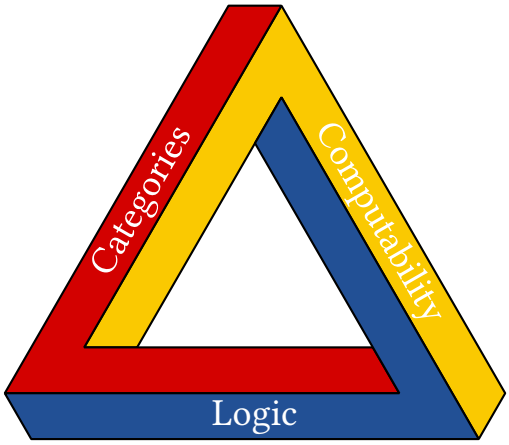
Categorical realizability

by

Tom de Jong

Lecture notes and exercises for the
Midlands Graduate School (MGS)

8–12 April 2024, Leicester, UK



School of Computer Science
University of Nottingham

January–March 2024

To Jaap van Oosten,
in recognition of his contributions to the teaching of mathematical logic.

Abstract

Realizability, as invented by Kleene, is a technique for elucidating the computational content of mathematical proofs. In this course we study realizability from a categorical perspective. Starting from an abstract and general model of computation known as a partial combinatory algebra (pca), we construct the category of assemblies over it. Intuitively, an assembly is a set together with computability data and an assembly map is a function of sets that is computable. Here, the notion of computability is prescribed by the pca. Through the framework of categorical logic, the assemblies give rise to the realizability interpretation of logic, which we spell out in detail.

The central theme of this course is the interplay between category theory, logic and computability theory. While some familiarity with basic category theory (e.g. (co)limits and adjunctions) is required for some parts of the notes, the course hopefully offers plenty to those unfamiliar with category theory.

Acknowledgements

It is my pleasure to express my sincere thanks to Jaap van Oosten—to whom I have dedicated these notes—for teaching an excellent course on category theory [vOos16] and introducing me to realizability when I was a master's student in mathematics at Utrecht University back in 2015–2018.

I thank Mark Williams for fixing some typos. For the illustration on the title page I adapted `tikz` code from *AboAmmar*'s answer on TEX StackExchange [Abo14].

Contents

Chapter 1

Introduction

Realizability originates with Kleene [Kle45] in the context of proof theory and sought to make a precise connection between intuitionistic (constructive) mathematics and computability theory. This led to an effective interpretation of intuitionistic number theory using computability theory, somewhat reminiscent to—but predating!—the Curry–Howard correspondence.

In the 1980's, Hyland introduced the *effective topos*: a category whose logic is governed by Kleene's realizability. Intuitively, the effective topos presents us with an alternative world of mathematics where—unlike in the category of sets and functions—"everything is computable". Actually, the effective topos is an instance of a general class of categories known as *realizability toposes* as originally developed by Hyland, Johnstone and Pitts [HJP80; Pit81].

Every realizability topos is parametrized by an abstract model of computation, known as a *partial combinatory algebra (pca)*. Classical computability theory based on partial Turing computable functions on natural numbers gives a pca known as *Kleene's first model* and the resulting realizability topos is Hyland's effective topos. While Hyland's topos is perhaps the best understand example, many other choices of pcas are possible with interesting realizability toposes as a result [vOos08].

A beautiful aspect of Hyland's discovery is that it enables us to study the interplay between category theory, logic and computability theory. A typical application of realizability categories is to provide semantics to (polymorphic) type theories like System F (which has no direct set-theoretic semantics [Rey84]); see also Section 1.4. Realizability has also found practical application in the form of extracting programs from mathematical proofs, see e.g. [BMSS11].

1.1 Aims

We hope that these notes provide a self-contained and accessible introduction to the categorical aspects of realizability for graduate students in theoretical computer science. More generally, we hope the reader will appreciate these notes as a testament of the

deep connections between category theory, logic and computability theory.

The notion of a realizability topos is fairly involved and for this reason we focus on the simpler categories of *assemblies* instead (although we include a brief epilogue on realizability toposes). While the category of assemblies lacks some features of a topos it provides more than enough structure for our present purposes. Besides, one should arguably first have a good grasp on the assemblies before studying realizability toposes.

Outline of these notes

- Chapter 2 introduces partial combinatory algebra (pcas) as abstract models of computation and illustrate them with various examples. Some familiarity with basic computability theory and topology is helpful, but certainly not required.
- Chapter 3 describes the category of assemblies and assembly maps over a fixed but arbitrary pca. Intuitively, an assembly is a set together with computability data and an assembly map is a function of sets that is computable. Here, the notion of computability is prescribed by the pca.

  We explore the categorical structure of this category and some familiarity with basic category theory—say (co)limits and adjunctions—is necessary in most places, although I am hopeful that those unfamiliar with category theory can still benefit from this chapter and are perhaps inspired to learn some category theory.
- Finally, in Chapter 4, we turn to the logical aspects of the category of assemblies. We spell out the realizability interpretation of first order logic that the assemblies give rise to. We illustrate the connections between category theory, logic and computability theory by studying certain *realizability predicates* from these three perspectives. For example, the semidecidable predicates can be characterized (1) categorically, as pullbacks of a certain two-element assembly; (2) computably, as computably enumerable subsets; and (3) logically, as those predicates which are presented by a binary sequence. Finally, we exploit these connections by describing a simple result from *synthetic computability theory*.

1.2 Exercises

The exercises are interspersed in the text, but each chapter ends with a list of its exercises for reference. There are 33 exercises in total.

1.3 References

In preparing these notes I have mainly used the standard textbook [vOos08] by van Oosten, as well as Bauer's excellent lecture notes [Bau23]. In a few places, e.g. Proposition 3.23, I have also consulted Streicher's notes [Str18]. The presentation of logic in the category of assemblies using realizability predicates owes a lot to Bauer's treatment, although I turned to [vOos08, Section 3.2.7] for Exercises 4.29 to 4.32, and included some additions in the form of Sections 4.2.3 and 4.4. In notation and terminology I have stayed close to Bauer's notes as I admire its readability.

1.4 Further reading

Natural candidates for further reading are the aforementioned notes by Bauer [Bau23] and Streicher [Str18], as well as the standard textbook by van Oosten [vOos08].

The categorical semantics of (polymorphic) type theories using realizability is treated in a variety of works, such as Reus's tutorial paper [Reu99], Streicher's notes [Str18], Amadio and Curien's textbook [AC98] and Jacobs's textbook [Jac99]; see also the references listed on [vOos08, p. 193].

Those looking to learn more about general models of (higher-order) computability can consult Longley and Normann's comprehensive textbook [LN15].

To those interested in the history of realizability we recommend Troelstra's proof-theoretic survey [Tro98] and van Oosten's essay [vOos02] for its categorical aspects.

If you are interested in the formalization of mathematics, then you should look at Chhabra's ongoing *Cubical Agda* development [Chh23][1].

[1] With the caveat that the combinatory algebras are assumed to be total—at least for the moment.

Chapter 2

Models of computation: partial combinatory algebras

The starting ingredient in categorical realizability is a general model of computation known as a *partial combinatory algebra*, or *pca* for short. The desire for a general model of computation is motivated by the many possible interesting examples. Moreover, in the next chapter, we construct the *category of assemblies* over an arbitrary pca. The construction itself works generally and is insensitive to any particular choice of pca. However, the choice of pca is reflected in the logical principles that the resulting category validates. Thinking of categories of assemblies as worlds of computable mathematics, we can thus build different worlds by varying our notion of pca.

Having said all this, in these notes we have chosen to work with the *untyped* notion of a pca. The *typed* notion, as developed by Longley[1] in his PhD thesis [Lon95], is more general and moreover has the advantage that it can capture (idealized) typed functional programming languages (such as PCF [Plo77; dJon23]). We only treat untyped pcas for simplicity and refer the interested reader to Bauer's more comprehensive notes [Bau23].

Definition 2.1 (Partial combinatory algebra (pca), **K**, **S**). A **partial combinatory algebra** (**pca**) is a set $\mathcal{A}$ together with a *partial* operation $\mathcal{A} \times \mathcal{A} \rightharpoonup \mathcal{A}$, denoted by juxtaposition, $(\mathtt{a}, \mathtt{b}) \mapsto \mathtt{a}\,\mathtt{b}$, such that there exist elements **K** and **S** satisfying:

(i) $(\mathbf{K}\,\mathtt{a})\,\mathtt{b} = \mathtt{a}$ for all elements $\mathtt{a}, \mathtt{b} \in \mathcal{A}$,

(ii) $((\mathbf{S}\,\mathtt{f})\,\mathtt{g})$ is defined for all elements $\mathtt{f}, \mathtt{g} \in \mathcal{A}$, and

(iii) $((\mathbf{S}\,\mathtt{f})\,\mathtt{g})\,\mathtt{a} \simeq (\mathtt{f}\,\mathtt{a})(\mathtt{g}\,\mathtt{a})$ for all elements $\mathtt{f}, \mathtt{g}, \mathtt{a} \in \mathcal{A}$.

The symbol $\simeq$ is *Kleene equality* and means: either both sides are undefined, or both are defined and are equal elements of $\mathcal{A}$. In particular, in (i), the operation $\mathbf{K}\,\mathtt{a}$ must be defined for all elements $\mathtt{a} \in \mathcal{A}$.

[1] Fun fact: John Longley is also the author of the recent *Castles in the Air*—an introduction to the world of mathematical logic cast in the form of a fantasy novel—as well as a semi-professional pianist.

We think of the elements of a pca as codes for programs that can also act as input. Accordingly, we pronounce `a b` as "`a` applied to `b`" and think of this as: apply the program with code `a` to input `b`.

In this light, the element **K**, which we call the *k-combinator*, acts as a parameterized constant program: it takes an input `a` and then always outputs `a` on any input `b`.

The element **S**, which we call the *s-combinator*[2], acts as parameterized application: it takes two codes for programs `f` and `g` and an input `a` and then applies `f a` to `g a`.

Notation 2.2. We will economize on parentheses and write `a b c` for `(a b) c`. We will always use the `teletype font` for arbitrary elements of a pca to reinforce the idea that these should be thought of as (codes of) programs. For fixed programming constructs, we will use this **bold font** instead.

The point of the k- and s-combinators is that they provide us with a (very minimal) programming interface which we will explore in Section 2.2. We prefer to give two examples first to strengthen our intuition for pcas.

2.1 Basic examples of pcas

To help build some intuition for pcas, we now consider three basic examples. The first example is a triviality, but we include it here because the category of assemblies (see Chapter 3) over it is equivalent to the familiar category of sets.

Example 2.3 (The trivial pca)**.** The **trivial pca** is the singleton set $\{\star\}$ with application map $(\star, \star) \mapsto \star$. Of course, $\mathbf{K} := \star$ and $\mathbf{S} := \star$.

Example 2.4 (Untyped λ-calculus as a pca, Λ)**.** Write Λ for the closed terms of the untyped λ-calculus quotiented by the equivalence relation generated by β-reduction. (E.g., for a closed λ-term t, we identify $(\lambda x.x)t$ and t.) With λ-calculus application the set Λ forms a pca with **K** and **S** given by the equivalence classes of $\lambda xy.x$ and $\lambda xyz.(xz)(yz)$, respectively.

The application function of Λ is actually total, i.e. it is defined on any two inputs. An example of a pca with a genuine partial application—which is also the prime example of a pca—is *Kleene's first model* [Kle45]:

Example 2.5 (Kleene's first model, $\mathcal{K}_1$)**.** We define a partial application function on the set of natural numbers $\mathbb{N}$: for natural numbers n and m we take $n\,m$ to be $\varphi_n(m)$, where φ_- denotes a Turing computable enumeration of the Turing computable partial functions on the natural numbers.
The existence of the k- and s-combinators follows from Kleene's S^m_n-theorem in computability theory, see e.g. [Bau23, Section 2.5.1 and Theorem 2.1.5] for details.
We write $\mathcal{K}_1$ for this partial combinatory algebra and return to it in Chapter 4.

[2]The letters *k* and *s* come from Moses Schönfinkel's combinatory logic [Sch24]. They respectively come from the German words *Konstanzfunktion* (constant function) and *Verschmelzungfunktion* (merge function). Of course, *Verschmelzungsfunktion* starts with a *v*, but Schönfinkel had to avoid confusion as there was also a swap-arguments combinator called the *Vertauschungsfunktion*.

2.2 Basic programming in pcas

The k- and s-combinators of a pca provide an interface for writing basic programs. For example, if we put[3] $\mathbf{I} := \mathbf{S}\mathbf{K}\mathbf{K}$ then $\mathbf{I}$ acts as the identity combinator: $\mathbf{I}\,\mathrm{a} = \mathbf{S}\mathbf{K}\mathbf{K}\,\mathrm{a} = \mathbf{K}\,\mathrm{a}(\mathbf{K}\,\mathrm{a}) = \mathrm{a}$ for any element a of our pca.

While theoretically possible, it is very inconvenient to program everything directly in terms of $\mathbf{K}$ and $\mathbf{S}$. Therefore, we will define something resembling λ-abstraction for our pca, allowing us to write the clearer $\mathbf{I} := \langle x \rangle.\, x$ instead.

Definition 2.6 (Terms)**.** We fix a countably infinite set of variables, typically denoted by $x, y, z, u, v, w, x_0, x_1, \ldots$, and inductively define the set of **terms** over a pca $\mathcal{A}$:

(i) a variable is a term,

(ii) an element of $\mathcal{A}$ is a term,

(iii) given two terms s and t, we may form a new term denoted by $s\,t$.

A term without variables is called **closed**.

Notation 2.7. If t is a term, then we write $t[\mathrm{a}_1/x_1, \ldots, \mathrm{a}_n/x_n]$ for the *closed* term obtained by substituting the element $\mathrm{a}_i \in \mathcal{A}$ for the variable x_i (which may or may not occur in t).

Definition 2.8 (Defined terms)**.** A closed term is **defined** if, when we interpret $s\,t$ as s applied to t in $\mathcal{A}$, all these applications are defined.
We extend this to terms with variables: such a term t is **defined** if for all possible substitutions of all variables in t by elements of $\mathcal{A}$, the resulting closed term obtained via substitution is defined.

For example, the terms $\mathbf{K}\,\mathbf{S}\,\mathbf{I}$ and $\mathbf{S}\,x\,y$ are defined.

Notation 2.9. We extend Kleene equality from closed terms to all terms over $\mathcal{A}$: given two terms s and t whose variables are among $x_1, \ldots, x_n$, we write $t_1 \simeq t_2$ if for all $\mathrm{a}_1, \ldots, \mathrm{a}_n \in \mathcal{A}$, we have

$$t_1[\mathrm{a}_1/x_1, \ldots, \mathrm{a}_n/x_n] \simeq t_2[\mathrm{a}_1/x_1, \ldots, \mathrm{a}_n/x_n]$$

as closed terms.

We now define something resembling λ-abstraction for pcas.

Definition 2.10 ("λ-abstraction" in a pca, $\langle x \rangle.\, t$)**.** For a variable x and a term t, we define a new term, denoted by $\langle x \rangle.\, t$, by recursion on terms:

- $\langle x \rangle.\, x := \mathbf{I} = \mathbf{S}\mathbf{K}\mathbf{K}$,
- $\langle x \rangle.\, y := \mathbf{K}\,y$ if y is a variable different from x,
- $\langle x \rangle.\, \mathrm{a} := \mathbf{K}\,\mathrm{a}$ for $\mathrm{a} \in \mathcal{A}$,
- $\langle x \rangle.\, (t_1\,t_2) := \mathbf{S}(\langle x \rangle.\, t_1)\,(\langle x \rangle.\, t_2)$.

[3] Note that this indeed defines an element of $\mathcal{A}$ by Definition 2.1(ii).

Exercise 2.11 (Combinatory completeness)**.** Prove that for every variable x and term t, the following properties of $\langle x \rangle . t$ hold:

(i) The variables of $\langle x \rangle . t$ are exactly those of t minus x.

(ii) The term $\langle x \rangle . t$ is defined.

(iii) For all $\mathrm{a} \in \mathcal{A}$, we have $(\langle x \rangle . t)\,\mathrm{a} \simeq t[\mathrm{a}/x]$.

Notation 2.12. We write $\langle xy \rangle . t$ for the term $\langle x \rangle . (\langle y \rangle . t)$ and similarly for more variables.

Suppose we have a term t featuring only the variables x, y and z, e.g., $t := z\,\mathbf{K}\,x(\mathbf{K}\,y)$. We think of t as a *partial* function $\mathcal{A} \times \mathcal{A} \times \mathcal{A} \rightharpoonup \mathcal{A}$, taking three inputs a, b and c that we may substitute in t for x, y and z, respectively, and that results in the term $\mathrm{c}\,\mathbf{K}\,\mathrm{a}(\mathbf{K}\,\mathrm{b})$ which has no variables. Combinatory completeness says that terms can be internalized in a pca. Indeed, we can represent t in $\mathcal{A}$ as $\langle xyz \rangle . t$. Note that this is indeed an element of $\mathcal{A}$ by virtue of Exercise 2.11(ii) and the fact that $\langle xyz \rangle . t$ is a closed term.

Remark 2.13. While similar to λ-abstraction, we deliberately do not use the λ-symbol, because it might suggest that $\langle x \rangle . t$ obeys the *β-law* when substituting terms, but due to the partial nature of the application map, this need not be the case [vOos08, p. 4].

It is now time to make use of combinatory completeness to construct some more combinators. We will do this in very much the same way as one encodes these constructs in the untyped λ-calculus.

Booleans and conditional

We define the booleans as the closed terms

$$\mathbf{true} := \langle xy \rangle . x \quad \text{and} \quad \mathbf{false} := \langle xy \rangle . y,$$

and the conditional as the closed term

$$\mathbf{if} := \langle x \rangle . x.$$

Note that for arbitrary $\mathrm{a}, \mathrm{b} \in \mathcal{A}$, we can calculate:

$$\mathbf{if}\,\mathbf{true}\,\mathrm{a}\,\mathrm{b} = \mathbf{true}\,\mathrm{a}\,\mathrm{b} = \mathrm{a} \quad \text{and} \quad \mathbf{if}\,\mathbf{false}\,\mathrm{a}\,\mathrm{b} = \mathbf{false}\,\mathrm{a}\,\mathrm{b} = \mathrm{b}\,.$$

Pairing and projection

We define the closed terms

$$\mathbf{pair} := \langle xyz \rangle . zxy, \quad \mathbf{fst} := \langle w \rangle . w\,\mathbf{true} \quad \text{and} \quad \mathbf{snd} := \langle w \rangle . w\,\mathbf{false}.$$

Exercise 2.14. For all elements $\mathrm{a}, \mathrm{b} \in \mathcal{A}$, prove that:

(i) $\mathbf{pair}\,\mathrm{a}\,\mathrm{b}$ is defined.

(ii) $\mathbf{fst}\,(\mathbf{pair}\,\mathrm{a}\,\mathrm{b}) = \mathrm{a}$ and $\mathbf{snd}\,(\mathbf{pair}\,\mathrm{a}\,\mathrm{b}) = \mathrm{b}$ hold.

Fixed points

We also have the fixed point combinators which are traditionally denoted by **Y** and **Z**:

$$\begin{aligned} \mathbf{Y} &:= \mathbf{W}\,\mathbf{W} && \text{with} && \mathbf{W} := \langle xy\rangle.\, y\,(x\,x\,y) \\ \mathbf{Z} &:= \mathbf{U}\,\mathbf{U} && \text{with} && \mathbf{U} := \langle xyz\rangle.\, y\,(x\,x\,y)\,z \end{aligned}$$

These combinators satisfy:

$$\mathbf{Y}\,\mathtt{f} \simeq \mathtt{f}\,(\mathbf{Y}\,\mathtt{f}), \quad \mathbf{Z}\,\mathtt{f} \text{ is defined} \quad \text{and} \quad \mathbf{Z}\,\mathtt{f}\,\mathtt{a} \simeq \mathtt{f}\,(\mathbf{Z}\,\mathtt{f})\,\mathtt{a}$$

for all elements $\mathtt{f}, \mathtt{a} \in \mathcal{A}$.

Curry numerals and fundamental arithmetic

For each natural number $n \in \mathbb{N}$, we define the corresponding **Curry numeral** $\overline{\mathtt{n}}$ inductively by:

$$\overline{0} := \mathbf{I} \quad \text{and} \quad \overline{\mathtt{n}+1} := \mathbf{pair}\,\mathbf{false}\,\overline{\mathtt{n}}\,.$$

Exercise 2.15. Define closed terms **succ**, **pred** and **iszero** such that for any natural number $n \in \mathbb{N}$ the following equations hold:

$$\begin{gathered} \mathbf{succ}\,\overline{\mathtt{n}} = \overline{\mathtt{n}+1}, \quad \mathbf{pred}\,\overline{0} = \overline{0}, \quad \mathbf{pred}\,\overline{\mathtt{n}+1} = \overline{\mathtt{n}}, \\ \mathbf{iszero}\,\overline{0} = \mathbf{true} \quad \text{and} \quad \mathbf{iszero}\,\overline{\mathtt{n}+1} = \mathbf{false}\,. \end{gathered}$$

Primitive recursion

The following **primitive recursion** combinator for Curry numerals will come in useful when we consider the natural numbers object in the category of assemblies later.

Exercise 2.16. Construct a closed term **primrec** such that for all $\mathtt{f}, \mathtt{a} \in \mathcal{A}$ and $n \in \mathbb{N}$ it satisfies:

$$\mathbf{primrec}\,\mathtt{a}\,\mathtt{f}\,\overline{0} = a \quad \text{and} \quad \mathbf{primrec}\,\mathtt{a}\,\mathtt{f}\,\overline{\mathtt{n}+1} \simeq \mathtt{f}\,\overline{\mathtt{n}}\,(\mathbf{primrec}\,\mathtt{a}\,\mathtt{f}\,\overline{\mathtt{n}})$$

Hint: The essential ingredients are a zero test, predecessor and repeated application which are provided by **iszero**, **pred** and **Z**, respectively.

As a final remark, we note that any nontrivial pca is necessarily infinite, as the following exercise shows:

Exercise 2.17. Prove that the following are equivalent for any pca $\mathcal{A}$:

(i) The booleans **true** and **false** are distinct elements of $\mathcal{A}$.

(ii) The Curry numerals $\overline{\mathtt{n}}$ in $\mathcal{A}$ are all distinct.

(iii) The pca $\mathcal{A}$ is not the trivial pca.

2.3 More examples of pcas

In this section we will present a few more examples of partial combinatory algebras. To motivate and introduce them, we offer the following informal explanation. Suppose we have a device which transforms the pitch of incoming audio. We might represent the incoming and outgoing audio as streams of bits, so that mathematically speaking, our in- and output are elements of $\{0,1\}^{\mathbb{N}}$, i.e. functions from the set of natural numbers $\mathbb{N}$ to the two-element set $\{0,1\}$. Our bit-stream transforming device is then a function $\{0,1\}^{\mathbb{N}} \to \{0,1\}^{\mathbb{N}}$. Since we don't want to wait forever on the output of our device, it seems reasonable that we assume it to start outputting after having only received finitely many bits. Thus, *the output depends on a finite amount of the input only*. Our device should therefore not be any old function $\{0,1\}^{\mathbb{N}} \to \{0,1\}^{\mathbb{N}}$, but rather one whose output depends on a finite amount of input only. This can be made mathematically precise by equipping the set $\{0,1\}^{\mathbb{N}}$ with a *topology* and by restricting our attention to *continuous* functions on such topologized sets.

For an introduction to and motivation of topology from a computer science perspective, Smyth's [Smy92] and Escardó's [Esc04] monographs, and Vickers's book [Vic96] are warmly recommended.

While Turing computable functions, or equivalently, Kleene's partial recursive functions, provide the foundation for computability theory with (encodings of) finite objects, an established theory of computability over infinite objects is Weihrauch's *Type Two Effectivity (TTE)* [Wei00]. This is a theory of computable analysis and becomes especially relevant when we are interested in exact real number computation. Its connections to realizability were explored in the PhD theses of Lietz [Lie04] and Bauer [Bau00] via Kleene's second model and its recursive variant (Examples 2.19 and 2.23 below).

We will be brief in our explanations of these partial combinatory algebras and hope that the interested reader will take the above paragraphs and the examples as an invitation to explore the fascinating connections between topology and computability, for instance by consulting Bauer's aforementioned notes on realizability [Bau23] and the references above.

Our first example is due to Scott [Sco76] and gives the powerset of the natural number the structure of a pca.

Example 2.18 (Scott's graph model, $\mathcal{S}$)**.** We make the powerset $\mathcal{P}(\mathbb{N})$ of the natural numbers into a pca that we denote by $\mathcal{S}$.
The idea behind application on $\mathcal{P}(\mathbb{N})$ is that when we apply a subset U to a subset V, then U can only use a finite amount of V to determine whether to include a number or not. We now make this idea formal.
We first fix bijections

$$[-,-]\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N} \qquad \text{and} \qquad \mathrm{enum}\colon \mathbb{N}\cong\mathcal{P}_{\mathrm{fin}}(\mathbb{N})$$

where the latter enumerates all finite subsets of natural numbers. We then define the application map $\mathcal{P}(\mathbb{N})\times\mathcal{P}(\mathbb{N})\to\mathcal{P}(\mathbb{N})$ as

$$(U,V)\mapsto\{n_{\mathrm{out}}\in\mathbb{N}\mid\exists(n_{\mathrm{in}}\in\mathbb{N}).[n_{\mathrm{in}},n_{\mathrm{out}}]\in U \text{ and } \mathrm{enum}(n_{\mathrm{in}})\subseteq V\}.$$

Thus, the "program" U encodes a list of input-output pairs where the input number encodes a finite subset of the argument V.

If we equip $\mathcal{P}(\mathbb{N})$ with the *Scott topology* whose basic opens are finite subsets of $\mathbb{N}$, then one can show that the application map is continuous. In fact, any continuous function $F : \mathcal{P}(\mathbb{N})^k \to \mathcal{P}(\mathbb{N})$ can be represented in $\mathcal{P}(\mathbb{N})$ by encoding the graph of F. (The details are spelled out in [dJon18, Example 2.3.4].) Under this correspondence it becomes easy to construct the elements **K** and **S** making $\mathcal{P}(\mathbb{N})$ into a pca.

The application in the above example is actually a *total* operation. An analogous pca whose application is genuinely partial is due to Kleene [KV65]:

Example 2.19 (Kleene's second model, $\mathcal{K}_2$)**.** Somewhat similar to the above example, we can define an application on the set $\mathbb{N}^{\mathbb{N}}$ of functions on $\mathbb{N}$ that makes it into a pca $\mathcal{K}_2$, known as Kleene's second model. The encodings required for the application map are a bit involved, see e.g. [Bau23, p. 30 and Section 2.1.2] or [vOos08, Section 1.4.3], but we point out that this pca is closely related to continuous functions $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ where we equip $\mathbb{N}^{\mathbb{N}}$ with the *Baire topology*.

Another example comes from *domain theory* [AC98] and is also due to Scott [Sco72]:

Example 2.20 (Scott's domain model of the untyped λ-calculus)**.** The carrier of this pca is Scott's domain D_∞ which is a model of the untyped λ-calculus via the isomorphism $\Phi\colon D_\infty \simeq {D_\infty}^{D_\infty}$ of domains. The map Φ sends an element $\sigma \in D_\infty$ to a Scott continuous function $\Phi(\sigma)\colon D_\infty \to D_\infty$. We define application on D_∞ as:

$$(\sigma, \tau) \mapsto \Phi(\sigma)(\tau).$$

Finally, we mention that Scott's graph model and Kleene's second model have effective variations that are examples of *elementary sub-pcas*.

Definition 2.21 (Elementary sub-pca)**.** An **elementary sub-pca** of a pca $\mathcal{A}$ is a subset $\mathcal{A}^\# \subseteq \mathcal{A}$ that is closed under the application of $\mathcal{A}$ and moreover contains the elements **K** and **S** from $\mathcal{A}$.

Example 2.22 (Scott's r.e. graph model, $\mathcal{S}^{\mathrm{re}}$)**.** We get an elementary sub-pca $\mathcal{S}^{\mathrm{re}}$ of $\mathcal{S}$ by restricting ourselves to the recursively enumerable subsets of $\mathbb{N}$.

Example 2.23 (Kleene's recursive second model, $\mathcal{K}_2^{\mathrm{rec}}$)**.** We get an elementary sub-pca $\mathcal{K}_2^{\mathrm{rec}}$ of $\mathcal{K}_2$ by restricting ourselves to the total recursive functions $\mathbb{N} \to \mathbb{N}$.

For even more examples of pcas, see [vOos08, Section 1.4].

2.4 List of exercises

1. Exercise 2.11: On the fundamental properties of $\langle x \rangle . t$.
2. Exercise 2.11: On the properties of the pairing and projection combinators.
3. Exercise 2.15: On fundamental arithmetic in a pca.
4. Exercise 2.16: On primitive recursion in a pca.
5. Exercise 2.17: On nontriviality of a pca.

Chapter 3

Categories of assemblies

In this chapter we describe the *category of assemblies* over a fixed but arbitrary pca. In particular, we show that the category has many desirable properties, for example it is (locally) cartesian closed and has finite colimits. In fact, the category has sufficient structure to support an interpretation of first order logic as we explore in Chapter 4.

Roughly speaking, the category of assemblies has sets with computability data as objects and computable functions between such sets as morphisms. The category of assemblies as a whole may be thought of as a world of computable mathematics.

Definition 3.1 (Assembly)**.** An **assembly** over a pca $\mathcal{A}$ is a set X together with a relation $\Vdash$ between $\mathcal{A}$ and X such that for all $x \in X$, there exists at least one element $\mathtt{a} \in \mathcal{A}$ with $\mathtt{a} \Vdash x$.

The relation $\mathtt{a} \Vdash x$ is pronounced as "$\mathtt{a}$ **realizes** x" and we also say that $\mathtt{a}$ is a **realizer** of x. We think of $\mathtt{a}$ as an *implementation* of $x \in X$ in the pca. The requirement on assemblies is that each element of the set must have at least one implementation.

Notation 3.2 ($|X|, \Vdash_X$)**.** Given an assembly X, we will write $|X|$ for its underlying set and $\Vdash_X$ for its relation between $\mathcal{A}$ and $|X|$.

Example 3.3 (Assembly of booleans, $\mathbf{2}$)**.** The **assembly of booleans**, denoted by $\mathbf{2}$, is defined as

$$|\mathbf{2}| := \{0, 1\} \quad \text{with realizers} \quad \mathbf{false} \Vdash_{\mathbf{2}} 0 \text{ and } \mathbf{true} \Vdash_{\mathbf{2}} 1,$$

where we recall the booleans **false** and **true** from Section 2.2.

Example 3.4 (Assembly of natural numbers, $\mathbf{N}$)**.** The **assembly of natural numbers**, denoted by $\mathbf{N}$, is defined as

$$|\mathbf{N}| := \mathbb{N} \qquad \text{and} \qquad \overline{\mathtt{n}} \Vdash_{\mathbf{N}} n \text{ for each } n \in \mathbb{N},$$

where we recall from Section 2.2 that $\overline{\mathtt{n}}$ is the n^{th} Curry numeral.

Example 3.5. Taking Kleene's first model as our pca, $\mathcal{A} = \mathcal{K}_1$, we can consider the assembly X of Turing computable functions:

$$|X| := \{f : \mathbb{N} \to \mathbb{N} \mid f \text{ is Turing computable}\} \qquad \text{and} \qquad m \Vdash_X f \iff \varphi_m = f$$

where we recall $\mathcal{K}_1$ and φ_- from Example 2.5. We remark that each $f \in |X|$ has infinitely many realizers.
Notice that, with this realizability relation, we cannot let $|X|$ be the set of *all* functions from $\mathbb{N}$ to $\mathbb{N}$, because then the set of realizers of a noncomputable function would be empty, which is not allowed by the definition of an assembly.

In the literature, assemblies over $\mathcal{K}_1$ are sometimes referred to as *ω-sets*.

3.1 Morphisms of assemblies

As mentioned in the opening paragraphs of this chapter, we wish to organize the assemblies into a category whose morphisms are "computable" functions between the underlying sets of assemblies. The computability requirement is made precise by requiring the existence of a *tracker* which we define now.

Definition 3.6 (Track)**.** For assemblies X and Y, we say that an element $\mathtt{t} \in \mathcal{A}$ **tracks** a function $f : |X| \to |Y|$ if for all $x \in |X|$ and $\mathtt{a} \in \mathcal{A}$, if $\mathtt{a} \Vdash_X x$, then $\mathtt{t}\,\mathtt{a}$ is defined and $\mathtt{t}\,\mathtt{a} \Vdash_Y f(x)$.

Notation 3.7. We will shorten the above to: "$\mathtt{t}\,\mathtt{a} \Vdash_Y f(x)$ for all $x \in |X|$ and $\mathtt{a} \Vdash_X x$". That is, we implicitly quantify over $\mathtt{a}$ and we implicitly assume that $\mathtt{t}\,\mathtt{a}$ is defined when we write $\mathtt{t}\,\mathtt{a} \Vdash_Y f(x)$.

Definition 3.8 (Assembly map)**.** An **assembly map** from an assembly X to an assembly Y is a function $f : |X| \to |Y|$ that is tracked by some element.

The existence of a tracker is a required *property* of an assembly map and *not* part of the data, so an assembly map is (unlike an assembly) *not* a pair of a function and a tracker; it is just a function for which there exists some tracker.

Proposition 3.9. *Assemblies and assembly maps form a category with composition given by composition of functions on underlying sets.*

Proof. We need to verify that composition is well defined, i.e., that if $f : X \to Y$ and $g : Y \to Z$ are assembly maps, then $g \circ f : |X| \to |Z|$ is tracked. Let $\mathtt{t}_\mathtt{f}$ and $\mathtt{t}_\mathtt{g}$ track f and g, respectively. We claim that $\langle x \rangle.\, \mathtt{t}_\mathtt{g}(\mathtt{t}_\mathtt{f}(x))$ tracks $g \circ f$. Indeed, the closed term $\langle x \rangle.\, \mathtt{t}_\mathtt{g}(\mathtt{t}_\mathtt{f}(x))$ is defined by construction, and if $\mathtt{a} \Vdash_X x$, then

$$(\langle x \rangle.\, \mathtt{t}_\mathtt{g}(\mathtt{t}_\mathtt{f}(x)))\,\mathtt{a} = \mathtt{t}_\mathtt{g}(\mathtt{t}_\mathtt{f}\,\mathtt{a}) \Vdash_Z g(f(x))$$

by choice of $\mathtt{t}_\mathtt{f}$ and $\mathtt{t}_\mathtt{g}$. Moreover, for each assembly X, we have an identity morphism on X given by the identity on $|X|$ and tracked by $\mathbf{I}$. Finally, associativity of composition holds because composing functions of sets is associative. □

Notation 3.10 ($\mathsf{Asm}_{\mathcal{A}}$)**.** We write $\mathsf{Asm}_{\mathcal{A}}$ for the category of assemblies over a pca $\mathcal{A}$.

Example 3.11. For the trivial pca $\mathcal{A} = \{\star\}$ we recover the familiar category Set of sets and functions as $\mathsf{Asm}_{\mathcal{A}}$.

Remark 3.12 (Relative categories of assemblies). An interesting variation on the category of assemblies is obtained if we start with a pca $\mathcal{A}$ and an elementary sub-pca $\mathcal{A}^{\#}$ (recall Definition 2.21 and Examples 2.22 and 2.23). While we still ask that the realizers of elements of sets come from the larger pca $\mathcal{A}$, we now require the trackers of assembly maps to come from the smaller sub-pca $\mathcal{A}^{\#}$ instead. The requirements on an elementary sub-pca guarantee that this is again a category which we call the **relative category of assemblies**.
Especially in the typical examples (2.22 and 2.23) where $\mathcal{A} = \mathcal{K}_2$ and $\mathcal{A}^{\#} = \mathcal{K}_2^{\mathrm{rec}}$, or $\mathcal{A} = \mathcal{S}$ and $\mathcal{A}^{\#} = \mathcal{S}^{\mathrm{re}}$, the idea of the relative categories of assemblies is nicely captured by Bauer's [Bau06; Bau23, p. 36 and 45] slogan

Topological data — computable functions!

3.2 Categorical constructions

This section shows that the category of assemblies has a rich categorical structure. In particular, we will construct finite (co)products, exponentials (in slices) and (co)equalizers of assemblies. This structure is important for the interpretation of first order logic in assemblies that we explore further in Chapter 4. But besides this, presenting the required constructions provides excellent opportunities for improving our understanding of the category of assemblies.

3.2.1 Cartesian closure and equalizers

We start by showing that the category of assemblies is cartesian closed and has equalizers. The description of the latter will prove useful in our study of regular monomorphisms of assemblies (Section 3.4.2).

Proposition 3.13 (Terminal object)**.** *The terminal object* $\mathbf{1}$ *in* $\mathsf{Asm}_{\mathcal{A}}$ *is given by*

$$|\mathbf{0}| := \{\star\} \quad \textit{and} \quad \mathrm{a} \Vdash_{\mathbf{1}} \star \textit{ for all } \mathrm{a} \in \mathcal{A}.$$

Proof. As in Set. □

The pairing and projection combinators from Section 2.2 are essential in the construction of finite products of assemblies.

Proposition 3.14 (Products)**.** *The product* $X \times Y$ *of two assemblies* X *and* Y *is given by*

$$|X \times Y| := |X| \times |Y| \quad \textit{and} \quad \mathbf{pair}\,\mathrm{a}\,\mathrm{b} \Vdash_{X\times Y} (x, y) \textit{ for } \mathrm{a} \Vdash_X x \textit{ and } \mathrm{b} \Vdash_Y y.$$

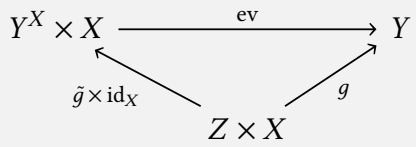
Proof. The projection maps $\pi_1 \colon X \times Y \to X$ and $\pi_2 \colon X \times Y \to Y$ are given by $(x, y) \mapsto x$ and $(x, y) \mapsto y$, and tracked by **fst** and **snd**, respectively. Moreover, every pair of assembly maps $f \colon Z \to X$ and $g \colon Z \to Y$ induces an assembly map $\langle f, g \rangle \colon Z \to X \times Y$ given by $z \mapsto (f(z), g(z))$ and tracked by $\langle u \rangle.\, \mathbf{pair}(\mathtt{t}_f\, u)(\mathtt{t}_g\, u)$ when $\mathtt{t}_f$ and $\mathtt{t}_g$ track f and g, respectively. □

So far, the underlying sets of the terminal object and product of two assemblies have been exactly as in Set, e.g. $|X \times Y|$ is the product of the two sets $|X|$ and $|Y|$ in Set. A notable exception to this is the exponential (a.k.a. "internal hom"): the object of morphisms between two assemblies. It would not make sense for the carrier of the exponential of assemblies X and Y to consist of all functions from $|X|$ to $|Y|$, because (a) the exponential is usually similar to the hom-set of morphisms from X to Y, and (b) it would not be clear what the realizers of an arbitrary function between carriers should be. Instead, the exponential is given by assembly maps only.

Proposition 3.15 (Exponentials). *The exponential Y^X of two assemblies X and Y is given by*

$$\left|Y^X\right| := \textit{the set of assembly maps from } X \textit{ to } Y \quad \textit{and} \quad \mathtt{t} \Vdash_{Y^X} f \textit{ if } \mathtt{t} \textit{ tracks } f.$$

Proof. The evaluation morphism $\mathrm{ev} \colon Y^X \times X \to Y$ given by $(f, x) \mapsto f(x)$ is tracked by $\langle u \rangle.\, \mathbf{fst}\, u(\mathbf{snd}\, u)$. Moreover, every $g \colon Z \times X \to Y$ induces a unique assembly map $\tilde{g} \colon Z \to Y^X$ making the diagram



commute. Indeed, there is a unique assignment $\tilde{g}(z) := (x \mapsto g(z, x))$ and this assignment is tracked by $\langle u \rangle.\, (\langle v \rangle.\, \mathtt{t}_g(\mathbf{pair}\, u\, v))$ when $\mathtt{t}_g$ tracks g. □

Thus, we conclude that the category $\mathrm{Asm}_{\mathcal{A}}$ is cartesian closed.

Finally, we construct equalizers in $\mathrm{Asm}_{\mathcal{A}}$. Their description will come in useful when we study regular monomorphisms in Section 3.4.2. We also note that we obtain an explicit construction of pullbacks when combined with the construction of products.

Proposition 3.16 (Equalizers). *The equalizer E of two assembly maps $f, g \colon X \to Y$ is given by*

$$|E| := \{x \in |X| \mid f(x) = g(x)\} \quad \textit{and} \quad \mathtt{a} \Vdash_E x \textit{ if } \mathtt{a} \Vdash_X x.$$

Proof. On the level of underlying sets, this is as in the category of sets, so it suffices to show that the relevant functions are tracked. The inclusion $i \colon |E| \to |X|$ is tracked by $\mathbf{I}$. Given an assembly map $h \colon D \to X$ such that $f \circ h = f \circ h$, the map $h \colon |D| \to |X|$ factors uniquely through i via $k \colon |D| \to |E|$ which is tracked by any tracker of h. □

3.2.2 Coproducts and coequalizers

We now move on to colimits in the category of assemblies.

Proposition 3.17 (Initial object). *The initial object* $\mathbf{0}$ *in* $\mathsf{Asm}_{\mathcal{A}}$ *is given by* $|\mathbf{0}| := \emptyset$ *with the empty realizability relation.*

Proof. As in Set. □

Proposition 3.18 (Coproducts). *The coproduct* $X + Y$ *of two assemblies* X *and* Y *is given by*

$$|X + Y| := |X| + |Y| \quad \textit{and} \quad \begin{aligned} \mathbf{left}\ a \Vdash_{X+Y} \mathrm{inl}(x) \ \textit{for}\ a \Vdash_X x, \\ \mathbf{right}\ b \Vdash_{X+Y} \mathrm{inr}(y) \ \textit{for}\ b \Vdash_Y y, \end{aligned}$$

where

$$\mathbf{left} := \mathbf{pair}\,\mathbf{false} \ \textit{and}\ \mathbf{right} := \mathbf{pair}\,\mathbf{true}\,.$$

Notice that the disjointness of the coproduct is witnessed by tagging the realizers with booleans.

Exercise 3.19. Prove Proposition 3.18.
Warning: Carefully check that the closed terms you give for the trackers are defined and thus give elements of the pca as required.

While there are several interesting assemblies whose carrier is the set $\{0, 1\}$ (as we will see in Chapter 4), the following exercise justifies the notation $\mathbf{2}$ for the assembly of booleans as defined in Example 3.3.

Exercise 3.20. Show that $\mathbf{1} + \mathbf{1} \cong \mathbf{2}$.

Finally, we construct coequalizers in $\mathsf{Asm}_{\mathcal{A}}$. Their description will be useful in our study of regular epimorphisms (Section 3.4.1).

Proposition 3.21 (Coequalizers). *The coequalizer* C *of assembly maps* $f, g\colon X \to Y$ *is given by*

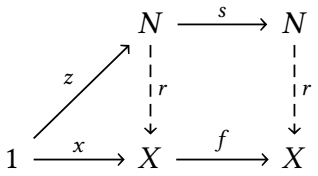
$$|C| := |Y|/{\sim} \quad \textit{and} \quad a \Vdash_C [y] \ \textit{if}\ a \Vdash_Y y' \ \textit{for some}\ y' \sim y,$$

where $\sim$ *is the least equivalence relation on* $|Y|$ *generated by* $f(x) \sim g(x)$ *for all* $x \in |X|$.

Proof. On the level of underlying sets, this is as in the category of sets, so it suffices to show that the relevant functions are tracked. The quotient map $q\colon |Y| \to |Y|/{\sim}$ is tracked by $\mathbf{I}$. Given an assembly map $h\colon Y \to Z$ such that $h \circ f = h \circ g$, the map $h\colon |Y| \to |Z|$ factors uniquely through q via $k\colon |Y|/{\sim} \to |Z|$ with $k([y]) := h(y)$. Moreover, the function k is tracked by any tracker of h. □

3.2.3 Natural numbers object

The notion of natural numbers can be captured categorically via a universal property which is due to Lawvere [Law63]. In a category $\mathcal{C}$, a **natural numbers object (nno)** is an object N equipped with morphisms $z\colon 1 \to N$ ("zero") and $s\colon N \to N$ ("successor") such that for all triples $(X, x\colon 1 \to X, f\colon X \to X)$ there is a unique morphism $r\colon N \to X$ (defined by "recursion") making the diagram



commute.

Exercise 3.22.

(i) Exhibit $\mathbb{N}$ as a nno in Set.

(ii) Exhibit $\mathbf{N}$ (from Example 3.4) as a nno in $\mathsf{Asm}_{\mathcal{A}}$.

Hint: Use Exercise 2.16.

Looking ahead to Chapter 4 we note that if we wish to interpret arithmetic in the category of assemblies, then its natural numbers object serves as the interpretation of the sort of natural numbers.

3.2.4 Dependent products and sums

For the purposes of these lecture notes, a proof sketch of the following result suffices. We discuss the significance of the result below.

Proposition 3.23. *For every morphism $f\colon X \to Y$ of assemblies, the pullback functor $f^*\colon \mathsf{Asm}_{\mathcal{A}}/Y \to \mathsf{Asm}_{\mathcal{A}}/X$ has both a left adjoint $\sum_f$ and a right adjoint $\prod_f$.*

Proof sketch. We only describe the constructions and leave the verification of the details to the interested reader. The left adjoint $\sum_f$ takes an object $g\colon Z \to X$ of $\mathsf{Asm}_{\mathcal{A}}/X$ to the object $f \circ g$ of $\mathsf{Asm}_{\mathcal{A}}/Y$. On morphisms it is the identity.
The right adjoint $\prod_f$ is more involved. Given an object $g\colon Z \to X$ of $\mathsf{Asm}_{\mathcal{A}}/X$, we consider the assembly P of "fiberwise maps". It is given by

$$|P| := \{(y, s) \mid s\colon f^{-1}(y) \to Z \text{ such that } \forall (x \in |f^{-1}(y)|)\,.\, s(x) \in |g^{-1}(x)|\},$$

where

$$|f^{-1}(y)| := \{x \in |X| \mid f(x) = y\} \quad \text{with realizers} \quad \mathtt{a} \Vdash_{f^{-1}(y)} x \text{ if } \mathtt{a} \Vdash_X x,$$

(and similarly for $g^{-1}(x)$), and for realizers, we put

$$\mathbf{pair}\,\mathtt{b}\,\mathtt{t} \Vdash_P (y, s) \text{ if } \mathtt{b} \Vdash_Y y \text{ and } \mathtt{t} \text{ tracks } s.$$

Now, P defines an object of $\mathsf{Asm}_{\mathcal{A}}/Y$ by considering the first projection $\pi_1\colon P \to Y$

which is tracked by **fst**. This projection map is the value of $\prod_f(g)$. Given a morphism

$$\begin{array}{ccc} Z & \xrightarrow{k} & W \\ & {\scriptstyle g}\searrow \quad \swarrow{\scriptstyle h} & \\ & X & \end{array}$$

in $\mathsf{Asm}_{\mathcal{A}}/X$, we define $\prod_f(k)\colon \prod_f(g) \to \prod_f(h)$ as a function on sets by $(y, s) \mapsto (y, h \circ s)$. This assignment can be shown to be tracked because h and s are. □

The above proposition is equivalent to the fact that the category $\mathsf{Asm}_{\mathcal{A}}$ is *locally cartesian closed*, i.e. that each slice category $\mathsf{Asm}_{\mathcal{A}}/X$ is cartesian closed. One may check that the exponential g^f of two objects $f, g \in \mathsf{Asm}_{\mathcal{A}}/X$ is given by $\prod_f(f^*(g))$.

The functors $\sum_f$ and $\prod_f$ also satisfy the so-called Beck–Chevalley condition [Str18, Theorem 4.4], which we don't spell out here. Intuitively, this condition expresses that $\prod$ and $\sum$ preserve substitution which is given by pullback (see e.g. [Bau12] for an informal explanation of the latter).

The adjoints allow us to interpret Martin-Löf dependent type theory, where, as the notation suggests, dependent sums and products are interpreted using $\sum$ and $\prod$, respectively. The interested reader may consult [See84; Jac99; Str91] to learn more. We implicitly rely on (a slight variation of) Proposition 3.23 for the interpretation of first order logic in the category of assemblies in Chapter 4, where we only need the adjoints for f a projection map. But our presentation in Chapter 4 will spell things out in more concrete terms, so understanding Proposition 3.23 in detail is not strictly required.

3.3 Relation to the category of sets

We introduce an adjunction

$$\mathsf{Asm}_{\mathcal{A}} \underset{\nabla}{\overset{\Gamma}{\underset{\longleftarrow}{\overset{\longrightarrow}{\bot}}}} \mathsf{Set}$$

relating the categories of sets and assemblies.

Definition 3.24 (Forgetful functor, Γ)**.** The **forgetful functor**

$$\Gamma\colon \mathsf{Asm}_{\mathcal{A}} \to \mathsf{Set}$$

is defined by taking an assembly X to its underlying set $|X|$ and a morphism $f\colon X \to Y$ of assemblies to the map of sets $f\colon |X| \to |Y|$.

Exercise 3.25. Show that Γ is naturally isomorphic to the **global sections** functor:

$$\begin{aligned} \mathsf{Asm}_{\mathcal{A}} &\to \mathsf{Set} \\ X &\mapsto \mathsf{Asm}_{\mathcal{A}}(\mathbf{1}, X) \\ f\colon X \to Y &\mapsto \text{post-composition with } f \end{aligned}$$

To get a functor $\nabla\colon \mathsf{Set} \to \mathsf{Asm}_{\mathcal{A}}$ we need a procedure for turning a set X into an assembly ∇X. Of course, it makes sense to let $|\nabla(X)|$ be the set X, but what should

the realizers be? In general, there is no reason why an element x of an arbitary set X should have an "implemention" in the pca $\mathcal{A}$. In light of this, one might be tempted to say that there are no realizers of x. This is not a good idea for two reasons:

(1) It does not define an assembly, because, by definition, every element should have at least one realizer.

(2) Even if the definition of assembly did allow for an empty set of realizers, then the tracking requirement on assembly maps would mean that there are no assembly maps $2 \to \nabla\{0,1\}$ which does not make sense if $\nabla\{0,1\}$ is an assembly with no computational data.

The solution is to say that *all* elements of $\mathcal{A}$ are actually realizers of $x \in |\nabla(X)|$. The idea is that $\nabla(X)$ carries no meaningful computational data, because all elements have the same set of realizers, so from this perspective all elements of the assembly look alike and computationally speaking we can't tell them apart.

Definition 3.26 (∇)**.** We define a functor

$$\nabla\colon \mathsf{Set} \to \mathsf{Asm}_{\mathcal{A}}$$

by mapping a set X to the assembly with carrier X and $a \Vdash_X x$ for *all* elements $a \in \mathcal{A}$. A map of sets $f\colon X \to Y$ gets send to f and is tracked by $\mathbf{I}$.

Exercise 3.27. Prove that Γ is left adjoint to ∇.

Notation 3.28 (η)**.** We write η for the unit of the adjunction $\Gamma \vdash \nabla$, i.e. for each assembly X, we have an assembly map

$$\eta_X\colon X \to \nabla|X|$$

given by the identity on $|X|$ and tracked by (for example) $\mathbf{I}$.

In particular, we have an assembly map $\nabla_2\colon 2 \to \nabla\{0,1\}$. We do not expect a map in the reverse direction as it would mean that we can compute the relevant boolean (**true** or **false**) without receiving any relevant computational input. Indeed, we have:

Exercise 3.29. Show that there are no assembly maps $f\colon \nabla\{0,1\} \to 2$ in $\mathsf{Asm}_{\mathcal{A}}$ unless the pca $\mathcal{A}$ is trivial.
Hint: Use Exercise 2.17.

To complete the section, the following exercises ask you to check that the functors Γ and ∇ do not have other adjoints.

Exercise 3.30. Show that ∇ does not have a right adjoint when $\mathcal{A}$ is nontrivial.

Exercise 3.31 (cf. [Zoe18, Lemma 5.1.7])**.** We assume that the pca $\mathcal{A}$ is nontrivial. The aim of these exercises is to conclude that Γ does not have a left adjoint.

(i) Show that, for any object $X \in \mathsf{Asm}_{\mathcal{A}}$, there are at most $|\mathcal{A}|$-many arrows from X to 2.

(ii) Use the above to prove that the $\mathcal{A}$-indexed coproduct of copies of $\mathbf{1}$ does not exist in $\mathsf{Asm}_{\mathcal{A}}$.

(iii) Conclude that Γ does not have a left adjoint.

The functor ∇ plays an important role in classifying regular monomorphisms and recovering classical logic within realizability logic as explained in Chapter 4.

3.4 Epimorphisms and monomorphisms

Epimorphisms and monomorphisms and their regular counterparts will be very important to the logical side of the category of assemblies (see Chapter 4), but studying them also provides excellent opportunities for improving our understanding and intuition of assemblies in general.

Proposition 3.32 (Characterization of epis and monos)**.** *For an assembly map f, we have the following equivalences:*

(i) $f : X \to Y$ is an epimorphism if and only if $f : |X| \to |Y|$ is surjective;

(ii) $f : X \to Y$ is a monomorphism if and only if $f : |X| \to |Y|$ is injective.

Proof. Surjectivity is clearly sufficient to force an assembly map to be epi. For the converse, we use that Γ preserves epimorphisms as it is a left adjoint[a]. Thus, if $f : |X| \to |Y|$ is an epi, then $\Gamma(f)$ must be an epi in Set, i.e. a surjection.
For the characterization of monomorphisms, injectivity is again clearly sufficient. Conversely, it follows from our construction of products and equalizers that Γ preserves monomorphisms (use the dual of [a]). Thus, if $f : |X| \to |Y|$ is a mono, then $\Gamma(f)$ must be a mono in Set, i.e. an injection. □

[a] In any category, a morphism f is an epi if and only if the square $\begin{smallmatrix} X & \xrightarrow{f} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathrm{id}} \\ Y & \xrightarrow{\mathrm{id}} & Y \end{smallmatrix}$ is a pushout. Since left adjoints preserve colimits (and identities), they also preserve epimorphisms.

At first sight, it is perhaps surprising that being an epimorphism or monomorphism depends solely on the underlying function of an assembly map with the realizers playing no role. The situation is very different for *regular* epimorphisms and monomorphisms.

3.4.1 Regular epimorphisms

Recall that a morphism $f : X \to Y$ is a **regular epimorphism** if it fits in a coequalizer diagram

$$Z \underset{h}{\overset{g}{\rightrightarrows}} X \xrightarrow{f} Y$$

for some morphisms g and h.

Exercise 3.33 (Characterization of regular epimorphisms)**.** Prove that an assembly map $f : X \to Y$ is a regular epimorphism if and only if $f : |X| \to |Y|$ is surjective and

there exists an element $\mathtt{s} \in \mathcal{A}$ such that for all $y \in |Y|$ and $b \Vdash_Y y$, we have $\mathtt{s}\,\mathtt{b} \Vdash_X x$ for some $x \in |X|$ with $f(x) = y$.

The element $\mathtt{s}$ in Exercise 3.33 effectively witnesses the surjectivity of f.

Exercise 3.34. Give an example of an epimorphism in $\mathsf{Asm}_{\mathcal{A}}$ which is not regular (for a nontrivial pca $\mathcal{A}$).

Proposition 3.35. *The regular epimorphisms in* $\mathsf{Asm}_{\mathcal{A}}$ *are stable under pullback along arbitrary assembly maps.*

Proof. Consider a pullback diagram

$$\begin{array}{ccc} X \times_Z Y & \xrightarrow{\pi_2} & Y \\ {\scriptstyle \pi_1}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ X & \xrightarrow{f} & Z \end{array}$$

with g a regular epimorphism. We must show that π_1 is also a regular epi. From the description of equalizers and products we can compute that

$$|X \times_Z Y| \coloneqq \{(x, y) \mid f(x) = g(y)\} \text{ with realizers}$$
$$\mathbf{pair}\,\mathtt{a}\,\mathtt{b} \Vdash_{X \times_Z Y} (x, y) \text{ for } \mathtt{a} \Vdash_X x \text{ and } \mathtt{b} \Vdash_Y y.$$

By assumption and Exercise 3.33 there exists an element $s \in \mathcal{A}$ such that for every $z \in |Z|$ and $\mathtt{c} \Vdash_Z z$ we have $\mathtt{s}\,\mathtt{c} \Vdash_Y y$ for some $y \in |Y|$ with $g(y) = z$. Now if $\mathtt{t}$ tracks f and we put

$$\mathtt{s}' \coloneqq \langle u \rangle.\, \mathbf{pair}\, u\, (\mathtt{s}(\mathtt{t}\, u)),$$

then for every $x \in |X|$ and $\mathtt{a} \Vdash_X x$ we have $\mathtt{s}'\,\mathtt{a} \Vdash_{X \times_Z Y} (x, y)$ for some $y \in |Y|$ with $g(y) = f(x)$. Hence, π_1 is a regular epi by Exercise 3.33, as desired. □

The importance of Proposition 3.35 is that, together with Proposition 3.23, it gives the category of assemblies enough structure to interpret first order logic as we explore in the next chapter.

We recall that in the category of sets, every function $f : A \to B$ factors as a regular epimorphism (= surjection) followed by a monomorphism (= injection):

$$\begin{array}{ccc} A & \xrightarrow{\quad f \quad} & B \\ & {\scriptstyle \tilde{f}}\searrow \quad \nearrow & \\ & \mathrm{im}(f) & \end{array}$$

where $\mathrm{im}(f) \coloneqq \{b \in B \mid \exists (a \in A).\, f(a) = b\}$.

The same is true in any category with finite limits and pullback stable regular epimorphisms. For the category of assemblies it is also straightforward to calculate the factorization directly, as we ask you to verify:

Exercise 3.36. Given an assembly map $f : X \to Y$, show how to factor it in $\mathsf{Asm}_{\mathcal{A}}$ as a regular epimorphism followed by a monomorphism.

3.4.2 Regular monomorphisms

There is also a nice characterization of regular monomorphisms which, as we explain in the next chapter, represent classical (or more precisely, $\neg\neg$-stable) subsets of assemblies.

Recall that a morphism $f\colon X \to Y$ is a **regular monomorphism** if it fits in an equalizer diagram

$$X \xrightarrow{f} Y \underset{h}{\overset{g}{\rightrightarrows}} Z$$

for some morphisms g and h.

Exercise 3.37 (Characterization of regular monomorphisms)**.** Prove that an assembly map $f\colon X \to Y$ is a regular monomorphism if and only if $f\colon |X| \to |Y|$ is injective and there exists an element $\mathtt{i} \in \mathcal{A}$ such that $\mathtt{i}\,\mathtt{b} \Vdash_X x$ for all $x \in |X|$ and $\mathtt{b} \Vdash_Y f(x)$.

Notice that the element $\mathtt{i}$ in Exercise 3.37 acts like an effective witness of left-cancellability of f: from a realizer of $f(x)$ we can effectively find a realizer of x.

Exercise 3.38. Give an example of a monomorphism in $\mathsf{Asm}_{\mathcal{A}}$ which is not regular (for a nontrivial pca $\mathcal{A}$).

3.5 From pcas to assemblies, functorially

In this chapter we constructed a category of assemblies $\mathsf{Asm}_{\mathcal{A}}$ over an arbitrary pca $\mathcal{A}$. The construction $\mathcal{A} \mapsto \mathsf{Asm}_{\mathcal{A}}$ is actually functorial in a suitable sense. To make this precise, we need a *category* of pcas. On first thought, one might think that a suitable notion of a morphism between pcas $\mathcal{A}$ and $\mathcal{B}$ is a map on the underlying sets of the pcas that preserves application. This turns out *not* to be an appropriate notion of morphism however. It would be if a pca should be thought of as an algebraic gadget, but it shouldn't: for example, the application map is associative if and only if the pca is trivial [vOos08, Proposition 1.3.1].

Instead, an appropriate notion of morphism, due to Longley [Lon95], is that of an *applicative morphism* (re-branded to *simulation* in [Bau23]). For lack of space, we will not go into the details here and instead give some relevant pointers to the literature.

The intuition is that a morphism between pcas from $\mathcal{A}$ to $\mathcal{B}$ should assign to each program $\mathtt{a} \in \mathcal{A}$ one or more programs in $\mathcal{B}$ that *simulate* $\mathtt{a}$ in $\mathcal{B}$. Moreover, similar to the requirement that assembly maps are tracked, the simulation should have an effective witness in the pca $\mathcal{B}$. With applicative morphisms between them we get a category of pcas. In fact, this category is preorder-enriched, so we even get a 2-category.

Now, one can show that every applicative morphism $\gamma\colon \mathcal{A} \to \mathcal{B}$ gives rise to a functor $F_\gamma\colon \mathsf{Asm}_{\mathcal{A}} \to \mathsf{Asm}_{\mathcal{B}}$. The induced functor F_γ is regular (i.e. it preserves finite limits and regular epis) and is a so-called *S-functor*: it is the identity on underlying sets and functions. In fact, every such functor arises from an applicative morphism. In the end, we actually get an equivalence of 2-categories [vOos08, Theorem 1.6.2] between:

- the 2-category of pcas with applicative morphisms and their preorders, and
- the 2-category of categories of assemblies with regular S-functors between them and (necessarily unique) natural transformations between the functors.

3.6 List of exercises

1. Exercise 3.19: On the universal property of coproducts.
2. Exercise 3.20: On the assembly of booleans as the coproduct $\mathbf{1} + \mathbf{1}$.
3. Exercise 3.22: On natural numbers objects.
4. Exercise 3.25: On the forgetful functor Γ and the global sections functor.
5. Exercise 3.25: On Γ being a left adjoint to ∇.
6. Exercise 3.29: On the nonexistence of assembly maps $\nabla\{0,1\} \to \mathbf{2}$.
7. Exercise 3.30: On the nonexistence of a right adjoint to ∇.
8. Exercise 3.31: On the nonexistence of a left adjoint to Γ.
9. Exercise 3.33: On characterizing the regular epimorphisms.
10. Exercise 3.34: On an example of a epimorphism which is not a regular.
11. Exercise 3.36: On factoring an assembly map as a regular epi followed by a mono.
12. Exercise 3.37: On characterizing the regular monomorphisms.
13. Exercise 3.38: On an example of a monomorphism which is not a regular.

Chapter 4

The realizability interpretation of logic

In this chapter we explore some of the logical aspects of the category of assemblies over a pca. The theory of categorical logic informs us that any sufficiently structured category supports an interpretation of logic. For example, in a regular category we can interpret a fragment of first order logic known as regular logic, while we can interpret higher order logic in a topos. We include a brief introduction to categorical logic in Section 4.1, but for a proper treatment we refer the reader to the textbooks [FS90] (first order logic), [LS86; MM94] (higher order logic), or the lecture notes [vOos16] (regular logic) and [Str04] (higher order logic). The key point is this:

> Categorical logic gives us a *uniform* way of interpreting logic in any sufficiently structured category.

Following Bauer's exposition in [Bau23] we describe how to interpret logic in the category of assemblies via the notion of a *realizability predicate* on an assembly. We emphasize that this is *not* some ad-hoc interpretation. Rather, it is a convenient unfolding of the uniform interpretation given to us by categorical logic. The structure established on the category of assemblies in Chapter 3 allows us to interpret full first order logic.

Having said all of this, the reader who finds themselves ill at ease with categorical logic can find solace in the fact that the particular—realizability—interpretation of logic is spelled out in more elementary terms in Section 4.2 and Section 4.2.2 in particular.

After establishing that the logic in the category of assemblies is given by realizability predicates, Section 4.3 isolates three particular classes of such predicates by logical means. These are the $\neg\neg$-stable, decidable and semidecidable realizability predicates. In assemblies over Kleene's first model these are shown to respectively correspond to ordinary (classical) subsets, computable subsets and computably enumerable subsets.

Finally, Section 4.4 illustrates the intimate connections between logic, computability and categories via *synthetic computability theory* [Bau06]. Specifically, we give a synthetic proof of a fundamental result in computability theory: a subset of the natural numbers is computable if and only if it and its complement are computably enumerable.

4.1 Categorical logic in a nutshell

Suppose we have a formula $\phi(x)$ with a free variable x. If we interpret x to range over some set X, then $\phi(x)$ determines a subset of X, namely $\{x \in X \mid \phi(x)\}$, the subset of elements x of X for which $\phi(x)$ holds. A conjunction of formulas also determines a subset: the intersection of the subsets determined by ϕ and ψ:

$$\{x \in X \mid \phi(x) \wedge \psi(x)\} = \{x \in X \mid \phi(x)\} \cap \{x \in X \mid \psi(x)\}.$$

Similarly, with conjunction and union of course. The formulas $\top$ (truth) and $\bot$ (falsity) determine the two extreme subsets X and $\emptyset$, respectively. The subsets of X are a partial order when equipped with the subset relation. Its greatest element is X, its least element is $\emptyset$, the intersection of two subsets is the greatest lower bound of the two subsets, while the union is given by the least upper bound. Thus, we see that we can interpret a formula as a subset and that the logical connectives are interpreted using operations (characterized by universal properties) on subsets.

From subsets to monos In categorical logic, we generalize this from the category of sets to arbitrary, sufficiently rich categories. Instead of subsets we consider monomorphisms, or really *subobjects*[1]. The monos into a fixed object X in a category form a preorder by setting $(M \hookrightarrow X) \leq (N \hookrightarrow X)$ if we have a map $M \to N$ making the triangle

$$\begin{array}{ccc} M & \hookrightarrow & X \\ & \searrow \quad \nearrow & \\ & N & \end{array}$$

commute. (One can check that the map $M \to N$ is necessarily a mono.)

This preorder generalizes the subsets with their subset relation. The relation $\leq$ is reflexive and transitive, but not necessarily antisymmetric. But we can consider the *poset reflection* of this preorder, where we quotient such that monos $M \hookrightarrow X$ and $N \hookrightarrow X$ are identified when $M \leq N$ and $N \leq M$ both hold. One may check that an element of the resulting poset is precisely a subobject.

Logical connectives and monos The formula $\top$ is interpreted as the greatest element in the preorder of monos into X, i.e. as $\mathrm{id}_X : X \to X$. The formula $\bot$ is interpreted as the least element in the preorder of monos into X, which in a cartesian closed category with an initial object 0 (like the category of assemblies) indeed exists and is given by the unique map[2] $0 \to X$.

Following the universal properties of subsets, we deduce that conjunction should be interpreted using the greatest lower bound in the preorder of monos. Assuming our category has pullbacks, one can check that taking the pullback

$$\begin{array}{ccc} M \wedge N & \hookrightarrow & N \\ \downarrow & \lrcorner & \downarrow \\ M & \hookrightarrow & X \end{array}$$

[1] A subobject of X is a monomorphism into X up to isomorphism in the slice category over X.

[2] The proof that this map is indeed a mono is short, but surprisingly tricky. The interested reader might like to prove it for themselves. In case they get stuck, see [McL92, Theorem 6.3] or [Fav24].

of two monos into X gives their greatest lower bound $M \wedge N \hookrightarrow X$. (In the diagram all maps are monos because those are stable under pullback.)

Interpreting disjunction is slightly more involved: for two monos $M, N \hookrightarrow X$ the induced map $M + N \to X$ from the coproduct is not in general a mono. Instead, we require that the category factors any map as a regular epi followed by a mono (like we have in the category of assemblies by Exercise 3.36) and we factor $M + N \to X$ as $M + N \twoheadrightarrow M \vee N \hookrightarrow X$.

It is an insight of Lawvere [Law69] that the quantifiers $\forall$ and $\exists$ can also be suitably captured by universal properties using adjunctions. Any morphism of the category $f : X \to Y$ induces a monotone map $f^* : \mathrm{Mono}(Y) \to \mathrm{Mono}(X)$ on the preorders of monos into Y and X respectively by pulling back along f. Viewing these preorders as categories, the map f^* has a right adjoint $\exists_f$ by defining $\exists_f(m)$ to be the monomorphism part of the factorization of $f \circ m$ as a regular epi followed by a mono. For the universal quantifier, we then require f^* to also have a left adjoint which we denote by $\forall_f$. For locally cartesian closed categories, like the category of assemblies (Proposition 3.23), we already have a left adjoint $\prod_f$ to the pullback functor which, being a left adjoint, preserves monos and hence restricts to a monotone map on the preorders of monos.

Sorts, relations and terms We assume to be given a many-sorted language. This means that variables, function symbols and relation symbols are typed by an assignment of *sorts*. The sort of a variable indicates what it is supposed to range over. Function symbols have a source and target sort and relation symbols has as many sorts as its arity. For example, if we are interested in arithmetic we might have a single sort N for the natural numbers and a function symbol s for the successor map with source and target sort N. We can build terms by e.g. applying a function symbol to a variable, subject to the condition that the sorts match up of course.

An interpretation of the language in the category is given by several assignments:

- each sort X is interpreted as an object $[\![X]\!]$ of the category;
- each function symbol f with source sort S and target sort T is interpreted as an arrow $[\![f]\!] : [\![S]\!] \to [\![T]\!]$ in the category;
- each relation symbol R with sorts $X_1, \ldots, X_n$ is interpreted as a subobject $[\![R]\!]$ of the product $[\![X_1]\!] \times \cdots \times [\![X_n]\!]$.

We can also interpret equality of two terms s and t by taking the equalizer of their interpretations $[\![s]\!]$ and $[\![t]\!]$.

For the example of the language of arithmetic, it would make good sense to require the category to have a natural numbers object and to use this as the interpretation of the sort of the natural numbers. If we added sorts N^N, $N^{(N^N)}$, etc. for functions then we would interpret these using exponentials in a cartesian closed category.

The purpose of the above is that it tells us that a formula $\phi(x)$ with a free variable x of sort X should be interpreted as a monomorphism into $[\![X]\!]$, i.e. it tells us which object to consider. The actual interpretation of $\phi(x)$ as mono into $[\![X]\!]$ can then be calculated by recursion on the structure of ϕ. (If ϕ has more than one free variable, say $x_1, \ldots, x_n$ of sorts $X_1, \ldots, X_n$, respectively, then we simply consider the object $[\![X_1]\!] \times \cdots \times [\![X_n]\!]$.)

We write $\forall(x : X).\phi(x)$ and $\exists(x : X).\phi(x)$ when quantifying over a sort X. The colon (:) should remind us that these quantifications are interpreted in the category.

Internal language Given a category $\mathcal{C}$, a particular language we might consider is the *internal language* of $\mathcal{C}$ where we add a sort for each object of $\mathcal{C}$, a function symbol for each morphism of $\mathcal{C}$, and finally a relation symbol for each subobject of $\mathcal{C}$. This language has a natural interpretation in $\mathcal{C}$: if we added a sort for an object X, then the interpretation of that sort is X; and similarly for the function and relation symbols. In this case we do not distinguish notationally between the object and the sort, the function symbol and the morphism, and the relation symbol and the subobject.

Soundness One can show that the poset of subobjects is always a *Heyting algebra* whenever the category has sufficient structure (e.g. when it is regular and the adjoint $\forall_f$ exists). As a consequence we have:

Theorem (Soundness theorem)**.** *If we can prove ψ from ϕ in constructive logic, then $[\![\phi]\!] \preceq [\![\psi]\!]$ holds in the poset of subobjects.*

The subobjects do *not* usually form a *Boolean algebra*, however, which means we do not have soundness with respect to classical logic, i.e. the law of excluded middle may not be validated by the interpretation in the category.

4.2 Categorical logic in categories of assemblies

We wish to describe the logical side of the category of assemblies. By Section 4.1 this means studying the monomorphisms of $\mathsf{Asm}_{\mathcal{A}}$. As often, it is advisable to first find a more convenient description of the monos in this category. For example, in the category of sets, it is often useful to work with subsets instead of injections, and in the category of presheaves on a category we similarly often choose to work with subpresheaves. Following Bauer [Bau23], we choose the preorder of *realizability predicates* on an assembly X as a convenient substitute for the preorder of monos into X.

Definition 4.1 (Realizability predicate)**.** A **realizability predicate** on an assembly X is a function $|X| \to \mathcal{P}(\mathcal{A})$.

Note that the definition of a realizability predicate makes sense even when X is just a set and not an assembly. However, we will shortly define a perorder on realizability predicates on an assembly X which *does* make essential use of the realizers of X.

Notation 4.2. We will typically write ϕ and ψ for realizability predicates.

We think of a realizability predicate ϕ on X as a logical predicate on X, and of $\phi(x)$ as the set of witnesses that ϕ holds for the element x.

Definition 4.3 (Preorder on realizability predicates, $\phi \preceq \psi$)**.** For two realizability predicates ϕ and ψ on an assembly X, we put $\phi \preceq \psi$ exactly if there exists $\mathtt{r} \in \mathcal{A}$ such that, for every $x \in |X|$, realizer $\mathtt{a} \Vdash_X x$ and witness $\mathtt{b} \in \phi(x)$, we have $\mathtt{r}\,\mathtt{a}\,\mathtt{b} \in \psi(x)$, where we implicitly include the requirement that $\mathtt{r}\,\mathtt{a}\,\mathtt{b}$ is defined.

Intuitively, we have $\phi \preceq \psi$ exactly if we can effectively calculate a witness that ψ holds at x form a witness that ϕ holds at x *and* a realizer of x. It is at this last point that we make essential use of the fact that X is an assembly and not just a set.

It is straightforward (and similar to checking that assemblies with assembly maps form a category) to check that $\preceq$ is indeed a preorder, i.e. that it is reflexive and transitive.

Notation 4.4 (Preorder of realizability predicates, $\mathbb{P}(X)$)**.** We write $\mathbb{P}(X)$ for the preorder of realizability predicates on an assembly X.

Every monomorphism $m\colon Y \hookrightarrow X$ on an assembly X determines a realizability predicate ϕ_m on X by:

$$\begin{aligned}\phi_m(x) &:= \{\mathtt{a} \in \mathcal{A} \mid y \in m^{-1}(x) \text{ and } \mathtt{a} \Vdash_Y y\} \\ &= \{\mathtt{a} \in \mathcal{A} \mid \exists (y \in |Y|).\, \mathtt{a} \Vdash_Y y \text{ and } m(y) = x\}.\end{aligned}$$

Notice that the $y \in |Y|$ is necessarily unique (if it exists), because m is injective.

Conversely, every realizability predicate on X determines a monomorphism of assemblies $[\phi] \hookrightarrow X$ via:

$$\begin{aligned}|[\phi]| &:= \{x \in |X| \mid \phi(x) \neq \emptyset\} \\ \mathbf{pair}\,\mathtt{a}\,\mathtt{b} \Vdash_{[\phi]} x &\iff \mathtt{a} \Vdash_X x \text{ and } \mathtt{b} \in \phi(x),\end{aligned}$$

where the inclusion $[\phi] \hookrightarrow X$ is tracked by **fst**.

Proposition 4.5. *For an assembly X, the above constructions constitute an isomorphism between the preorder of monos into X and the preorder of realizability predicates on X.*

Exercise 4.6. Prove the proposition.

4.2.1 The Heyting prealgebra of realizability predicates

We describe the structure on the preorder of realizability predicates required for interpreting first order logic. The interpretation itself is detailed in Section 4.2.2.

Definition 4.7. There are two extreme examples of realizability predicates on an assembly X:

$$\begin{aligned}\bot(x) &:= \emptyset, \\ \top(x) &:= \mathcal{A}.\end{aligned}$$

For realizability predicates ϕ and ψ on a set X, we define three new realizability predicates on X by:

$$\begin{aligned}(\phi \wedge \psi)(x) &:= \{\mathbf{pair}\,\mathtt{a}\,\mathtt{b} \mid \mathtt{a} \in \phi(x) \text{ and } \mathtt{b} \in \psi(x)\}, \\ (\phi \vee \psi)(x) &:= \{\mathbf{left}\,\mathtt{a} \mid \mathtt{a} \in \phi(x)\} \cup \{\mathbf{right}\,\mathtt{b} \mid \mathtt{b} \in \psi(x)\}, \quad \text{(recall Proposition 3.18)} \\ (\phi \Rightarrow \psi)(x) &:= \{\mathtt{r} \in \mathcal{A} \mid \text{for every } \mathtt{a} \in \phi(x) \text{ we have } \mathtt{r}\,\mathtt{a} \in \psi(x)\}.\end{aligned}$$

Exercise 4.8. Check that $\bot$, $\top$, $\wedge$, $\vee$ and $\Rightarrow$ as defined above make the preorder $\mathbb{P}(X)$ of realizability predicates on an assembly X into a Heyting prealgebra.
(For a short definition of the latter: it is a preorder that, when viewed as a category, has finite (co)limits and exponentials.)

In light of Proposition 4.5 and the fact that the Heyting (pre)algebra operations are characterized by universal properties we know that the operations defined above correspond to the relevant operations on monos. For example, $[\bot]$ is the monomorphism $0 \hookrightarrow X$ and $[\phi \wedge \psi]$ is the meet (= greatest lower bound) of the monos $[\phi]$ and $[\psi]$.

The interpretation of the quantifiers is given by adjoints, as briefly discussed in Section 4.1. We explicitly describe these adjoints in terms of the preorder of realizability predicates, where the natural-bijection-between-hom-sets definition of adjoints simplifies to the condition of a (monotone) *Galois connection*: a map l between preorders is left adjoint to r exactly when $l(x) \leq y \iff x \leq r(y)$ holds.

Proposition 4.9. *For an assembly map $f : X \to Y$ and a realizability predicate ϕ on X, the realizability predicate $\forall_f(\phi)$ on Y defined by*

$$\forall_f(\phi)\,(y) := \{\mathtt{t} \in \mathcal{A} \mid \textit{for every } x \in f^{-1}(y) \textit{ and } \mathtt{a} \Vdash_X x, \textit{ we have } \mathtt{t}\,\mathtt{a} \in \phi(x)\}$$

satisfies

$$f^*(\psi) \leq \phi \iff \psi \leq \forall_f(\phi)$$

for all realizability predicates ψ on Y.
In other words, $\forall_f : \mathbb{P}(X) \to \mathbb{P}(Y)$ is a right adjoint to $f^* : \mathbb{P}(Y) \to \mathbb{P}(X)$.

Proof. We spell out what each of $f^*(\psi) \leq \phi$ and $\psi \leq \forall_f(\phi)$ amounts to.
The former requires the existence of an element $\mathtt{r}_1 \in \mathcal{A}$ such that for every $x \in |X|$, $\mathtt{a} \Vdash_X x$ and $\mathtt{c} \in \psi(f(x))$, we have $\mathtt{r}_1\,\mathtt{a}\,\mathtt{c} \in \phi(x)$.
The latter requires the existence of an element $\mathtt{r}_2 \in \mathcal{A}$ such that for every $y \in |Y|$, $\mathtt{b} \Vdash_Y y$ and $\mathtt{c} \in \psi(y)$ we have $\mathtt{r}_2\,\mathtt{b}\,\mathtt{c} \in \forall_f(\phi)\,(y)$. That is, $\mathtt{r}_2\,\mathtt{b}\,\mathtt{c}$ should satisfy $\mathtt{r}_2\,\mathtt{b}\,\mathtt{c}\,\mathtt{a} \in \phi(x)$ for all $\mathtt{a} \Vdash_X x$ with $f(x) = y$.
Now notice that given such an $\mathtt{r}_1$, the program $\langle vwu\rangle.\ \mathtt{r}_1\,u\,w$ does the job of $\mathtt{r}_2$.
Conversely, given such an $\mathtt{r}_2$, the program $\langle uw\rangle.\ \mathtt{r}_2(\mathtt{t}_f\,u)\,w\,u$, where $\mathtt{t}_f$ is a tracker of f, does the job of $\mathtt{r}_1$. □

For an alternative proof, one may verify that $\forall_f(\phi)$ is the realizability predicate determined by the monomorphism $\prod_f([\phi])$, where we recall $\prod_f$ from Proposition 3.23.

Proposition 4.10. *For an assembly map $f : X \to Y$ and a realizability predicate ϕ on X, the realizability predicate $\exists_f(\phi)$ on Y defined by*

$$\exists_f(\phi)\,(y) := \bigcup_{x \in f^{-1}(y)} \{\mathbf{pair}\,\mathtt{a}\,\mathtt{b} \mid \mathtt{a} \Vdash_X x \textit{ and } \mathtt{b} \in \phi(x)\}$$

satisfies

$$\exists_f(\phi) \leq \psi \iff \phi \leq f^*(\psi)$$

for all realizability predicates ψ on Y.
In other words, $\exists_f : \mathbb{P}(X) \to \mathbb{P}(Y)$ is a left adjoint to $f^* : \mathbb{P}(Y) \to \mathbb{P}(X)$.

Exercise 4.11. Prove the proposition either directly or by checking that $\exists_f(\phi)$ is the

realizability predicate ϕ_m determined by m in the factorization of the top composite

$$\begin{array}{ccccc} [\phi] & \hookrightarrow & X & \xrightarrow{f} & Y \\ & \searrow & & \nearrow_{m} & \\ & & M & & \end{array}$$

as a regular epimorphism followed by a monomorphism (recall Exercise 3.36).

In the situations that will be of us interest to us, f will also be surjective when we consider $\forall_f$ and $\exists_f$. If this is the case, then $\forall_f(\phi)$ admits a definition that is more symmetric to that of $\exists_f(\phi)$, because in this case we have

$$\forall_f(\phi)\,(y) = \bigcap_{x \in f^{-1}(y)} \{\mathtt{t} \in \mathcal{A} \mid \mathtt{t}\,\mathtt{a} \in \phi(x) \text{ for all } \mathtt{a} \Vdash_X x\}$$

for any realizability predicate ϕ on X.

4.2.2 The realizability interpretation of logic

Suppose we are given a formula ϕ in a language whose sorts and relation and function symbols have been assigned an interpretation in $\mathsf{Asm}_{\mathcal{A}}$. If ϕ has free variables $x_1, \ldots, x_n$ of sorts $X_1, \ldots, X_n$, then, following the preceding development, we may interpret ϕ as a realizability predicate on the assembly $[\![X_1]\!] \times \cdots \times [\![X_n]\!]$. We write $[\![\phi]\!]$ for this predicate. Thus, for each $\vec{x} \in [\![X_1]\!] \times \cdots \times [\![X_n]\!]$, we have a subset $[\![\phi]\!](\vec{x}) \subseteq \mathcal{A}$ of realizers.

Using the constructions of Section 4.2.1 we prove the following recursive characterization of membership of such subsets, where we use $\vec{x}_{|_\phi}$ for the restriction of a tuple to those elements that pertain to the domain of $[\![\phi]\!]$ (and similarly for terms).

Proposition 4.12. *The realizability predicates arising from first order logic obey*

$$\begin{array}{lcl}
\mathtt{r} \in [\![s = t]\!](\vec{x}) & \iff & [\![t]\!](\vec{x}_{|_t}) = [\![s]\!](\vec{x}_{|_s}) \\
\mathtt{r} \in [\![\bot]\!](\vec{x}) & \iff & \textit{never}, \\
\mathtt{r} \in [\![\top]\!](\vec{x}) & \iff & \textit{always}, \\
\mathtt{r} \in [\![\phi \wedge \psi]\!](\vec{x}) & \iff & \mathbf{fst}\,\mathtt{r} \in [\![\phi]\!](\vec{x}_{|_\phi}) \textit{ and } \mathbf{snd}\,\mathtt{r} \in [\![\psi]\!](\vec{x}_{|_\psi}) \\
\mathtt{r} \in [\![\phi \vee \psi]\!](\vec{x}) & \iff & \Big(\mathtt{r} = \mathbf{left}\,\mathtt{r'} \textit{ and } \mathtt{r'} \in [\![\phi]\!](\vec{x}_{|_\phi})\Big) \textit{ or} \\
 & & \Big(\mathtt{r} = \mathbf{right}\,\mathtt{r'} \textit{ and } \mathtt{r'} \in [\![\psi]\!](\vec{x}_{|_\psi})\Big), \\
\mathtt{r} \in [\![\phi \Rightarrow \psi]\!](\vec{x}) & \iff & \textit{if } \mathtt{a} \in [\![\phi]\!](\vec{x}_{|_\phi}), \textit{ then } \mathtt{r}\,\mathtt{a} \in [\![\psi]\!](\vec{x}_{|_\psi}), \\
\mathtt{r} \in [\![\forall(x:X).\phi]\!](\vec{x}) & \iff & \textit{if } x \in |[\![X]\!]| \textit{ and } \mathtt{a} \Vdash_{[\![X]\!]} x, \textit{ then } \mathtt{r}\,\mathtt{a} \in [\![\phi]\!](\vec{x}, x), \\
\mathtt{r} \in [\![\exists(x:X).\phi]\!](\vec{x}) & \iff & \textit{there is an } x \in |[\![X]\!]| \textit{ such that} \\
 & & \mathbf{fst}\,\mathtt{r} \Vdash_{[\![X]\!]} x \textit{ and } \mathbf{snd}\,\mathtt{r} \in [\![\phi]\!](\vec{x}, x).
\end{array}$$

A *closed* formula ϕ corresponds to a realizability predicate on $\mathbf{1}$ and may thus be identified with a single subset of $\mathcal{A}$. We say that such a ϕ is **realized**, or **valid in** $\mathsf{Asm}_{\mathcal{A}}$ if we have an element of this subset.

If we take $\mathcal{A} := \mathcal{K}_1$ and consider the language of arithmetic, then we recover Kleene's original *number realizability* [Kle45]. The choice of $\mathcal{A} := \mathcal{K}_2$ and the language of analysis recovers Kleene's *function realizability* [KV65].

Proposition 4.12 may appear to be a formalization of the so-called *Brouwer–Heyting–Kolmogorov (BHK) interpretation* and is often presented as such. However, this is not historically accurate, see [vOos02, p. 241]. It is also worth pointing out that Kleene's realizability predates the Curry–Howard correspondence.

It is worth spelling out the realizability interpretation of (double) negation. Recall that $\neg\phi$ is defined as $\phi \Rightarrow \bot$, so that we have:

Lemma 4.13. *The realizability predicates interpreting (double) negations satisfy*

$$\begin{aligned} \mathtt{r} \in [\![\neg\phi]\!](\vec{x}) &\iff \{\mathtt{r} \in \mathcal{A} \mid [\![\phi]\!](\vec{x}) = \emptyset\}, \\ \mathtt{r} \in [\![\neg\neg\phi]\!](\vec{x}) &\iff \{\mathtt{r} \in \mathcal{A} \mid [\![\phi]\!](\vec{x}) \neq \emptyset\}. \end{aligned}$$

In particular, we see that $[\![\neg\neg\phi]\!](\vec{x})$ has no computational content as any element of $\mathcal{A}$ acts as a realizer whenever $[\![\phi]\!](\vec{x})$ is nonempty. This suggests a connection to the functor $\nabla\colon \mathsf{Set} \to \mathsf{Asm}_{\mathcal{A}}$ (from Section 3.3) which we indeed explore in Section 4.3.1.

We repeat that the realizability interpretation of first order logic—by virtue of the Heyting prealgebra structure—validates constructive logic, i.e. first order logic without excluded middle. In fact, as Exercise 4.16 shows, the logic governing realizability predicates is never classical unless the pca is trivial (in which case the category of assemblies is the familiar category of sets).

4.2.3 Revisiting (regular) epis and monos

A nice way of getting some familiarity with realizability logic is by proving the following characterizations of (regular) epis and monos in the internal logic of $\mathsf{Asm}_{\mathcal{A}}$. We recall our convention to use the same letters for both the formal symbol in the internal language and its interpretation in the category, e.g. if f is an assembly map, then we formally have a function symbol $\ulcorner f \urcorner$ in our language with interpretation $[\![\ulcorner f \urcorner]\!] := f$; but we'll simply reuse the letter f for this function symbol.

To appreciate Exercise 4.14, recall that an assembly map is an epi if and only if it's surjective, while it's regular epi if and only if it's surjective *and* we have an element in $\mathcal{A}$ that witnesses surjectivity (Exercise 3.33). This difference in computational content is reflected in the (non)use of the double negation in the first items of the exercise below.

Exercise 4.14. Use Exercises 3.33 and 3.37 to show that for an assembly map $f\colon X \to Y$, we have the following logical characterizations:

(i) f is a regular epimorphism if and only if

$$\forall(x : X).\exists(y : Y).f(x) = y$$

is realized;

(ii) f is an epimorphism if and only if

$$\forall(x : X).\neg\neg(\exists(y : Y).f(x) = y)$$

is realized;

(iii) f is a monomorphism if and only if

$$\forall(x, x' : X).(f(x) = f(x') \Rightarrow x = x')$$

is realized;

(iv) f is a regular monomorphism if and only if

$$\forall(y : Y).(\neg\neg(\exists(x : X).f(x) = y) \Rightarrow \exists!(x : X).f(x) = y)$$

is realized.

The quantifier $\exists!$ means "there exists a unique ... with ...".

Phrased in English, f is a regular monomorphism if and only if the statement

> "For all y, if the primage of f at y is nonempty, then we can (effectively) find a unique x with $f(x) = y$."

is realized.

4.3 Two-element assemblies as classifiers

This section introduces three sets of realizability predicates, namely the $\neg\neg$-*stable*, *decidable* and *semidecidable* predicates. These are shown to be *classified* by three different assemblies all of which have the set $\{0, 1\}$ as their carriers, but different realizers. For example, the $\neg\neg$-stable realizability predicates are classified by $\nabla\{0, 1\}$, while the assembly $\mathbf{2}$ of booleans classifies the decidable realizability predicates. We moreover give explicit connections to computability theory by specializing to the category of assemblies over Kleene's first model.

4.3.1 Double negation stable realizability predicates

We have already seen that the monomorphisms of assemblies are given exactly by realizability predicates. The *regular* monos can be characterized as a subset of those realizability predicates, namely those that are $\neg\neg$-stable.

Definition 4.15 ($\neg\neg$-stability)**.** A realizability predicate ϕ on an assembly X is said to be **$\neg\neg$-stable** if

$$\forall(x : X).(\neg\neg\phi(x) \Rightarrow \phi(x))$$

is realized.

In some of the literature (on topos theory), one also sees the name $\neg\neg$-*closed*. Bauer uses the word *classical* in [Bau23]. This is reasonable terminology, because in classical logic everything is $\neg\neg$-stable. Moreover, as we will see the $\neg\neg$-stable realizability predicates correspond to ordinary 'classical' subsets.

In realizability, not all predicates are $\neg\neg$-stable, as we ask you to verify by means of the following exercise:

Exercise 4.16. Show that if all realizability predicates of $\mathsf{Asm}_{\mathcal{A}}$ are $\neg\neg$-stable, then the pca $\mathcal{A}$ is trivial.
Hint: For $\mathrm{a}, \mathrm{b} \in \mathcal{A}$, consider a suitable realizability predicate on $\nabla\{0,1\}$.

In fact, we have already seen the $\neg\neg$-stable realizability predicates because (seen as monomorphisms) they are precisely the regular monos, as we ask you to verify.

Exercise 4.17. Prove that a realizability predicate ϕ on an assembly X is $\neg\neg$-stable if and only if its corresponding monomorphism $[\phi] \hookrightarrow X$ is regular.

The $\neg\neg$-stable predicates have no computational content as made precise by the following result:

Proposition 4.18. *Every $\neg\neg$-stable realizability predicate ϕ on an assembly X is uniquely determined by a subset $A \subseteq |X|$ such that we have a pullback diagram*

$$\begin{array}{ccc} [\phi] & \longrightarrow & \nabla A \\ \downarrow & \lrcorner & \downarrow \\ X & \xrightarrow{\eta_X} & \nabla|X| \end{array}$$

in $\mathsf{Asm}_{\mathcal{A}}$.

Proof. Given a $\neg\neg$-stable realizability predicate ϕ on X, we define

$$A := \{x \in |X| \mid \phi(x) \neq \emptyset\} = |[\phi]|.$$

We may compute the pullback of $\nabla A \hookrightarrow \nabla|X|$ along η_X as the assembly P with

$$|P| := A \quad \text{and} \quad \mathrm{a} \Vdash_P x \iff a \Vdash_X x.$$

The identity on A gives functions between $|[\phi]|$ and $|P|$. It remains to see that they are tracked. Towards P, the map is tracked by **fst**. In the other direction, we get a tracker by the assumption that ϕ is $\neg\neg$-stable.
For the converse, we note that the above computation indeed shows that such a pullback corresponds to a $\neg\neg$-stable realizability predicate, because the realizers of the pullback are just the realizers of X. □

In fact, the $\neg\neg$-stable realizability predicates arise as pullbacks of a single map:

Proposition 4.19. *Every $\neg\neg$-stable realizability predicate ϕ on an assembly X is uniquely determined by a map $\chi\colon X \to \nabla\{0,1\}$ such that we have a pullback diagram*

$$\begin{array}{ccc} [\phi] & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star \mapsto 1} \\ X & \xrightarrow{\chi} & \nabla\{0,1\} \end{array}$$

in $\mathsf{Asm}_{\mathcal{A}}$.

Proof. Given $A \subseteq |X|$, the map $\chi' \colon \nabla X \to \nabla\{0,1\}$ with $\chi'(x) = \begin{cases} 0 & \text{if } x \notin A \\ 1 & \text{if } x \in A \end{cases}$ gives a pullback square

$$\begin{array}{ccc} \nabla A & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star \mapsto 1} \\ \nabla|X| & \xrightarrow{\chi'} & \nabla\{0,1\} \end{array}$$

so the result follows from Proposition 4.18 and pullback pasting. □

The map $\mathbf{1} \to \nabla\{0,1\}$ is said to be a **classifier** for the $\neg\neg$-stable realizability predicates (subobjects).

Finally, we can also internalize Proposition 4.18 as:

Exercise 4.20. For an assembly X, the exponential $(\nabla\{0,1\})^X$ is isomorphic to $\nabla(\mathcal{P}(|X|))$.

4.3.2 Decidable realizability predicates

After the $\neg\neg$-stable realizability predicates we now consider the subset of *decidable* realizability predicates. We show these to be classified by the assembly of booleans and explore the connection to computable subsets in assemblies over Kleene's first model.

Definition 4.21 (Decidability)**.** A realizability predicate ϕ on an assembly X is said to be **decidable** if

$$\forall(x : X).(\phi(x) \vee \neg\phi(x))$$

is realized.

The decidable realizability predicates form a subset of the $\neg\neg$-stable ones.

Lemma 4.22. *Every decidable realizability predicate is $\neg\neg$-stable.*

Proof. This is a nice opportunity to make use of the soundness of constructive logic, so that we don't need to concern ourselves with realizers. That is, we give a constructive proof that decidability implies $\neg\neg$-stability. If ϕ is decidable, then we only have to consider two cases: if $\phi(x)$ holds, then we trivially get $\neg\neg\phi(x) \implies \phi(x)$; while if $\neg\phi(x)$ hold, then the assumption $\neg\neg\phi(x)$ leads to a contradiction, allowing us to conclude $\phi(x)$. □

Notice that Lemma 4.22 in combination with Exercise 4.16 implies that not all realizability predicates are decidable (unless the pca is trivial).

Proposition 4.23. *The decidable realizability predicates are classified by* $\mathbf{2}$.

Proof. Suppose ϕ is a decidable realizability predicate on an assembly X. Then the function $\chi \colon |X| \to |\mathbf{2}|$ defined as

$$\chi(x) \coloneqq \begin{cases} 0 & \text{if } \phi(x) = \emptyset, \\ 1 & \text{if } \phi(x) \neq \emptyset; \end{cases}$$

is tracked because ϕ is decidable, so we get an assembly map $\chi \colon X \to \mathbf{2}$. Computing the pullback P in

$$\begin{array}{ccc} P & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star \mapsto 1} \\ X & \xrightarrow{\chi'} & \mathbf{2} \end{array}$$

we get

$$|P| \coloneqq \{x \in |X| \mid \phi(x) \neq \emptyset\} = |[\phi]| \quad \text{and} \quad a \Vdash_P x \iff a \Vdash_X x.$$

But, recalling the proof of Proposition 4.18, we see that P and $[\phi]$ are isomorphic as ϕ is $\neg\neg$-stable by Lemma 4.22.
Conversely, given such a pullback $[\phi]$, the tracker of $X \to \mathbf{2}$ witnesses the fact that $\forall(x : X).\neg\neg\phi(x) \vee \neg\phi(x)$ is realized. But the square

$$\begin{array}{ccc} \mathbf{1} & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star \mapsto 1} \\ \mathbf{2} & \hookrightarrow & \nabla\{0,1\} \end{array}$$

is a pullback, so by pullback pasting we see that $[\phi]$ is classified by $\nabla\{0,1\}$. Therefore, by Proposition 4.19, the predicate ϕ is $\neg\neg$-stable, so in fact $\forall(x : X).\phi(x) \vee \neg\phi(x)$ is realized, as desired. □

For the remainder of this subsection we take $\mathcal{A} \coloneqq \mathcal{K}_1$, i.e., we work with assemblies over Kleene's first model (Example 2.5).

Exercise 4.24. Show that the natural numbers object in $\mathsf{Asm}_{\mathcal{K}_1}$ is isomorphic to the assembly $\mathbf{N}$ with carrier $\mathbb{N}$ and realizers $n \Vdash_{\mathbf{N}} n$ for each $n \in \mathbb{N}$.

Similarly, one may show that in $\mathsf{Asm}_{\mathcal{K}_1}$ we can take the numbers 0 and 1 as the respective realizers of the elements $0, 1 \in |\mathbf{2}|$ of the assembly of booleans.

Recall from computability theory that a subset $A \subseteq \mathbb{N}$ is **computable** if we have a total (Turing) computable function $\chi \colon \mathbb{N} \to \mathbb{N}$ such that $\chi(n) = 1 \iff n \in A$. We say that χ **computes** A. Note: instead of "computable", some (older) textbooks will use the terminology "recursive", or (potentially confusing for us) "decidable".

The following exercises show that the decidable realizability predicates of $\mathsf{Asm}_{\mathcal{K}_1}$ correspond precisely to computable subsets.

Exercise 4.25.

(i) Show that the exponential $\mathbf{2}^{\mathbf{N}}$ is isomorphic to the assembly $\mathbf{C}$ with

$$|\mathbf{C}| \coloneqq \{A \subseteq \mathbb{N} \mid A \text{ is computable}\}, \text{ and}$$
$$n \Vdash_{\mathbf{C}} A \iff \varphi_n \text{ computes } A.$$

(ii) Show that there is a bijection between computable subsets and pullback squares

$$\begin{array}{ccc} \bullet & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star\mapsto 1} \\ \mathbf{N} & \longrightarrow & \mathbf{2} \end{array}$$

(iii) Conclude that there is a bijection between computable subsets and decidable realizability predicates on $\mathbf{N}$.

4.3.3 Semidecidable realizability predicates

We introduce a third and final class of realizability predicates: the semidecidable ones.

Definition 4.26 (Semidecidability)**.** A realizability predicate ϕ on an assembly X is said to be **semidecidable** if

$$\forall(x : X).\exists(\sigma : 2^{\mathbf{N}}).((\exists(n : \mathbf{N}).\sigma(n) = 1) \iff \phi(x))$$

is realized.

Notice the inherent essential asymmetry of semidecidability: the assertion that the predicate is true can be made by making finitely many observations (keep querying the sequence until we see a 1); on the other hand, to conclude that the predicate is false we would need evidence that the sequence is 0 *everywhere*.

Definition 4.27 (Assembly of semidecidable truth values, Σ)**.** The **assembly of semidecidable truth values**, denoted by Σ, has carrier $\{0, 1\}$ and realizers

$$\mathtt{r} \Vdash_{\Sigma} b \text{ such that } \mathtt{r}\,\overline{\mathtt{k}} \in \{\mathbf{true}, \mathbf{false}\} \text{ for all } k \in \mathbb{N}, \text{ and we have}$$
$$(\exists(n \in \mathbb{N}).\, \mathtt{r}\,\overline{\mathtt{n}} = \mathbf{true}) \iff (b = 1).$$

Thus, the realizers of Σ are codes for binary sequences of booleans and such a code for a sequence realizes the element 1 precisely when the sequence is true somewhere. It should therefore come as no surprise that we have:

Exercise 4.28. The assembly Σ (with distinguished element $1 \in |\Sigma|$) classifies the semidecidable realizability predicates.

For the remainder of this subsection we again work with Kleene's first model only.

Exercise 4.29. Show that Σ is isomorphic to the assembly Σ' with carrier $\{0, 1\}$ and realizers

$$n \Vdash_{\Sigma'} 0 \iff n \notin K \quad \text{and} \quad n \Vdash_{\Sigma'} 1 \iff n \in K,$$

where $K := \{n \in \mathbb{N} \mid \varphi_n(n) \text{ is defined}\}$ is the *(diagonal) Halting set.*
Note: This requires a little bit of computability theory.

It follows from Exercise 4.29 that while both inclusions

$$2 \hookrightarrow \Sigma \hookrightarrow \nabla\{0, 1\}$$

are mono and epi, neither of them is regular mono or regular epi, as it would imply (check!) that membership of the Halting set K is computable which it (famously) isn't.

Recall from computability theory that a subset $A \subseteq \mathbb{N}$ is **computably enumerable** (or **c.e.** for short) if we have a partial computable function $e\colon \mathbb{N} \to \mathbb{N}$ such that $e(n)$ is defined if and only if $n \in A$. We say that e **enumerates** A. Note: instead of "computably enumerable", some (older) textbooks will use the terminology "recursively enumerable", or (potentially confusing for us) "semidecidable". A standard example of a computably enumerable subset that is not computable is the Halting set K.

The following exercises explore the connections between the semidecidable realizability predicates of $\mathsf{Asm}_{\mathcal{K}_1}$ and computably enumerable subsets.

Exercise 4.30. Show that the exponential $\Sigma^{\mathbf{N}}$ is isomorphic to the assembly CE with

$$|\mathsf{CE}| \coloneqq \{A \subseteq \mathbb{N} \mid A \text{ is computably enumerable}\}, \text{ and}$$
$$n \Vdash_{\mathsf{CE}} A \iff \varphi_n \text{ enumerates } A.$$

Exercise 4.31. Show that for an assembly X, we have a bijection between

(i) pullbacks

$$\begin{array}{ccc} \bullet & \longrightarrow & \mathbf{1} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \star \mapsto 1} \\ X & \longrightarrow & \Sigma \end{array}$$

(ii) subsets $X' \subseteq X$ and c.e. subsets $A \subseteq \mathbb{N}$ such that for all $x \in X$, we have

$$x \in X' \Rightarrow \{n \in \mathbb{N} \mid n \Vdash_X x\} \subseteq A, \text{ and}$$
$$x \notin X' \Rightarrow \{n \in \mathbb{N} \mid n \Vdash_X x\} \cap A = \emptyset.$$

In particular, taking $X \coloneqq \mathbf{N}$, we see that semidecidable realizability predicates on $\mathbf{N}$ correspond to computably enumerable subsets.

Recall **Rice's Theorem** from computability theory:

Theorem (Rice). *Suppose that P is a subset of $\mathbb{N}$ such that*

(i) P is an index set*: if $n \in P$ and $\varphi_n = \varphi_m$, then $m \in P$;*

(ii) P is nontrivial*, i.e. $P \neq \emptyset$ and $P \neq \mathbb{N}$.*

Then P is not computable.

Rice's Theorem is often informally phrased as: "every nontrivial semantic property of partial computable functions is undecidable".

In the category $\mathsf{Asm}_{\mathcal{K}_1}$ it has the following incarnation:

Exercise 4.32. Show that the exponential $2^{\mathsf{CE}} \cong 2^{(\Sigma^{\mathbf{N}})}$ is isomorphic to $\mathbf{2}$.

4.4 Very first steps in synthetic computability theory

We end this chapter by giving an example of *synthetic computability theory* [Bau06]. At a high level, the idea is to use the internal logic of $\mathsf{Asm}_{\mathcal{K}_1}$ to develop computability theory[3]. The point of restricting to the internal logic is that everything is automatically computable: you never need to check computability or reason explicitly about Turing machines for example. A downside (depending on your perspective) is that we have to give up on using classical logic, because it is not valid in the category as we have seen.

In computability theory, a basic but fundamental result is the following:

Theorem (Post). *If $A \subseteq \mathbb{N}$ and its complement $\mathbb{N} \setminus A$ are computably enumerable, then A is computable.*

It is not hard to prove this theorem, but we use it here as an illustration of what a synthetic development might look like.

Theorem 4.33 (Post's theorem, synthetically)**.** *For a realizability predicate ϕ on an assembly $X \in \mathsf{Asm}_{\mathcal{K}_1}$, if ϕ and $\neg\phi$ are semidecidable, then ϕ is decidable.*

We prove Theorem 4.33 via two general lemmas. However, the final argument will need one additional logical axiom, namely Markov's Principle, that is not provable in plain constructive logic, but that it is valid in the internal logic of $\mathsf{Asm}_{\mathcal{K}_1}$.

Definition 4.34 (Markov's Principle)**.** The statement that every binary sequence that is not 0 everywhere must contain a 1 is known as **Markov's Principle**. More formally, it is the statement:

$$\forall\left(\sigma : 2^{\mathbf{N}}\right).\neg(\forall(n : \mathbf{N}).\sigma(n) = 0) \to (\exists(n : \mathbf{N}).\sigma(n) = 1).$$

Exercise 4.35.

(i) Show that Markov's Principle is equivalent—over constructive logic—to

$$\forall\left(\sigma : 2^{\mathbf{N}}\right).\neg\neg(\exists(n : \mathbf{N}).\sigma(n) = 1) \to (\exists(n : \mathbf{N}).\sigma(n) = 1).$$

(ii) Show that Markov's Principle is realized if and only if every semidecidable realizability predicate is $\neg\neg$-stable.

(iii) Show that Markov's Principle is realized over $\mathcal{K}_1$.

As announced, we now prove two general lemmas. Note that the proofs make no mention of computability and simply restrict to constructively sound reasoning.

Lemma 4.36. *If ϕ and ψ are semidecidable realizability predicates on an assembly X, then $\phi \vee \psi$ is again semidecidable.*

Proof. Let $x \in |X|$ be arbitrary. We reason purely in constructive logic. Suppose there

[3] Really, the internal logic of the realizability topos; see Chapter 5.

exist binary sequences σ and τ such that

$$(\exists(n:\mathbb{N}).\sigma(n)=1) \iff \phi(x) \quad \text{and} \quad (\exists(n:\mathbb{N}).\tau(n)=1) \iff \psi(x).$$

Then

$$(\exists(n:\mathbb{N}).(\sigma\oplus\tau)(n)=1) \iff \phi(x)\vee\psi(x),$$

where $\sigma\oplus\tau$ is the binary sequence obtained by taking the maximum of the outputs of σ and τ at each index. □

Lemma 4.37. *For any realizability predicate ϕ, we have that $\neg\neg(\phi\vee\neg\phi)$ and $\top$ are equivalent in the preorder of realizability predicates.*

Proof. If ϕ is a realizability predicate on an assembly X, then it suffices to show that $\forall(x:X).\neg\neg(\phi(x)\vee\neg\phi(x))$ is realized. We show that $\neg\neg(p\vee\neg p)$ holds generally in constructive logic. Assume $\neg(p\vee\neg p)$ with the aim of deriving a contradiction. Since p implies $p\vee\neg p$, we derive $\neg p$. But then $p\vee\neg p$ holds again which contradicts our assumption. □

We are now ready to prove Theorem 4.33:

Proof of Theorem 4.33. Suppose that ϕ and $\neg\phi$ are semidecidable. By Lemma 4.36 we know that $\phi\vee\neg\phi$ is again semidecidable. Moreover, by Markov's Principle, the predicate $\phi\vee\neg\phi$ is $\neg\neg$-stable. But $\forall(x:X).\neg\neg(\phi(x)\vee\neg\phi(x))$ is true by Lemma 4.37, so we get $\forall(x:X).\phi(x)\vee\neg\phi(x)$, i.e. ϕ is decidable, as desired. □

Once again, we stress the purely logical flavour of the above arguments—with the exception of checking that Markov's Principle is valid in $\mathsf{Asm}_{\mathcal{K}_1}$ which should be done once and can then be taken as an additional axiom to the synthetic development.

Finally, we recover Post's result by specializing Theorem 4.33 to the assembly of natural numbers and by exploiting the correspondence between the (semi)decidable realizability predicates and computable/c.e. subsets as explored in Sections 4.3.2 and 4.3.3.

4.5 List of exercises

1. Exercise 4.6: On monomorphisms as realizability predicates and vice versa.
2. Exercise 4.8: On the Heyting prealgebra of realizability predicates.
3. Exercise 4.11: On $\exists_f$ being left adjoint to f^* as maps between preorders of realizability predicates.
4. Exercise 4.14: On logical characterizations of (regular) epis and monos.
5. Exercise 4.16: On the fact that not all realizability predicates are $\neg\neg$-stable.
6. Exercise 4.17: On the $\neg\neg$-stable realizability predicates as the regular monos.
7. Exercise 4.20: On exponentials of $\nabla\{0,1\}$.
8. Exercise 4.24: On the natural numbers object in $\mathsf{Asm}_{\mathcal{K}_1}$.

9. Exercise 4.25: On the correspondence between decidable realizability predicates in $\mathsf{Asm}_{\mathcal{K}_1}$ and computable subsets of the natural numbers.
10. Exercise 4.28: On the classifier of semidecidable realizability predicates.
11. Exercise 4.29: On the assembly of semidecidable truth values in $\mathsf{Asm}_{\mathcal{K}_1}$.
12. Exercise 4.30: On computably enumerable subsets and the exponential of the assemblies of natural numbers and semidecidable truth values in $\mathsf{Asm}_{\mathcal{K}_1}$.
13. Exercise 4.31: On computably enumerable subsets and pullbacks of the assembly of semidecidable truth values in $\mathsf{Asm}_{\mathcal{K}_1}$.
14. Exercise 4.32: On Rice's theorem in $\mathsf{Asm}_{\mathcal{K}_1}$.
15. Exercise 4.35: On Markov's Principle.

Chapter 5

Epilogue: towards realizability toposes

In this epilogue we briefly discuss what is perhaps the elephant in the room (pun intended[1]): the **realizability topos** associated to a pca. Indeed, the notion of a realizability topos is noticeably absent from these notes, essentially because its construction is relatively complicated compared to the category of assemblies.

Working with an (elementary) topos has some benefits of course. The primary feature of a topos is the existence of a classifier $\top\colon 1 \to \Omega$ for *all* monomorphisms—the category of assemblies only has classifiers for special subclasses (e.g. $\neg\neg$-stable or decidable monos) as we have seen. A consequence is that every monomorphism is regular (because any mono into X is the equalizer of its characteristic map $\chi\colon X \to \Omega$ and the composite $X \to \mathbf{1} \xrightarrow{\top} \Omega$). Thus, while regular monos and monos are distinct in the category of assemblies, the passage to the realizability topos unifies these two concepts.

Here we briefly sketch the construction of the realizability topos over a pca. The objects of the realizability topos are **partial equivalence relations**: sets X equipped with a $\mathcal{P}(\mathcal{A})$-valued predicate $\sim : X \times X \to \mathcal{P}(\mathcal{A})$ such that transitivity and symmetry of $\sim$ are "realized". For the latter we set up a Heyting prealgebra of $\mathcal{P}(\mathcal{A})$-valued predicates similar to that of the realizability predicates in Section 4.2 with the notable exception that we now work with bare sets, so the elements of our sets are not equipped with realizers. In fact, while the relation $\sim$ is not assumed to be reflexive in general, elements of $x \sim x$ are taken as realizers of x and evidence that x "exists". In general, we think of $\sim$ as an equality predicate on X.

A morphism between such pairs $(X, \sim_X)$ and $(Y, \sim_Y)$ is defined to be a $\mathcal{P}(\mathcal{A})$-valued predicate F on $X \times Y$ such that the statement that F is a *strict* functional relation are realized. Strictness means $\forall(x \in X).\forall(y \in Y).(F(x,y) \Rightarrow x \sim_X x \wedge y \sim_Y y)$, that is, F

[1] For the uninitiated: Johnstone's two volume compendium on topos theory is titled *Sketches of an Elephant: A Topos Theory Compendium* and commonly referred to as "the Elephant". A third volume, covering realizability toposes was planned but has yet to be published.

relates only elements that "exist". Moreover, F should respect the equality predicates, i.e. $\forall(x, x' \in X).\forall(y, y' \in Y).(F(x, y) \wedge (x \sim_X x') \wedge (y \sim_Y y') \Rightarrow F(x', y'))$ must be realized. Finally, F should be a *functional* relation, so we also require single-valuedness and totality, in the sense that $\forall(x \in X).\forall(y, y' \in Y).(F(x, y) \wedge F(x, y') \to y \sim_Y y')$ and $\forall(x \in X).(x \sim_X x \to \exists(y \in Y).F(x, y))$ are realized.

One can show that the resulting category is indeed a topos and that it houses the category of assemblies as a full subcategory. In fact, the category of assemblies is precisely the subcategory of $\neg\neg$-separated objects of the realizability topos, i.e. those objects X for which $\forall(x, y : X).(\neg\neg(x = y) \to (x = y))$ holds in the internal logic. This is perhaps foreseen by the "classical" interpretation of equality in the realizability logic of assemblies (Proposition 4.12).

The realizability topos over Kleene's first model is known as **the effective topos** as was detailed in Hyland's landmark paper [Hyl82]. The more general theory, known as **tripos theory**, was the subject of Pitts's PhD thesis [Pit81] and his joint paper with Hyland and Johnstone [HJP80]. The interested reader can consult van Oosten's textbook [vOos08], Streicher's notes [Str18], or Zoethout's master's thesis [Zoe18].

Bibliography

[Abo14] AboAmmar. *Add text to Penrose triangle*. TEX StackExchange answer. Aug. 31, 2014. URL: https://tex.stackexchange.com/a/198782 (cit. on p. iii).

[AC98] Roberto M. Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998. DOI: 10.1017/cbo9780511983504 (cit. on pp. 3, 10).

[Bau00] Andrej Bauer. "The Realizability Approach to Computable Analysis and Topology". PhD thesis. Carnegie Mellon University, 2000. Available as technical report CMU-CS-00-164 (cit. on p. 9).

[Bau06] Andrej Bauer. "First Steps in Synthetic Computability Theory". In: *Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI)*. Ed. by M. Escardó, A. Jung, and M. Mislove. Vol. 155. Electronic Notes in Theoretical Computer Science. 2006, pp. 5–31. DOI: 10.1016/j.entcs.2005.11.049 (cit. on pp. 13, 23, 37).

[Bau12] Andrej Bauer. *Substitution is pullback*. Blog post. Sept. 28, 2012. URL: https://math.andrej.com/2012/09/28/substitution-is-pullback/ (cit. on p. 17).

[Bau23] Andrej Bauer. "Notes on realizability". Lecture notes. May 6, 2023. URL: https://www.andrej.com/zapiski/MGS-2022/notes-on-realizability.pdf (cit. on pp. 2–5, 9–10, 13, 21, 23, 26, 31).

[BMSS11] Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger. "Minlog - A Tool for Program Extraction Supporting Algebras and Coalgebras". In: *Algebra and Coalgebra in Computer Science: 4th International Conference, CALCO 2011. Winchester, UK, August/September 2011. Proceedings*. Ed. by Andrea Corradini, Bartek Klin, and Corina Cîrstea. Vol. 6859. Lecture Notes in Computer Science. Springer, 2011, pp. 393–399. DOI: 10.1007/978-3-642-22944-2_29 (cit. on p. 1).

[Chh23] Rahul Chhabra. *Experiments with Realizability in Univalent Type Theory*. https://rahulc29.github.io/realizability/.
Cubical Agda development accompanying an upcoming BTech thesis. 2023. URL: https://github.com/rahulc29/realizability (cit. on p. 3).

[dJon18] Tom de Jong. "Realizability with Scott's Graph Model".
Master's thesis. Utrecht University, 2018. DOI: 20.500.12932/29734 (cit. on p. 10).

[dJon23] Tom de Jong. "Domain Theory and Denotational Semantics".
Lecture notes and exercises for the Midlands Graduate School (MGS), 2–6 April 2023, Birmingham, UK. 2023. URL: https://github.com/tomdjong/MGS-domain-theory/blob/main/README.md (cit. on p. 4).

[Esc04] Martín Escardó. "Synthetic Topology: of Data Types and Classical Spaces".
In: *Proceedings of the Workshop on Domain Theoretic Methods for Probabilistic Processes (DTMPP 2003)*.
Ed. by J. Desharnais and P. Panangaden. Vol. 87.
Electronic Notes in Theoretical Computer Science. 2004, pp. 21–156.
DOI: 10.1016/j.entcs.2004.09.017 (cit. on p. 9).

[Fav24] Naïm Favier. *Answer to the Mathematics Stack Exchange post titled "In a CCC with the initial object, can I have a morphism A -> 0?"* Mar. 8, 2024.
URL: https://math.stackexchange.com/questions/3977129/in-a-ccc-with-the-initial-object-can-i-have-a-morphism-a-0/4877251#4877251 (cit. on p. 24).

[FS90] Peter J. Freyd and Andre Scedrov. *Categories, Allegories*. Vol. 39.
North-Holland Mathematical Library. Elsevier, 1990.
DOI: 10.1016/s0924-6509(08)x7003-5 (cit. on p. 23).

[HJP80] J. M. E. Hyland, P. T. Johnstone, and A. M. Pitts. "Tripos theory".
In: *Mathematical Proceedings of the Cambridge Philosophical Society* 88.2 (1980), pp. 205–232. DOI: 10.1017/S0305004100057534 (cit. on pp. 1, 41).

[Hyl82] J. M. E. Hyland. "The Effective Topos".
In: *The L. E. J. Brouwer Centenary Symposium: Proceedings of the Conference held in Noordwijkerhout, 8-13 June, 1981*. Vol. 110.
Studies in Logic and the Foundations of Mathematics.
North-Holland Publishing Company, 1982, pp. 165–216.
DOI: 10.1016/S0049-237X(09)70129-6 (cit. on p. 41).

[Jac99] Bart Jacobs. *Categorical Logic and Type Theory*. Vol. 141.
Studies in Logic and the Foundations of Mathematics. Elsevier, 1999.
DOI: 10.1016/s0049-237x(98)x8028-6 (cit. on pp. 3, 17).

[Joh02] Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium*.
2 vols. Oxford Logic Guides. Oxford University Press, 2002 (cit. on p. 40).

[Kle45] S. C. Kleene. "On the interpretation of intuitionistic number theory".
In: *Journal of Symbolic Logic* 10.4 (1945), pp. 109–124.
DOI: 10.2307/2269016 (cit. on pp. 1, 5, 30).

[KV65] Stpehen Cole Kleene and Richard Eugene Vesley. *The Foundations of Intuitionistic Mathematics: Especially in relation to recursive functions.* Vol. 39. Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, 1965. DOI: 10.1016/s0049-237x(08)x7064-8 (cit. on pp. 10, 30).

[Law63] F. William Lawvere. "Functorial Semantics of Algebraic Theories and Some Algebraic Problems in the context of Functorial Semantics of Algebraic Theories". PhD thesis. Columbia University, 1963. URL: http://www.tac.mta.ca/tac/reprints/articles/5/tr5abs.html. Republished in: *Reprints in Theory and Applications of Categories*, No. 5 (2004), pp. 1–121 (cit. on p. 16).

[Law69] F. William Lawvere. "Adjointness in Foundations". In: *dialectica* 23.3–4 (1969), pp. 281–296. DOI: 10.1111/j.1746-8361.1969.tb01194.x (cit. on p. 25).

[Lie04] Peter Lietz. "From Constructive Mathematics to Computable Analysis via the Realizability Interpretation". PhD thesis. Technischen Universität Darmstadt, 2004. URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/528 (cit. on p. 9).

[LN15] John Longley and Dag Normann. *Higher-Order Computability.* Theory and Applications of Computability. Springer, 2015. DOI: 10.1007/978-3-662-47992-6 (cit. on p. 3).

[Lon95] John R. Longley. "Realizability Toposes and Language Semantics". PhD thesis. University of Edinburgh, 1995. DOI: 1842/402. Available as technical report ECS-LFCS-95-332 (cit. on pp. 4, 21).

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic.* Vol. 7. Cambridge studies in advanced mathematics. Cambridge University Press, 1986 (cit. on p. 23).

[McL92] Colin McLarty. *Elementary Categories, Elementary Toposes.* Oxford University Press, 1992. DOI: 10.1093/oso/9780198533924.001.0001 (cit. on p. 24).

[MM94] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Universitext. Springer, 1994. DOI: 10.1007/978-1-4612-0927-0 (cit. on p. 23).

[Pit81] Andrew Mawdesley Pitts. "The Theory of Triposes". PhD thesis. University of Cambridge, 1981. URL: https://www.cl.cam.ac.uk/~amp12/papers/thet/thet.pdf (cit. on pp. 1, 41).

[Plo77] G. D. Plotkin. "LCF considered as a programming language". In: *Theoretical Computer Science* 5.3 (1977), pp. 223–255. DOI: 10.1016/0304-3975(77)90044-5 (cit. on p. 4).

[Reu99] Bernhard Reus. "Realizability Models for Type Theories". In: *Tutorial Workshop on Realizability Semantics and Applications (associated to FLoC'99, the 1999 Federated Logic Conference)*. Ed. by Lars Birkedal, Jaap van Oosten, Giuseppe Rosolini, and Dana S. Scott. Vol. 23. Electronic Notes in Theoretical Computer Science 1. Elsevier, 1999, pp. 128–158. DOI: 10.1016/s1571-0661(04)00108-2 (cit. on p. 3).

[Rey84] John C. Reynolds. "Polymorphism is not set-theoretic". In: *Semantics of Data Types: International Symposium. Sophia-Antipolis, France, June 1984. Proceedings.* Ed. by Gilles Kahn, David B. MacQueen, and Gordon Plotkin. Vol. 173. Lecture Notes in Computer Science. Springer-Verlag, 1984, pp. 145–156. DOI: 10.1007/3-540-13346-1_7 (cit. on p. 1).

[Sch24] M. Schönfinkel. "Über die Bausteine der mathematischen Logik". In: *Mathematische Annalen* 92.3 (1924), pp. 305–316. DOI: 10.1007/BF01448013 (cit. on p. 5).

[Sco72] Dana Scott. "Continuous lattices". In: *Toposes, Algebraic Geometry and Logic*. Ed. by F. W. Lawvere. Vol. 274. Lecture Notes in Mathematics. Springer, 1972, pp. 97–136. DOI: 10.1007/BFB0073967 (cit. on p. 10).

[Sco76] Dana Scott. "Data Types as Lattices". In: *SIAM Journal on Computing* 5.3 (1976), pp. 522–587. DOI: 10.1137/0205037 (cit. on p. 9).

[See84] R. A. G. Seely. "Locally cartesian closed categories and type theory". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 95.1 (1984), pp. 33–48. DOI: 10.1017/s0305004100061284 (cit. on p. 17).

[Smy92] M. B. Smyth. "Topology". In: *Background: Mathematical structures*. Ed. by S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum. Vol. 1. Handbook of Logic in Computer Science. Clarendon Press, 1992, pp. 641–762 (cit. on p. 9).

[Str04] Thomas Streicher. "Introduction to CATEGORY THEORY and CATEGORICAL LOGIC". Lecture notes. 2004. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/CTCL.pdf (cit. on p. 23).

[Str18] Thomas Streicher. "Realizability". Lecture notes. 2018. URL: https://www2.mathematik.tu-darmstadt.de/~streicher/REAL/REAL.pdf (cit. on pp. 2–3, 17, 41).

[Str91] Thomas Streicher. *Semantics of Type Theory*. Birkhäuser, 1991. DOI: 10.1007/978-1-4612-0433-6 (cit. on p. 17).

[Tro98] A. S. Troelstra. "Realizability". In: *Handbook of Proof Theory*. Ed. by Samuel R. Buss. Vol. 137. Studies in Logic and the Foundations of Mathematics. Elsevier, 1998, pp. 407–473. DOI: 10.1016/s0049-237x(98)80021-9 (cit. on p. 3).

[Vic96] Steven Vickers. *Topology via Logic.*
Cambridge Tracts in Theoretical Computer Science.
Cambridge University Press, 1996 (cit. on p. 9).

[vOos02] Jaap van Oosten. "Realizability: a historical essay".
In: *Mathematical Structures in Computer Science* 12.3 (2002).
DOI: 10.1017/s0960129502003626 (cit. on pp. 3, 30).

[vOos08] Jaap van Oosten. *Realizability: An Introduction to its Categorical Side.*
Ed. by S. Abramsky, S. Artemov, D. M. Gabbay, A. Kechris, A. Pillay, and R. A. Shore. Vol. 152. Studies in Logic and the Foundations of Mathematics.
Elsevier, 2008. DOI: 10.1016/s0049-237X(08)X8001-2
(cit. on pp. 1–3, 7, 10, 21, 41).

[vOos16] Jaap van Oosten. "Basic Category Theory and Topos Theory".
Lecture notes with exercises. Feb. 2016. URL:
https://www.staff.science.uu.nl/~ooste110/syllabi/cattop16.pdf
(cit. on pp. iii, 23).

[Wei00] Klaus Weihrauch. *Computable Analysis: An Introduction.*
Texts in Theoretical Computer Science. An EATCS Series. Springer, 2000.
DOI: 10.1007/978-3-642-56999-9 (cit. on p. 9).

[Zoe18] Jetze Zoethout. "Slices of Realizability Topoi".
Master's thesis. Utrecht University, 2018. DOI: 20.500.12932/28690
(cit. on pp. 18, 41).