvee_{x^*\in X}((x',x^*)\in\sigma_{X,X}(x,x^*)))=\mathtt{T}\}]\\ &=[x\mapsto\{x'\in X\,|\,(\bigvee_{x^*\in X}((x',x^*)\in\uparrow\{x^*\}\times\uparrow\{x\}))=\mathtt{T}\}]\\ &=[x\mapsto\{x'\in X\,|\,(\bigvee_{x^*\in X}(x'\in\uparrow\{x^*\}\ \wedge\ x^*\in\uparrow\{x\}))=\mathtt{T}\}]\\ &=[x\mapsto\{x'\in X\,|\,x'\in\uparrow\{x\}\}]\\ &=[x\mapsto\uparrow\{x\}]\\ &=\mathrm{id}^{\mathbf{UPos}}_{X}.\end{aligned}$$

□

Lemma 9.42. *The* **LPos** *category is a traced monoidal category with trace defined for any posets* $A, B, X \in \mathsf{Ob}(\mathbf{LPos})$ *as*

$$\begin{aligned}\mathrm{Tr}^X_{A,B} \colon\ & (A \otimes_{\mathbf{LPos}} X \to \mathsf{L}(B \otimes_{\mathbf{LPos}} X)) \to (A \to \mathsf{L}B)\,,\\ & l \mapsto [a \mapsto \{b \in B \mid (\textstyle\bigvee_{x\in X}((b,x) \in l(a,x))) = \mathtt{T}\}]\,.\end{aligned}$$

We will also denote the trace for morphisms in **LPos** *by* $\mathrm{Tr}_{\mathbf{LPos}}$.

Proof. Analogous to the proof of Lemma 9.41. □

Lemma 9.43. *The* $\multimapdotbothvert$ *functor preserves traces. In other words:*

$$\multimapdotbothvert(\mathrm{Tr}_{\mathbf{DP}}(d)) = \mathrm{Tr}_{\mathbf{DP}}\left(\multimapdotbothvert(d)\right),$$

for all $d \in \mathrm{Hom}_{\mathbf{DP}}(A \times X, B \times X)$, *and for all* $A, B, X \in \mathsf{Ob}(\mathbf{DP})$.

Proof. Recall that the trace of d is:

$$\begin{aligned}\mathrm{Tr}_{\mathbf{DP}}(d) \colon A^{\mathrm{op}} \times B &\to \mathtt{Bool},\\ \langle a, b\rangle &\mapsto \bigvee_{x\in X} d((a,x),(b,x)).\end{aligned}$$

Hence dual of the trace of d is:

$$\begin{aligned}\multimapdotbothvert(\mathrm{Tr}_{\mathbf{DP}}(d)) \colon B \times A^{\mathrm{op}} &\to \mathtt{Bool},\\ \langle b, a\rangle &\mapsto \bigvee_{x\in X} d((a,x),(b,x)).\end{aligned}$$

The dual of d is:

$$\begin{aligned}\multimapdotbothvert(d) \colon (B \times X) \times (A \times X)^{\mathrm{op}} &\to \mathtt{Bool},\\ \langle (b,x),(a,x)\rangle &\mapsto d((a,x),(b,x)).\end{aligned}$$

And the trace of the dual of d is:

$$\begin{aligned}\mathrm{Tr}_{\mathbf{DP}}\left(\multimapdotbothvert(d)\right) : B \times A^{\mathrm{op}} &\to \mathtt{Bool},\\ \langle b, a\rangle &\mapsto \bigvee_{x\in X^{\mathrm{op}}} \multimapdotbothvert(d)((b,x),(a,x)) = \bigvee_{x\in X^{\mathrm{op}}} ((a,x),(b,x)),\end{aligned}$$

and hence $\multimapdotbothvert(\mathrm{Tr}_{\mathbf{DP}}(d)) = \mathrm{Tr}_{\mathbf{DP}}\left(\multimapdotbothvert(d)\right)$. □

Lemma 9.44. *The* $\swarrow$ *functor preserves traces. In other words:*

$$\swarrow(\mathrm{Tr}_{\mathbf{UPos}}(u)) = \mathrm{Tr}_{\mathbf{LPos}}(\swarrow(u)),$$

for all $u \in \mathrm{Hom}_{\mathbf{UPos}}(A \otimes_{\mathbf{UPos}} X, B \otimes_{\mathbf{UPos}} X)$, *and for all* $A, B, X \in \mathsf{Ob}(\mathbf{UPos})$.

Proof. Recall from Lemma 9.41 that

$$\begin{aligned}\mathrm{Tr}_{\mathbf{UPos}}(u)\colon A &\to \mathsf{U}B,\\ a &\mapsto \{b \in B \mid \textstyle\bigvee_{x\in X}(b,x) \in u(a,x) = \mathtt{T}\}\,.\end{aligned}$$

Then, applying the $\swarrow^{\!\circ}$ functor to it we obtain:

$$\begin{aligned}\swarrow^{\!\circ}(\mathrm{Tr}_{\mathbf{UPos}}(u))\colon A^{\mathrm{op}} &\to \mathsf{L}(B^{\mathrm{op}}),\\ a &\mapsto \{b \in B \mid \textstyle\bigvee_{x\in X}(b,x) \in u(a,x) = \mathtt{T}\}\,.\end{aligned}$$

On the other hand, first applying $\swarrow^{\!\circ}$ to u:

$$\begin{aligned}\swarrow^{\!\circ}(u)\colon (A \otimes_{\mathbf{LPos}} X)^{\mathrm{op}} &\to \mathsf{L}((B \otimes_{\mathbf{LPos}} X)^{\mathrm{op}}),\\ \langle a, x\rangle &\mapsto u(a,x).\end{aligned}$$

Taking its trace we obtain:

$$\begin{aligned}\mathrm{Tr}_{\mathbf{LPos}}(\swarrow^{\!\circ}(u))\colon A^{\mathrm{op}} &\to \mathsf{L}(B^{\mathrm{op}}),\\ a &\mapsto \{b \in B \mid \textstyle\bigvee_{x\in X^{\mathrm{op}}}(b,x) \in u(a,x) = \mathtt{T}\}\,,\end{aligned}$$

hence it holds that $\swarrow^{\!\circ}(\mathrm{Tr}_{\mathbf{UPos}}(u)) = \mathrm{Tr}_{\mathbf{LPos}}(\swarrow^{\!\circ}(u))$. □

Lemma 9.45. *The ${}_{\circ\!}\nearrow$ functor preserves traces. In other words:*

$${}_{\circ\!}\nearrow(\mathrm{Tr}_{\mathbf{LPos}}(l)) = \mathrm{Tr}_{\mathbf{UPos}}({}_{\circ\!}\nearrow(l)),$$

for all $l \in \mathrm{Hom}_{\mathbf{LPos}}(A \otimes_{\mathbf{LPos}} X, B \otimes_{\mathbf{LPos}} X)$, and all $A, B, X \in \mathsf{Ob}(\mathbf{LPos})$.

Proof. Analogous to the proof of Lemma 9.44. □

Lemma 9.46. *The Π_f^{DP} functor preserves traces. In other words:*

$$\Pi_f^{DP}(\mathrm{Tr}_{\mathbf{DP}}(d)) = \mathrm{Tr}_{\mathbf{UPos}}\left(\Pi_f^{DP}(d)\right),$$

for all $d \in \mathrm{Hom}_{\mathbf{DP}}(A \times X, B \times X)$, and all $A, B, X \in \mathsf{Ob}(\mathbf{DP})$.

Proof. Recall that the trace of d is:

$$\begin{aligned}\mathrm{Tr}_{\mathbf{DP}}(d)\colon A^{\mathrm{op}} \times B &\to \mathtt{Bool},\\ \langle a, b\rangle &\mapsto \bigvee_{x\in X} d((a,x),(b,x)).\end{aligned}$$

Applying the Π_f^{DP} functor to it we obtain:

$$\begin{aligned}\Pi_f^{\mathrm{DP}}(\mathrm{Tr}_{\mathbf{DP}}(d))\colon A &\to \mathsf{U}B,\\ a &\mapsto \{b \in B \mid \textstyle\bigvee_{x\in X} d((a,x),(b,x)) = \mathtt{T}\}\,.\end{aligned}$$

However, first applying Π_f^{DP} to d we obtain:

$$\begin{aligned}\Pi_f^{\mathrm{DP}}(d) \colon A \otimes_{\mathbf{UPos}} X &\to B \otimes_{\mathbf{UPos}} X, \\ \langle a, x\rangle &\mapsto \{\langle b, x\rangle \in B \otimes_{\mathbf{UPos}} X \mid d((a,x),(b,x)) = \mathtt{T}\}\,.\end{aligned}$$

And then applying the trace to the result:

$$\begin{aligned}\mathrm{Tr}_{\mathbf{UPos}}(\Pi_f^{\mathrm{DP}}(d)) \colon A &\to \mathsf{U}B, \\ a &\mapsto \{b \in B \mid \textstyle\bigvee_{x \in X} d((a,x),(b,x)) = \mathtt{T}\}\,,\end{aligned}$$

hence $\Pi_f^{\mathrm{DP}}(\mathrm{Tr}_{\mathbf{DP}}(d)) = \mathrm{Tr}_{\mathbf{UPos}}\left(\Pi_f^{\mathrm{DP}}(d)\right)$. □

Lemma 9.47. *The Π_r^{DP} functor preserves traces. In other words:*

$$\Pi_r^{DP}(\mathrm{Tr}_{\mathbf{DP}}(d)) = \mathrm{Tr}_{\mathbf{LPos}}\left(\Pi_r^{DP}(d)\right),$$

for all $d \in \mathrm{Hom}_{\mathbf{DP}}(A, B)$, and all $A, B \in \mathsf{Ob}(\mathbf{DP})$.

Proof.

$$\begin{aligned}\Pi_r^{\mathrm{DP}}(\mathrm{Tr}_{\mathbf{DP}}(d)) &= \swarrow\left(\Pi_f^{\mathrm{DP}}\left(⧟\,(\mathrm{Tr}_{\mathbf{DP}}(d))\right)\right) \\ &= \swarrow\left(\Pi_f^{\mathrm{DP}}\left(\mathrm{Tr}_{\mathbf{DP}}\left(⧟(d)\right)\right)\right) \\ &= \swarrow\left(\mathrm{Tr}_{\mathbf{UPos}}\left(\Pi_f^{\mathrm{DP}}\left(⧟(d)\right)\right)\right) \\ &= \mathrm{Tr}_{\mathbf{LPos}}\left(\swarrow\left(\Pi_f^{\mathrm{DP}}\left(⧟(d)\right)\right)\right) \\ &= \mathrm{Tr}_{\mathbf{LPos}}\left(\Pi_r^{\mathrm{DP}}\left(⧟\left(⧟(d)\right)\right)\right) \\ &= \mathrm{Tr}_{\mathbf{LPos}}\left(\Pi_r^{\mathrm{DP}}(d)\right).\end{aligned}$$

We use the results from Lemmas 9.20, 9.22, 9.43, 9.44, and 9.46 in the above proof, as well as the fact that $⧟$ is an involute functor. Alternatively, one can use the same technique as we used for Lemma 9.46 to reach the same result directly. □

9.7 Bonus: Interpretation with monads

In this section we will explore some monadic connections underlying the structure of co-design. While the results we present here are not necessary for following the next chapter, we believe they will be of interest to the readers curious about co-design.

We start by recalling the definitions of monads and Kleisli categories (Lane, 1998, Ch. VI). Then we introduce the U functor and the U monad that it gives rise to.

Definition 9.48 (Monad)**.** Let $\mathbf{C}$ be a category. A *monad* on $\mathbf{C}$ consists of:

i. a functor $T\colon \mathbf{C} \to \mathbf{C}$;
ii. a natural transformation $\eta\colon \mathrm{id}_{\mathbf{C}} \Rightarrow T$ called *unit*;
iii. a natural transformation $\mu\colon T \mathbin{⨟} T \Rightarrow T$ called *composition* or *multiplication*.

For every $X \in \mathsf{Ob}(\mathbf{C})$, the constituents must satisfy *left and right unitality*:

$$\begin{tikzcd} T(X) \arrow[r, "\eta_{T(X)}"] \arrow[dr, "\mathrm{id}^{\mathbf{C}}_{T(X)}"'] & T(T(X)) \arrow[d, "\mu_X"] \\ & T(X) \end{tikzcd} \qquad \begin{tikzcd} T(X) \arrow[r, "T(\eta_X)"] \arrow[dr, "\mathrm{id}^{\mathbf{C}}_{T(X)}"'] & T(T(X)) \arrow[d, "\mu_X"] \\ & T(X) \end{tikzcd}$$

and *associativity*:

$$\begin{tikzcd} T(T(T(X))) \arrow[r, "T(\mu_X)"] \arrow[d, "\mu_{T(X)}"] & T(T(X)) \arrow[d, "\mu_X"] \\ T(T(X)) \arrow[r, "\mu_X"] & T(X) \end{tikzcd}$$

Definition 9.49 (Kleisli morphism)**.** Let $\langle T, \eta, \mu \rangle$ be a monad on a category $\mathbf{C}$. A *Kleisli morphism* of T from X to Y is a morphism $k\colon X \to TY$ of $\mathbf{C}$.

Definition 9.50 (Kleisli composition)**.** Let $\langle T, \eta, \mu \rangle$ be a monad on a category $\mathbf{C}$. Let $k\colon X \to TY$ and $h\colon Y \to TZ$. We define the *Kleisli composition* of k and h to be the morphism $k \mathbin{⨟} h\colon X \to TZ$, given by

$$X \xrightarrow{k} TY \xrightarrow{Th} TTZ \xrightarrow{\mu} TZ.$$

Definition 9.51 (Kleisli category)**.** Let $\langle T, \eta, \mu \rangle$ be a monad on a category $\mathbf{C}$. The *Kleisli category of* T is the category $\mathbf{C}_T$ with:

i. $\mathsf{Ob}(\mathbf{C}_T) = \mathsf{Ob}(\mathbf{C})$, i.e; it has the same objects as $\mathbf{C}$;
ii. $\mathrm{Hom}_{\mathbf{C}_T}(X, Y) = \mathrm{Hom}_{\mathbf{C}}(X, TY)$, i.e; the morphisms are the Kleisli morphisms of T (Definition 9.49);
iii. identities are given by the units $\eta\colon X \to TX$ for each object X;
iv. the composition of morphisms is given by the Kleisli composition (Definition 9.50).

Definition 9.52 (Category **Pos**)**.** The category **Pos** is defined by:

i. *Objects*: The objects of this category are all posets.
ii. *Morphisms*: The morphisms between any pair of posets X, Y are the monotone maps from X to Y.
iii. *Identity morphism*: The identity morphism for the poset X is the identity function id_X, which maps every element of X to itself.

iv. *Composition operation*: The composition operation is function composition.

Definition 9.53 (The U functor)**.** The *U functor (endofunctor)* on **Pos** maps:

i. A poset $P \in \mathsf{Ob}(\mathbf{Pos})$ to its upper set $\mathsf{U}P$, which is also an object in **Pos** (Lemma 7.2),
ii. A morphism (monotone function) $f : P \to Q$ to the monotone function:

$$\begin{aligned} U(f) \colon \mathsf{U}P &\to \mathsf{U}Q, \\ P' &\mapsto \uparrow\left(\bigcup_{p \in P'} \{f(p)\}\right). \end{aligned}$$

We will use the following results throughout the rest of this section:

Lemma 9.54. *Given two posets P, Q, a monotone functions $f : P \to Q$, and a family of singleton sets $\{S_i\}_{i \in I}$, $S_i = \{s_i\}$ with $s_i \in P$, the following holds true:*

$$\uparrow\left(\bigcup_{p \in \uparrow \bigcup_{i\in I} S_i} \{f(p)\}\right) = \uparrow\left(\bigcup_{i \in I} \{f(s_i)\}\right).$$

Proof. First, let's show that

$$\uparrow\left(\bigcup\nolimits_{p \in \uparrow \bigcup_{i\in I} S_i} \{f(p)\}\right) \subseteq \uparrow\left(\bigcup\nolimits_{i \in I} \{f(s_i)\}\right).$$

Take a

$$q \in \uparrow\left(\bigcup\nolimits_{p \in \uparrow \bigcup_{i\in I} S_i} \{f(p)\}\right).$$

That means that there exists a

$$q' \in \bigcup\nolimits_{p \in \uparrow \bigcup_{i\in I} S_i} \{f(p)\}$$

such that $q' \leq_Q q$. Hence, there is a $p' \in \uparrow \bigcup_{i \in I} S_i$ such that $q' = f(p')$. If p' is in this upper closure, then there must be an $i' \in I$ such that $s_{i'} \leq_P p'$, and due to the monotonicity of f we get:

$$f(s_{i'}) \leq_Q f(p') = q' \leq_Q q.$$

As $s_{i'}$ is in the union in the right-hand set, and as any $q^* \in Q$ such that $f(s_{i'}) \leq_Q q^*$ is in its upper closure, a must also be in it.

Now, let's show that

$$\uparrow\left(\bigcup\nolimits_{p \in \uparrow \bigcup_{i\in I} S_i} \{f(p)\}\right) \supseteq \uparrow\left(\bigcup\nolimits_{i \in I} \{f(s_i)\}\right).$$

Take a

$$q \in \uparrow \left(\bigcup_{i \in I} \{f(s_i)\}\right).$$

Then there's a $i' \in I$ such that $f(s_{i'}) \leq_Q q$. But then this $f(s_{i'})$ is also in the union in the right-hand side, hence any $q^* \geq_Q f(s_{i'})$ must be in the upper closure of that union. Hence, q is in the right-hand side set. As the two sets on the side of the equation in the lemma are subsets of each other, they must be equal. □

Lemma 9.55. *Given two posets P, Q and a monotone functions $f : P \to Q$, the following holds true for all $p \in P$:*

$$\uparrow\left(\bigcup_{p' \in \uparrow\{p\}} \{f(p')\}\right) = \uparrow\{f(p)\}.$$

Proof.
Short proof: Follows directly from Lemma 9.54 when we take a family of singleton sets with only the set $\{p\}$.
Full proof: First, let's show that

$$\uparrow\left(\bigcup_{p' \in \uparrow\{p\}} \{f(p')\}\right) \subseteq \uparrow\{f(p)\}.$$

Take a

$$q \in \uparrow\left(\bigcup_{p' \in \uparrow\{p'\}} \{f(p)\}\right).$$

Then there must exist a $p' \in P$ such that $p \leq_P p'$ and $f(p') \leq_P q$. But due to the monotonicity of f, we know that $f(p) \leq_P f(p') \leq_P q$ and hence it must hold that $q \in \uparrow\{f(p)\}$.

Now, let's show that

$$\uparrow\left(\bigcup_{p' \in \uparrow\{p\}} \{f(p')\}\right) \supseteq \uparrow\{f(p)\}.$$

Take a q in $\uparrow\{f(p)\}$. Then that means that $f(p) \leq_P q$. But $f(p) \in \bigcup_{p' \in \uparrow\{p\}} \{f(p')\}$, hence q must be in its upper closure. As the two sets on the side of the equation in the lemma are subsets of each other, they must be equal. □

Lemma 9.56. *Given a set A and a family $\{A_i\}_{i \in I}$ of subsets of A indexed by I, the following holds true:*

$$\bigcup \bigcup_{i \in I} \{A_i\} = \bigcup_{i \in I} A_i.$$

Note that $\{A_i\}$ is a singleton set with an element a subset of A, hence a set of sets.

Lemma 9.57. *The U functor is indeed a functor.*

Proof. U respects the source and targets of arrows. Furthermore, it maps id_P to $\mathrm{id}_{\mathsf{U}P}$. Finally, we need to show that U respects also morphism composition.

Take $f : P \to Q$ and $g : Q \to R$. Then:

$$U(f \mathbin{;} g) = \left[P' \mapsto \uparrow \left(\bigcup_{p \in P'} \{g(f(p))\}\right)\right].$$

At the same time:

$$\begin{aligned} U(f) \mathbin{;} U(g) &= \left[P' \mapsto \uparrow \left(\bigcup_{p \in P'} \{f(p)\}\right)\right] \mathbin{;} \left[Q' \mapsto \uparrow \left(\bigcup_{q \in Q'} \{g(q)\}\right)\right] \\ &= \left[P' \mapsto \uparrow \left(\bigcup_{q \in \left(\uparrow \bigcup_{p \in P'} \{f(p)\}\right)} \{g(q)\}\right)\right] \\ &= \left[P' \mapsto \uparrow \left(\bigcup_{p \in P'} \{g(f(p))\}\right)\right], \end{aligned}$$

where the last equality is due Lemma 9.54. Hence U is indeed a functor. □

Definition 9.58 (The U monad)**.** We define the U *monad* on **Pos** consisting of:

i. The functor U (Definition 9.53).
ii. The unit natural transformation $\eta^U \colon \mathrm{id}_{\mathbf{Pos}} \Rightarrow U$, which associates to every $P \in \mathsf{Ob}(\mathbf{Pos})$ the following morphism in **Pos**:

$$\begin{aligned} \eta^U_P \colon P &\to \mathsf{U}P, \\ p &\mapsto \uparrow\{p\}. \end{aligned}$$

iii. The composition natural transformation $\mu^U \colon U \mathbin{;} U \Rightarrow U$, which associates to every $P \in \mathsf{Ob}(\mathbf{Pos})$ the following morphism in **Pos**:

$$\begin{aligned} \mu^U_P \colon \mathsf{U}(\mathsf{U}P) &\to \mathsf{U}P, \\ P'' &\mapsto \bigcup_{P' \in P''} P'. \end{aligned}$$

We will refer both to the monad and its constitute functor as U. The difference should be clear from the context.

Remark 9.59*.* Notice that the unit natural transformation of the U monad uses the fact that every element of the poset can be uniquely identified with an upper set (Lemma 7.14). In a way, one can think of the upper sets as "generalized objects" of the poset.

Lemma 9.60. *The U monad is indeed a monad.*

Proof. Proving that U is a monad is a multi-step exercise. In particular, we need to show the following things:

i. That η^U is indeed a natural transformation;

ii. That μ^U is indeed a natural transformation;
iii. That the left unitality holds;
iv. That the right unitality holds;
v. That the associativity holds.

η^U **is indeed a natural transformation.** We need to show that for every $f \in \mathrm{Hom}_{\mathbf{Pos}}(P, Q)$ it holds that

$$\mathrm{id}_{\mathbf{Pos}}(f) \fatsemi \eta^U_Q = \eta^U_P \fatsemi U(f).$$

The left-hand side equals:

$$\mathrm{id}_{\mathbf{Pos}}(f) \fatsemi \eta^U_Q = f \ \fatsemi\ \left[q \mapsto \uparrow\{q\}\right] = \left[p \mapsto \uparrow\{f(p)\}\right].$$

The right-hand side equals:

$$\begin{aligned}
\eta^U_P \fatsemi U(f) &= \left[p \mapsto \uparrow\{p\}\right] \ \fatsemi\ \left[P' \mapsto \uparrow \textstyle\bigcup_{p' \in P'} \{f(p')\}\right] \\
&= \left[p \mapsto \uparrow \left(\textstyle\bigcup_{p' \in \uparrow\{p\}} \{f(p')\}\right)\right] \\
&= \left[p \mapsto \uparrow\{f(p)\}\right],
\end{aligned}$$

where the last equality is due Lemma 9.55.

μ^U **is indeed a natural transformation.** We need to show that for every $f \in \mathrm{Hom}_{\mathbf{Pos}}(P, Q)$ it holds that

$$U(U(f)) \ \fatsemi\ \mu^U_Q = \mu^U_P \fatsemi U(f).$$

The left-hand side equals:

$$\begin{aligned}
U(U(f)) \ \fatsemi\ \mu^U_Q &= U\left(\left[P' \mapsto \uparrow\left(\textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right]\right) \ \fatsemi\ \mu^U_Q \\
&= \left[P'' \mapsto \uparrow\left(\textstyle\bigcup_{P' \in P''} \left\{\uparrow\left(\textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right\}\right)\right] \ \fatsemi\ \mu^U_Q \\
&= \left[P'' \mapsto \uparrow\left(\bigcup_{P' \in P''} \left\{\uparrow\left(\bigcup_{p \in P'}\{f(p)\}\right)\right\}\right)\right] \ \fatsemi\ \left[Q'' \mapsto \bigcup_{Q' \in Q''} Q'\right] \\
&= \left[P'' \mapsto \bigcup_{Q' \in \uparrow\left(\bigcup_{P' \in P''}\left\{\uparrow\left(\bigcup_{p \in P'}\{f(p)\}\right)\right\}\right)} Q'\right] \\
&= \left[P'' \mapsto \bigcup \uparrow\left(\textstyle\bigcup_{P' \in P''} \left\{\uparrow\left(\textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right\}\right)\right] \\
&= \left[P'' \mapsto \bigcup \left(\textstyle\bigcup_{P' \in P''} \left\{\uparrow\left(\textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right\}\right)\right] \\
&= \left[P'' \mapsto \left(\textstyle\bigcup_{P' \in P''} \uparrow\left(\textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right)\right] \\
&= \left[P'' \mapsto \uparrow\left(\textstyle\bigcup_{P' \in P''} \textstyle\bigcup_{p \in P'}\{f(p)\}\right)\right],
\end{aligned}$$

where we use the facts that union of upper sets is an upper set (Lemma 7.15), Lemma 9.56, and the fact that the union of upper set is the upper closure of the union of sets. The right-hand side equals:

$$
\begin{aligned}
\mu_P^U \mathbin{⨾} U(f) &= [P'' \mapsto \textstyle\bigcup_{P' \in P''} P'] \mathbin{⨾} \left[P' \mapsto \uparrow\left(\bigcup_{p \in P'} \{f(p)\}\right)\right] \\
&= \left[P'' \mapsto \uparrow\left(\bigcup_{p \in \bigcup_{P' \in P''} P'} \{f(p)\}\right)\right] \\
&= \left[P'' \mapsto \uparrow\left(\textstyle\bigcup_{P' \in P''} \bigcup_{p \in P'} \{f(p)\}\right)\right].
\end{aligned}
$$

Left unitality holds. We heed to show that for all $P \in \mathsf{Ob}(\mathbf{Pos})$ it holds that:

$$\eta_{U(P)}^U \mathbin{⨾} \mu_P^U = \mathrm{id}_{U(P)}^{\mathbf{Pos}}.$$

We have:

$$
\begin{aligned}
\eta_{U(P)}^U \mathbin{⨾} \mu_P^U &: \mathsf{U}U \to \mathsf{U}U, \\
P' &\mapsto \bigcup_{P'' \in \uparrow\{P'\}} P'' = P',
\end{aligned}
$$

where the equality is due to the upper sets of an upper set P' being subsets of P', hence their union is again P'. The is the same as the identity morphism on $U(P) = \mathsf{U}P$ in **Pos** (Definition 9.52).

Right unitality holds. We need to show that for all $P \in \mathsf{Ob}(\mathbf{Pos})$ it holds that:

$$U(\eta_P^U) \mathbin{⨾} \mu_P^U = \mathrm{id}_{U(P)}^{\mathbf{UPos}}.$$

We have:

$$U(\eta_P^U) \mathbin{⨾} \mu_P^U : \mathsf{U}P \to \mathsf{U}P,$$

$$
\begin{aligned}
U(\eta_P^U) \mathbin{⨾} \mu_P^U &= \left[P' \mapsto \textstyle\bigcup_{P'' \in U(\lambda p \uparrow\{p\})(P')} P''\right]. \\
&= \left[P' \mapsto \textstyle\bigcup_{P'' \in \uparrow(\bigcup_{p \in P'}\{\uparrow\{p\}\})} P''\right] \\
&= \left[P' \mapsto \bigcup \uparrow\left(\textstyle\bigcup_{p \in P'}\{\uparrow\{p\}\}\right)\right] \\
&= \left[P' \mapsto \uparrow \bigcup \textstyle\bigcup_{p \in P'}\{\uparrow\{p\}\}\right] \\
&= \left[P' \mapsto \uparrow \textstyle\bigcup_{p \in P'} \uparrow\{p\}\right] \\
&= [P' \mapsto \uparrow P'] \\
&= [P' \mapsto P'],
\end{aligned}
$$

where we again use the facts that union of upper sets is an upper set as shown in Lemma 7.15, Lemma 9.56, the fact that the union of upper set is the upper closure of the union of sets, and that taking the upper closure of an upper set is redundant.

Associativity holds. We need to show that for all $P \in \mathsf{Ob}(\mathbf{Pos})$ it holds that:

$$U(\mu^U_P) \fatsemi \mu^U_P = \mu^U_{U(P)} \fatsemi \mu^U_P .$$

The left-hand side of this equation is:

$$\begin{aligned}
U(\mu^U_P) \fatsemi \mu^U_P &= U\left([P'' \mapsto \textstyle\bigcup_{P' \in P''} P']\right) \;\fatsemi\; [P'' \mapsto \textstyle\bigcup_{P' \in P''} P'] \\
&= [P''' \mapsto \uparrow(\textstyle\bigcup_{P'' \in P'''} \{\bigcup_{P' \in P''} P'\})] \;\fatsemi\; [P'' \mapsto \textstyle\bigcup_{P' \in P''} P'] \\
&= \left[P''' \mapsto \bigcup_{P^* \in \uparrow(\bigcup_{P'' \in P'''} \{\bigcup_{P' \in P''} P'\})} P^*\right] \\
&= \left[P''' \mapsto \bigcup \uparrow(\textstyle\bigcup_{P'' \in P'''} \{\bigcup_{P' \in P''} P'\})\right] \\
&= \left[P''' \mapsto \uparrow \bigcup \textstyle\bigcup_{P'' \in P'''} \{\bigcup_{P' \in P''} P'\}\right] \\
&= [P''' \mapsto \uparrow \textstyle\bigcup_{P'' \in P'''} \bigcup_{P' \in P''} P'] \\
&= [P''' \mapsto \textstyle\bigcup_{P'' \in P'''} \bigcup_{P' \in P''} P'] ,
\end{aligned}$$

where we use Lemmas 7.15 and 9.56, the fact that the union of upper sets is the upper closure of the union of sets, and the fact that taking the upper closure of an upper set is redundant. The right-hand side of this equation is:

$$\begin{aligned}
\mu^U_{U(P)} \fatsemi \mu^U_P &= [P''' \mapsto \textstyle\bigcup_{P'' \in P'''} P''] \;\fatsemi\; [P'' \mapsto \textstyle\bigcup_{P' \in P''} P'] \\
&= \left[P''' \mapsto \textstyle\bigcup_{P' \in \bigcup_{P'' \in P'''} P''} P'\right] \\
&= [P''' \mapsto \textstyle\bigcup_{P'' \in P'''} \bigcup_{P' \in P''} P'] .
\end{aligned}$$

□

Lemma 9.61. **UPos** *is the Kleisli category of* U.

Proof. Going through the points of Definition 9.51 we have:

i. Objects of both **UPos** and **Pos** are posets.
ii. $\mathrm{Hom}_{\mathbf{UPos}}(P, Q)$ is a set of monotone functions from P to $\mathsf{U}Q$. These are also the morphisms of $\mathrm{Hom}_{\mathbf{Pos}}(P, \mathsf{U}Q)$.
iii. Identities in **UPos** are:
$$\begin{aligned}
\mathrm{id}_A \colon A &\to \mathsf{U}A, \\
a &\mapsto \uparrow\{a\},
\end{aligned}$$
as in Definition 9.3. This is the same as the unit natural transformation η^U (Definition 9.58).

iv. The composition of morphisms in **UPos** was defined as the fish operator $\ltimes$ (Definition 9.3). We now have to show that the Kleisli composition on the monad U is the exact same operation. Take two morphisms $f : P \to \mathsf{U}Q$, $g : Q \to \mathsf{U}R$ in **UPos**. Their Kleisli composition is:

$$P \xrightarrow{f} \mathsf{U}Q \xrightarrow{U(g)} \mathsf{UU}R \xrightarrow{\mu^U_R} \mathsf{U}R.$$

$$\begin{aligned}
f \mathbin{⨟} U(g) \mathbin{⨟} \mu^U_R &= [p \mapsto f(p)] \mathbin{⨟} \left[Q' \mapsto \uparrow\left(\textstyle\bigcup_{q\in Q'}\{g(q)\}\right)\right] \mathbin{⨟} \mu^U_R \\
&= \left[p \mapsto \uparrow\left(\textstyle\bigcup_{q\in f(p)}\{g(q)\}\right)\right] \mathbin{⨟} [R'' \mapsto \textstyle\bigcup_{R'\in R''} R'] \\
&= \left[p \mapsto \textstyle\bigcup_{R'\in\uparrow(\bigcup_{q\in f(p)}\{g(q)\})} R'\right] \\
&= \left[p \mapsto \bigcup \uparrow\left(\textstyle\bigcup_{q\in f(p)}\{g(q)\}\right)\right] \\
&= \left[p \mapsto \uparrow\bigcup\left(\textstyle\bigcup_{q\in f(p)}\{g(q)\}\right)\right] \\
&= \left[p \mapsto \uparrow\bigcup \textstyle\bigcup_{q\in f(p)}\{g(q)\}\right] \\
&= [p \mapsto \uparrow\textstyle\bigcup_{q\in f(p)} g(q)] \\
&= [p \mapsto \textstyle\bigcup_{q\in f(p)} g(q)],
\end{aligned}$$

which is precisely the $\ltimes$ operator. In the above we use the fact that the union of upper sets is the upper closure of the union of sets, that taking the upper closure of an upper set is redundant, as well as Lemma 9.56.

□

Chapter 10

Design problems and compositional computation

Now we can finally see how the theory of co-design that we developed in the previous chapters can fit in the theory of compositional computational systems from Part I. The reader most certainly made the connection that perhaps a design *problem* will be a *problem* in some compositional computational system. And similarly, the methods for solving design problems which we discussed in Chapter 8 would be procedures in the respective system.

10.1 Design problems are problems

Let's start by taking another look at design problems. Recall that design problems are feasibility relations between two posets and are represented as the morphisms of the **DP** category. Solutions, on the other hand, are functions from the elements of one poset to the antichains of another poset. Finally, recall that the crux of the story in Chapter 8 was that not all design problems can be solved using the solutions techniques that we provided. Hence, we need to restrict ourselves only to well-behaved design problems Definition 8.39. To do that, we define the $\mathbf{DP}^\star$ category of well-behaved design problems:

Definition 10.1 (The $\mathbf{DP}^\star$ category)**.** The $\mathbf{DP}^\star$ category is a subcategory (Definition 2.21) of the **DP** category that has:

i. the same objects as **DP**, i.e. posets, and
ii. morphisms between any two posets P and Q being the subset

$$\mathrm{Hom}_{\mathbf{DP}^\star}(P, Q) \subseteq \mathrm{Hom}_{\mathbf{DP}}(P, Q)$$

of well-behaved design problems (Definition 8.39).

We will also define the category of functions with range antichains which will represent solution maps of design problems (Definition 8.15):

Definition 10.2 (The **APos** category)**.** The **APos** category has:

i. objects which are posets;
ii. morphisms between two objects $A, B \in \mathsf{Ob}(\mathbf{APos})$ which are solution maps, i.e. functions of the form

$$f : A \to \mathcal{A}B;$$

iii. composition of two morphisms $\alpha : A \to \mathcal{A}B$ and $\beta : B \to \mathcal{A}C$ defined as in Lemma 8.18:

$$\begin{aligned} \alpha \fatsemi \beta : \quad & A \to \mathcal{A}C, \\ & a \mapsto \operatorname*{Min}_{\leq C} \bigcup_{s \in \alpha(a)} \beta(s); \end{aligned}$$

iv. identity morphism for an object A is the solution map of the identity feasibility relation (Lemma 8.19):

$$\begin{aligned} \mathrm{id}_A : \quad & A \to \mathcal{A}A \\ & a \mapsto \{a\}. \end{aligned}$$

Therefore, we can now consider a compositional system, where the objects of $\mathbf{Pro\mathring{b}}$ and $\mathbf{Pro\mathring{c}}$ are posets, with morphisms being respectively design problems (the morphisms of $\mathbf{DP}^\star$) and solution maps (the morphisms of **APos**). Note that this is a valid setting because posets are categories (Lemma 2.17), hence every poset can be a statement or answer semicategory of a problem. We have that the objects of $\mathbf{Pro\mathring{b}}$ and $\mathbf{Pro\mathring{c}}$ are the same as the objects of $\mathbf{DP}^\star$ and **APos**. Furthermore, the morphisms of $\mathbf{DP}^\star$ and **APos** are (some of the) morphisms in $\mathbf{Pro\mathring{b}}$ and $\mathbf{Pro\mathring{c}}$.

The morphisms of $\mathbf{Pro\mathring{b}}$ and $\mathbf{Pro\mathring{c}}$ had to be kinded function with some rigs N and M as their kind sets. We showed in Lemma 5.21 and Section 6.3, any binary relation can be represented as a kinded function with a kind set `Bool`. And a design problem was defined as a binary relation (Definition 2.14). Hence we know that all design problems appear as morphisms in the category $\mathbf{Pro\mathring{b}}(\mathbf{Pos}, B)$.

A solution map can also be represented as a binary relation. Hence, we also get that all solution maps appear as morphisms in the category $\mathbf{Pro\mathring{c}}(\mathbf{Pos}, B)$. Therefore, the solutions of design problems would end up in a category

$$\mathbf{Laputa}\,(\mathbf{Pos}, B, B, \cdot)\,,$$

for some suitable choice of a solution judgement map.

But what should that solution judgement map be? Well, we need it to be one that verifies that the antichain returned by the procedure represents the upper set of resources of the design problem. And, of course, that should hold for any choice of a functionality. Hence we can take a map:

$$\begin{aligned}\Psi^{\mathbf{DP}^{\star}}_{F,R} : \left(F \xrightarrow{B} R\right) \times \left(F \xrightarrow{B} R\right) &\to \texttt{Bool},\\ \langle \Phi, h_{\Phi}\rangle &\mapsto \bigwedge_{f\in F}\bigwedge_{r\in R} \Phi(f,r) \implies \exists r' \in R \text{ s.t. } h_{\Phi}(f,r') \wedge (r' \leq_R r).\end{aligned}$$

In the above we use the fact that $h_{\Phi}(f,r')$ would be true only for a r' that is an element of the minimal antichain of the upper set of resources that can provide f, but we want the implication to hold for all elements of the upper set, not only of the antichain.

Now we can claim that design problems are problems, solution maps are procedures, and solution maps that solve design problems are solutions in the compositional computational system

$$\mathbf{Laputa}\left(\mathbf{Pos}, B, B, \Psi^{\mathbf{DP}^{\star}}\right).$$

Then if we have two problems $d : F \nrightarrow R$ and $e : G \nrightarrow S$, they would look like that in this compositional computational system:

$$\begin{tikzcd} d \arrow[d, "\left\langle h_d, \mathrm{id}^{\mathbf{Pro\mathring{c}}(\mathbf{Pos},B)}_{\mathsf{U}R}\right\rangle"'] & & e \arrow[d, "\left\langle h_d, \mathrm{id}^{\mathbf{Pro\mathring{c}}(\mathbf{Pos},B)}_{\mathsf{U}S}\right\rangle"] \\ \mathrm{id}^{\mathbf{Pro\mathring{b}}(\mathbf{Pos},B)}_{\mathsf{U}R} & & \mathrm{id}^{\mathbf{Pro\mathring{b}}(\mathbf{Pos},B)}_{\mathsf{U}S} . \end{tikzcd} \tag{10.1}$$

Remark 10.3. You probably noticed that there is no morphism between the two problems in Equation (10.1). That does not mean that non-trivial problem reduction is impossible in this setting. For example you can consider reducing the product problem of d with an identity problem to d itself, or reducing three problems in series to the middle problem with the morphisms being the solution maps of the other two. Even more interestingly, in co-design there is also the concept of "higher-order" design problems (Censi et al., 2020). These are design problems over design problems. Hence, by solving the outer one, we end up at a new design problem, something that can be an arrow in this compositional computational system.

10.2 Design problems are a problem

A rather curious feature of co-design is that it can be to be (a part of) a compositional computational system at various levels of abstraction and representation. In the previous section we said that posets are categories, hence the objects of **Pos** form a subcategory of the semicategories and therefore can act as statement and answer semicategories in the definitions of $\mathbf{Pro\mathring{b}}$ and $\mathbf{Pro\mathring{c}}$. Then, we ended up in a situation where every object of

$$\mathbf{Laputa}\left(\mathbf{Pos}, B, B, \Psi^{\mathbf{DP}^\star}\right)$$

is a design problem and every morphism is a pair of solution maps.

However, as *all* (well-behaved) design problems form the $\mathbf{DP}^\star$ category, we can take this to be a statement semicategory, while the category **APos** can be an answer category. Then we obtain a single problem, with statements being *all* design problems and answers being *all* solution maps. So let's see how exactly this would work out and what would be the compositional computational system that can host this problem.

Let's use the theory that we developed in Chapter 9. Hence, we would be working with categories such as **UPos**, **LPos** and $\mathbf{DP}^\star$ (again, here we restrict ourselves only to well-behaved problems), and with functors such as $\Pi_f^{\mathbf{DP}^\star}, \Pi_r^{\mathbf{DP}^\star}, \circ\!\!\nearrow, \swarrow\!\!\circ$, and $\multimapdotbothvert$. This means that the compositional computational system we will end up being in is one where problems and procedures are functors between categories. We already saw such a system in Section 6.4. It was called **LAnd** and was defined as:

$$\mathbf{LAnd} \coloneqq \mathbf{Laputa}\left(\mathbf{Cat}, B, B, \Psi^{\mathbf{And}}\right).$$

Problems and procedures are now just different functors, i.e. morphisms in **Cat**. We have two problems: $\Pi_f^{\mathbf{DP}^\star}$ and $\Pi_r^{\mathbf{DP}^\star}$ which, as part of the $\mathbf{Pro\mathring{b}}(\mathbf{Cat}, B)$ category, look like:

$$\begin{tikzcd}
 & \mathbf{UPos} \\
\mathbf{DP}^\star \arrow[ur, "\Pi_f^{\mathbf{DP}^\star}"] \arrow[r, "\Pi_r^{\mathbf{DP}^\star}"'] & \mathbf{LPos}.
\end{tikzcd} \tag{10.2}$$

And the functors which represent procedures in $\mathbf{Pro\mathring{c}}(\mathbf{Cat}, B)$ are:

$$\begin{tikzcd}
\mathbf{APos} \arrow[r, "\uparrow"] & \mathbf{UPos} \arrow[d, bend left, "{\swarrow\!\circ}"] \\
\mathbf{DP}^\star \arrow[u, "S"] \arrow[loop left, "\multimapbothvert"] & \mathbf{LPos}. \arrow[u, bend left, "{\circ\!\nearrow}"]
\end{tikzcd} \tag{10.3}$$

There are two new functors there.

First, there is S. It simply maps a (primal) design problem to its corresponding solution map in **APos**. So formally, we can define it as:

$$\begin{aligned}
S: \qquad\qquad \mathbf{DP}^\star &\to \mathbf{APos}, \\
P &\mapsto P, \\
(d : F \twoheadrightarrow R) &\mapsto \left[f \xmapsto{F \to \mathcal{A}R} \operatorname*{Min}_{\leq_R}\{r \in R \mid d(f,r) = \mathtt{T}\} \right].
\end{aligned}$$

Note that this is not a process that can in general be computed. However, it does represent the action of the solutions maps as introduced in Chapter 8. If every atomic design problem comes with its own solution map (which is indeed how such problems are solved in practice (Censi, 2016)) then numerically evaluating this map would amount to solving the design problem. Furthermore, for design problems which are composed out of other design problems we just need to apply the same composition operations to their solution maps, as described in Chapter 8.

The $\uparrow$ functor acts exactly like the $\uparrow$ operator (Definition 7.1):

$$\begin{aligned}
\uparrow: \qquad\qquad \mathbf{APos} &\to \mathbf{UPos}, \\
P &\mapsto P, \\
(h : F \to \mathcal{A}R) &\mapsto \left[f \xmapsto{F \to \mathsf{U}R} \uparrow h(f) \right].
\end{aligned}$$

Now that the problems and procedures have been described in Equations (10.2) and (10.3) we are left with the task of identifying some of the procedures as solutions to some of the problems. The first important observation is that the problem functor $\Pi_f^{\mathbf{DP}^\star}$ and the procedure functor composition $S \fatsemi \uparrow$ map objects and morphisms in the exact same way, hence they trivially satisfy Equation (6.5). The other crucial realization is that every problem $\Pi_r^{\mathbf{DP}^\star}$ can be represented as a $\Pi_f^{\mathbf{DP}^\star}$ problem. We hinted at this in Lemma 9.20. Hence we also have that the problem functor $\Pi_r^{\mathbf{DP}^\star}$ and the procedure functor composition

$$\multimapbothvert \fatsemi S \fatsemi \uparrow \fatsemi {\swarrow\!\circ}$$

maps objects and morphisms in the exact same way, hence they trivially satisfy Equation (6.5). Formally, we have:

$$\begin{aligned}
\Psi^{\mathbf{LAnd}}_{\mathbf{DP}^\star,\mathbf{UPos}}\left(\Pi^{\mathbf{DP}^\star}_f,\ S \fatsemi \uparrow\right) &= \mathtt{T},\\
\Psi^{\mathbf{LAnd}}_{\mathbf{DP}^\star,\mathbf{LPos}}\left(\Pi^{\mathbf{DP}^\star}_r,\ \multimapbothvert \fatsemi S \fatsemi \uparrow \fatsemi \swarrow\right) &= \mathtt{T}.
\end{aligned}$$

It is now even more clear that the dual problem $\Pi^{\mathbf{DP}^\star}_r$ reduces to the primal problem $\Pi^{\mathbf{DP}^\star}_f$ in the **LAnd** category. Of course, the opposite reduction is also possible despite it perhaps being of little practical utility. In other words, we have the following structure in **LAnd**:

$$\begin{tikzcd}
\mathrm{id}^{\mathbf{Pro\mathring{b}}(\mathbf{Cat},B)}_{\mathbf{APos}} & \Pi^{\mathbf{DP}^\star}_f \arrow[l, "\langle S{,}\uparrow\rangle"'] \arrow[r, bend left, "\langle \multimapbothvert{,}\nearrow\rangle"] & \Pi^{\mathbf{DP}^\star}_r . \arrow[l, bend left, "\langle \multimapbothvert{,}\swarrow\rangle"]
\end{tikzcd} \tag{10.4}$$

Here we have the three problems: the primal co-design problems $\Pi^{\mathbf{DP}^\star}_f$, the dual co-design problems $\Pi^{\mathbf{DP}^\star}_r$ and the identity problem on the **APos** category $\mathrm{id}^{\mathbf{Pro\mathring{b}}(\mathbf{Cat},B)}_{\mathbf{APos}}$. We also have the pairs of procedures that allow problem reduction among these problems. And of course, as before, solving a problem would mean reducing it to an identity problem, of which we have only one in Equation (10.4). Checking that this structure satisfies the definition of the **Laputa** category (Definition 6.17) is straight-forward so we will omit it.

Remark 10.4. Compare the structure of the two compositional computational system that we saw in this chapter: Equation (10.1) and Equation (10.4). As mentioned in the beginning of this section, in Equation (10.1) we see individual design problems explicitly appearing at this level of abstraction. In Equation (10.4), however, we all design problems are "hidden" under the objects.

10.3 Design problems are a problem with lots of structure

A large part of Chapter 9 dealt with the rich structure that co-design has. We talked about monoidality, locally-lattical structures, and traces. However, we have seen none of that here yet. This section remedies that by illustrating how one can have compositional systems with a lot of structure and how co-design is a potent example of a problem in such a system.

Table 10.1: Guide to the proofs of the compositional properties of the categories used in this section.

	Monoidal category	Locally lattical category	Traced category
UPos	Lemma 9.23	Lemma 9.31	Lemma 9.41
LPos	Lemma 9.25	Lemma 9.32	Lemma 9.42
APos	Lemma 10.5	Lemma 10.5	Lemma 10.5
DP*	(Censi et al., 2020)	Lemma 9.30 and (Censi et al., 2020)	Lemma 9.40 and (Censi et al., 2020)

Table 10.2: Guide to the proofs of the compositional properties of the functors used in this section.

	Functor	Strong monoidal functor	Preserving bounded lattical structure	Preserving traces
S		Lemma 10.9	Lemma 10.10	Lemma 10.11
$\uparrow$		Lemma 10.8	Lemma 10.8	Lemma 10.8
⧟	Definition 9.12	Lemma 9.26	Lemma 9.33	Lemma 9.43
$\swarrow$	Lemma 9.9	Lemma 9.27	Lemma 9.34	Lemma 9.44
$\nearrow$	Lemma 9.9	Lemma 9.27	Lemma 9.35	Lemma 9.45
$\mathrm{id}^{\mathbf{APos}}$	By definition	Lemma 10.7	Lemma 10.7	Lemma 10.7
$\Pi_r^{\mathbf{DP}^\star}$	Lemma 9.19	Lemma 9.29	Lemma 9.37	Lemma 9.47
$\Pi_f^{\mathbf{DP}^\star}$	Lemma 9.17	Lemma 9.28	Lemma 9.36	Lemma 9.46

First, let's ensure that all our categories and functors indeed have this structure. As Table 10.1 summarizes, we already showed that the categories **DP***, **UPos**, and **LPos** are symmetric monoidal, traced, and have hom-sets with bounded lattice structure (i.e. with posetal order, joins, meets, tops, and bottoms). In Chapter 9 we also showed that the functors $\Pi_f^{\mathbf{DP}^\star}$, $\Pi_r^{\mathbf{DP}^\star}$, ⧟, $\swarrow$, $\nearrow$ preserve the operations underlying these structures. The exact proofs are listed in Table 10.2. That means that whether we first apply the operation and then the functor or first the functor and then the operation should not matter for the end result.

In the formulation in Equation (10.4) of the co-design problems and solutions in a compositional computational setting there is one additional category, **APos**, and three new functors: S, $\uparrow$, and the identity functor on **APos**. So let's show that they too possess the same properties as the constructions from Chapter 9.

Let's first take a look at **APos**. We will formalize everything we need about its compositional structure in the following lemma:

Lemma 10.5. **APos** *has the following properties:*

A. *It is a monoidal category when considering the following additional structure:*

i. *Tensor product* $\otimes$*, with the operation on objects being the poset product as defined in Definition 2.6, and operation on two morphisms* $f: A \to \mathcal{A}B$ *and* $g: C \to \mathcal{A}D$

$$\begin{aligned} f \otimes g: A \times C &\to \mathcal{A}(B \times D), \\ \langle a, c \rangle &\mapsto f(a) \times g(c), \end{aligned}$$

which is the same formulation as the solution map of the product design problem (Lemma 8.20).

ii. *Unit object being the identity poset I (Definition 2.4).*

iii. *Left unitor being the pair of morphisms*

$$\begin{aligned} \lambda_A: I \otimes A &\to \mathcal{A}A, \\ \langle \iota, a \rangle &\mapsto \{a\}, \end{aligned}$$

and

$$\begin{aligned} \lambda_A^{-1}: A &\to \mathcal{A}(I \otimes A), \\ \langle a \rangle &\mapsto \{\iota \times a\}. \end{aligned}$$

iv. *Right unitor being the pair of morphisms*

$$\begin{aligned} \rho_A: A \otimes I &\to \mathcal{A}A, \\ \langle a, \iota \rangle &\mapsto \{a\}, \end{aligned}$$

and

$$\begin{aligned} \rho_A^{-1}: A &\to \mathcal{A}(A \otimes I), \\ \langle a \rangle &\mapsto \{a \times \iota\}. \end{aligned}$$

v. *Associator being the pair of morphisms*

$$\begin{aligned} \alpha_{AB,C}: (A \otimes B) \otimes C &\to \mathcal{A}A \times (\mathcal{A}B \times \mathcal{A}C), \\ \langle (a, b), c \rangle &\mapsto \{a\} \times \{b \times c\}, \end{aligned}$$

and

$$\begin{aligned} \alpha_{A,BC}: A \otimes (B \otimes C) &\to (\mathcal{A}A \times \mathcal{A}B) \times \mathcal{A}C, \\ \langle a, (b, c) \rangle &\mapsto \{a \times b\} \times \{c\}. \end{aligned}$$

B. *It has bounded lattice structure on its hom-sets* $\mathrm{Hom}_{\mathbf{APos}}(A, B)$ *for any two posets* $A, B \in \mathsf{Ob}(\mathbf{APos})$ *with:*

i. *Order: for any two solution maps* $a_1, a_2: A \to \mathcal{A}B$*, it holds that* $a_1 \leq_{\mathbf{APos}(A,B)} a_2$ *iff*

$$a_1(a) \leq_{\mathcal{A}B} a_2(a), \quad \forall a \in A,$$

where the order is the order on antichains defined in Lemma 8.4;

ii. *Joins: for any two solution maps $a_1, a_2 \colon A \to \mathcal{A}B$, it holds that*

$$(a_1 \sqcup a_2)(a) = \operatorname*{Min}_{\leq_B} \{x_1 \sqcup x_2 \mid x_1 \in a_1(a), x_2 \in a_2(a)\}, \quad \forall a \in A,$$

the same as the definition of the solution map of the intersection design problems (Lemma 8.24);

iii. *Meets: for any two solution maps $a_1, a_2 \colon A \to \mathcal{A}B$, it holds that*

$$(a_1 \sqcap a_2)(a) = \operatorname*{Min}_{\leq_B} (a_1(a) \cup a_2(a)), \quad \forall a \in A,$$

the same as the definition of the solution map of the sum design problems (Lemma 8.21);

iv. *Top: the map $\top_{\mathbf{APos}(A,B)} : a \mapsto \emptyset$;*

v. *Bottom: the map $\bot_{\mathbf{APos}(A,B)} : a \mapsto \bot_B$.*

C. *It is a traced monoidal category with trace defined for any posets $A, B, X \in \mathsf{Ob}(\mathbf{APos})$ as*

$$\begin{aligned} \mathrm{Tr}^X_{A,B} \colon\ & (A \times X \to \mathcal{A}(B \times X)) \to (A \to \mathcal{A}B), \\ & f \mapsto \left[a \mapsto \operatorname*{Min}_{\leq_B} \left\{ b \in B \mid \left(\bigvee_{x \in X} \left((b, x) \in {\uparrow} f(a, x) \right) \right) = \mathtt{T} \right\} \right]. \end{aligned}$$

We will also denote the trace for morphisms in **APos** *by* $\mathrm{Tr}_{\mathbf{APos}}$.

Sketch of a proof. The structure of **APos** is very similar to the structure of **UPos** and **LPos** so the reader can refer to the proof for these two categories. □

Remark 10.6. It is important to mention that we restrict our interest only to well-behaved design problems and sets (Definition 8.39). For these, the joins and meets are always required to exist, and bottoms of the posets B must also exist because the well-definedness requires that all subsets of B^{op} have a join. Hence, strictly speaking the above definition is valid only for a subset of all possible subsets. Also note that our definition of trace uses the upper closure operator $\uparrow$ which means that it is biased with semantic meaning appropriate for primal design problems and not dual design problems.

Let's now look at the identity functor on the **APos** category and its properties.

Lemma 10.7. *The identity functor* id : **APos** $\to$ **APos** *is a strong monoidal functor and preserves the orders, joins, meets, tops, and bottoms of the hom-sets of* **APos**, *as well as traces in* **APos**.

Sketch of a proof. The fact that the identity functors on monoidal categories are monoidal functors can be seen immediately from the commutative diagrams in Definition 2.31. The same holds for the preserving the lattical structure and traces. Whether we first do nothing to some elements and then we perform an operation on them, or we first do the operation and then do nothing, the result is going to be exactly the same. □

Lemma 10.8. *The upper closure functor* $\uparrow$: **APos** $\to$ **UPos** *is a strong monoidal functor and preserves the orders, joins, meets, tops, and bottoms of the hom-sets of* **UPos***, as well as traces in* **UPos***.*

Sketch of a proof. The upper closure functor behaves similarly to the $\Pi_f^{\mathbf{DP}^\star}$ functor. As the proof of the above properties follows closely the proof of these properties for the $\Pi_f^{\mathbf{DP}^\star}$ functor, we will omit it from here. □

Now, let's focus on the S functor. Ideally we'd like it to have the same properties as the functors in Chapter 9: monoidality, as well as preserving the lattical structure and traces of $\mathbf{DP}^\star$. The following three lemmas show that S indeed possesses these properties.

Lemma 10.9. *The S functor is a strong monoidal functor (Definition 2.31).*

Proof. We will only show that given a design problems $d_1 : F_1 \twoheadrightarrow R_1$ and $d_2 : F_2 \twoheadrightarrow R_2$ it holds that:

$$S(d_1) \otimes_{\mathbf{APos}} S(d_2) = S(d_1 \otimes_{\mathbf{DP}^\star} d_2). \tag{10.5}$$

The associativity and unitality properties are straight-forward to show so will omit their proofs.

First, we have that:

$$S(d_1) = \left[f_1 \xmapsto{F_1 \to \mathcal{A}R_1} \operatorname*{Min}_{\leq_{R_1}} \left\{ r_1 \in R_1 \mid d_1(f_1, r_1) = \mathtt{T} \right\} \right],$$

and

$$S(d_2) = \left[f_2 \xmapsto{F_2 \to \mathcal{A}R_2} \operatorname*{Min}_{\leq_{R_2}} \left\{ r_2 \in R_2 \mid d_2(f_2, r_2) = \mathtt{T} \right\} \right].$$

Then, following applying the tensor product in **APos** we obtain:

$$S(d_1) \otimes_{\mathbf{APos}} S(d_2) = \left[\langle f_1, f_2 \rangle \mapsto \operatorname*{Min}_{\leq_{R_1}} \left\{ r_1 \in R_1 \mid d_1(f_1, r_1) \right\} \times \operatorname*{Min}_{\leq_{R_2}} \left\{ r_2 \in R_2 \mid d_2(f_2, r_2) \right\} \right].$$

Now, let's take a look at the right-hand side of Equation (10.5):

$$\begin{aligned} S(d_1 \otimes_{\mathbf{DP}^\star} d_2) &= \left[\langle f_1, f_2 \rangle \mapsto \operatorname*{Min}_{\leq_{(R_1 \times R_2)}} \left\{ \langle r_1, r_2 \rangle \mid (d_1 \otimes d_2)(\langle f_1, f_2 \rangle, \langle r_1, r_2 \rangle) \right\} \right] \\ &= \left[\langle f_1, f_2 \rangle \mapsto \operatorname*{Min}_{\leq_{(R_1 \times R_2)}} \left\{ \langle r_1, r_2 \rangle \mid d_1(f_1, r_1) \wedge d_2(f_2, r_2) \right\} \right] \\ &= \left[\langle f_1, f_2 \rangle \mapsto \operatorname*{Min}_{\leq_{R_1}} \left\{ r_1 \in R_1 \mid d_1(f_1, r_1) \right\} \times \operatorname*{Min}_{\leq_{R_2}} \left\{ r_2 \in R_2 \mid d_2(f_2, r_2) \right\} \right]. \end{aligned}$$

Hence Equation (10.5) indeed holds true. □

Lemma 10.10. *The S functor preserves the bounded lattice structure of the hom-sets of $\mathbf{DP}^\star$. In other words, for any $A, B \in \mathsf{Ob}(\mathbf{DP}^\star)$ and any $d_1, d_2 \in \mathrm{Hom}_{\mathbf{DP}^\star}(A, B)$:*

i. *if $d_1 \leq_{\mathbf{DP}^\star(A,B)} d_2$, then $S(d_1) \leq_{\mathbf{APos}(A,B)} S(d_2)$;*

ii. *$S(d_1 \wedge d_2) = S(d_1) \sqcup S(d_2)$;*

iii. *$S(d_1 \vee d_2) = S(d_1) \sqcap S(d_2)$;*

iv. *$S(\top_{\mathbf{DP}^\star(A,B)}) = \top_{\mathbf{APos}(A,B)}$;*

v. *$S(\bot_{\mathbf{DP}^\star(A,B)}) = \bot_{\mathbf{APos}(A,B)}$.*

Proof. Order preserving follows directly from the way the order is defined on the hom-sets of the two categories. On $\mathbf{DP}^\star$ it is:

$$d_1 \leq_{\mathbf{DP}^\star(A,B)} d_2 \iff \big(d_2(a,b) \implies d_1(a,b),\ \forall a \in A, b \in B\big).$$

And on **APos**:

$$S(d_1) \leq_{\mathbf{APos}(A,B)} S(d_2) \iff \big(\uparrow S(d_1)(a) \supseteq \uparrow S(d_2)(a),\ \forall a \in A\big).$$

The right-hand sides of both are equivalent because of the monotonicity of the design problems. Now, let's take a look at the joins:

$$\begin{aligned} S(d_1 \wedge d_2) &= S([\langle a,b\rangle \mapsto d_1(a,b) \wedge d_2(a,b)]) \\ &= \left[a \mapsto \underset{\leq_B}{\mathrm{Min}}\{b \in B \mid d_1(a,b) \wedge d_2(a,b)\}\right] \\ &= \left[a \mapsto \underset{\leq_B}{\mathrm{Min}}(\{b \in B \mid d_1(a,b)\} \cap \{b \in B \mid d_2(a,b)\})\right] \\ &= \left[a \mapsto \underset{\leq_B}{\mathrm{Min}}\left(\uparrow \underset{\leq_B}{\mathrm{Min}}\{b \in B \mid d_1(a,b)\} \cap \uparrow \underset{\leq_B}{\mathrm{Min}}\{b \in B \mid d_2(a,b)\}\right)\right] \\ &= \left[a \mapsto \underset{\leq_B}{\mathrm{Min}}\{b_1 \sqcup b_2 \mid b_1 \in \{b \in B \mid d_1(a,b)\}, b_2 \in \{b \in B \mid d_2(a,b)\}\}\right] \\ &= S(d_1) \sqcup S(d_2), \end{aligned}$$

where we used the fact that d_1 and d_2 are well-behaved and Lemma 8.23. The meets go similarly:

$$\begin{aligned}
S(d_1 \vee d_2) &= S\left([\langle a, b\rangle \mapsto d_1(a,b) \vee d_2(a,b)]\right) \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B} \{b \in B \mid d_1(a,b) \vee d_2(a,b)\}\right] \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B} (\{b \in B \mid d_1(a,b)\} \cup \{b \in B \mid d_2(a,b)\})\right] \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B} \left(\operatorname*{Min}_{\leq_B} \{b \in B \mid d_1(a,b)\} \cup \operatorname*{Min}_{\leq_B} \{b \in B \mid d_2(a,b)\}\right)\right] \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B} (S(d_1)(a) \cup S(d_2)(a))\right] \\
&= S(d_1) \sqcap S(d_2).
\end{aligned}$$

The top elements are mapped as:

$$\begin{aligned}
S\left(\top_{\mathbf{DP}^\star(A,B)}\right) &= S\left(\left[\langle a, b\rangle \xmapsto{A^{\mathrm{op}} \times B \to \mathtt{Bool}} \mathtt{F}\right]\right) \\
&= \left[a \xmapsto{A \to \mathcal{A}B} \varnothing\right] \\
&= \top_{\mathbf{APos}(A,B)}.
\end{aligned}$$

And the bottom elements are mapped as:

$$\begin{aligned}
S\left(\bot_{\mathbf{DP}^\star(A,B)}\right) &= S\left(\left[\langle a, b\rangle \xmapsto{A^{\mathrm{op}} \times B \to \mathtt{Bool}} \mathtt{T}\right]\right) \\
&= \left[a \xmapsto{A \to \mathcal{A}B} \bot_B\right] \\
&= \bot_{\mathbf{APos}(A,B)}.
\end{aligned}$$

Thus, S indeed preserves the bounded lattice structure of the hom-sets of $\mathbf{DP}^\star$. □

Lemma 10.11. *The S functor preserves traces. In other words:*

$$S\left(\mathrm{Tr}_{\mathbf{DP}^\star}(d)\right) = \mathrm{Tr}_{\mathbf{APos}}\left(S(d)\right),$$

for all $d \in \mathrm{Hom}_{\mathbf{DP}^\star}(A \times X, B \times X)$ and for all $A, B, X \in \mathsf{Ob}(\mathbf{DP}^\star)$.

Proof. The left-hand side evaluates to:

$$\begin{aligned}
S\left(\mathrm{Tr}_{\mathbf{DP}^\star}(d)\right) &= S\left(\left[\langle a, b\rangle \mapsto \bigvee_{x \in X} d\left(\langle a, x\rangle, \langle b, x\rangle\right)\right]\right) \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B} \left\{b \in B \mid \bigvee_{x \in X} d\left(\langle a, x\rangle, \langle b, x\rangle\right)\right\}\right].
\end{aligned}$$

The right-hand side evaluates to:

$$\begin{aligned}
&\mathrm{Tr}_{\mathbf{APos}}(S(d)) \\
&= \mathrm{Tr}_{\mathbf{APos}}\left(\left[\langle a,b\rangle \mapsto \operatorname*{Min}_{\leq_{(B\times X)}} \{\langle b,x\rangle \in B\times X \mid d\left(\langle a,x\rangle,\langle b,x\rangle\right)=\mathtt{T}\}\right]\right) \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B}\left\{b\in B \mid \bigvee_{x\in X}\left(\langle b,x\rangle \in \uparrow \operatorname*{Min}_{\leq_{(B\times X)}}\{\langle b,x\rangle \mid d\left(\langle a,x\rangle,\langle b,x\rangle\right)=\mathtt{T}\}\right)\right\}\right] \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B}\left\{b\in B \mid b \in \uparrow \operatorname*{Min}_{\leq_B}\left\{b\in B \mid \bigvee_{x\in X} d(\langle a,x\rangle,\langle b,x\rangle)=\mathtt{T}\right\}\right\}\right] \\
&= \left[a \mapsto \operatorname*{Min}_{\leq_B}\left\{b\in B \mid \bigvee_{x\in X} d\left(\langle a,x\rangle,\langle b,x\rangle\right)\right\}\right].
\end{aligned}$$

Hence, the S functor indeed preserves traces. □

We just concluded the crusade for compositional properties that started in Chapter 9. The result is that all the functors that appear as problems and procedures in Equation (10.4) preserve tensor products, orders, joins, meets, tops, bottoms, and traces. This is summarized in Table 10.2.

The importance of this result can be summarized as:

> *The answer of a composition of design problems is the composition of the answers of the problems.*

Thus one can choose whether to first compute the composite of two problems (and we put the trace here too, even though it's unitary) or to first compute the problems independently and then compose the answers. Which one is preferable might depend on which one results in less required resources.

One can expect, though, that co-design is not the only problem-solving framework that exhibits this property. Essentially any other system where where problem statements and answers form monoidal locally lattical traced categories and problems and procedures are strong monoidal functors that preserve traces and the bounded lattical structure of hom-sets would exhibit the exact same property. Therefore, it makes sense to consider a compositional computational system that contains them.

We will refer to this system as $\mathbf{LAnd}^\dagger$ and will formally be defined as:

$$\mathbf{LAnd}^\dagger := \mathbf{Laputa}\left(\mathbf{Cat}^\dagger, B, B, \Psi^{\mathbf{And}}\right), \tag{10.6}$$

with the $\mathbf{Cat}^\dagger$ category being the subcategory of $\mathbf{Cat}$ (Definition 2.37) with objects being monoidal locally lattical traced categories and morphisms being strong monoidal functors that preserve traces and the bounded lattical structure of hom-sets.

Remark 10.12. The **DP** category and design problems have even more structure than we discussed here. It is of especial interest that **DP** is a also compact closed category (Censi et al., 2020). In essence, this means that co-design problems can act as functionalities and resources of other design problems. This self-referentiality points to the computational richness of co-design. By using the exact same approach as above, we can further extend the example in this section, and the $\mathbf{LAnd}^\dagger$ compositional computational system to work with functors which preserve compact closed structure.

10.4 Answers of design problems can also have extra structure

Solving design problems is a computational process. And as every other computational process, it takes up some resources such as CPUs, energy and time. Earlier in this thesis we defined the procedures in **Lagado** in a way that accounts for the usage of these computational resources. Then, we showed that **Lagado** is one instance of a compositional computational theory (Section 6.3). Just now, we showed that the extra structure of the co-design problems can be represented as the $\mathbf{LAnd}^\dagger$ category. This leaves us with an interesting question: Can we have the best of both worlds? Compositional structures *and* computational resource accountability? The answer thankfully is "yes, of course!".

Recall our definition of the $\mathbf{LAnd}^\dagger$ system. We will modify it a bit to get to our desired new compositional computational system. The resources part affects only the kind of the procedures, hence we can leave the subcategory of **SemiCat** the same ($\mathbf{Cat}^\dagger$) and the rig for the problems also the same (B). We already saw how resources can be represented as a rig in the definition of $M^{\mathbf{Lagado}}$ in Section 6.3. Given a monoidal poset of resources R, we defined it as:

$$M^{\mathbf{Lagado}} := \langle \{\mathtt{F}\} \cup \mathsf{U}R, +_M, \mathtt{F}, \times_M, R \rangle \,.$$

We can use the exact same structure here as well.

Finally, we also need to adjust the solution judgement map. We can define it similarly to $\Psi^{\mathbf{Lagado}}$:

$$\begin{aligned}
\Psi^{\mathbf{LagadoLAnd}}_{\mathbf{T},\mathbf{A}} \colon \left(\mathbf{T} \xrightarrow{B} \mathbf{A}\right) \times \left(\mathbf{T} \xrightarrow{M^{\mathbf{Lagado}}} \mathbf{A}\right) &\to \mathtt{Bool}, \\
\langle \Pi, \Delta \rangle &\mapsto \begin{cases} \mathtt{T} & \text{if } \Delta(t,a) \neq \mathtt{F} \implies \Pi(t, \Delta(t,a)) = \mathtt{t}, \\ & \forall t \in \mathsf{Ob}(T), \forall a \in \mathsf{Ob}(A), \\ \mathtt{F} & \text{otherwise.} \end{cases}
\end{aligned}$$

Thus, we now define the new compositional computational system that has both the compositional properties of $\mathbf{LAnd}^\dagger$ and the resource-awareness of **Lagado**:

$$\mathbf{LagadoLAnd}^\dagger \coloneqq \mathbf{Laputa}\left(\mathbf{Cat}^\dagger, B, M^{\mathbf{Lagado}}, \Psi^{\mathbf{LagadoLAnd}}\right). \tag{10.7}$$

If we didn't care about the compositional structure, but still wanted the resource awareness, we could use the **Cat** category as a basis:

$$\mathbf{LagadoLAnd} \coloneqq \mathbf{Laputa}\left(\mathbf{Cat}, B, M^{\mathbf{Lagado}}, \Psi^{\mathbf{LagadoLAnd}}\right). \tag{10.8}$$

Of course, the variety of structures and properties is by far not limited by the above examples. One can also consider representing approximate solutions, uncertainty in the problems (Censi, 2017), or any other structures and properties they might be interested in.

Chapter 11

Conclusion

Throughout this thesis, we poked and prodded problem-solving from several points of view and with various properties, applications, and examples in mind. We showed that one can indeed formalize the problem of problem-solving in rich and powerful structures. We even saw that in some cases the resulting compositional computational systems contain themselves, hence can be used to reason about their own problem-solving.

The current work can serve as a guide to the critical thinker on how to formally reason about concepts such as "problems" and "solutions". We hope that we not only present a new perspective, but also a more efficient and communicable way of thinking. Our ultimate desire while working on this thesis was to help the reader reach a state where they can solve problems that they couldn't solve before or solve problems much faster and with fewer resources and headaches than before.

However, apart from the philosophical aspect of problem-solving, there are also very pragmatic real-world opportunities that stem from this work. By formally defining problems and procedures as separate entities, related solely by a solution judgment map, we effectively decouple the acts of stating a problem and solving it. This can be put into practice by creating a software system for computation where the user only needs to know which problem they want to solve and how to describe its specific instance (the statement). Solving this problem instance in the best way possible would be then left for the system to do. This is similar to declarative programming where a program describes what computation should be performed but not how. The backend of such a system would only have to find all the paths from this problem to identity problems in some compositional computational system, and to then select the one that results in the lowest resource consumption for that specific problem instance. Some paths would be optimal for some instances (e.g. small ones), others would be optimal for others (e.g. large ones). However, the optimal path need not be only determined by the one that can most efficiently solve the problem instance from scratch. We can also use cached results from previous (and seemingly unrelated) problems. Sometimes solving a problem via a less efficient path can be more efficient if it gets reduced to a problem that has already been solved and whose answer is cached and can be readily reused. One can, of course, imagine that optimizing who, where, and how performs the actual computation in such a system can also be automatically determined. This is too, after all, a problem to be solved.

Apart from building a software implementation of compositional computational systems, there are also a number of theoretical questions that are still unaddressed. First, as of now, we do not have concrete algorithms that select the best execution path for a given problem. Introducing the option to reuse previously cached results in the path selection would also be a critical point for utilizing the full potential of this framework. Developing such algorithms would be necessary for the development of the software system described above.

We also did not fully address the question about the nature of the compositional computational systems which are rich in structure, such as **LagadoLAnd**. One of the key observations there was that we can choose whether first to solve two subproblems and then to compose the solutions or to first compose the problems and then solve the result. One can imagine how this question can be represented as two different paths in another compositional computational system. Hence we can be talking about various ways of reducing problems with such a rich structure. This is still an area that needs further research.

Another important observation that was merely restricted to a remark in the current work is that the compositional computational systems can also form an order (Remark 6.18). On one hand, this allows us to speak of some systems being "more powerful" than other systems. Furthermore, if we also have joins in this order, then we can also combine problems and procedures from different systems as they would both be represented in their join. And if there are joins, then that would raise the natural question of whether there's a terminal object, a terminal compositional computational system that can represent any problem, procedure, and solution.

Our choice of basing **Lagado** on type theoretical concepts also leaves a few open research directions. Perhaps, the most interesting being whether we can not only require that procedures are backed by a computational process but we actually formally provide it. And a great candidate for describing computational processes in this setting is Lambda calculus as it fits nicely with the types we already use in the **Prob** category. Lambda calculus also interacts well with the (semi)categorical structure that we want our statements and answers to have. The connections between Lambda calculus and category theory are studied by Crole (1994). Investigating the intersection between Lambda calculus, category theory, and the theory of compositional computational systems is hence of further interest.

However, if we consider procedures with Lambda terms as descriptors of the computational process, then such a system would also bring along many fundamental results about computability. For instance, such a system can be truly restricted only to computable procedures thanks to the Church-Turing thesis (Turing, 1937). Further studying the connections of the current theory with these fundamental results from computability theory can then connect the present work with the key findings of mathematical logic and the theory of computation.

Finally, we need to disclose that the claim that some compositional computational systems contain themselves and hence can be used to reason about their own problem-solving can, in fact, suffer from some fundamental limitations. Such self-referential statements and claims might remind the reader about the proof of Gödel's incompleteness results (Gödel, 1931). Hence, a critical future task should be to study whether similar limitations apply here and if they do, then how would they connect to this fundamental result from the field of mathematical logic. We would like to thank Jacopo Tani for raising our attention to this potential connection.

While the theory presented in this work can stand by itself and presents virtue both in philosophical and pragmatic ways, there are still several key theoretical questions that we believe deserve further investigation. These questions can connect the ideas behind compositional computational systems with some of the most fundamental results in mathematics. Studying such connections holds the potential of not only validating the work we presented here but also to bridge the intuitive notions of problem-solving that we based this thesis on with the key results and questions of meta-mathematics.

Bibliography

Bellman, Richard (1958). “On a Routing Problem”. In: *Quarterly of Applied Mathematics* 16.1, pp. 87–90.

Censi, Andrea (2016). “A Mathematical Theory of Co-Design”. In: arXiv: `1512.08055 [cs.LO]`.

Censi, Andrea (2017). “Uncertainty in Monotone Codesign Problems”. In: *IEEE Robotics and Automation Letters* 2.3, pp. 1556–1563.

Censi, Andrea, David I. Spivak, Joshua Tan, and Gioele Zardini (2020). “Mathematical Foundations of Engineering Co-Design”. In preparation.

Coppersmith, Don and Shmuel Winograd (1990). “Matrix Multiplication via Arithmetic Progressions”. In: *Journal of Symbolic Computation* 9.3, pp. 251–280.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009). *Introduction to Algorithms*. 3rd ed. MIT Press.

Crole, Roy L. (1994). *Categories for Types*. 1st ed. Cambridge University Press.

Curry, Haskell B. (1952). “On the Definition of Substitution, Replacement and Allied Notions in a Abstract Formal System”. In: *Revue Philosophique de Louvain* 50.26, pp. 251–269.

Davey, Brian A and Hilary A Priestley (2002). *Introduction to Lattices and Order*. Cambridge University Press.

Dijkstra, Edsger W (1959). “A Note on Two Problems in Connexion with Graphs”. In: *Numerische mathematik* 1.1, pp. 269–271.

Eilenberg, Samuel and Saunders MacLane (1945). “General Theory of Natural Equivalences”. In: *Transactions of the American Mathematical Society* 58.2, pp. 231–294.

Fong, Brendan and David I. Spivak (2019). *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*. Cambridge University Press.

Ford, Lester R. (1956). *Network Flow Theory*. RAND Corporation, Santa Monica.

Gierz, Gerhard, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D Lawson, Michael Mislove, and Dana S Scott (2003). *Continuous Lattices and Domains*. Cambridge University Press.

Gödel, Kurt (1931). "Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme I". In: *Monatshefte für mathematik und physik* 38.1, pp. 173–198.

Hoeffding, Wassily (1963). "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301, pp. 13–30.

Jonathan Swift (1726). *Gulliver's Travels*.

Jungnickel, Dieter (2013). *Graphs, Networks and Algorithms*. Algorithms and Computation in Mathematics. Springer Berlin Heidelberg.

Lane, Saunders M. (1998). *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York.

Mehlhorn, Kurt and Peter Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox*. Springer Science & Business Media.

Moore, Edward F (1959). "The Shortest Path Through a Maze". In: Proceedings of the International Symposium on Switching Theory, pp. 285–292.

Pierce, Benjamin C. (2002). *Types and Programming Languages*. MIT Press.

Robinson, Sara (2005). "Toward an Optimal Algorithm for Matrix Multiplication". In: *SIAM News* 38.9, p. 3.

Romani, Francesco (1980). "Shortest-path Problem is not Harder than Matrix Multiplication". In: *Information Processing Letters* 11.3, pp. 134–136.

Shimbel, Alfonso (1954). "Structure in Communication Nets". In: Proceedings of the Symposium on Information Networks. Polytechnic Institute of Brooklyn, pp. 119–203.

Strassen, Volker (1969). "Gaussian Elimination is Not Optimal". In: *Numerische Mathematik* 13.4, pp. 354–356.

The Univalent Foundations Program (2013). *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study.

Turing, Alan M (1937). "Computability and λ-definability". In: *The Journal of Symbolic Logic* 2.4, pp. 153–163.

Watanabe, Osamu (1981). "A Fast Algorithm for Finding All Shortest Paths". In: *Information Processing Letters* 13.1, pp. 1–3.

Weiss, Eric A. (1985). "Jonathan Swift's Computing Invention". In: *Annals of the History of Computing* 7.2, pp. 164–165.