SAARLAND UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

BACHELOR'S THESIS

POST'S PROBLEM AND THE PRIORITY METHOD IN SYNTHETIC COMPUTABILITY

Author
Haoyi Zeng

Supervisor
Prof. Dr. Gert Smolka

Advisors
Dr. Yannick Forster
Dr. Dominik Kirst

Reviewers
Prof. Dr. Gert Smolka
Dr. Dominik Kirst

Submitted: 20th August 2024

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.

Erklärung

Ich erkläre hiermit, dass die vorliegende Arbeit mit der elektronischen Version übereinstimmt.

Statement

I hereby confirm the congruence of the contents of the printed data and the electronic version of the thesis.

Saarbrücken, 20th August, 2024 Haoyi Zeng

Life begins after the bachelor's thesis.

Prof. Gert Smolka

Abstract

Post's problem, posed by Post in 1944, has been a crucial question driving research in computability theory for more than a decade. It asks whether semi-decidable yet undecidable predicates exist that cannot be proven undecidable via Turing reduction from the halting problem. This problem remained open until Friedberg and Muchnik achieved a breakthrough independently in 1956/57 with a positive solution by developing the priority method.

In this thesis, we mechanise a solution to Post's problem within synthetic computability using the proof assistant Coq. The synthetic approach to computability is grounded in constructive mathematics, where every function can axiomatically be considered computable. Therefore, it eliminates the need to prove that a function is computable within a model of computation, such as Turing machines or the λ-calculus.

Using the priority method, our mechanisation follows Lerman and Soare's solution to Post's problem by constructing low simple predicates. This approach is more accessible compared to Friedberg and Muchnik's construction. To achieve this, we define step-indexing and use functions for oracle machines based on Forster, Kirst, and Mück's definition of synthetic oracle computability. Furthermore, we mechanise Nemoto's result that the limited principle of omniscience is sufficient to show the existence of low simple predicates.

Let me show you some magic.

Dominik Kirst

Acknowledgements

First, my deepest gratitude goes to Prof. Smolka for offering me a thesis at his chair and his passionate lectures that made me decide to study computer science at Saarland. Equal thanks are due to my two excellent advisors, Yannick and Dominik, without whom I can't imagine how I would have completed this project. In particular, I benefited from Yannick's intuition, both for synthetic computability and navigating the way to the hotel. Dominik, I am immensely grateful for showing me the magic in your office three years ago - Curry-Howard correspondence; although I was not yet a student then, this guided me towards constructive type theory.

I am deeply grateful for the support and help I received from my friends while writing this thesis. Over the past three years in Saarland, they have been my classmates, colleagues, students, tutors, and advisors, and their company has been a significant part of my bachelor's journey. Vielen Danke! Special thanks to Aaron, Janis, Fabian, Johannes, Nils, Niklas, Benjamin and Simon.

I extend my gratitude to Prof. Derek Dreyer and Prof. Jan Reineke, who provided valuable feedback on my thesis. Additionally, I am grateful for their generous funding, which allowed me to present this project at the TYPES conference in Copenhagen.

I'd like to thank my family's endless support, as well as the support from the cat support team, Baloo and Virgo. Also, thanks to Dize's assistance, which allowed me to submit this thesis from Lausanne.

Finally, I give heartfelt thanks to Zheyuan, without whom I would not be where I am today.

Contents

Chapter 1

Introduction

"As a result we are left completely on the fence as to whether there exists a recursively enumerable set of positive integers of absolutely lower degree of unsolvability than the complete set K*, ..."*

EMIL L. POST

In this thesis, we mechanise a solution to Post's problem [52] within synthetic computability using the Coq proof assistant. Based on Forster, Kirst, and Mück's definition of synthetic oracle computability [18, 21], we introduce the definition of step-indexing and use functions for oracle machines, the priority method, and limit computability in synthetic computability. A synthetic solution to Post's problem is then provided by constructing low simple predicates, following Lerman and Soare's approach [44].

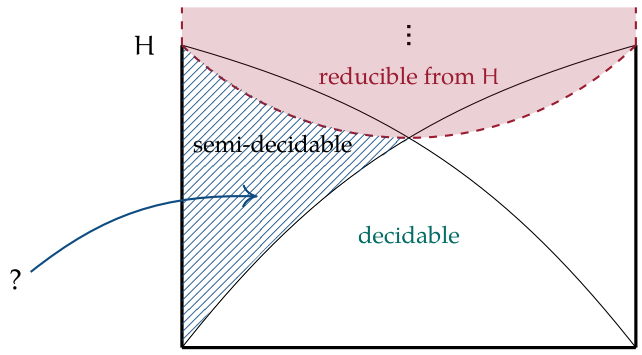

Figure 1.1: Post's problem is whether the shaded area is inhabited.

In this introductory section, we will informally outline the key concepts involved in our discussion. The central topic is Post's problem, which was posed by Emil Post in 1944 [52]. This problem is about whether semi-decidable predicates exist that cannot be proved undecidable by Turing reduction from the halting problem H. As shown in Figure 1.1, the existence of solutions to Post's problem would imply a rich order structure of predicates up to Turing reduction. This problem has been a long-standing open problem driving research in computability theory for over a decade, until a breakthrough came with a new method by Friedberg and Muchnik independently in 1956 and 1957 [27, 46]. With this new method, they proved what is now known as the Friedberg-Muchnik theorem, which implies a positive solution to Post's problem. The method they introduced, called the priority method, has since become a cornerstone in computability theory, essential for exploring and understanding the degrees of undecidability [42, 44, 59]. Post's problem and the use of the priority method are discussed in virtually every textbook on advanced computability theory (e.g. [62, 54, 61, 49]).

From the perspective of machine-checked proofs, the interactive theorem proving community has successfully mechanised cutting-edge mathematics in several proof assistants. However, as will be discussed later, mechanising computability theory presents unique challenges.

Synthetic Computability In the mechanisation of computability theory, a main intricacy is the use of models of computation (e.g. Turing machines, λ-calculus) for formal proofs [23, 25, 40, 26, 24]. For instance, the usual (also known as analytic) approach defines decidability of a predicate P as

$$\exists f : \mathbb{N} \to \mathbb{B}.\ f \text{ is computable} \land \forall n.\ n \in P \leftrightarrow f(n) = \mathsf{true}.$$

Based on this definition, proving decidability requires showing that the Boolean function f is computable within a model of computation. In a mechanised setting, this requires a tedious amount of proofs about uninteresting details that usually stay invisible on paper. This slowed progress immensely, and ultimately lead towards the development of synthetic computability theory, which avoids much of the tedium. In order to avoid these difficulties arising from any specific model, synthetic computability is proposed as a solution, pioneered by Richman, Bridges, and Bauer [6, 53, 3], which exploits constructive mathematics as its foundation. In a synthetic approach to computability theory, *every* function is considered computable axiomatically. It is possible since only classical logic introduces non-computable functions. Thus, the decidability of a predicate P is now defined as $\exists f : \mathbb{N} \to \mathbb{B}.\ \forall n.\ n \in P \leftrightarrow f(n) = \mathsf{true}$, which eliminates the need to prove that f is computable within a model. With this, one can focus on the mathematical essence behind computability theory and no longer needs to rely on any model of computation. This approach has proven successful: Many topics have been developed synthetically [18, 16, 21, 19, 34].

While synthetic computability significantly reduces efforts of mechanisation by treating computability as a primitive concept. Advanced computability theory involves concepts like oracle machines and Turing reductions, which necessitate an understanding of *oracle computability*. This concept, however, is not straightforward in the context of synthetic computability. The first definition is proposed by Bauer [4], with subsequent variations by Forster, Kirst, and Mück. [21, 20], and work on oracle modalities by Swan [64].

Constructive Reverse Mathematics Through the calculus of inductive constructions (CIC) [12, 11, 50], the underlying theory behind the Coq proof assistant, synthetic computability becomes usable in Coq [16, 22]. This is achieved by assuming axioms common in synthetic computability, such as Church's thesis (CT) and its variants [38, 17, 15], which internalise the fact that all functions are computable. Moreover, since CIC is a constructive system where the law of excluded middle (LEM) remains consistent even when assuming axioms (e.g. CT) for synthetic computability [15], it is natural to study the equivalence between different strengths of classical logic axioms, such as LEM, and theorems in computability theory. In full generality, this line of research is known as constructive reverse mathematics [32, 14]. For instance, Post's theorem[1] is equivalent to Markov's principle [65, 22], a principle that is strictly weaker than LEM. Further, early work suggests there might be rich connections between various theorems in computability and respective classical axioms expressible in constructive mathematics [16, 18, 21, 19, 34, 48]; the study of this connection is left as further work.

Goals of this Thesis In 2021, Andrej Bauer posed the challenge [4]:

"give a synthetic proof of Friedberg-Muchnik theorem".

The goal of this thesis is to solve this challenge. Due to the historical importance of Post's problem, successfully mechanising a solution within synthetic computability is considered a milestone. Furthermore, Achieving this could further set the stage for a later investigation of Post's problem from the perspective of constructive reverse mathematics.

Our Approach Historically, Friedberg and Muchnik solved Post's problem by first introducing the priority method to construct two semi-decidable predicates that are not Turing-reducible to each other, implying a predicate solving Post's problem

[1]Post's theorem, which states that a predicate is decidable if and only if it and its complement are both semi-decidable, is entirely different from Post's problem.

exists [27, 46]. Later, as a byproduct of Lerman and Soare's work [44], low simple predicates, which are semi-decidable yet undecidable predicates that are not Turing reducible from the halting problem, are constructed. This is considered the easiest solution to Post's problem [62, 29]. Based on this approach, Nemoto proves that the limited principle of omniscience is sufficient to show the existence of low simple predicates within an analytic framework [48].

To mechanise a solution to Post's problem, we follow Lehmann and Soare's construction alongside Nemoto's proof. We introduce step-indexing and use functions for oracle computability based on the definition of synthetic oracle computability by Forster, Kirst, and Mück. Additionally, we incorporate Forster and Jahn's synthetic notion of simple predicates [18].

Since Turing reductions are defined by oracle computability, constructing specific Turing reductions is rather complicated due to the difficulty of using oracle computability. Therefore, in the construction of low simple predicates, we employ the notion of limit computability, following Soare's approach [62]. This concept, independently developed by Gold and Shoenfield [28, 57], is defined by computable functions, and can demonstrate that a predicate is Turing reducible to the halting problem without referring to a concrete reduction.

1.1 Contributions

This thesis makes the following contributions:

- We give the first synthetic solution to Post's problem following Lerman and Soare's construction [44], which has been mentioned as challenging future work in several publications [18, 21, 20]. In particular, we mechanise the existence of low simple predicates in synthetic computability.
- We introduce a general framework to formalise arguments based on the priority method in synthetic computability. Within this framework, the construction of low simple predicates is proven modularly.
- We define step-indexing and use functions for oracle machines (see Definitions 3.22 and 3.30) built upon the definition of oracle computability by Forster, Kirst, and Mück [20, 21]. These notions are central to the construction of solutions for Post's problem.
- We introduce a synthetic notion of limit computability to enable the construction of low simple predicates, along with a synthetic proof of the limit lemma, based on the synthetic arithmetical hierarchy and Post's theorem [21].
- We adapt Nemoto's analytic proof [48] that the limited principle of omni-

science is sufficient to show the existence of low simple predicates in synthetic computability.

- All the results have been mechanised in the proof assistant Coq and each theorem and definition has links to the web version of the Coq code.

A generated Coq documentation is available at:

`https://ps.uni-saarland.de/~zeng/bachelor/coqdoc/thesis`

1.2 Outline

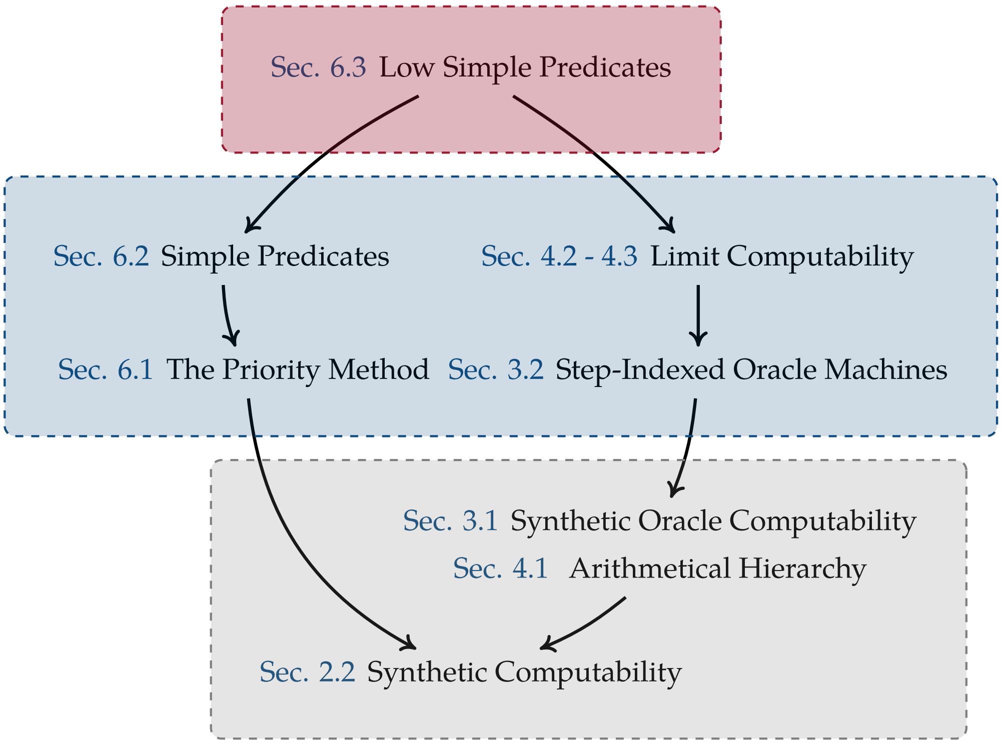
Sec. 6.3 Low Simple Predicates
Sec. 6.2 Simple Predicates
Sec. 4.2 - 4.3 Limit Computability
Sec. 6.1 The Priority Method
Sec. 3.2 Step-Indexed Oracle Machines
Sec. 3.1 Synthetic Oracle Computability
Sec. 4.1 Arithmetical Hierarchy
Sec. 2.2 Synthetic Computability

Figure 1.2: A dependency graph for the contents of this thesis.

The structure of this thesis follows a bottom-up approach in Chapter 2 to 4, until reaching the point where Post's problem can be stated in terms of synthetic computability. Subsequently, the construction and correctness proof of the solution to Post's problem are presented in a top-down manner in Chapter 6.

The readers may find Figure 1.2 useful, as it shows the dependencies of this project and the foundation upon which it is built. The grey part represents the results that already existed before [21, 20], serving as the starting point of this project. This part is not the author's contribution; the author's contributions come in the blue and red parts.

The blue part includes the technical lemmas and the necessary constructions that make the discussion of Post's problem in synthetic computability possible.

If the readers are only interested in the solution to Post's problem, the red part illustrates the construction based on the abstract properties and lemmas that have been established in the blue part.

Here are the contents of each chapter:

- In Chapter 2, we introduce the basic concepts of the calculus of inductive constructions (CIC) and synthetic computability.
- In Chapter 3, we review the concepts of oracle computability within synthetic computability, then construct and verify the step-indexed oracle machines and use functions.
- In Chapter 4, we introduce a synthetic notion of limit computability, and prove the limit lemma.
- In Chapter 5, we review the background of Post's problem and the Friedberg-Muchnik theorem, as well as Lerman and Soare's construction.
- In Chapter 6, we first establish the priority method in synthetic computability, and use this does construct low simple predicates, thereby solving Post's problem.
- In Chapter 7, we conclude the thesis and discuss future work.

Chapter 2

Preliminaries

In this thesis, we are working on the calculus of inductive constructions (CIC), the theory implemented by the proof assistant Coq [12, 11, 50]. In this chapter, we first briefly introduce basic notions of CIC and then discuss basic definitions of synthetic computability. Finally, we present classical axioms related to the results of this thesis.

2.1 The Calculus of Inductive Constructions

In CIC, a term t that has type T is denoted by $t : T$. For example, $0 : \mathbb{N}$ means that 0 has type $\mathbb{N}$. Inductive propositions and types can be defined in the propositional universe $\mathbb{P}\mathsf{rop}$ or in the type universe $\mathbb{T}\mathsf{ype}$. However, eliminating a proposition for constructing a type is not always allowed and must be based on specific constraints. We start with the usual definitions of inductive types and some notations.

2.1.1 Basic Notations

These are some basic inductive types that will be used in this thesis:

- Unit: $\mathbb{1} : \mathbb{T}\mathsf{ype} ::= \star : \mathbb{1}$
- Booleans: $\mathbb{B} : \mathbb{T}\mathsf{ype} ::= \mathsf{true} : \mathbb{B} \mid \mathsf{false} : \mathbb{B}$
- Natural numbers: $\mathbb{N} : \mathbb{T}\mathsf{ype} ::= \mathsf{O} : \mathbb{N} \mid \mathsf{S} : \mathbb{N} \to \mathbb{N}$
- Option types: $T^? : \mathbb{T}\mathsf{ype} ::= \ulcorner\_\urcorner : T \to T^? \mid \mathsf{none} : T^?$
- Product types: $X \times Y : \mathbb{T}\mathsf{ype} ::= (\_\,,\,\_) : X \to Y \to X \times Y$
- Lists: $T^* : \mathbb{T}\mathsf{ype} ::= [\,] : T^* \mid \_ :: \_ : T \to T^* \to T^*$

- Sum types: $X + Y : \mathbb{T}\mathrm{ype} ::= \mathsf{inl} : X \to X + Y \mid \mathsf{inr} : Y \to X + Y$

In CIC, two similar yet distinct notions exist: the sigma type $\Sigma\, x.T\ x : \mathbb{T}\mathrm{ype}$ for $T : X \to \mathbb{T}\mathrm{ype}$, and the existential type $\exists x.P\ x : \mathbb{P}\mathrm{rop}$ for $P : X \to \mathbb{P}\mathrm{rop}$. While both types present the existence of a witness x satisfying $T\ x$ or $P\ x$, respectively. They reside in different universes – $\mathbb{T}\mathrm{ype}$ and $\mathbb{P}\mathrm{rop}$, respectively. The sigma type can project the witness x to construct functions via projection π, whereas the elimination of the existential type can only be used in proofs. For instance, if we have

$$h := \forall n : \mathbb{N}.\ \Sigma\ m : \mathbb{N}.\ P\ n\ m,$$

we can define a function $f\ n := \pi(h\ n)$ and show that $\forall n.\ P\ n\ (f\ n)$. However, given a proof of $\forall m.\ \exists n.\ P\ n\ m$. For any n, an m obtained from this proof, such that $P\ n\ m$ holds, can only be used for proofs, specifically for constructing terms of type $\mathbb{P}\mathrm{rop}$.

By default, the capital letters $X\ Y\ Z : \mathbb{T}\mathrm{ype}$ are used to arbitrary types, $P\ Q : \mathbb{P}\mathrm{rop}$ are used to arbitrary propositions, while $p\ q : X \to \mathbb{P}\mathrm{rop}$ generally express arbitrary predicates over X.

When the symbol $x \in A$ is used, it is a recursively defined inclusion if A is a list, and an alias of $A\ x$ if A is a predicate. Similarly, the symbol $A \subseteq B$ is an abbreviation of $\forall x.\ A\ x \to B\ x$ when A and B are predicates, and A is a sublist of B when they are lists. The notation $l_1 \sqsubseteq l_2$ is used to denote that l_1 is a prefix of l_2.

The complement of a predicate $p : X \to \mathbb{P}\mathrm{rop}$ is defined as: $\overline{p}\ x \ := \ \neg p\ x$.

For predicates $p, q : X \to \mathbb{P}\mathrm{rop}$, we define the equivalence between them up to a list l as: $p \equiv_l q \ := \ \forall x.\ x \in l \to p\ x \leftrightarrow q\ x$.

Not only with a list l, we also use a natural number n to denote the equivalence up to n if the type X is $\mathbb{N}$: $p \equiv_n q \ := \ \forall x.\ x < n \to p\ x \leftrightarrow q\ x$.

This notation is also used for both predicates and boolean functions, e.g., for $f : \mathbb{N} \to \mathsf{bool}$ and $p : \mathbb{N} \to \mathbb{P}\mathrm{rop}$:

$$f \equiv_n p : \forall x.\ x < n \to f(x) = \mathsf{true} \leftrightarrow p\ x$$

Since sum types is often used in our discussion to distinguish between different results, we conventionally define $\mathsf{ask} \ := \ \mathsf{inl}$ and $\mathsf{out} \ := \ \mathsf{inr}$.

Definition 2.1 (Logical Decidability) *A predicate* $p : X \to \mathbb{P}\mathrm{rop}$ *is logical decidable if:*

$$\forall x.\ p\ x \lor \neg\ p\ x$$

Definition 2.2 (Characteristic Relation) *The characteristic relation* $\hat{p} : X \to \mathbb{B} \to \mathbb{P}\mathrm{rop}$ *of a predicate* $p : X \to \mathbb{P}\mathrm{rop}$ *is defined by:*

$$\hat{p} \coloneqq \lambda x\, b.\ \begin{cases} p\ x & \textit{if } b = \mathsf{true} \\ \neg\, p\ x & \textit{if } b = \mathsf{false} \end{cases}$$

2.1.2 Partial Functions

In CIC, all functions are total functions. However, one can define an interface of partial function by its properties and then implement it with total functions. The interfaces of partial functions include the following operations:

- $\mathsf{part}\ A : \mathbb{T}\mathrm{ype}$
- $\downarrow\ : \mathsf{part}\, A \to A \to \mathbb{P}\mathrm{rop}$
- $\mathsf{ret}\ : A \to \mathsf{part}\ A$
- $\gg\!= : \mathsf{part}\, A \to (A \to \mathsf{part}\, B) \to \mathsf{part}\, B$
- $\mathsf{seval}\ : \mathsf{part}\, A \to \mathbb{N} \to A^?$

Those operations satisfy the following properties:

- $x \downarrow a_1 \to x \downarrow a_2 \to a_1 = a_2$
- $\mathsf{ret}\, a \downarrow a$
- $x \gg\!= f \downarrow b \leftrightarrow \exists a.\ x \downarrow a \land f\ a \downarrow b$
- $x \downarrow a \leftrightarrow \exists n.\ \mathsf{seval}\ x\ n = \ulcorner a \urcorner$
- $\mathsf{seval}\ x\ n = \ulcorner a \urcorner \to \forall m \geqslant n.\ \mathsf{seval}\ x\ m = \ulcorner a \urcorner$

Based on this interface, we use the notation $X \rightharpoonup Y \coloneqq X \to \mathsf{part}\, Y$ to make it look like a function. One possible implementation of partial functions is to define $\mathsf{part}\, A$ as a step-indexed option type $\mathbb{N} \to A^?$, we refer to Forster's thesis [16] for me details. The notation $x \downarrow_n a$ denotes that the evaluation of x is terminated on a within n steps $\mathsf{seval}\ x\, n = \ulcorner a \urcorner$.

2.2 Synthetic Computability

In a synthetic approach to computability theory [7, 53, 3], every function is considered computable. This allows us to define notions such as (semi-) decidability and many-one reductions without referring to a specific model of computation.

2.2.1 Basic Definitions

We summarise basic definitions in synthetic computability and refer to Forster's thesis [16] for detailed discussions.

Definition 2.3 *For any predicate* $p : X \to \mathbb{P}\mathrm{rop}$*, we define the following properties:*

- *Decidability:* $\exists f : X \to \mathbb{B}.\ \forall x.\ p\ x \leftrightarrow f(x) = \mathsf{true}$*, denoted by* $\mathcal{D}(p)$*. We call* f *a decider of* p*.*
- *Semi-decidability:* $\exists f : X \to \mathbb{N} \to \mathbb{B}.\ \forall x.\ p\ x \leftrightarrow \exists n.\ f(x, n) = \mathsf{true}$*, denoted by* $\mathcal{S}(p)$*. We call* f *a semi-decider of* p*.*

For any type X*, we define the following properties:*

- *Discreteness:* $\forall x\ y : X.\ (x = y) + (x \neq y)$*.*
- *Enumerability:* $\exists f : \mathbb{N} \to X.\ \forall x.\ \exists n.\ f(n) = x$*.*
- *Data Type:* X *is both discrete and enumerable.*

There is an alternative definition of semi-decidability via partial functions:

$$\mathcal{S}(p) \coloneqq \exists f : X \rightharpoonup \mathbb{1}.\ \forall x.\ p\ x \leftrightarrow f\ x \downarrow \star$$

Definition 2.4 (Many-One Reductions) *A predicate* $p : X \to \mathbb{P}\mathrm{rop}$ *is many-one reducible to* $q : Y \to \mathbb{P}\mathrm{rop}$ *if there exists a function* $f : X \to Y$ *that maps instances of* p *to instances of* q*, formally:*

$$q \preceq_m p \coloneqq \forall x.\ q\ x \leftrightarrow p\ (f\ x)$$

Fact 2.5 *For any predicate* p*:*

- *If* p *is decidable, then* $\overline{p}$ *is decidable.*
- *For any predicate* q*, if* $q \preceq_m p$*, then* q *is decidable if* p *is decidable and* q *is semi-decidable if* p *is semi-decidable.*

2.2.2 Church's Thesis

In CIC, every definable function is intuitively computable. This fact can be internalised by assuming axioms like Church's thesis (CT), which states a step-indexed universal function that computes all functions $\mathbb{N} \to \mathbb{N}$. Since CT maintains consistency with LEM in CIC [15], this framework allows us to analyse which classical axioms are sufficient or necessary to demonstrate specific results.

In this thesis, we assume the axiom of enumerability of partial functions (EPF), which is equivalent to CT [17]. This axiom states that there exists a partial universal function that enumerates all partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$.

Definition 2.6 (Enumerability of Partial Functions (EPF)) *There is an enumerator of partial functions $\theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$, such that for any partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, there exists a code $c : \mathbb{N}$ such that θ_c and f coincide:*

$$\Sigma\, \theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N}).\ \forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\ \exists c : \mathbb{N}.\ \forall x\, v.\ \theta_c\ x \downarrow v \leftrightarrow f\ x \downarrow v$$

Given θ, we define $\mathcal{W}_c\ x\ :=\ \theta_c \downarrow x$ as the c-th semi-decidable predicate. The notation $\mathcal{W}_c[n]$ is used to denote the c-th semi-decidable predicate at step n. i.e., it can be defined as:

$$\mathcal{W}_c[n]\ x\ :=\ \theta_c \downarrow_n x$$

Therefore, we can show that $\mathcal{W}_c\ x \leftrightarrow \exists n.\ \mathcal{W}_c[n]\ x$ and the predicate $\mathcal{W}_c[n]$ is decidable.

Definition 2.7 (The Halting Problem) *The synthetic self halting problem is the predicate that determines whether a given partial function halts on its corresponding code:*

$$\mathsf{H}\ x\ :=\ \exists v.\ \theta_x\ x \downarrow v$$

Lemma 2.8 *The halting problem is semi-decidable but not decidable.*

2.2.3 Classical Logic

The law of excluded middle (LEM) is a classical axiom, which states that a proposition is either true or false. It is not provable in CIC without any axioms. Assuming LEM allows for classical reasoning.

Definition 2.9 $\mathsf{LEM}\ :=\ \forall p : \mathbb{P}\mathrm{rop}.\ p \vee \neg\, p$

For some results in this thesis, weaker classical principles, such as Markov's principle (MP) or the limited principle of omniscience (LPO) are enough. LPO and MP are common used in reverse construction mathematics. They are strictly weaker than LEM, but still allow for some classical reasoning. Neither of them is provable in CIC.

Definition 2.10 $\mathsf{MP}\ :=\ \forall f : \mathbb{N} \to \mathbb{B}.\ \neg\neg(\exists n.\ f\ n = \mathsf{true}) \to \exists n.\ f\ n = \mathsf{true}$

Definition 2.11 $\mathsf{LPO}\ :=\ \forall f : \mathbb{N} \to \mathbb{B}.\ (\exists n.\ f\ n = \mathsf{true}) \vee (\forall n.\ f\ n = \mathsf{false})$

Lemma 2.12 *Assuming* LPO, *the halting problem is logical decidable.*

Proof Since the halting problem H is semi-decidable, there is a semi-decider $f : X \to \mathbb{N} \to \mathbb{B}$. For any given x, we apply LPO on $f\ x$. If $\exists n.\ f\ x\ n = \mathsf{true}$, then $\mathsf{H}\ x$ holds. If $\forall n.\ f\ x\ n = \mathsf{false}$, then $\neg\mathsf{H}\ x$ holds, otherwise, there is a number n such that $f\ x\ n = \mathsf{true}$, which contradicts the assumption. □

Chapter 3

Oracle Computability

Oracle computability is a concept initially introduced in Turing's PhD thesis [66] and later developed by Post in his seminal paper [52], used to describe a very general reduction between two predicates, now called Turing reduction.

Turing reductions are more general than many-one reduction (see Definition 2.4) we discussed earlier. Specifically, in many-one reduction, a computable function maps an instance of a predicate p to another predicate q, thereby allowing the computable function that decides q also to decide p. It turns out that the answer of q is invisible to the reduction, and therefore, the answer of q cannot affect the computation of reduction. However, Turing reducibility allow for different computations based on the answers.

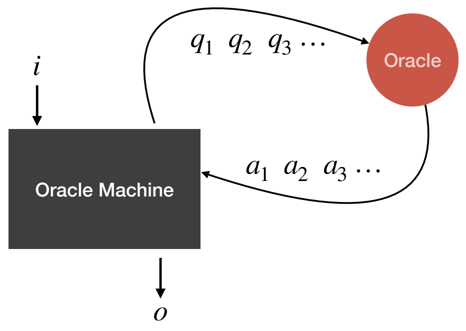

Figure 3.1: Oracle Machine

Imagine an oracle that can solve the halting problem H. Suppose we want to know whether x is in $\overline{H}$, i.e., whether the x-th partial function diverges. We could ask the oracle if x is in H, then $\overline{H}\ x$ if the answer is not, and vice versa. Any many-one reduction cannot capture this process, while it can be captured by more general reductions such as truth-table reductions or Turing reductions.

In a nutshell, Turing reductions allow for querying the oracle at any point, and decisions about whether to continue asking or computing a result are made based on the answers. In a general sense, divergence is also a possible behaviour. This process can be generalised to any question and answer, meaning the question does not necessarily have to be a natural number, and the answer is more than simply "correct" or "incorrect".

For any input type I, output type O, and a relation $R : Q \to A \to \mathbb{P}\mathsf{rop}$, where Q is the question type and A is the answer type, oracle machines can be described by Figure 3.1. E.g., for an input i, the oracle machine asks the oracle with q_1, and based on the answer a_1, it proceeds to ask q_2, and so on, until it either outputs an answer o or diverges.

Based on these intuitions, we summarise how to render oracle machines synthetically and their basic properties in Section 3.1. The introduction is based on Forster, Kirst, and Mück's works [16, 21, 20] that establish the foundation and applications of oracle computability in synthetic computability. In Section 3.2, we then discuss the step-indexed oracle machines, which approximate the oracle machines through computable functions with indexed steps.

3.1 Synthetic Oracle Computability

In this section, we summarise basic concepts introduced by Bauer, Forster, Kirst, and Mück. For more detailed discussions and examples, we refer to their papers [4, 21, 20].

The first definition of synthetic oracle computability is given by Bauer [4]. Later, Forster and Kirst describe a reformulation in CIC [16], along with another suggestion that enables the connection of Post's theorem and the arithmetical hierarchy [19]. However, the existence of an enumerator of oracle machines for all these definitions necessitates an axiom not derived from the commonly accepted axiom of synthetic computability, CT (see Definition 2.2.2). This requirement leaves a gap in the consistency status.

Forster, Kirst, and Mück propose a stricter notion that requires the interactions between oracle machines and oracles to be *sequential continuous*. [20]. This approach enables the construction of an enumerator of oracle machines based on CT. Building on this, we can define concepts such as the relative halting problem and the

Turing jump without relying on assumptions beyond CT.

In addition to these approaches, Andrew Swan proposes a formulation of Turing reducibility as higher modalities [64].

3.1.1 Oracle Computability

In synthetic computability, oracle computability is modelled by relation transformers $F : (Q \to A \to \mathbb{P}\mathrm{rop}) \to (I \to O \to \mathbb{P}\mathrm{rop})$ that is captured by a *sequential computation*. Such transformers take a relation $R : Q \to A \to \mathbb{P}\mathrm{rop}$ as input and output a relation $F\,R : I \to O \to \mathbb{P}\mathrm{rop}$ for any types Q, A, I, and O.

Definition 3.1 (Oracle Computability) *Let $F\colon (Q \to A \to \mathbb{P}\mathrm{rop}) \to (I \to O \to \mathbb{P}\mathrm{rop})$ be a relation transformer, and $R\colon Q \to A \to \mathbb{P}\mathrm{rop}$ be a relation. F is oracle computable if there is a partial function $\tau\colon I \to A^* \rightharpoonup Q + O$ such that for any input i and output o:*

$$F\,R\,i\,o \leftrightarrow ((\tau\,i); R \vdash qs; as) \wedge (\sigma\ as \downarrow \mathsf{out}\ o)$$

We call τ a sequential computation. In this context, the interrogation relation $\sigma; R \vdash qs; as$ is defined as:

$$\frac{}{\sigma; R \vdash [\,]; [\,]} \qquad \frac{\sigma; R \vdash qs; as \quad \sigma\ as \downarrow \mathsf{ask}\ q \quad R\ q\ a}{\sigma; R \vdash qs +\!\!+ [q]; as +\!\!+ [a]}$$

where the notation $l_1 +\!\!+ l_2$ is list concatenation.

The intuition behind this definition is that a transformer F is oracle computable if there is a sequential computation τ computing based on an input i and the response from an oracle R. For example, consider a sequential computation τ, it begins with an empty answer list $[\,]$, and τ may either ask a question, produce an output or diverge. If τ poses a question q to R, then for any answer a for which $R\ q\ a$ holds, τ continues the computation with the list $[a]$. It may ask another question and continues until it either produces an output or diverges based on some answer list.

This process is precisely what was described at the beginning of this chapter. Sequential computation τ is used to reflect the computational behaviour of oracle machines. Based on this definition, several basic properties of sequential computations and interrogations can be derived.

Lemma 3.2 *Let $F\colon (Q \to A \to \mathbb{P}\mathrm{rop}) \to (I \to O \to \mathbb{P}\mathrm{rop})$ be a relation transformer witnessed by the sequential computation $\tau\colon I \to A^* \rightharpoonup Q + O$, and $R : Q \to A \to \mathsf{Prop}$ be a relation.*

- *If R is functional, then $F\,R$ is functional as well.*
- *For any $\sigma : A^* \rightharpoonup Q + O$, if $\sigma; R \vdash qs; as$ and $R' \equiv_{qs} R$, then $\sigma; R' \vdash qs; as$.*

- *For any* $\sigma : A^* \rightharpoonup Q + O$ *and functional relation* R*, if* $\sigma; R \vdash qs_1; as_1$ *and* $\sigma; R \vdash qs_2; as_2$ *hold,* $|qs_1| \leqslant |qs_2|$ *implies* $qs_1 \sqsubseteq qs_2$ *and* $as_1 \sqsubseteq as_2$*.*

Proof The proof of the first lemma is merely a derivation from the definition. The second lemma is proved by induction on the given interrogation. The third lemma is shown by induction on the length of the question list. □

This synthetic notion of oracle computability is stronger than the one defined by *modulus continuity*, as introduced in the following definition. The latter states that a modulus always exists to control the process of the relation transformer, although this modulus may not be computable.

Definition 3.3 (Modulus Continuity) *For any transformer* F*, relation* R*, input* i*, and output* o *with* $F\ R\ i\ o$*, a list of questions* $r : Q^*$ *is called a modulus (of continuity) if for any other relation* R' *that is equivalent to* R *over* r*, i.e.,* $R \equiv_r R'$*,* $F\ R'\ i\ o$ *holds, formally:*

$$F\ R\ i\ o \to \exists r.\ \forall R'.\ R \equiv_r R' \to F\ R'\ i\ o$$

This definition is strictly weaker than the sequential continuity we have discussed, as illustrated by the following lemma.

Lemma 3.4 *If* F *is oracle computable, then* F *is modulus continuous, but the converse does not hold.*

Proof If F is oracle computable, then by the definition there is a sequential computation τ such that $F\ P\ i\ o \leftrightarrow (\tau\ i); R \vdash qs; as \wedge \sigma\ as \downarrow \mathsf{out}\ o$. Then the list qs is the modulus of F by the basic property of interrogation.

A transformer that is modulus continuous but is not sequential continuous is given by $F := \exists q.\ R\ q\ \mathsf{true}$. Details of the proof can be found in Lemma 2 of the paper by Forster et al. [20]. □

3.1.2 Turing Reductions

A predicate p is Turing reducible to a predicate q if there exists a computable function that can decide p with the capability to ask the decider of q. Considering that every function is computable in our constructive meta-theory, a naive approach may just assume the existence of a function that decides q. However, such an assumption would directly break the faithfulness of synthetic computability.

Therefore, the definition must naturally live in the propositional universe, where the reduction is conceptualized as a predicate transformer that transforms q into p.

Definition 3.5 (Turing Reduction) *A predicate* $p : X \to \mathbb{P}\mathrm{rop}$ *is Turing reducible to a predicate* $q : Y \to \mathbb{P}\mathrm{rop}$ *if there is a oracle computable* $F : (Y \to \mathbb{B} \to \mathbb{P}\mathrm{rop}) \to (X \to \mathbb{B} \to \mathbb{P}\mathrm{rop})$ *that transforms* q *to* p*:*

$$p \preceq_T q \;:=\; \exists F.\, F \textit{ is oracle computable} \land \forall x\, b.\, \hat{p}\ x\ b \leftrightarrow F\ \hat{q}\ x\ b,$$

where the notation $\hat{p}$ *maps the predicate* q *to a relation (see Definition 2.2).*

We next discuss some basic properties of Turing reducibility.

Lemma 3.6 *Turing reducibility is a preorder, namely, it is reflexive and transitive.*

Proof The reflexivity comes from a trivial transformer that maps q to q with following computable tree τ:

$$\begin{aligned} \tau\ i\ [] &:= \mathsf{ask}\ i \\ \tau\ i\ (a :: l) &:= \mathsf{out}\ a \end{aligned}$$

The transitivity comes from composing two sequential computations τ_1 and τ_2, see Theorem 21 in [20]. □

Lemma 3.7 *Many-one reducibility implies Turing reducibility.*

Proof For any predicates p and q with $p \preceq_m q$, let f be the many-one reduction. Then the oracle computable transformer $F\ R\ i\ o \;:=\; R\ (f\ i)\ o$ is defined by defining following τ:

$$\begin{aligned} \tau\ i\ [] &:= \mathsf{ask}\ (f\ i) \\ \tau\ i\ (a :: l) &:= \mathsf{out}\ a \end{aligned}$$ □

A fundamental property of Turing reductions is that once a given oracle is decidable, the oracle machine should behave like a computable function. In other words, an oracle machine with a computable oracle does not enhance computability to a level stronger than a computable function.

To establish the above lemma, we start with a simple lemma that demonstrates how to convert a partial function into a total one.

Lemma 3.8 *For any Boolean partial function* $f : X \rightharpoonup \mathbb{B}$ *and predicate* $p : X \to \mathbb{P}\mathrm{rop}$*, if* $\forall x.\, p\ x \leftrightarrow f\ x \downarrow \mathsf{ret}\, \mathsf{true}$ *and* $\forall x.\, \exists b.\, f\ x \downarrow \mathsf{ret}\, b$*, then* p *is decidable.*

Proof Since $f\ x$ terminates for any x, a total function that decides p can be constructed by extracting the results of termination since $\mathbb{N}$ is discrete. □

We then can show the following lemma by assuming the classical principle MP (see Definition 2.10).

Lemma 3.9 *Assuming* MP, *if* q *is decidable and* $p \preceq_T q$, *then* p *is also decidable.*

Proof Let f be the decider of q. Since the oracle is decidable, the sequential computation τ of F that reduces q to p can be computed by an f such that

$$\forall x\ b.\ \hat{p}\ x\ b \leftrightarrow f\ (\lambda y.\ \mathsf{ret}\ (g\ y))\ x \downarrow b,$$

By MP, we can show that f always terminates, therefore, a decider of p exists by lemma 3.8. □

We define the notion of oracle semi-decidability, which is a generalisation of semi-decidability. If a oracle computable transformer that uses an oracle q is a semi-decider of p, then we say p is semi-decidable in q.

Definition 3.10 (Oracle Semi-Decidable) *A predicate* p *is semi-decidable in the predicate* q, *if there exists a oracle computable transformer* F *such that* $F\ \hat{q}$ *accepts* p*:*

$$\mathcal{S}_q(p) \coloneqq \exists F.\ F \textit{ is oracle computable} \wedge \forall x.\ p\ x \leftrightarrow F\ \hat{q}\ x\ \star$$

We present several properties associated with oracle semi-decidability:

Fact 3.11 *For any predicate* q, *if* p *is semi-decidable, then* $\mathcal{S}_q(p)$.

Proof Let $f : X \rightharpoonup \mathbb{1}$ be the semi-decider for q, the proof comes with the computable transformer $F\ R\ i\ o \coloneqq f\ i \downarrow o$ by the computable tree $\tau \coloneqq f\ i \gg\!= \lambda o.\ \mathsf{ret}\ (\mathsf{out}\ o)$. □

Lemma 3.12 *If* $p \preceq_T q$, *then both* $\mathcal{S}_q(p)$ *and* $\mathcal{S}_q(\overline{p})$.

Proof Let F be the sequential transformer that reduces q to p. Then the transformer $F'\ R\ x\ \star \coloneqq F\ R\ x\ \mathsf{true}$ witnesses $\mathcal{S}_q(p)$. In particular, the sequential computation τ' of F' is defined by outputting $\star$ if F outputs true, otherwise, it diverges. The proof of $\mathcal{S}_q(\overline{p})$ is similar. □

The reverse direction of the above lemma also holds, which generalises the famous theorem by Post. Post's theorem states that any predicate is decidable if and only if both it and its complement are semi-decidable. In the context of oracle semi-decidability, we have the following formulation of Post's theorem.

Theorem 3.13 *If* $\mathcal{S}_q(p)$ *and* $\mathcal{S}_q(\overline{p})$, *then* $p \preceq_T q$,

Proof Since the proof is rather complicated, we refer to Theorem 35 by Forster et al. [21]. □

3.1.3 Turing Jump

As previously mentioned, this definition of oracle computability allows for the enumeration of all oracle machines based on the common axiom of synthetic computability – CT. Under our setting, oracle machines are oracle computable transformers.

In this thesis, we assume EPF (see Definition 2.6), which is equivalent to CT. To construct the enumerator, we derive the following lemma from EPF.

Lemma 3.14 *There exists an enumerator for any family of partial functions, i.e.:*

$$\exists\theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N}).\ \forall f : \mathbb{N} \to (\mathbb{N} \to \mathbb{N}).\ \exists\gamma : \mathbb{N} \to \mathbb{N}. \forall n\ x\ v.\ \theta_{\gamma(n)} x \downarrow v \leftrightarrow f_n\ x \downarrow v$$

Based on this axiom, since any oracle computable transformer can be represented through a sequential computation, this representation takes the form of a partial function between two datatypes when the types of questions, answers, inputs, and outputs are all datatypes. Therefore, the enumerator for oracle machines can be constructed by enumerating all partial functions between two datatypes.

Theorem 3.15 *There exists an enumerator of transformers* $\Xi : \mathbb{N} \to (\mathbb{N} \to \mathbb{B} \to \mathbb{P}\mathrm{rop}) \to (\mathbb{N} \to \mathbb{B} \to \mathbb{P}\mathrm{rop})$ *such that for any oracle computable transformer* F*, there is an index* c*, such that:*

$$\forall\, R\ x\ b.\ F\ R\ x\ b\ \leftrightarrow\ \Xi_c\ R\ x\ b$$

Proof The construction is generally applicable to arbitrary datatypes. However, in this thesis, we fix the types of questions, answers, inputs, and outputs to be $\mathbb{N}$, $\mathbb{B}$, $\mathbb{N}$, and $\mathbb{B}$, respectively. Given this context, the sequential computations have type $\mathbb{N} \to \mathbb{B}^* \rightharpoonup \mathbb{N} + \mathbb{B}$. Consequently, an enumerator $\xi : \mathbb{N} \to (\mathbb{N} \to \mathbb{B}^* \rightharpoonup \mathbb{N} + \mathbb{B})$ can be constructed by:

$$\xi_c\ x\ l\ :=\ \theta_c(\iota_1(x, l)) \gg\!= \lambda v.\ \mathsf{ret}\,(\rho_2\ v)$$

with

$$\forall\tau.\ \exists\gamma.\ \forall n\ x\ l\ v.\ \xi_{\gamma(n)}\ x\ l \downarrow v \leftrightarrow \tau_n\ x\ l \downarrow v$$

where the functions $\rho_1 : \mathbb{N} \to \mathbb{N} \times \mathbb{B}^*$, $\rho_2 : \mathbb{N} \to \mathbb{N} + \mathbb{B}$ and ι_1, ι_2 indicate the enumerability of $\mathbb{N} \times \mathbb{B}^*$ and $\mathbb{N} + \mathbb{B}$, respectively, i.e. $\forall n.\ \iota_{1,2}(\rho_{1,2}(n)) = n$.

As ξ_c is the c-th oracle machine, for any given oracle R and input x, the predicate determining whether the c-th oracle machine is terminated at x can be defined as:

$$\Xi_c\ R\ x\ b\ :=\ \exists qs\ as.\ \xi_c; R \vdash qs; as \wedge \xi_c\ x\ as \downarrow b$$

As a result, the enumerator Ξ is constructed by ξ, and the theorem is proved. □

When it does not cause confusion, for any sequential computation $\tau : I \to A^* \to Q + O$ that only involves datatypes, due to the existence of the enumerator, we sometimes denote it by a code $e : \mathbb{N}$. That is, $e := \xi_e$.

Given a predicate p, the relative halting problem in p (also called Turing jump of p), refers to whether the c-th oracle machine terminates on the input c with the oracle p, can be defined in a manner similar to the halting problem, with the enumerator of partial functions being replaced by the enumerator for oracle machines.

Definition 3.16 *The Turing jump of a predicate* $p : \mathbb{N} \to \mathbb{P}\mathrm{rop}$ *i.e.,* p' *is defined as:*

$$p'\ x := \Xi_x\ \hat{p}\ x\ \star$$

For any predicate p, we define iterated Turing jumps as follows

$$p^{(0)} := p \quad p^{(n+1)} := (p^{(n)})',$$

and a trivial decidable predicate $\mathbb{O} := \lambda x.\ \mathsf{true}$ is used as a basis.

Notice that the Turing jump of a predicate is semi-decidable in itself, but its complement is not.

Lemma 3.17 $\mathcal{S}_p(p')$ *and* $\neg\mathcal{S}_p(\overline{p'})$.

Proof The first is proved by the transformer $\lambda R\ c\ o.\ \Xi_c\ R\ c\ \star$. For the second one, assume that there exists a transformer F that gives $\forall x.\ \neg p'x \leftrightarrow F\ R\ x\ \star$. By definition, we have $\forall x.\ \neg p'x \leftrightarrow \neg\Xi_x\ R\ x\ \star$. Fact 3.15 yields $F\ R\ x \leftrightarrow \Xi_c\ R\ x$ for some c. Thus, the contradiction $\Xi_c\ R\ c\ \star \leftrightarrow \neg\Xi_c\ R\ c\ \star$ is obtained. □

The Turing jump of a predicate is strictly higher than itself in the order defined by Turing reduction, as the following lemma shows.

Lemma 3.18 *For any predicate* p*:*

$$p' \not\preceq_T p \text{ and } p \preceq_T p'$$

Proof If p' is Turing reducible to p, by Lemma 3.12, the complement of p' is semi-decidable in p, which contradict Lemma 3.17. For the second part, we refer to the proof of Lemma 16 in [21]. □

We conclude this recap section with the following lemma, which connects all the definitions above. The detailed proof can be found in Forster et al. [21].

Lemma 3.19 *For any predicate* p *and* q*:*

$$\mathcal{S}_q(p) \leftrightarrow p \preceq_m q'$$

3.2 Step-Indexed Oracle Machines

So far, we have discussed several essential properties of oracle computability. However, since oracle computability is defined in the propositional universal, oracle computable transformers can not be executed. To analyse them, the concepts of *step-indexed execution* and *use functions* are helpful. In this section, we render those concepts synthetically.

We use the following shorthand, where P is a predicate over $\mathbb{N}$:

$$\lim_{x\to\infty} P(x) \;:=\; \exists y.\, \forall x.\, (y \leqslant x \to P(x))$$

3.2.1 Step-Indexed Execution

Step-indexed execution allows the computation to stop at any point to analyse the execution. For instance, the operator seval (see Definition 2.1.2) is the step-indexed execution of partial functions.

However, step-indexed execution for oracle machines is not straightforward. The potential undecidability of an oracle may cause execution to become stuck after a query is sent. For a decidable oracle, however, oracle machines function equivalently to computable functions. Nonetheless, there are remaining possibilities for progress. If the oracle is semi-decidable, we can execute oracle machines with step-indexing and derive non-trivial conclusions. First, we need to index semi-decidable predicates by steps.

Definition 3.20 (Σ_1 Approximation) *For any predicate $p : X \to \mathbb{P}\mathrm{rop}$, a Σ_1 approximation of p is a monotonic sequence $f : \mathbb{N} \to X \to \mathbb{B}$, such that for any $x : X$:*

$$p\ x \leftrightarrow \exists n.\ f(x, n) = \mathsf{true}$$

A sequence $f : \mathbb{N} \to X \to \mathbb{B}$ is monotonic if:

$$\forall n\ x.\ f(n, x) = \mathsf{true} \to \forall m.\ n \leqslant m \to f(m, x) = \mathsf{true}.$$

Lemma 3.21 *A predicate $p : X \to \mathbb{P}\mathrm{rop}$ is semi-decidable if and only if there is a Σ_1 approximation for p.*

Proof A Σ_1 approximation is a semi-decider. For the reverse direction, a semi-decider $g : X \to \mathbb{N} \to \mathbb{B}$ can be extended to a Σ_1 approximation by stabilisation. With the Boolean or function $\_ \,||\, \_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$, we can define:

$$\begin{aligned} f\ 0\ x \;&:=\; g\ x\ 0 \\ f\ (n+1)\ x \;&:=\; g\ x\ (n+1) \,||\, f\ n\ x \end{aligned}$$

It's easy to verify that f Σ_1 approximates p. □

We also denote the Σ_1 approximation of a predicate p by p_n, a family of functions such that $f(x, n) = \mathsf{true} \leftrightarrow p_n\ x$. Then, we can define the step-indexed execution of an oracle machine with a semi-decidable oracle p as follows:

Definition 3.22 (Step-Indexed Oracle Machines) *For any given datatype T, a function $\phi : (\mathbb{N} \to \mathbb{N}^* \rightharpoonup \mathbb{N}+T) \to (\mathbb{N} \to \mathbb{P}\mathrm{rop}) \to \mathbb{N} \to \mathbb{N} \to T^?$ is called a step-indexed oracle machine, if it meets the following requirement for any oracle machine e with a semi-decidable oracle $p : \mathbb{N} \to \mathbb{P}\mathrm{rop}$:*

$$\forall x\ b.\ \Xi_e\ \hat{p}\ x\ b \to \lim_{n\to\infty} (\phi_e^p(x)[n] = \ulcorner b \urcorner)$$

where we abbreviate $\phi_e^{p_n}\ x\ n$ as $\phi_e^p(x)[n]$.

To construct step-indexed oracle machines, we have to examine the definition of oracle computability to execute sequential computation. This means we can decide how many steps to explore and how much depth to probe each time for a given sequential computation.

Definition 3.23 (Step-Indexed Execution) *Let $h : Q \to A$ be a function and $\tau : I \to A^* \rightharpoonup Q + O$ be a sequential computation. For the maximum steps i that can be explored, and the maximum depth j, step-indexed execution can be defined by recursively exploring the sequential computation τ:*

$$\phi_\tau^h\ x\ i\ j := \begin{cases} \ulcorner \mathsf{out}\ \ o \urcorner & \textit{if } (\tau\ x\ [\,])\downarrow_j \mathsf{out}\ \ o \\ \ulcorner \mathsf{ask}\ \ q \urcorner & \textit{if } (\tau\ x\ [\,])\downarrow_j \mathsf{ask}\ \ q \textit{ and } i = 0 \\ \Phi^h_{\lambda r.\ \tau\ x\ h(q)::r}\ x\ i'\ j & \textit{if } (\tau\ x\ [\,])\downarrow_j \mathsf{ask}\ \ q \textit{ and } i = i' + 1 \\ \mathsf{none} & \textit{otherwise} \end{cases}$$

There are monotonic properties associated with the indices of step-indexed execution ϕ.

Fact 3.24 *For any given function $h : Q \to A$ and any sequential computation $\tau : I \to A^* \rightharpoonup Q + O$, the following properties hold:*

- $\phi_\tau^h\ x\ i\ j = \ulcorner v \urcorner$ *implies that for any j' where $j \leqslant j'$, $\phi_\tau^h\ x\ i\ j' = \ulcorner v \urcorner$.*
- $\phi_\tau^h\ x\ i\ j = \ulcorner \mathsf{out}\ o \urcorner$ *implies that for any i' where $i \leqslant i'$, $\phi_\tau^h\ x\ i'\ j = \ulcorner \mathsf{out}\ o \urcorner$.*

Proof For the first fact, by using induction on step i and case analysis on the output of $\tau\ x\ [\,]$, we only need to show that if $x\downarrow_j o$ holds, then $x\downarrow_{j'} o$ also holds for a deeper j'. In turn, this follows the monotonicity of seval.

For the second fact, it suffices to show that if $\phi_\tau^h\ x\ i\ j = \ulcorner \mathsf{out}\ o \urcorner$, then $\phi_\tau^h\ x\ (i+1)\ j = \ulcorner \mathsf{out}\ o \urcorner$. This can also be proved by induction on i and case analysis on $\tau\ x\ [\,]$.

The critical point is that the induction must move τ outside the induction scope, as τ may be replaced by a new one in the inductive case to enable the induction hypothesis. □

Based on this definition, we present some technical lemmas that hold intuitively but are intricate in formal proofs. We use the notation $\hat{g}\ x\ y\ :=\ g\ x = y$ to turn a function into a relation.

Fact 3.25 *For any function $g : Q \to A$ and a sequential computation $\tau : A^* \rightharpoonup Q + O$, if the interrogation always yields results $qs; as$, and for that sequence of answers, the oracle machine always terminates. Then τ always terminates for any prefix of answers as':*

$$\tau; \hat{g} \vdash qs; as \to \tau\ as \downarrow v \to \exists j.\ \forall as' \sqsubseteq as.\ \exists w.\ (\tau\ x\ as') \downarrow_j w$$

Proof By induction on the interrogation. For the base case where the answer list is empty, the result is straightforward. For the inductive case, perform a case analysis on $as' \sqsubseteq as +\!\!+ [a]$. If $as' = as +\!\!+ [a]$, then the result follows directly from the assumption. If $as' \sqsubseteq as$, it is proved by the induction hypothesis. □

To help us with the proof, notice that when defining the step-indexed execution of a sequential computation τ, all answers are added to τ, i.e., we get another τ, and the following theorem helps us to run the step-indexed execution.

Fact 3.26 *For any function $g : Q \to A$ and a sequential computation $\tau : A^* \rightharpoonup Q + O$, if $\tau; \hat{g} \vdash qs; as$ and $\forall as' \sqsubseteq as.\ \exists w.\ (\tau\ x\ as') \downarrow_j w$:*

$$\phi^g_{\lambda r.\ \tau\ x\ (as +\!+ r)}\ x\ n\ j = \ulcorner v \urcorner \leftrightarrow \phi^g_\tau\ x\ (|as| + n)\ j = \ulcorner v \urcorner$$

Proof By induction on the interrogation and then making use of the basic properties of lists. □

Fact 3.27 *For any sequential computation $\tau : A^* \rightharpoonup Q + O$ and functions $g, h : Q \to A$, we have:*

$$\tau; \hat{g} \vdash qs; as \to \tau\ as \downarrow \mathsf{out}\ o \to \exists i\ j.\ \forall h.\ \tau; \hat{h} \vdash qs; as \to\ \ \phi^h_\tau\ i\ j = \ulcorner \mathsf{out}\ o \urcorner$$

Proof By Fact 3.25, there exists a depth j such that the step-indexed oracle machine does not get stuck. Let the number of steps be $|as|+1$. The result can then be verified by Fact 3.26. □

Lemma 3.28 *For any family of functions $g : \mathbb{N} \to Q \to A$, and a sequential computation $\tau : I \to A^* \rightharpoonup Q + O$.*

$$(\lim_{k\to\infty} \tau; \hat{g}_k \vdash qs; as) \to \tau\ as \downarrow \mathsf{out}\ v \to (\lim_{k\to\infty} \phi^{g_k}_\tau\ n\ n = \ulcorner \mathsf{out}\ v \urcorner)$$

Proof For a sufficiently large k, $\tau\colon \hat{g}_k \vdash qs; as$ holds, and Fact 3.27 provides a sufficiently large depth j and step i such that the step-indexed oracle machine halts. Let $c = \max(k, i, j)$. For any c' where $c \leqslant c'$, we have $\phi_\tau^{g_{c'}}\ c'\ c' = \ulcorner \mathsf{out}\ \nu \urcorner$ because it is monotonic in both step and depth according to Fact 3.24. Additionally, $\tau; \hat{g}_{c'} \vdash qs; as$ holds by assumption. □

As there are two indices for the step-indexed execution, we can unify indices to obtain step-indexed oracle machines by defining the following:

$$\phi_e^f\ x\ n \coloneqq \phi_e^f\ x\ n\ n$$

Theorem 3.29 *For any semi-decidable predicate* $p : \mathbb{N} \to \mathbb{P}\mathrm{rop}$ *and an input* $x : \mathbb{N}$,

$$\Xi_e\ \hat{p}\ x\ b \to \lim_{n \to \infty} \phi_e^{p_n}\ x\ n = \ulcorner b \urcorner,$$

where b *is an instance of datatypes* T.

Proof By definition of $\Xi_e\ \hat{p}\ x$, we have $\exists qs\ as.\ \xi_e; \hat{p} \vdash qs; as$. Since p has a Σ_1 approximation, there is a sufficient large k such that $\xi_e; \hat{p}_{k'} \vdash qs; as$ for any $k' \geqslant k$. By Fact 3.28, we obtain the desired result. □

3.2.2 Use Functions

Since step-indexed oracle machines can approximate oracle machines, the information used over the computation can be collected and reflect the behaviour of oracle machines. In this section, we introduce another concept connecting to the step-indexed oracle machines, namely *use functions*.

Use functions compute the maximum number of questions a step-indexed oracle machine asks for a given input and steps. For instance, for a computation $\phi_e^p(x)[n] = \ulcorner \nu \urcorner$, if this computation asks questions $q_1, \ldots, q_m$ to the oracle, the use function $u_e^p(x)[n]$ outputs the value $\max(q_1, \ldots, q_m) + 1$. If $\phi_e^p(x)[n] = \mathsf{none}$, then the use function $u_e^p(x)[n]$ outputs 0. In other words, use functions compute the part of the oracle that has been used. If a value greater than the result of the use function (if not 0) is added to the oracle for the same inputs and number of steps, the step-indexed oracle machines will have the same result.

Definition 3.30 (Use Functions) *Given a step-indexed oracle machine* ϕ, *a use function* $u : \mathbb{N} \to (\mathbb{N} \to \mathbb{P}\mathrm{rop}) \to \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ *is a function that satisfies the following properties: Let* $w = u_e^q(x)[n]$,

$$\phi_e^q(x)[n] \downarrow \to \forall p.\ p \equiv_w \hat{q}_n \to \Xi_e\ \hat{p}\ e \star\,, \tag{3.1}$$

$$\phi_e^q(x)[n] \downarrow \to q_{n+1} \equiv_w q_n \to \phi_e^q(x)[n+1] \downarrow\,, \tag{3.2}$$

$$\phi_e^q(x)[n] \downarrow \to q_{n+1} \equiv_w q_n \to u_e^q(x)[n+1] = w, \tag{3.3}$$

where we abbreviate $\phi_e^q(x)[n] = \ulcorner \mathsf{out}\ \star \urcorner$ *as* $\phi_e^q(x)[n] \downarrow$, *and* $u_e^{q_n}\ x\ n$ *as* $u_e^q(x)[n]$.

The first property (3.1) is the completeness of step-indexed oracle machines, which states if the step-indexed execution of an oracle machine terminates at step n, then for any oracle p equivalent to p_n up the result of the use function, this oracle machine with p terminates as well. Based on the properties (3.2) and (3.3), for any given step n with the termination of step-indexed execution, the step-indexed execution, as well as the use function, behave the same at step $n+1$ if the p_{n+1} is equivalent to p_n up to the result of the use function.

We next introduce the construction of use functions.

Definition 3.31 (Information Predicates) *For any function* $f : \mathbb{N} \to \mathbb{B}$ *and* $e, n : \mathbb{N}$, *we define the information predicate over a question list* qs *and an answer list* as *as:*

$$\begin{aligned}\iota_e^f(x)[n \mid qs; as] \;:=\;& \tau\ as \downarrow \mathsf{out}\ \star \\ & \wedge \forall p.\ p \equiv_{qs} \hat{f} \to \tau\ x; \hat{p} \vdash as; qs\end{aligned}$$

We now show that an information predicate can be derived from a computation of step-indexed oracle machine.

Lemma 3.32 (Extraction Functions) *There is an extraction function* E *such that for any proof* h *of computation* $\phi_e^f(x)[n]\downarrow$, $E\ h$ *outputs a question list* qs *and an answer list* as, *together with a certificate showing their properties.*

$$E\colon \phi_e^f(x)[n]\downarrow \to \Sigma\ qs\ as.\ \iota_e^f(x)[n \mid qs; as]$$

Proof By induction on n, consider the base case where both qs and as are $[\,]$. The hypothesis provides sufficient information for this case.

For the inductive case, performing the case analysis on H, if the result of the sequential computation $\tau\,[\,]$ is an answer, let $qs = as = [\,]$. The assumption is sufficient.

If the result of the computation is a question q, we get qs' and as' for the step $n-1$ by the induction hypothesis. Now let $qs = q :: qs'$ and $as = f(q) :: as'$. The remaining proof only requires the basic properties of interrogation.

Fact 3.33 *For any fixed sequential computation and step-index, the output of step-indexed oracle machines is decidable:*

$$\phi_\tau^f(x)[n]\downarrow \;+\; \neg\phi_\tau^f(x)[n]\downarrow$$

As we can decide whether $\phi_\tau^f\ x\ n\ m\downarrow$ using its proof, the use function can be defined as follows:

Definition 3.34 (Use Function)

$$u_e^f(x)[n] := \begin{cases} \max(\pi\,(E\ h)) + 1 & \textit{if } h := \phi_\tau^f(x)[n] \downarrow \\ 0 & \textit{otherwise} \end{cases}$$

The idea behind this definition is to determine if a computation has terminated. If the computation is not complete and returns none, the use function outputs 0. Conversely, if the computation finishes with the proof $H : \phi_\tau^f(x)[n] \downarrow$, inserting this proof into the extraction function E yields an informative type. The first element – a list of questions qs, is then extracted using the projection π_1. Then the use function outputs one more than the maximum value in qs to differentiate from cases where the computation has not finished. We start with some basic facts about use functions.

Fact 3.35 *For any given computation $\phi_e^f(x)[n]$, we have:*

- $\forall m.\ u_e^f(x)[n] = m + 1 \leftrightarrow \phi_e^f(x)[n] \downarrow$
- $u_e^f(x)[n] = 0 \leftrightarrow \neg\phi_e^f(x)[n] \downarrow$

Proof The proof is straightforward by the definition of use functions. □

The correctness of use functions follows the definition of information predicate, and we provide the following proof of the correctness of property (3.1) for use functions (see Definition 3.30):

Theorem 3.36 *Let $w = u_e^q(x)[n]$,*

$$\phi_e^q(x)[n] \downarrow \to \forall p.\ p \equiv_w \hat{q}_n \to \Xi_e\ \hat{p}\ e\ \star$$

Proof If the computation has terminated, according to the definition of use function, w is determined as $\max(qs) + 1$. Given that p and $\hat{q}_n$ are equivalent up to w, this implies they are equivalent up to qs, denoted as $p \equiv_{qs} \hat{q}_n$.

Therefore, by applying the information predicate, we show that $\tau\ as \downarrow out\ \star \wedge \tau\ x; \hat{p} \vdash as; qs$, which is $\Xi_e\ \hat{p}\ e\ \star$. □

To establish the last two properties, a crucial technical lemma tells us when two step-indexed executions will have the same outputs:

Lemma 3.37 *For any functions f, g, we have the following property:*

$$(\phi_e^f(x)[n] \downarrow) \to (\tau\ as \downarrow out\ \star) \to (\tau\ x; \hat{f} \vdash as; qs) \to (\tau\ x; \hat{g} \vdash as; qs) \to (\phi_e^g(x)[n] \downarrow)$$

Proof Proving this lemma involves more details and trivial facts, but the core idea is simple: If two step-indexed executions on interrogations that behave the same should turn out to be the same. By induction on one of the interrogation and analysing the other one, we can get that at each step of the step-indexed execution, they are the same. □

We then verify the properties (3.2) and (3.2) hold for our construction of u, which implies that u are use functions according to the definition (see Definition 3.30).

Theorem 3.38 *Let* $w = u^q_e(x)[n]$,

$$(\phi^q_e(x)[n]\downarrow) \to (q_{n+1} \equiv_w q_n) \to (\phi^q_e(x)[n+1]\downarrow)$$

Proof With the information predicate, we get $\tau\ x; \hat{q}_n \vdash as; qs$. Since $q_{n+1} \equiv_w q_n$, we have $\tau\ x; \hat{q}_{n+1} \vdash as; qs$. By Lemma 3.37 and the monotonicity of step-indexed oracle machines, we obtain the goal $\phi^q_e(x)[n+1]\downarrow$. □

Theorem 3.39 *Let* $w = u^q_e(x)[n]$,

$$(\phi^q_e(x)[n]\downarrow) \to (q_{n+1} \equiv_w q_n) \to (u^q_e(x)[n+1] = w)$$

Proof If $u^q_e(x)[n+1]$ is not equal to 0, we then verify the list of questions that used to compute the use function is the same as the one in the step-indexed execution. Therefore, the same as $u^q_e(x)[n]$. If $u^q_e(x)[n+1]$ is equal to 0, similar to the proof of the previous theorem, we can get the fact $\phi^q_e(x)[n+1]\downarrow$, so that the computation must be terminated, and the use function is not 0. □

Chapter 4

Limit Computability

Limit computability is a more general notion than semi-decidability. Similar to the definition of semi-decidable, a predicate is limit computable if a *limit decider* decides the predicate. Shoenfield first introduced this concept in 1959 [58]. However, due to overlooking Shoenfield's work, Gold reintroduced this concept in 1964 [28].

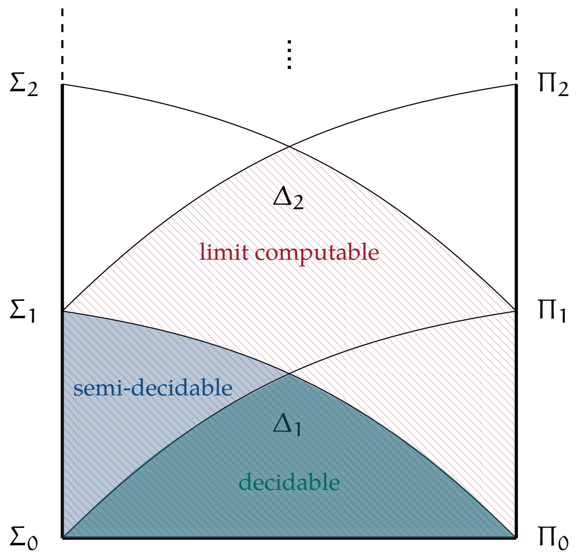

Figure 4.1: Arithmetical Hierarchy

To build a complete big picture of computability theory, we begin in Section 4.1 by briefly reviewing the arithmetical hierarchy due to Kleene [36] and Mostowski [45], classify the undecidability of predicates based on the complexity of formulas that define them. We aim to give the basic concepts and intuitions based on the recent development of the arithmetical hierarchy and Post's theorem in synthetic computability by Forster, Kirst, and Mück [21].

This overview will examine how limit computability fits within this hierarchy, as illustrated in Figure 4.1 (more details will be presented in the following sections). In Section 4.3, we will establish that a predicate is limit computable if and only if it is Turing reducible to the halting problem H, known as the limit lemma. Due to the straightforward definition of limit computability, any proof of Turing reducibility to H can be replaced by showing limit computability.

4.1 Arithmetical Hierarchy

This section briefly reviews Forster, Kirst and Mück's work on the arithmetical hierarchy in synthetic computability [21]. In the arithmetical hierarchy, all predicates are classified into Σ_n and Π_n using first-order formulas. For instance, for any semi-decidable predicate p, there exists a decidable arithmetic formula R such that $p\ x$ if and only if $\exists n. R(x, n)$.

Considering that repeated quantifiers can be folded into one, we can generalise this concept by alternating different quantifiers. We define the Σ_n as follows:

$$\Sigma_n \coloneqq \{p \mid \forall x.\ p\ x \leftrightarrow \exists y_1\ \forall y_2\ \exists y_3\ \forall y_4\ \dots\ R(x, y_1, \dots, y_n) \land \mathcal{D}(R)\}$$

According to above definition, Σ_1 refers to semi-decidable predicates, and Σ_0 denotes decidable predicates. In the definition of Σ_n, formulas begin with an existential quantifier. If we start with a universal quantifier instead, we can define the Π_n classes.

$$\Pi_n \coloneqq \{p \mid \forall x.\ p\ x \leftrightarrow \forall y_1\ \exists y_2\ \forall y_3\ \exists y_4\ \dots\ R(x, y_1, \dots, y_n) \land \mathcal{D}(R)\}$$

Similarly, Π_0 denotes decidable predicates. Additionally, the concept of Δ_n is referring a predicate for both in Π_n and Σ_n.

$$\Delta_n \coloneqq \Sigma_n \cap \Pi_n$$

In synthetic computability, the arithmetical hierarchy is defined mutually as inductive predicates [21].

Definition 4.1 (Arithmetical Hierarchy)

$$\frac{\forall v : \mathbb{N}^k.\ p\ v \leftrightarrow f\ v = \mathsf{true}}{\Sigma_0^k\ p} \qquad \frac{\Pi_n^{k+1}\ q \quad \forall v : \mathbb{N}^k.\ p\ v \leftrightarrow \exists x.\ q(x :: v)}{\Sigma_{n+1}^k\ p}$$

$$\frac{\forall v : \mathbb{N}^k.\ p\ v \leftrightarrow f\ v = \mathsf{true}}{\Pi_0^k\ p} \qquad \frac{\Sigma_n^{k+1}\ q \quad \forall v : \mathbb{N}^k.\ p\ v \leftrightarrow \forall x.\ q(x :: v)}{\Pi_{n+1}^k\ p}$$

Similarly, we have the following definition of Δ_n:

$$\Delta_n^k\ p := \Sigma_n^k\ p \wedge \Pi_n^k\ p$$

Note that the superscript k here is different from the definitions in traditional textbooks. It denotes the arity of the predicate, whereas traditional textbooks usually use the superscript 0 to denote a predicate over $\mathbb{N}$. Our definitions of Σ_n^k and Π_n^k are the same as Σ_n^0 and Π_n^0 in textbooks. We omit the arity k when it is clear from the context.

Based on this synthetic definition of the arithmetical hierarchy, we establish the following facts:

Fact 4.2 *For any predicate p and q, the following facts hold:*

- p *is semi-decidable if and only if it is in* Σ_1,
- *If* $n \leqslant m$, *then* $\Sigma_n \subseteq \Sigma_m$ *and* $\Pi_n \subseteq \Pi_m$,
- $\Sigma_n \subseteq \Pi_{n+1}$ *and* $\Pi_n \subseteq \Sigma_{n+1}$,
- *if* $q \preceq_m p$, *then* $\Sigma_n\ p$ *implies* $\Sigma_n\ q$ *and* $\Pi_n\ p$ *implies* $\Pi_n\ q$.

As pointed out in Post's hierarchy theorem (see Theorem 4.6) below, there are significant connections between oracle computability and the arithmetical hierarchy as Figure 4.1 shown. To establish this connection, classical axioms are needed. To further explore these connections, we introduce an arithmetical hierarchy of the law of excluded middle as proposed by Akama et al. [1].

Definition 4.3 *For an arbitrary hierarchy $n : \mathbb{N}$, we have the following axioms:*

$$\begin{aligned}
\Sigma_n\text{-LEM} &:= \forall k.\ \forall p : \mathbb{N}^k \to \mathbb{P}\text{rop}.\ \Sigma_n\ p \to \forall v.\ p\ v \vee \neg\ p\ v \\
\Pi_n\text{-LEM} &:= \forall k.\ \forall p : \mathbb{N}^k \to \mathbb{P}\text{rop}.\ \Pi_n\ p \to \forall v.\ p\ v \vee \neg\ p\ v \\
\Sigma_n\text{-DNE} &:= \forall k.\ \forall p : \mathbb{N}^k \to \mathbb{P}\text{rop}.\ \Sigma_n\ p \to \forall v.\ \neg\neg\ p\ v \to p\ v \\
\Pi_n\text{-DNE} &:= \forall k.\ \forall p : \mathbb{N}^k \to \mathbb{P}\text{rop}.\ \Pi_n\ p \to \forall v.\ \neg\neg\ p\ v \to p\ v
\end{aligned}$$

Fact 4.4 *These logical axioms satisfy the following facts:*

- Σ_n-LEM $\to \Sigma_n$-DNE.
- Π_n-LEM $\to \Pi_n$-DNE.
- Σ_n-DNE $\leftrightarrow \Pi_{n+1}$-DNE.
- Π_{n+1}-LEM $\to \Sigma_n$-LEM.
- Σ_n-LEM $\to \Pi_n$-LEM

- Σ_{n+1}-DNE $\to \Sigma_n$-LEM.

With this arithmetical hierarchy, we can see the connection to other axioms:

Fact 4.5

- Σ_0-LEM *holds constructively, and thus all* 0 *levels axioms.*
- Σ_1-LEM $\leftrightarrow$ LPO.
- Σ_1-DNE $\leftrightarrow$ MP.
- Π_1-DNE *holds constructively.*

To conclude, we have the following theorem:

Theorem 4.6 (Post's Hierarchy Theorem) *Assuming* Σ_n-LEM, *for any predicate* p, *the following facts hold:*

- $\Sigma_{n+1}\ p$ *if and only if* $\mathcal{S}_q(p)$ *for some* q *in* Π_n,
- $\Sigma_{n+1}\ p$ *if and only if* $\mathcal{S}_q(p)$ *for some* q *in* Σ_n,
- $\Sigma_n \mathbb{O}^{(n)}$,
- *if* $\Sigma_n\ p$ *then* $p \preceq_m \mathbb{O}^{(n)}$, *and thus* $p \preceq_T \mathbb{O}^{(n)}$,
- $\Sigma_{n+1}\ p$ *if and only if* $\mathcal{S}_{\mathbb{O}^{(n)}}(p)$.

For detailed proofs we refer to the paper by Forster et al. [21].

4.2 Synthetic Limit Computability

In mathematics, a limit is the value that a function reaches when the index goes infinite. We say the limit of a function $f : \mathbb{N} \to X$ is the value b when there is a bound n, such that $f(m) = b$ for any m after n, formally:

$$\lim_{n\to\infty} f(n) = b \;\coloneqq\; \exists m.\, \forall n \geqslant m.\, f\ n = b$$

Definition 4.7 (Limit Computability) *A relation* $R : X \to Y \to \mathbb{P}\mathrm{rop}$ *is limit computable if there is a function* $f : X \to \mathbb{N} \to Y$, *such that* $R\ x\ y$ *if and only if the limit of* $f\ x$ *in* n *is* y, *formally:*

$$\forall x\ y.\ R\ x\ y \leftrightarrow \lim_{n\to\infty} f(x, n) = y$$

We call f *a limit decider. For any predicate* $p : X \to \mathbb{P}\mathrm{rop}$, p *is limit computable if* $\hat{p} : X \to \mathbb{B} \to \mathbb{P}\mathrm{rop}$ *is limit computable.*

By unfolding the definition of limit computable, a predicate $p : X \to \mathbb{P}\mathrm{rop}$ is limit computable if there is a function $f : X \to \mathbb{N} \to \mathbb{P}\mathrm{rop}$ such that:

$$p\ x \leftrightarrow \lim_{x \to \infty} f(x, n) = \mathsf{true} \quad \text{and} \quad \neg\, p\ x \leftrightarrow \lim_{x \to \infty} f(x, n) = \mathsf{false}$$

In other words, an element x is in p if the decider f is convergent to $\mathtt{true}$, and verse vice. An element x is not in p if and only if the decider f is convergent to false. As Figure 4.2 shows, if x is in a limit computable predicate p, there is bound n, such that for any m, where $n \leqslant m$, there is $f(x, m) = \mathsf{true}$ for the limit decider f.

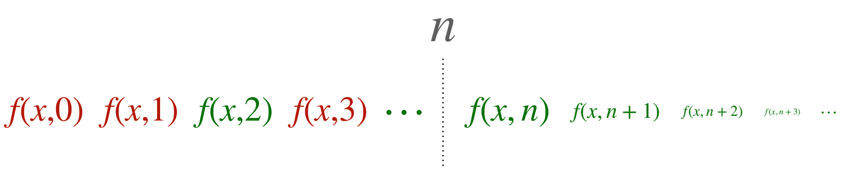

Figure 4.2: Limit Computable

We briefly discuss why we choose this definition as the definition of limit computability in synthetic computability, considering different classically equivalent definitions of limit computability. For instance, one can define limit computability of $p : X \to \mathbb{P}\mathrm{rop}$ as:

$$p\ x \leftrightarrow \lim_{n \to \infty} f(x, n) = \mathsf{true} \text{ and } \exists b.\ \lim_{n \to \infty} f(x, n) = b$$

By this definition, a limit decider f is a convergent function such that $p\ x$ if and only if f converges to true. It's easy to verify that these two definitions are equivalent by assuming LEM. However, the latter implies that p is logically decidable since f is either convergent to true or false. This concludes that our definition of limit computable is a more constructive version.

By unfolding our definition of limits, we have:

$$p\ x \leftrightarrow \exists n.\ \forall m.\ n \leqslant m \to f(x, m) = \mathsf{true}$$
$$\neg\, p\ x \leftrightarrow \exists n.\ \forall m.\ n \leqslant m \to f(x, m) = \mathsf{false}$$

As $n \leqslant m$ is decidable and so is $f(x, m) = \mathsf{true}$, we can then have a function that decides whether $n \leqslant m \to f(x, m) = \mathsf{true}$ holds. Hence, by the definition of the arithmetical hierarchy, we can show the following fact:

Fact 4.8 *For any limit computable predicate p, both p and $\overline{p}$ are in Σ_2. This also means that p is classically in Π_2, and together p is classically in Δ_2.*

4.3 Limit Lemma

A central result of limit computability is the limit lemma, which states that a predicate is limit computable if and only if it is Turing reducible to H. By using results from before, deriving the limit lemma is straightforward.

Lemma 4.9 *Assuming Σ_1-LEM, for any limit computable predicate p, p is Turing reducible to H.*

Proof Suppose that p is limit computable, then both p and $\overline{p}$ are in Σ_2 by Fact 4.8. Since H is a semi-decidable predicate, we have $S_H(p)$ and $S_H(\overline{p})$ by Post's hierarchy theorem (see Theorem 4.6) and Σ_1-LEM. Therefore, p is Turing reducible to H by applying Post's theorem (see Theorem 3.13). □

To prove that reducibility to H implies limit computability, we need to construct a limit decider from a Turing reduction, i.e., an oracle machine. This requires using step-indexed oracle machines based on the results established in Chapter 3.

Lemma 4.10 *For any predicate p, by assuming the logical decidability of p, if p is Turing reducible to H, then p is limit computable.*

Proof Suppose that p is Turing reducible to H, then there is a Turing reduction from p to H. By the definition of Turing reduction, we have an oracle computable transformer $F : (\mathbb{N} \to \mathbb{B} \to \mathbb{P}\mathrm{rop}) \to (\mathbb{N} \to \mathbb{B} \to \mathbb{P}\mathrm{rop})$ such that:

$$\forall x\, b.\, \hat{p}\ x\ b \leftrightarrow F\ \hat{H}\ x\ b$$

By properties of step-indexed oracle machines (see Definition 3.22), we have a function $\phi : \mathbb{N} \to \mathbb{N} \to \mathbb{B}^?$ such that:

$$F\ \hat{H}\ x\ b \to \lim_{n\to\infty} \phi(x)[n] = \ulcorner b \urcorner$$

Let the limit decider $f : X \to \mathbb{N} \to \mathbb{B}$ be:

$$f(x, n) \;:=\; \begin{cases} b & \text{if } \phi_e^H(x)[n] = \ulcorner b \urcorner \\ \mathsf{false} & \text{otherwise} \end{cases}$$

We then have $F\ \hat{H}\ x\ b \to \lim_{n\to\infty} f(x, n) = b$. Therefore,

$$\hat{p}\ x\ b \to \lim_{n\to\infty} f(x, n) = b$$

In order to show $\lim_{n\to\infty} f(x, n) = b \to \hat{p}\ x\ b$, we can do a case analysis on $p\ x$ since p is logically decidable. We consider the proof when $b = \mathsf{true}$. The proof of $b = \mathsf{false}$ is analogous. If $p\ x$ holds, then we obtain $\hat{p}\ x\ \mathsf{true}$ directly. If $\neg p\ x$

holds, it implies that $\lim_{n\to\infty} f(x, n) = \mathsf{false}$ by the above fact, which contradicts the assumption $\lim_{n\to\infty} f(x, n) = \mathsf{true}$. Therefore, we show that f is a limit decider for p:

$$\forall x\ b.\ \hat{p}\ x\ b \leftrightarrow \lim_{n\to\infty} f(x, n) = b$$

So p is limit computable. □

Since Σ_1-LEM is equivalent to LPO (see Fact 4.5), we can show that any limit computable predicate is Turing reducible to H by assuming LPO (see Lemma 4.9). By assuming LPO, if a predicate p is Turing reducible to H, then we have p is logically decidable (see Lemma 2.12). Therefore, we can show that LPO is sufficient for the limit lemma.

This lemma provides us with a convenience: When we need to show that a predicate is Turing reducible to the halting problem, instead of constructing a reduction, it is suffices to show that there is a limit decider for this predicate.

Chapter 5

Post's Problem

Post's problem is about whether there exists a semi-decidable predicate that is undecidable and *strictly easier* than the halting problem [52]. "Easier" here is defined based on a general reduction – Turing reduction (see Chapter 3 for a formal definition). Intuitively, a predicate being "easier" than the halting problem means that even if there is a machine that could answer any question about this predicate, the halting problem still cannot be solved by asking this machine questions arbitrary often.

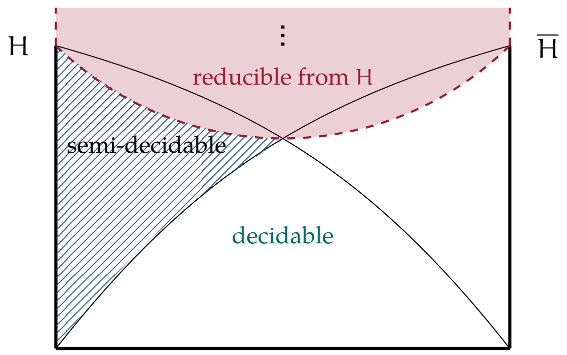

Figure 5.1: Post's Problem

Twelve years after the publication of Post's paper, Friedberg and Muchnik independently solved the problem by developing the priority method [27, 46], demonstrating the existence of the predicate in Post's problem. Therefore, as shown in Figure 5.1, we can find some predicates in the shaded area. The priority method, a crucial tool in the Friedberg-Muchnik construction, was central to this result. Its ability to construct rich semi-decidable predicates make it essential to this field [47, 44, 55, 56].

This chapter is not technical, we aim to review the meaning of Post's problem, including why it is mathematically interesting and how it has been solved historically. Instead, the next chapter will be about technical implementation. In Section 5.1, we will introduce variants of Post's problem under different reducibility notions, as well as the notion of Turing degree, a way of looking at the structure of undecidability other than the arithmetical hierarchy (see Section 4.1). Section 5.2 briefly reviews how the Friedberg-Muchnik theorem solves Post's problem and the intuition behind it. In the final section, we will briefly discuss Lerman and Soare's construction of low simple predicates, and by comparing this with the Friedberg-Muchnik construction, we will explain why we chose it as our source.

5.1 Post's Problem

Post's 1944 paper [52] introduced many essential concepts of computability theory. This section will review the paper in general and explore how Post naturally presented this open problem. For discussions on synthetic computability, we refer to Forster, Kirst, and Mück [18, 21] . We will focus on predicates that are easier than the halting problem. According to the limit lemma (see Section 4.3), those predicates are limit computable.

5.1.1 Many-One Degrees

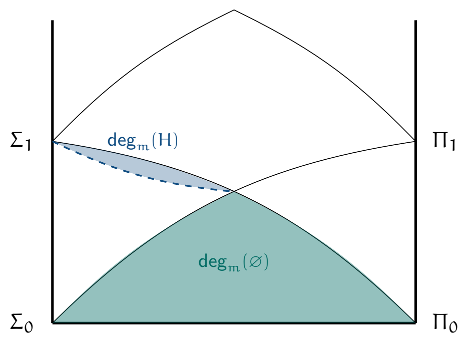

Figure 5.2: Many-one Degrees

Since a reduction introduces an order strucutre of predicates, we can then classify undecidability by different reductions. We start from the most straightforward reduction – the many-one reduction. The many-one degree of a predicate p is defined as:

$$\mathsf{deg}_m(p) := \{q \mid p \preceq_m q \land q \preceq_m p\}$$

As shown in Figure 5.2, since decidable predicates are many-one reducible to each other, all decidable predicates are fitted into the many-one degree of the trivial decidable predicate $\varnothing$. Predicates in the many-one degree of H are semi-decidable as well by the property of many-one reductions. Similar to Post's problem, we can then ask whether there exists a semi-decidable predicate A, such that

$$\varnothing \prec_m A \prec_m H,$$

where the notation $p \prec_m q$ denotes $p \preceq_m q \land p \notin \mathrm{deg}_m(q)$.

This problem is a variant of Post's problem with respect to many-one degrees, which was solved by Post's construction of *simple* predicates. A simple predicate is a semi-decidable predicate whose complement does not contain any infinite semi-decidable predicates.

A simple predicate is semi-decidable yet undecidable and not in the many-one degree of H, which provides a positive solution to the above question. The existence of simple predicates was shown in Post's 1944 paper [52] and a synthetic construction of simple predicates was established by Forster and Jahn in 2023 [18].

Theorem 5.1 (Post, 1944) *There exists a simple predicate A, such that $\varnothing \prec_m A \prec_m H$.*

However, the simpleness of predicates does not imply they cannot be Turing reduced from the halting problem, as Turing reducibility is more general than many-one reducibility. Next, we will examine a reduction stricter than Turing reducibility but more general than many-one reducibility: truth-table reducibility.

5.1.2 Truth-Table Degrees

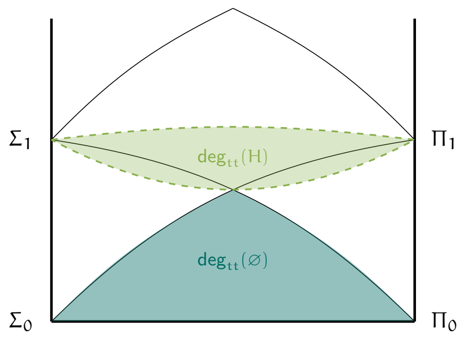

Figure 5.3: Truth-table Degrees

Truth-table reduction is analogous to Turing reduction. For a predicate p to be truth-table reducible to q, denoted as $p \preceq_{tt} q$, it means that to solve a problem p, the reduction describes the answer as a truth table of some finite number of questions to q. Since this table is bounded, truth-table reduction is strictly weaker than Turing reduction, and therefore also called bounded Turing reduction. We do not give formal definitions but will show Post's progress towards solving Post's problem in his 1944 paper, in which he solved another variant of Post's problem with respect to truth-table reduction.

Again, we can define the truth-table degree as $\mathrm{deg}_{tt}(p) \coloneqq \{q \mid p \preceq_{tt} q \land q \preceq_{tt} p\}$.

As Figure 5.3 shows, the truth-table degree of the halting problem is not necessarily semi-decidable, and more semi-decidable predicates can be found in the truth-table degree of the halting problem. Therefore, we can ask whether there is a semi-decidable predicate A such that: $\varnothing \prec_{tt} A \prec_{tt} H$.

Similarly, by constructing predicates with more restrictions than simple predicates, also called hyper-simple predicates, Post proved the existence of a truth-table degree lying between the truth-table degree of H and $\varnothing$ in 1944 [52]. Forster and Jahn's 2023 paper [18] provided a synthetic construction of hyper-simple predicates as well.

Theorem 5.2 (Post, 1944) *There exists a hyper-simple predicate* A*, such that* $\varnothing \prec_{tt} A \prec_{tt} H$.

Up to this point, Post left Post's problem as an open problem at the end of his paper. Following the paths of hyper-simple predicates and Post's defined hyper-hyper-simple predicates seemed like a natural way to solve Post's problem; however, this was proven impossible [61, 41].

5.1.3 Turing Degrees

Similarly, we can define Turing degrees as follows:

$$\mathrm{deg}_T(p) \coloneqq \{q \mid p \preceq_T q \land q \preceq_T p\}$$

As Figure 5.4 demonstrates, the Turing degree of H includes all the limit computable predicates that can be reduced from the halting problem, as it contains more semi-decidable predicates than the truth-table degree of H. Post's problem asks whether there exists a semi-decidable predicate that lies between Turing degrees of H and $\varnothing$.

This problem is so challenging that it wasn't solved until 12 years later.. Before that, the closest result was achieved by Post and Kleene [37], where they proved that:

Theorem 5.3 (Kleene-Post, 1954) *There are two Turing-incomparable degrees* A *and* B*, such that* $A \preceq_T H$ *and* $B \preceq_T H$.

Since A and B are Turing incomparable, they are not Turing reducible to each other. This result shows that there exists a Turing degree that lies between the Turing

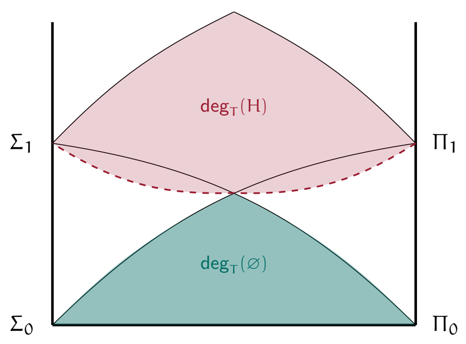

Figure 5.4: Turing Degrees

degrees of H and $\varnothing$. The only gap in solving Post's problem is that this degree is only guaranteed to be limit computable (or reducible to H), and not necessarily semi-decidable. A synthetic proof of the Kleene-Post theorem is constructed by Forster, Kirst, and Mück in 2024 [21].

5.2 Friedberg-Muchnik Theorem

Eventually, Post's problem was solved by the priority method, specifically the simplest priority method known as the finite injury method. This method establishes the property of a predicate by constructing semi-decidable predicates and verifying that they satisfy a series of requirements. During the verification process, some requirements are violated (called injuries), but the requirements all hold in the end because the number of injuries is finite. This explains the name of the method. Using this method, Friedberg and Muchnik proved the essential theorem.

Theorem 5.4 (Friedberg-Muchnik, 1956-1957) *There exist semi-decidable yet undecidable predicates A and B, such that A and B are Turing incomparable.*

This theorem shows that there are two predicates that are not Turing reducible to each other. So, at least one of the predicates is not reducible to the halting problem, and such a predicate is also semi-decidable and undecidable. Therefore, the theorem provide a positive solution to Post's problem.

We start with the requirements to be satisfied by the constructed predicates. These requirements are infinite and indexed by a natural number e. We only provide the informal intuition behind the requirements here:

$$\begin{aligned} R_{2e} &\coloneqq \text{e-th oracle machine cannot be used to reduce A to B} \\ R_{2e+1} &\coloneqq \text{e-th oracle machine cannot be used to reduce B to A} \end{aligned}$$

If there are predicates A and B, and such a requirement R_e is satisfied for all e, then for any oracle machine denoted by a code e, e can not be used to reduce A to B or B to A. Hence, these two predicates are not Turing reducible to each other.

Next, we briefly discuss the construction of these two predicates. In the finite injury method, predicates are constructed step by step (for a formal definition, see Section 6.1). At each step, an element is added, and the process at each step is computable, ensuring that the constructed predicates are semi-decidable. Our task is to control what kind of elements can be added.

Informally, in the Friedberg-Muchnik construction, an element is added to A when it can be used to destroy the possibility of some oracle machine being a reduction of B to A. The difficulty is that once such an element is added to A, as A changes, the behaviour of the oracle machines changes when A acts as an oracle. Therefore, only the element greater than the use functions (see Definition 3.30) is added at each step, i.e., that do not affect the behaviour of the oracle machines. However, this doesn't mean everything is resolved since the newly added elements can still break some other satisfied requirements, i.e., injured requirements. Then, by showing that R_e is injured at most a finite times for any e, all requirements are eventually satisfied. We point out that R_e can be only injured at most $2^e - 1$ times, which is one of the reasons why the next construction – Soare's construction is more accessible than Friedberg-Muchnik's construction.

5.3 Lerman and Soare's Construction

Lerman and Soare construct low simple predicates using the finite injury method. As the name implies, the constructed predicate is both low and simple. Simpleness implies the predicate is undecidable, semi-decidable, and not many-one reducible from H. A predicate p is low if the Turing jump of p is Turing reducible to the halting problem H. If the predicate p is low, it is not Turing reducible from H. Otherwise, by the transitivity of Turing reducibility, the Turing jump of this predicate would be Turing reducible to itself, which contradicts the property of Turing jump (see Lemma 3.18). We will give a formal discussion thereof in the next chapter (see Section 6.3). Hence, lowness implies that H is not Turing reducible to p. This means that the existence of low simple predicates provides a positive solution to Post's problem with respect to Turing degrees. Thus the following theorem is a satisfying answer to Post's problem.

Theorem 5.5 (Lerman-Soare, 1980) *There exists a low simple predicate A.*

The construction is done by extending the predicates step by step. The requirements R_e are defined via positive requirements P_e and negative requirements N_e.

Positive requirements stay satisfied in later stages once satisfied at step n. Negative requirements N_e will be finitely injured at some steps, but are eventually satisfied. Informally, the requirements are:

$$\begin{aligned} P_e &\coloneqq \text{the } e\text{-th semi-decidable predicate intersects with } A \\ N_e &\coloneqq \text{the } e\text{-th oracle machine with oracle } A \text{ terminates on } e \text{ if} \\ &\ \text{it terminates on infinitely many approximations} \end{aligned}$$

The first list of requirements ensures that the constructed predicate is simple, as it cannot contain any infinite semi-decidable predicate. Showing the lowness of a predicate p is difficult; instead of directly showing that the Turing jump p' is Turing reducible to the halting problem, N_e requirements show that p' is limit computable. This implies that p' is Turing reducible to H by the limit lemma (see Section 4.3).

As we will see in the next chapter (see Section 6), the crucial part of the verification is that N_e can only be injured a finite number of times, more precisely, at most $2 \cdot e$ times. This property makes this construction more accessible than the Friedberg-Muchnik construction.

Chapter 6

Low Simple Predicates

We have discussed the solution to Post's problem in the last chapter. In this chapter, we follow the solution by Lehmann and Soare [44], constructing low simple predicates, which, as discussed in the previous chapter, requires the priority method. The idea behind the priority method is to build a semi-decidable predicate step by step. At each step, at most one element is added to the predicate by a computable process. The constructed predicate should then be verified to meet infinite requirements. In other words, the priority method consists of construction and verification.

With the intention of developing a general framework that supports various constructions in synthetic computability, we introduce a modular approach that abstracts away the concrete construction. This abstraction avoids the tedious reasoning required by direct definitions. For instance, as shown in Figure 6.1, our construction of low simple predicates can be divided into three modules. First, we construct predicates that are semi-decidable under arbitrary *extension*. Then, we add an extension parameterised by a *wall*, so the predicate becomes simple. Finally, a low wall is inserted to achieve lowness.

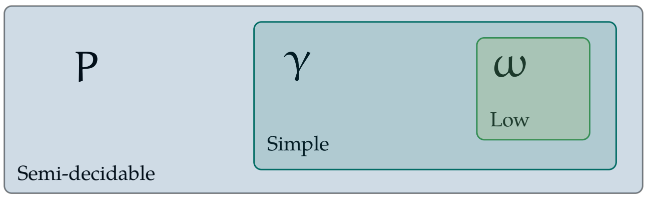

Figure 6.1: Overview of the construction of the low simple predicate.

In Section 6.1, we will develop this modular approach to the priority method in synthetic computability and demonstrate how to construct concrete examples. In

Section 6.3.1, we will use our method to construct simple predicates, following Lerman and Soare's method [44, 62] and using the framework for infinite predicates developed by Jahn and Forster [21]. We show the existence of low simple predicates by using the properties of step-indexed oracle computability in Section 6.3. Finally, we will discuss the result and the reverse mathematics analysis of it.

6.1 The Priority Method

The priority method constructs desired semi-decidable predicates with a dynamic view of semi-decidability. Instead of using logical properties to define a predicate, a semi-decidable predicate is considered to be a predicate that can be enumerated by an enumerator. These enumerators can then be defined recursively on the steps. We discuss this construction using an inductive predicate and abstract the process of deciding what will be added at each step to a predicate. This predicate takes all the information from the n-th stage to determine whether an element will be added at the n-th stage. In this way, we can abstract the decision process from the construction to analyse the constructed predicate.

6.1.1 Construction

We start by describing the properties that should be satisfied by an *extension* used to construct semi-decidable predicates.

Definition 6.1 (Extensions) *A predicate $\gamma : \mathbb{N} \to \mathbb{N}^* \to \mathbb{N} \to \mathbb{P}\mathrm{rop}$ is an extension if it is computable and unique, formally, for any $n : \mathbb{N}$ and $L : \mathbb{N}^*$, the following properties hold:*

$$(\Sigma x.\, \gamma_n^L\, x) + (\forall x.\, \neg\, \gamma_n^L\, x)$$
$$\forall x\, y.\, \gamma_n^L\, x \to \gamma_n^L\, y \to x = y$$

This means that, for any given L and n, either there exists a unique element x such that $\gamma_n^L\, x$, or for all x it is $\neg\gamma_n^L\, x$. An extension can be used to decide at step n whether an element should be added, i.e., we define how an extension finitely extends a predicate.

Definition 6.2 *Given any extension $\gamma : \mathbb{N} \to \mathbb{N}^* \to \mathbb{N} \to \mathbb{P}\mathrm{rop}$, the construction of the priority method, is defined as an inductive predicate $\rightsquigarrow : \mathbb{N} \to \mathbb{N}^* \to \mathbb{P}\mathrm{rop}$:*

$$\frac{}{0 \rightsquigarrow [\,]} \qquad \frac{n \rightsquigarrow L \quad \gamma_n^L\, x}{n+1 \rightsquigarrow x :: L} \qquad \frac{n \rightsquigarrow L \quad \forall x.\, \neg\, \gamma_n^L\, x}{n+1 \rightsquigarrow L}$$

Based on this construction, we define the predicate constructed by the extension γ as:

$$p_\gamma\, x \;:=\; \exists n\, L.\, (n \rightsquigarrow L) \land (x \in L)$$

As shown in Figure 6.2, a unique list L is associated with each step n. We call L the n-th stage if $n \rightsquigarrow L$. From step n to $n+1$, the stage either remains unchanged or gains a new element. Since the process of each step is computable, the cumulative stage can be computed. Therefore, checking whether a stage includes an element is semi-decidable, implying that p_γ is semi-decidable.

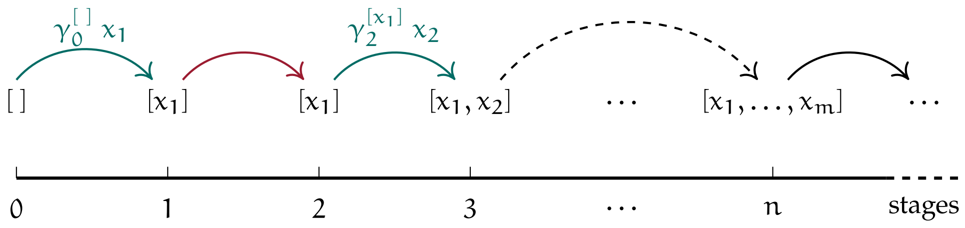

Figure 6.2: The Priority Method

Based on this intuition, we verify semi-decidability of our construction, starting with facts about the construction.

Fact 6.3 *For any extension γ, stages are unique, i.e., for any n and lists L_1, L_2:*

$$n \rightsquigarrow L_1 \to n \rightsquigarrow L_2 \to L_1 = L_2$$

Fact 6.4 *For any extension γ, stages are cumulative, i.e., for any $n_1, n_2 : \mathbb{N}$:*

$$n_1 \rightsquigarrow L_1 \to n_1 \rightsquigarrow L_2 \to n_1 \leqslant n_1 \to L_1 \subseteq L_2$$

For each joined element, the stage to which it was added is obtained by the following lemma.

Fact 6.5 *For any extension γ, if $n \rightsquigarrow x :: L$, then there must be a step m, where $m \leqslant n$, at which x is added:*

$$n \rightsquigarrow x :: L \to \exists m.\ m \leqslant n \land m \rightsquigarrow L \land \gamma_m^L\ x$$

Proof By induction on $n \rightsquigarrow x :: L$, the remainder is straightforward. □

Since extensions are computable, we can extract a function that computes the stage for any step:

Lemma 6.6 (Stage Functions) *For any extension γ, there is a stage function $\Gamma : \mathbb{N} \to \mathbb{N}^*$ such that:*

$$n \rightsquigarrow \Gamma_n \ \textit{ and } \ |\Gamma_n| \leqslant n$$

Proof The function Γ can be defined as follows:

$$\begin{aligned}\Gamma_0 &:= [\,] \\ \Gamma_{n+1} &:= \text{if } \gamma_n^{\Gamma_n}\ x \text{ then } x :: \Gamma_n \text{ else } \Gamma_n\end{aligned}$$

By induction on n, we then verify that Γ_n is the n-th stage. □

Based on the correctness of the stage function Γ, we have:

$$p_\gamma\ x \leftrightarrow \exists n.\ x \in \Gamma_n$$

Theorem 6.7 *For any extension γ, the predicate p_γ is semi-decidable.*

Proof Since it is decidable whether an element is inside a list or not, let $f\ x\ n\ := x \in \Gamma_n$ be the semi-decider, it is easy to see that $p_\gamma\ x \leftrightarrow \exists n.\ f\ x\ n$. Note that f can be turned into a Σ_1 approximation of p_γ, for simplicity of writing, we also use Γ_n to denote the Σ_1 approximation. □

Therefore, as shown in the Figure 6.3, we finished the first step of the construction: For any extension γ, the predicate p_γ is semi-decidable. Next we are free to construct a wide variety of semi-decidable predicates with a choice of extension.

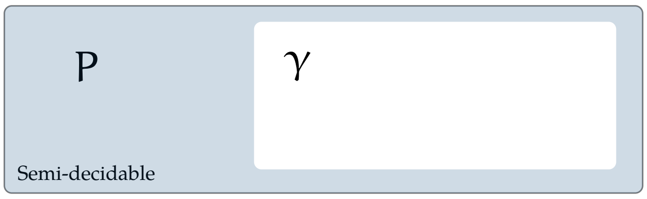

Figure 6.3: A semi-decidable predicate based on the finite extension method.

In addition to the construction, in the priority method, we will also provide a sequence of requirements $R : \mathbb{N} \to (\mathbb{N} \to \mathbb{P}\text{rop}) \to \mathbb{P}\text{rop}$ and prove that the constructed predicate of extension γ satisfies these requirements, i.e., $\forall e.\ R_e\ p_\gamma$. And for the method we use, the simplest case of the priority method - finite injury method, where the term *finite injury* refers to the possibility that some requirements in P_e might be broken during the construction of P due to the satisfaction of other requirements, also known as *injury*. However, because the injury is finite, P will satisfy all P_e as required.

6.1.2 Concrete Example: The Halting Problem

The priority method is divided into two phases: construction, which specifies an extension, and verification, which verifies that the constructed predicate satisfies a sequence of requirements. We illustrate this process by constructing a trivial example – the halting problem.

Construction

We first consider a concrete predicate ϑ that accepts an element x at step n if x is the least element such that x-th partial function θ_x halts at x within n steps, formally:

$$\vartheta^L_n\ x \ \coloneqq\ x \text{ is } \mu\, y.\ \exists v.\ \theta_y(y) \downarrow_n v$$

The notation x is $\mu\, y.\ p\ y$ denotes that x is the least element that satisfies the predicate p as follows:

$$x \text{ is } \mu\, y.\ p\ y \ \coloneqq\ p\ x \land \forall y < x.\ \neg p\ y$$

Due to the uniqueness of the least element and the decidability of whether a partial function halts within a number of steps, we can prove that this ϑ is an extension. Thus, p_ϑ is semi-decidable by Theorem 6.7.

Verification

To demonstrate the verification process, our example is very simple. The number of injuries to requirements is 0, meaning that once a requirement is satisfied, it will always be satisfied for a larger number of steps. To establish the properties we need, requirements R are as follows:

$$R_e\ p \ \coloneqq\ (\theta_e\ e \downarrow)\ \to\ e \in p$$

Now we prove that p_ϑ satisfies R_e. Notice that R_e can be sorted by its index e, indicating priorities:

$$R_0 > R_1 > \ldots > R_n > \ldots$$

This is the origin of the name "priority method". The least predicate μ in the extension is essential, as it ensures that at any given n, if multiple e satisfy the condition $\exists v.\ \theta_e(e)\downarrow_n v$ (meaning these requirements R_e *receive attention* at stage n), we always choose the one with the highest priority, i.e., the one with the smallest e, to trigger the extension of the stage (i.e., to *activate* R_e).

For example, at some n, if e is the smallest such that $\exists v.\ \theta_e(e)\downarrow_n v$ holds, then by adding e to p_ϑ, R_e holds. Since we do not remove any elements, R_e will always hold. Thus, we need to prove that for all e, R_e will be activated at least once.

The sketch of the proof is as follows: for a given e, we choose the step k such that all $R_{e'}$ have been activated for $e' < e$. Since $\theta_e\ e \downarrow$, we have a step k' such that $\exists v.\ \theta_e(e)\downarrow_{k'} v$. Then, for the step $s = \max(k, k')$, we know that R_e will receive attention and is the smallest element that satisfies the condition. Thus, at step $s+1$, R_e is satisfied by p_ϑ.

Since p_ϑ satisfies R_e and has $\exists\, L\, n.\, \vartheta^L_n\, x$ for all elements x in p_ϑ, we get $p_\vartheta\, x \leftrightarrow \theta_x\, x \downarrow$, and hence p_ϑ is the halting problem. While this is a relatively boring example, it demonstrates the basic construction ideas. We will see how to construct more interesting predicates in following sections.

6.2 Simple Extension

Post defined and constructed simple predicates [52] to demonstrate the existence of semi-decidable yet undecidable predicates that are not many-one reducible from the halting problem. As an initial attempt at answering Post's problem, simple predicates provide a valuable template for constructing undecidable predicates.

We start with simple predicates, following Forster and Jahn's definition in synthetic computability [18]. They defined and constructed a simple predicate and proved some properties related to it:

Definition 6.8 (Finiteness) *A predicate* $p : \mathbb{N} \to \mathbb{P}\mathrm{rop}$ *is finite if there is a list that lists all elements in* P.

$$\mathcal{L}(p) \;:=\; \exists L.\, \forall x.\, p\, x \leftrightarrow x \in L$$

A non-finite predicate p is a predicate that is not finite, i.e., $\neg\mathcal{L}(p)$.

Definition 6.9 (Simple Predicates) *A predicate* p *is simple if* p *is semi-decidable, and its complement* $\overline{p}$*, is non-finite and does not have a semi-decidable infinite subpredicate.*

$$\mathrm{simple}\; p \;:=\; \mathcal{S}(p) \wedge \neg\mathcal{L}(\overline{p}) \wedge \neg\exists q.\, \mathcal{S}(q) \wedge \neg\mathcal{L}(q) \wedge q \subseteq \overline{p}$$

Theorem 6.10 *Simple predicates are undecidable.*

Proof To prove the undecidability, we assume that a simple predicate p is decidable. Then, the complement of p is also decidable and non-finite, meaning that $\overline{p}$ contains a non-finite and semi-decidable predicate – itself, which contradicts the definition of simple predicates. □

6.2.1 Requirements

In Section 5.3, we have discussed the requirements of low simple predicates. To satisfy the simpleness part, the constructed predicate has to satisfy the requirements P_e, which states that the e-th semi-decidable predicate intersects with p, i.e., $\mathcal{W}_e \# p$ as in the following definition:

Definition 6.11 (Disjointness) *For any list* L*, the disjointness of* L *and the predicate* p *is defined as:*

$$L \# p \;:=\; \forall x.\, x \in L \to \neg p\, x$$

This definition is used to check whether a list L is disjoint from a predicate p, and it is decidable when the predicate p is decidable. Based on that, we can define the requirements for simple predicates as follows:

$$P_e\ A \coloneqq \neg\mathcal{L}\ (\mathcal{W}_e) \to \neg\mathcal{W}_e \# A$$

A predicate that satisfies these requirements with some additional restrictions is a simple predicate.

Lemma 6.12 *If there is an extension γ, such that p_γ satisfies the requirement P_e for all e, then p_γ is simple if $\overline{p_\gamma}$ is non-finite.*

Proof Since any predicate constructed by the priority method is semi-decidable, and the non-finiteness of the complement predicate comes from the premise, we only need to show that $\overline{p_\gamma}$ does not contain any infinite semi-decidable predicate. By the definition of the requirements, we have $\neg\mathcal{W}_e \# p_\gamma$ for any non-finite semi-decidable predicate, and the rest is straightforward. □

6.2.2 Construction

The idea behind the simple extension is that for any step n and the n-th stage, we check the finite fragment of the first n semi-decidable predicates – elements accepted within the first n steps of the decider. Then, if there exists such a finite fragment of a semi-decidable predicate that is still disjoint from the current stage and includes an element large enough, we add this element to the current stage. Notice that when multiple predicates satisfy this requirement, we only focus on the one with the smallest code; if multiple elements fulfil this requirement, we also only consider the least. As this extension is not straightforward, so we break it down into several steps.

Recall that our notation $\mathcal{W}_e[n] : \mathbb{N} \to \mathbb{P}\mathrm{rop}$ represents the decidable predicate obtained by running n steps of the semi-decider for the e-th semi-decidable predicate introduced by the EPF. This notation also denotes a Σ_1 approximation of the e-th semi-decidable predicate via index n.

Definition 6.13 (Simple Extension) *At any stage n and for any list L, the extension checks whether L intersects with the first n elements of the e-th semi-decidable predicate $\mathcal{W}_e$ or not. If there are such e and x that $x \in \mathcal{W}_e[n]$ and $2e < x$, we pick the least e and the corresponding least x as the next element to add to the predicate.*

$$\begin{aligned}
\alpha_n\ e\ x &\coloneqq x \in \mathcal{W}_e[n] \land 2e < x \\
\beta_n^L\ e &\coloneqq L \# \mathcal{W}_e[n] \land \exists x.\ \alpha_n\ e\ x \\
\gamma_n^L\ x &\coloneqq \exists e.\ e < n \land e \textit{ is } \mu\, e.\ \beta_n^L\ e \land x \textit{ is } \mu\, x.\ \alpha_n\ e\ x
\end{aligned}$$

In this definition, $\beta_n^L\ e$ indicates that the e-th semi-decidable predicate may satisfy the requirement, which means that the corresponding least element always exists as the following fact shows.

Fact 6.14 $\beta_n^L\ e \to \exists x.\ x\ \mathit{is}\ \mu\ x.\ \alpha_n\ e\ x$

Proof By the definition of $\beta_n^L\ e$, we have $\exists x.\ \alpha_n\ e\ x$. The least element always exists by a linear search function and the decidability of α. □

Lemma 6.15 *The predicate γ defined above is an extension function (in the sense of Definition 6.1).*

Proof Based on the definitions above, predicates α, β, and γ contain only decidable predicates, such as $\mathcal{W}_e[n]$ and $L\ \#\ \mathcal{W}_e[n]$, and all variables are bound, so it can be verified that these predicates are decidable. The uniqueness comes from the last predicate. □

While the proof of the priority method is tedious, the intuition behind it is simple. By abstracting two essential predicates, the following verification process focuses only on how these two operations change with the number of steps.

The first predicate describes a requirement that receives attention; we directly specify the requirement with the highest priority (i.e., with the smallest e). In this case, this requirement will be activated when we proceed to the next step.

Definition 6.16 (Receiving Attention) *The requirement P_e receives attention at the n-th step if $\mathbb{R}_n\ e$ holds, where:*

$$\mathbb{R}_n\ e \coloneqq e < n \land e\ \mathit{is}\ \mu\ e.\ \beta_n^{\Gamma_n}\ e$$

The predicate $\beta_n^{\Gamma_n}\ e$ indicates that the e-th semi-decidable predicate is to receive attention, as long as $e < n$, there must be a possibly smaller e receiving attention.

Fact 6.17 $e < n \to \beta_n^{\Gamma_n}\ e \to \exists e' \leqslant e.\ \mathbb{R}_n\ e'$

Proof Similarly to Fact 6.14, the least element can be obtained by existence and decidability. □

Fact 6.18 *For any e and n, the predicate $\mathbb{R}_n\ e$ is decidable.*

Proof By the structure of the definition, each part is decidable, and the combination is as well. □

Fact 6.19 $\forall e\ e'.\ \mathbb{R}_n\ e \to \mathbb{R}_n\ e' \to e = e'$

Proof By the uniqueness of the least element, if two elements satisfy the requirement, they must be the same. □

The second predicate we care about is act, meaning that this requirement has been activated or, that it can no longer receive attention.

Definition 6.20 (Acting) *The requirement P_e acts at n-th step if $\mathbb{A}_n\ e$ holds, where:*

$$\mathbb{A}_n\ e \coloneqq \neg\ \Gamma_n \# \mathcal{W}_e[n]$$

This definition of $\mathbb{A}_n\ e$ states that the e-th predicate already intersects with the n-th stage of the constructed predicate, so we do not need to consider adding elements to satisfy the requirement any more. This implies that the final constructed predicate intersects with the e-th predicate as well.

Fact 6.21 $\mathbb{A}_n\ e \to \neg\mathcal{W}_e \# p_\gamma$

Proof Since the construction of p_γ is cumulative by Fact 6.23, if the e-th predicate intersects with the n-th stage, it will also intersect with the final stage. □

Fact 6.22 *For any e and n, $\mathbb{A}_n\ e$ is decidable.*

Proof By the structure of the definition, each part is decidable, and the combination is as well. □

6.2.3 Verification

Satisfaction To prove that the constructed predicate p_γ is simple, we need to show that p_γ satisfies the requirements P_e for all e. We start with the basic properties of predicates $\mathbb{R}$ and $\mathbb{A}$.

Fact 6.23 *For any e and n, once a requirement acts, it always acts for a larger number of steps:*

$$\mathbb{A}_n\ e \to \forall m \geqslant n.\ \mathbb{A}_m\ e$$

Proof If the requirement P_e acts, there is an element x such that $x \in \Gamma_n$ and $x \in \mathcal{W}_e[n]$, by the cumulative property of the construction, x remains in the predicate for larger steps m, so that $x \in \Gamma_m$ and $x \in \mathcal{W}_e[m]$ as well. □

Fact 6.24 *Once a requirement P_e receives attention at n-th stage, this requirement will act for a larger number of steps, formally:*

$$\mathbb{R}_n\ e \to \forall m > n.\ \mathbb{A}_m\ e$$

Proof If the requirement P_e receives attention at step n, there is an element x such that $\gamma_n^{\Gamma_n}\ e\ x$ will be added to the predicate. Thus, P_e will act at step $n+1$. By the above Fact 6.23, P_e will act at a larger step. □

Fact 6.25 *If a requirement acts, it no longer receives attention:*

$$\mathbb{A}_n\ e \to \neg\mathbb{R}_n\ e$$

Proof If a requirement P_e acts at step n, the predicate $\alpha_n\ e$ does not hold for any step n. Therefore, the requirement will not act at any step. □

By combining above facts, we can prove the following corollary:

Corollary 6.26 $\mathbb{R}_n\ e \to \forall m > n.\ \neg\mathbb{R}_m\ e$

Next, properties wrapped in double negation need to be established:

Fact 6.27 $\forall e.\ \neg\neg\ \forall^\infty n.\ \neg\mathbb{R}_n\ e$

Proof To prove this negative goal, we perform a case analysis on $\exists n.\ \mathbb{R}_n\ e$. If such an n exists, then P_e act at subsequent steps, and $\neg\mathbb{R}_m$ holds for all $m > n$. The other case is trivial. □

Lemma 6.28 $\forall e.\ \neg\neg\ \exists s.\ \forall e' < e.\ \forall s' > s.\ \neg\mathbb{R}_{s'}\ e'$

Proof Since e' is bounded by e, we proceed by induction on e and apply Fact 6.27. □

By the definition of requirements P_e, we only need to consider non-finite predicates $\neg\mathcal{L}\ (\mathcal{W}_e)$, for which we establish the following property:

Fact 6.29 $\neg\mathcal{L}\ (\mathcal{W}_e) \to \neg\neg\exists n\ x.\ \alpha_n\ e\ x$

Proof Consider a non-finite $\mathcal{W}_e$, which ensures that there exists an n such that a sufficiently large x is in $\mathcal{W}_e[n]$ under double negation. Since our goal is to prove a negative goal, we are able to extract the n and x to obtain a large enough witness that makes $\alpha_n\ e\ x$ hold. □

To conclude, we need one more step to ensure that all requirements will act:

Corollary 6.30 $\neg\mathcal{L}\ (\mathcal{W}_e) \to \neg\neg\ (\exists n.\ \mathbb{R}_n\ e \vee \mathbb{A}_n\ e)$

Proof For any given e, as stated by Lemma 6.28, there is a step s that satisfies $\forall e' < e.\ \forall s' > s.\ \neg\mathbb{R}_{s'}\ e'$. Given that $\neg\mathcal{L}\ (\mathcal{W}_e)$, as shown by Fact 6.29, there exists a step n such that $\alpha_n\ e\ x$ is satisfied. Let m be the maximum of s, n, and e. If $\mathcal{W}_e[m]\ \#\ \Gamma_m$, then $\mathbb{A}_m\ e$ holds. Otherwise, the requirement P_e is the highest priority requirement. Consequently, this results in $\mathbb{R}_m\ e$ being satisfied. □

Fact 6.31 $\neg\mathcal{L}\ (\mathcal{W}_e) \to \neg\neg\ (\exists n.\ \mathbb{A}_n\ e)$

Proof By the above Fact, if $\mathbb{R}_n\ e$ holds for a given m, pick $n := m+1$ and apply Fact 6.24. □

Theorem 6.32 *The predicate p_γ satisfies requirements P_e for all e:*

$$\forall e.\ P_e\ p_\gamma$$

Proof Consider a non-finite predicate $\mathcal{W}_e$, of which we know $\neg\neg\mathbb{A}_n\ e$ from the given Fact 6.31. Since the goal of our proof is to establish a negative proposition, $\neg\mathcal{W}_e \# p_\gamma$, we can then use the premise $\mathbb{A}_n\ e$ to finish the proof by Fact 6.21. □

Non-Finiteness The non-finiteness of $\overline{p_\gamma}$ comes from the fact that if there are n requirements acting in some stage, then this stage contains at most n elements with the maximum element greater than $2n$. To extract this information from the constructed predicate, we define the following predicate:

Definition 6.33 (Selected) *An element x is selected by the requirement*

$$\mathbb{S}_e\ x := \exists n.\ \mathbb{R}_n\ e \land \gamma_n^{\Gamma_n}\ x$$

The predicate $\mathbb{S}_n\ x$ picks the element x used to make P_e act; for a fixed e, the selected elements are unique.

Fact 6.34 $\forall x\ x'.\ \mathbb{S}_e\ x \to \mathbb{S}_e\ x' \to x = x'$

Proof By the uniqueness of the least predicate. □

The selected predicate $\mathbb{S}_e\ x$ maps the acting P_e to the chosen element x, so the following properties can be extracted from the constructed predicate.

Fact 6.35 *For any $e, x : \mathbb{N}$:*

$$p_\gamma\ x \land x \leqslant 2n \to \exists e.\ \mathbb{S}_e\ x \land e < n$$

Proof By $p_\gamma\ x$, we know that there exists an n such that $x \in \Gamma_n$, and induction is performed on Γ_n. The case of the empty list is trivial. For the list $a :: L$, if $x \neq a$, then we are done by the induction hypothesis. If $x = a$, then we can find the step m at which a is added by the property of the priority method's construction, and know that it satisfies a certain extension $\gamma_m^{\Gamma_m}\ a$ of e. By the definition of the extension, we can prove the selected predicate $\mathbb{S}$ and that $e < n$. □

We can then generalise this property to arbitrary lists.

Fact 6.36 *For any list $L : \mathbb{N}^*$:*

$$(\forall x.\ x \in L \to x \in p_\gamma \land x \leqslant 2n) \to (\forall x.\ x \in L \to \exists e.\ \mathbb{S}_e\ x \land e < n)$$

Proof Induction on the list L and apply the above Fact 6.35. □

Theorem 6.37 *The complement of* p_γ *is non-finite:*

$$\neg\mathcal{L}(\,\overline{p_\gamma}\,)$$

Proof The intuition behind this proof stems from the fact that for any n, it is possible to find a list of length n contained within the predicate $\overline{p_\gamma}$. This is because Γ_n of length n must contain only n elements from the set $\{0, ..., 2n\}$. For a detailed proof, refer to Lemma 63 of Forster and Jahn [18]. □

Corollary 6.38 *The predicate* p_γ *is simple.*

Proof By Lemma 6.12, the predicate $\overline{p_\gamma}$ is non-finite by Theorem 6.37, and p_γ satisfies the requirements P_e by Theorem 6.32. □

6.3 Low Wall Functions

With the general extension scheme γ, we have constructed simple predicates in synthetic computability. However, it can be shown that these predicates are Turing-reducible from the halting problem [62]. To further enrich this construction with additional properties, we observe that parts of this extension can be parameterised by a function that meets specific conditions. We refer to such a function as a "wall function" because it determines the wall height that an added element x must overcome.

6.3.1 Wall Functions

Definition 6.39 (Wall Functions) *A wall function is defined as a function* $\omega : \mathbb{N} \to \mathbb{N}^* \to \mathbb{N} \to \mathbb{N}$ *that meets the following conditions:*

$$2 \cdot e \leqslant \omega_n^{\Gamma_n}(e)$$
$$\neg\neg\exists b.\ \lim_{n\to\infty} \omega_n^{\Gamma_n}(e) = b$$

The first condition ensures that the wall function is high enough to establish the non-finiteness as shown in Theorem 6.37. The second condition allows us to prove that the wall converges under double negation since the requirements are nagated. Based on this definition, the extension γ can be parameterised by a wall function ω as follows, where the red parts of the formulas highlight the differences between the old and new definitions:

$$\begin{aligned}
\alpha(\omega)_n^L\ e\ x &:= x \in \mathcal{W}_e[n] \land \omega_n^L(e) < x \\
\beta(\omega)_n^L\ e &:= L \,\#\, \mathcal{W}_e[n] \land \exists x.\ \alpha(\omega)_n^L\ e\ x \\
\gamma(\omega)_n^L\ x &:= \exists e.\ e < n \land e \text{ is } \mu\, e.\ \beta(\omega)_n^L\ e \land x \text{ is } \mu x.\ \alpha(\omega)_n^L\ e\ x
\end{aligned}$$

We add a parameter L to α, so that the wall function can also access information of the current stage. Notice that predicates $\mathbb{A}$, $\mathbb{R}$, and $\mathbb{S}$ that we have defined all depend on the definition of β and γ, thus the definition does not require any changes when a wall function is fixed.

Theorem 6.40 *For any wall function ω, the predicate $P_{\gamma(\omega)}$ is a simple predicate.*

Proof By introducing this change, the structure of the proof remains essentially unchanged. The primary adjustment required was in the proof of Lemma 6.28, because the convergent values of the wall function can be extracted to prove a negative goal, ensuring that the lemma remains valid. $\square$

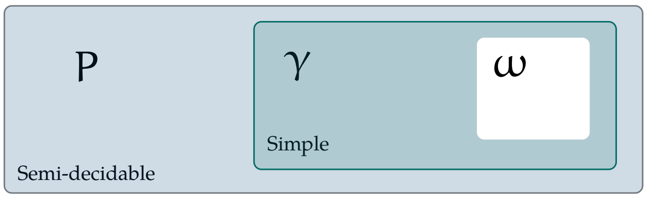

Figure 6.4: A simple predicate based on the simple extension.

Thus, as demonstrated in Figure 6.4, we have now completed the second part of the construction. For any wall function ω, we obtain a simple predicate $P_{\gamma(\omega)}$. There is a trivial example:

Example Let $\omega_n^L\ e := 2 \cdot e$, since ω is a wall function, the constructed $P_{\gamma(\omega)}$ is a simple predicate.

The classical axioms have not been used so far. Our proof is entirely constructive due to the choice of an appropriate definition of infinite lists; for further details, see the paper by Forster and Jahn [18].

6.3.2 Lowness

Before giving a concrete definition of a wall function solving Post's problem, we give a formal definition of low simple predicates. A low simple predicate is both low and simple. As the simpleness ensures that the predicate is undecidable and semi-decidable, the only missing property to solve Post's problem is that the predicate is not Turing reducible from the halting problem H, which can be derived from the lowness as follows.

Definition 6.41 (Lowness) *A predicate p is low if and only if its Turing jump is Turing reducible to the halting problem, formally:*

$$\text{low } p := p' \preceq_T H$$

Lemma 6.42 *For any low predicate* p, p *is not reducible from the halting problem:*

$$H \not\preceq_T p$$

Proof By the definition of low predicates, if both $H \preceq_T p$ and $p' \preceq_T H$ hold, transitivity of Turing reduction gives $p' \preceq_T p$, which contradicts Lemma 3.18. $\square$

A low simple predicate is a predicate that is both low and simple. We conclude:

Corollary 6.43 *The existence of low simple predicates gives a positive solution to Post's problem.*

6.3.3 Requirements

The requirements of lowness claim that the e-th oracle machine with oracle A terminates on e if it terminates on infinitely many approximations.

$$N_e\ A \coloneqq \exists^\infty n.\ \phi_e^A(e)[n] \downarrow \to \phi_e^A(e) \downarrow$$

We then show the connection between these requirements and lowness:

Lemma 6.44 *If there is an extension* γ*, such that* p_γ *satisfies the requirement* N_e *for all* e*, then the Turing jump of* p_γ *is limit computable if the step-indexed oracle machine* $\phi_e^{p_\gamma}(e)[n]$ *is convergent.*

Proof To show the lowness of p_γ, we need to prove that p'_γ is limit computable by the limit lemma 3.28. The limit computability of p'_γ can be witnessed by a limit decider $\phi_e^{p_\gamma}(e)[n]$. Since the step-indexed oracle machine is convergent, it is not difficult to get this fact along with its properties 3.22 and requirements N_e. $\square$

6.3.4 Construction

To construct a low simple predicate, we have to define an appropriate concrete wall function. In this section, we define the wall function as the maximum of the function $2 \cdot e$ and the use function on all inputs e' less than e. As we discussed in Section 3.2.2, the use function $u_e^A(x)[n]$ captures the highest question asked during the computation of $\Phi_e^A(x)[n]$. A critical feature of this function is that if an element greater than $u_e^A(x)[n]$ is added to the predicate A, it does not effect the outcome of the computation $\Phi_e^A(x)[n]$.

Definition 6.45 (Low Wall) *The low wall function is defined as follows:*

$$\begin{aligned} U_e^L(e)[n] &\coloneqq \max_{e' \leqslant e}(u_{e'}^L(e')[n]) \\ \omega_n^L(e) &\coloneqq \max(2 \cdot e, U_e^L(e)[n]) \end{aligned}$$

where the notation $\max_{e' \leqslant e}(f(e'))$ denotes the maximum of $f(e')$ for all e' less than or equal to e. The superscript L on a use function denotes the decidable predicate $\lambda x.\ x \in L$.

According to the definition of the wall function, it is easy to see this function is greater than $2 \cdot e$, there is only one condition that need to be verified:

Fact 6.46 *For any e, there exists a bound m such that for any n greater than m, the element added at the n-th step exceeds the value specified by the wall function. Formally stated:*

$$\neg\neg\ (\forall^\infty n.\ \gamma(\omega)_n^{\Gamma_n}\ x \to \omega_n^L(e) < x)$$

Proof By Lemma 6.28, we have $\exists s.\ \forall e' < e.\ \forall s' > s.\ \neg \mathbb{R}_{s'}\ e'$, where the double negation is eliminated as the goal is wrapped in double negation. Let the bound be $s+1$. According to the definition of $\gamma(\omega)$, an element is added to the predicate when some N_e' acts. If e' is greater than e, we obtain the conclusion by the definition of ω. If $e' \leqslant e$, it contradicts the assumption that any N_e' cannot receive attention when $e' \leqslant e$. □

The previous statements are lemmas contributing to this fact:

Fact 6.47 *The low wall function ω is convergent:*

$$\neg\neg\exists b.\ \lim_{n\to\infty} \omega_n^{\Gamma_n}(n)(e) = b$$

Proof By the above Fact 6.46, there is a bound m, such that $\forall n.\ \gamma(\omega)_n^{\Gamma_n}\ x \to \omega_n^L(e) < x$, where $m \leqslant n$. Since the goal is negative, we can do case analysis on whether for all $k > m$, the use function $u_e^{x \in L}(e)[k]$ is equal to 0. If it is the case, $2 \cdot e$ bounds the wall function. If there is a k greater than m such that $u_e^{x\in L}(e)[k]$ is equal to some non-zero number c. The wall function bound by $\max(2 \cdot e, c)$ as the result of the use function is c for any step larger than k. □

Corollary 6.48 *The function ω is a wall function (see Definition 6.39).*

So far we already show that the predicate $p_{\gamma(\omega)}$ is a simple predicate based on the definition of ω. However, in order to establish the lowness, the limit of the wall function should be obtained without double negation.

6.3.5 Verification

In order to prove lowness, we must show that step-indexed oracle machines are convergent, which requires us to eliminate the double negation that occurs inside the theorems. We check the statements where double negation have occurred previously (see Fact 6.27 and Lemma 6.28). Even though it is enough to show the simpleness, for the lowness, it is essential to note that we proved these lemmas with case analysis on $\exists n.\ \mathbb{R}_n\ e$. Since $\mathbb{R}_n\ e$ is decidable, this means that by assuming Σ_1-LEM, we can do case analysis on $\exists n.\ \mathbb{R}_n\ e$.

Lemma 6.49 $\Sigma_1\text{-LEM} \to \forall e.\ \forall^\infty n.\ \neg \mathbb{R}_n\ e$

Proof By the definition of Σ_1-LEM, the decidable predicate $\mathbb{R}_n\ e$ allows the case analysis on $\exists n.\ \mathbb{R}_n\ e$, the remainder is the same as Lemma 6.27. □

Lemma 6.50 $\Sigma_1\text{-LEM} \to \forall e.\ \exists s.\ \forall e' < e.\ \forall s' > s.\ \neg \mathbb{R}_{s'}\ e'$

Proof By induction on e and then applying Lemma 6.49. □

Based on the introduced logical axioms Σ_1-LEM, we can further eliminate the double negation in the proof of convergence of the wall function.

Fact 6.51 *For any* e*, by assuming* Σ_1-LEM*:*

$$\forall^\infty n.\ \gamma(\omega)_n^{\Gamma_n}\ x \to \omega_n^L(e) < x$$

Proof The proof is similar to Fact 6.46. Yet note that we can use Lemma 6.50 to avoid double negation since we have assumed Σ_1-LEM. □

Lemma 6.52 *For any* e*, by assuming* Σ_1-LEM*:*

$$\exists b.\ \lim_{n \to \infty} \omega_n^{\Gamma_n}(n)(e) = b$$

Proof The proof is similar to Lemma 6.47. However, we use the above Fact 6.51 with Σ_1-LEM. Moreover, a case analysis for some bound m of $\forall k > m.\ u_e^{x \in L}(e)[k] = 0$ can be performed through Π_1-LEM, as it follows from Σ_1-LEM. □

Based on the above facts, we show that the constructed predicate $p_{\gamma(\omega)}$ satisfies all the requirements and is convergent.

Theorem 6.53 *Assuming* Σ_1-LEM*, the predicate* $P_{\gamma(\omega)}$ *satisfies the requirements* N_e*.*

Proof Assume infinitely many n, such that $\phi_e^{p_{\gamma(\omega)}}(e)[n] \downarrow$. As there is a bound m such that any added element after m is greater than the use function, together with the property of the use function (see Definition 3.30), the oracle machine terminates on e if the corresponding step-indexed oracle machine terminates on e at any step after m. Since there are infinitely many steps such that the step-indexed oracle machine terminates on e, the oracle machine terminates on e. This demonstrates that the requirements P_e is satisfied. □

Since we know that the wall function is convergent for any e, that means that for any e, there exists a bound where the results of subsequent step-indexed oracle machines will no longer change, and will either always terminate or always diverge. Notice that this bound is not the bound for the convergence of the wall function, but the existence of such bound is provable under assumption Σ_1-LEM:

Theorem 6.54 *Assuming* Σ_1-LEM*, the step-indexed oracle machine* $\phi_e^{P_{\gamma(\omega)}}(e)[n]$ *is convergent.*

Proof By Fact 6.51, there is a bound m such that for any $n > m$, we have $\gamma(\omega)_n^{\Gamma_n}\ x \to \omega_n^L(e) < x$. By applying Σ_1-LEM (see Fact 4.5), if $\forall n.\ \phi_e^{P_{\gamma(\omega)}}(e)[n] = \mathsf{none}$, then this step-indexed oracle machine is convergent to none. If there is a $n > m$ such that $\forall n.\ \phi_e^{P_{\gamma(\omega)}}(e)[n] \downarrow$, based on the property of step-indexed oracle machine (see Definition 3.30), the output of it will not change after m as the wall function is convergent (see Lemma 6.52). Therefore, the step-indexed oracle machine is convergent to $\ulcorner\star\urcorner$. $\square$

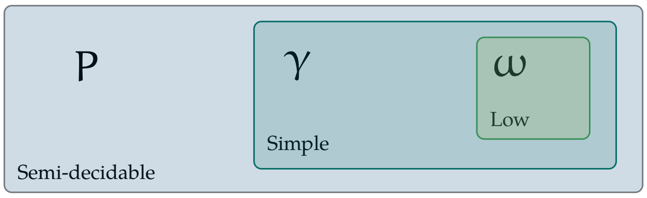

Figure 6.5: Construction of the low simple predicate.

As Figure 6.5 illustrates, with this appropriate wall function ω, we have completed the final piece of the jigsaw puzzle, and the constructed predicate is now low and simple. With all theorems established above, we can obtain the following conclusion:

Corollary 6.55 *Assuming* Σ_1-LEM*, there exists a low simple predicate.*

This result gives a positive answer to Post's problem:

Corollary 6.56 *Assuming* Σ_1-LEM*, there is a semi-decidable yet undecidable predicate that is not Turing-reducible from the halting problem.*

Since Σ_1-LEM is equivalent to LPO, we have shown that LPO is sufficient to show the existence of a low simple predicate.

Chapter 7

Conclusion

The central achievement of this thesis is the synthetic solution to Post's problem, which is accomplished by constructing low simple predicates within synthetic computability. We demonstrate how to construct these predicates using the priority method. This represents an initial step towards mechanising advanced computability, as many advanced results are based on the priority method. In addition to this core result, we define step-indexed execution and use functions for oracle machines and limit computability in synthetic computability.

In this thesis, the main difficulties come from two places. The first difficulty was that the existing definitions of oracle computability were lacking, and the second difficulty was that the priority method is simply difficult even outside a proof assistant.

To expand on this further, the problem of existing studies on synthetic oracle computability is that they only focus on properties for arbitrary oracles. To construct a solution to Post's problem, it is necessary to consider the properties of oracle machines when given a semi-decidable oracle, i.e., the step-indexing and use functions for such oracle machines. The definition of these properties needs to be specific to a certain definition of oracle computability.

In section 7.1, we give an overview of Coq development. Related and future work are presented in Sections 7.2 and 7.3.

7.1 Mechanisation in Coq

We mechanise all results in the proof assistant Coq. For all definitions, facts, lemmas, and theorems, a link to the `HTML` page of Coq development is available. A GitHub repository contain all the results can be found in following link:

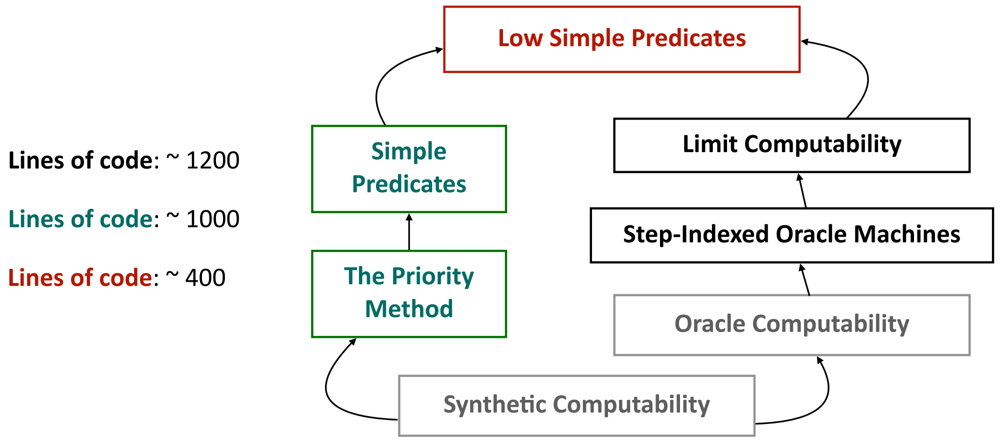

Figure 7.1: Overview of the Coq development

`https://ps.uni-saarland.de/~zeng/bachelor/coqdoc/thesis`

Our development is based on the *Synthetic Computability Theory in Coq*[20, 17, 18], a repository contributed by Forster, et al. We reuse their basic definitions of synthetic computability, oracle computability, etc. The dependencies of the whole project are shown in the Diagram 7.1: the grey part is already in the library, the green part is the priority method and related technology, the **black part** is the limit computability-related content, and finally, the red part combines all these to get the final result.

Mechanisation increases the confidence of the proofs, as we only need to trust the core rules of the proof assistant. An interesting point is that the constructions of the priority method typically follow a specific pattern. In the mechanisation of low simple predicates, we try to minimise repetition, which enhances modularity. For instance, we can reduce duplication of proofs by abstracting the extension function (see Definition 6.1) and wall function (see Definition 6.39) to manage our proofs. This process is aided by mechanisation. As shown by the following code statistics, the mechanisation of the priority method is manageable, and under our modularity approach, verifying the topmost layer – the crucial lowness – requires only 400 lines of code.

	Specification	**Proofs**
Limit Computability	66	243
Step-Indexed Oracle Machines	246	673
The Priority Method	110	321
Simple Predicates	134	488
Low Simple Predicates	88	357
Total	**644**	**2082**

7.2 Related Work

Synthetic Computability Synthetic computability was first proposed by Richmann and Bridges [53, 7], who study computability under a constructive system in the sense of Bishop [5]. Bauer further developed synthetic computability based on Hyland's effective topos [3]. They all assume the axiom of countable choice, such that the law of excluded middle cannot be assumed, otherwise one could construct a non-computable function. Forster et al. [22, 16] used CIC as a constructive system to study synthetic computability by assuming synthetic Church's thesis.

Post's Problem Post presented Post's problem in 1944 [52], and before the solution given by Friedberg and Muchnik, the closest result is the Post-Kleene theorem [37], in which they proved the existence of two limit computable predicates that are Turing-incomparable. In 1956 and 1957, Friedberg and Muchnik independently developed the priority method [27, 46], then solved Post's problem by constructing two semi-decidable predicates that are Turing-incomparable. Afterwards, more solutions were proposed, including the method we are following, i.e., the construction of low simple predicates, which is a by-product of Lerman and Soare's work in constructing d-simple set [44]. By reducing some constraints, the construction in the paper gives a low simple predicate. This method is simpler than Friedberg-Muchnik's construction, and is called the "simplest" solution in Soare's textbook [62]. Kučera proposes a solution without the priority method [39].

Forster and Jahn construct simple and hypersimple predicates in synthetic computability [18, 33]. Through these constructions, they provide a synthetic solution to Post's problem with respect to many-one and truth-table degrees. However, in this thesis, we consider Post's problem with respect to Turing degrees. In Post's paper presenting Post's problem, he already constructed simple predicates and hypersimple predicates but left the problem with respect to Turing degrees open until Friedberg and Muchnik solved it. Forster, Kirst, and Mück have mechanised the proof of the Kleene-Post theorem in synthetic computability [35]. All their results are mechanised in the Coq proof assistant.

Mechanisation of Synthetic Computability Mechanising synthetic computability begins with Forster, Kirst, and Smolka's work in the proof assistant Coq [22]. Their subsequent work establishes a Coq library of *synthetic computability* that has mechanised the arithmetical hierarchy, the Kleene-Post's theorem, and Post's hierarchy theorem, among other results [18, 21]. The results of this thesis, i.e., the limit lemma and the existence of low simple predicates, will also be included in this library. Swan's work on oracle modalities is mechanised in the proof assistant Agda [64].

Synthetic Oracle Computability The first definition of oracle computability was given by Bauer [4], defining it as a form of continuity in the setting of the intuitionistic effective topos. Later, Forster and Kirst describe a reformulation in constructive type theory [16], along with another suggestion that enables the connection of Post's theorem and the arithmetical hierarchy [19].

However, all these definitions require an axiom not derived from the common axiom of synthetic computability CT, thus leaving a gap in the consistency status. Forster, Kirst, and Mück suggest another definition of oracle computability based on sequential continuity [20], where the enumerability of oracle machines is derived from CT. In this thesis, we use this definition of oracle computability.

Swan introduces another definition of oracle computability based on higher modalities in homotopy type theory [64]. In this framework, he characterizes oracle computations through 0-truncated $\neg\neg$-sheafification.

7.3 Future Work

In this thesis, we have proved the existence of low simple predicates (see Corollary 6.55), which provides a positive solution to Post's problem in synthetic computability theory. However, Friedberg and Muchnik's original construction provides a positive solution to Post's problem by constructing two semi-decidable and Turing-incomparable degrees [27, 46]. Although we have mechanised a solution to Post's problem, it would be a natural follow-up to mechanise Friedberg and Muchnik's original construction in synthetic computability.

The two main results in this thesis – the limit lemma and the existence of low simple predicates. They are both proved by assuming the classical axiom LPO. Since LPO is sufficient to prove both theorems, from the point of view of reverse constructive mathematics, it will be an interesting question to find the classical axiom required to precisely prove this theorem, i.e., to show that some theorem (e.g. the limit lemma or existence of low simple predicates) are equivalent to this axiom under our constructive setting.

We introduce the definition of step-indexing for oracle machines (see Section 3.2). However, in synthetic computability, depending on the definition of oracle computability, one has different formulations to describe the step-indexed execution of oracle machines. In future work, one can study step-indexed execution on top of these different definitions of oracle computability [4, 64] and re-examine the construction of low simple predicates.

We have discussed the priority method in synthetic computability and have given a construction of low simple predicates. Since the priority method is a fundamental technique in computability theory, it has contributed to many significant results, such as the Sacks' splitting theorem and the Sacks' jump inversion theorem [43, 55, 56]. It has also played a role in other areas, such as effective model theory and complexity theory [2, 30]. Therefore, it would be interesting to mechanise more results in synthetic computability to further bridge the gap between mechanisation and paper proofs.

Bibliography

[1] Yohji Akama, Stefano Berardi, Susumu Hayashi, and Ulrich Kohlenbach. An arithmetical hierarchy of the law of excluded middle and related principles. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 192–201. IEEE Computer Society, 2004. doi:10.1109/LICS.2004.1319613.

[2] Eric Allender, Luke Friedman, and William I. Gasarch. Limits on the computational power of random strings. *Inf. Comput.*, 222:80–92, 2013. URL: https://doi.org/10.1016/j.ic.2011.09.008, doi:10.1016/J.IC.2011.09.008.

[3] Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. doi:10.1016/j.entcs.2005.11.049.

[4] Andrej Bauer. Synthetic mathematics with an excursion into computability theory (slide set). *University of Wisconsin Logic seminar*, 2020. URL: http://math.andrej.com/asset/data/madison-synthetic-computability-talk.pdf.

[5] Errett Bishop. Foundations of constructive analysis. 1967.

[6] Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1987.

[7] Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97. Cambridge University Press, 1987.

[8] Mario Carneiro. Lean4lean: Towards a formalized metatheory for the lean theorem prover. *arXiv preprint arXiv:2403.14064*, 2024.

[9] Liron Cohen, Yannick Forster, Dominik Kirst, Bruno da Rocha Paiva, and Vincent Rahli. Separating Markov's Principles. In *39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24)*, Talinn, Estonia, July 2024.

URL: https://inria.hal.science/hal-04584831, doi:10.1145/3661814.3662104.

[10] S. Barry Cooper. Chapter 4 - local degree theory* *preparation of this paper partially supported by e.p.s.r.c. research grants nos. gr/h91213 and gr/h02165, and by ec human capital and mobility network complexity, logic and recursion theory. In Edward R. Griffor, editor, *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, pages 121–153. Elsevier, 1999. URL: https://www.sciencedirect.com/science/article/pii/S0049237X99800202, doi:10.1016/S0049-237X(99)80020-2.

[11] Thierry Coquand. Metamathematical investigations of a calculus of constructions. Technical Report RR-1088, INRIA, 1989. URL: https://inria.hal.science/inria-00075471.

[12] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76(2/3):95–120, 1988. doi:10.1016/0890-5401(88)90005-3.

[13] Thierry Coquand and Bassel Mannaa. The independence of markov's principle in type theory. *Log. Methods Comput. Sci.*, 13(3), 2017. doi:10.23638/LMCS-13(3:10)2017.

[14] Hannes Diener and Hajime Ishihara. *Bishop-Style Constructive Reverse Mathematics*, pages 347–365. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-59234-9_10.

[15] Yannick Forster. Church's thesis and related axioms in coq's type theory. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 21:1–21:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: https://doi.org/10.4230/LIPIcs.CSL.2021.21, doi:10.4230/LIPICS.CSL.2021.21.

[16] Yannick Forster. Computability in constructive type theory. 2021. doi:10.22028/D291-35758.

[17] Yannick Forster. Parametric church's thesis: Synthetic computability without choice. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science - International Symposium, LFCS 2022, Deerfield Beach, FL, USA, January 10-13, 2022, Proceedings*, volume 13137 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2022. doi:10.1007/978-3-030-93100-1_6.

[18] Yannick Forster and Felix Jahn. Constructive and synthetic reducibility degrees: Post's problem for many-one and truth-table reducibility in Coq. In

CSL 2023-31st EACSL Annual Conference on Computer Science Logic, 2023. doi:10.4230/LIPIcs.CSL.2023.21.

[19] Yannick Forster and Dominik Kirst. Synthetic Turing reducibility in constructive type theory. 28th International Conference on Types for Proofs and Programs (TYPES 2022), 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_64.pdf.

[20] Yannick Forster, Dominik Kirst, and Niklas Mück. Oracle computability and turing reducibility in the calculus of inductive constructions. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 155–181. Springer, 2023. doi:10.1007/978-981-99-8311-7_8.

[21] Yannick Forster, Dominik Kirst, and Niklas Mück. The Kleene-Post and Posts theorem in the calculus of inductive constructions. 2024. doi:10.4230/LIPIcs.CSL.2024.29.

[22] Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in coq, with an application to the entscheidungsproblem. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 38–51. ACM, 2019. doi:10.1145/3293880.3294091.

[23] Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 114–128, 2020. doi:10.1145/3372885.3373816.

[24] Yannick Forster, Steven Schäfer, Simon Spies, and Kathrin Stark. Call-by-push-value in coq: operational, equational, and denotational theory. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 118–131. ACM, 2019. doi:10.1145/3293880.3294097.

[25] Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in coq. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2017. doi:10.1007/978-3-319-66107-0_13.

[26] Yannick Forster and Gert Smolka. Call-by-value lambda calculus as a model of computation in coq. *J. Autom. Reason.*, 63(2):393–413, 2019. URL: `https://doi.org/10.1007/s10817-018-9484-2`, `doi:10.1007/S10817-018-9484-2`.

[27] Richard M Friedberg. Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944). *Proceedings of the National Academy of Sciences*, 43(2):236–238, 1957. `doi:10.1073/pnas.43.2.236`.

[28] E. Mark Gold. Limiting recursion. *J. Symb. Log.*, 30(1):28–48, 1965. `doi:10.2307/2270580`.

[29] YUZHOU GU. (t)uring degrees, 2017. URL: `https://www.math.ias.edu/~yuzhougu/data/turing.pdf`.

[30] Leo Harrington. Recursively presentable prime models. *J. Symb. Log.*, 39(2):305–309, 1974. `doi:10.2307/2272643`.

[31] William Alvin Howard. The formulae-as-types notion of construction. In Haskell Curry, Hindley B., Seldin J. Roger, and P. Jonathan, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[32] Hajime Ishihara. Reverse mathematics in bishop?s constructive mathematics. *Philosophia Scientiae*, pages 43–59, 2006. `doi:10.4000/philosophiascientiae.406`.

[33] Felix Jahn. *Synthetic One-One, Many-One, and Truth-Table Reducibility in Coq*. PhD thesis, Bachelors thesis, Saarland University, 2020.

[34] Dominik Kirst, Yannick Forster, and Niklas Mück. Synthetic Versions of the Kleene-Post and Posts Theorem. 28th International Conference on Types for Proofs and Programs (TYPES 2022), 2022. URL: `https://types22.inria.fr/files/2022/06/TYPES_2022_paper_65.pdf`.

[35] Dominik Kirst, Niklas Mück, and Yannick Forster. Synthetic versions of the kleene-post and posts theorem.

[36] S. C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53(1):41–73, 1943. URL: `http://www.jstor.org/stable/1990131`.

[37] S. C. Kleene and Emil L. Post. The upper semi-lattice of degrees of recursive unsolvability. *Annals of Mathematics*, 59(3):379–407, 1954. URL: `http://www.jstor.org/stable/1969708`.

[38] Georg Kreisel. Mathematical logic. In Georg Kreisel, editor, *Lectures on Modern Mathematics*, pages 95–195. Wiley, 1965.

[39] Antonín Kučera. An alternative, priority-free, solution to post's problem. In Jozef Gruska, Branislav Rovan, and Juraj Wiedermann, editors, *Mathematical Foundations of Computer Science 1986*, pages 493–500, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[40] Fabian Kunze, Gert Smolka, and Yannick Forster. Formal small-step verification of a call-by-value lambda calculus machine. In Sukyoung Ryu, editor, *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*, volume 11275 of *Lecture Notes in Computer Science*, pages 264–283. Springer, 2018. doi:10.1007/978-3-030-02768-1_15.

[41] Alistair H. Lachlan. On the lattice of recursively enumerable sets. *Transactions of the American Mathematical Society*, 130:1–37, 1968. URL: https://api.semanticscholar.org/CorpusID:5622297.

[42] Alistair H Lachlan. The priority method for the construction of recursively enumerable sets. In *Cambridge Summer School in Mathematical Logic: Held in Cambridge/England, August 1–21, 1971*, pages 299–310. Springer, 2006. doi:10.1007/BFb0066779.

[43] Steffen Lempp. Priority arguments in computability theory, model theory, and complexity theory. *Lecture notes*, 2012. URL: https://people.math.wisc.edu/~slempp/papers/prio.pdf.

[44] Manuel Lerman and Robert Soare. d-simple sets, small sets, and degree classes. *Pacific Journal of Mathematics*, 87(1):135–155, 1980. doi:10.2140/pjm.1980.87.135.

[45] Andrzej Mostowski. On definable sets of positive integers. *Fundamenta Mathematicae*, 34(1):81–112, 1947. URL: http://eudml.org/doc/213118.

[46] Albert A Muchnik. On the unsolvability of the problem of reducibility in the theory of algorithms. In *Dokl. Akad. Nauk SSSR*, volume 108, pages 194–197, 1956.

[47] Michael E. Mytilinaios and Theodore A. Slaman. 2-collection and the infinite injury priority method. *The Journal of Symbolic Logic*, 53(1):212–221, 1988. URL: http://www.jstor.org/stable/2274439.

[48] Takako Nemoto. Computability theory over intuitionistic logic. Logic Colloquium 2024, European Summer Meeting of the Association for Symbolic Logic, Gothenburg, Sweden, 2024.

[49] Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992. doi:10.2307/2274492.

[50] Christine Paulin-Mohring. Inductive definitions in the system coq - rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993. URL: `https://doi.org/10.1007/BFb0037116`, `doi:10.1007/BFB0037116`.

[51] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Ctlin Hricu, Vilhelm Sjöberg, and Brent Yorgey. *Logical Foundations*, volume 1 of *Software Foundations*. Electronic textbook, 2023. URL: `http://softwarefoundations.cis.upenn.edu`.

[52] Emil L Post. Recursively enumerable sets of positive integers and their decision problems. 1944. `doi:10.1090/s0002-9904-1944-08111-1`.

[53] Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983. `doi:10.2307/2273473`.

[54] Hartley Rogers Jr. *Theory of recursive functions and effective computability*. MIT press, 1987. `doi:10.2307/2271523`.

[55] Gerald E. Sacks. On the degrees less than 0. *Annals of Mathematics*, 77(2):211–231, 1963. URL: `http://www.jstor.org/stable/1970214`.

[56] Gerald E Sacks. Recursive enumerability and the jump operator. *Transactions of the American Mathematical Society*, 108(2):223–239, 1963.

[57] J. R. Shoenfield. On degrees of unsolvability. *Annals of Mathematics*, 69(3):644–653, 1959. URL: `http://www.jstor.org/stable/1970028`.

[58] Joseph R Shoenfield. On degrees of unsolvability. *Annals of mathematics*, 69(3):644–653, 1959. `doi:10.2307/1970028`.

[59] Stephen G Simpson. Degrees of unsolvability: a survey of results. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 631–652. Elsevier, 1977. `doi:10.1016/S0049-237X(08)71117-0`.

[60] Gert Smolka. *Computational Type Theory and Interactive Theorem Proving with Coq*. 2024.

[61] Robert I Soare. Recursively enumerable sets and degrees. *Bulletin of the American Mathematical Society*, 84(6):1149–1181, 1978.

[62] Robert I Soare. Turing computability. *Theory and Applications of Computability. Springer*, 2016. `doi:10.1007/978-3-642-31933-4`.

[63] Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Win-

terhalter. The metacoq project. *Journal of automated reasoning*, 64(5):947–999, 2020.

[64] Andrew W Swan. Oracle modalities. *arXiv preprint arXiv:2406.05818*, 2024. doi:10.48550/arXiv.2406.05818.

[65] A.S. Troelstra and D. Dalen. *Constructivism in Mathematics: An Introduction*. Number Vol. I in Constructivism in Mathematics. North-Holland, 1988. URL: https://books.google.de/books?id=EubuAAAAMAAJ.

[66] Alan M. Turing. *Systems of Logic Based on Ordinals*. PhD thesis, Princeton University, NJ, USA, 1938. URL: https://doi.org/10.1112/plms/s2-45.1.161, doi:10.1112/PLMS/S2-45.1.161.