Saarland University

Faculty of Mathematics and Computer Science

Bachelor's Thesis

# Synthetic One-One, Many-One, and Truth-Table Reducibility in Coq

**Author**
Felix Jahn

**Advisor**
Yannick Forster

**Reviewers**
Prof. Dr. Gert Smolka
Prof. Dr. Markus Bläser

Submitted: September 15th, 2020

**Eidesstattliche Erklärung**

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

**Statement in Lieu of an Oath**

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

**Einverständniserklärung**

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

**Declaration of Consent**

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

F. Jahn

Saarbrücken, September 15th, 2020

# Abstract

Reducibility is an essential concept for undecidability proofs in computability theory. The idea behind reductions was conceived by Turing, who introduced the later so-called Turing reduction based on oracle machines. In 1944, Post furthermore introduced with one-one, many-one, and truth-table reductions in comparison to Turing reductions more specific reducibility notions. Post then also started to analyze the structure of the different reducibility notions and their computability degrees. Most undecidable problems were reducible from the halting problem, since this was exactly the method to show them undecidable. However, Post was able to construct also semidecidable but undecidable sets that do not one-one, many-one, or truth-table reduce from the halting problem.

This thesis formalizes and mechanizes parts of the traditional one-one, many-one, and truth-table reducibility theory based on a synthetic approach in the constructive type theory of the proof assistant Coq. The core idea of synthetic computability theory is to work in a setting that allows to identify the notion of a function with the notion of a computable function. As a programming language, Coq forms such a synthetic setting by guaranteeing all definable and therefore appearing functions to be immediately computable which avoids complex constructions in an underlying computational model.

We show positive results like Myhill's isomorphism theorem, order properties of reductions, and characterize both many-one and truth-table reducibility in terms of one-one reductions in axiom-free Coq. Distinctions of reducibility degrees as stated for instance by Post are in contrast only provable using particular properties of computational models like a universal machine. Constructing such a machine is not possible when working synthetically without a concrete computational model in hand. Assuming however abstract synthetic axioms concerning the computability of functions allows us to prove also negative results synthetically. We follow Post's construction of a simple set and achieve an intermediate many-one degree between decidable and many-one complete sets as well as further distinctions of reducibility degrees like many-one and truth-table completeness.

## Acknowledgements

First and foremost, I have to thank Yannick: Without his LaTex template, the thesis would look different ;-) Seriously, very special thanks to Yannick for the great guidance throughout the whole process, from motivating the topic to multiple proofreading, for innumerable exciting ideas and his inspiration for research shared in countless discussions. His outstanding scientific advice but especially his support beyond that were anything else than self-evident. Without him, this thesis would definitely look different, it would not even exist.

Of course I would like to thank Professor Smolka, who introduced and inspired me to the field of Computational Logic, gave me the opportunity to write the thesis at his chair, and supervised the project. But also apart from this thesis, he mentored and supported me throughout my whole studies and was always available for valuable advice.

I also want to thank all the other people of the Programming System Lab, who made me fell comfortable at the chair right from the start.

Many thanks also to Professor Bläser for arousing my interest in computability theory early on and for reviewing this thesis.

Big thanks to all my fellow students I met and who accompanied and supported me throughout the years and to all the people who fought their way through my lines while proofreading, special thanks to Laura, Marcel, Jannis, and Simon.

Finally, I would like to thank those who always gave me their backing, who gave me energy and motivation for my studies, but also reminded me again and again that there is a life outside of university. Many thanks to my parents, to Mirjana, and to all my friends!

# Contents

# Chapter 1

# Introduction

The foundation of what we know as computer science and especially theoretical computer science is what is nowadays called computability theory, started by the pioneering works of Church, Turing, Gödel, Post, Rosser, Kleene, and others in the 1930's. After the Church-Turing thesis and the equivalence proof of several computational models, Turing introduced the central concept of so-called reductions, which initially served as a method for showing undecidability. Additionally, also further questions regarding the various reduction concepts immediately arose and provided many more insights into computability theory.

Working in the proof assistant Coq, we want to formalize and mechanize numerous traditional results concerning one-one, many-one, and truth-table reductions in constructive type theory. The thesis takes an approach known as synthetic computability theory, which will allow to abstract from an underlying computational model throughout the formalization and mechanization. Instead, we can focus on a clear presentation and an advanced analysis of the mathematical and logical foundations of the studied computability results.

**Computability Theory** The research on computability theory was motivated by the question, which functions are "effectively calculable" and associated with that which problems can be determined by an "effective computation". Independently to the development of mechanical calculator machines, theoretical computability models were developed in order to improve the understanding of the nature of computable functions.

Positive results like the actual computability of functions could be justified by a concrete description of the respective calculation: Human beings or also machines follow an exactly given instruction and carry out the computation without further insights, intuition, or ingenuity only based on the calculation "recipe". Such functions were then called to be effectively calculable. However, this frequently used but informal expression did not allow to address also negative results. Showing for instance the undecidability of a certain problem and therefore the non-existence of effective calculations for particular functions was not provable without a formal notion of such calculations.

Therefore, the intuitive notion of an effectively calculable function should be characterized more formally by finding various (possibly incomplete) notions of computation: Numerous models like $\lambda$-calculus introduced by Church [6][7] and Kleene and Rosser [23], Gödel's (general) $\mu$-recursive functions [20] as well as the nowadays most famous Turing machines [36] were developed and the computability of functions could be discussed under the perspective of the respective models.

Furthermore and already with the introduction of their computability models, Church and Turing stated both in 1936 the thesis that $\lambda$-calculus or respectively Turing machines compute exactly the effectively calculable functions and that their models do not only cover a part of those functions.

While it was at first controversial or at least unclear whether "Church's thesis" and/or "Turing's thesis" should be believed, Turing could support and substantiate both theses: He showed still in the same year 1936 the computability models $\lambda$-calculus and Turing machines to describe exactly the same functions and thus "Church's thesis" and "Turing's thesis" to be equivalent [36]. The subsequently called "Church-Turing thesis" made therefore the claim that "effectively calculable" functions are exactly characterized by Turing machines or any other equivalent and consequently so-called Turing-complete model. Even though this statement is neither provable nor refutable but rather of philosophical nature, it was widely accepted and became the basis of further computability theory.

Based on the now consolidated concept of a computable function, the question of computationally decidable problems could also be investigated further. This was in some sense the founding moment of the theoretical research that had now the possibility to formally define and prove undecidable problems such as the well-known halting problem as also already done by Church and Turing in their initial papers.

**Reductions** When considering further seemingly undecidable problems, the methodology of so-called reductions was developed by Turing and extended by Post. Reducing a certain problem $A$ to another problem $B$ was supposed to reduce the question of the decidability of $A$ by the help of the reduction to $B$. Following this idea, the concept of a Turing reduction from $A$ to $B$ was defined as the possibility to construct a deciding algorithm for $A$ out of a deciding algorithm for $B$[1] [37]. Conversely, reductions could then be used to show further problems undecidable:

[1] "Deciding algorithm" is actually a somewhat imprecise term, which should above all convey intuition. Turing reductions actually use so-called oracle machines, see chapter 8.4 for more details.

If an already proven undecidable problem A such as the halting problem could be reduced to a further problem B, then also B had to be undecidable to not contradict the undecidability of A.

In addition to decidability, two further essential concepts of computability theory developed in the investigation of undecidable problems: Even if many problems such as the halting problem are undecidable, they are semidecidable or equivalently recursively enumerable. Semidecidability of a problem can be characterized as the existence of a program (or Turing Machine, or $\lambda$-term, or other computational model) which terminates exactly on the instances of the problem. This, in turn, led Post [28] to define the so-called reduction completeness with respect to various reducibility notions. A problem P is for example called Turing (reduction) complete, if it is itself semidecidable and if every other semidecidable problem is Turing reducible to P. By the above mentioned understanding of semidecidability, the halting problem is the canonical example for complete problems.

**Post's Problem & Computability Degrees** On the other hand, as described above, it was possible to prove the undecidability of many problems by reduction from the halting problem. It even seemed that all semidecidable but undecidable problems are reducible from the halting problem. In 1944, this question was raised by Post in his paper "Recursively enumerable sets of positive integers and their decision problems" [28]: He specifically asked for the existence of a semidecidable but undecidable set, which is not Turing reducible from the halting problem. This question became then known as Post's problem[2]. Even though he was not able to find a set solving his problem for Turing reductions, he approached the problem piecemeal and developed during the process important reducibility theory. He introduced further reducibility notions like the today well-known one-one, many-one, and truth-table reductions. Those notions were actually stronger reducibility notions than the Turing reduction. He could then construct sets, so-called simple and hyper-simple sets answering the question of Post's problem with respect to the stronger reducibility notions positively. It was not until 1955 that Muchnik [24] and Friedberg [17] independently solved the original Post's problem with respect to Turing reductions: Also for this reducibility notion, fitting sets solving the problem could be constructed by the so-called priority method.

The newly developed reducibility notions also gave room for further interesting research: Many-one reductions transport for example not only decidability but also semidecidability, which makes these reducibility notions very useful in order to classify problems into different computability classes. Furthermore, all the reducibility notions form a preorder and the computability degrees corresponding to the different reductions were considered.

[2]not to be confused with the undecidable Post's correspondence problem

Those computability degrees, defined as the existence of reductions from set $A$ to $B$ as well as conversely from $B$ to $A$, form an equivalence relation on sets and distinguish problems in different classes of computability with respect to certain reduction notions. The definition of complete problems implies those problems to form the class with maximal computability degree on the semidecidable sets. Myhill [25], Post [28], Nerode [26], Fischer [11], and many others continued to analyze the structure, coherences and similarities as well as differences of reductions and those degrees with respect to different reducibility notions more closely.

**Traditional Computability Theory** Since these developments in the 1930's and 1940's, many other important and interesting results about reductions were been found and proven. In addition, many presentations and textbooks of computability theory were been published in the second half of the 20th century, like the literature of Rogers [31], Cutland [8] and Soare [32], where the theory of reductions plays an essential role. All this traditional computability theory works in the usual classical logic system: The proofs use the classical principle of excluded middle or related further classical proof principles. Various axioms of choice are also frequently used as a support at appropriate points. Furthermore, there is no deviation from the concept of a concrete model of computation, which was established at the latest by the Church-Turing thesis: First, one or more computability models are introduced to define the computability of a function based on those concrete model. With this definition and the underlying model of computation further results are then shown.

**Constructive Mathematics** In addition to classical mathematics, various other formal logic systems developed in the course of the 20th century that differed significantly from the standard classical concept of logic. Such a new concept is constructive mathematics, which does not include the law of excluded middle and therefore no double negation elimination [5]. Instead, it follows the idea that a statement like for example the existence of a mathematical object is only provable by constructing the object. Various mathematical fields were then looked at again from a constructive point of view, which led to new insights into the basics of logic as well as a deeper understanding of the pure mathematical fields. Analyzing existing results under a constructive perspective made it possible to work out exactly which principles of classical logic are used at which points. In the attempt to constructivise those proofs, other and possibly simpler proof lines could be found or correlations discovered that explain why different axioms are even necessary to show certain theorems. These connections and necessities of axioms became the core question of the research field called reverse mathematics, among others explored under a constructive perspective by Ishihara [22] refining Friedman's reverse mathematics [18]. Also in the area of computability theory, this new logical approach as well as a further analysis under the reverse mathematical viewpoint provided new insights. For example Post's theorem, stating that semidecidability of both the predicate and its complement implies decidability, could be shown to be equivalent to the classical axiom of Markov's principle (cf. Troelstra and van Daalen [35]).

**Synthetic Computability Theory** With synthetic computability theory another approach deviating from traditional theory was developed by among others Richman [30] and Beeson [4]: Instead of introducing a concrete model of computation, one works in a setting that allows to identify the notion of a function with the notion of a computable function. Such settings are based on various general theories of mathematical foundations justifying the computability of occurring functions. A prominent example is Hyland [21], who uses category theory to construct what he calls the "effective topos" as well as Bauer [3] and Richman[30] using (constructive) set theory. The computability of the appearing functions, that is guaranteed in a certain way, allows to abstract from the often fiddly details in traditional settings when dealing with for instance Turing machines. The level of abstraction offered by this synthetic approach pays off especially in the formalization and even more blatantly in the mechanization of computability theory as observed by Forster, Kirst, and Smolka [14]. Due to the inconvenience of using concrete computational models formally, these very places are often quite hand-wavy in traditional presentations. Programming in a rudimentary computability model becomes soon extremely complex and exhausting. Therefore, one is traditionally often content with an informal description of the program or uses phrases like "obviously computable". A synthetic approach now offers the possibility of formalization and mechanization without having to work primarily on those details. In return, it allows to focus the attention on the "pure" mathematical theory.

**Coq's Type Theory** The synthetic approach is maybe most natural in type theory, where a notion of computation natively exists. Based on the so-called polymorphic calculus of cumulative inductive constructions [34], the type theory of the interactive proof assistant Coq is used also in general to mechanize various fields of mathematics and computer science in its constructive logic system. Under the assumption of axioms, one can also use further proof principles in Coq, in its axiom-free type theory it is however purely constructive.

Also different models of computation are already mechanized in Coq or other proof mechanization tools: There are several developments that mechanize the notion of Turing machines (cf. Asperti and Ricciotti [1][2], Xu, Zhang, and Urban [38] or Forster, Kunze, and Wuttke [15]), Carneio and Larchey-Wendling work on a mechanization of μ-recursive functions in Coq, and also different λ-calculi like the full λ-calculus (Norrish [27])or the weak call-by-value λ-calculus (Forster and Smolka [13]) are mechanized as models of computation. Even though these works as well as related work on further models like stack or counter machines includes often not only the model itself but also first basic results in computability theory by using their underlying model, it became quite hard to work formally with the concrete model when proving advanced computability results.

However and similar to constructive set theory also Coq's type theory can serve as a synthetic setting in order to mechanize computability theory synthetically: Beside the foundation of constructive mathematics, Coq also serves as a typed programming language. This guarantees all functions definable in Coq and therefore all functions occurring during the mechanization to be (effectively) computable functions. One is again able to abstract from an explicit and often difficult computation argument for an appearing function, since it is already given by defining the function in Coq. Coq itself justifies as a programming language the computability of functions and therefore the validity of computability theory in its synthetic setting.

Coq's type theory is therefore optimally suited for constructive and synthetic formalizations and mechanizations of computability theory. This was for instance already explored in the works of Forster, Kirst, and Smolka [14], who among other things prove results about synthetic decidability, many-one reductions, and some concrete problems like the Entscheidungsproblem. Also the numerous mechanizations of reductions in the Coq library of undecidable problems by Forster et al. [16] take the advantage of Coq's synthetic setting.

**Synthetic Computability Axioms** All of this work mechanizing computability theory synthetically has in common, that it focuses on positive results like using Coq to construct and verify deciders or reduction functions. Following further traditional results and their presentations of for instance undecidable problems or the non-existence of particular reductions reveals another crucial difference of the synthetic approach[3]: In contrast to the traditional presentations, synthetic computability theory abstracts from the concrete model of computation and can therefore not prove and especially use certain properties of this model. Traditional work is for example based on a universal machine simulating a concretely given instance of the computational model (or its encoding) on a given input. Without a computational model in hand, it is not possible to construct such a universal machine and one can therefore neither define numerous undecidable problems like the halting problem nor show other negative results.

To show such results synthetically nevertheless, one has to assume the necessary computational properties as abstract axioms. By assuming for instance an abstract imitation of a universal machine as done in the fundamental axiom of synthetic computability theory called Church's thesis (cf. Troelstra and van Daalen [35]) one adds among other things undecidable problems to the setting. Assuming such synthetic computability axioms was explored more closely by for instance

[3]not only appearing in the synthetic setting formed by Coq but appearing in general when working synthetically

Richman [30] and Bauer [3], and in parts already formalized and mechanized by Forster [12]. Similar to the constructive view on classical proofs, also the synthetic approach using axioms leads to a precise analysis on what properties of the computational model are used at which point. In addition, it opens again the field for questions of reverse mathematics regarding the weakest necessary set of axioms for certain results.

**Thesis Structure & Contributions** This thesis aims to continue the constructive formalization and mechanization of synthetic computability theory in the field of reducibility theory. By using synthetic axioms, one of the main contributions of the thesis is the formalization and mechanization of also negative results in synthetic computability theory. There are already mechanizations of the halting problem as well as further undecidable problems on concrete Turing machines or in lambda calculus [13]. In contrast to those often inconvenient proofs because of the above described problematic regarding the formal work with concrete models, we can present clear and clean proofs and address more advanced results of reducibility theory.

Therefore, we want to introduce different reducibility notions like one-one, many-one, and truth-table reductions and show different results about these reducibility notions and their corresponding degrees. We start in chapter 3 with the definitions of basic terms of computability theory like (semi-)decidability and enumerability as well as the different notions of reductions and prove their basic properties like closure and transport properties. We will furthermore introduce axioms in synthetic computability theory. It turns out, that it is not necessary to assume all the strong properties of concrete computability models like for example a universal machine. The for our results actually required axioms are discussed and justified more precisely in chapter 3.3.

Chapter 4 continues with a formalization of Myhill's isomorphism theorem, stating the coincide of recursive isomporphisms and the one-one degree. This proof is as its name suggests due to Myhill and based on the notion of finite correspondence sequences [25]. We focus on further characterizations of reductions and their degrees and look at so-called cylinders and truth-table cylinders, that can express many-one reductions in terms of one-one reductions as well as truth-table reductions in terms of one-one reductions.

We come back to the work of Post and want to address Post's problem with respect to many-one reducibility and formalize his class of simple sets. Using the universe of propositions included in Coq's type theory, we represent traditional sets (of for example natural numbers) as predicates over those types. Accordingly, we will from now on only speak about predicates instead of sets or problems[4]. For rea-

[4]with the exception of the "halting problem"

sons of consistency and understanding, this will be maintained even in passages regarding traditional mathematic or computability theory.

In order to formalize Post's simple predicates that are defined as a subclass of infinite predicates, we will have to address a formalization of infinite predicates. The definition and behavior of those predicates in our constructive and synthetic setting turns out to be highly interesting as well as essential for the actual results concerning reducibility theory. Therefore, we formalize firstly different notions of infinite predicates and their properties like for instance different closure properties of in chapter 5. We thereby detect serious differences to the classical understanding of infinite predicates by recognizing significant constructive differences between various notions of infinite predicates, that are in contrast classically seen to be equivalent. Chapter 5 can be read largely independently from the computability theory formalized in the rest of the thesis. In particular, it is possible to read this chapter in altered order.

Using the work about infinite predicates and carefully chosen synthetic axioms, we can then construct simple predicates in chapter 6 and show their characteristics, that allow us to derive further results regarding reducibility notions: Firstly, we can formalize Post's problem for many-one reductions and therefore a distinction of different many-one degrees on the class of semidecidable but undecidable predicates. Simple predicates yield furthermore a distinction of one-one and many-one reductions on this class of predicates as well as a distinction of many-one and truth-table completeness.

All of those mentioned results from chapters 2 – 6 are not only formalized but also mechanized in Coq's proof assistant. The definitions, lemmas and theorems are hyperlinked to a version of the development viewable in a webbrowser, the source code is available under:

`https://github.com/uds-psl/synthetic-reducibility-in-coq`

We take a closer look at the mechanization in chapter 7 and point out some of the important decisions as well as occurring difficulties during this development.

Lastly, we want to discuss possible future work in the fields discussed by this thesis. Since the theory of reductions is traditionally quite diverse and far-reaching, there are many further results that are of great interest to analyze under a constructive and synthetic perspective as well as to formalize and mechanize those results. However, chapter 8 not only lists this possible future work, but also discusses in detail further traditional computability results not addressed in this thesis. Besides this embedding of our work in traditional theory, we also take a look at possible future work regarding the foundations of our constructive and synthetic approach to computability theory.

Chapter 2

# Technical Preliminaries

In this chapter, we give a short introduction in Coq's type theory and present basic definitions, notations, and results required for our work.

## 2.1 Type Theory

We formalize and mechanize reducibility theory in the constructive type theory of Coq [34], that is based on a universe of types $\mathbb{T}$ including an impredicative subuniverse of propositions $\mathbb{P}$. We mainly use the following (inductive) types throughout this thesis:

$$\begin{aligned}
n : \mathbb{N} &::= 0 \mid \mathsf{S}n && \text{(natural numbers)}\\
b : \mathbb{B} &::= \mathsf{true} \mid \mathsf{false} && \text{(booleans)}\\
o : \mathcal{O}(X) &::= \mathsf{None} \mid \mathsf{Some}\, x \quad \text{where } x : X && \text{(options)}\\
L : \mathcal{L}(X) &::= [\,] \mid x :: L \quad \text{where } x : X && \text{(lists)}\\
X + Y &::= \mathsf{L}x \mid \mathsf{R}y \quad \text{where } x : X \text{ and } y : Y && \text{(sums)}\\
X \times Y &::= (x, y) \quad \text{where } x : X \text{ and } y : Y && \text{(products)}
\end{aligned}$$

We use the common operations on natural numbers and Cantor's bijective pairing function $\langle\cdot,\cdot\rangle : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ with the respective inverses $\pi_1, \pi_2 : \mathbb{N} \to \mathbb{N}$ . $\vee_{\mathbb{B}}$ is the boolean disjunction, $\wedge_{\mathbb{B}}$ the boolean conjunction, and $\neg_{\mathbb{B}}$ the boolean negation. $\mathsf{hd}$ projects the first element of a non-empty lists. We overload $\pi_1$ and $\pi_2$ by denoting also the projections out of a pair by these notions.

The function type for functions from type $X$ to type $Y$ is denoted as $X \to Y$. Furthermore, Coq includes a dependent function type $\forall x : X.tx$, where the type of the function value depends on the argument $x$, i.e. $t : X \to \mathbb{T}$. We write functions of both types $X \to Y$ and $\forall x : X.tx$ as $\lambda x : X.s$, where $s : Y$ or $s : tx$ respectively and $x$ may occur in $s$.

Similarly, Coq includes dependent pairs, where the type of the second component depends on the first component of the pair:

$$\Sigma x : X.\ tx ::= (x, s) \quad \text{where } t : X \to \mathbb{T} \text{ and } x \text{ may occur in } s : tx.$$

The proposition $\top$ expresses truth, $\bot$ falsity, $\wedge$ logical conjunction, $\vee$ logical disjunction, and $\to$ logical implication.
We denote logical universal quantification as $\forall x.px$ and logical existential quantification as $\exists x.px$, where $p : X \to \mathbb{P}$.

$\bot$ implies every other proposition, i.e. $\forall P : \mathbb{P}.\ \bot \to P$. Logical negation of $P : \mathbb{P}$ is defined as $\neg P := P \to \bot$. For our purposes crucially, one can prove the following logical statement purely constructive:

**Lemma 2.1** *For all propositions* $P : \mathbb{P}$*, we have* $\neg\neg(P \vee \neg P)$.

The Lemma especially allows to logically decide every proposition when proving $\bot$.

For $\neg\neg\exists x.px$, we say that "there weakly exists $x$ such that $px$" where again $p : X \to \mathbb{P}$.

We represent sets with elements of a type $X$ as a predicate over $X$, i.e. a function $p : X \to \mathbb{P}$ such that $px$ holds exactly for those $x$ that are in the set.

While sum types and dependent pairs can be eliminated in arbitrary contexts, propositional disjunctions and existential quantification can only be eliminated when proving propositions. Especially for $p : X \to \mathbb{P}$, $\exists x.px$ implies not in general $\Sigma x.px$.

## 2.2 Preliminary Definitions

We define basic notions for functions, types, and predicates.

**Definition 2.2** *A function* $f : X \to Y$ *is called,*

1. ***surjective****, if* $\forall y.\ \exists x.\ fx = y$.
2. ***injective****, if* $\forall x_1 x_2.\ fx_1 = fx_2 \to x_1 = x_2$.
3. ***bijective****, if* $f$ *is surjective and injective.*

*We define surjections and injections from predicates* $p : X \to \mathbb{P}$ *to* $q : Y \to \mathbb{P}$ *:*

4. $f : p \twoheadrightarrow q := \forall y.\ qy \to \exists x.\ px \wedge fx = y$.
5. $f : p \hookrightarrow q := \forall x_1 x_2.\ px_1 \to px_2 \to fx_1 = fx_2 \to q(fx_1) \wedge x_1 = x_2$

*We write* $p \twoheadrightarrow q$*, if there exists* $f : p \twoheadrightarrow q$ *and write* $p \hookrightarrow q$ *if there exists* $f : p \hookrightarrow q$.

6. *We call types* $X$ *and* $Y$ ***isomorphic****, if there exists a bijection* $f : X \to Y$*. In this case, we write* $X \cong Y$*.*

We identify types $X$ with their constant true predicate $\lambda x : X.\ \top$ and use the above definitions 4. and 5. also for types. Accordingly, we write for instance $p \twoheadrightarrow X$, $X \hookrightarrow p$, or $X \hookrightarrow Y$.

**Definition 2.3** *Let* $X : \mathbb{T}$, $p : X \to \mathbb{P}$, *and* $q : X \to \mathbb{P}$. *We define,*

1. *the* ***complement*** *of* $p$ *as* $\overline{p} := \lambda x.\ \neg px$.
2. *the* ***intersection*** *of* $p$ *and* $q$ *as* $p \wedge q := \lambda x.\ px \wedge qx$.
3. *the* ***union*** *of* $p$ *and* $q$ *as* $p \vee q := \lambda x.\ px \vee qx$.
4. *the* ***difference*** *of* $p$ *and* $q$ *as* $p \setminus q := \lambda x.\ px \wedge \neg qx$.
5. *the* ***subset of predicates*** *as* $p \subseteq q := \forall x.\ px \to qx$.
6. *a* ***certifying decider*** *for* $p$ *to be a function of type* $\forall x.\ px + \neg px$.
7. $p$ *to be* ***stable****, if* $\forall x.\ \neg\neg px \to px$.
8. $X$ *to be* ***discrete****, if there exists a certifying decider for the predicate* $\lambda(x_1, x_2).\ x_1 = x_2$.
9. $X$ *to be* ***enumerable****, if* $\mathbb{N} \twoheadrightarrow X$.
10. $X$ *to be a* ***datatype****, if* $X$ *is discrete and enumerable.*

Although the above notions as well as all following definitions regarding predicates are defined only on unary predicates, we use them for $n$-ary relations via (implicit) uncurrying. In functional definitions, we denote the computation deciding a predicate $p$ with certifying decider as $\ulcorner px \urcorner$.

## 2.3 Witness Operator and Inverse Functions

Coq's type theory allows to prove axiom-freely a so-called witness-operator, that given a witness proof for the satisfiability of a predicate $p$ with certifying decider computes an element in $p$.

**Lemma 2.4** *For predicates* $p : \mathbb{N} \to \mathbb{P}$ *with certifying decider* $f : \forall x.\ px + \neg px$, *there is a witness operator* $\omega_{\mathbb{N}} : (\exists x.px) \to \mathbb{N}$, *that computes an element satisfying* $p$, *i.e.*

$$\forall H : (\exists x.px).\ p(\omega_{\mathbb{N}} H).$$

**Proof** For every predicate $p$, we define an inductive guard predicate $G : \mathbb{N} \to \mathbb{P}$ by the rule:

$$\frac{\overline{p}n \to G(Sn)}{Gn}$$

The induction principle for $\mathsf{G}$

$$\forall q : \mathbb{N} \to \mathbb{T}.\ (\forall n.\ (\overline{p}n \to q(Sn)) \to qn) \to \forall n.\ Gn \to qn$$

allows us to proof the key property of $\mathsf{G}$:

1. $\forall n.\ Gn \to \Sigma x.px$ by induction on $G$ for the constant predicate $qn := \Sigma x.px$ and deciding $pn$. In the case, $\overline{p}n$, we use the inductive hypothesis.

Furthermore, the guard predicate $G$ is defined such that

2. $(\exists n.Gn) \to G0$ holds by induction on $n$ using the fact $G(Sn) \to Gn$, and

3. $(\exists x.px) \to (\exists n.Gn)$.

Therefore, we define $\omega_{\mathbb{N}}$ by applying 3, 2, and 1. □

The introduced witness operator $\omega_{\mathbb{N}}$ for predicates over natural numbers yields version fore more general types:

**Corollary 2.5** *For predicates* $p : X \to \mathbb{P}$ *with certifying decider* $f : \forall x.\ px + \neg px$ *and an enumerator* $E : \mathbb{N} \twoheadrightarrow X$*, there is a witness operator* $\omega_X : (\exists x.px) \to X$ *such that*

$$\forall H : (\exists x.px).\ p(\omega_X H).$$

**Proof** By applying $\omega_{\mathbb{N}}$ to the natural number predicate $\lambda n.p(En)$. □

Notice, that we need both the certifying decider as well as the enumerator in hand when applying the witness operator. Using $\omega_X$, we can define inverse functions of bijections.

**Lemma 2.6** *Let* $X$ *be enumerable,* $Y$ *a discrete type and* $f : X \to Y$ *a bijective function. Then, there is an inverse function* $f^{-1}$ *such that*

$$\forall y.\ f(f^{-1}y) = y \ \textit{ and } \ \forall x.\ f^{-1}(fx) = x.$$

**Proof** For all elements $y : Y$, the predicate $p_y := \lambda x.\ fx = y$

1. has a certifying decider, since $Y$ is discrete, and

2. has a witness for its satisfiability $H_y : \exists x.fx = y$ by the surjectivity of $f$.

Given $y : Y$, we can therefore use $\omega_X$ on the predicate $p_y$ and define □

Notice, that we use the injectivity of $f$ only in the last proof step. Therefore, one can also define for a surjection $f$ the right-inverse function $f^{-1}$ such that $\forall y.\ f(f^{-1}y) = y$.

## 2.4 List Predicates and List Functions

Lists play an essential throughout the whole development. We introduce basic predicates and operations on lists. Thereby, we overload the subset notation " $\subseteq$ " multiple times.

**Definition 2.7** *We define inductive predicates* $\in : X \to \mathcal{L}(X) \to \mathbb{P}$ *for* ***list membership*** *and* $\# : \mathcal{L}(X) \to \mathbb{P}$ *for* ***duplicate-free*** *lists by the following inductive rules:*

$$\frac{}{x \in (x :: L)} \qquad \frac{x \in L}{x \in (a :: L)} \qquad\qquad \frac{}{\#[\,]} \qquad \frac{x \notin L \quad \#L}{\#(x :: L)}$$

*Furthermore, we define*

1. *the* ***subset of lists*** *as* $L_1 \subseteq L_2 := \forall x \in L_1.\, x \in L_2$.
2. *the* ***subset of lists in predicates*** *as* $L \subseteq p := \forall x \in L.\, px$.
3. *the* ***subset of predicates in lists*** *as* $p \subseteq L := \forall x.\, px \to x \in L$.

We denote the length of a list $L$ as $|L|$ and denote the append of lists $L_1, \cdots, L_n$ as $L_1 \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} L_n$. We define recursively the mapping of a function pointwise to all elements of a list:

$$\begin{aligned} \mathsf{map} :\ & (X \to Y) \to \mathcal{L}(X) \to \mathcal{L}(Y) \\ \mathsf{map}\ f\ [\,] &:= [\,] \\ \mathsf{map}\ f\ (x :: L) &:= (fx) :: (\mathsf{map}\ f\ L) \end{aligned}$$

We introduce basic properties of the above functions and predicates. Notice especially the results for injective mappings.

**Lemma 2.8** *For all lists* $L$ *and functions* $f$,

1. $|\mathsf{map}\ f\ L| = |L|$.
2. $\forall y.\, y \in (\mathsf{map}\ f\ L) \leftrightarrow \exists x \in L.\, fx = y$.

*For all predicates* $p$*, lists* $L \subseteq p$*, and injections* $f : p \hookrightarrow Y$ :

3. *For all* $x$ *with* $px$ : $x \in L \leftrightarrow fx \in (\mathsf{map}\ f\ L)$.
4. $\#L \leftrightarrow \#(\mathsf{map}\ f\ L)$.

**Proof** 1. and 2. by induction on $L$ or $L_1$, 3. follows with 2., and 4. with 3. $\square$

For lists of pairs, we define both projections and a swapping function:

$$\begin{aligned} \pi_1 &: \mathcal{L}(X \times Y) \to X \qquad & \pi_2 &: \mathcal{L}(X \times Y) \to Y \\ \pi_1 L &:= \mathsf{map}\ \pi_1\ L & \pi_2 L &:= \mathsf{map}\ \pi_2\ L \end{aligned}$$

$$\begin{aligned} \overset{\leftrightarrow}{\cdot} &: \mathcal{L}(X \times Y) \to \mathcal{L}(Y \times X) \\ \overset{\leftrightarrow}{L} &:= \mathsf{map}\ (\lambda(x, y).\ (y, x))\ L \end{aligned}$$

Those functions come with the following basic results:

**Lemma 2.9** *For all lists of pairs* $L : \mathcal{L}(X \times Y)$*,*

1. $|\pi_1 L| = |\pi_2 L| = |\overset{\leftrightarrow}{L}| = |L|$.
2. $\forall x.\ x \in \pi_1 L \leftrightarrow \exists y.(x, y) \in L$ *and* $\forall y.\ y \in \pi_2 L \leftrightarrow \exists x.(x, y) \in L$*;*
   *furthermore* $\forall xy.\ (x, y) \in L \leftrightarrow (y, x) \in \overset{\leftrightarrow}{L}$.

*If* $f$ *is in addition an enumerator of* $Y$ *and* $X \times Y$ *is discrete, then:*

3. *Given* $x \in \pi_1 L$*, one can compute* $y$ *with* $(x, y) \in L$.

**Proof** 1. and 2. Straightforward with Lemma 2.8.
3. By using the witness operator $\omega_Y$ for the predicate $\lambda y.\ (x, y) \in L$. □

In the remaining chapter, we assume the underlying list type to be discrete. Therefore, there is not only a certifying decider for $\lambda x_1 x_2.\ x_1 = x_2$ but also for $\lambda x L.\ x \in L$. We define recursive functions removing a given element from a list, computing the difference of two lists and computing the maximum of lists over natural numbers:

$$\begin{aligned} - &: \mathcal{L}(X) \to X \to \mathcal{L}(X) \\ [\,] - a &:= [\,] \\ (x :: L) - a &:= \mathsf{let}\ L_1 := L - a\ \mathsf{in}\ \mathsf{if}\ \ulcorner x = a \urcorner\ \mathsf{then}\ L_1\ \mathsf{else}\ x :: L_1 \end{aligned}$$

$$\begin{aligned} \setminus &: \mathcal{L}(X) \to \mathcal{L}(X) \to \mathcal{L}(X) \\ [\,] \setminus L_2 &:= [\,] \\ (x :: L_1) \setminus L_2 &:= \mathsf{let}\ L := L_1 \setminus L_2\ \mathsf{in}\ \mathsf{if}\ \ulcorner x \in L_2 \urcorner\ \mathsf{then}\ L\ \mathsf{else}\ x :: L \end{aligned}$$

$$\begin{aligned} \max &: \mathcal{L}(\mathbb{N}) \to \mathbb{N} \\ \max[\,] &:= 0 \\ \max(x :: L) &:= \mathsf{let}\ m := \max L\ \mathsf{in}\ \mathsf{if}\ \ulcorner x > m \urcorner\ \mathsf{then}\ x\ \mathsf{else}\ m \end{aligned}$$

The functions come with the expected properties. Notice furthermore, that $L{-}x$ and $L \setminus L_2$ are duplicate-free, if $L$ is duplicate-free. Analogously to the removing function filtering a list for a given element, one can also define a more general function filtering lists with respect to every other predicate with certifying decider.

The removing function allows us to prove two variants of the pigeonhole principle.

**Lemma 2.10 (Pigeonhole Principles)** *Let $X$ be discrete, $L_1, L_2 : \mathcal{L}(X)$ with $\#L_1$. Then,*

1. $|L_1| > |L_2| \to \exists x.\, x \in L_1 \land x \notin L_2$.
2. $L_1 \subseteq L_2 \to |L_1| \leqslant |L_2|$.

**Proof**
1. By induction on $L_1$ for generalized $L_2$. In the step case $(a :: L_1)$, we use the inductive hypothesis for $L_2 - a$.
2. Assuming $L_1 > |L_2|$ contradicts $L_1 \subseteq L_2$ by 1. □

Chapter 3

# Synthetic Computability Theory

Coq's type theory forms a synthetic setting for computability theory, that we want to use for our aspired formalization and mechanization of reducibility theory. Instead of following the traditional presentations of computability theory by introducing a concrete model of computation and formalize results based on this model, we work directly in a programming language that justifies the computability of occurring functions: A function defined in Coq's typed programming language is immediately guaranteed to be indeed an effectively computable function.

Also in the sense of the prominent and widely accepted Church-Turing thesis, that states all effectively calculable functions to be computable by the various notions of Turing complete models, one can argue the computability of Coq functions. The functions definable in Coq form a subclass of Turing computable functions, such that it is sound with traditional computability theory to assume Coq functions to be computable in the traditional sense.

The synthetic approach offers a significant benefit in the formalization of computability theory. We do not have to deal with formal definitions of for instance Turing machines, which becomes highly complex during proofs and even more during their mechanization. We will not have to struggle through inconvenient constructions of Turing machines or similar computational models to justify the computability of a certain function; the function is justified to be computable as a Coq function.

We will explore this advantage of the synthetic setting already in this chapter when defining notions of computability theory and proving basic results concerning those notions. It then pays off even stronger in the more profound results presented in the further chapters, whose clean formalization is only made possible by this approach.

This chapter reviews formalizations from Forster, Kirst, and Smolka [14] and extends those results to truth-table reductions and the notions of computability degrees and introduces furthermore the for our intended results necessary synthetic axioms based on the work from Forster [12].

## 3.1 Basic Notions of Computability Theory

We start by introducing basic notions and define them in our synthetic setting. Recall that we formalize sets and decision problems in Coq's type theory as predicates.

A predicate is traditionally understood to be decidable if there exists a computable decider for the predicate. Besides varying formulated specifications of those deciders, the traditional definitions contain always the explicit condition of this decider to be computable in the sense of their underlying computability notion. We can omit this explicit condition: every decider is immediately computable as a Coq function. Similarly, we can simplify the definition of semidecidable and (recursively) enumerable predicates.

**Definition 3.1** *A predicate* $p : X \to \mathbb{P}$ *is called*

1. ***decidable****, if there exists a function* $f : X \to \mathbb{B}$ *such that*

   $$\forall x.\ px \leftrightarrow fx = \mathsf{true}.$$

   *In this case, we write* $\mathcal{D}\,p$ *and call* $f$ *a* ***decider*** *for* $p$*.*

2. ***semidecidable****, if there exists a function* $f : X \to \mathbb{N} \to \mathbb{B}$ *such that*

   $$\forall x.\ px \leftrightarrow \exists n.\ fxn = \mathsf{true}.$$

   *In this case, we write* $\mathcal{S}\,p$ *and call* $f$ *a* ***semidecider*** *for* $p$*.*

3. ***enumerable****, if there exists a function* $f : \mathbb{N} \to \mathcal{O}(X)$ *such that*

   $$\forall x.\ px \leftrightarrow \exists n.\ fn = \mathsf{Some}\ x.$$

   *In this case, we write* $\mathcal{E}\,p$ *and call* $f$ *an* ***enumerator*** *of* $p$*.*

4. ***strongly enumerable****, if there exists a function* $f : \mathbb{N} \to X$ *such that*

   $$\forall x.\ px \leftrightarrow \exists n.\ fn = x.$$

   *In this case, we call* $f$ *a* ***strong enumerator*** *of* $p$*.*

Notice the only slight difference in the definition of enumerability and strong enumerability in the type of the respective enumerator. The not so common notion of strong enumerability will be of major help in the construction of a simple predicate in chapter 6.1.

In order to address the aspired reducibility theory, we continue by introducing the in our synthetic setting definable different notions of reducibility. Turing reductions seem to be not possible to define in a synthetic setting (cf. chapter 8), but we can formalize the further notions that are due to Post [28].

The nowadays most prominent reduction is the so-called many-one reduction. It demands a reduction function mapping exactly the instances of one predicate to instances of another one. Again, this function is traditionally explicitly required to be a computable function. Synthetically, we can omit this condition.

A to many-one reducibility closely related notion is the so-called one-one reduction, that can be understood roughly speaking as an injective many-one reduction.

**Definition 3.2 (One-One Reduction)**
$p : X \to \mathbb{P}$ *is* ***one-one reducible*** *to* $q : Y \to \mathbb{P}$*, if there exists a function* $f : X \to Y$ *such that*

$$\text{injective } f \wedge \forall x.\ px \leftrightarrow q(fx).$$

*In this case, we write* $p \preceq_1 q$ *and call* $f$ *a* ***one-one reduction*** *from* $p$ *to* $q$*.*

**Definition 3.3 (Many-One Reduction)**
$p : X \to \mathbb{P}$ *is* ***many-one reducible*** *to* $q : Y \to \mathbb{P}$*, if there exists a function* $f : X \to Y$ *such that*

$$\forall x.\ px \leftrightarrow q(fx).$$

*In this case, we write* $p \preceq_m q$ *and call* $f$ *a* ***many-one reduction*** *from* $p$ *to* $q$*.*

Furthermore, Post [28] introduced the notion of truth-table reductions, again defined from one predicate $p : X \to \mathbb{P}$ to another predicate $q : Y \to \mathbb{P}$. Given an element $x : X$, this reduction builds up a query list of elements in $Y$ and states additionally an associated condition. Intuitively, this condition is allowed to use a decider for $q$ on the elements in the query list to then combine the resulting truth-table to a decision of $px$. Both, the computation of the query list as well as its condition are again required but synthetically immediately guaranteed to be computable functions.

We formalize this intuition of the truth-table condition as a function allowed to use a decider on the query list by the notion of corresponding lists under a predicate. The condition will not work directly on the query list but on a boolean list containing the decisions for $q$ for the query list. We say that the query list corresponds under the predicate $q$ to such a boolean list.

**Definition 3.4 (Truth-Table Reduction)** *Let* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$*.*
*We define an inductive predicate* $\mathrel{\widehat{=}}_q\ : \mathcal{L}(Y) \to \mathcal{L}(\mathbb{B}) \to \mathbb{P}$ *by the following rules:*

$$\frac{}{[\,] \mathrel{\widehat{=}}_q [\,]} \qquad\qquad \frac{qy \leftrightarrow b = \mathsf{true} \qquad L_Y \mathrel{\widehat{=}}_q L}{(y :: L_Y) \mathrel{\widehat{=}}_q (b :: L)}$$

*For* $L_Y \mathrel{\widehat{=}}_q L$*, we say that* $L_Y$ ***corresponds under*** $q$ *to* $L$*.*

$\mathsf{p}$ *is **truth-table reducible** to* $\mathsf{q}$*, if there exist two functions* $f : X \to \mathcal{L}(Y)$ *and* $\alpha : \forall x : X.\ \mathbb{B}^{|fx|} \to \mathbb{B}$ *such that*

$$\forall x : X.\ \forall L.\ (fx) \mathrel{\widehat{=}_q} L \to (px \leftrightarrow \alpha x L = \mathsf{true}).$$

*In this case, we write* $p \preceq_{tt} q$*, call* $f$ ***query function** and* $\alpha$ ***truth-table condition**.*

Note several details about the truth-table condition in the above definition. Firstly, we are not at all interested in the behavior of $\alpha$ on boolean lists that the query list $fx$ does not corresponds to under $q$.

Secondly, we use a dependent type $\forall x : X.\ \mathbb{B}^{|fx|} \to \mathbb{B}$ for the condition: $\alpha$ works not on general boolean lists but only on lists with the same length as the query list. This will allow us to follow the traditional presentations and embed the finite function type $\mathbb{B}^{|fx|} \to \mathbb{B}$ injectively into the natural numbers in order to express truth-table as one-one reductions in the next chapter. This would be clearly impossible for the uncountable function space $\mathcal{L}(\mathbb{B}) \to \mathbb{B}$.

Only because thirdly the premise $(fx) \mathrel{\widehat{=}_q} L$ implies both lists to have the same length and therefore $L : \mathbb{B}^{|fx|}$, the definition is well-typed. When defining a concrete truth-table condition, this allows us furthermore to assume the argument $L$ to have a particular length and match for instance in the case $|fx| = 1$ on singleton lists.

We discuss the exact mechanization in Coq in more detail in chapter 7, thinking about the truth-table condition to be defined on boolean lists with particular length suffices for the following formalizations.

By requiring two predicates to reduce to each other in both directions, we obtain so-called computability degrees[1]. Those computability degrees are defined for each of the different reducibility notions individually again as a relation on two predicates.

**Definition 3.5 (Computability Degree)** *Predicates* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$ *have the same **computability degree** with respect to some reducibility notion* $\preceq$*, if*

$$p \preceq q \land q \preceq p.$$

*The degree with respect to* $\preceq_1$ *is called* 1*-degree and we write* $p \equiv_1 q$*, with respect to* $\preceq_m$ $m$*-degree and* $p \equiv_m q$*, with respect to* $\preceq_{tt}$ $tt$*-degree and* $p \equiv_{tt} q$*.*

Besides those computability degrees based on various notions of reductions, Myhill [25] presents a further computability degree called recursive isomorphism. Similar to one-one reductions as injective many-one reductions, one can think about

[1]also called degrees of computation or degrees of (un-)decidability

isomorphisms as bijective many-one reductions. The bijectivity of the "reduction function" implies the notion to be indeed a computability degree notion, like it will be shown in the next section 3.2.

**Definition 3.6 (Isomorphism)** *A predicate* $p : X \to \mathbb{P}$ *is called* ***isomorphic***[2] *to a predicate* $q : Y \to \mathbb{P}$*, if there exists a function* $\varphi : X \to Y$ *such that*

$$\mathsf{bijective}\ \varphi \land \forall x.\ px \leftrightarrow q(\varphi x).$$

*In this case, we write* $p \equiv q$ *and call* $\varphi$ *an* ***isomorphism****.*

The definition of isomorphisms is in our constructive setting slightly deviating from the "usual" or algebraic understanding of isomorphisms: One would expect $\varphi$ to have an inverse function $\varphi^{-1}$, such that $\varphi$ and $\varphi^{-1}$ cancel each other out. Following this, we could have defined two predicates $p$ and $q$ isomorphic as

$$\exists \varphi \varphi^{-1}.\ \big(\forall x.\ px \leftrightarrow q(\varphi x)\big) \land \big(\forall x.\ \varphi^{-1}(\varphi x) = x \land \forall y.\ \varphi(\varphi^{-1} y) = y\big).$$

The bijectivity of $\varphi$ would then easily follow with the "canceling out" property. Since it is for arbitrary types not clear whether such an inverse function is computable, we can in general not construct an inverse function out of our definition of isomorphisms. However, we stay not only faithful with traditional computability theory by following their definitions (cf. Myhill [25]), but could also show the equivalence of these both notions for predicates on datatypes[3] using the inverse function from Lemma 2.6.

Lastly, we introduce the notion of complete predicates. Those predicates are defined to be semidecidable, such that in addition every other semidecidable predicate should reduces to those complete predicates. This directly implies complete predicates to form the subclass of semidecidable predicates with maximal computability degree. Similarly to computability degrees, we again obtain different completeness notions for different reductions.

**Definition 3.7 (Reduction Completeness)**
*A predicate* $p : X \to \mathbb{P}$ *is called* ***complete*** *with respect to some reducibility notion* $\preceq$*, if*

$$\mathcal{S}p \land \forall Y.\ X \cong Y \to \mathsf{discrete}\ Y \to \forall q : (Y \to \mathbb{P}).\ \mathcal{S}q \to q \preceq p.$$

*Completeness with respect to* $\preceq_1$ *is called 1-completeness, with respect to* $\preceq_m$ $m$*-completeness, and with respect to* $\preceq_{tt}$ $tt$*-completeness.*

[2]The traditional term "recursively isomorphic" is redundant in our setting: every isomorphism is as a function in our synthetic setting recursive/computable.

[3]and enumerable types with decidable equality are actually the types traditional computability theory works in

The predicates required to reduce to complete predicates are restricted to work over isomorphic types. Otherwise, Cantor's theorem stating the impossibility of finding an injection from $(X \to \mathbb{B})$ to $X$ itself would imply the non-existence of 1-complete predicates. We discuss further remarks regarding the mechanization of complete predicates in chapter 7.

## 3.2 Basic Computability Theory Results

We have to start by showing various basic results regarding the introduced notions in computability theory. Besides proving numerous well-known traditional results about for instance (semi-)decidability or reductions now synthetically, we will discuss some further properties of constructive and type theoretical nature. So is our first result stating characterizations of decidability.

**Lemma 3.8** 1. *A predicate $p : X \to \mathbb{P}$ is decidable iff it has a certifying decider.*

2. *Decidable predicates are stable.*

**Proof** 1. Firstly assume a decider $f$ for $p$ and an element $x$. If $fx = \mathsf{true}$, then we can show $px$, else $fx = \mathsf{false}$ and $\neg px$ holds. Otherwise assume a certifying decider $d : \forall x.\ px + \neg px$. Now, $\lambda x.$ if $dx$ then true else false decides $p$.

2. Assume $\neg\neg px$. With 1., we either have already $px$ or $\neg px$, a contradiction. □

We prove connections between the notions of decidability, semidecidability, and (strongly) enumerability introduced in Definition 3.1. With the definitions of those notions simplified by our synthetic setting, also the formal proofs become significantly simpler: We no longer need to reason about the computability of functions occurring throughout the proofs, but are done by constructing and verifying those functions.

**Lemma 3.9** 1. *Decidable predicates are semidecidable.*

2. *Semidecidable predicates over enumerable types are enumerable.*
3. *Enumerable predicates over discrete types are semidecidable.*
4. *Enumerable predicates with $\exists x_0 . px_0$ are strongly enumerable.*
5. *Strongly enumerable predicates are enumerable.*

**Proof** 1. Let $f : X \to \mathbb{B}$ be a decider for $p$, then $\lambda xn.fx$ is a semidecider for $p$.

2. Let $f : X \to \mathbb{N} \to \mathbb{B}$ be a semidecider for $p$ and $E : \mathbb{N} \twoheadrightarrow X$ an enumerator of $X$. Then, $\lambda\langle n_1, n_2\rangle.$ if $f(En_1)n_2$ then Some $(En_1)$ else None enumerates $p$.
3. Let $f : \mathbb{N} \to \mathcal{O}(X)$ be an enumerator for $p$ and $X$ discrete. Then $\lambda xn.$ if $fn$ is Some $x_1$ then $\ulcorner x = x_1 \urcorner$ else false is a semidecider for $p$.

4. Let $f : \mathbb{N} \to \mathcal{O}(X)$ be an enumerator for $p$ and assume $px_0$ for some $x_0$. Then $\lambda n.\ \mathsf{if}\ fn\ \mathsf{is\ Some}\ x\ \mathsf{then}\ x\ \mathsf{else}\ x_0$ is a strong enumerator.

5. Straightforward. □

Besides the advantages of the synthetic approach, our work is also made simpler by the possibility to work based on the type theory directly on a logical level in order to show for example closure properties.

**Lemma 3.10 (Closure Properties)**

1. *Decidable predicates are closed under complement, conjuction, and disjunction.*

2. *Semidecidable predicates are closed under conjunction and disjunction.*

**Proof** 1. Straightforward by Lemma 3.8.1 and combining the logical operations respectively to obtain certifying deciders.

2. Let $f_p$ and $f_q$ be semideciders for predicates $p$ and $q$ respectively.
The semidecidability of $p \vee q$ is straightforward by $\lambda xn.\ (f_p xn) \vee_{\mathbb{B}} (f_q xn)$.
In order to show $p \wedge q$ semidecidable, we use bounded search on the index $n$ to define corresponding monotone semideciders $f'_p$ and $f'_q$ such that

   (a) $\forall xn.\ fxn = \mathsf{true} \to f'xn = \mathsf{true}$,

   (b) $\forall xn.\ f'xn = \mathsf{true} \to \exists n_0.fxn_0 = \mathsf{true}$, and

   (c) $\forall xn_1.\ f'xn_1 = \mathsf{true} \to \forall n_2 \geqslant n_1.\ f'xn_2 = \mathsf{true}$.

   Then, $\lambda xn.\ (f'_p xn) \wedge_{\mathbb{B}} (f'_q xn)$ is a desired semidecider for $p \wedge q$. □

Continuing with reductions, we can show that all reducibility notions form a preorder (a reflexive and transitive relation) on predicates. Therefore, their by definition furthermore symmetric computability degrees form an equivalence relation, dividing predicates into different computability classes. Again, we will not have to argue about computability of now the constructed reduction functions.

**Lemma 3.11 (Preorder & Equivalence Relation)**

1. $\preceq_1$, $\preceq_m$, *and* $\preceq_{tt}$ *are reflexive and transitive.*

2. $\equiv_1$, $\equiv_m$, *and* $\equiv_{tt}$ *form an equivalence relation on predicates.*

3. *Isomorphism* $\equiv$ *forms an equivalence relation on predicates over datatypes.*

**Proof**

1. Straightforward for $\preceq_1$ and $\preceq_m$ by picking the identity and composition respectively. For $\preceq_{tt}$, the query function $\lambda x.[x]$ with truth-table condition $\lambda x[b].b$ shows the reflexivity.
   The proof of transitivity is quite technical[4], we informally outline the proof. So let $p \preceq_{tt} q$ via $f_1$ and $\alpha_1$ and $q \preceq_{tt} r$ via $f_2$ and $\alpha_2$. For a given $x$, we let $f_1 x = [y_1, \dots, y_n]$ and pick the query list as
   $$(f_2 y_1) +\!\!+ \dots +\!\!+ (f_2 y_n).$$
   Given a corresponding list $L_1 +\!\!+ \dots +\!\!+ L_n$, we pick the condition value
   $$\alpha_1 x(\alpha_2 y_1 L_1, \dots, \alpha_2 y_n L_n).$$
2. Reflexivity and transitivity follow with 1., symmetry by definition.
3. Reflexivity follows with the identity and transitivity with the composition. In order to show symmetry we have to construct an inverse function for isomorphisms $f : X \to Y$, which works with Lemma 2.6 for enumerable types $X$ and discrete types $Y$. □

As mentioned and partially obvious, one can show implications between the various notions of reductions. This transfers to the notions of computability degrees and completeness with respect to the different reductions.

**Lemma 3.12 (Inclusions)**

1. $\preceq_1 \subseteq \preceq_m \subseteq \preceq_{tt}$.
2. $\equiv_1 \subseteq \equiv_m \subseteq \equiv_{tt}$. *On datatypes, we have furthermore* $\equiv \subseteq \equiv_1$.
3. $1$*-complete predicates are* $m$*-complete,* $m$*-complete predicates are* $tt$*-complete.*

**Proof**

1. $\preceq_1 \subseteq \preceq_m$ is straightforward, for $\preceq_m \subseteq \preceq_{tt}$ we assume $p \preceq_m q$ via $f$ and pick the query list $\lambda x.[fx]$ with condition $\lambda x[b].b$.
2. The first part follows with 1., the second part with Lemma 3.11.3.
3. Follows with 1. □

The above result states, that every $tt$-degree consists out of one or multiple $m$-degrees; every $m$-degree consists out of one or multiple $1$-degrees.

[4]Mainly because of our mechanization of truth-table conditions as functions over lists with particular length.

The reason for introducing the concept of reductions was to use this method in order to show further problems undecidable starting from an already proven undecidable problem. We can indeed show, that decidability transports through reductions; through some reducibility notions also semidecidability transports.

**Lemma 3.13 (Transport)**

1. *Let* $p \preceq_1 q$, *or* $p \preceq_m q$, *or* $p \preceq_{tt} q$. *Then,* $\mathcal{D}q \to \mathcal{D}p$.
2. *Let* $p \preceq_1 q$ *or* $p \preceq_m q$. *Then,* $\mathcal{S}q \to \mathcal{S}p$.

**Proof** By Lemma 3.12, it suffices to show the claims for the weakest reduction:

1. Let $p \preceq_{tt} q$ via $f$ and $\alpha$ and $\mathcal{D}q$ via $d$. Then $\mathcal{D}p$ via $\lambda x.\ \alpha x(\mathsf{map}\ d\ (fx))$.
2. Let $p \preceq_m q$ via $f$ and $\mathcal{S}q$ via $s$. Then $\mathcal{S}p$ via $\lambda xn.s(fx)n$. □

Notice especially the contrapositions of the statements in the above Lemma stating the initial motivation for reductions formally: If an undecidable (not semidecidable) predicate $p$ reduces to $q$, then also $q$ is undecidable (not semidecidable).

We can show interesting relations between reductions and the complements of predicates. Here again our constructive setting comes into play, since we need at certain points in the proofs double negation elimination. We cannot show this proof principle but have to assume stability of predicates at those points.

**Lemma 3.14** *1.* $p \preceq_1 q \to \overline{p} \preceq_1 \overline{q}$ *and* $p \preceq_m q \to \overline{p} \preceq_m \overline{q}$.
*If* $q$ *is stable, also* $p \preceq_{tt} q \to \overline{p} \preceq_{tt} \overline{q}$.

2. $\overline{p} \preceq_{tt} p$ *and for stable* $p$ *also* $p \preceq_{tt} \overline{p}$.

**Proof** 1. In the cases $\preceq_1$ and $p \preceq_m q$ via the reduction $f$, $f$ is also a reduction justifying $\overline{p} \preceq \overline{q}$. For $p \preceq_{tt} q$ via $f$ and $\alpha$, $\overline{p} \preceq_{tt} \overline{q}$ holds via $f$ and $\lambda xL.\ \alpha x(\mathsf{map}\ \neg_{\mathbb{B}}\ L)$. The stability of $q$ is necessary to show that $(fx) \mathrel{\widehat{=}}_{\overline{q}} L$ implies $(fx) \mathrel{\widehat{=}}_{q} (\mathsf{map}\ \neg_{\mathbb{B}}\ L)$.

2. $\overline{p} \preceq_{tt} p$ and $p \preceq_{tt} \overline{p}$ holds via $\lambda x.[x]$ and $\lambda x[b].\neg_{\mathbb{B}} b$. In the second case, the stability is necessary to show that $\overline{p}x \leftrightarrow b = \mathsf{true}$ implies $px \leftrightarrow \neg_{\mathbb{B}} b = \mathsf{true}$. □

Lastly, we show that many-one and truth-table reductions are not only a preorder on predicates, but form an upper semilattice, i.e. every two predicates have a least upper bound in the sense of the preorder formed by reductions[5]. Type theory allows an easy definition of this least upper bound also called join by using the sum type.

[5]Semilattices as well as lattices are in order theory actually only defined on partially ordered sets. We can state the existence of a least upper bound formally also for our preorder.

**Lemma 3.15** *For all predicates $p$ and $q$, there exists a least upper bound with respect to many-one and truth-table reductions, i.e.*

$$\forall pq.\, \exists j.\, p \preceq j \land q \preceq j \land \forall r.\, p \preceq r \to q \preceq r \to j \preceq r$$

*when replacing $\preceq$ consistently with $\preceq_m$ or $\preceq_{tt}$.*

**Proof** Let $p : X \to \mathbb{P}$ and $q : Y \to \mathbb{P}$. We define the join of $p$ and $q$ as follows

$$\begin{aligned} &\mathsf{join}\, p\, q : X + Y \to \mathbb{P} \\ &\mathsf{join}\, p\, q(Lx) := px \\ &\mathsf{join}\, p\, q(Ry) := qy \end{aligned}$$

$\mathsf{join}\, p\, q$ is indeed the least upper bound of $p$ and $q$ w.r.t. $\preceq_m$ (1.) and $\preceq_{tt}$ (2.):

1. $p \preceq_m \mathsf{join}\, p\, q$ via $L$ and $p \preceq_m \mathsf{join}\, p\, q$ via $R$. Furthermore $p \preceq_m r$ via $f_1$ and $q \preceq_m r$ via $f_2$ implies $\mathsf{join}\, p\, q \preceq_m r$ via $\lambda s.\, \mathsf{match}\ s\ [Lx \Rightarrow f_1 x \mid Ry \Rightarrow f_2 y]$.
2. $p \preceq_{tt} \mathsf{join}\, p\, q$ and $p \preceq_{tt} \mathsf{join}\, p\, q$ follow with 1. by Lemma 3.12. For $p \preceq_{tt} r$ via $f_1$ and $\alpha_1$ and $q \preceq_{tt} r$ via $f_2$ and $\alpha_2$, $\mathsf{join}\, p\, q \preceq_m r$ follows via:
   $\lambda s.\, \mathsf{match}\ s\ [Lx \Rightarrow f_1 x \mid Ry \Rightarrow f_2 y]$ and $\lambda sL.\, \mathsf{match}\ s\ [Lx \Rightarrow \alpha_1 x L \mid Ry \Rightarrow \alpha_2 y L]$.

□

Since the reduction function $\lambda s.\, \mathsf{match}\ s\ [Lx \Rightarrow f_1 x \mid Ry \Rightarrow f_2 y]$ is not necessarily injective, $\mathsf{join}\, p\, q$ does not form analogously a least upper bound for $p$ and $q$ also with respect to one-one reducibility.

## 3.3 Axioms of Synthetic Computability Theory

We showed numerous generic facts regarding the introduced computability theory notions. Furthermore, we could also show various concrete positive results in our setting by for instance showing different predicates (semi-)decidable. However, the axiom-free synthetic setting will not allow us to show concrete negative results. We do not have a model of computation in hand that allows us methods like diagonalization to show as in traditional presentations predicates like the halting problem undecidable.

It is even worse, since we cannot even define predicates working on instances of a concrete computational model or its encodings. Since functions in Coq's type theory are also guaranteed to be total, there is also no possibility to define a variant of the halting problem via the definedness of a given function.

In order to formalize results like Post's problem stating the existence of a particular undecidable predicate as well as distinctions of different reducibility notions on the class of semidecidable but undecidable predicates we are forced to construct undecidable predicates.

Since this is as mentioned not possible in an axiom-free synthetic setting, we will need to assume carefully chosen axioms at certain points of the development. We want to introduce the axioms used in our work and argue those axioms to be faithful with traditional computability theory as well as consistent in our setting. Therefore, the section is partly informal and uses at some points traditional notations to show the analogy to traditional computability. The parts that are marked as definitions or lemmas are formalized and mechanized and can therefore be used in the further development. Works by Richman[30] or Bauer[3] discuss the foundations of axioms of synthetic computability more closely as well as Forster [12], who furthermore mechanized various results exactly in our synthetic setting.

The fundamental axiom of synthetic computability theory assumes similar to the essential property of concrete computational models a universal machine working on program indices. The axiom called Church's thesis[6] now states, that for every computable function there exists a corresponding index computing this function.

Formally one can use a step-interpreter function

$$T : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \mathcal{O}\mathbb{N}$$

that computes for encodings $c$ of for instance $\lambda$-terms, inputs $x$ and step indices $n$ in the case of termination after $n$ steps the output. This computation behavior implies $T$ to be monotone in the step index, i.e.

$$Tcxn_1 = \mathsf{Some}\ y \to \forall n_2 \geqslant n_1.\, Tcxn_2 = \mathsf{Some}\ y.$$

Based on $T$, one defines a program index $c : \mathbb{N}$ to compute a certain total function $f : \mathbb{N} \to \mathbb{N}$ as computability relation

$$c \sim f := \forall x. \exists n. Tcxn = \mathsf{Some}\ (fx).$$

Finally, the axiom of Church's thesis for total functions is then defined as:

$$\mathsf{CT} := \forall f.\ \exists c.\ c \sim f.$$

Adapting the presentation to obtain **CT** also for partial functions is on paper straightforward; mechanizing this in Coq requires however naturally to deal with partial functions that are not easy possible to imitate.

One can show **CT** to imply weaker computability axioms, by for instance using $T$ to define a function $\varphi : \mathbb{N} \to (\mathbb{N} \to \mathcal{O}(\mathbb{N}))$ as $\varphi c \langle x, n \rangle := Tcxn$, that enumerates all

[6]not to confuse with the Church-Turing thesis of traditional computability theory

enumerators $f : \mathbb{N} \to \mathcal{O}(\mathbb{N})$ modulo range equivalence, i.e. for every enumerator $\varphi$ hits an enumerator with the same range:

$$\forall f.\ \exists c.\ \forall x.\ (\exists n. \varphi c n = \mathsf{Some}\ x) \leftrightarrow (\exists n. f n = \mathsf{Some}\ x).$$

Forster [12] formalizes type theoretically that the enumerability of enumerators modulo their ranges allows furthermore to derive Bauer's enumerability axiom (**EA**) assuming an enumerator for the class of enumerable predicates. This enumerator $\mathcal{W} : \mathbb{N} \to (\mathbb{N} \to \mathbb{P})$ is defined using $\varphi$ as $\mathcal{W}cx := \exists n. \varphi c n = \mathsf{Some}\ x$. We define the specification of $\mathcal{W}$ equivalently as the enumerability of semidecidable predicates denoted by **ES**:

$$\mathbf{ES} := \forall p : \mathbb{N} \to \mathbb{P}.\ \mathcal{S}p \leftrightarrow \exists c. \forall x. (\mathcal{W}cx \leftrightarrow px).$$

One can think about $\mathcal{W}$ as a variant of the halting problem with the first argument as the encoding of programs and the second as their inputs: "$\mathcal{W}cx$ holds iff the program with index $c$ terminates on input $x$." Furthermore, we can define the special halting problem $\mathcal{W}_0$ using $\mathcal{W}$ as $\mathcal{W}_0 c := \mathcal{W}cc$.

**ES** allows us to prove further basic properties of $\mathcal{W}$. By assuming $\mathcal{W}$, we add especially a first undecidable predicate to our constructive setting, such that assuming all propositions to be decidable would be no longer consistent.

**Lemma 3.16** *Assume **ES**, then*

1. *there exists* $c_\top$ *with* $\forall x.\ \mathcal{W}c_\top x$.
2. *there exists* $c_\bot$ *with* $\forall x.\ \neg\mathcal{W}c_\bot x$.
3. $\mathcal{W}_0 \preceq_1 \mathcal{W}$.
4. $\overline{\mathcal{W}_0}$ *is not semidecidable and* $\mathcal{W}_0$ *undecidable.*
5. $\overline{\mathcal{W}}$ *is not semidecidable and* $\mathcal{W}$ *undecidable.*

**Proof**
1. Follows by **ES** with the semidecidability of $\lambda x.\top$.
2. Analog to 1. with $\lambda x.\bot$.
3. Via the injection $\lambda c.(c, c)$.
4. Assume $\mathcal{S}\overline{\mathcal{W}_0}$. By **ES**, there is an index $c$ with $\forall x. \mathcal{W}cx \leftrightarrow \overline{\mathcal{W}_0}x$. For $x := c$ a contradiction, since $\mathcal{W}cc \leftrightarrow \overline{\mathcal{W}}cc$. Therefore, $\overline{\mathcal{W}_0}$ is not semidecidable and $\mathcal{W}_0$ with Lemma 3.9 undecidable.
5. Follows with 3. and 4. by Lemma 3.13. □

The above Lemma justifies our interpretation of $\mathcal{W}$ as the halting problem in a sense, that it shows some well-known properties of the halting problem for $\mathcal{W}$. Unfortunately, another prominent and important property of the halting problem, namely its semidecidability, cannot be concluded from **ES**.

One solution would be to assume some stronger computability axiom or an additional weak choice axiom. By using $\varphi$, we could easily define a semidecider for $\mathcal{W}$ as $\lambda(c, x)n.\ \varphi cn = \mathsf{Some}\ x$.

It will however turn out that our aspired construction of a simple predicate does not force us to assume full **CT** but works out by assuming $\mathcal{W}$ and its semidecidability. In order to base the development on the weakest possible set of axioms, we will therefore simply assume $\mathcal{SW}$ at some point, which allows us furthermore to prove the completeness of $\mathcal{W}$:

**Lemma 3.17** *Assume* ***ES*** *and* $\mathcal{SW}$. *Then* $\mathcal{W}$ *is* 1*-complete and thus* $\mathrm{m}$*- and* $\mathrm{tt}$*-complete.*

**Proof** $\mathcal{SW}$ follows by assumption. By the definition of 1-completeness, we now have to consider semidecidable predicates over all isomorphic and discrete types, but we omit the technical details of coding and encoding here[7]. For a semidecidable predicate $p : \mathbb{N} \to \mathbb{P}$, **ES** yields an index $c$ with $\forall x. \mathcal{W}cx \leftrightarrow px$. Therefore, $p \preceq_1 \mathcal{W}$ via the injection $\lambda x.(c, x)$. $\square$

We have shown our formalization of the halting problem $\mathcal{W}$ to be a predicate with maximal computability degree on the class of semidecidable predicates. Thinking about Post's problem, this implies the existence of a semidecidable but undecidable predicate $p$ with $\mathcal{W} \not\preceq p$ to be equivalent to the existence of a semidecidable but undecidable non-complete predicate. We prove this equivalence in its positive formulation:

**Lemma 3.18** *Assume* ***ES***, $\mathcal{SW}$ *and let* $X$ *be discrete,* $\mathbb{N} \cong X$ *and* $p : X \to \mathbb{P}$ *semidecidable. For all introduced notions of reducibility* $\preceq$*, we have*

$$\mathcal{W} \preceq p \leftrightarrow \mathsf{complete}\ p.$$

**Proof** For $\mathcal{W} \preceq p$, $p$ is complete by the transitivity of reductions and 3.17. Conversely, $\mathcal{W}$ reduces as a semidecidable predicate to $p$. $\square$

Another type of axioms concerns the computability of particular program indices. It is for instance easily possible to show with a concrete model of computation and its concrete encoding function in hand, that one can compute the indices of programs halting exactly on a finite list of inputs. Such a computation would work by computing the index of a program that easily hardcodes the list of accepted inputs.

[7]See chapter 7 for further details.

This is in our synthetic setting again not provable, but we have to assume this abstract property of computational models. One way to do so is to assume a concatenation operator $\mathsf{cns} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$, that given an element $a$ and an index $c$ halting exactly on the elements of a list $L$ computes an index halting exactly on elements of $a :: L$. Formally we specify $\mathsf{cns}$ by

$$\forall cL.\ (\forall x.\mathcal{W}cx \leftrightarrow x \in L) \to \forall ax.\mathcal{W}(\mathsf{cns}\,a\,c)x \leftrightarrow x \in (a :: L).$$

Together with the index $c_\perp$ from Lemma 3.16, we can show that given a list $L$ one can compute an index halting exactly on the elements of $L$ using $\mathsf{cns}$.

**Lemma 3.19** *Assume the* $\mathsf{cns}$*-operator with its specification and an index* $c_\perp$ *with* $\forall x.\ \neg\mathcal{W}c_\perp x$. *Then one can compute for all lists* $L$ *an index* $c_L$ *such that*

$$\forall x.\mathcal{W}c_L x \leftrightarrow x \in L.$$

**Proof** We prove the claim by induction on $L$. In the base case we choose $c_{[]} = c_\perp$, in the step case $(a :: L)$, the inductive hypothesis yields an index $c_L$ and $\mathsf{cns}\,a\,c_L$ is the correct new index. □

Lastly, the proof of Post's problem for many-one reductions uses at a certain point the traditionally well-known $S^m_n$-theorem. The theorem also establishes a particular computation of program indices that can be stated using traditional notation as

$$\Sigma S : \mathbb{N} \to \mathbb{N} \to \mathbb{N}.\forall cxy.\ \phi_c\langle x, y\rangle = \phi_{(Scx)}y,$$

where $\phi_c : \mathbb{N} \rightharpoonup \mathbb{N}$ is the calculating function for a program index $c$ and is in particular allowed to be partial.

We will again not need the theorem in its full strength, but are fine by using a strictly weaker corollary. This can be deduced from an even simpler version of the $S^m_n$-theorem presented for example by Cutland [8], who formulates it as[8]

$$\text{Simple } S^m_n\text{: } \forall f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}.\ \exists k : \mathbb{N} \to \mathbb{N}.\ \forall cx.\ f(c, x) = \phi_{(kc)}x.$$

This simpler version directly implies a special case of the $S^m_n$-theorem: Given a total function $f' : \mathbb{N} \to \mathbb{N}$, we use the theorem for $f(c, x) := \phi_c(f'x)$ and obtain

$$\forall f : \mathbb{N} \to \mathbb{N}.\ \exists k : \mathbb{N} \to \mathbb{N}.\ \forall cx.\ \phi_c(fx) = \phi_{(kc)}x.$$

Requiring $\phi_c(fx) = \phi_{(kc)}x$ for the partial function $\phi_c$ implies in particular that the program with index $c$ halts on input $fx$ if and only if the program with index $kc$ halts

[8]We omit that the function $f$ is in the traditional formulation required to be computable at this anyway already informal point mixing the traditional and synthetic setting.

on $x$. This allows us to return now again to the purely synthetic formalization: the intuition of a program with index $c$ halting on input $x$ is as discussed exactly our interpretation of $\mathcal{W}cx$ such that we can formally define a weak axiom related to the $S^m_n$-theorem as follows:

$$\mathsf{SMN'} : \forall f : \mathbb{N} \to \mathbb{N}.\ \exists k : \mathbb{N} \to \mathbb{N}.\ \forall cx.\ \mathcal{W}c(fx) \leftrightarrow \mathcal{W}(kc)x.$$

We discussed how to derive $\mathcal{W}$ and its semidecidability formally from the fundamental synthetic computability axiom $\mathsf{CT}$. It should be also possible to deduce the $\mathsf{cns}$-operator as well as the $S^m_n$-theorem from $\mathsf{CT}$. Thereby, it seems however necessary to use the partial version of $\mathsf{CT}$ for showing $\mathsf{SMN'}$, since the axiom bases on the partiality of the calculating function $\phi_c$[9]. This room for future work will be discussed in more detail in chapter 8.

Notice, that we formulated all axioms in this chapter for predicates and programs over the natural numbers. It is also possible to assume the axioms for arbitrary types, e.g. $\varphi : \forall X.\ \mathbb{N} \to (\mathbb{N} \to \mathcal{O}(X))$ or $\mathcal{W} : \forall X.\ \mathbb{N} \to X \to \mathbb{P}$ with the corresponding specifications with $x$ as an element of a general type $X$.

We will come back and use the introduced axioms in chapter 6 when it becomes necessary to make further assumptions; the particular points at which synthetic axioms are added to our setting are clearly marked.

Since we will not use classical proof principles or choice axioms in the further work, assuming those axioms that seem to follow as discussed from $\mathsf{CT}$ seems to be consistent in our setting (cf. Swan and Uemura [33]). We argued furthermore that all the axioms represent abstract properties corresponding to properties of concrete computational models, such that our synthetic setting containing axioms stays faithful with traditional computability theory.

## 3.4 Recursive $\mu$-Operator

In the computational models used in traditional presentations of computability theory it is possible to implement a so-called recursive $\mu$-operator, which is for instance directly included in the definition of $\mu$-recursive functions. This operator proceeds an upwards search on natural numbers: it returns the least number satisfying a particular decidable condition or is undefined if there exists no such number.

Coq's programming language will not allow us to implement such a strong $\mu$-operator in general. We only deal with total functions in Coq's inductive system and can therefore not define a partial $\mu$-function.

However, we can obtain similarly to the witness operator (Lemma 2.4) a variant of a recursive $\mu$-operator in Coq that given an additional witness for the existence of

[9]For total $\phi_c$, the program with index $c$ always terminates and $\mathsf{SMN'}'$ becomes trivial but useless.

an element satisfying a particular predicate returns the least number satisfying the predicate. The witness serves as a kind of termination argument for the traditional "unbounded" and therefore potentially diverging search algorithm.

**Lemma 3.20** *For predicates* $p : \mathbb{N} \to \mathbb{P}$ *with certifying decider* $f : \forall x.px + \neg px$*, there is a recursive operator* $\mu_\mathbb{N} : (\exists n.pn) \to \mathbb{N}$*, that computes the least number satisfying* $p$*, i.e.*

$$\forall H : (\exists n.pn).\ p(\mu_\mathbb{N} H) \land \forall n.\ pn \to \mu_\mathbb{N} H \leqslant n.$$

**Proof** The proof works by a small adaption of the proof of the witness operator in Lemma 2.4. We only have to adapt claim 1. in the Proof 2.4 now stating that $Gn$ allows a computation of the least number greater or equal $n$ satisfying $p$, formally:

1. $Gn \to \Sigma x \geqslant n.\ px \land \forall y \geqslant n.\ py \to x \leqslant y$, which follows again by induction on $G$[10].

$\mu_\mathbb{N}$ then follows, since we used claim 1. for $n = 0$. □

By the specification of the number computed by $\mu_\mathbb{N}$ as the least number satisfying the predicate, this number must be unique and especially independent of the particular witness proof.

**Lemma 3.21** *For all proofs* $H_1$ *and* $H_2$ *of* $\exists n.pn$ *:* $\mu_\mathbb{N} H_1 = \mu_\mathbb{N} H_2$.

**Proof** By $p(\mu_\mathbb{N} H_1)$ and $p(\mu_\mathbb{N} H_2)$, we derive $\mu_\mathbb{N} H_1 \leqslant \mu_\mathbb{N} H_2 \leqslant \mu_\mathbb{N} H_1$. □

We would like to further improve this result by finding a minimization operator for not only decidable but also enumerable predicates. Unfortunately, such a computation quickly becomes inconsistent in traditional computability theory as well as in our synthetic setting when only adding weak additional axioms. We can show, that assuming a recursive $\mu$-operator for enumerable predicates with the exact same specification as $\mu_\mathbb{N}$ implies all semidecidable predicates to be already decidable.

**Lemma 3.22** *Assume for predicates* $p : \mathbb{N} \to \mathbb{P}$ *with enumerator* $f : \mathbb{N} \to \mathcal{O}(\mathbb{N})$ *an operator* $\hat{\mu}_\mathcal{E} : (\exists n.pn) \to \mathbb{N}$*, such that*

$$\forall H : (\exists n.pn).\ p(\hat{\mu}_\mathcal{E} H) \land \forall n.\ pn \to \hat{\mu}_\mathcal{E} H \leqslant n.$$

*Then, all semidecidable predicates are decidable.*

**Proof** We assume $\hat{\mu}_\mathcal{E}$. Let $p := \lambda f : \mathbb{N} \to \mathbb{N}.\ \exists n.fn = 0$, then:

1. $\mathcal{D}p$: Given a function $f$, the predicate $\lambda y.\exists n.fn = y$ (i.e. the range of $f$) is enumerable via $\lambda n.\mathsf{Some}\ (fn)$. Furthermore, there is a witness for $\exists y.\exists n.fn = y$ by for example $f0$. Therefore, $\hat{\mu}_\mathcal{E}$ computes the least number in the range of $f$. If this number is $0$, $pf$ holds; if it is larger $\neg pf$ holds.

[10]for the in contrast to Proof 2.4 no longer constant predicate $qn := \Sigma x \geqslant n.\ px \land \forall y.\ py \to x \leqslant y$

2. $(\lambda f : \mathbb{N} \to \mathbb{B}.\ \exists n.fn = \mathsf{true}) \preceq_m p$ via the reduction $\lambda f.(\lambda n.\mathsf{if}\ fn\ \mathsf{then}\ 0\ \mathsf{else}\ 1)$.
3. $\mathcal{D}(\lambda f.\exists n.fn = \mathsf{true})$ with 1. and 2. by Lemma 3.13. We call the decider $d$.

Now let $p$ be semidecidable via $f : X \to \mathbb{N} \to \mathbb{B}$. Then, $\lambda x.\ d(fx)$ is a decider for $p$. □

Clearly, the decidability of boolean satisfiability as well as for instance the decidability of the semidecidable halting problem are inconsistent with traditional computability theory. In our setting, assuming the above operator becomes also formally contradictory when working with the by **ES** undecidable predicate $\mathcal{W}$ after assuming its semidecidability.

However, we are able to weaken the specification of the recursive $\mu$-operator for enumerable predicates in a way that it becomes on the one hand provable but stays on the other hand strong enough for our purposes. Given a predicate and a witness, we are actually only interested in the possibility to compute a unique[11] element satisfying the predicate. Using the already introduced $\mu_{\mathbb{N}}$ operator, we can show this unique computation not only for enumerable predicates on natural numbers, but for enumerable predicates over arbitrary types.

**Lemma 3.23** *For predicates $p : X \to \mathbb{P}$ with an enumerator $f : \mathbb{N} \to \mathcal{O}(X)$, there is a recursive operator $\mu_{\mathcal{E}} : (\exists x.px) \to X$, that computes a unique element satisfying $p$, i.e.*

$$\forall H_1 H_2 : (\exists x.px).\ p(\mu_{\mathcal{E}} H_1) \land \mu_{\mathcal{E}} H_1 = \mu_{\mathcal{E}} H_2.$$

**Proof** Follows with $\mu_{\mathbb{N}}$ used for the decidable predicate $\lambda n.\ \exists x.fn = \mathsf{Some}\ x$ and Lemma 3.21. □

By using $\mu_{\mathbb{N}}$ in the above mentioned way, we actually minimize the argument of the enumerator $f$: We compute the least $n$ such that $fn = \mathsf{Some}\ x$ and pick this unique $x$ as the desired element. This computation is trivial for predicates with even strongly enumerators $f$ by simply picking $f0$.

$\mu_{\mathcal{E}}$ implies the existence of a witness operator $\omega_{\mathcal{E}}$ for predicates with enumerator as well as similarly to Corollary 2.5 a $\mu_X$-operator for predicates with certifying decider over enumerable types. Notice however, that analogously to the operators $\omega_{\mathbb{N}}$ and $\omega_X$ also $\omega_{\mathcal{E}}$, $\mu_{\mathbb{N}}$, $\mu_X$, and $\mu_{\mathcal{E}}$ require us to have the deciders and/or enumerators in hand. The possibility to compute a unique element in predicates via $\mu$-operators will become essential especially in chapter 6 when constructing a simple predicate.

[11] in the sense of Lemma 3.21

# Chapter 4

# Reduction Characterizations

In the last chapter, we introduced the approach of synthetic computability theory, defined basic notions in this setting, and showed several well-known properties of those notions. Our goal in this chapter is now to take a closer look at the properties of the different notions of reductions and formalize certain relations between those reductions and their computability degree.

## 4.1 Myhill's Isomorphism Theorem

We start by formalizing Myhill's isomorphism theorem (Myhill [25]), stating that predicates are recursive isomorphic if and only if they have the same computability degree with respect to one-one reductions. Myhill's isomorphism theorem reminds of the Cantor-Schröder-Bernstein theorem, turning injections $f : A \to B$ and $g : B \to A$ into a bijection between $A$ and $B$. While this theorem does not speak about computability but relies (at least in this formulation) highly on classical notation and choice-axioms, we can prove Myhill's isomorphism theorem purely constructive and even without assuming synthetic axioms.

Showing that two isomorphic predicates have also the same 1-degree was already shown[1] in the last chapter by the basic inclusion properties. In order to construct an isomorphism between predicates with the same 1-degree, we loosely follow the presentation of Myhill himself and Rogers [31] by using so-called (finite) correspondence sequences[2].

Correspondence sequences defined with respect to predicates $p$ and $q$ are lists of pairs whose both projections are duplicate-free and that are pointwise applied to $p$ and $q$ respectively equivalent.

Recall for the formalization, the functions $\pi_1$, $\pi_2$, and $\overset{\leftrightarrow}{\cdot}$ working on lists of pairs, which were defined in chapter 2.4.

[1]for predicates over datatypes

[2]Not confuse with corresponding lists used in the context of truth-table reductions.

**Definition 4.1** *We call a list* $\mathsf{L} : \mathcal{L}(\mathsf{X} \times \mathsf{Y})$ *a **correspondence sequence** for predicates* $\mathsf{p} : \mathsf{X} \to \mathbb{P}$ *and* $\mathsf{q} : \mathsf{Y} \to \mathbb{P}$ *if*

$$\#(\pi_1 \mathsf{L}) \land \#(\pi_2 \mathsf{L}) \land \forall (x, y) \in \mathsf{L}.\ \mathsf{p}x \leftrightarrow \mathsf{q}y.$$

We establish basic properties of those correspondence sequences.

**Lemma 4.2** *For all correspondence sequences* $\mathsf{L}$ *for predicates* $\mathsf{p}$ *and* $\mathsf{q}$*,*

1. $\overset{\leftrightarrow}{\mathsf{L}}$ *is a correspondence sequence for* $\mathsf{q}$ *and* $\mathsf{p}$*.*
2. *we have the following one-one properties:*
   - $(x, y_1) \in \mathsf{L} \to (x, y_2) \in \mathsf{L} \to y_1 = y_2$ *for all* $x, y_1$*, and* $y_2$*.*
   - $(x_1, y) \in \mathsf{L} \to (x_2, y) \in \mathsf{L} \to x_1 = x_2$ *for all* $x_1, x_2$*, and* $y$*.*

**Proof** Let $\mathsf{L}$ be a correspondence sequence for $\mathsf{p}$ and $\mathsf{q}$.

1. $\pi_1 \overset{\leftrightarrow}{\mathsf{L}} = \pi_2 \mathsf{L}$ and $\pi_2 \overset{\leftrightarrow}{\mathsf{L}} = \pi_1 \mathsf{L}$ are duplicate-free by assumption. $(y, x) \in \overset{\leftrightarrow}{\mathsf{L}}$ implies $(x, y) \in \mathsf{L}$ implies $\mathsf{q}y \leftrightarrow \mathsf{p}x$.
2. Both claims are proven by induction on $\mathsf{L}$. In the step case, we distinguish the four cases obtained in the first claim by $(x, y_1), (x, y_2) \in (x_0, y_0) :: \mathsf{L}$ (second claim symmetric) and use $\#(x_0 :: \pi_1 \mathsf{L})$ and $\#(y_0 :: \pi_2 \mathsf{L})$ to close those cases. □

Notice that the proof of the one-one property of a correspondence sequence uses only that both projections were assumed to be duplicate-free. In fact, those properties are even equivalent: A list of pairs is duplicate-free in both projections if and only if it fulfills the above stated one-one property. Therefore, this one-one property together with the pointwise equivalence of $\mathsf{p}$ and $\mathsf{q}$ would have been an equivalent way to define correspondence sequences.

We continue to work over discrete types $\mathsf{X}$ and $\mathsf{Y}$. This allows not only to decide list membership in both the projections and the lists of pairs itself, but given $x : \mathsf{X}$ also to compute (if existent) a corresponding element $y : \mathsf{Y}$ (i.e. $(x, y) \in \mathsf{L}$) and vice versa.

Thus, we can travel for a fixed list $\mathsf{L} : \mathcal{L}(\mathsf{X} \times \mathsf{Y})$ and function $f : \mathsf{X} \to \mathsf{Y}$ through the list in a certain way: Given a starting value $x$, we check whether $fx$ is in the second projection. If so, we continue the travel with a to $fx$ corresponding new starting value. We compute the trace of this travel recursively for a given step index:

$$\begin{aligned}
\mathsf{trace}\ x\ n &:= [\,] && \text{if } fx \notin \pi_2 \mathsf{L} \\
\mathsf{trace}\ x\ 0 &:= [x] && \text{if } (x', fx) \in \mathsf{L} \text{ for some } x' \\
\mathsf{trace}\ x\ (\mathsf{S}n) &:= x :: (\mathsf{trace}\ x'\ n) && \text{if } (x', fx) \in \mathsf{L} \text{ for some } x'
\end{aligned}$$

Especially for one-one reductions $\mathsf{f}$ and correspondence sequences $\mathsf{L}$, this computed trace has very particular properties. First of all, the one-one properties of both $\mathsf{f}$ and $\mathsf{L}$ imply the trace to be duplicate-free, if the projection does not contain the starting value:

**Lemma 4.3** *Let* $p \preceq_1 q$ *via* $f$ *and* $L$ *be a correspondence sequence for* $p$ *and* $q$*. For all starting values* $x$ *and step indices* $n$*,*

1. $\forall x' \in \mathsf{trace}\ x\ n.\ \ x' = x \lor \exists y' \in (\mathsf{map}\ f\ (\mathsf{trace}\ x\ n)).\ (x', y') \in L.$
2. *if* $x \notin (\pi_1 L)$ *then* $\#(\mathsf{trace}\ x\ n)$*.*

**Proof** Let $x : X$ and $n : \mathbb{N}$.

1. Follows by induction on $n$ with generalized starting value $x$. In the case $(x', fx) \in L$, we use the inductive hypothesis for the starting value $x'$.
2. We prove the stronger claim

$$\big(\forall y.(x, y) \in L \to y \notin (\mathsf{map}\ f\ (\mathsf{trace}\ x\ n))\big) \to \#(\mathsf{trace}\ x\ n).$$

   by induction on $n$ again with generalized starting value $x$.
   The base case and the step case for $fx \notin \pi_2 L$ are trivial. In the step case with $(x', fx) \in L$ for some $x'$, we assume $H : \forall y.(x, y) \in L \to y \notin (\mathsf{map}\ f\ (\mathsf{trace}\ x\ n))$ and have to show $x :: (\mathsf{trace}\ x'\ n)$ to be duplicate-free.

   - $x \notin (\mathsf{trace}\ x'\ n)$: Assuming $x \in (\mathsf{trace}\ x'\ n)$ implies by 1. either $x = x'$ and therefore $(x, fx) \in L$ or $(x, y') \in L$ for some $y' \in (\mathsf{map}\ f\ (\mathsf{trace}\ x'\ n))$. Both contradicts $H$ for $fx$ or $y'$ respectively.
   - $\#(\mathsf{trace}\ x'\ n)$: We apply the inductive hypothesis for the starting value $x'$ and are left to show $\forall y.(x', y) \in L \to y \notin (\mathsf{map}\ f\ (\mathsf{trace}\ x'\ n))$. But since $(x', fx) \in L$, assuming $(x', y) \in L$ implies $fx = y$ by Lemma 4.2. Therefore by the injectivity of $f$,

$$y \notin (\mathsf{map}\ f\ (\mathsf{trace}\ x'\ n)) \text{ iff } x \notin (\mathsf{trace}\ x'\ n), \qquad \text{(Lemma 2.8)}$$

     which was already shown in the above point. □

Notice the crucial and not obviously to found generalization of the induction statement proving the trace to be duplicate-free. A naive induction has no chance to succeed: By construction, the new starting value in the step case $x'$ corresponds to $fx$ such that $x' \in \pi_1 L$. Therefore, the inductive hypothesis would not be applicable since its premise requiring $x' \notin \pi_1 L$ would not be satisfied.

In the special way we navigate through the list, this travel has besides the step index another termination condition: If $fx$ is not in the projection, we cannot find a corresponding element $x'$ to continue the travel. For reductions $f$ and corresponding sequences $L$ we can however use this case to compute an element with certain properties, relying on the fact that $f$ and $L$ respect the particular predicates. If in the other case, the travel terminates only due to the step index, we know the exact length of our trace:

**Lemma 4.4** *Let $p \preceq_1 q$ via $f$ and $L$ be a correspondence sequence for $p$ and $q$. Given a starting value $x$ and step index $n$,*

$$\textit{either one can compute } y \notin \pi_2 L \textit{ with } px \leftrightarrow qy \qquad \textbf{\textit{or}} \qquad |\mathsf{trace}\ x\ n| = Sn.$$

**Proof** The claim is again proven by induction on $n$ with generalized starting value. For $fx \notin \pi_2 L$, $y := fx$ is the desired element since $p \preceq_1 q$ via $f$.
For $(x', fx) \in L$, the base case is trivial by $|[x]| = 1$, in the step case the inductive hypothesis yields either an element $y \notin \pi_2 L$ with

$$\begin{aligned} px &\leftrightarrow q(fx) && p \preceq_1 q \text{ via } f \\ &\leftrightarrow px' && (x', fx) \in L \\ &\leftrightarrow qy. && \text{IH} \end{aligned}$$

In the other case of the inductive hypothesis,

$$|x :: (\mathsf{trace}\ x'\ n)| = S(|\mathsf{trace}\ x'\ n|) \overset{\text{IH}}{=} S(Sn). \qquad \square$$

We want to use this computation to extend correspondence sequences with further elements. However, the second case of the above lemma yields no computation at all. This problem is solved by the already proven fact that the trace is duplicate-free and therefore does not loop. Hence, the travel through the list will eventually terminate always in the first case at least when using a large enough amount of fuel.

**Lemma 4.5** *Let $p \preceq_1 q$ via $f$ and $L$ be a correspondence sequence for $p$ and $q$. Given $x$, one can compute a correspondence sequence $L'$ for $p$ and $q$ with $L \subseteq L'$ and $x \in \pi_1 L'$.*

**Proof** For $x \in L$, we are done by picking $L' := L$. So assume $x \notin \pi_1 L$ and use Lemma 4.4 for $x := x$ and $n := |L|$; we have two cases:

1. For the computed element $y \notin \pi_2 L$ with $px \leftrightarrow qy$, $L' := (x, y) :: L$ is again a correspondence sequence for $p$ and $q$.
2. $|\mathsf{trace}\ x\ |L|| = S(|L|)$ is contradictory: By the definition of trace, we know $\mathsf{map}\ f\ (\mathsf{trace}\ x\ |L|) \subseteq (\pi_2 L)$ and with $x \notin \pi_1 L$ Lemma 4.3 implies $\#(\mathsf{trace}\ x\ |L|)$.

Since $f$ is injective, also $\#(\mathsf{map}\ f\ (\mathsf{trace}\ x\ |L|))$ by Lemma 2.8. Therefore, the pigeonhole principle 2.10 implies

$$|\mathsf{trace}\ x\ |L|\,| = |\mathsf{map}\ f\ (\mathsf{trace}\ x\ |L|)| \leqslant |\pi_2 L| = |L|,$$

but we assumed $|\mathsf{trace}\ x\ |L|\,| = S(|L|) > |L|$. □

We come back to the actual statement of Myhill's isomorphism theorem and prove it firstly only for predicates over natural numbers. Therefore, we fix from now on two predicates $p : \mathbb{N} \to \mathbb{P}$ and $q : \mathbb{N} \to \mathbb{P}$ and their opposite one-one reductions

$$p \preceq_1 q \ \text{ via } f_1 \text{ and } q \preceq_1 p \ \text{ via } f_2.$$

The results up to this point for general discrete types are therefore now used for $\mathbb{N}$. Notice furthermore, that we can now apply the important Lemma 4.5 in two ways: Either to the reduction $f_1$ and correspondence sequences for $p$ and $q$, but also to the reduction $f_2$ and correspondence sequences now for $q$ and $p$. The following proof relies exactly on this observation and shows an even stronger extension of correspondence sequences, containing a given element now in both projections.

**Lemma 4.6** *Let $L$ be a correspondence sequence for $p$ and $q$. Given $n$, one can compute a list $L_n$ such that*

1. *$L_n$ is a correspondence sequence list for $p$ and $q$,*
2. *$L \subseteq L_n$,*
3. *$n \in \pi_1 L_n$ and $n \in \pi_2 L_n$.*

**Proof** Given $n$ and $L$, we proceed as follows:

- For $p \preceq_1 q$ via $f_1$ and the correspondence sequence $L$ for $p$ and $q$, Lemma 4.5 computes a correspondence sequence $L' \supseteq L$ for $p$ and $q$ with $n \in \pi_2 L'$.
- For $q \preceq_1 p$ via $f_2$ and the by Lemma 4.2 correspondence sequence $\overset{\leftrightarrow}{L'}$ for $q$ and $p$, Lemma 4.5 computes a correspondence sequence $L'' \supseteq \overset{\leftrightarrow}{L'}$ for $q$ and $p$ with $n \in \pi_2 L'_n$.
- We pick $L_n := \overset{\leftrightarrow}{L''}$. □

The computation from the last Lemma can be expressed as a function

$$\mathsf{extend} : \mathbb{N} \to \mathcal{L}(\mathbb{N} \times \mathbb{N}) \to \mathcal{L}(\mathbb{N} \times \mathbb{N}),$$

that given a correspondence sequence also returns an (extending) correspondence sequence.

Since $[\,]$ is a correspondence sequence for all predicates, it can serve as the starting point for a recursive computation using $\mathsf{extend}$:

$$\begin{aligned}\varphi_{\mathcal{L}} &: \mathbb{N} \to \mathcal{L}(\mathbb{N} \times \mathbb{N})\\ \varphi_{\mathcal{L}}\ 0 &:= \mathsf{extend}\ 0\ [\,]\\ \varphi_{\mathcal{L}}\ (Sn) &:= \mathsf{extend}\ (Sn)\ (\varphi_{\mathcal{L}}\ n)\end{aligned}$$

The properties of $\mathsf{extend}$ stated in Lemma 4.6 imply similar properties for $\varphi_{\mathcal{L}}$:

**Lemma 4.7** 1. *For all* $n$, $\varphi_{\mathcal{L}} n$ *is a correspondence sequence for* $p$ *and* $q$.

2. $\varphi_{\mathcal{L}}$ *is monotonic, i.e.* $\forall n_1 \leqslant n_2. \varphi_{\mathcal{L}} n_1 \subseteq \varphi_{\mathcal{L}} n_2$.

3. *For all* $n$, $n \in \pi_1(\varphi_{\mathcal{L}} n)$ *and* $n \in \pi_2(\varphi_{\mathcal{L}} n)$.

**Proof** 1. Follows with Lemma 4.6.1.

2. First notice that $\varphi_{\mathcal{L}} n \subseteq \varphi_{\mathcal{L}}(Sn)$ holds by Lemma 4.6.2. The claim then follows easily by induction on $n_2$.

3. Follows with Lemma 4.6.3. □

Due to the above results, one can think about $\varphi_{\mathcal{L}}$ as the listing of our intended isomorphism, that eventually contains all natural numbers in both projections. Therefore, we can define the isomorphism by computing for a given $n$ its corresponding element in $\pi_2(\varphi_{\mathcal{L}} n)$ using the witness operator[3] as already shown in Lemma 2.9.3.

**Lemma 4.8** *There is a function* $\varphi : \mathbb{N} \to \mathbb{N}$, *such that* $(n, \varphi n) \in \varphi_{\mathcal{L}} n$ *for all* $n$.

**Proof** Follows with Lemma 2.9.3 and Lemma 4.7.3. □

To close Myhill's isomorphism theorem, we have to prove the constructed function to be indeed an isomorphism between $p$ and $q$.

**Lemma 4.9** *The function* $\varphi$ *is*

1. *injective,*
2. *surjective, and*
3. $pn \leftrightarrow q(\varphi n)$ *for all* $n$.

**Proof** 1. Assume $n_1 \leqslant n_2$ (the case $n_2 \leqslant n_1$ is symmetric) and let $\varphi n_1 = \varphi n_2$. Then $(n_1, \varphi n_1) \in \varphi_{\mathcal{L}} n_1 \subseteq \varphi_{\mathcal{L}} n_2$ and $(n_2, \varphi n_2) \in \varphi_{\mathcal{L}} n_2$ by Lemma 4.7. Since $\varphi_{\mathcal{L}} n_2$ is a correspondence sequence we obtain $n_1 = n_2$ by Lemma 4.2.

[3]for decidable predicates over natural numbers

2. For $y : \mathbb{N}$, we have $(n, y) \in \varphi_{\mathcal{L}} y$ for some $n$ by Lemma 4.7.3. Then $\varphi n = y$: Similar to 1., assume $n \leqslant y$ (the other case is again symmetric) such that $(n, \varphi n), (n, y) \in \varphi_{\mathcal{L}} y$ and therefore $\varphi n = y$ by Lemma 4.2.

3. Follows since $(n, \varphi n)$ is in the correspondence sequence $\varphi_{\mathcal{L}} n$ for $p$ and $q$. □

We successfully defined and verified an isomorphism between two predicates over natural numbers with opposite one-one reductions:

**Corollary 4.10** *For all predicates* $p$ *and* $q$ *over* $\mathbb{N}$,

$$p \equiv_1 q \to p \equiv q.$$

**Proof** Via the constructed isomorphism $\varphi$ and its specification in Lemma 4.9. □

With the crucial direction of Myhill's isomorphism theorem proven, we conclude the theorem itself. We improve the result even further, by considering not only predicates over natural numbers but over general discrete types isomorphic to $\mathbb{N}$.

**Theorem 4.11 (Myhill's Isomorphism Theorem)** *Let* $X$ *and* $Y$ *be discrete types with* $\mathbb{N} \cong X$ *and* $\mathbb{N} \cong Y$. *For all predicates* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$,

$$p \equiv_1 q \leftrightarrow p \equiv q.$$

**Proof** Let $\mathbb{N} \cong X$ via the bijection $f_X$ and $\mathbb{N} \cong Y$ via $f_Y$. The forward direction follows with Corollary 4.10 for the "isomorphic"[4] natural number predicates $\lambda n.p(f_X n)$ and $\lambda n.q(f_Y n)$. The backward direction from Lemma 3.12.2, since X and Y are datatypes. □

As mentioned above, it was with regard to the Cantor-Bernstein-Schröder theorem that was shown by Pradic and Brown [29] to be equivalent to excluded middle not clear that we can prove Myhill's isomorphism theorem constructively axiom-free. Even though the two theorems remind to each other, neither of them is a special case of the other. On the one hand, only Myhill's isomorphism theorem talks about computable functions, on the other hand the functions in Cantor-Bernstein-Schröder's theorem operate not over the full underlying type, but only on the elements satisfying the considered predicates.

[4]This point requires the type restriction. See also chapter 7 for further details about this topic.

## 4.2 Cylinders & Many-One Reductions as One-One Reductions

We continue with discussing also the relation between one-one and many-one reducibility in more detail. It turns out, that the product predicate of predicates $p : X \to \mathbb{P}$ and $q : Y \to \mathbb{P}$ defined over the product $X \times Y$ of the underlying types plays an essential role for this analysis. Formally we define the product type as

$$p \times q := \lambda(x, y).\ px \land qy.$$

Notice at first the following basic properties of products concerning reductions:

**Lemma 4.12** *For all predicates* $p : X \to \mathbb{P}$*,*

1. $p \preceq_1 p \times X$.
2. $p \times Y \preceq_m p$ *for all types* $Y$.
3. $p \equiv_m p \times X$.
4. $q \preceq_1 r \to p \times q \preceq_1 p \times r$ *for all predicates* $q : Y \to \mathbb{P}$*, and* $r : Z \to \mathbb{P}$*.*

**Proof**
1. Via $\lambda x.(x, x)$.
2. Via $\lambda(x, y).x$.
3. By 1. and 2.
4. Let $q \preceq_1 r$ via $f$, then $p \times q \preceq_1 p \times r$ via $\lambda(x, y).(x, fy)$. □

Product predicates allow us to define the notion of so-called cylinder predicates. A predicate $p : X \to \mathbb{P}$ belongs to this class of cylinder predicates, if one can find a further predicate $q$, such that $p$ and the product $q \times X$[5] have the same 1-degree.

**Definition 4.13 (Cylinder)** *A predicate* $p : X \to \mathbb{P}$ *is called* ***cylinder****, if there exists a predicate* $q : X \to \mathbb{P}$ *with* $p \equiv_1 q \times X$*.*

The chosen definition of cylinders follows Rogers' traditional presentations of computability theory [31]. One can however show interesting characterizations of those cylinders, that would have allowed also differing definitions. The first one says, that a predicate $p$ is a cylinder if and only if all many-one reductions to $p$ can be also expressed as an one-one reduction.

**Lemma 4.14** *For types* $X$ *with* $X \times X \hookrightarrow X$ *and predicates* $p : X \to \mathbb{P}$*,*

$$\mathsf{cylinder}\ p \leftrightarrow \forall Y.(Y \hookrightarrow X) \to \forall r : Y \to \mathbb{P}.r \preceq_m p \to r \preceq_1 p.$$

[5] Recall that we identify $X$ with $\lambda x : X.\top$.

**Proof** Let $X \times X \hookrightarrow X$.

- $\rightarrow$: Let $p \equiv_1 q \times X$ for some $q$, $[\cdot] : Y \hookrightarrow X$ an injective encoding, and $r : Y \rightarrow \mathbb{P}$ with $r \preceq_m p$. Then we have with Lemma 4.12,

  $$r \preceq_m p \preceq_m q \times X \preceq_m q \quad \text{via some function } f.$$

  Therefore, $r \preceq_1 q \times X$ via the injection $\lambda x.(fx, [x])$ and therefore with $p \equiv_1 q \times X$ finally $r \preceq_1 q \times X \preceq_1 p$.

- $\leftarrow$: Assume $\forall Y.(Y \hookrightarrow X) \rightarrow \forall r : Y \rightarrow \mathbb{P}.r \preceq_m p \rightarrow r \preceq_1 p$. Since $X \times X \hookrightarrow X$ and $p \times X \preceq_m p$ by Lemma 4.12, we obtain $p \times X \preceq_1 p$. Therefore, $p \equiv_1 p \times X$ with Lemma 4.12 such that $p$ is a cylinder. □

Together with the basic properties of product predicates concerning reductions, the above characterization allows us to conclude further equivalent properties of cylinder predicates:

**Lemma 4.15** *For types $X$ with $X \times X \hookrightarrow X$ and predicates $p : X \rightarrow \mathbb{P}$, the following statements are equivalent.*

1. *$p$ is a cylinder.*
2. $p \equiv_1 p \times X$.
3. $p \preceq_1 p \times X$.

**Proof** 2. $\rightarrow$ 1. is straightforward and 2. $\leftrightarrow$ 3. follows with Lemma 4.12 such that it suffices to show 1. $\rightarrow$ 3.: Assume $p$ to be a cylinder. We apply the characterization from Lemma 4.14 with $Y := X \times X$ and $r := p \times X$ such that $p \preceq_m p \times X$ suffices to show $p \preceq_1 p \times X$. This follows again with Lemma 4.12. □

The equivalence in the above lemma implies all predicates of the form $p \times X$ (with $X \times X \hookrightarrow X$) to be a cylinder. Therefore, one calls $p \times X$ also the "cylindrification" of $p$. By combining this fact with the proven cylinder characterizations, one can express many-one reductions as one-one reductions on those cylinder predicates over types with certain embedding properties.

We will however construct the aspired reductions explicitly, which shows a more general result assuming weaker properties of the underlying types.

**Theorem 4.16 ($\preceq_m$ in terms of $\preceq_1$)**
*For types $X$ and $Y$ with $X \times X \hookrightarrow Y$[6] and predicates $p : X \rightarrow \mathbb{P}$ and $q : Y \rightarrow \mathbb{P}$.:*

$$p \preceq_m q \leftrightarrow p \times X \preceq_1 q \times Y.$$

[6]Proving the result using cylinders requires the additional assumption $Y \times Y \hookrightarrow Y$.

**Proof** Let $\langle \cdot, \cdot \rangle : X \times X \hookrightarrow Y$ an injective encoding.

$\rightarrow$: Assume $p \preceq_m q$ via $f$. Then $p \times X \preceq_1 q \times Y$ via $g := \lambda(x_1, x_2).\ (fx_1, \langle x_1, x_2 \rangle)$:

- $g$ is injective: $g(x_1, x_2) = g(x_1', x_2')$ implies $\langle x_1, x_2 \rangle = \langle x_1', x_2' \rangle$ which implies $(x_1, x_2) = (x_1', x_2')$ by the injectivity of $\langle \cdot, \cdot \rangle$.
- By the reduction property of $f$ we have furthermore:

$$(p \times X)(x_1, x_2) \leftrightarrow px_1 \leftrightarrow q(fx_1) \leftrightarrow (q \times Y)(g(x_1, x_2)).$$

$\leftarrow$: Assume $p \times X \preceq_1 q \times Y$ via $f$. Then $p \preceq_m q$ via $g := \lambda x. \pi_1(f(x, x))$:

$$px \leftrightarrow (p \times X)(x, x) \leftrightarrow (q \times Y)(f(x, x)) \leftrightarrow q(gx). \qquad \square$$

Stating the reduction functions explicitly allows us also to observe more clearly the reason, why we can express many-one reductions in terms of one-one reductions: We decode $(x_1, x_2)$ as an $Y$ value by using the type assumption $X \times X \hookrightarrow Y$, which ensures the injectivity of the reduction. This traditionally only implicitly used (e.g. for $X = Y = \mathbb{N}$ or isomorphic types) but essential property of the underlying type could be precisely detected by the type theoretical analysis.

The characterization of many-one reductions yields also further insights in the structure of reduction degrees: Every $m$-degree, consisting out of (potentially) multiple 1-degrees contains a maximum 1-degree, which is obtained by "cylindrification", i.e. building the product with the underlying type. In addition the characterization implies the structure of the preorder $\preceq_m$ to be reflected in the structure of $\preceq_1$ by their corresponding cylinder predicates.

We will come back to the notion of cylinder predicates in chapter 6, where we show that cylinders can be used to distinguish one-one and many-one reducibility.

## 4.3 Truth-Table Reductions as One-One Reductions

We go one step further by looking for a characterization of truth-table reductions. Similar to the "cylindrification" of a predicate, we can again find an adaption of predicates that allows to express truth-table reduction as one-one reductions.

This adaption works by so-called truth-table cylinders. The truth-table cylinder of a predicate $p : X \to \mathbb{P}$ expects as arguments a query list $L_X : \mathcal{L}(X)$ and a condition $\alpha : \mathbb{B}^{|L_X|} \to \mathbb{B}$ that fits to the (length of the) query list. Since the type of the condition is therefore dependent on the query list, truth-table cylinders work over the dependent type $\Sigma\, L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}$. The truth-table cylinder denoted as $p^{tt}$ of $p$ holds now for those $(L_X, \alpha)$, where $\alpha$ is a correct "truth-table condition" for $L_X$:

$$p^{tt} : (\Sigma\, L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}) \to \mathbb{P}$$
$$p^{tt} := \lambda(L_X, \alpha : \mathbb{B}^{|L_X|} \to \mathbb{B}).\ \forall L.\ L_X \mathrel{\widehat{=}_p} L \to \alpha L = \mathsf{true}.$$

The definition is again well-typed, since $L_X \mathrel{\widehat{=}_p} L$ implies $|L| = |L_X|$. We start to prove some basic facts of those truth-table cylinders as well as an auxiliary result regarding corresponding list:

**Lemma 4.17** *For all predicates* $p : X \to \mathbb{P}$*,*

1. *if* $p$ *is stable, then* $p \preceq_1 p^{tt}$*.*
2. $p^{tt} \preceq_{tt} p$*.*
3. $p^{tt}$ *is stable.*
4. *for all* $L_X : \mathcal{L}(X)$*, there weakly exists* $L$[7] *with* $L_X \mathrel{\widehat{=}_p} L$*.*

**Proof** 1. Via $\lambda x.\,([x], \lambda[b].b)$. The first component $[x]$ ensures the injectivity of the reduction, showing the backward direction of the reduction property requires then the stability of $p$.

2. Via $\lambda(L_X, \alpha).\, L_X$ and $\lambda(L_X, \alpha)L.\, \alpha L$. The truth-table property of this reduction follows directly by the analog truth-table cylinder property of $p^{tt}$

3. For $\neg\neg p^{tt}(L_X, \alpha)$, we have to show for all $L$, that $L_X \mathrel{\widehat{=}_p} L$ implies $\alpha L = \mathsf{true}$. But for $\alpha L = \mathsf{false}$, we get $\neg p^{tt}(L_X, \alpha)$, a contradiction.

4. By induction on $L$. With $\bot$ to show, we can decide $px$ in the step case $(x :: L)$. □

Note the influence of the constructive setting to the above as well as to the following results: We have to care about the stability of predicates and can in contrast to traditional presentations only show the week existence of corresponding lists.

In order to prove, that truth-table cylinders indeed help to express truth-table in terms of one-one reductions, we first show an auxiliary result. Under certain assumptions regarding the underlying predicate types, we can show that every truth-table reduction can be transformed into an injective reduction to the truth-table cylinder of the targeted predicate, i.e.

**Lemma 4.18** *Let* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$ *with* $X \hookrightarrow Y$*. If* $p$ *is stable, then*

$$p \preceq_{tt} q \to p \preceq_1 q^{tt}.$$

**Proof** Let $[\cdot] : X \hookrightarrow Y$ be an injective encoding, $p$ be stable, and $p \preceq_{tt} q$ via $f$ and $\alpha : \forall x.\, \mathbb{B}^{|fx|} \to \mathbb{B}$, i.e. $\mathsf{TT} : \forall x L.\, (fx) \mathrel{\widehat{=}_q} L \to px \leftrightarrow \alpha x L$. Then the function

$$g : X \to \Sigma\, L_Y : \mathcal{L}(Y).\, \mathbb{B}^{|L_Y|} \to \mathbb{B}$$
$$gx := ([x] :: (fx), \lambda(b :: L).\alpha x L)$$

is a one-one reduction establishing $p \preceq_1 q^{tt}$:

[7]Recall that weak existence denotes the double negated existence of – in this case – L.

- First of all, $g$ is well typed: The query list component of $gx$ has length $1 + |fx|$. Therefore, the condition component is of type $\mathbb{B}^{1+|fx|} \to \mathbb{B}$, such that we can match on its argument $b :: L$ with $|L| = |(fx)|$. Hence, $\alpha x$ can be applied to $L$.
- $g$ is injective by the injectivity of $[\cdot]$:

$$gx_1 = gx_2 \to [x_1] :: (fx_1) = [x_2] :: (fx_2) \to [x_1] = [x_2] \to x_1 = x_2$$

- $px \to q^{tt}(gx)$: Assume $px$ and $([x] :: (fx)) \mathrel{\widehat{=}_q} (b :: L)$ and therefore also $(fx) \mathrel{\widehat{=}_q} L$. Now $(TTxL)$ implies $\alpha xL = \mathsf{true}$.
- $q^{tt}(gx) \to px$: The stability of $p$ allows to obtain a list $(b :: L)$ with $([x] :: (fx)) \mathrel{\widehat{=}_q} (b :: L)$ by Lemma 4.17 [8]. Therefore also again $(fx) \mathrel{\widehat{=}_q} L$, such that assuming $q^{tt}(gx)$ implies $\alpha xL = \mathsf{true}$ and with $(TTxL)$ finally $px$. □

We want to use this result now with $p^{tt}$ in the place of $p$. Since we already showed, that $p \preceq_{tt} q$ implies by transitivity $p^{tt} \preceq_{tt} q$, we can express truth-table reduction in terms of one-one reductions. However, the above Lemma requires the important assumption concerning the embedding of the underlying types, such that using the Lemma as intended requires an embedding of the depended type of $p^{tt}$ into the target type. We first prove the result with this particular assumption:

**Lemma 4.19** *Assume types* $X$ *and* $Y$ *with* $(\Sigma L_X.\ \mathbb{B}^{|L_X|} \to \mathbb{B}) \hookrightarrow Y$. *For all predicates* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$ *with stable* $p$,

$$p \preceq_{tt} q \leftrightarrow p^{tt} \preceq_1 q^{tt}.$$

**Proof** Let $X$ and $Y$ be as assumed and $p$ be stable.

$\to$: For $p \preceq_{tt} q$, Lemma 4.17 yields also $p^{tt} \preceq_{tt} p \preceq_{tt} q$. Since we assumed $(\Sigma L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}) \hookrightarrow Y$ and know $p^{tt}$ to be stable by Lemma 4.17, we can apply Lemma 4.18 to the reduction $p^{tt} \preceq_{tt} q$ and obtain $p^{tt} \preceq_1 q^{tt}$.

$\leftarrow$: For $p^{tt} \preceq_1 q^{tt}$, we derive

| | |
|---|---|
| $p \preceq_1 p^{tt}$ | Lemma 4.17.1 and the stability of $p$ |
| $\preceq_1 q^{tt}$ | assumption |
| $\preceq_{tt} q$ | Lemma 4.17.2 |

and therefore $p \preceq_{tt} q$. □

[8]without stability only yielding the weak existence of such a list

It remains to show, that assuming $\Sigma\, L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}$ to be embeddable into $Y$ can be justified for at least some types $X$ and $Y$, such that the above result is not worthless. Especially the encoding of the second component of the dependent type, namely a function seems to be problematic.

Here comes however into play, that the function type of the truth-table condition was carefully chosen: In contrast to the more general but uncountable function type $\mathcal{L}(\mathbb{B}) \to \mathbb{B}$, the type $\mathbb{B}^n \to \mathbb{B}$ is not only countable but finite for every $n$: Given a fixed number of boolean values, there are only finitely many possibilities to combine them to one boolean result. There are especially exactly $2^n$ possible inputs for this function, such that the behavior of the function is uniquely determined by its computation on those finite inputs.

This argumentation clearly uses the (classical) principle of functional extensionality: If functions agree on all arguments, they are already equal and therefore especially in the sense of the counting intuition above the same element. Therefore, we have to assume for the encoding of the dependent type the constructively not provable functional extensionality for the condition function:

$$\mathsf{FE}_\alpha{}^{9} := \forall n.\ \forall \alpha_1 \alpha_2.\ (\forall L : \mathbb{B}^n.\ \alpha_1 L = \alpha_2 L) \to \alpha_1 = \alpha_2.$$

Under this axiom together with some basic embedding assumptions of the underlying types $X$ and $Y$, we can show the embedding of $\Sigma\, L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}$ into $Y$:

**Lemma 4.20** *Assume* $\mathsf{FE}_\alpha$ *and types* $X$ *and* $Y$ *with*

$$(X \hookrightarrow Y) \textit{ and } (\mathbb{N} \hookrightarrow Y) \textit{ and } (Y \times Y \hookrightarrow Y).$$

*Then,* $(\Sigma\, L_X : \mathcal{L}(X).\ \mathbb{B}^{|L_X|} \to \mathbb{B}) \hookrightarrow Y$ *and therefore for all predicates* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$ *with stable* $p$*:*

$$p \preceq_{tt} q \leftrightarrow p^{tt} \preceq_1 q^{tt}.$$

**Proof** By $Y \times Y \hookrightarrow Y$ it suffices to show both $\mathcal{L}(X) \hookrightarrow Y$ (1.) and $(\mathbb{B}^n \to \mathbb{B}) \hookrightarrow Y$ (2.). So let $[\cdot]_X :\ X \hookrightarrow Y$, $[\cdot]_\mathbb{N} :\ \mathbb{N} \hookrightarrow Y$, and $\langle \cdot, \cdot \rangle :\ Y \times Y \hookrightarrow Y$ be encodings.

1. We encode the list $[x_1, \ldots, x_n]$ injectively into $Y$ as
$$\langle [n]_\mathbb{N}, \langle [x_1]_X, \langle \ldots \langle [x_{n-1}]_X, [x_n]_X \rangle \ldots \rangle\rangle\rangle.$$

2. Given an element $\alpha : \mathbb{B}^n \hookrightarrow \mathbb{B}$, we notice that the type $\mathbb{B}^n$ has exactly $2^n$ elements, such that we can list all possible inputs of $\alpha$ in a list $L$. Under $\mathsf{FE}_\alpha$, the function $\alpha$ is therefore uniquely encoded as $(\mathsf{map}\ \alpha\ L)$. This boolean list can then be encoded into $Y$ similarly to the encoding in 1.[10] $\square$

[9]The principle is derivable out of the general formulation of functional extensionality: $\mathsf{FE} := \forall X(T : X \to \mathbb{T}).\forall(f_1 f_2 : \forall x.Tx).(\forall x.f_1 x = f_2 x) \to f_1 = f_2$.

[10]Replace $[\cdot]$ with $[\cdot]_\mathbb{N}$ and interpret the boolean values as $0$ and $1$.

Notice that the assumed embeddings into $Y$ hold again for the types computability theory mainly works with like $\mathbb{N}$ or other isomorphic types.

**Corollary 4.21** *For* $p : \mathbb{N} \to \mathbb{P}$ *and* $q : \mathbb{N} \to \mathbb{P}$ *with stable* $p$,

$$p \preceq_{tt} q \leftrightarrow p^{tt} \preceq_1 q^{tt}.$$

Therefore and similar to cylinder predicates (for $\preceq_m$), we showed that the structure of now $\preceq_{tt}$ is reflected in the structure of $\preceq_1$ and that every $tt$-degree contains a maximal 1-degree, both by the truth-table cylinders of the particular predicates.

Furthermore, an easy adaption of the above proof shows that truth-table cylinders can express truth-table reductions in the exact same way also in terms of many-one reductions. Since many-one reductions are no longer required to be injective, the characterization in terms of many-one reducibility works for general predicate types without assuming the above embedding properties.

Lastly we mention, that there seems a possibility to improve the result, namely a formalization that allows to drop the assumption $\mathsf{FE}_\alpha$. Instead of defining truth-table conditions as function of type $\mathbb{B}^n \to \mathbb{B}$, one could encode the computed results of $\alpha$ for each of the fixed number of $2^n$ possible inputs into a list of this length. We discussed that $\alpha$ is therewith uniquely specified. The condition would then be a boolean list with fixed length, i.e.

$$\alpha : \mathbb{B}^{(2^n)}.$$

We saw in the last Lemma how to encode lists with fixed length, such that functional extensionality would be no longer required. However, it would become quite inconvenient to work with the then unavoidable decoding and encoding when working with truth-table cylinders and reductions. It seems possible that this refinement of our result could build up on existent work concerning finite sets by Frumin et al [19] or on the currently ongoing work from Amorim and Blanco [9], who develop a library for finite functions that supports extensional equality.

# Chapter 5

# Infinite Predicates

Many traditional presentations of mathematics and also computability theory do not focus on a formal description of infinite predicates. Instead, they use many different – often intuitive – ways to think about and for example prove a predicate to be infinite. We want to take a closer look at and formalize some of these different notions for infinite predicates and discuss their constructively provable connections. Interestingly, the notions differ a lot in their strength and properties and are constructively nowhere near to be equivalent. This chapter does not commit to one concrete definition of infinite predicates, but discusses different options to define infinite predicates.

## 5.1 Functional Infinite Predicates

We start by looking at two notions for a predicate $p$, that either require a surjection from $p$ into the natural numbers, or require an injection from the natural numbers into $p$. This was already defined and denoted in chapter 2 as:

$$p \twoheadrightarrow \mathbb{N} \quad \text{(surjection)} \qquad\qquad \mathbb{N} \hookrightarrow p \quad (\mathsf{injection})$$

Even though these notions are classically[1] seen to be equivalent by using the right or left inverse function respectively, it seems for us to be in general not possible to construct an injection $\mathbb{N} \hookrightarrow p$ out of a surjection $p \twoheadrightarrow \mathbb{N}$ or vice versa. We have to be satisfied with the following result:

**Lemma 5.1** *Let $p : X \to \mathbb{P}$ be enumerable. If $p \twoheadrightarrow \mathbb{N}$, then $\mathbb{N} \hookrightarrow p$.*

**Proof** Similar to the construction of the inverse function (Lemma 2.6). Assume $\mathcal{E}p$ and $f : p \twoheadrightarrow \mathbb{N}$, i.e. $\forall n.\exists x.px \wedge fx = n$. The witness operator for enumerable predicates $\omega_{\mathcal{E}}$ yields an inverse function $f^{-1}$ with $p(f^{-1}n)$ and $f(f^{-1}n) = n$. Therefore, $f^{-1} : \mathbb{N} \hookrightarrow p$. □

[1] In some cases under the use of a choice axiom.

It turns out, that these formalizations of infinite predicates are one of the strongest one could choose. Both notions imply a weaker way to think about infinite predicates, namely predicates containing elements of any number.

## 5.2 Predicates Containing Elements of Any Number

We formalize this property by requiring the predicate to have arbitrarily large, finite subsets, i.e. the existence of arbitrary long, duplicate-free lists, that are subsets of the predicate:

**Definition 5.2** *We say that a predicate* $p : X \to \mathbb{P}$ ***contains elements of any number****, if*

$$\forall n : \mathbb{N}.\ \exists L : \mathcal{L}(X).\ |L| = n \land \#L \land L \subseteq p.$$

**Lemma 5.3** *For* $p : X \to \mathbb{P}$*,*

1. *if* $p \twoheadrightarrow \mathbb{N}$*, then* $p$ *contains elements of any number.*
2. *if* $\mathbb{N} \hookrightarrow p$*, then* $p$ *contains elements of any number.*

**Proof**
1. Assume $f : p \twoheadrightarrow \mathbb{N}$. We proceed by induction on $n$. The base case is straightforward by picking $[]$. In the step case, the inductive hypothesis yields a list $L$. By $f : p \twoheadrightarrow \mathbb{N}$, there exists an element $x$ with $px$ and $fx = \max(\mathsf{map}\ f\ L) + 1$. Therefore, $x \notin L$ and we are done by picking $x :: L$.
2. Assume $f : \mathbb{N} \hookrightarrow p$ and $n : \mathbb{N}$. The list $\mathsf{map}\ f\ [0, \ldots, n-1]$ has length $n$, is duplicate-free by Lemma 2.8, and is a subset of $p$, since $p(fn')$ holds for all $n'$.

□

One can find many equivalent formulations for predicates containing elements of any number. All of them can be understood as notions that require continuously the existence of further elements satisfying the predicate. We state a few equivalent formulations of this "infinite class" in the next lemma. Even though the implications are not hard to prove, this "toolbox" of notions becomes precious when working with concrete predicates. For many different predicates, it provides well matching and therefore well applicable criteria.

**Lemma 5.4** *Let* $p : X \to \mathbb{P}$ *be a predicate over a discrete type* $X$*. Then the following statements are equivalent:*

1. $p$ *contains elements of any number.*
2. $\forall n.\ \exists L : \mathcal{L}(X).\ |L| \geqslant n \land \#L \land L \subseteq p.$[2]
3. $\forall L : \mathcal{L}(X).\ \exists x.\ px \land x \notin L.$

*For* $X = \mathbb{N}$*, it is also equivalent:*

4. $\forall n.\ \exists x \geqslant n.\ px.$

[2]This formulation simply replaces "$\exists L.|L| = n$" with "$\exists L.|L| \geqslant n$" in Definition 5.2.

**Proof** Let $X$ be discrete and $p : X \to \mathbb{P}$.
1. $\to$ 2.: Straightforward.
2. $\to$ 3.: For $L : \mathcal{L}(X)$, the assumption for $n = |L| + 1$ provides a duplicate-free list $L_0$ with $|L_0| > |L|$ and $L_0 \subseteq p$. Therefore, the pigeonhole principle 2.10 yields $x \in L_0$ and hence $px$ but with $x \notin L$.
3. $\to$ 1.: Straightforward by induction on $n$.
3. $\leftrightarrow$ 4.: Use the assumption for $[0, \dots, n]$ and $\max L + 1$ respectively. □

Lemma 5.3 stated sufficient criteria for these equivalent properties of a predicate $p$. Showing implications in the other direction, for example showing the existence of a surjection from a predicate $p$ containing elements of any number into $\mathbb{N}$ is not possible in general. However, we can use a similar idea as in Lemma 5.1 to construct an injection from $\mathbb{N}$ into an enumerable predicate $p$.

**Lemma 5.5** *Let $X$ be discrete and $p : X \to \mathbb{P}$ enumerable. If $p$ contains elements of any number, then $\mathbb{N} \hookrightarrow p$.*

**Proof** Assume with Lemma 5.4 $\forall L.\ \exists x.\ px \land x \notin L$. We again use $\omega_{\mathcal{E}}$ that yields a function $f_{\mathcal{L}} : \mathcal{L}(X) \to X$ with $p(f_{\mathcal{L}} L)$ and $f_{\mathcal{L}} L \notin L$. We can use this function now recursively to obtain a function $f : \mathbb{N} \to \mathcal{L}(X)$ defined as

$$\begin{aligned} f0 &:= [\,] \\ f(Sn) &:= \mathsf{let}\ L := fn\ \mathsf{in}\ f_{\mathcal{L}} L :: L \end{aligned}$$

with the following two properties:

1. $\forall n_1 \leqslant n_2 . fn_1 \subseteq fn_2$.
2. $f(Sn)$ is not empty and for its first element $x_n := \mathsf{hd}\ (f(Sn))$, we have $px_n$ and $x_n \notin f_{\mathcal{L}} n$.

Therefore, $(\lambda n.x_n) : \mathbb{N} \hookrightarrow p$:
Let $x_{n_1} = x_{n_2}$. Then $p(x_{n_1})$ and $n_1 < n_2$ ($n_2 < n_1$ symmetrically) is contradictory, since $x_{n_2} = x_{n_1} \in f(Sn_1) \subseteq fn_2$ by 1. but $x_{n_2} \notin fn_2$ by 2. Hence, $n_1 = n_2$. □

## 5.3 Non-Finite Predicates

Related to predicates containing elements of any number are non-finite predicates. Instead of demanding the existence of arbitrarily large, finite subsets, we do not want the predicate to have a finite superset. While a finite superset – formalized again as a list – is allowed to contain also elements not satisfying the predicate $p$, an even stronger notion of finiteness demands the existence of a list containing exactly the elements from $p$.

**Definition 5.6 (Finite Predicates)**
*A predicate* $p : X \to \mathbb{P}$ *is called* ***finite****, if there exists a list* $L : \mathcal{L}(X)$ *with* $p \subseteq L$.
*If there exists a list* $L : \mathcal{L}(X)$ *with* $\forall x.px \leftrightarrow x \in L$, *then* $p$ *is called* ***strongly finite****.*

Although the both notions of finiteness are classically equivalent by simply removing elements in $\overline{p}$ from $L$, the equivalence holds constructively only for decidable predicates. Therefore, only non-finite predicates are also not strongly finite, the other direction does not hold in general.

The following Lemma states that non-finiteness is one of the weakest notions for infinite predicates and yields, combined with the equivalences in Lemma 5.4, many sufficient criteria to show non-finiteness of a predicate:

**Lemma 5.7** *Let* $X$ *be discrete and* $p : X \to \mathbb{P}$ *a predicate containing elements of any number. Then* $p$ *is not finite.*

**Proof** Towards a contradiction, assume $p$ to be finite via the list $L$. By Lemma 5.4.3, there exists $x \notin L$ with $px$. A contradiction, since $p \subseteq L$. □

The backward direction is again only provable using classical axioms, like the full law of De Morgan for existential and universal quantifiers.

As mentioned above, all the criteria in Lemma 5.4 can be useful to prove a predicate to be non-finite, however it comes with an unpleasant restriction. The Lemma requires us to show the existence of arbitrarily many elements in the predicate, which is as far as we conjecture only possible by computing arbitrarily many of those elements. Therefore, we would have to enumerate at least an infinite subset of the predicate. This makes the Lemma hard to apply for some forms of predicates, like for example not enumerable ones. By requiring not the existence but the weak existence of elements of any number, the criterion remains valid and turns out to be a characterization of non-finite predicates. Also all equivalences in Lemma 5.4 stay provable when double negating all existential quantifiers, e.g.

$$\text{"}\forall n.\neg\neg\exists L.|L| = n \land \#L \land L \subseteq p\text{"} \quad \text{instead of} \quad \text{"}\forall n.\exists L.|L| = n \land \#L \land L \subseteq p\text{"}.$$

We refer to these double negated statements as the weak existence version of a statement. This results in another class of equivalent notions:

**Lemma 5.8** *Let* $X$ *be discrete. Non-fininiteness of a predicate* $p : X \to \mathbb{P}$ *is equivalent to the weak existence versions of the statements in Lemma 5.4.*

**Proof** We only show

$$\neg\mathsf{finite}\ p \leftrightarrow \forall L.\neg\neg\exists x.px \land x \notin L,$$

the implications between the modified statements are proven similarly to Lemma 5.4 after eliminating the double negations on both sides.

So first assume $p$ to be non-finite and a list $L$ such that $\neg\exists x.px \wedge x \notin L$. But then, $p$ is finite via $L$. Otherwise assume $\forall L.\neg\neg\exists x.px \wedge x \notin L$ and $p$ to be finite via $L_0$. Then the assumption is violated for $L := L_0$. □

We saw many different ways to formalize a predicate being infinite. By assuming the law of excluded middle and the axiom of choice, all notions would become equivalent. While also in our axiom-free constructive setting some of the notions do coincide and build a class of equivalent notions, there are however also implications that cannot be proven constructively or only with further assumptions regarding the predicate. The following diagram summarizes provable implications between the different notions of infinite predicates $p : X \to \mathbb{P}$ over a discrete type $X$:

$$\begin{tikzcd}
p \twoheadrightarrow \mathbb{N} \arrow[rr, dashed] \arrow[dr] & & \mathbb{N} \hookrightarrow p \arrow[dl] \\
& \text{$p$ contains elements of any number} \arrow[ul, dashed, bend left] \arrow[ur, dashed, bend right] \arrow[d] & \\
& \text{$p$ is not finite} \arrow[d] & \\
& \text{$p$ is not strongly finite} \arrow[u, dotted, bend right] &
\end{tikzcd}$$

$\longrightarrow$ : for all $p$
$\dashrightarrow$ : for $\mathcal{E}p$
$\cdots\rightarrow$ : for $\mathcal{D}p$

## 5.4 Properties of Infinite Predicates

We want to discuss properties of infinite predicates with respect to the different notions. We will observe further differences between the notions, for example regarding their closure properties. Again, axiom-free constructive logic cannot prove for every notion all the intuitively valid and in traditional presentations used properties of infinite predicates.

Under all notions, there exists an infinite predicate. For example the constant predicate $\lambda n : \mathbb{N}.\top$ is infinite for all notions of infinity, since it is canonical bijective to $\mathbb{N}$ via $\lambda n.n$. Similarly, this holds for many other basic predicates over natural numbers like even or odd numbers. Furthermore, one clearly expects the existence of elements satisfying an infinite predicate. This holds also for most of the notions, however and analogously to the characterizations of non-finiteness, we can show only the weak existence of an element in non (strongly) finite predicates.

**Lemma 5.9 (Non-Emptiness)**

1. *If* $p$ *contains elements of any number, there exists* $x$ *with* $px$.
2. *If* $p$ *is not (strongly) finite, there weakly exists* $x$ *with* $px$.

**Proof** 1. Use the assumption for $n = 1$ and pick the head of the list.

2. We show the positive contraposition $\neg\exists x.px \rightarrow (\text{strongly})\ \text{finite}\ p$. So assume $p$ with $\neg\exists x.px$. Then $[]$ contains exactly the elements satisfying $p$. □

Also looking closer at closure properties of infinite predicates reveals some interesting as well as useful results. However, we again have to accept some limitations, as not all closures are provable for all the notions of infinite predicates. Even closure under supersets (i.e. $q \supseteq p$ is infinite if $p$ is infinite) is not provable for not strongly finite predicates in general and works only for decidable predicates $p$. All other notions are actually easy to show closed under supersets. It becomes even more difficult when considering closures under surjections and injections between two predicates. We are again facing the problem of not being able to construct right or left inverse functions between predicates $p$ and $q$, which would be necessary to transport some notions of infinity through $p \twoheadrightarrow q$ or $p \hookrightarrow q$. We state provable closures in the following two lemmas:

**Lemma 5.10 (Closure under Surjections)** *Let* $p : X \rightarrow \mathbb{P}$, $q : Y \rightarrow \mathbb{P}$, *and* $p \twoheadrightarrow q$.

1. *If* $q \twoheadrightarrow \mathbb{N}$, *then also* $p \twoheadrightarrow \mathbb{N}$.
2. *If* $q$ *is not finite, also* $p$ *is not finite.*

**Proof** Assume $f : p \twoheadrightarrow q$.

1. Let $f_q : q \twoheadrightarrow \mathbb{N}$, then $(\lambda x.\ f_q(fx)) : p \twoheadrightarrow \mathbb{N}$.
2. We show the contraposition $\text{finite}\ p \rightarrow \text{finite}\ q$. If $p \subseteq L$, then $q \subseteq \text{map}\ f\ L$: $qy$ implies $fx = y$ for some $x$ with $px$ and hence $x \in L$. Thus, $y = fx \in (\text{map}\ f\ L)$. □

**Lemma 5.11 (Closure under Injections)** *Let* $p : X \rightarrow \mathbb{P}$, $q : Y \rightarrow \mathbb{P}$, *and* $p \hookrightarrow q$.

1. *If* $\mathbb{N} \hookrightarrow p$, *then also* $\mathbb{N} \hookrightarrow q$.
2. *If* $p$ *contains elements of any number, also* $q$ *contains elements of any number.*

*Let furthermore* $X$ *and* $Y$ *be discrete, then:*

3. *If* $p$ *is not finite, also* $q$ *is not finite.*

**Proof** Assume $f : p \hookrightarrow q$.

1. Let $f_p : \mathbb{N} \hookrightarrow p$, then $(\lambda n.\ f(f_p x)) : p \hookrightarrow \mathbb{N}$.
2. Assume for all $n$ the existence of a duplicate-free list $L_n \subseteq p$ with $|L_n| = n$. Then $|\mathsf{map}\ f\ L_n| = |L_n| = n$, $(\mathsf{map}\ f\ L_n)$ is duplicate-free, and $x \in (\mathsf{map}\ f\ L_n)$ implies $x \in L$ (Lemma 2.8 and the injectivity of $f$ ) implies $px$ .
3. We use lemma 5.8 on both sides to show

$$\big(\forall n.\neg\neg\exists L.|L| = n \land \#L \land L \subseteq p\big) \longrightarrow \forall n.\neg\neg\exists L.|L| = n \land \#L \land \forall y \in L.qy,$$

which follows similar to claim 2. □

Like in the discussion of implications between different notions of infinite predicates, we also focused in our presentation of their properties on provable results in an axiom-free constructive setting. However, it could be also very interesting to look closer at properties of infinite predicates under various further assumptions like choice axioms or in classical logic as well as to discuss reversely, whether some – constructively not provable – implications between infinite notions imply for instance already a weak choice axiom.

The different notions for infinite predicates, their different properties and especially their different non-provable properties show, that there is neither a general way to think constructively about a predicate being infinite nor one correct definition of infinite predicates. Instead, it depends a lot on the actually used properties of infinite predicates and the concrete predicates that one likes to prove being infinite predicates. In contrast to traditional presentations, we cannot throw all notions in one bucket and think about all of them as the same mathematical description of infinity. It is not possible to readily use different notions in the same proof, since they are not necessarily equivalent. Instead, we have to study very carefully the for a particular development necessary and essentially used properties. The definition of infinite predicates has to follow this careful analysis in order to be the best working notion for the particularly aspired constructive formalization.

# Chapter 6

# Simple Predicates, Post's Problem & Distinctions

In chapter 4, we characterized different reductions and their reduction degrees. Isomorphic predicates have by Myhill's isomorphism theorem also the same 1-degree, predicates with the same 1-degree have clearly also the same $\mathrm{m}$-degree. We found connections between one-one, many-one, and truth-table reductions. However, it is not clear whether those different reducibility terms and therefore their computability degree do coincide on undecidable predicates, or whether we can find a distinction between these notions. Another question appears when analyzing the inner structure of the computability degrees on semidecidable predicates. By definition, the class of $\mathrm{m}$-complete (1-complete, $\mathrm{tt}$-complete) predicates has the maximal $\mathrm{m}$-degree (1-degree, $\mathrm{tt}$-degree) on semidecidable predicates. But can we distinguish different classes of semidecidable but undecidable predicates by their degree or are all those predicates already complete? Since the halting problem is 1-complete, one could equivalently ask, whether there is an undecidable but semidecidable predicate $p$ with $\mathcal{W} \not\preceq p$. Emil Post raised this questions in 1944 with respect to Turing reducibility [28], which became known as Post's problem asking:

"Is there an semidecidable but undecidable predicate, such that the halting problem does not Turing reduce to this predicate?"

This question could be answered positively independently by Muchnik [24] and Friedberg [17] in the 1950's by the so-called priority method. Post himself could solve the problem with respect to many-one reducibility by inventing so-called simple predicates[1]. Interestingly this class of simple predicates also yields a distinction of one-one, many-one, and truth-table reducibility on semidecidable but undecidable predicates.

Post used a property, that the complement of every $\mathrm{m}$-complete predicate contains a semidecidable and infinite subset. Therefore, Post defined a predicate to be simple if it is semidecidable and its complement contains no semidecidable and infinite

[1] He solved the problem also for truth-table reductions via hyper-simple predicates; see chapter 8.3.

subset even though the complement is itself infinite[2]. Furthermore, Post showed the existence of a simple predicates by constructing such a predicate out of the halting problem.

We want to formalize the notion of simple predicates, the construction of a simple predicate, and show the characteristic properties of those predicates in order to conclude results regarding reducibility degrees. The traditional presentations use essentially properties of their underlying model of computation like a universal machine. As discussed in chapter 3, we can follow this presentations by using carefully chosen axioms of synthetic computability theory. Furthermore, we will have to work with infinite predicates, which becomes as already indicated in the last chapter a crucial and interesting aspect right from the start. We discussed that some notions of infinite predicates require us to compute and therefore enumerate an infinite subset of the predicate when showing it infiniteness. However, the complement of a simple predicate should be provable to be infinite even though it contains by definition no semidecidable – thus enumerable – and infinite subset. This leads to the following, on the first view paradox result, stating that the existence of a simple predicate is contradictory for some notions of infinite predicates:

**Lemma 6.1** *There exists no predicate* $p$ *over a discrete type, with the following properties:*

1. $\mathcal{S}p$,
2. $\overline{p}$ *contains no semidecidable subset* $q$ *with* $\mathbb{N} \hookrightarrow q$, *and*
3. $\mathbb{N} \hookrightarrow \overline{p}$.

**Proof** Assume such a predicate $p$ with $f : \mathbb{N} \hookrightarrow \overline{p}$ and consider the predicate $q := \lambda x.\ \exists n. fn = x$, i.e. the range of $f$. Therefore, $q$ is semidecidable and since $f : \mathbb{N} \hookrightarrow \overline{p}$, we have furthermore $q \subseteq \overline{p}$ and $f : \mathbb{N} \hookrightarrow q$, which contradicts 2. □

This supposed contradiction to the existence of a simple predicate is due to our synthetic approach: We identify the notion of a function with the notion of a computable function, such that $f : \mathbb{N} \hookrightarrow p$ is analogously to for instance deciders or reductions immediately computable. Only therefore, the range of $f$ is semidecidable as the range of a computable function.

This observation forces us to choose a definition of infinite predicates, that requires no computation of elements in the predicate. We can preclude many infinite notions introduced in Chapter 5 to be the correct one for our purpose to formalize and show the existence of a simple predicate. Finally by the lack of provable closures for not strongly finite predicates, it turns out that the for our particular problem matching definition for an infinite predicate is non-finiteness.

[2]Finite complements contain no infinite subset, however those predicates are at least classically clearly decidable.

**Definition 6.2 (Infinite Predicates)**
*We call a predicate* $p : X \to \mathbb{P}$ *from now on **infinite**, if it is not finite, i.e.*

$$\neg \exists L. p \subseteq L.$$

$p$ *is called **co-infinite**, if* $\overline{p}$ *is infinite.*

We use this definition to formally define simple predicates as follows:

**Definition 6.3 (Simple Predicates)** *A predicate* $p : X \to \mathbb{P}$ *is called **simple**, if*

$$\mathcal{S}p \wedge (\forall q. \mathcal{S}q \to \mathsf{infinite}\ q \to q \not\subseteq \overline{p}) \wedge \mathsf{infinite}\ \overline{p}.$$

Besides the choice of the correct infinity definition, notice a second delicately balanced decision in formalizing the notion of simple predicates. Classically, $q \not\subseteq \overline{p}$ is equivalent to $\exists x. qx \wedge px$; constructively, this is however again not the case and requiring $q \not\subseteq \overline{p}$ is the strict weaker notion. Since the hardest part of the following work with simple predicates turns out to be the existence proof of such a predicate, we chose the easier provable notion $q \not\subseteq \overline{p}$.

The first basic result regarding simple predicates is easy to see and states that the class of simple predicates is indeed a subclass of undecidable predicates.

**Lemma 6.4** *Simple predicates are undecidable.*

**Proof** Let $p$ be simple and decidable. Therefore, $\overline{p}$ is semidecidable and infinite by assumption, and $\overline{p} \subseteq \overline{p}$. A contradiction to $\forall q. \mathcal{S}q \to \mathsf{infinite}\ q \to q \not\subseteq \overline{p}$. □

## 6.1 Construction of a Simple Predicate

We want to show the existence of a simple predicate. Therefore, we firstly start to construct a potential simple predicate and then secondly also prove this predicate to be indeed simple. Since on the one hand assuming the decidability of all predicates is consistent in an axiom-free constructive setting but simple predicates were on the other hand already shown to be undecidable, such a construction cannot work without further assumptions. We discussed different axioms of synthetic computability theory already in Chapter 3. Some of those axioms will be successively assumed throughout this chapter, which will be marked in red at the particular points. We introduced an enumerator $\mathcal{W} : \mathbb{N} \to X \to \mathbb{P}$ for semidecidable predicates, that can be interpreted as the halting problem. We roughly follow the original construction of Post and its presentation by Rogers, modifying the halting problem over natural numbers in a specific way to obtain a simple predicate. Therefore we assume from now on

$$\mathcal{W} : \mathbb{N} \to \mathbb{N} \to \mathbb{P} \text{ with its specification } \mathbf{ES} := \forall p : \mathbb{N} \to \mathbb{P}. \mathcal{S}p \leftrightarrow \exists c. \forall x. (\mathcal{W}cx \leftrightarrow px).$$

As mentioned in chapter 3, **ES** does not imply the semidecidability of $\mathcal{W}$. Since the intended construction of a simple predicate (and at the latest its semidecidability) heavily relies on this property of $\mathcal{W}$, we furthermore assume from now on

$$\mathcal{SW}.$$

Recall the several results and properties of $\mathcal{W}$ like its 1-completeness that we can use from now on with the above assumptions made. We start to adapt $\mathcal{W}$ by considering the auxiliary predicate $C : \mathbb{N} \times \mathbb{N} \to \mathbb{P}$ and introduce notions for the domain of $C$ and the range of an index $c$ under $C$:

$$\begin{aligned} C(c, x) &:= \mathcal{W}cx \wedge x > 2c \\ \mathsf{Dom}_C &:= \lambda c.\ \exists x. C(c, x) \\ \mathsf{Ran}_C\, c &:= \lambda x.\ C(c, x) \end{aligned}$$

Intuitively, $C(c, x)$ describes that the program with index $c$ halts on input $x$ with $x > 2c$. Therefore, $\mathsf{Dom}_C\, c$ holds iff the program with index $c$ has an halting input larger than $2c$. Notice the following basic properties of $C$ and its domain.

**Lemma 6.5** *1. $C$ and therefore $\mathsf{Dom}_C$ are not empty.*

*2. $C$ and $\mathsf{Dom}_C$ are strongly enumerable.*

**Proof** 1. Recall the index $c_\top$ with $\forall x. \mathcal{W}c_\top x$. Hence, $C(c_\top, 2c_\top + 1)$ and $\mathsf{Dom}_C\, c_\top$.

2. Semidecidability and enumerability do coincide on the datatype $\mathbb{N} \times \mathbb{N}$. Therefore, the claim follows by the assumption $\mathcal{SW}$, closure properties and 1. □

Unfortunately, $\mathsf{Ran}_C\, c$ can be empty for some indices, e.g. for the empty index $c_\perp$ since $\mathcal{W}c_\perp x$ holds for no $x$. Therefore, we will not be able to construct a strong enumerator but can at least prove the enumerability of $\mathsf{Ran}_C\, c$.

**Lemma 6.6** *For every $c$, $\mathsf{Ran}_C\, c$ is enumerable.*

**Proof** Follows again by closure properties with Lemma 6.5. □

We would like to define the simple predicate as the range of $C$. However, $C$ is clearly not co-infinite since for instance $\mathsf{Ran}_C\, c_\top x$ holds for every $x > 2c_\top$ such that the complement of $\mathsf{Ran}_C\, c_\top$ is bounded. We handle this problem by trying to find a function $\psi$, that computes a unique element in $\mathsf{Ran}_C\, c$ and then define the simple predicate as the range of $\psi$. Again because of the possible emptiness of $\mathsf{Ran}_C\, c$, $\psi$ cannot be a total function. Traditionally, one would simply solve this issue by using a partial function. In our case, we have to give $\psi$ a second parameter, namely a proof for $\mathsf{Dom}_C\, c = \exists x. C(c,x)$ and then compute a unique element in $\mathsf{Ran}_C\, c = \lambda x.\ C(c,x)$. We discussed exactly this kind of computation for decidable predicates by the operator $\mu_{\mathbb{N}}$ as well as for enumerable predicates by $\mu_{\mathcal{E}}$. Since $\mathsf{Ran}_C\, c$ cannot be shown decidable in general[3], we use given $c$ in $\mathsf{Dom}_C$ the recursive $\mu_{\mathcal{E}}$-operator from Lemma 3.23 for the enumerable predicate $\mathsf{Ran}_C\, c$ to define the function $\psi$ and our potential simple predicate as its range:

**Definition 6.7** *We define the function* $\psi : \forall c. \mathsf{Dom}_C\, c \to \mathbb{N}$ *via the* $\mu_{\mathcal{E}}$*-operator as*

$$\psi c H := \mu_{\mathcal{E}} H.$$

*Furthermore, we define the predicate* $S : \mathbb{N} \to \mathbb{P}$ *as the range of* $\psi$*, i.e.*

$$S x := \exists c (H : \mathsf{Dom}_C\, c).\ \psi c H = x.$$

By the specification of $\mu_{\mathcal{E}}$, we know that firstly the result of $\psi c H$ does not depend on the actual proof $H$ and secondly that $C(c, \psi c H)$ holds for all $c$ with $\mathsf{Dom}_C\, c$. In particular, $S$ is subset of the range of $C$ and $\psi c H > 2c$ for all $c$, which will become a key property of $S$ to prove its co-infinity.

In order to enumerate $S$, i.e. the range of $\psi$, we want to return the result of $\psi$ for every $c$ with $\mathsf{Dom}_C\, c$. By the type of $\psi$ it is clearly necessary to find a computation of proofs $H : \mathsf{Dom}_C\, c$ – the second parameter of $\psi$ – in order to enumerate its range. Fortunately, we can strongly enumerate $\mathsf{Dom}_C$ by some function $E : \mathbb{N} \to \mathbb{N}$, which gives us not only the possibility to enumerate the elements $c$ in $\mathsf{Dom}_C$, but also allows us to compute a proof of $\mathsf{Dom}_C\, (E n)$ for every given $n$.

**Lemma 6.8** *$S$ is strongly enumerable and therefore semidecidable.*

**Proof** The strong enumerator for $S$ is defined as follows:
Given $n$, we compute $E n$ and a proof $H : \mathsf{Dom}_C\, (E n)$, then return $\psi (E n) H$.

We continue to prove the properties $S$ should fulfill to be a simple predicate.

**Lemma 6.9** *$\overline{S}$ contains no semidecidable and infinite subset.*

**Proof** Assume a semidecidable and infinite predicate $q$ with $q \subseteq \overline{S}$. By **ES**, there is an index $c$ with $\forall x. \mathcal{W} c x \leftrightarrow q x$. We derive a contradiction by showing $q$ finite via $[0, \dots, 2c]$, i.e. showing that all elements satisfying $q$ are not larger than $2c$: Assume $q x$ – therefore $\mathcal{W} c x$ – and $x > 2c$. Hence, $C(c, x)$ and such that there exists $H : \mathsf{Dom}_C\, c$. Then

$$C(c, \psi c H) \Rightarrow \mathcal{W} c (\psi c H) \Rightarrow q(\psi c H) \Rightarrow \overline{S}(\psi c H),$$

a contradiction to the definition of $S$ as the range of $\psi$. □

[3]Traditionally it is easy to show that $\mathsf{Ran}_C\, c$ is actually undecidable for some indices $c$ using a universal machine

We already mentioned, that $\psi$ does not depend on the exact proof $H : \mathsf{Dom}_C\, c$. Therefore we omit from now on the second parameter and simply write "$\psi c$" and think consequently about $Sx$ as "$\exists c.\psi c = y$".

In order to show $S$ simple, it remains to show its co-infinity. Therefore, we first need some preparation:

**Definition 6.10** *We say that* $L$ *is a* ***listing of*** $p$ *up to a bound* $b$ *if* $L$ *is duplicate-free and*

$$\forall x.x \in L \leftrightarrow (px \land x \leqslant b).$$

**Lemma 6.11** *For all predicates* $p$ *and bounds* $b$*,*

1. *there weakly exists a listing of* $p$ *up to* $b$*, and*
2. *if* $L$ *is a listing of* $p$ *up to bound* $b$*, then* $[0, \dots, b] \setminus L$ *is a listing of* $\overline{p}$ *up to* $b$*.*

**Proof** 1. We prove the claim $\neg\neg\exists L.\#L \land \forall x.x \in L \leftrightarrow (px \land x \leqslant b)$ by induction on the bound $b$. Since we have to prove $\bot$, we decide $p0$ in the base case and $p(b+1)$ in the step case and construct the correct duplicate-free listing respectively.

2. Follows straightforward by the properties of " $\setminus$". □

With this, we are ready to again look at the predicate $S$ and show that it is indeed co-infinite. Now, the invested work in the last chapter to find many different characterizations of non-finite predicates pays out. Intuitively, $\overline{S}$ is infinite, since the list of numbers $[0, \dots, 2n]$ can contain at most $n$ elements in $S$ and therefore at least $n + 1$ in $\overline{S}$ for all natural numbers $n$: By $\psi c > 2c \geqslant 2n$ for $c \geqslant n$, at most $\{\psi 0, \psi 1, \dots, \psi(n-1)\}$ are the elements less or equal than $2n$ satisfying $S$. This intuition helps us to find the best fitting infinite criterion for $\overline{S}$. Since we already know, that $\overline{S}$ contains no semidecidable and infinite subset, we will not be able to compute elements of any number in $\overline{S}$. However it turns out, that we can show the weak existence of arbitrary long, duplicate-free lists containing only elements from $\overline{S}$. We formalize this intuition by looking first at the lists containing elements in $S$ up to the particular bound $2n$.

**Lemma 6.12** *For every duplicate-free list* $L$ *with* $\forall x \in L.Sx \land x \leqslant 2n$*,*

1. *there exists a duplicate-free list* $L_C$ *with* $|L_C| = |L|$ *and s.t.* $\forall c \in L_C.c < n \land \psi c \in L$*.*
2. $|L| \leqslant n$*.*

**Proof** Assume $\#L$ and $\forall x \in L. Sx \land x \leqslant 2n$.

1. By induction on $L$. In the step case $x :: L$, the inductive hypothesis yields a duplicate-free list $L'_C$ with $|L'_C| = |L|$ and $\forall c \in L'_C. c < n \land \psi c \in L$. Since $Sx$ and $x \leqslant 2n$ hold by assumption, there is $c < n$ with $x = \psi c$ and therefore $c \notin L'_C$. Finally, we choose $L_C := (c :: L'_C)$.
2. Consider the duplicate-free list $L_C$ from 1. Then $|L| = |L_X| \leqslant |[0, \dots, n]| = n$. □

By this auxiliary result, it is straightforward to prove that listings of $S$ up to $2n$ have a maximal length of $n$.

**Lemma 6.13** *For all $n$, there weakly exists a listing $L$ of $S$ up to $2n$ with $|L| \leqslant n$.*

**Proof** By Lemma 6.11, it is sufficient to show that all listings $L$ of $S$ up to $2n$ have a length of at most $n$. This follows by 6.12. □

**Lemma 6.14** *$S$ is co-infinite.*

**Proof** We show the infinity of $\overline{S}$ with Lemma 5.8 by the characterization

$$\forall n. \neg\neg\exists L. |L| \geqslant n \land \#L \land L \subseteq \overline{S}.$$

So let $n$ be a natural number and consider the weakly existent listing $L_S$ of $S$ (Lemma 6.11.1) with $|L_S| \leqslant n$ (Lemma 6.13). Then $[0, \dots, 2n] \setminus L_S$ is a weakly existent listing of $\overline{S}$ by Lemma 6.11.2, and therefore duplicate-free and a subset of $\overline{S}$. Finally $|L_S| \leqslant n$ implies $|[0, \dots, 2n] \setminus L_S| \geqslant n$. □

We successfully proved the constructed predicate $S$ to be simple and therefore the existence of a simple predicate.

**Theorem 6.15** *The predicate $S$ is a simple predicate.*

**Proof** Follows with Lemmas 6.8, 6.9, and 6.14. □

## 6.2 Properties of Simple Predicates & Post's Problem w.r.t. $\preceq_m$

Now that we have shown the existence of a simple predicate, we want to look closer at the properties of this predicate class. As mentioned in the introduction of this chapter, simple predicates provide a further understanding of the structure of reducibility degrees.

First of all, they answer the questions negatively, whether all semidecidable but undecidable predicates have the cylinder property introduced in chapter 4, which will then allow us conclude a distinction of one-one and many-one reducibility on undecidable predicates.

For proving simple predicates to not fulfill the cylinder property, Rogers uses Myhill's isomorphism theorem: By the equivalence of $\equiv_1$ and $\equiv$, he shows both the simple property and the cylinder property to be recursively invariant, i.e. that those notions transport through a computable bijection. This would also work out in our setting, however Myhill's isomorphism theorem is restricted to types $X \cong \mathbb{N}$. We can find a for our formalization actually simpler proof, that shows the generalized result:

**Lemma 6.16** *Simple predicates are no cylinders.*

**Proof** By Lemma 4.15 $p$ is a cylinder iff $\mathbb{N} \times p \preceq_1 p$. So let $p$ be simple and $\mathbb{N} \times p \preceq_1 p$ via $f$. We have to show $\perp$, therefore Lemma 5.9 and the co-infinity of $p$ yield an element $x_0$ with $\overline{p}x_0$. Then $\overline{p}$ contains the semidecidable (1.) and infinite (2.) subset (3.) $q := \lambda x.\, \exists n. x = f(x_0, n)$ (i.e. the range of $\{x_0\} \times \mathbb{N}$ under $f$), which contradicts the second condition of $\mathsf{simple}\ p$:

1. $\mathcal{S}q$: As the range of a semidecidable predicate, $q$ is also semidecidable.
2. $\mathsf{infinite}\ q$: By the closure of infinity under an injections, it suffices to show $\lambda(x, n).\ x = x_0$ to be infinite. But $\lambda(x, n).n$ is a surjection from this predicate into $\mathbb{N}$.
3. $q \subseteq \overline{p}$: Assume $x$ with $qx$ and $px$ and therefore $x = f(x_0, n)$ for some $n$. Hence, $px = p(f(x_0, n))$ implies $px_0$ since $p \times \mathrm{N} \preceq_1 p$ via $f$. But we assumed $\overline{p}x_0$, a contradiction. □

The above lemma shows the existence of undecidable predicates that do not fulfill the cylinder property and yields a distinction of one-one and many-one reducibility.

**Theorem 6.17 (Distinction of $\preceq_1$ and $\preceq_m$)**
*$\preceq_1$ and $\preceq_m$ and therefore $\equiv_1$ and $\equiv_m$ do not coincide on undecidable predicates.*

**Proof** The simple predicate $S$ is by Lemma 6.16 no cylinder and therefore $S \times \mathbb{N} \not\preceq_1 S$ and $S \not\equiv_1 S \times \mathbb{N}$ by Lemma 4.15, but $S \times \mathbb{N} \preceq_m S$ and $S \equiv_m S \times \mathbb{N}$ by Lemma 4.12. □

We come back to the original paper of Post [28] and his motivation to invent simple predicate. Those predicates gave Post an example of semidecidable but undecidable predicates that are not $m$-complete. As mentioned, his definition of simple predicates was reasoned by a property of $m$-complete predicates as predicates, whose complements contain a semidecidable and infinite subset. In order to formalize Post's problem with respect to many-one reduction, we want to show this property of $m$-complete predicates. The proof relies on so-called productive and creative predicates. Since the definition of those notions uses the enumerator of semidecidable predicates $\mathcal{W}$, we have to restrict the definition to predicates over the natural numbers.

**Definition 6.18 (Productive and Creative Predicates)**

1. *A function* $g : \mathbb{N} \to \mathbb{N}$ *is a* ***productive function for*** $p : \mathbb{N} \to \mathbb{P}$*, if*

$$\forall c. \mathcal{W}c \subseteq p \to (p \setminus \mathcal{W}c)(gc).$$

2. $p$ *is called* ***productive*** *if it has a productive function.*
3. $p$ *is called* ***creative*** *if it is semidecidable and* $\overline{p}$ *is productive.*

The canonical example for a creative predicate is the special halting problem $\mathcal{W}_0 : \mathbb{N} \to \mathbb{P}$, i.e. the predicate describing that the program with index $n$ halts on input $n$: It is semidecidable since we assumed the semidecidability of $\mathcal{W}$ and $\overline{\mathcal{W}_0}$ is productive via $g := \lambda n.n$ since $\mathcal{W}_0$ was defined as $\mathcal{W}_0 n := \mathcal{W}nn$.

The usage of productive and creative predicates for our purposes becomes clear by the next result. We show that $m$-complete predicates are also creative and therefore their complements productive. Proving this implication needs a further axiom in synthetic computability theory, namely the in chapter 3 discussed corollary out of the (simple) $S^m_n$-theorem. We assume from now on

$$\mathsf{SMN'} := \forall f. \exists k. \forall c x. \mathcal{W}(kc)x \leftrightarrow \mathcal{W}c(fx).$$

**Lemma 6.19** *For all predicates* $p : \mathbb{N} \to \mathbb{P}$ *and* $q : \mathbb{N} \to \mathbb{P}$*,*

1. *productive* $p \to \neg \mathcal{S}p$.
2. $p \preceq_m q \to$ *productive* $p \to$ *productive* $q$.
3. *creative* $p \to \neg \mathcal{D}p$.
4. $p \preceq_m q \to$ *creative* $p \to$ *productive* $\overline{q}$.
5. $m$*-complete* $p \to$ *creative* $p$.

**Proof** 1. Let $p$ be productive via $g$ and semidecidable. By **ES**, there exists $c$ with $H : \forall x. \mathcal{W}cx \leftrightarrow px$. Therefore, $\mathcal{W}c \subseteq p$ which implies by assumption $(p \setminus \mathcal{W}c)(gc)$. A contradiction towards $H(gc)$.

2. Let $p \preceq_m q$ via $f$ and $p$ productive via $g$. By SMN', there is a function $k : \mathbb{N} \to \mathbb{N}$ with $H : \forall cx. W(kc)x \leftrightarrow Wc(fx)$. Then $q$ is productive via $\lambda c.\, f(g(kc))$: $Wc \subseteq q$ implies $W(kc) \subseteq p$ by $H$ and $p \preceq_m q$ via $f$. Hence, $(q \setminus \mathcal{W}c)(f(g(kc)))$.

3. Follows with 1.
4. Follows with 2.
5. Let $p$ be $m$-complete. Then $p$ is semidecidable and the creative predicate $\mathcal{W}_0$ reduces to $p$. Hence, $\overline{p}$ is productive by 4. $\square$

By the last result, it will now suffice to construct a semidecidable and infinite subset for productive predicates in order show the existence of such a subset in the complement of a $m$-complete predicate. We will now focus on this construction for a fixed productive predicate $p$. We then try to recursively build up arbitrarily long, duplicate-free lists of elements satisfying $p$. This recursive construction will need as a further axiom the computability of an index halting exactly on a given list $L$. Formally we assume from now on,

$$\textsf{LIST-ID} := \forall L. \Sigma c_L. \forall x. \mathcal{W} c x \leftrightarrow x \in L.$$

As shown in 3.19, this axiom can be derived out of the $\mathsf{cns}$-operator.

So fix $p : \mathbb{N} \to \mathbb{P}$ be productive via $g : \mathbb{N} \to \mathbb{N}$ and therefore

$$P : \forall c. \mathcal{W} c \subseteq p \to (p \setminus \mathcal{W} c)(gc).$$

Consider the following recursive function $f_{\mathcal{L}} : \mathbb{N} \to \mathcal{L}(\mathbb{N})$

$$\begin{aligned} f_{\mathcal{L}} 0 &:= [\,] \\ f_{\mathcal{L}}(Sn) &:= \mathsf{let}\ L := f_{\mathcal{L}} n\ \mathsf{in}\ (gc_L) :: L \end{aligned}$$

with the following properties:

**Lemma 6.20** *For all natural numbers $n$,*

1. $f_{\mathcal{L}} n \subseteq p$,
2. $|f_{\mathcal{L}} n| = n$, *and*
3. $f_{\mathcal{L}} n$ *is duplicate-free.*

**Proof** All three claims follow by induction on $n$:

1. The base case is contradictory, in the step case let $L := f_{\mathcal{L}} n$ and assume $x \in (gc_L) :: L$. For $x \in L$, the inductive hypothesis implies $px$. For $x = gc_L$, we have to show $px = p(gc_L)$, which follows with $P$ if we have $\mathcal{W} c_L \subseteq p$. But for $x_0$ with $\mathcal{W} c_L x_0$ and therefore $x_0 \in L$, the inductive hypothesis again implies $px_0$.
2. Straightforward.

3. The base case is again trivial, in the step case $\mathsf{L} := \mathsf{f}_{\mathcal{L}}\mathfrak{n}$ is duplicate-free by the inductive hypothesis. Therefore, it suffices to show $\mathsf{gc}_\mathsf{L} \notin \mathsf{L}$. But $\mathcal{W}\mathsf{c}_\mathsf{L} \subseteq \mathsf{p}$ by 1. and therefore $\mathsf{P}$ implies $(\mathsf{p} \setminus \mathcal{W}\mathsf{c}_\mathsf{L})(\mathsf{gc}_\mathsf{L})$ and in particular $\overline{\mathcal{W}}\mathsf{c}_\mathsf{L}(\mathsf{gc}_\mathsf{L})$. Hence, $\mathsf{gc}_\mathsf{L} \in \mathsf{L}$ would imply $\mathcal{W}\mathsf{c}_\mathsf{L}(\mathsf{gc}_\mathsf{L})$, a contradiction. □

This allows us to conclude, that $\mathsf{p}$ contains a semidecidable and infinite predicate. We define this predicate as exactly those elements, that appear in the lists $\mathsf{f}_{\mathcal{L}}\mathfrak{n}$ for some $\mathfrak{n}$. Showing, that this predicate is a predicate as desired is easy by using the matching non-finiteness criterion from the last chapter:

**Lemma 6.21** *Productive predicates* $\mathsf{p}$ *contain a semidecidable and infinite predicate, i.e.*

$$\exists q.\mathcal{S}q \wedge \mathsf{infinite}\ q \wedge q \subseteq p.$$

**Proof** For $q := \lambda x.\ \exists n. x \in f_{\mathcal{L}}n$, we conclude:

1. $\mathcal{S}q$ via $\lambda x n.\ \ulcorner x \in f_{\mathcal{L}}n \urcorner$.
2. $\mathsf{infinite}\ q$, since $q$ contains elements of any number $n$:

$$|f_{\mathcal{L}}n| = n \wedge \# f_{\mathcal{L}}n \wedge f_{\mathcal{L}} \subseteq q \qquad \text{(Lemma 6.20)}$$

3. $q \subseteq p$, since $qx$ implies $x \in f_{\mathcal{L}}n$ for some $n$ and therefore $px$ again by Lemma 6.20. □

We can combine our results with the construction of the simple predicate $\mathsf{S}$ in the last section to show the existence of not $\mathfrak{m}$-complete predicates in order to finish the formalization of Post's problem with respect to many-one reductions.

**Corollary 6.22** *Simple Predicates are not* $\mathfrak{m}$*-complete.*

**Proof** Follows with Lemmas 6.19 and 6.21. □

**Theorem 6.23 (Post's Problem w.r.t. $\preceq_m$)**

1. *There exists a semidecidable but undecidable predicates that is not* $\mathfrak{m}$*-complete.*
2. *There exists a semidecidable but undecidable predicate* $\mathsf{p}$*, such that the halting problem does not many-one reduce to* $\mathsf{p}$*.*

**Proof** Both claims are equivalent by Lemma 3.18 and follows. 1. follows with the simple predicate $\mathsf{S}$ from Theorem 6.15 with Lemma 6.4 and Corollary 6.22. □

Post's problem with respect to many-one reductions also answers the problem with respect to one-one reduction: Simple predicates are not $\mathfrak{m}$-complete and hence not 1-complete. Furthermore, $\mathcal{W}$ was shown to be 1-complete such that simple predicates do also not one-one reduce to the halting problem.

## 6.3 Distinction of Many-One and Truth-Table Reducibility

We saw in the last section, that simple predicates are useful to solve Post's problem for many-one reductions and therefore distinguish different classes of 1- and $m$-degrees over the semidecidable but undecidable predicates. Furthermore, they yield a distinction of one-one and many-one reducibility, since simple predicate do not fulfill the cylinder property.

It turns out that the class of simple predicates contains also predicates that let us distinguish many-one and truth-table reductions. We know from chapter 3, that $p \preceq_m q$ implies $p \preceq_{tt} q$ and therefore for their computability degrees, that $\equiv_m$ is a subset of $\equiv_{tt}$. However, this subset is a strict one: We can construct a truth-table complete predicate, that is not many-one complete and has therefore different $m$- and $tt$-degrees.

The construction of such a predicate works by modifying the simple predicate $S$, which is again inspired by the presentations of Rogers. He united $S$ with particular lists of numbers in the following way

$$Sx \lor \left(\exists n. \mathcal{W}_0 n \land x \in [2^n - 1, \dots, 2^{n+1} - 2]\right)$$

and showed this disjunction to be still simple and therefore not $m$-complete but $tt$-complete. However, following this construction in our setting comes with two problems: In order to show the disjunction co-infinite, one first has to show $\mathcal{W}_0$ co-infinite. This is for us not possible without further assumptions like the possibility to find arbitrarily large program indices with the same behavior. Furthermore, showing the modified predicate $tt$-complete traditionally works by a reduction from $\mathcal{W}_0$. However, showing $\mathcal{W}_0$ to be $tt$-complete or equivalently $\mathcal{W} \preceq_{tt} \mathcal{W}_0$ seems to require stronger axioms if not even a universal machine.

It turns out, that both problems can be solved using $\mathcal{W}$ instead of $\mathcal{W}_0$. We therefore unite $S$ with the lists $[2^n - 1, \dots, 2^{n+1} - 2]$ for those $n$ that encode a pair $(c, x)$ fulfilling $\mathcal{W}cx$ . Formally the new predicate $S^* : \mathbb{N} \to \mathbb{P}$ is defined as

$$S^* x := Sx \lor \left(\exists n. \mathcal{W}(\pi_1 n)(\pi_2 n) \land x \in [2^n - 1, \dots, 2^{n+1} - 2]\right).$$

The most difficult part of showing $S^*$ simple is again the co-infinity proof. We can however come back and use some already proven results that help us to show $S^*$ co-infinite: Properties of $S$ but especially the work from the last chapter about infinite predicates pays again off by providing the matching criterion to show $S^*$ co-infinite.

**Lemma 6.24** 1. *For all* $x, n_1$, *and* $n_2$,

$$x \in [2^{n_1} - 1, \dots, 2^{n_1+1} - 2] \to x \in [2^{n_2} - 1, \dots, 2^{n_2+1} - 2] \to n_1 = n_2.$$

2. *For all* $n$*, there weakly exists* $x \in [2^n - 1, \dots, 2^{n+1} - 2]]$ *with* $\overline{S}x$*.*

3. $\lambda n.\ \mathcal{W}(\pi_1 n)(\pi_2 n)$ *is co-infinite.*

4. $S^*$ *is co-infinite.*

**Proof** 1. Follows since $[2^{n_1} - 1, \dots, 2^{n_1+1} - 2]$ and $[2^{n_2} - 1, \dots, 2^{n_2+1} - 2]$ are disjoint for $n_1 \neq n_2$.

2. The following implication holds for all lists $L$ and predicates $p$:

$$(L \nsubseteq p) \to \neg\neg\exists x \in L.\overline{p}x.$$

It is proven by induction on $L$ and deciding $pa$ in the step case $a :: L$. It therefore suffices to show $[2^n - 1, \dots, 2^{n+1} - 2] \nsubseteq S$. But since $2^{n+1} - 2 = 2 \cdot (2^n - 1)$, assuming $[2^n - 1, \dots, 2^{n+1} - 2] \subseteq S$ implies by Lemma 6.12 (with $n := 2^n - 1$) $|[2^n - 1, \dots, 2^{n+1} - 2]| \leqslant 2^n - 1$ . Contradictory, since $|[2^n - 1, \dots, 2^{n+1} - 2]| = 2^n$.

3. $\mathbb{N} \hookrightarrow (\lambda n.\ \mathcal{W}(\pi_1 n)(\pi_2 n))$ via $\lambda n.\ \langle c_\perp, n\rangle$. The claim follows by infinite criteria.

4. We show $S^*$ co-infinite by the equivalent criterion

$$\forall n. \neg\neg\exists x \geqslant n \land \overline{S^*}x.$$

By 3., there weakly exist arbitrarily large elements $n_1$ with $\overline{\mathcal{W}}(\pi_1 n_1)(\pi_2 n_1)$. For $n : \mathbb{N}$, choose such an $n_1 > n$. By 2., there weakly exists an element $x \in [2^{n_1} - 1, \dots, 2^{n_1+1} - 2]$ with $\overline{S}x$. Therefore, $x \geqslant n_1 - 1 \geqslant n$ and by 1.,

$$\overline{S}x \land \neg\exists n_2.\mathcal{W}(\pi_1 n_2)(\pi_2 n_2) \land x \in [2^{n_2} - 1, \dots, 2^{n_2+1} - 2] \Leftrightarrow \overline{S^*}x. \qquad \square$$

Since $S^*$ should be a simple predicate, we know for the same reasons as for $S$ that the above proof cannot be simplified using other notions as infinite criterion but must again work by a characterization requiring the weak existence.

Showing the remaining two properties of $S^*$ to be a simple predicate becomes relatively easy using the already proven corresponding properties of $S$:

**Lemma 6.25** $S^*$ *is a simple predicate.*

**Proof** Follows with

1. $\mathcal{S}S^*$: Straightforward by closure properties of semidecidable predicates and the semidecidability of $S$ and $\mathcal{W}$.

2. $\overline{S^*}$ contains no semidecidable and infinite subset: $S \subseteq S^*$ implies $\overline{S^*} \subseteq \overline{S}$ and $\overline{S}$ contains no such predicate by Lemma 6.9.

3. Lemma 6.24. $\square$

In order to show the truth-table completeness of $S^*$, we need a small auxiliary lemma characterizing lists containing only elements in $S^*x$. Even though the formulation of the statement is technical, it follows easily by the definition of $S^*$ as a disjunction.

**Lemma 6.26** *For all lists* $L \subseteq S^*$, *we have*

$$L \subseteq S \lor \left(\exists n x. \mathcal{W}(\pi_1 n)(\pi_2 n) \land x \in L \land x \in [2^{n_1} - 1, \dots, 2^{n_1+1} - 2]\right).$$

**Proof** Straightforward by induction on $L$. □

As already mentioned, we show $S^*$ $m$-complete by a reduction from $\mathcal{W}$. Given an instance $n := \langle c, x\rangle$ of the halting problem, we use the query list $[2^n - 1, \dots, 2^{n+1} - 2]$ and check whether all entries of the corresponding boolean list are true. Recall the already many times used property of $S$, that not all elements of particular lists – in this case the list $[2^n - 1, \dots, 2^{n+1} - 2]$ – can satisfy $S$. This observation in combination with the above lemma will allow us to show that the informal reduction above is indeed a truth-table reduction form $\mathcal{W}$ to $S^*$.

**Lemma 6.27** $S^*$ *is* $\mathsf{tt}$*-complete.*

**Proof** Since $S^*$ is semidecidable, it suffices by Lemma 3.18 to show $\mathcal{W} \preceq_{tt} S^*$. The truth-table reduction works via

$$\begin{aligned} f &:= \lambda(c, x).\ \mathsf{let}\ n := \langle c, x\rangle\ \mathsf{in}\ [2^n - 1, \dots, 2^{n+1} - 2] \\ \text{and}\quad \alpha &:= \lambda(c, x)\ L_B.\ \mathsf{if}\ \ulcorner \forall b \in L. b = \mathsf{true} \urcorner\ \mathsf{then\ true\ else\ false}. \end{aligned}$$

For $n := \langle c, x\rangle$ and $[2^n - 1, \dots, 2^{n+1} - 2] \mathrel{\hat{=}}_{S^*} L_B$, we have to show

$$\mathcal{W}cx \leftrightarrow \alpha(c, x)L_B = \mathsf{true}.$$

$\rightarrow$: By $\mathcal{W}cx \Rightarrow [2^n - 1, \dots, 2^{n+1} - 2] \subseteq S^* \Rightarrow \forall b \in L. b = \mathsf{true} \Rightarrow \alpha(c, x)L_B = \mathsf{true}$.

$\leftarrow$: We have $\alpha(c, x)L_B = \mathsf{true} \Rightarrow \forall b \in L. b = \mathsf{true} \Rightarrow [2^n - 1, \dots, 2^{n+1} - 2] \subseteq S^*$. By Lemma 6.26, there are two cases:

1. $[2^n - 1, \dots, 2^{n+1} - 2] \subseteq S$. Contradiction again by Lemma 6.12.
2. $\exists n_1 x. \mathcal{W}(\pi_1 n_1)(\pi_2 n_1) \land x \in [2^n - 1, \dots, 2^{n+1} - 2] \land x \in [2^{n_1} - 1, \dots, 2^{n_1+1} - 2]$. By 6.24, we conclude $n_1 = n$ and therefore $\mathcal{W}cx$. □

Remember that simple predicates gave us an answer to Post's problem with respect to $\preceq_m$. The above results shows, that at least not every simple predicates serves as an example answering Post's problem for truth-table reducibility since we saw that $\mathcal{W}$ does actually truth-table reduce to a simple predicate.

However, the constructed simple predicate $S^*$ and its $\mathtt{tt}$-completeness allows us to distinguish many-one and truth-table reducibility and their respective completeness classes. Notice that we use the result, that simple are not $\mathtt{m}$-complete and have therefore still to assume not only $\mathcal{W}$ and its semidecidability, but also the two further axioms **SMN'** and **LIST− ID**.

**Theorem 6.28** (**Distinction of $\preceq_{\mathtt{m}}$ and $\preceq_{\mathtt{tt}}$**)

1. *$\mathtt{m}$-completeness and $\mathtt{tt}$-completeness do not coincide.*
2. *$\preceq_{\mathtt{m}}$ and $\preceq_{\mathtt{tt}}$ as well as $\equiv_{\mathtt{m}}$ and $\equiv_{\mathtt{tt}}$ do not coincide on semidecidable but undecidable predicates.*

**Proof** $S^*$ is simple and therefore not $\mathtt{m}$-complete by Corollary 6.22, it is however $\mathtt{tt}$-complete by Lemma 6.27. 2. follows with 1. and $\mathcal{W} \preceq_{\mathtt{tt}} S^*$ from the Proof 6.27. □

We saw that the special class of simple predicates allows us to get a clearer view on the structure of the different reducibility degrees. This class of semidecidable predicates with complements containing though infinite no semidecidable and infinite subset is a class of undecidable but not $\mathtt{m}$-complete predicates. It therefore distinguishes the semidecidable but undecidable predicates in different $\mathtt{m}$-degrees and answers the question of Post's problem with respect to many-one reductions: There are semidecidable but undecidable predicates, that are not many-one reducible from the halting problem. Furthermore, simple predicates allow us to find both predicates distinguishing $\preceq_1$ and $\preceq_{\mathtt{m}}$ as well as $\preceq_{\mathtt{m}}$ and $\preceq_{\mathtt{tt}}$ on the class of semidecidable but undecidable predicates.

# Chapter 7

# Mechanization Details

We finish our formalization at this point and want to take a closer look at the mechanization. Besides general remarks about the mechanized reducibility theory in Coq, we also point out some important decision details throughout the development.

**Coq Development** We roughly divided the proof files with respect to the chapter structure of the thesis. There is furthermore a small file containing the main definitions of the thesis like the different reducibility notions and synthetic axioms. The main results of chapter 6 like Post's problem for many-one reductions and reducibility distinctions are summarized in a small separate file, where we state in each theorem the particularly required synthetic axioms explicitly.

| **Content** | **Specification** | **Proofs** |
|---|---|---|
| General Preliminaries | 94 | 80 |
| List Preliminaries | 105 | 207 |
| Main Definitions | 72 | 0 |
| Synthetic Computability Theory | 266 | 550 |
| Recursive μ-Operator | 105 | 141 |
| Corresponding Lists (for truth-table reductions) | 22 | 96 |
| Myhill's Isomorphism Theorem | 109 | 279 |
| Reduction Characterization | 152 | 262 |
| Infinite Predicates | 102 | 308 |
| Simple Predicates | 239 | 504 |
| Post's Problem & Reduction Distinctions | 134 | 8 |
| **TOTAL** | 1400 | 2435 |

Table: Overview of the Coq Development, Source Code available under `https://github.com/uds-psl/synthetic-reducibility-in-coq`

**External Code** Besides the use of Coq's Standard Library – here especially results regarding lists and their properties and the Equations package – we drew upon external code at two points in the preliminary developments. Firstly, the code in the separate file `Cantor_Pairing_Coq.v` proving Cantor's pairing function on natural numbers is written by Andrej Dudenhefner. Secondly we extend the witness operator proof via the guard predicate from 2019's "Introduction of Computational Logic" lecture at Saarland University written by Gert Smolka in the file `Recursive_Mu_Operator_Coq.v` to a proof of the recursive $\mu$-operator for decidable as well as enumerable predicates. The concerning parts are marked in the proof scripts.

**General Types** As far as possible, we tried to mechanize the presented proofs for predicates over arbitrary types. However, some results are based on properties of the traditionally underlying type of natural numbers. Therefore, we either had to restrict the base types to be isomorphic to $\mathbb{N}$ like in Myhill's isomorphism theorem or we assumed explicit properties of the underlying types. This allowed us to mechanize the most general version of for example reducibility characterizations and stated explicitly which property of the underlying types are required. An exception is the work in chapter 6: Even though we could define simple predicates over general types, we constructed and proved the relevant properties only for predicates over $\mathbb{N}$. This allowed us again to use properties of natural numbers but even more significantly, it was therefore sufficient to assume the necessary axioms of synthetic computability theory only for the type $\mathbb{N}$. Instead of quantifying for instance $\mathcal{W}$ over arbitrary types, we assumed weaker versions of the axioms. Even though this led to a no longer general typed development, the main results derived from with simple predicates remained comparably strong: The distinctions of different computability degrees worked by constructing distinguishing counterexamples that were then chosen as predicates over the natural numbers.

**Reduction Completeness** Generalizing the underlying predicate type was not only desirable but at some points also necessary. However, this brought also some difficulties: Recall for instance the quite technical Definition 3.7 of completeness with respect to different reducibility notions of a predicate $p : X \to \mathbb{P}$:

$$\text{complete } p := \mathcal{S}p \land \forall Y.\, X \cong Y \to \text{discrete } Y \to \forall q : (Y \to \mathbb{P}).\, \mathcal{S}q \to q \preceq p.$$

We had to restrict the type of predicates that should reduce to $p$ to isomorphic and discrete types. Otherwise it would not have been possible to show for example the 1-completeness of $\mathcal{W}$, since applying **ES** is only possible for predicates over natural numbers. To do so, we compose a given predicate with the isomorphism provided by $\mathbb{N} \cong Y$[1] to obtain an "isomorphic predicate" $q' : \mathbb{N} \to \mathbb{P}$. Furthermore, the then

[1]Actually, the type of $\mathcal{W} : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ (by uncurrying $\mathcal{W} : (\mathbb{N} \times \mathbb{N}) \to \mathbb{P}$) allows in the first step only to assume an isomorphism $\mathbb{N} \times \mathbb{N} \cong Y$. But then $\mathbb{N} \cong \mathbb{N} \times \mathbb{N} \cong Y$.

constructed reduction function works the other way around, such that we have to invert the isomorphism. This in turn is only possible for discrete types $Y$[2]. A similar construction with "isomorphic predicates" is done in the generalized Myhill's isomorphism theorem 4.11, that is proven for isomporphic to $\mathbb{N}$ and discrete types. Regarding the completeness, there is an easy but minimal generalization opportunity by requiring only $X \twoheadrightarrow Y$ instead of $X \cong Y$ and then using the right inverse function.

**Truth-Table Reductions** Another highly technical part in the development that has to be discussed is the mechanization of truth-table reductions. Recall their Definition 3.4

$$p \preceq_{tt} q := \exists f : X \to \mathcal{L}(Y).\ \exists \alpha : \forall x.\mathbb{B}^{|fx|} \to \mathbb{B}.\ \forall xL.(fx) \mathrel{\widehat{=}_q} L \to (px \leftrightarrow \alpha x L = \mathsf{true})$$

with the special type $\alpha : \forall x.\mathbb{B}^{|fx|} \to \mathbb{B}$ of the truth-table condition, required to find an embedding for the later defined truth-table cylinders in chapter 4.

We mechanize the function type not by using Coq's vector type from the standard library – the type is quite tedious to handle – but by adding a proof for the correct length of the boolean list as an further argument, i.e.

$$\alpha : \forall x \forall L : \mathcal{L}(\mathbb{B}).\ |L| = |fx| \to \mathbb{B}.$$

The mechanization of truth-table cylinders in chapter 4.3 worked in the same way. The additional argument suffices then to show by induction on $n$, that the type $\forall L : \mathcal{L}(\mathbb{B}).\ |L| = n \to \mathbb{B}$ contains exactly $2^n$ elements and that the truth-table cylinder can be therefore encoded into a countable type.

Proving this result is only possible by using proof irrelevance for the proof of the length of the boolean list, i.e. $\forall H_1 H_2 : |L| = |fx|, H_1 = H_2$, which is however provable for equality over discrete types already in axiom-free Coq.

Lastly, notice similar to the type-theoretical characterization of decidability from Lemma 3.8 a quite useful characterization of truth-table reducibility:

$$p \preceq_{tt} q \leftrightarrow \exists f.\ \mathsf{inhabited}\ \big(\forall x(L : \mathbb{B}^{|fx|}).\Sigma b.(fx) \mathrel{\widehat{=}_q} L \to (px \leftrightarrow b = \mathsf{true})\big).$$

whereby the proposition $\mathsf{inhabited}\ X$ simply states that there exists an element in $X$.

We mechanize $L : \mathbb{B}^{|fx|}$ again by requiring a proof for the correct list length. However, we can now use this proof to define the truth-table condition within a proof

[2]Notice that $X \cong Y$ for a discrete $X$ does not imply the discreteness of $Y$, since we defined surjectivity not as a computational existence.

script instead within an explicit functional definition. This pays off especially when dealing with proofs for the length of query lists, since we do not have to process or rewrite in those elements of highly dependent types in within proof term.

**Total Functions** A decisive aspect of computability theory are partial functions, traditionally for example already included in the definitions of semidecidability or enumerability.

Coq's type theory works solely with total function types and therefore guarantees occurring functions not only to be computable but also to be total. Defining recursive functions in Coq only works by recursion on an inductive datatype like the natural numbers or lists or by giving an explicit termination argument like required for size recursive functions.

Even though it is in some ways possible to imitate partial functions in Coq, they are uncomfortable to handle, such that we decided at multiple points in the development to find solutions that stayed in the familiar total function space.

Those solution were various and differed depending on the concrete situation. Recall for instance the method of step indices and option types, used to define semideciders and enumerators as total functions.

Another pleasant concept was used in the construction of the simple predicate $S$ as the range of the traditionally partial function $\psi$ (Definition 6.7): We added a proof justifying the definedness of the function for the given input to the type of $\psi$. Since we had to apply $\psi$ in the following only on elements in its support, the situation was ideally suited for this solution.

We combined this concept with step indices in the formalization of Myhill's isomorphism theorem when traveling through the correspondence sequence. The search algorithm could potentially loop and therefore diverge for starting values already appearing in the correspondence sequence. Since we applied the obtained algorithm also in this case only on converging inputs, requiring an additional proof (for the starting value to be fresh) worked out again beautifully in this situation allowing us to work only with total functions.

# Chapter 8

# Further Results in Reducibility Theory & Future Work

We finished our formalization with proving different distinctions of reducibility degrees in Chapter 6. This chapter gives an overview on further results in reducibility theory, which could be interesting and informative to formalize, and discusses in addition possible future work related to this thesis.

## 8.1 One-One and Many-One Completeness

By constructing the simple predicate $S^*$, we were able to distinguish $\mathfrak{m}$-completeness and $\mathfrak{tt}$-completeness. Similarly, one could also ask whether there is a distinction of 1-completeness and $\mathfrak{m}$-completeness. We saw that one-one and many-one reductions are not equivalent on the class of semidecidable but undecidable predicates and so are their computability degrees not equivalent. However, searching for an semidecidable predicate yielding a distinction of their completeness degrees, i.e. a predicate that is $\mathfrak{m}$-complete but not 1-complete, fails. For example the well-known $\mathfrak{m}$-complete halting problem was shown to be also 1-complete. Instead, one can show the both notions of 1- and $\mathfrak{m}$-completeness equivalent. While the forward direction is obvious, Rogers presents an highly involved proof for the backward direction [31]. He defines so-called canonical indices that encode finite predicates in a special way using base-2-exponentation. These indices are then used to one-one reduce the (1-complete) halting problem to every $\mathfrak{m}$-complete predicate.

The classical proof presented by Rogers uses as above mentioned finite predicates as well as further properties of infinite predicates. Most likely, we can define canonical indices not for finite predicates but for duplicate-free lists of natural numbers and prove similar properties for those canonical indices now for lists. However, it is not clear whether our definition of infinite predicates that we decided on in chapter 6 will be the working notion for infinite predicates for this proof. Rogers uses the classical readily to prove fact, that undecidable predicates have to be infinite, which seems however to be constructively not provable with our definition.

Another approach to this proof could be a characterization of 1-complete predicates Post mentioned when explaining his motivation to define simple predicates [28]. He stated, that a semidecidable predicate is 1-complete if and only if it is infinite and its complement contains a semidecidable and infinite subset. We already showed by the notion of productive and creative predicates, that the complement of $m$-complete predicates contains such a particular subset. Therefore, this sufficient criterion for 1-completeness could be used to find a less technical and easier to formalize proof than the one in Rogers' presentation. Also in this second possible approach, it would be again interesting to analyze the particular role of infinite predicates in the proof.

Furthermore, the equivalence of 1- and $m$-completeness could then be combined with already formalized work to conclude an even more unexpected result: By Myhill's isomorphism theorem two $m$-complete predicates have not only the same 1-degree but are also isomorphic, such that there exists even a bijection between all $m$-complete predicates.

## 8.2 Bounded Truth-Table Reductions

We discussed Post's problem with respect to different notions of reductions: While simple predicates could solve the problem for one-one and many-one reductions in general, we saw that at least some simple predicates are $tt$-complete and therefore fail to solve Post's problem for truth-table reductions. Post [28] introduced a further notion of reducibility, that is quite similar to truth-table reductions: He searched for an intermediate reduction between many-one and truth-table reductions, such that simple predicates are still provable to be not complete with respect to this new notion. He came up with so-called bounded truth-table reductions: For two predicates $p : X \to \mathbb{P}$ and $q : Y \to \mathbb{P}$, a bounded truth-table reduction consists again out of a query function $f : X \to \mathcal{L}(Y)$ and a truth-table condition $\alpha : \forall x. \mathbb{B}^{|fx|} \to \mathbb{B}$. However, there has to be now a fixed bound for the length of the query list, i.e. the length of $fx$ must be bounded by some natural number $b$ for all $x$. Formally, the definition of such bounded truth-table reductions could work as a straightforward adaption of truth-table reductions:

$$p \preceq_{btt} q := \exists b : \mathbb{N}. \exists f \alpha.\ \forall x L_B. |fx| \leqslant b \land \left(fx \mathrel{\widehat{=}_q} L_B \to (px \leftrightarrow \alpha x L_B = \mathsf{true})\right).$$

This reduction relation is again a preorder and its computability degree, defined analogously to the computability degrees of other reducibility notions, forms an equivalence relation. Showing the transitivity should work by adapting the technical construction showing transitivity of truth-table reductions in Lemma 3.11. Bounded truth-table reductions are furthermore indeed an intermediate reducibility notion, i.e.

$$\preceq_m \subsetneq \preceq_{btt} \subsetneq \preceq_{tt} .$$

Notice, that both subsets are also strict on the semidecidable but undecidable predicates and imply a distinction of both many-one and bounded truth-table reducibility as well as bounded truth-table and (unbounded) truth-table reducibility on this predicate class. The first distinction is due to Fischer [11], who used the constructed predicate $S^*$ from chapter 6 to show $S^* \times S^* \not\preceq_m S^*$ but $S^* \times S^* \preceq_{btt} S^*$.

The second distinction follows by showing Post's actual reason to invent the notion of bounded truth-table reductions: Simple predicates serve as a solution of Post's problem with respect to this intermediate notion of reducibility. By looking closely at the truth-table reduction $\mathcal{W} \preceq_{tt} S^*$ (Proof 6.27), that showed the simple predicate $S^*$ to be tt-complete, we see that this reduction is actually not bounded: The length of the query-list $[2^n - 1, \dots, 2^{n+1} - 2]$ has no upper bound such that the proof does not show $S^*$ to be btt-complete. Instead one can actually prove all simple predicates to be not btt-complete.

Formalizing this new notion of bounded truth-table reductions and especially the proof of Post's problem with respect to this reducibility notion could be an interesting further step, that could take use of our previous results regarding simple predicates.

## 8.3 Hyper-Simple Predicates & Post's Problem w.r.t $\preceq_{tt}$

Post answered the question of Post's problem not only for many-one and bounded truth-table reductions by simple predicates. He was also able to find a further class of semidecidable but undecidable predicates, that yield a solution to the problem with respect to truth-table reducibility. As mentioned multiple times above, general simple predicates were not sufficient to show the existence of a not tt-complete, semidecidable, but undecidable predicate. However, Post came up with a subclass of simple predicates, the so-called hyper-simple predicate, and showed this class to be again not empty and this time to be in general not tt-complete [28].

He strengthened the second condition of simple predicates (i.e. that complements of simple predicates are not allowed to contain a semidecidable and infinite subset) to define this stronger class of hyper-simple predicates. While he used so-called mutually exclusive finite sequences, Rogers presented an easier accessible definition of hyper-simple predicates by first defining majorizing functions for infinite predicates. Informally a function $f$ majorizes a particular infinite predicate $p$, if $\forall n.\ fn \geqslant x_n$, where $(x_n)_{n \in \mathbb{N}}$ are the elements satisfying $p$ in strictly increasing order. Hyper-simple predicates can then be defined to be semidecidable and co-infinite predicates, such that no computable function majorizes the complement of the predicate. This definition can be shown stronger as the definition of simple predicates; hence every hyper-simple predicate is indeed also simple.

Both in this thesis constructed simple predicates $S$ and $S^*$ can be majorized by a computable function and are therefore not hyper-simple. Therefore, a future work could be a construction of a hyper-simple predicate and proving the necessary properties of those hyper-simple predicates in order to formalize Post's problem with respect to truth-table reducibility. By the definition of hyper-simple predicates via majorizing functions of infinite predicates, it will be again essential to analyze the use of infinite predicates in this proof under our constructive perspective.

## 8.4 Turing Reductions and Post's Problem for Turing Reductions

Initially, Post raised the question whether there is a non-complete semidecidable but undecidable predicate with respect to the notion of Turing reducibility. As mentioned, he introduced further and stronger notions of reductions when trying to solve the problem. We saw how to formalize those different reductions in a synthetic setting and were able to use the advantages of this synthetic approach in order to formalize Post's problem with respect to many-one reducibility. In this chapter, we discussed furthermore the promising idea to follow the construction of hyper-simple predicates for a formalization of Post's problem for truth-table reductions. It would be of course of great interest to address also the original Post's problem with respect to Turing reducibility by for example formalizing the proofs of Muchnik [24] or Friedberg [17] including their independently invented priority method.

However, it is quite unclear whether such a formalization has a chance to succeed in a synthetic setting. Besides potential difficulties in the concrete proofs, it is already questionable how to define the notion of Turing reducibility synthetically. Traditional presentations define Turing reductions using so-called oracle machines. One can think of such oracles for a particular predicate as "black boxes" deciding the predicate and of an oracle machine as a Turing machine (or any other equivalent computational model) using such a "black box" oracle. A Turing reduction from a predicate $p$ to $q$ demands now the existence of an oracle machine deciding $p$, if the machine is used with an oracle for $q$. Formalizing this intuition of oracle machines deciding predicates could now result in the idea to define Turing reductions of a predicate $p$ to $q$ as the existence of a decider for $p$ when given a decider for $q$, i.e. as

$$\mathcal{D}q \to \mathcal{D}p.$$

However, this definition identifies oracle machines with deciders, which results in a small but significant problem. The oracle machine is only required to be a decider for $q$ when used with the oracle for $p$ and is not further specified for the usage with arbitrary other oracles. In particular, it must only compute a total function when used with the correct oracle but can compute in general a partial function. This characteristic also explains the power of Turing reductions in order to show problems undecidable: We can consult the given oracle as often as we like and are for the moment not restricted by a termination requirement. The above considered synthetic approach corresponds instead to the concept of total Turing reductions, that was traditionally shown to be actually equivalent to truth-table reductions by Nerode [26]. Intuitively, the termination – known already "a priori" – of machines computing a total function fixes the number of elements decided by the oracle during computation. Those elements can then be used as the sufficient query-list in a truth-table reduction.

The inconveniences and difficulties working with seemingly necessary partial functions in Coq's type theory were already discussed. Another way to address the formalization of Turing reductions and in a further step Post's problem would be to deviate from the synthetic approach and to work with concrete models of computation. One could try to add the concept of oracles to some versions of $\lambda$-calculi and use this as a concrete model as explored for instance by Forster and Smolka [13] to follow again the traditional proofs in this setting. Whether and to what extent this approach promises success, however, is unclear.

## 8.5 Precise Structure of Computability Degrees

We were able to find characterizations as well as multiple distinctions of different reducibility notions. This helped us to get a clearer view on the structure of computability degrees on the class of semidecidable but undecidable predicates. However, we contented ourselves with distinctions of in each case only two different classes by for example finding a many-one degree that could be distinguished into two classes of predicates with different one-one degrees. Similarly, we were able to distinguish two semidecidable but undecidable classes of $\mathtt{m}$-degrees, namely $\mathtt{m}$-complete and (not $\mathtt{m}$-complete) simple predicates.

However, one can analyze the structure of computability degrees even further, which will be rewarded by a very clear picture of the structure of those different classes. Dekker [10] showed for instance that the $\mathtt{m}$-degree of every simple predicate includes an infinite collection of 1-degrees. Furthermore, these 1-degrees can be ordered with respect to $\preceq_1$ with the order type of integers. This formulation means that there is neither a lower bound nor an upper bound for different 1-degrees inside the $\mathtt{m}$-degree of every simple predicate.

A similar result is due to Fischer [11], who showed that the degree of $\mathtt{tt}$-complete predicates includes an infinite collection of $\mathtt{m}$-degrees with order type of positive integers: For any given $\mathtt{m}$-degree consisting out of $\mathtt{tt}$-complete predicates, there is a class of $\mathtt{tt}$-complete predicates with a larger $\mathtt{m}$-degree. An analog proof shows the same result for $\mathtt{btt}$-complete instead of $\mathtt{tt}$-complete predicates.

It seems likely, that at least some of the mentioned further results regarding the structure of reducibility degrees can be deduced feasibly by using formalizations contributed by this thesis.

## 8.6 Necessary Axioms and Questions of Reverse Mathematics

Lastly it would be also interesting to take a closer look at the foundations of our work. We used a synthetic approach to our formalization and assumed therefore some selected axioms of synthetic computability theory. Even though we discussed that the proofs can not work without assuming any axioms, there could be room for improvement. Can some of the axioms be weakened or are there even other ways to proof our main results that allow to dispense on some of the axioms? We formalized and mechanized mainly the existing proofs from traditional presentations of computability theory. Maybe one can find new approaches to the proven results, working better in our constructive and synthetic setting.

Furthermore, one might want to focus on a more formal reasoning for assuming those axioms of synthetic computability theory. As mentioned, the axioms we assumed in the development should be derivable from Church's thesis: The enumerator of semidecidable predicates $\mathcal{W}$ and its semidecidability is already proven to follow from Church's thesis by Forster[12]. The $\mathsf{cns}$-operator and therefore the computability of program-indices deciding list-membership for a given list should be deducible from Church's thesis by implementing this operator in λ-calculus. The last axiom $\mathsf{SMN'}$ follows as discussed easily by the $S_n^m$-theorem. However, formalizing the $S_n^m$-theorem requires us to work with partial functions, such that it seems that one has to derive $\mathsf{SMN'}$ from Church's thesis for partial functions. This seems again to be possible by implementing the $S_n^m$-operator in λ-calculus.

Related to the search of as weak as possible axioms for our proofs are further questions in reverse mathematics regarding the topics of this thesis. We saw for example how crucial it is to choose the correct definition of infinite predicates. Some intended and lastly proven results were shown to be even contradictory for other conceivable notions of infinity due to our synthetic interpretation of functions, even though one understands the notions classically to be equivalent. Therefore, it is on the one hand of course interesting to analyze which further axioms are actually consistent with our work and their axioms. It is already known, that assuming both the full law of excluded middle together with choice axioms is not consistent with Church's thesis (cf. Troelstra and van Daalen [35]). It seems likely, that even without assuming Church's thesis in its full strength, classical and choice axioms become inconsistent to our development. On the other hand, one could search for implications between aspects of our work and for example choice axioms. Does for instance assuming certain notions of infinite predicates to be equivalent imply some kind of choice axiom? Or is only the existence of a simple predicate or the distinction of computability degrees enough to follow some weak axiom of synthetic computability theory? A positive answer to such a question could again justify the need of axioms for our work.

Chapter 9

# Conclusion

The motivation for this thesis was to formalize and mechanize reducibility theory from a constructive point of view. The aspired results were intended to especially include also negative results like distinctions of different reducibility degrees. In order to formalize advanced and more involved results, we chose a synthetic approach to our work which used – in order to address also negative results – synthetic computability axioms. This thesis is the first to explore such a constructive formalization and mechanization of synthetic computability theory based on the assumption of carefully chosen synthetic axioms.

At first, however, we started in an axiom-free constructive setting and focused besides basic facts of one-one, many-one, and truth-table reductions different characterizations of those notions and their degrees. We constructed for the proof of Myhill's isomorphism theorem a bijective isomorphism out of two injections, what was somewhat unexpected to work out without stronger choice or synthetic axioms. Furthermore we followed the traditional presentations to formalize adaptions of predicates to express both many-one and truth-table reductions in terms of one-one reductions:

$$p \preceq_m q \leftrightarrow p \times X \preceq_1 q \times Y$$
$$p \preceq_{tt} q \leftrightarrow \quad p^{tt} \preceq_1 q^{tt}$$

For $p : X \to \mathbb{P}$, $q : Y \to \mathbb{P}$, and certain embeddings for the underlying types $X$ and $Y$ or $X = Y = \mathbb{N}$.
The second equivalence requires $p$ to be stable.

This analysis gave us further insights in the structure and relationship of the different reduction preorders and their corresponding computability degrees. The formalization in type theory allowed us furthermore to analyze precisely, which properties of the underlying types are essential for these results.

We then added synthetic computability axioms such as the enumerator of semidecidable predicates $\mathcal{W}$ to our context and were therefore able to define and prove undecidable predicates. During chapter 6, the following four axioms were successively assumed:

- $\mathcal{W} : \mathbb{N} \to \mathbb{N} \to \mathbb{P}$ and its specification **ES** $:= \forall p : \mathbb{N} \to \mathbb{P}. \mathcal{S}p \leftrightarrow \exists c. \forall x. (\mathcal{W}cx \leftrightarrow px)$
- $\mathcal{S}\mathcal{W}$
- **SMN'** $:= \forall f. \exists k. \forall cx. \mathcal{W}(kc)x \leftrightarrow \mathcal{W}c(fx)$
- **LIST− ID** $:= \forall L. \Sigma c_L. \forall x. \mathcal{W}cx \leftrightarrow x \in L$

In order to follow the traditional presentations of Post's problem for many-one reductions, we formalized the notion of simple predicates and constructed with $S$ and $S^*$ two of those simple predicates. This allowed us not only – as stated by Post's problem for many-one reductions – to find an intermediate $m$-degree between decidable and $m$-complete predicates, but yielded also a distinction of one-one and many-one as well as many-one and truth-table reducibility. All in all, we formalized for the class of semidecidable but undecidable predicates the following inclusion hierarchy of various reducibility notions that transports to their reducibility degree:

$$\begin{array}{ccccccc} & & \preceq_1 & \overset{(1)}{\subsetneq} & \preceq_m & \overset{(2)}{\subsetneq} & \preceq_{tt} \\ \equiv & \overset{(M)}{=} & \equiv_1 & \overset{(1)}{\subsetneq} & \equiv_m & \overset{(2)}{\subsetneq} & \equiv_{tt} \end{array}$$

(M): Axiom-free proof by Myhill's isomorphism theorem
(1): Inclusion axiom-free; strictness by assuming $\mathcal{W}$, **ES**, and $\mathcal{S}\mathcal{W}$
(2): Inclusion axiom-free; strictness by assuming $\mathcal{W}$, **ES**, $\mathcal{S}\mathcal{W}$, **SMN'**, and **LIST− ID**

The synthetic approach led us work out which computability properties are actually necessary for the particular proofs and revealed that we can dispense on a strong universal machine that is ubiquitous in all traditional presentations.

Besides the formalization in a synthetic setting and the mechanization in Coq of the above summarized computability theory, a further contribution of the thesis is to constructivize the already existent classical proofs.

Thereby, infinite predicates were found to be particularly interesting. Constructively, the different notions of infinite predicates turned out to differ significantly in their strength and properties such that a precise study of the role of infinite predicates for the aspired results was crucial. This analysis is indispensable for all constructive formalization, but in combination with our synthetic approach it was given another tricky aspect: listing infinite elements of a predicate is synthetically only possible by computing those elements. We had therefore not only to find the working definition of infinite predicates but to also come up with the right infinite criterion by using the notion of weak existence that avoids a computation.

Also apart from infinite predicates, we could follow the classical proofs by inserting double negations at exactly selected points. However, these adaptions were only temporarily inside the proofs and did not affected the final results: The proven theorems are faithful formalizations of traditional computability theory and do not state for instance only the weak existence of a simple predicate[1].

All of this work, the formalization, the mechanization, and the constructivization of reducibility theory was only made possible by our synthetic approach. First of all and most obviously, it would have been quite hard to implement all the constructed functions like for instance Myhill's isomorphism or also $\psi$ used to define the simple predicate $S$ formally in a concrete computational model. Traditional presentations simply leave out this inconvenient but the in their setting necessary construction of e.g. Turing machines. Working synthetically in Coq's programming language provided not only a pleasant way to define occurring functions, but guaranteed also immediately the computability of those functions. In contrast to traditional work, all functions are in our synthetic setting formally justified to be indeed computable functions.

Furthermore, synthetic computability theory revealed by minimizing technical distractions a clear view on the formal proof arguments: While the mechanization of computations in concrete models is often a hurdle, the synthetic work in Coq simplified the proofs dramatically. This helped us to gain deeper insights into the "purely" mathematical aspects necessary for the formalization. This was for example well to explore in the whole chapter 3, in the precise termination argumentation for the trace function in Myhill's isomorphism theorem, or in the co-infinity proof of the simple predicate $S$.

The elaboration of the constructive formalization would be disproportionately more difficult when working in a traditional setting. Even if the finally presented proofs became not too complex but remained manageable to understand, we meanwhile dealt with even synthetically already complex proofs that were only then be simplified to achieve the final presentation. These more complex intermediate steps would have blown up an explicit computational models to a point of unusability but could be reached working synthetically due to the explored and discussed inherent benefits.

However, it might now be possible to mirror the formalizations and mechanizations of reducibility theory in a well fitting computational model as near as possi-

[1]Except to the reduction characterization in Chapter 4.3, where we assumed a predicate to be stable.

ble to Coq like λ-calculus. Based on the clean synthetic presentations of the formal proofs in this thesis, one could bluntly copy the development and implement all constructed functions as well as the assumed synthetic axioms in a mechanized model. Partly, this should even be automatable by using frameworks for this translation. Mathematically more interesting would be however the question, whether there is a short cut that avoids this complete translation by for instance restating some of the theorems in some way that allows translating the formalized reducibility theory to an underlying model of computation by only local adaptions.

These and other questions about the foundations of synthetic computability theory but also the analysis, formalization, and mechanization of further reducibility theory results offer as discussed much room for future work. The synthetic approach is optimally suited for this purpose and, as explored by this thesis, can also address traditional results that use certain properties of their underlying computational model by adding abstract synthetic computability axioms. This enables the opportunity to follow traditional computability theory over long distances and to study its theory from a different perspective under several new aspects.

# Bibliography

[1] Andrea Asperti and Wilmer Ricciotti. Formalizing turing machines. In Luke Ong and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, pages 1–25, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32621-9.

[2] Andrea Asperti and Wilmer Ricciotti. A formalization of multi-tape turing machines. *Theoretical Computer Science*, 603, 07 2015.

[3] Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

[4] Michael J. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Current Topics in Microbiology and Immmunology. Springer-Verlag, 1985. ISBN 9780387121734.

[5] Douglas S. Bridges, Fred Richman, John W.S. Cassels, London Mathematical Society, and Nigel J. Hitchin. *Varieties of Constructive Mathematics*. Lecture note series. Cambridge University Press, 1987. ISBN 9780521318020.

[6] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932. ISSN 0003486X.

[7] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936. ISSN 00029327, 10806377.

[8] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory*. Cambridge University Press, 1980. ISBN 9780521294652.

[9] Arthur Azevedo de Amorim and Rob Blanco. URL `https://github.com/arthuraa/extructures/blob/master/theories/fmap.v`. At this point in time, there is no associated paper yet.

[10] James C. E. Dekker. A theorem on hypersimple sets. *Proceedings of the American Mathematical Society*, 5:791–796, 2001.

[11] Patrick C. Fischer. Theory of provable recurive functions. 1962. Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge, Mass.

[12] Yannick Forster. Church's thesis and related axioms in coq's type theory. 2020.

[13] Yannick Forster and Gert Smolka. Call-by-value lambda calculus as a model of computation in Coq. *Journal of Automated Reasoning*, 63(2):393–413, 2019.

[14] Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

[15] Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of turing machines in coq. In *9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20–21, 2020*, New York, NY, USA, 2020. ACM.

[16] Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq Library of Undecidable Problems. *The Sixth International Workshop on Coq for Programming Languages*, 2020.

[17] Richard M. Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability. *Proceedings of the National Academy of Sciences*, 43(2): 236–238, 1957. ISSN 0027-8424. doi: 10.1073/pnas.43.2.236.

[18] Harvey Friedman. Some Systems of Second Order Arithmetic and their Use. *Proceedings of the International Congress of Mathematicians*, 1:235–242, 1975.

[19] Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. Finite sets in homotopy type theory. pages 201–214, 01 2018. doi: 10.1145/3167085.

[20] Kurt Gödel. On undecidable propositions of formal mathematical systems. Lecture notes by Stephen C. Kleene and J. Barkely Rosser, Princeton University. Reprinted in (Gödel, 1986, 338–371).

[21] Martin E. Hyland. The effective topos. In Anne S. Troelstra and Dirk van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 165 – 216. Elsevier, 1982.

[22] Hajime Ishihara. Reverse mathematics in bishop's constructive mathematics. *Philosophia Scientiae*, 6:43–59, 09 2006. doi: 10.4000/philosophiascientiae.406.

[23] Stephen C. Kleene and John B. Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, 36(3):630–636, 1935. ISSN 0003486X.

[24] Albert A. Muchnik. On the unsolvability of the problem of reducibility in the

theory of algorithms. *Doklady Akademii Nauk SSSR (N.S.)*, 108:194–197, 1956. In Russian.

[25] John Myhill. Creative sets. *Mathematical Logic Quarterly*, 1(2):97–108, 1955. doi: 10.1002/malq.19550010205.

[26] Anil Nerode. General topology and partial recursive functions. *Summaries of talks presented at the Summer Institute for Symbolic Logic, Cornell University*, pages 247–251, 1957.

[27] Michael Norrish. Mechanising lambda-calculus using a classical first order theory of terms with permutations. *Higher-Order and Symbolic Computation*, 19:169–195, 09 2006. doi: 10.1007/s10990-006-8745-7.

[28] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bull. Amer. Math. Soc.*, 50(5):284–316, 05 1944.

[29] Pierre Pradic and Chad Brown. Cantor-bernstein implies excluded middle. *ArXiv*, abs/1904.09193, 2019.

[30] Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48 (3):297–803, 1983.

[31] Hartley Rogers. *Theory of Recursive Functions and Effective Computability*. Higher Mathematics Series. McGraw-Hill, 1967.

[32] Robert I. Soare. *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer Berlin Heidelberg, 1999. ISBN 9783540152996.

[33] Andrew Swan and Taichi Uemura. On church's thesis in cubical assemblies. *ArXiv*, abs/1905.03014, 2019.

[34] The Coq Development Team. The coq proof assistant, version 8.11.0. January 2020. URL `https://doi.org/10.5281/zenodo.3744225`.

[35] Anne S. Troelstra and Dirk van Dalen. Constructivism in Mathematics. 1988. Amsterdam, Netherlands.

[36] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 01 1936. ISSN 0024-6115. doi: 10.1112/plms/s2-42.1.230.

[37] Alan M. Turing. Systems of logic based on ordinals†. *Proceedings of the London Mathematical Society*, s2-45(1):161–228, 1939. doi: 10.1112/plms/s2-45.1.161.

[38] Jian Xu, Xingyuan Zhang, and Christian Urban. Mechanising turing machines and computability theory in isabelle/hol. pages 147–162, 2013.