Saarland University
Faculty of Mathematics and Computer Science

Bachelor's Thesis

Mechanizing Second-Order Logic in Coq

Author
Mark Koch

Supervisor
Prof. Gert Smolka

Advisor
Dominik Kirst

Reviewers
Prof. Gert Smolka
Prof. Bernd Finkbeiner

Submitted: 23rd August 2021

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.

Saarbrücken, 23rd August, 2021

Abstract

Second-order logic is an extension of first-order logic, allowing quantifiers to not only range over individuals, but also over properties of individuals. This leads to major meta-theoretic differences compared to its first-order counterpart.

In the first part of this thesis, we analyse standard Tarski semantics for second-order logic and verify the categoricity of second-order Peano arithmetic. We refute the standard first-order model-theoretic results regarding compactness and upward Löwenheim-Skolem, and show that every sound and enumerable second-order deduction system is inherently incomplete, assuming Markov's principle and employing a reduction from the solvability of Diophantine equations (Hilbert's tenth problem). Additionally, we verify that second-order validity and satisfiability in the empty signature as well as in second-order Peano arithmetic are neither decidable, nor enumerable under Markov's principle. We make use of the synthetic approach to computability theory, showing those results up to decidability of the halting problem.

In the second part of this thesis, we switch to Henkin semantics. We prove that under this semantics, second-order logic reduces to mono-sorted first-order logic. This allows us to transport first-order compactness, Löwenheim-Skolem and completeness to Henkin semantics, restoring the model theory that failed for standard semantics.

All results in this thesis are formalized in the calculus of inductive constructions and mechanized in the Coq proof assistant.

Acknowledgements

First and foremost, I want to thank my advisor Dominik Kirst for his exceptional support and guidance throughout the development of this thesis. I am very grateful for his advice and many helpful discussions and ideas. I also appreciate Dominik's feedback and valuable suggestions on various drafts of this thesis.

I also want to thank Professor Smolka for the opportunity to write this thesis at his chair and for introducing me to the subject of computational logic. Moreover, I am grateful for his support and mentoring during my Bachelor's studies. In addition, I thank Professor Finkbeiner for reviewing this thesis.

Finally, I want to thank my family and friends for their support throughout my studies. In particular, I thank Niklas for proof-reading this thesis.

Contents

Chapter 1

Introduction

Second-order logic used to enjoy wide popularity as a foundational system for mathematics among late 18th and early 19th century logicians. While it has been gradually superseded by fist-order logic and set theory, the discussion around the merits of second-order logic is still a hot topic to this day (for example [56, 66]). On the other hand, second-order logic has found success in the domain of theoretical computer science with fruitful applications in graph theory [7] and complexity theory [11, 30].

The delineating feature between first- and second-order logic is concerned with quantification. Quantified statements are ubiquitous in mathematics, take for instance the sentence "for every natural number x there is a number y that is greater than x", which a logician would write as $\forall x.\, \exists y.\, y > x$. This is an example of universal ($\forall$) and existential ($\exists$) quantification stating that a property holds for all objects x or for at least one object y. Allowing such quantifiers that range over individuals of some domain of discourse (in our case natural numbers) results in what is known today as first-order logic.[1] However, consider the usual formulation of the principle of induction for the natural numbers: "All properties P are satisfied by all numbers if P holds for 0 and $n + 1$ whenever it holds for n." Here, we have not only quantified over numbers, but also over *properties* of numbers. Quantifying over properties (or functions, or sets) is quite natural and commonplace in everyday mathematical arguments. Extending first-order logic with those so-called second-order quantifiers results in second-order logic. This can again be extended to third-order logic encompassing properties of properties and so on, culminating in the generalization of higher-order logic and type theory.

While the conscious distinction between first- and second-order logic is essential in today's view, it certainly was not when the notion of quantifiers was originally

[1] Extending propositional logic which is only concerned the logical operations of conjunction, disjunction, implication, and negation.

devised. The contemporary use of predicate logic and ordered quantification can be traced back to the works of Frege [16, 17], as well as Russell and Whitehead [54]. Their goal was to express all of mathematics through pure logic, resulting in the programme of logicism. In his *Begriffsschrift* [16] in 1879, Frege gave the first account of a calculus for predicate logic, extending propositional logic with a notion of quantification. His system freely used second-order quantifiers and also the precursor of the theory of types developed in Russel and Whitehead's *Principia Mathematica* [54] is decisively higher-order. Similarly, Zermelo's [69] original conception of set theory was also second-order.

It was only realised later that the addition of second-order quantification, albeit seeming like a small step conceptionally, drastically alters the behaviour of the logic and makes it significantly more expressive. The most prominent example of this concerns again the natural numbers. Following the programme of Frege, there was great interest in developing an axiomatic basis for arithmetic. This task was championed by Dedekind [9] and Peano [47] who were able to give a set of axioms that uniquely characterize the natural numbers, the key ingredient being the second-order induction axiom mentioned above. In 1915, Löwenheim [40] demonstrated that this feat would not have been possible in first-order logic. In fact, he showed that without the power of second-order quantification, first-order logic is not able to uniquely characterize any infinite structure. This observation sparked great interest in first-order logic and laid the groundwork for the domain of model theory, which together with further results contributed to the prominence that first-order logic enjoys today. Wether the relative weakness of first-order logic is seen as an advantage or disadvantage of course depends on perspective, but at least it shows that the power of the second-order quantifier is really necessary to achieve Dedekind's and Peano's goals.

However, this expressive power also comes at a cost, as the model theory carefully built up for first-order logic must of course fail in the second-order case. Even more damning is the realisation that Gödels completeness theorem, stating that all valid first-order statements are provable, does no longer hold as soon as second-order quantification is involved: There is no hope for a second-order notion of provability that encompasses all true statements. The situation changed somewhat in 1950 when Henkin [26] introduced his general models that restore completeness, sacrificing the expressivity of the standard semantics. Finally, there is the critique that second-order logic depends too heavily on the set theory in which it is usually defined,[2] with Quine famously referring to it as "set theory in sheep's clothing" [49]. Proponents of second-order logic like Shapiro [56] on the other hand argue that

[2] For example, one can construct a second-order sentence that is valid iff the continuum hypothesis holds in the meta-theory. Thus, the validity of statements can depend on problems that even the underlying meta-theory cannot solve [67].

this mathematical character is not problematic and that second-order logic can in fact be regarded as a suitable foundational system.

The purpose of this thesis is to give a formal account of second-order logic and its relationship to first-order logic with a particular focus on (in)completeness and undecidability. A formal treatment of meta-logic and proof theory requires great attention to detail and at times fairly technical arguments. Carrying out such analysis in full detail on paper is tedious and quite error prone. The advent of interactive theorem proving provides an attractive alternative to this approach: So called *proof assistants* are software system that aide in the construction of formal proofs and guarantee their correctness. We refer to the process of formalizing proofs in such an assistant as *mechanization*.

We have mechanized all results in this thesis in the Coq proof assistant [1]. Thus, our analysis is carried out in Coq's logic, which is based on constructive type theory. This type-theoretic foundation we build on is different from the usual set-theoretic meta-theory employed in the literature, leading to some interesting consequences which we will discuss throughout the thesis. We also want to point out that Coq's logic is constructive. Constructivism is a field of mathematics concerned with obtaining results without relying on classical assumptions like the law of excluded middle (LEM). Since functions defined in a constructive manner are always computable this opens up a convenient alternative to the traditional characterization of computability theory: Instead of relying on a concrete model of computation, like Turing machines, computability can be framed in terms of those intrinsic computations. This approach is called *synthetic computability theory* [2, 51].

Our work lies in the context of an ongoing project to formalize computability theory in Coq using this synthetic approach. The Coq Library of Undecidability Proofs [12] is a collection of various problems that are shown to be synthetically undecidable, based on the initial Coq formalization of synthetic computability theory developed in [14]. The relevant undecidability results mechanized in thesis are intended as a contribution to this library. Additionally, we build on previous Coq formalizations of first-order logic developed in [15] and [32]. The work of Kirst and Hermes [31] investigating synthetic undecidability and incompleteness of first-order axiom systems is particularly relevant and serves as the basis for our second-order analysis of those concepts.

The full Coq development accompanying this thesis is available at:

`https://www.ps.uni-saarland.de/~koch/bachelor.php`

The definitions and theorems are hyperlinked to the formalized versions viewable online. Appendix A remarks on some differences between the mechanization and the presentation on paper.

1.1 Contributions

To the best of our knowledge, we contribute the first formalization of second-order logic mechanized in a proof assistant. We formalize well-known results, largely following the exposition by Shapiro [56] and transport them to the type-theoretic setting. Our results can be split into two distinct parts. The first half is concerned with the standard Tarski semantics, where

- We show that second-order Peano arithmetic is categorical.
- We conclude the failure of the upward Löwenheim-Skolem and compactness theorems using this categoricity result. Employing the failure of compactness, we present a concise constructive proof demonstrating that there is no deduction system that is complete for all decidable theories.
- We verify undecidability of second-order validity and satisfiability for the empty signature and in second-order Peano arithmetic via a reduction from the decidability of Diophantine equations (Hilbert's tenth problem). Furthermore, we prove that those problems are also not recursively enumerable, assuming Markov's principle.
- We obtain the full incompleteness result under Markov's principle, stating that there is no deduction system that is sound, complete and enumerable via a computability argument making use of the aforementioned enumerability results.

In the second half of the thesis we switch to Henkin semantics, where

- We show that second-order validity and provability reduce to their mono-sorted first-order counterparts, verifying a proof by Nour and Raffalli [43].
- We conclude that second-order logic with Henkin semantics is complete for a Gentzen-style second-order natural deduction system with full comprehension as a consequence of the first-order completeness theorem.
- We demonstrate that further properties like the Löwenheim-Skolem theorems and compactness can be transported from first-order logic via the presented reduction.

The reduction to mono-sorted first-order logic might also be used to easily transport future mechanizations of similar first-order model-theoretic properties to second-order logic.

1.2 Overview

After establishing some preliminary definitions in Chapter 2, we introduce the syntax and standard semantics of second-order logic, along with a natural deduction system in Chapter 3. Chapter 4 is concerned with second-order Peano arithmetic and mainly focused on the categoricity property that is then used to refute the aforementioned meta-theoretic properties for second-order logic. Chapter 5 opens with a brief overview of incompleteness and establishes various undecidability results that yield the incompleteness of second-order logic. Henkin semantics are introduced in Chapter 6 and employed in the following Chapters 7, 8, and 9 to reduce second-order to mono-sorted first-order logic. Afterwards, the reduction is used to transport completeness and further meta-theoretic properties from first- to second-order logic with Henkin semantics in Chapter 10. We discuss our results and comment on related and future work in Chapter 11. Finally, we remark on the accompanying Coq mechanization in Appendix A.

Chapter 2

Preliminaries

We begin by establishing some preliminary definitions and notions that we will use in the subsequent chapters. First, we outline the type theory used throughout this thesis. Then, we discuss the concepts of synthetic computability theory that will be employed in later chapters.

2.1 Type Theory

All results in this thesis are formalized in the framework of constructive type theory as implemented in the Coq proof assistant. Coq implements the *polymorphic calculus of cumulative inductive constructions* (pCuIC) [63, 5, 45], which features an impredicative universe $\mathbb{P}$ of propositions and a countably infinite hierarchy of predicative universes $\mathbb{T}_0 \subseteq \mathbb{T}_1 \subseteq \cdots$ with $\mathbb{P} \subseteq \mathbb{T}_0$. We usually omit the universe level and just write $\mathbb{T}$ for all $\mathbb{T}_i$. The type universes $\mathbb{T}$ contain computational information whereas predicates typically live in $\mathbb{P}$, where elimination into $\mathbb{T}$ is only allowed in restricted cases.

We use dependent function types $\forall x : X.\, f\,x$ with $f : X \to \mathbb{T}$ where the return type of the function depends on the argument x. The ordinary function type from type X to Y, denoted by $X \to Y$, is represented as $\forall\_ : X.\, Y$. We denote such functions as $\lambda x : X.\, t$. Dependent function types also double as universal quantification $\forall x : X.\, p\,x$ and logical implication $X \to Y$ for $p : X \to \mathbb{P}$ and $Y : \mathbb{P}$. Furthermore, we use falsity ($\bot$), logical conjunction ($\land$), logical disjunction ($\lor$), logical equivalence ($\leftrightarrow$) and existential quantification ($\exists x.\, p\,x$) all placed in $\mathbb{P}$ (with elimination of $\lor$ and $\exists$ blocked).

Coq also supports inductive type definitions. Figure 2.1 shows common inductive types used throughout the thesis. $\mathbb{B}$ is the type of Booleans with the truth values true and false. Natural numbers are defined as the type $\mathbb{N}$ with the zero constant $0 : \mathbb{N}$ and the successor function $S : \mathbb{N} \to \mathbb{N}$. We use the usual decimal notation to refer to numbers. Furthermore, we make use of a Cantor pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and corresponding projections π_1 and π_2, embedding pairs of natural numbers into the natural numbers. Pairs $X \times Y$ contain a value of type X and a value of type Y, whereas the sum type $X + Y$ may contain either one. The option type $\mathcal{O}(X)$ either contains an element $x : X$, denoted by $^\circ x$, or is $\emptyset$ which represents the absence of a value.

We write $\mathcal{L}(X)$ for the type of lists over X. They are constructed from the empty list nil and the cons operator $\_ :: \_ : X \to \mathcal{L}(X) \to \mathcal{L}(X)$. We use the notation $[x_1, ..., x_n]$ to refer to the list $x_1 :: ... :: x_n :: \mathsf{nil}$. Additionally, we write $A +\!\!+ B$ for concatenation, $x \in A$ for membership, $|A|$ for length and $A \subseteq B$ for inclusion. We overload the function application notation such that $f\,A$ denotes the point-wise application of a function $f : X \to Y$ to a list $A : \mathcal{L}(X)$. Finally, we use list comprehension, written as $[\, f\,x_1 \dots x_n \mid x_1 \in A_1, ..., x_n \in A_n, p\,x_1 \dots x_n \,]$ where $f : X_1 \to ... \to X_n \to Y$, $A_i : \mathcal{L}(X_i)$ and $p : X_1 \to ... \to X_n \to \mathbb{B}$. This operation returns a list containing the values $f\,x_1 \dots x_n$ for all combinations of $x_i \in A_i$ that satisfy the Boolean test p.

Vectors can be seen as lists of fixed length. We denote a vector over X of length $n : \mathbb{N}$ as the type X^n. Vectors are constructed from the empty vector $\mathsf{nil} : X^0$ and the cons operation $:: : X \to X^n \to X^{S\,n}$. We overload the previously defined operations and notations for lists to vectors.

Finally, sigma types $\Sigma x.\, p\,x$ represent dependent pairs where the type of the second component depends on the value of the first one. They are the counterpart of dependent function types and their components can always be obtained. This is different from the existential quantifier $\exists x.\, p\,x$ whose witness is only accessible when constructing a proof of a proposition in $\mathbb{P}$.

$$
\begin{aligned}
\mathbb{B} : \mathbb{T} &:= \mathsf{true} \mid \mathsf{false} && \\
\mathbb{N} : \mathbb{T} &:= 0 \mid S\,n && (n : \mathbb{N}) \\
X \times Y &:= (x, y) && (x : X, y : Y) \\
X + Y &:= L\,x \mid R\,y && (x : X, y : Y) \\
\mathcal{O}(X) &:= {}^\circ x \mid \emptyset && (x : X) \\
\mathcal{L}(X) &:= \mathsf{nil} \mid x :: A && (x : X, A : \mathcal{L}(X)) \\
X^n &:= \mathsf{nil} : X^0 \mid (x :: A) : X^{S\,n} && (x : X, n : \mathbb{N}, A : X^n) \\
\Sigma x : X.\, p\,x &:= (x, y) && (x : X, p : X \to \mathbb{P}, y : p\,x)
\end{aligned}
$$

Figure 2.1: Inductive type definitions.

2.2 Synthetic Computability Theory

Usually, computability theory is defined in terms of a concrete model of computation, for example Turing machines or the λ-calculus. However, constructive type theory allows for an interesting alternative to this approach: Every function that can be defined in a constructive theory must also be computable. In terms of (axiom-free) Coq this means that in order to show that a function is computable it suffices to give an implementation of it. The approach of relying on this inherent notion of computation is called "synthetic computability theory" [2, 14, 51]. Working in this setting is significantly less tedious than formalizing results in an explicit model of computation, where all low-level details that are usually left out on paper would need to be mechanized. In this section, we present the main notions of synthetic computability following Forster, Kirst and Smolka [14].

2.2.1 Basic Definitions

We begin by defining the notion of decidability for predicates:

Definition 2.1 (Decidable Predicates) *A predicate* $p : X \to \mathbb{P}$ *is decidable if there is a boolean decider* $f : X \to \mathbb{B}$ *such that* $\forall x.\, p\,x \leftrightarrow f\,x = \mathsf{true}$*. We also generalize this notion to predicates with more than one argument in the expectable way. Finally, we write* $\ulcorner p\,x \urcorner$ *for the implicit invocation of a decider.*

Types with decidable equality are called discrete:

Definition 2.2 (Discrete Types) *A type* X *is called discrete if there is an equality decider for* $\lambda xy : X.\, x = y$.

Another well-known concept of computability theory is (recursive) enumerability. Traditionally, a predicate $p : X \to \mathbb{P}$ is called enumerable if there is a Turing machine (or other computable function) that outputs all values for which p holds. In the synthetic setting we could capture this notion by a function $\mathbb{N} \to X$ whose image are exactly the values that satisfy p. However, in order to also incorporate empty predicates, we actually use an option type for the codomain:

Definition 2.3 (Enumerability)

1. *A predicate* $p : X \to \mathbb{P}$ *is enumerable, if there is an enumerator* $f : \mathbb{N} \to \mathcal{O}(X)$*, such that* $\forall x.\, px \leftrightarrow \exists n.\, f\,n = {}^{\circ}x$*. We say that* p *is co-enumerable if its complement* $\overline{p}$*, defined as* $\overline{p} = \lambda x.\, \neg p\,x$*, is enumerable. We also generalize these notions to predicates with more than one argument in the expectable way.*
2. *A type* X *is called enumerable, if there is an enumerator* $f : \mathbb{N} \to \mathcal{O}(X)$*, such that* $\forall x.\, \exists n.\, f\,n = {}^{\circ}x$.

Forster, Kirst and Smolka also introduce the notion of list enumerators $L : \mathbb{N} \to \mathcal{L}(X)$ to enable more convenient enumerability proofs. The idea is to allow more than one value to be enumerated at once. This coincides with the interpretation of enumerability in Definition 2.3:

Fact 2.4 (List Enumerators)

1. *A predicate* $p : X \to \mathbb{P}$ *is enumerable iff there is a list enumerator* $L : \mathbb{N} \to \mathcal{L}(X)$ *such that* $\forall x.\, p\,x \leftrightarrow \exists n.\, x \in L\,x$.
2. *A type* X *is enumerable, if there a list enumerator* $L : \mathbb{N} \to \mathcal{L}(X)$, *such that* $\forall x.\, \exists n.\, x \in L\,n$.

Proof We can turn enumerators $f : \mathbb{N} \to \mathcal{O}(X)$ into list enumerators $L : \mathbb{N} \to \mathcal{L}(X)$ and vice versa, such that $(\exists n.\, f\,n = {}^\circ x) \leftrightarrow (\exists n.\, x \in L\,x)$ for all x. Given f or L, we obtain the other one by

$$f\,n := (L\,(\pi_1\,n))[\pi_2\,n] \qquad\qquad L\,n := \begin{cases} [x] & \text{if } f\,n = {}^\circ x \\ \mathsf{nil} & \text{if } f\,n = \emptyset \end{cases}$$

where $A[n] : \mathcal{O}(X)$ denotes element access on lists. □

List enumerators allow for a general and scalable technique that can be used to show enumerability of inductive types and predicates. The idea is to use cumulative list enumerators (satisfying $L\,n \subseteq L\,(Sn)$) to successively construct new values from the previously enumerated ones. We will see an example of this in the next chapter (Fact 3.6).

2.2.2 Post's Theorem and Markov's Principle

Post's theorem is a standard result from computability theory that states that problems that are both enumerable and co-enumerable (called *bi-enumerable*) are also decidable. As it turns out, this result cannot be proven constructively [6]. However, we do not require full classical logic in order to obtain it. There is an axiom called *Markov's principle* (MP) that is strictly weaker than the law of excluded middle (LEM) [27] and deemed constructive by many. As we will see, MP is both sufficient and necessary to prove Post's theorem.

Definition 2.5 (Markov's Principle) MP *states that for all functions* $f : \mathbb{N} \to \mathbb{B}$ *it holds that* $\neg\neg(\exists n.\, f\,n = \mathsf{true})$ *implies* $\exists n.\, f\,n = \mathsf{true}$.

MP is independent in Coq's type theory [6] and a consequence of LEM. However, it is accepted by the Russian school of constructive mathematics [36]. It has many equivalent characterizations, for example we can also write it as $\neg(\forall n.\, f\,n = \mathsf{false}) \to \exists n.\, f\,n = \mathsf{true}$. The computational justification behind this axiom is unbounded linear search: If we know that f does not return false for all inputs, we can start a search by checking first 0, then 1, and so on, until we find an n for which $f\,n = \mathsf{true}$. The principle idea of linear search is actually available in Coq. From propositional knowledge $\exists n.\, f\,n = \mathsf{true}$ we can obtain a computational witness $\Sigma n.\, f\,n = \mathsf{true}$. Since the existential quantifier cannot be eliminated into $\mathbb{T}$ the proof is not obvious and makes use of the idea of unbounded search.

Fact 2.6 (Unbounded Search) $\exists n.\, f\, n = \mathsf{true}$ *implies* $\Sigma n.\, f\, n = \mathsf{true}$ *for all* $f : \mathbb{N} \to \mathbb{B}$.

Proof We outline the proof idea described in [58]. We define an inductive predicate $T : \mathbb{N} \to \mathbb{P}$ with the single constructor $C : f\, n = \mathsf{false} \to T\,(S n) \to T\, n$. Intuitively, this is the certificate that allows to continue the linear search as long as f returns false. We do this by defining a function $\mu : \forall n.\, T\, n \to \Sigma m.\, p\, m = \mathsf{true}$ by recursion on the proof of $T\, n$:

$$\mu\, n\,(C\, g) := \begin{cases} (n, h) & \text{if we have } h : f\, n = \mathsf{true} \\ \mu\,(S n)\,(g\, h) & \text{if we have } h : f\, n = \mathsf{false} \end{cases}$$

The only remaining obligation is to give a proof of $T\,0$ to start the search. Since this is propositional, we can eliminate the existential quantifier and obtain the witness n which immediately gives us $T\, n$. Since $T\,(S n) \to T\, n$, we can get $T\,0$ by an induction on n. □

We have seen that the propositional knowledge $\exists n.\, f\, n = \mathsf{true}$ can be used as a guarantee that the search terminates eventually. Intuitively, Markov's principle extends on this, by allowing to perform this search, even if only classical evidence $\neg\neg\exists n.\, f\, n = \mathsf{true}$ of termination is available. This enables us to define a boolean decider built from an enumerator and a co-enumerator, yielding Posts's theorem. Interestingly, MP is really necessary to prove this, since we can also show that it is implied by Post's theorem [2, 15]:

Fact 2.7 (Post's Theorem) *Bi-enumerable predicates over discrete types are decidable iff* MP *holds.*

Proof We show both directions separately:

- → Suppose we have MP as well as an enumerator f and a co-enumerator g of a problem $p : X \to \mathbb{P}$ where X is discrete. It suffices to show

  $$\exists n.\, \ulcorner f\, n = {}^\circ x \urcorner \mid \ulcorner g\, n = {}^\circ x \urcorner$$

  for all x where $\mid$ denotes boolean disjunction. That is because a boolean decider can simply obtain the witness with Fact 2.6. By MP it then suffices to show the double negation of this statement, such that we can argue classically: We either have $p\, x$ such that x is enumerated by f, or we have $\neg p\, x$ such that x is enumerated by g.

- ← Assume $\neg\neg\exists n.\, f\, n = \mathsf{true}$. It suffices to show that $P := \lambda x : \mathbb{N}.\ \exists n.\, f\, n = \mathsf{true}$ is decidable. By assumption this is the case if P is bi-enumerable. We can show that p is co-enumerated by $\lambda n.\, \emptyset$ and enumerated by

  $$g\, n := \begin{cases} {}^\circ\pi_1\, n & \text{if } f\,(\pi_2\, n) = \mathsf{true} \\ \emptyset & \text{if } f\,(\pi_2\, n) = \mathsf{false} \end{cases}$$

2.3 Reductions

After defining the positive notions of decidability and enumerability, we also want definitions of negative ones like undecidability. Note, that we cannot simply define it as the negation of decidability, as constructive type theory is consistent with the assumption that every predicate is decidable. This means that it is not possible to prove that a predicate cannot be decided by a boolean function. Instead, we characterize synthetic undecidability as the existence of a chain of reductions from a problem that is widely known to be undecidable, for example the halting problem on Turing machines. Decidability of the target would then imply that this seed problem is also decidable, which is rejected. Optionally, a contradiction can the be derived by assuming computability axioms. The definition of (many-one) reductions is again straightforward in the synthetic setting:

Definition 2.8 (Many-One Reductions) *Given predicates* $p : X \to \mathbb{P}$ *and* $q : Y \to \mathbb{P}$, *we call a function* $f : X \to Y$ *(many-one) reduction if* $\forall x.\, P\, x \leftrightarrow Q\, (f\, x)$. *We say* p *reduces to* q *and write* $p \preccurlyeq q$ *if such a function exists.*

Fact 2.9 *Let* $p \preccurlyeq q$. *Then* p *is decidable if* q *is.*

Proof Let f be a decider for q and g the function witnessing the reduction. Then $f \circ g$ is a decider for p. □

The Coq Library of Undecidability Proofs [12] contains many problems that are proven to be synthetically undecidable (for example [14, 13, 37, 59, 10]). The reduction chains in the library will serve as a starting point for our own reductions later in this thesis.

Chapter 3

Second-Order Syntax, Standard Semantics, and Natural Deduction

In this chapter, we formally introduce second-order logic. Generally, there are three things that are needed to define a formal logic. First, the syntax describes the rules from which logical sentences can be constructed. Thus, we begin by defining the syntax of second-order logic in Section 3.1. Secondly, semantics are used to give meaning to the syntax by defining what it means for a sentence to be valid. In Section 3.2 we discuss the standard semantics of second-order logic. Finally, a notion of provability is needed to syntactically verify that sentences are true. To this end, we introduce a second-order natural deduction system in Section 3.3.

3.1 Syntax

The syntax of second-order logic is based on the one of first-order logic. Therefore, we believe it is instructive to briefly recap first-order logic and also introduce its syntax.

The syntax of both first- and second-order logic can be divided into two distinct parts: Terms constitute the objects we want to reason about, whereas formulas describe properties of those objects and represent statements that can be true or false. While the so called *logical symbols* like quantifiers and connectives can be chosen fairly canonically, the so called *non-logical symbols* needed to represent constants, functions, and predicates vary widely between applications. For example, in order to study Peano arithmetic in Chapter 4 we will need a constant 0, a unary successor function S, and two binary functions $+$ and $\times$. Other languages like ZF set theory in turn require completely different symbols to describe the concepts they want to reason about. Since many of the abstract properties we investigate in this thesis do not depend on the specific choice of non-logical symbols, we define the syntax over a signature Σ that generalizes this notion:

Definition 3.1 (Signature) *A signature $\Sigma = (\mathcal{F}_\Sigma, \mathcal{P}_\Sigma)$ consists of a type $\mathcal{F}_\Sigma$ of function* *symbols and a type $\mathcal{P}_\Sigma$ of predicate symbols. We write $\mathcal{F} : \mathcal{F}_\Sigma$ and $\mathcal{P} : \mathcal{P}_\Sigma$ for members of those types. The arity of the symbols is denoted by $|\mathcal{F}| : \mathbb{N}$ and $|\mathcal{P}| : \mathbb{N}$. Constants can be represented by nullary function symbols.*

We fix a signature Σ for the remainder of this chapter.

3.1.1 First-Order Syntax

We begin by giving the syntax of first-order logic following [15].

Definition 3.2 (First-Order Syntax) *The type of terms $\mathfrak{T}$ and first-order formulas $\mathfrak{F}_1$ is defined by*

$$
\begin{aligned}
t : \mathfrak{T} &::= x_i \mid \mathcal{F}\,\boldsymbol{v} && (i : \mathbb{N}, \mathcal{F} : \mathcal{F}_\Sigma, \boldsymbol{v} : \mathfrak{T}^{|\mathcal{F}|}) \\
\varphi, \psi : \mathfrak{F}_1 &::= \dot{\bot} \mid \mathcal{P}\,\boldsymbol{v} \mid \varphi \mathrel{\dot{\to}} \psi \mid \varphi \mathrel{\dot{\land}} \psi \mid \varphi \mathrel{\dot{\lor}} \psi \mid \dot{\forall}\varphi \mid \dot{\exists}\varphi && (\mathcal{P} : \mathcal{P}_\Sigma, \boldsymbol{v} : \mathfrak{T}^{|\mathcal{P}|})
\end{aligned}
$$

We write $\dot{\neg}\varphi$ for $\varphi \mathrel{\dot{\to}} \dot{\bot}$ and $\varphi \mathrel{\dot{\leftrightarrow}} \psi$ for $(\varphi \mathrel{\dot{\to}} \psi) \mathrel{\dot{\land}} (\psi \mathrel{\dot{\to}} \varphi)$. Furthermore, we use $\dot{\square}$ as a placeholder for any of the connectives $\dot{\to}$, $\dot{\land}$, and $\dot{\lor}$, and $\dot{\nabla}$ for $\dot{\forall}$ and $\dot{\exists}$. We write $\mathfrak{T}(\Sigma)$ and $\mathfrak{F}_1(\Sigma)$ if we want to emphasize the dependency on the signature.

Terms consist of variables (x_i) and function applications $(\mathcal{F}\,\boldsymbol{v})$. Formulas are constructed from the falsity constant $(\dot{\bot})$, atomic formulas $(\mathcal{P}\,\boldsymbol{v})$, implications $(\dot{\to})$, conjunctions $(\dot{\land})$, disjunctions $(\dot{\land})$, universal quantifiers $(\dot{\forall})$, and existential quantifiers $(\dot{\exists})$.

One might wonder why both $\dot{\forall}$ and $\dot{\exists}$ are missing a binding variable. Usually, one would expect to write quantifiers like $\forall x.\ \varphi$. The lack of names is due to the fact that we use so called *de Bruijn binders* instead. They were originally designed by Nicolaas de Bruijn to allow for a machine-friendly representation of binders in lambda terms [8]. But as it turns out, this approach also works well in other situations. Relevant for us, it is well suited for fomalization [60] and usually results in less overhead compared to dealing with names when trying to mechanize syntax with binders. However, we will resort to the usual named syntax when writing formulas on paper to increase legibility.

The general idea behind the de Bruijn encoding is that variables are represented by the number of quantifiers that shadow their binder. We call those numbers (de Bruijn) indices. For example, the formula $\exists a.\ \mathrm{P}(a) \land \forall b.\ \mathrm{Q}(a, b)$ is represented by $\dot{\exists}\,\mathrm{P}(x_0) \land \dot{\forall}\,\mathrm{Q}(x_1, x_0)$. If an index exceeds the number of quantifiers, we say that the variable is free. Free variables are identified with the index remaining after subtracting the number of shadowing binders. For example, in the formula $\dot{\forall}\,\mathrm{P}(x_3) \mathrel{\dot{\lor}} \dot{\exists}\,\mathrm{P}(x_4)$, both indices refer to the free variable with index 2. One benefit of de Bruijn binders it that α-equivalence of formulas[1] reduces to simple equality.

[1] This refers to the fact that bound variables can be consistently renamed. For example $\forall x.\ \mathrm{P}(x)$ is equivalent to $\forall y.\ \mathrm{P}(y)$.

On top of that, this approach allows for a straightforward definition of substitution, sidestepping the issues with unwanted capturing of named binders.

3.1.2 Second-Order Syntax

The terms of second-order logic are exactly the terms $\mathfrak{T}$ of first-order logic. Second-order formulas $\mathfrak{F}_2$ are obtained by extending $\mathfrak{F}_1$ with two new constructs: Besides the individual quantifiers $\dot\forall$ and $\dot\exists$ we add n-ary predicate quantifiers $\dot\forall^n_p$ and $\dot\exists^n_p$, along with n-ary predicate variables p^n_i for all $n : \mathbb{N}$:

Definition 3.3 (Second-Order Syntax) *The type $\mathfrak{F}_2$ of second-order formulas is defined by extending Definition 3.2 with*

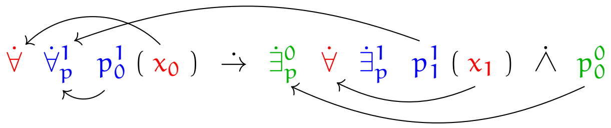
$$\varphi, \psi : \mathfrak{F}_2 ::= \ldots \mid p^n_i\, \boldsymbol{v} \mid \dot\forall^n_p\, \varphi \mid \dot\exists^n_p\, \varphi \qquad (i, n : \mathbb{N}, \boldsymbol{v} : \mathfrak{T}^n)$$

We use the type $\mathfrak{P}_n$ to denote n-ary predicates, that is variables p^n_i and predicate symbols $\mathcal{P} : \Sigma_p$ with $|\mathcal{P}| = n$. We extend the $|\cdot|$ notation to predicate variables and write $|P| = n$ for the arity of any predicate $P : \mathfrak{P}_n$.

Note, that we annotate quantifiers with the arity of the predicate they quantify over. Alternatively, they could quantify over predicates of all arities and the right one is picked based on each usage. This, however, would allow unintuitive formulas like $\forall P.\, P(x) \to P(a, b)$ where P is unary and binary at the same time. Thus, additional well-formedness requirements would need to be put in place to rule out such formulas. In our approach, the quantifier would instead specify the arity of P. For example, if P is unary, the binary occurrence should no longer be bound to the quantifier but be considered free instead. This of course will make the semantics slightly more complicated as the scoping of variables now depends on arity.

Importantly, predicate variables also live in a different scope than the individuals. This means that predicate variables count de Bruijn indices independently of individual variables and predicate variables of other arities. It also means that for determining the binder of a variable p^n_i, only predicate quantifiers $\dot\forall^n_p$ and $\dot\exists^n_p$ of the same arity are taken into account.

Example 3.4 (De Bruijn Scopes) *In the following formula, the arrows point from variables to the quantifier they are bound to:*


$\dot\forall\ \dot\forall^1_p\ p^1_0\,(\,x_0\,)\ \dot\to\ \dot\exists^0_p\ \dot\forall\ \dot\exists^1_p\ p^1_1\,(\,x_1\,)\ \dot\wedge\ p^0_0$

With named binders this formula would be written as $\forall a.\, \exists P.\, P(a) \to \exists Q.\, \forall b.\, \exists R.\, P(a) \wedge Q.$

Besides predicate quantifiers, some authors also consider function quantifiers as part of second-order logic (for example [56, 67]). In the textbook setting they simply allow for more convenient notation and do not bring any additional expressive power with them. However, the treatment of function quantifiers is a bit more involved in our setting of constructive type theory. We will discuss this later in the context of semantics (Section 3.2). To this end, we also define the syntactic fragment $\mathfrak{F}_2^F$ that extends $\mathfrak{F}_2$ with function quantifiers and variables:

Definition 3.5 (Second-Order Syntax with Function Quantifiers) *The types $\mathfrak{T}_2^F$ and $\mathfrak{F}_2^F$ of second-order terms and formulas with function quantifiers is defined by extending Definition 3.3 with*

$$\begin{aligned} t : \mathfrak{T}_2^F &::= \ldots \mid f_i^n\, \boldsymbol{v} && (i, n : \mathbb{N}, \boldsymbol{v} : (\mathfrak{T}_2^F)^n) \\ \varphi, \psi : \mathfrak{F}_2^F &::= \ldots \mid \dot{\forall}_f^n\, \varphi \mid \dot{\exists}_f^n\, \varphi && (n : \mathbb{N}) \end{aligned}$$

The de Bruijn scoping works analogously to $\mathfrak{F}_2$.

While de Bruijn indices are well suited for a formal treatment of syntax, the resulting formulas tend to be difficult to understand, especially when different scopes are involved as in Example 3.4. Therefore, we fall back to named binders when giving examples or stating concrete formulas, and keep in mind that the formalization uses the de Bruijn paradigm behind the scenes. In this case we also drop the subscript p and f annotation of second-order quantifiers and distinguish the different kinds of variables by names, using lower case letters a, b, c, x, y, z for individuals, f, g, h for functions and upper case letters X, Y, Z, P, Q, R for predicates. We also leave out the aritiy annotations if arities can be derived from the context.

3.1.3 Properties of the Syntax

Discreteness and enumerability of Σ also extends to the syntax. Therefore, we frequently require the signature to be discrete and enumerable, meaning that $\mathcal{F}_\Sigma$ and $\mathcal{P}_\Sigma$ are required to have those properties.

Fact 3.6 (Discreteness and Enumerability)

1. *If Σ is discrete, then $\mathfrak{F}_1$, $\mathfrak{F}_2$, and $\mathfrak{F}_2^F$ are also discrete.*
2. *If Σ is enumerable, then $\mathfrak{F}_1$, $\mathfrak{F}_2$, and $\mathfrak{F}_2^F$ are also enumerable.*

Proof We prove both statements separately.

1. Given formulas φ and ψ we need to decide if φ and ψ are equal. This follows by induction on φ and case analysis on ψ and requires a similar property for terms, also shown by induction.

2. We only discuss the case of $\mathfrak{F}_1$ as the proof can easily be extended to the other fragments. By Fact 2.4 is suffices to give a list enumerator for $\mathfrak{F}_1$. We also get list enumerators $L_{\mathcal{F}_\Sigma}$ and $L_{\mathcal{P}_\Sigma}$ for $\mathcal{F}_\Sigma$ and $\mathcal{P}_\Sigma$. We begin by defining a list enumerator $L_{\mathfrak{T}}$ for terms:

$$\begin{aligned} L_{\mathfrak{T}}\,0 &:= [\,] \\ L_{\mathfrak{T}}\,(S n) &:= L_{\mathfrak{T}}\,n \mathbin{+\!\!+} [\,x_n\,] \mathbin{+\!\!+} [\,\mathcal{F}\,v \mid \mathcal{F} \in L_{\mathcal{F}_\Sigma}\,n, \boldsymbol{v} \in (L_{\mathfrak{T}}\,n)^{|\mathcal{F}|}\,] \end{aligned}$$

where $\mathcal{A}^n$ returns a list of all vectors of length n that can be constructed from values in the list A. $L_{\mathfrak{T}}$ is cumulative, that is $L_{\mathfrak{T}}\,n \subseteq L_{\mathfrak{T}}\,(S n)$. We can see that $L_{\mathfrak{T}}$ uses the previously enumerated values to construct new terms, thus enumerating every term eventually. Formally, we can show $\forall t : \mathfrak{T}.\, \exists n.\, L_{\mathfrak{T}}\,n = t$ by induction on t. A list enumerator for $\mathfrak{F}_1$ is defined in a similar way. □

Next, we introduce the notion of substitution.

Definition 3.7 (Substitutions)

1. *A capture avoiding individual instantiation of a formula φ with a parallel substitution $\sigma : \mathbb{N} \to \mathfrak{T}$ is denoted by $\varphi[\sigma]$. This operation is extended to terms and vectors and is defined by*

$$\begin{array}{ccc} x_i[\sigma] := \sigma\, i & (\mathcal{P}\,\boldsymbol{v})[\sigma] := \mathcal{P}\,\boldsymbol{v}[\sigma] & (\dot\nabla\ \varphi)[\sigma] := \dot\nabla\ \varphi[x_0 \cdot {\uparrow}\sigma] \\ (\mathcal{F}\,\boldsymbol{v})[\sigma] := \mathcal{F}\,\boldsymbol{v}[\sigma] & (p_i^n\,\boldsymbol{v})[\sigma] := p_i^n\,\boldsymbol{v}[\sigma] & (\dot\nabla_p^n\ \varphi)[\sigma] := \dot\nabla_p^n\ \varphi[\sigma] \\ \dot\bot[\sigma] := \dot\bot & (\varphi\,\dot\Box\,\psi)[\sigma] := \varphi[\sigma]\,\dot\Box\,\psi[\sigma] & \end{array}$$

where $t \cdot \sigma$ denotes the extension of a substitution σ with a term t and ${\uparrow}\sigma$ denotes the lifting of the so called shift substitution $\uparrow$ to σ. They are defined by

$$\begin{array}{cc} (t \cdot \sigma)\,0 := t & \uparrow i := x_{i+1} \\ (t \cdot \sigma)\,(S i) := \sigma\, i & ({\uparrow}\sigma)\, i := (\sigma\, i)[\uparrow] \end{array}$$

As a useful shorthand we write $\varphi[t]$ for the substitution $\varphi[t \cdot \lambda i.\, x_i]$ instantiating the index 0 with t and reducing the other indices by one.

2. *A capture avoiding predicate instantiation with arity n of a formula φ with a parallel substitution $\sigma : \mathbb{N} \to \mathfrak{P}_n$ is denoted by $\varphi[\sigma]_p^n$. It is defined by*

$$\begin{array}{ll} \dot\bot[\sigma]_p^n := \dot\bot & (\varphi\,\dot\Box\,\psi)[\sigma]_p^n := \varphi[\sigma]\,\dot\Box\,\psi[\sigma]_p \\ (\mathcal{P}\,\boldsymbol{v})[\sigma]_p^n := \mathcal{P}\,\boldsymbol{v} & (\dot\nabla\ \varphi)[\sigma]_p^n := \dot\nabla\ \varphi[\sigma]_p \\ (p_i^m\,\boldsymbol{v})[\sigma]_p^n := \begin{cases} (\sigma\, i\, m)\ \boldsymbol{v} & \textit{if } m = n \\ p_i^m\,\boldsymbol{v} & \textit{otherwise} \end{cases} & (\dot\nabla_p^m\ \varphi)[\sigma]_p^n := \begin{cases} \dot\nabla_p^m\ \varphi[p_0^n \cdot {\uparrow}\sigma]_p^n & \textit{if } m = n \\ \dot\nabla_p^m\ \varphi & \textit{otherwise} \end{cases} \end{array}$$

Where $\mathsf{P} \cdot \sigma$ and $\uparrow$ are defined in the same way as in (1). Again, we use the shorthand $\varphi[\mathsf{P}]^n_p$ to denote the substitution $\varphi[\mathsf{P} \cdot \lambda i.\, p^n_i]^n_p$.

A parallel substitution σ must specify an instantiation for every possible variable index. While those are infinitely many, it is also clear that each individual formula φ can only contain finitely many free variables. Therefore, the values from σ will not be used from a certain index i onwards since all free variables are below i. To capture this intuition, we define the notion of bounds on free variables occurring in formulas:

Definition 3.8 (Bounds)

1. *We say b is a bound on the free individual variables occurring in φ, denoted by $\sup(\varphi) \leqslant b$, if the indices of all free individual variables in φ are less than b. This notion is also extended to terms. In particular, we have*

$$\sup(x_i) \leqslant b := i < b \qquad \sup(\dot\nabla\, \varphi) \leqslant b := \sup(\varphi) \leqslant Sb$$

$$\sup(\dot\nabla^n_p\, \varphi) \leqslant b := \sup(\varphi) \leqslant b.$$

   We say a variable x_i is fresh or unused in φ, if it does not occur in φ. This is certainly the case if $i \geqslant b$ for $\sup(\varphi) \leqslant b$.

2. *We say b is a bound on the free predicate variables of arity n occurring in φ, denoted by $\sup^n_p(\varphi) \leqslant b$, if the indices of all free predicate variables of arity n in φ are less than b. In particular, we have*

$$\sup^n_p(p^m_i) \leqslant b := \begin{cases} i < b & \textit{if } m = n \\ \top & \textit{otherwise} \end{cases}$$

$$\sup^n_p(\dot\nabla^m_p\, \varphi) \leqslant b := \begin{cases} \sup^n_p(\varphi) \leqslant Sb & \textit{if } m = n \\ \sup^n_p(\varphi) \leqslant b & \textit{otherwise} \end{cases}$$

   Again, we say a variable p^n_i is fresh or unused in φ, if it does not occur in φ. This is certainly the case if $i \geqslant b$ for $\sup^n_p(\varphi) \leqslant b$.

3. *A formula φ is closed if $\sup(\varphi) \leqslant 0$ and $\sup^n_p(\varphi) \leqslant 0$ for all n.*

Fact 3.9 (Bounded Substitution) *Let $\varphi : \mathfrak{F}_2$, $b : \mathbb{N}$, and $\sigma_1, \sigma_2 : \mathbb{N} \to \mathfrak{T}$ and $\tau_1, \tau_2 : \mathbb{N} \to \mathfrak{P}_n$ for $n : \mathbb{N}$ be substitutions. Then*

1. *If $\sup(\varphi) \leqslant b$ and $\sigma_1\ i = \sigma_1\ i$ for all $i < b$, then $\varphi[\sigma_1] = \varphi[\sigma_2]$.*
2. *If $\sup^n_p(\varphi) \leqslant b$ and $\tau_1\ i = \tau_2\ i$ for all $i < b$, then $\varphi[\tau_1]^n_p = \varphi[\tau_2]^n_p$.*
3. *If φ is closed, then $\varphi[\sigma_1] = \varphi[\sigma_2]$ and $\varphi[\tau_1]^n_p = \varphi[\tau_2]^n_p$.*

Proof (1) and (2) are proven by induction on φ. (3) immediately follows from (1) and (2). □

Bounds can easily be computed for any formula:

Fact 3.10 *Let $\varphi : \mathfrak{F}_2$. Then*

1. *One can construct a b with $\sup(\varphi) \leqslant b$.*
2. *For all n, one can construct a b with $\sup_p^n(\varphi) \leqslant b$.*

Proof By induction on φ. □

3.2 Semantics

Next, we give meaning to the previously defined syntax. That is, we want to define what it means for a formula of second-order logic to be valid. This is determined by a semantic relation $\vDash$. We begin by defining the standard semantics of second-order logic for the syntactic fragment $\mathfrak{F}_2$ in Section 3.2.1. Afterwards, we discuss how to extend the semantics to $\mathfrak{F}_2^F$ in Section 3.2.2.

3.2.1 Standard Tarski Semantics

The notion of model-theoretic semantics discussed here is due to Alfred Tarski [61, 62]. He argued that the so called *object logic* that is being discussed (for us, this is second-order logic) should be interpreted by lifting statements into the *meta-logic* in which the analysis is carried out (in our case this is constructive type theory).

However, in order to give meaning to all second-order formulas we need some additional information. For example, to figure out if a formula $\exists x.\, P(x)$ is valid, it is necessary to know from which universe of values x may be picked from. This set of values that quantifiers range over is also called *domain of discourse*. The choice of domain can easily change whether a formula is considered valid or not. Secondly, we need to know how the non-logical symbols should be interpreted. For example, consider the formula $\varphi := \dot\forall xy.\, x + y = y + x$. On its face, the symbols "+" and "=" are just syntax with no direct meaning associated to them. If we interpreted those symbols as the usual addition function and equality predicate, we would end up with a valid formula stating the commutativity of addition (provided our domain consists of numbers). However, there are many other binary functions and predicates one could come up with, for which φ would be invalid.

Hence, the semantics cannot be universally defined on its own, but depends on the domain and symbol interpretation. This additional information needed to define validity is provided by a so called *model*:

Definition 3.11 (Model) *A model $\mathcal{M}$ consists of a domain $D : \mathbb{T}$ and an interpretation $\mathcal{I}$ for function and predicate symbols. The interpretations are given by $\mathcal{F}^{\mathcal{I}} : D^{|\mathcal{F}|} \to D$ and $\mathcal{P}^{\mathcal{I}} : D^{|\mathcal{P}|} \to \mathbb{P}$ for $\mathcal{F} : \Sigma_f$ and $\mathcal{P} : \Sigma_p$.*

Finally, we also have to consider formulas that contain free variables. For instance, the validity of the formula $x = 0$ depends on the value of x. In order to make sense of such formulas, we employ so called *environments* ρ that specify the values of free variables. Thus, we end up with a ternary satisfiability relation $\mathcal{M}, \rho \vDash \varphi$ that states whether φ is valid in a given model and environment:

Definition 3.12 (Standard Semantics) *Given a model $\mathcal{M}$ consisting of a domain D and an interpretation $\mathcal{I}$, as well as an environment $\rho = (\rho_i, \rho_p)$ consisting of assignments*

$$\rho_i : \mathbb{N} \to D \qquad\qquad \rho_p : \mathbb{N} \to \forall n.\, D^n \to \mathbb{P}$$

for individual and predicate variables, we define term evaluation $[\![\cdot]\!]_\rho : \mathfrak{T} \to D$ and formula satisfiability $\rho \vDash \cdot : \mathfrak{F}_2 \to \mathbb{P}$ by

$$\begin{array}{ccc} & \rho \vDash \dot{\bot} := \bot & \rho \mathrel{\dot{\Box}} \varphi \mathrel{\dot{\rightarrow}} \psi := (\rho \vDash \varphi) \mathrel{\Box} (\rho \vDash \psi) \\ [\![x_i]\!]_\rho := \rho_i\, i & \rho \vDash \mathcal{P}\, \boldsymbol{v} := \mathcal{P}^{\mathcal{I}}\, [\![\boldsymbol{v}]\!]_\rho & \rho \vDash \dot{\nabla}\, \varphi := \nabla d : D.\, d \cdot \rho \vDash \varphi \\ [\![\mathcal{F}\, \boldsymbol{v}]\!]_\rho := \mathcal{F}\, [\![\boldsymbol{v}]\!]_\rho & \rho \vDash p_i^n\, \boldsymbol{v} := \rho_p\, i\, n\, [\![\boldsymbol{v}]\!]_\rho & \rho \vDash \dot{\nabla}_p^n\, \varphi := \nabla P : D^n \to \mathbb{P}.\, P \cdot \rho \vDash \varphi \end{array}$$

where $\Box$ and ∇ are the meta-logical counterparts of the logical symbols. The modified environments $d \cdot \rho$ and $P \cdot \rho$ occurring in the quantifier cases extend the individual or predicate environment with an additional value. They are defined by

$$\begin{array}{ll} (d \cdot \rho_i)\, 0 := d & (d \cdot \rho_i)\, (S i) := \rho_i\, i \\ (P \cdot \rho_p)\, 0\, n := \begin{cases} P & \textit{if } |P| = n \\ \rho_p\, 0\, n & \textit{otherwise} \end{cases} & (P \cdot \rho_p)\, (S i)\, n := \begin{cases} \rho_p\, i\, n & \textit{if } |P| = n \\ \rho_p\, (S i)\, n & \textit{otherwise} \end{cases} \end{array}$$

We write $\mathcal{M}, \rho \vDash \varphi$ and $[\![t]\!]_\rho^{\mathcal{M}}$ when we want to make the used model explicit. We also extend satisfiability to theories $\mathcal{T} : \mathfrak{F}_2 \to \mathbb{P}$ and write $\mathcal{M}, \rho \vDash \mathcal{T}$ if $\mathcal{M}, \rho \vDash \varphi$ for all $\varphi \in \mathcal{T}$. Furthermore, we write $\mathcal{M} \vDash \varphi$ if $\mathcal{M}, \rho \vDash \varphi$ for all ρ. Similarly, we say that $\mathcal{M}$ is a model of $\mathcal{T}$, written $\mathcal{M} \vDash \mathcal{T}$, if $\mathcal{M}, \rho \vDash \mathcal{T}$ for all ρ. Finally, we say that φ is valid under $\mathcal{T}$, written $\mathcal{T} \vDash \varphi$, if $\mathcal{M}, \rho \vDash \varphi$ for all $\mathcal{M}$ and ρ with $\mathcal{M}, \rho \vDash \mathcal{T}$.

Overall, we can see that $\vDash$ maps logical connectives and quantifiers to their meta-level counterpart, adding quantified variables to the environment. Hence, the initial choice of ρ only affects the free variables in a formula φ. Therefore, the environment is irrelevant if φ is closed:

Fact 3.13 $\mathcal{M}, \rho_1 \vDash \varphi$ *implies* $\mathcal{M}, \rho_2 \vDash \varphi$ *for all* $\mathcal{M}$, ρ_1, ρ_2, *and closed* φ.

Proof It suffices to prove

$$\forall b.\ \mathrm{sup}\,(\varphi) \leqslant b \to (\forall i.\, i < b \to \rho_1\, i = \rho_2\, i) \to \mathcal{M}, \rho_1 \vDash \varphi \leftrightarrow \mathcal{M}, \rho_2 \vDash \varphi$$

and a similar lemma for predicates by induction on φ. □

3.2.2 Extension to Function Quantifiers

While $\vDash$ can easily be adapted to the first-order fragment $\mathfrak{F}_1$ by removing the unnecessary cases, extending it to $\mathfrak{F}_2^F$ poses an interesting question. The approach that first comes to mind is to interpret function quantifiers $\dot\forall_f^n$ by quantifying over functions $D^n \to D$. This is also in line with the way we interpreted function symbols.

Definition 3.14 (Standard Semantics for Function Quantifiers) *Consider a model* $\mathcal{M}$ *with domain* D*. Environments* ρ *are extended with an additional component* $\rho_f : \mathbb{N} \to \forall n.\, D^n \to D$ *assigning values to function variables. We extend term evaluation and formula satisfiability from Definition 3.12 to* $\mathfrak{T}_2^F$ *and* $\mathfrak{F}_2^F$ *by adding the following cases:*

$$[\![f_i^n]\!]_\rho := \rho_f\; i\; n \qquad \rho \vDash \dot\nabla_f^n\, \varphi := \nabla f : D^n \to D.\, f \cdot \rho \vDash \varphi$$

However, most people working in second-order logic use classical set theory as their meta-logic. There, n-ary functions can simply be considered as the $(n+1)$-ary relation containing the graph of the function. Hence, extending second-order logic with function quantifiers does not add anything new since it can all be reduced back to predicates.[2] The same also holds for function symbols; they just serve as a notational convenience in their setting.

But the concept of functions in type theory does not directly match up with functions in set theory. The function space $D^n \to D$ is underspecified and therefore this correspondence to relations is not available without axioms. As a result, the semantics given above behaves slightly differently than one would normally expect. We will later see an example of this in Chapter 4.

One approach to remedy this and to get the same properties as the set-theoretic semantics in the literature is to interpret functions in a different way. We define $X \rightsquigarrow Y$ as the type of total and functional relations from X to Y:

$$X \rightsquigarrow Y := \Sigma F : X \to Y \to \mathbb{P}.\, \mathsf{total}\, F \land \mathsf{functional}\, F$$

where $\mathsf{total}\, F := \forall x.\, \exists y.\, F\, x\, y$ and $\mathsf{functional}\, F := \forall x y y'.\, F\, x\, y \to F\, x\, y' \to y = y'$. Interpreting functions as $D^n \rightsquigarrow D$ is more faithful to the traditional set-theoretic approach and leads to the following relational semantics:

Definition 3.15 (Relational Standard Semantics for Function Quantifiers) *The function interpretation and variable assignment in a relational Model* $\mathcal{M}_R$ *have types* $\mathcal{F}^\mathcal{I} : D^{|\mathcal{F}|} \rightsquigarrow D$ *and* $\rho_f : \mathbb{N} \to \forall n.\, D^n \rightsquigarrow D$*. Term evaluation turns into a relation* $[\![\cdot]\!]_\rho : \mathfrak{T}_2^F \to D \to \mathbb{P}$ *and we get formula satisfiability* $\rho \vDash \varphi$ *with*

$$[\![x_i]\!]_\rho\; d := \rho_i\; i = d \qquad \rho \vDash \mathcal{P}\,\boldsymbol{v} := \exists \boldsymbol{w} : D^{|\mathcal{P}|}.\, [\![\boldsymbol{v}]\!]\, \boldsymbol{w} \land \mathcal{P}^\mathcal{I}\, \boldsymbol{w}$$

$$[\![\mathcal{F}\,\boldsymbol{v}]\!]_\rho\; d := \exists \boldsymbol{w} : D^{|\mathcal{F}|}.\, [\![\boldsymbol{v}]\!]_\rho\, \boldsymbol{w} \land \mathcal{F}^\mathcal{I}\, \boldsymbol{w}\, d \qquad \rho \vDash \dot\forall_f^n\, \varphi := \forall f : D^n \rightsquigarrow D.\, f \cdot \rho \vDash \varphi$$

[2] One could also get rid of predicates and in turn express them using their characteristic function. Thus, functions and relations can be used interchangeably.

The remaining cases are defined in a similar way.

Fact 3.16 *The evaluation relation* $[\![\cdot]\!]_\rho$ *is total and functional.*

Proof By induction on the terms. □

One way to bridge the gap between the two semantics is to alter our meta-theory by assuming unique choice. It is defined by

$$\mathsf{UC} := \forall XY : \mathbb{T}.\, \forall R : X \rightsquigarrow Y.\, \forall x.\, \Sigma y.\, R\, x\, y$$

Note that this is not a propositional axiom, but instead a stronger computational operator for the totality of the relation.

Fact 3.17 *Assuming* UC*, one can turn* $F : X \rightsquigarrow Y$ *into* $\hat{F} : X \to Y$*. Similarly, one can turn* $f : X \to Y$ *into* $\tilde{f} : X \rightsquigarrow Y$ *with* $\tilde{f} := \lambda xy.\, f\, x = y$*. Lifting those translations to models and environments yields the following results:*

1. $\mathcal{M}, \rho \vDash \varphi$ *iff* $\tilde{\mathcal{M}}, \tilde{\rho} \vDash \varphi$ *for all models* $\mathcal{M}$ *and environments* ρ.
2. $\mathcal{M}_R, \rho \vDash \varphi$ *iff* $\hat{\mathcal{M}}_R, \hat{\rho} \vDash \varphi$ *for all relational models* $\mathcal{M}_R$ *and environments* ρ.

Proof We show (1) by induction on φ. (2) follows from (1) and the fact that $\tilde{\hat{\mathcal{M}}}_R$ and $\tilde{\hat{\rho}}$ behave the same as $\mathcal{M}_R$ and ρ. □

The downside of the relational semantics is that it is quite tedious to work with and mechanize because of the overhead introduced by always needing to invoke totality and functionality. Generally, we will not focus on $\mathfrak{F}_2^F$ on paper and instead present the results only in terms of $\mathfrak{F}_2$ most of the time. We refer to the underlying Coq mechanization that extends most results regarding Tarski semantics to $\mathfrak{F}_2^F$ using the more convenient function semantics from Definition 3.14.[3] When appropriate, we will comment whether and how the results can be extended to function quantifiers.

3.3 Natural Deduction

Next, we introduce the notion of provability via a relation $\vdash$. For this, we use a variant of Gentzen's natural deduction system [18, 19] defined on the syntactic fragment $\mathfrak{F}_2$. It is based on the first-order system presented in [15].

Definition 3.18 (Natural Deduction) *We represent the deduction system as an inductive predicate* $\vdash\, : \mathcal{L}(\mathfrak{F}_2) \to \mathfrak{F}_2 \to \mathbb{P}$*. Its rules are given by*

$$\textsc{Ctx}\,\frac{\varphi \in A}{A \vdash \varphi} \qquad \textsc{Exp}\,\frac{A \vdash \dot{\bot}}{A \vdash \varphi} \qquad \text{II}\,\frac{A, \varphi \vdash \psi}{A \vdash \varphi \dot{\rightarrow} \psi} \qquad \text{IE}\,\frac{A \vdash \varphi \dot{\rightarrow} \psi \qquad A \vdash \varphi}{A \vdash \psi}$$

[3] The main exception were relational semantics are required will be discussed in Section 4.2.

$$\text{CI}\,\frac{A \vdash \varphi \qquad A \vdash \psi}{A \vdash \varphi \mathbin{\dot{\wedge}} \psi} \qquad \text{CE}_1\,\frac{A \vdash \varphi \mathbin{\dot{\wedge}} \psi}{A \vdash \varphi} \qquad \text{CE}_2\,\frac{A \vdash \varphi \mathbin{\dot{\wedge}} \psi}{A \vdash \psi}$$

$$\text{DI}_1\,\frac{A \vdash \varphi}{A \vdash \varphi \mathbin{\dot{\vee}} \psi} \qquad \text{DI}_2\,\frac{A \vdash \psi}{A \vdash \varphi \mathbin{\dot{\vee}} \psi}$$

$$\text{DE}\,\frac{A \vdash \varphi \mathbin{\dot{\vee}} \psi \qquad A, \varphi \vdash \theta \qquad A, \psi \vdash \theta}{A \vdash \theta}$$

$$\textsc{AllI}\,\frac{A[\uparrow] \vdash \varphi}{A \vdash \dot{\forall}\, \varphi} \qquad \textsc{AllE}\,\frac{A \vdash \dot{\forall}\, \varphi}{A \vdash \varphi[t]}$$

$$\text{ExI}\,\frac{A \vdash \varphi[t]}{A \vdash \dot{\exists}\, \varphi} \qquad \text{ExE}\,\frac{A \vdash \dot{\exists}\, \psi \qquad A[\uparrow], \psi \vdash \varphi}{A \vdash \varphi}$$

$$\textsc{AllI}_p\,\frac{A[\uparrow]^n_p \vdash \varphi}{A \vdash \dot{\forall}^n_p\, \varphi} \qquad \textsc{AllE}_p\,\frac{A \vdash \dot{\forall}^n_p\, \varphi}{A \vdash \varphi[P]^n_p}$$

$$\text{ExI}_p\,\frac{A \vdash \varphi[P]^n_p}{A \vdash \dot{\exists}^n_p\, \varphi} \qquad \text{ExE}_p\,\frac{A \vdash \dot{\exists}^n_p\, \psi \qquad A[\uparrow]^n_p, \psi \vdash_2 \varphi}{A \vdash \varphi}$$

$$\textsc{Peirce}\,\frac{}{A \vdash_c ((\varphi \mathbin{\dot{\rightarrow}} \psi) \mathbin{\dot{\rightarrow}} \varphi) \mathbin{\dot{\rightarrow}} \varphi} \qquad \textsc{Compr}\,\frac{P \textit{ not free in } \varphi}{A \vdash \dot{\exists} P.\, \dot{\forall} x_1 ... x_n.\, P(x_1, ..., x_n) \mathbin{\dot{\leftrightarrow}} \varphi}$$

We use two versions of the deduction system denoted by $\vdash_c$ *and* $\vdash_i$ *distinguishing between classical and intuitionistic logic. The* (Peirce) *rule is only available in the classical system* $\vdash_c$ *and may not be used in* $\vdash_i$. *We write* $\vdash$ *if the choice between* $\vdash_c$ *and* $\vdash_i$ *does not matter.*

The (Ctx) rule allows us to derive assumed formulas and the (Exp) rule is used to deduce arbitrary formulas from a contradiction. Most of the other rules can be categorized into either introduction rules that state how the different logical connectives can be proven (denoted by I), or elimination rules that state what can be derived from them (denoted by E). For example, an implication can be proven by showing that the conclusion holds if the premise is assumed (II), and can be used to derive the conclusion whenever the premise holds (IE). A conjunction is shown by proving that both parts hold (CI), and can derive both of them (CE_1, CE_2). A disjunction can be proven by demonstrating that one of the formulas hold (DI_1, DI_2), and can be used to derive a formula, whenever the formula holds if either one is assumed (DE). Universally quantified formulas are treated rather non-standardly as they are derived by showing that they hold in a shifted context (AllI) which only works in a de Bruijn setting. Through the shift, the zero index becomes free which simulates an arbitrary but fixed object. This approach allows for an easier mechanization of abstract properties like weakening. However, we will show later that the more traditional approach of replacing the index zero with a fresh variable is also admissible. Furthermore, a universally quantified formula can be used to prove the formula for any term t (AllE). Similarly, an existentially quantified formula is shown by demonstrating that it holds for some term t (ExI), and can be used to deduce every formula that holds, whenever the former is assumed in the shifted context (ExE). The (AllI$_p$), (AllE$_p$), (ExI$_p$) and (ExE$_p$) rules are extensions of the same notions to predicate quantifiers. The classical (Peirce) axiom states that every formula holds, if it holds when it implies another formula. We chose this approach of enforcing classicality since it does not rely on falsity.

Finally, we have the so called comprehension axiom (Compr). It is not adapted from the first-order system, but a completely new axiom that is introduced for second-order logic. To understand its motivation, consider that in order to prove $\dot\exists\,\varphi$, it suffices to construct a term t for which φ holds. This term can be arbitrarily complex and tailored to satisfy φ. However, in the predicate case our choices are more limited. For example, with the rules discussed before we would not even be able to prove $\dot\exists P.\,\dot\forall x.\,P(x)$ stating that there is a true predicate, as long as we do not already have such a predicate in the context or a symbol with this meaning. That is, because the only predicates we can construct in order to show $\dot\exists_p^n\,\varphi$ are variables or symbols. Thus, the comprehension axiom is introduced to certify the existence of further predicates. It states that for each formula φ there is a predicate P that extensionally behaves the same as the n-ary property expressed by φ. Importantly, P may not occur freely in φ.[4] To refer to the comprehension axiom later on we define the following shorthand:

$$\mathsf{Compr}_\varphi^n := \dot\exists P.\,\dot\forall x_1 \ldots x_n.\,P(x_1, \ldots, x_n) \mathrel{\dot\leftrightarrow} \varphi \qquad \text{where } P \text{ may not occur freely in } \varphi$$

If we remove the (AllI$_p$), (AllE$_p$), (ExI$_p$), (ExE$_p$) and (Compr) rules, we end up with a deduction system for the first-order fragment $\mathfrak{F}_1$. This system, which we will write as $\vdash_1$, exactly matches the one from [15]. We could also add analogous rules for function quantifiers to extend $\vdash$ to the syntactic fragment $\mathfrak{F}_2^F$. Function comprehension can be similarly defined as

$$\mathsf{FuncCompr}_\varphi^n := \mathsf{total}_\varphi^n \rightarrow \mathsf{functional}_\varphi^n \rightarrow \dot\exists f.\,\dot\forall x_1 \ldots x_n.\,\varphi[f(x_1, \ldots, x_n)].$$

The assumptions $\mathsf{total}_\varphi^n := \dot\forall x_1 \ldots x_n.\,\dot\exists y.\,\varphi$ and $\mathsf{functional}_\varphi^n := \dot\forall y y' x_1 \ldots x_n.\,\varphi[y] \mathrel{\dot\rightarrow} \varphi[y'] \mathrel{\dot\rightarrow} y = y'$[5] assert that φ describes a $(n+1)$-ary relation that is total and functional in the last argument. In this case, there is a function f that computes a value

[4] Otherwise the axiom would be inconsistent: We could set $\varphi := \dot\neg P(x)$ and get $\dot\exists P.\,\dot\forall x.\,P(x) \mathrel{\dot\leftrightarrow} \dot\neg P(x)$.

[5] The equality $y = y'$ can be expressed in second-order logic via the Leibniz characterization $\dot\forall P.\,P(y) \mathrel{\dot\rightarrow} P(y')$.

that satisfies this relation. In the remainder of this thesis, we will not explore this further and only focus on $\mathfrak{F}_2$ in the context of deduction.

While the context of $\vdash$ is represented by finite lists A, we can also extend the notion of provability to potentially infinite theories $\mathcal{T} : \mathfrak{F}_2 \to \mathbb{P}$:

Definition 3.19 (Deduction under Theories) *A formula φ is provable under a theory $\mathcal{T} : \mathfrak{F}_2 \to \mathbb{P}$, written $\mathcal{T} \vdash \varphi$, if there is a list $A \subseteq \mathcal{T}$, such that $A \vdash \varphi$. We also extend the classicality annotations to provability under theories.*

This approach makes the deduction system compact by definition. This is intuitively valid, since a derivation under a theory $\mathcal{T}$ can only use finitely many assumptions. Therefore, the formula can also be derived from a list A if A contains all those assumptions. The benefit of reducing provability under theories back to lists is that we do not need to formalize yet another deduction system. Furthermore it fits better to the intuition that deduction should conceptually be finitary.

Next, we show two different weakening properties of $\vdash$. We can replace the context with one that subsumes the original one and proofs remain valid under substitutions.

Fact 3.20 (Weakening) *The following rules hold for $\vdash$:*

$$\textsc{Weak}\;\frac{A' \vdash \varphi \qquad A' \subseteq A}{A \vdash \varphi} \qquad \textsc{WeakS}\;\frac{A \vdash \varphi}{A[\sigma] \vdash \varphi[\sigma]} \qquad \textsc{WeakS}_p\;\frac{A \vdash \varphi}{A[\sigma]^n_p \vdash \varphi[\sigma]^n_p}$$

Proof By induction on the derivations. □

Using the weakening results we can show that variants of the (AllI) and (ExE) rule, that do not shift the context, but instead substitute a fresh variable, are admissible. Those are in fact the standard rules most commonly used, especially outside of the de Bruijn setting. We call this variant *named* and remark that it is helpful when working on concrete derivations.

Lemma 3.21 (Named Quantifier Rules) *The following rules hold for $\vdash$:*

$$\textsc{AllI}'\;\frac{A \vdash \varphi[x_i] \qquad x_i \notin A, \dot\forall\varphi}{A \vdash \dot\forall\varphi}$$

$$\textsc{ExE}'\;\frac{A \vdash \dot\exists\psi \qquad A, \varphi[x_i] \vdash \psi \qquad x_i \notin A, \dot\exists\varphi}{A \vdash \varphi}$$

$$\textsc{AllI}'_p\;\frac{A \vdash \varphi[p_i]^n_p \qquad p^n_i \notin A, \dot\forall^n_p\varphi}{A \vdash \dot\forall^n_p\varphi}$$

$$\text{ExE}'_p \; \frac{A \vdash \dot{\exists}^n_p \psi \qquad A, \varphi[p_i]^n_p \vdash \psi \qquad p^n_i \notin A, \dot{\forall}^n_p \varphi}{A \vdash \varphi}$$

where $x_i \notin A$ *and* $p^n_i \notin A$ *denote that the variables do not occur in* A.

Proof We only discuss (AllI') as the other cases are proven similarly. It suffices to show that $A[\uparrow] \vdash \varphi$ iff $A \vdash \varphi[x_i]$.

$\rightarrow$ Follows from Fact 3.20 with the substitution $[x_i]$ since $A[\uparrow][x_i] = A$.

$\leftarrow$ Follows from Fact 3.20 with the substitution

$$\sigma\, j := \begin{cases} x_0 & \text{if } i = j \\ x_{j+1} & \text{otherwise} \end{cases}$$

since x_i does not occur in A and x_{i+1} does not occur in φ. □

In order for deduction systems to be a useful means to prove statements, they should be enumerable. This is also the case for $\vdash$:

Fact 3.22 $\lambda\varphi.\, A \vdash \varphi$ *is enumerable for all* A.

Proof Similar to the proof of Fact 3.6. □

Additionally, we want that $\vdash$ only proves valid statements. This property is called soundness.

Theorem 3.23 (Soundness) *Let* $\mathcal{T}$ *be a theory and* φ *a formula. Then*

1. $\mathcal{T} \vdash_i \varphi \to \mathcal{T} \vDash \varphi$
2. $\mathcal{T} \vdash_c \varphi \to \mathcal{T} \vDash \varphi$ *under* LEM

Proof We discuss both claims:

1. If $\mathcal{T} \vdash_i \varphi$, we have a list $A \subseteq \mathcal{T}$ with $A \vdash_i \varphi$. Thus, it suffices to prove $A \vdash_i \varphi \to A \vDash \varphi$ since $A \vDash \varphi$ implies $\mathcal{T} \vDash \varphi$. This is done via an induction on $A \vdash_i \varphi$. We only discuss the cases of universally quantified formulas and comprehension. The remaining ones are similar or straightforward.

(AllI) Given a model $\mathcal{M}$ and an environment ρ with $\mathcal{M}, \rho \vDash A$, we have to show $\mathcal{M}, d \cdot \rho \vDash \varphi$ for some $d : D$. We know $A[\uparrow] \vDash \varphi$ from the inductive hypothesis, such that it suffices to prove $\mathcal{M}, d \cdot \rho \vDash A[\uparrow]$. We can generally show

$$\mathcal{M}, \rho \vDash \psi[\sigma] \leftrightarrow \mathcal{M}, (\lambda x.\, [\![\sigma\, x]\!]^{\mathcal{M}}_{\rho}, \rho_p) \vDash \psi$$

for all σ and ρ by induction on ψ. Therefore, it is enough to instead prove $\mathcal{M}, (\lambda x.\, [\![\uparrow x]\!]^{\mathcal{M}}_{d \cdot \rho}, \rho_p) \vDash \psi$ for all $\psi[\uparrow] \in A[\uparrow]$. This holds by assumption, since $[\![\uparrow x]\!]^{\mathcal{M}}_{d \cdot \rho} = \rho_i\, x$.

(AllE) Given a model $\mathcal{M}$ and an environment ρ with $\mathcal{M}, \rho \vDash A$, we have to show $\mathcal{M}, \rho \vDash \varphi[t]$ for some term t. Again, we can move the substitution into the environment such that it suffices to show $\mathcal{M}, t \cdot \rho \vDash \varphi$. This holds by the inductive hypothesis.

(Compr) Given a model $\mathcal{M}$ and an environment ρ with $\mathcal{M}, \rho \vDash A$, we have to show $\mathcal{M}, \rho \vDash \mathsf{Compr}^n_\varphi$. Comprehension is satisfied by taking the predicate $\lambda \boldsymbol{v}.\, \mathcal{M}, \boldsymbol{v} \cdot \rho \vDash \varphi$, where $\boldsymbol{v} \cdot \rho$ adds all components of $\boldsymbol{v}$ to the environment.

2. Similar to the proof of (1). In the case of (Peirce), we have to show that an instance of the Peirce rule holds in $\mathbb{P}$. This follows as a consequence of LEM.

□

Remark 3.24 *The dependency on* LEM *in the classical case is actually necessary to get this strong sense of soundness. As we have* $\vdash_c \mathsf{P} \mathbin{\dot\vee} \dot\neg\mathsf{P}$ *for any nullary predicate variable* P*, soundness also yields* LEM*. A different approach taken in* [15] *is to define the notion of classical models. There, a model* $\mathcal{M}$ *is considered classical if it validates all instances of Peirce's law, i.e.* $\mathcal{M} \vDash ((\varphi \mathbin{\dot\to} \psi) \mathbin{\dot\to} \varphi) \mathbin{\dot\to} \varphi$ *for all* φ, ψ*. If validity is restricted to classical models, soundness is provable without* LEM (*at the cost of fewer models being definable*).

Chapter 4

Categoricity of Second-Order Peano Arithmetic

An important area where second-order logic differs from its first-order counterpart is the notion of categoricity. A theory $\mathcal{T}$ is called *categorical*, if all models of $\mathcal{T}$ are equal up to isomorphism. Thus, $\mathcal{T}$ uniquely determines its domain and the interpretation of its function and predicate symbols. In other words, $\mathcal{T}$ is strong enough to uniquely characterize its own intended structure.

Interestingly, there are no categorical first-order theories (at least none that admit an infinite model). One example of this is Peano arithmetic which tries to axiomatize the natural numbers with addition and multiplication. First-order logic admits so called non-standard models of Peano arithmetic that are not isomorphic to the natural numbers. This is a consequence of the upward Löwenheim-Skolem theorem (which is not mechanized in this thesis):

Theorem 4.1 (Upward Löwenheim-Skolem) *Every countable first-order theory with a countably infinite model has a model of every infinite cardinality.*

There is also the downwards version stating that each countable first-order theory with an infinite model also has a countable model. Overall, this justifies the previous statement that first-order logic is not able to characterize any infinite structures in a categorical way. In this chapter, we demonstrate that this is not the case for second-order logic. The expressive power gained by the additional quantifiers allows it to characterize many structures in a categorical way, including Peano arithmetic. We begin by defining second-order Peano arithmetic in Section 4.1 and show its categoricity in Section 4.2. Finally, we investigate some immediate consequences of this result in Section 4.3 and will also make further use of it in Chapter 5.

4.1 Second-Order Peano Arithmetic

The signature Σ_{PA} of second-order Peano arithmetic contains symbols for the constant zero, the successor, addition and multiplication functions, as well as an equality predicate:

$$(O, S\_, \_ \oplus \_, \_ \otimes \_; \_ \equiv \_)$$

The addition of an equality symbol might seem puzzling at first glance, since equality is actually second-order definable via the Leibniz characterization $x = y \sim \dot\forall P.\, P(x) \dot\rightarrow P(y)$. We settled on the more uncommon symbol approach to stay in closer correspondence to first-order Peano arithmetic and allow for easier arguments in the first-order fragment later on.

Figure 4.1 shows the formulas making up the theory PA_2 of second-order Peano arithmetic.

$$\begin{array}{llll}
\oplus\text{-zero}: & \dot\forall x.\, O \oplus x \equiv x & \oplus\text{-rec}: & \dot\forall xy.\, (Sx) \oplus y \equiv S(x \oplus y) \\
\otimes\text{-zero}: & \dot\forall x.\, O \otimes x \equiv O & \otimes\text{-rec}: & \dot\forall xy.\, (Sx) \otimes y \equiv y \oplus x \otimes y \\
O\text{-succ}: & \dot\forall x.\, O \equiv Sx \dot\rightarrow \dot\bot & S\text{-inj}: & \dot\forall xy.\, Sx \equiv Sy \dot\rightarrow x \equiv y \\
\equiv\text{-refl}: & \dot\forall x.\, x \equiv x & \equiv\text{-symm}: & \dot\forall xy.\, x \equiv y \dot\rightarrow y \equiv x
\end{array}$$

$$\text{induction}_2: \quad \dot\forall P.\ P\ O \dot\rightarrow (\dot\forall x.\, P\ x \dot\rightarrow P\ (Sx)) \dot\rightarrow \dot\forall x.\, P\ x$$

Figure 4.1: Axioms of PA_2.

The axiomatisation of equality might seem surprisingly concise with neither transitivity, nor congruence axioms for S, $\oplus$, and $\otimes$ being present. But as it turns out, the axioms given here already suffice to characterize equality (see Fact 4.3). The reason for this is the strong induction$_2$ axiom. It is the only one that exploits the power of second-order logic via the predicate quantifier and gives us a corresponding induction principle in models of PA_2:

Fact 4.2 (Induction principle) *Let $\mathcal{M}$ be a model of PA_2 with domain D and interpretation $\mathcal{I}$. The the induction principle $\forall P : D \to \mathbb{P}.\, P\, O^{\mathcal{I}} \to (\forall x.\, P\, x \to P\, (S^{\mathcal{I}}\, x)) \to \forall x.\, P\, x$ can be used for properties on D.*

Proof Follows directly from $\mathcal{M} \vDash \text{induction}_2$. □

In first-order Peano arithmetic (PA_1) the induction axiom needs to be replaced by an axiom scheme:

$$\text{induction}_1: \quad \varphi[O] \dot\rightarrow (\dot\forall x.\, \varphi[x] \dot\rightarrow \varphi[Sx]) \dot\rightarrow \dot\forall x.\, \varphi[x] \qquad \text{for all } \varphi : \mathfrak{F}_1$$

Thus, PA_1 only allows to induce on first-order definable properties, which is of course weaker than the principle obtained in Fact 4.2. For example, we show in the next Fact that the interpretation of $\equiv$ in PA_2 corresponds to equality, which is not possible in PA_1.[1]

[1] For example, consider the intensional model with domain $\mathbb{N} \times \mathbb{B}$ and $(n_1, \_) \equiv^{\mathcal{I}} (n_2, \_) := n_1 = n_2$.

Fact 4.3 *Let $\mathcal{M}$ be a model of PA_2 with domain D and interpretation $\mathcal{I}$. Then $x \equiv^{\mathcal{I}} y$ iff $x = y$ for all $x, y : D$.*

Proof By induction on x and case analysis on y using Fact 4.2. The different cases follow with O-succ, S-inj, $\equiv$-refl and $\equiv$-symm. □

The so called standard model for PA_2 are the natural numbers:

Definition 4.4 (Standard Model) *The standard model of PA_2 consists of the domain $\mathbb{N}$ and interprets zero, successor, addition, multiplication and equality with the usual operations on $\mathbb{N}$. We slightly overload the notation and denote this model by $\mathbb{N}$.*

4.2 Categoricity

The categoricity result for PA_2 is originally due to Dedekind [9], but we follow the more contemporary proof by Shapiro [56]. We fix two models $\mathcal{M}_1$ and $\mathcal{M}_2$ of PA_2 and write D_1 and D_2 for the domains, as well as $\mathcal{I}_1$ and $\mathcal{I}_2$ for the interpretations. The goal is to give an isomorphism between $\mathcal{M}_1$ and $\mathcal{M}_2$.

There is not much hope for being able to define this isomorphism as a function $D_1 \to D_2$. The only means of determining the structure of $x : D_1$ is via the induction axiom which is inherently propositional and cannot be used to compute a corresponding $y : D_2$. Thus, we need to give the isomorphism as a relation $D_1 \to D_2 \to \mathbb{P}$:

Definition 4.5 *We define the relation $\cong : D_1 \to D_2 \to \mathbb{P}$ inductively by $O^{\mathcal{I}_1} \cong O^{\mathcal{I}_2}$ and $S^{\mathcal{I}_1}\, x \cong S^{\mathcal{I}_2}\, y$ if $x \cong y$. We also extend $\cong$ to vectors, functions, predicates and environments in the pointwise way.*

Fact 4.6 *$\cong$ is total, surjective, functional, and injective, i.e.*

1. $\forall x.\, \exists y.\, x \cong y$
2. $\forall y.\, \exists x.\, x \cong y$
3. $\forall x y y'.\, x \cong y \to x \cong y' \to y = y'$
4. $\forall x x' y.\, x \cong y \to x' \cong y \to x = x'$

Totality and surjectivity can also be extended to vectors, predicates and environments.

Proof Follows by induction. □

Fact 4.7 *$\cong$ respects the structure of the models, i.e. for all x, x', y, y' it holds that:*

1. $O^{\mathcal{I}_1} \cong O^{\mathcal{I}_1}$
2. $x \cong y \to S^{\mathcal{I}_1}\, x \cong S^{\mathcal{I}_1}\, y$
3. $x \cong y \to x' \cong y' \to x \oplus^{\mathcal{I}_1} x' \cong y \oplus^{\mathcal{I}_2} y'$
4. $x \cong y \to x' \cong y' \to x \otimes^{\mathcal{I}_1} x' \cong y \otimes^{\mathcal{I}_2} y'$

5. $x \cong y \to x' \cong y' \to (x \equiv^{\mathcal{I}_1} x' \leftrightarrow y \equiv^{\mathcal{I}_2} y')$

Proof (1) and (2) hold by definition. (3) and (4) follow by induction on x using the axioms for $\oplus$ and $\otimes$. (5) follows by induction on x using Fact 4.3. □

We can conclude that $\cong$ is the isomorphism we were looking for and that PA_2 is indeed categorical. A direct consequence of this is that satisfiability of formulas agrees in both models:

Lemma 4.8 (Agreement) *Let* ρ_1 *and* ρ_2 *be environments with* $\rho_1 \cong \rho_2$*. Then* $\mathcal{M}_1, \rho_1 \vDash \varphi$ *iff* $\mathcal{M}_2, \rho_2 \vDash \varphi$ *for all* $\varphi : \mathfrak{F}_2$*.*

Proof By induction on φ with ρ_1 and ρ_2 generalized.

- If $\varphi = (s \equiv t)$, we have to show $([\![s]\!]^{\mathcal{M}_1}_{\rho_1} \equiv^{\mathcal{I}_1} [\![t]\!]^{\mathcal{M}_1}_{\rho_1}) \leftrightarrow ([\![s]\!]^{\mathcal{M}_2}_{\rho_2} \equiv^{\mathcal{I}_2} [\![t]\!]^{\mathcal{M}_2}_{\rho_2})$. By Fact 4.7 it suffices to prove $[\![t]\!]^{\mathcal{M}_1}_{\rho_1} \cong [\![t]\!]^{\mathcal{M}_2}_{\rho_2}$ for all t by induction on t using the remaining points from Fact 4.7.
- If $\varphi = p^n_i\, \boldsymbol{v}$, we have to show $\rho_1\, [\![\boldsymbol{v}]\!]^{\mathcal{M}_1}_{\rho_1} \leftrightarrow \rho_2\, [\![\boldsymbol{v}]\!]^{\mathcal{M}_2}_{\rho_2}$. This holds since $[\![\boldsymbol{v}]\!]^{\mathcal{M}_1}_{\rho_1} \cong [\![\boldsymbol{v}]\!]^{\mathcal{M}_2}_{\rho_2}$ like in the previous case and $\rho_1 \cong \rho_2$.
- If $\varphi = \dot\forall\, \psi$, we have to show $(\forall d_1 : D_1.\, d_1 \cdot \rho_1 \vDash \psi) \leftrightarrow (\forall d_2 : D_2.\, d_2 \cdot \rho_2 \vDash \psi)$.
  - $\to$ By surjectivity (Fact 4.6), we obtain $d_1 : D_1$ with $d_1 \cong d_2$, such that $d_1 \cdot \rho_1 \cong \rho_2 \cdot \rho_2$ which allows us to use the inductive hypothesis on the assumption.
  - $\leftarrow$ Symmetrical to the other direction using totality.
- If $\varphi = \dot\forall^n_p\, \psi$, we can proceed like the previous case, using the fact that surjectivity and totality transport to predicates.

The remaining cases are straightforward or follow in a similar way. □

Interestingly, totality and subjectivity do not transport to functions. For example, we cannot show that for all $f_1 : (D_1)^n \to D_1$ there is a $f_2 : (D_2)^n \to D_2$ with $f_1 \cong f_2$. That is again, because $\cong$ is not computable, meaning that although knowing that $\cong$ is total, given an $x : D_1$, we cannot compute a y with $x \cong y$. Consequently, we cannot extend Lemma 4.8 to the fragment $\mathfrak{F}^F_2$ if we use the function semantics. The relational semantics, on the other hand, does not face this problem as computability is not required to translate between the domains:

Lemma 4.9 (Agreement for Relational Semantics) *We can turn* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *into relational models* $\mathcal{M}^R_1$ *and* $\mathcal{M}^R_2$*. Let* ρ^R_1 *and* ρ^R_2 *be relational environments with* $\rho^R_1 \cong \rho^R_2$*. Then* $\mathcal{M}^R_1, \rho^R_1 \vDash \varphi$ *iff* $\mathcal{M}^R_2, \rho^R_2 \vDash \varphi$ *for all* $\varphi : \mathfrak{F}^F_2$*.*

Proof Similar to the proof of Lemma 4.8, using the fact that $\cong$ is surjective and total for $(D_i)^n \rightsquigarrow D_i$. □

Finally, we remark that $\cong$ becomes computational if a recursion principle is available for the domains (for example if one of the domains is $\mathbb{N}$):

Fact 4.10 *If a recursion principle* $\forall P : D_1 \to X.\, P\, O^{\mathcal{I}_1} \to (\forall x.\, P\, x \to P\,(S^{\mathcal{I}_1}\, x) \to \forall x.\, P\, x$ *is available for* D_1*, we have* $\forall x.\, \Sigma y.\, x \cong y$*. The same holds symmetrically for* y*.*

Proof Analogous to the totality proof, using the recursion principle on x. □

4.3 Consequences of Categoricity

We showed that all models of PA_2 are isomorphic to the standard model $\mathbb{N}$. This means that the upward Löwenheim-Skolem theorem fails for second-order logic:

Fact 4.11 (Failure of Upward Löwenheim-Skolem) *There is a countable second-order theory with only countably infinite models.*

Proof PA_2 is finite and thus countable. Let $\mathcal{M}$ be a model of PA_2 with domain D. D is in bijection with $\mathbb{N}$ via $\cong$ and therefore countably infinite. □

Similarly, the downwards Löwenheim-Skolem theorem can be refuted by showing the categoricity of second-order real analysis [56] or set theory[2] [33]. Thus, second-order logic is expressive enough to distinguish different infinite cardinalities. Furthermore, we can use the categoricity result to refute yet another of the main properties of first-order logic. A logic is called compact if every theory $\mathcal{T}$ has a model if every finite subset of $\mathcal{T}$ has a model. While first-order logic is compact, second-order logic is not:

Fact 4.12 (Failure of Compactness) *Not all second-order theories* $\mathcal{T}$ *have a model if every finite subset of* $\mathcal{T}$ *has a model.*

Proof We define the infinite theory $\mathcal{T}_> := \mathsf{PA}, \varphi_{>0}, \varphi_{>1}, \ldots$ where

$$\varphi_{>n} := x_0 \not\equiv O \mathrel{\dot\land} x_0 \not\equiv S\,O \mathrel{\dot\land} \ldots \mathrel{\dot\land} x_0 \not\equiv \underbrace{S(\ldots(S\,O))}_{n \text{ times}}$$

states that the variable x_0 does not have the value of any of the numerals up to n. Every finite subset $A \subseteq \mathcal{T}_>$ is satisfied by the standard model $\mathbb{N}$ by choosing a large enough value for x_0. However, there is no model of the whole theory $\mathcal{T}_>$. Otherwise, $\mathcal{T}_>$ would also be satisfied by the standard model, because of categoricity. This is not possible since there is no value $k : \mathbb{N}$ that x_0 can be assigned to, since $\varphi_{>k}$ is contained in $\mathcal{T}_>$. □

[2] Second-order ZF is categorical for equipotent models.

Finally, we can use the proof above to show another major point of disagreement between first- and second-order logic. There is no sound deduction systems $\vdash : \mathcal{L}(\mathfrak{F}_2) \to \mathfrak{F}_2 \to \mathbb{P}$ that, lifted to theories as described in Definition 3.19, fulfills $\mathcal{T} \vDash \varphi \to \mathcal{T} \vdash \varphi$ for all decidable theories $\mathcal{T}$ and φ. We call this *infinitary incompleteness* since it does not rule out completeness for finite contexts.

Theorem 4.13 (Infinitary Incompleteness) *Every sound second-order deduction system $\vdash : \mathcal{L}(\mathfrak{F}_2) \to \mathfrak{F}_2 \to \mathbb{P}$ is not infinitary complete.*

Proof Let $\vdash$ be sound and infinitary complete. It suffices to show that this implies that the theory $\mathcal{T}_>$ from the proof of Fact 4.12 is compact, contradicting the argument in the proof. Assume that all finite contexts $A \subseteq \mathcal{T}_>$ have a model. Suppose there where no model of $\mathcal{T}_>$ (since we originally wanted to show a negative claim, we can argue by contradiction without requiring LEM). Then $\mathcal{T}_> \vDash \dot{\bot}$ and by infinitary completeness $\mathcal{T}_> \vdash \dot{\bot}$ since $\mathcal{T}_>$ is decidable. Hence, there is also a finite context $A \subseteq \mathcal{T}_>$ with $A \vdash \dot{\bot}$ and by soundness $A \vDash \dot{\bot}$. This is not possible, since by assumption $A \subseteq \mathcal{T}_>$ has a model. □

Remarkably, we do not even need to put any computability requirements like enumerability on the system in order for it to be infinitary incomplete. The only assumption we have is that the system is sound and that the lifting to theories makes it compact. As a consequence, we can directly conclude that the natural deduction system from Chapter 3 is not infinitary complete:

Corollary 4.14 (Infinitary Incompleteness of $\vdash$)

1. $\vdash_i$ *is not infinitary complete.*
2. $\vdash_c$ *is not infinitary complete under* LEM.

Proof Follows by Theorem 4.13 since $\vdash$ is sound. Note that classical soundness requires LEM. □

However, it might still be the case that there is a deduction system that is complete when restricted to finite contexts A, which we call *finitary completeness*. For example, the "system" $A \vdash \varphi := A \vDash \varphi$ is sound and finitary complete, but not infinitary complete by Theorem 4.13. Of course, no one would consider this a sensible deduction system and we should be more restrictive with this term. As it turns out, it suffices to require enumerability of deduction systems in order to rule out the one above and show that no finitary complete one can exist. Establishing this stronger incompleteness result requires a more intricate argument involving computability and Gödel's first incompleteness theorem, which will be investigated in the next chapter.

Chapter 5

Incompleteness and Undecidability of Standard Semantics

At the end of the previous chapter we already got a first taste of the inherent incompleteness of second-order logic. In this chapter, we strengthen this by also refuting finitary completeness, which is stronger than the result from Chapter 4 in the sense that it also rules out completeness for finite contexts. This requires a very different proof strategy and is the result that is usually presented in the literature (for example [56]). The proof also requires Markov's principle and the standard assumption that deduction systems are enumerable.

We begin by giving an overview on the subject of (in)completeness and discuss some intuition on the proof in Section 5.1. Then, we first show an undecidability result in Section 5.2 and use it to obtain incompleteness in Section 5.3 via a computability argument. Section 5.4 is concerned with extending the result to arbitrary signatures. Finally, we use the same reduction to obtain undecidability of second-order validity, satisfiability, and Peano arithmetic in Section 5.5.

5.1 Overview

The terms "completeness" and "incompleteness" are used to describe many different concepts in mathematical logic. For our discussion in this chapter, there are two conflicting notions that are relevant. To avoid confusion we begin by describing them following the exposition by Read [50].

The concept we referred to before when speaking of "(in)completeness" is usually called *deduction-(in)completeness*. That is, a logic is considered deduction-complete if there is a sound and effective deduction system $\vdash$ that proves all valid formulas. As discussed before, one can also differentiate between the finitary and infinitary notion with infinitary deduction-completeness being stronger than finitary deduction-completeness and vice versa for incompleteness. For the discussion in this chapter, the distinction however does not matter, so we leave out the "finitary" and "infinitary" descriptors. The term "effective" has historical origins and can be understood more precisely as the requirement that provability in $\vdash$ should be enumerable.[1] Therefore, validity in deduction-complete logics is also enumerable, that is, there is an algorithm that enumerates all valid statements. In 1929, Gödel showed his famous completeness theorem [20], stating that first-order logic is deduction-complete. Thus, semantic entailment $\vDash_1$ and syntactic entailment $\vdash_1$ coincide for first-order logic.

Apart from this, there is the notion of *negation-(in)completeness* that is concerned with axiomatisations or theories. A theory $\mathcal{T}$ is considered negation-complete if $\mathcal{T}$ can prove or disprove every closed formula, that is $\mathcal{T} \vdash \varphi$ or $\mathcal{T} \vdash \dot{\neg}\varphi$ for all closed φ. In 1931, Gödel showed that each enumerable consistent[2] first-order axiomatisation of the natural numbers must inherently be negation-incomplete. This result is known as Gödel's first incompleteness theorem [21]. One can also view it under the perspective of computability theory, since in this particular case, his result shows that first-order satisfiability in $\mathbb{N}$ is not enumerable. So while there is a program that outputs the first-order statements that hold in all models (cf. completeness theorem), there is no program that generates the ones that hold in $\mathbb{N}$.

In light of the categoricity results discussed in Chapter 4, one might already anticipate that the situation must look differently in second-order logic: The notions "holding in all models" and "holding in $\mathbb{N}$" fall together when talking about second-order Peano arithmetic, since $\mathbb{N}$ is the only model of PA_2 (up to isomorphism). And indeed, if second-order logic were deduction-complete then PA_2 would be negation-complete. This would yield a program that enumerates all truths about the natural numbers, contradicting the incompleteness theorem. Hence, second-order deduction-incompleteness is usually obtained as a direct corollary of Gödel's first incompleteness theorem.

In his proof of the first incompleteness theorem, Gödel constructs a self-referential statement that can neither be proven nor disproven. This result has already been formalized in various proof assistants [55, 44, 24, 46]. In contrast to these direct proofs, Kirst and Hermes [31] approximate the result, employing the paradigm of synthetic computability. They mechanized the following theorem, where the theory Q' contains the addition and multiplication axioms from the previous chapter:

Theorem 5.1 (Kirst and Hermes [31]) *Let an extension $\mathcal{A} \supseteq \mathsf{Q}'$ be given that is satisfied by the standard model $\mathbb{N}$. Assuming* LEM, *negation-completeness of $\mathcal{A}$ (i.e. $\mathcal{A} \vdash_1^c \varphi$*

[1] A different way of putting it is that the rules of the system should be decidable. We actually did not need this requirement in Chapter 4, but it is certainly a sensible restriction. Otherwise such systems would not be very useful for proving statements.

[2] A theory is consistent if it does not proof $\dot{\perp}$. Gödel actually required so called ω-consistency which comes with some further restrictions that were later lifted by Rosser [53].

or $\mathcal{A} \vdash_1^c \dot\neg\varphi$ for all closeed first-order φ) would imply the decidability of the halting problem of Turing machines.

So instead of constructing an explicit independent Gödel sentence, they approximate the traditional incompleteness result by showing that negation-completeness would imply the decidability of the halting problem. Based on the intuition given above, we can show the following result:

Fact 5.2 *Assuming* LEM, $\mathsf{PA}_2 \vDash \varphi$ *or* $\mathsf{PA}_2 \vDash \dot\neg\varphi$ *for all closed* $\varphi : \mathfrak{F}_2$.

Proof By LEM we either have $\mathbb{N} \vDash \varphi$ or $\mathbb{N} \vDash \dot\neg\varphi$. The claim follows by Lemma 4.8 since all models of PA_2 behave the same as $\mathbb{N}$. □

Thus, PA_2 would be negation-complete for any complete second-order deduction system. Unfortunately, this does not suffice to invoke Theorem 5.1 since Kirst and Hermes employ a classical first-order system $\vdash_1^c$. But their proof could be extended to any system that is assumed to be sound, enumerable and complete, which would yield a classical proof of second-order deduction-incompleteness.

However, a different, more attractive approach is to follow the computability argument sketched above. As mentioned before, the set of closed first-order formulas that hold in $\mathbb{N}$ is not enumerable. To show this, it suffices in this particular case to prove that it is not decidable, using the same reduction as Kirst and Hermes. This yields a proof of deductive-incompleteness that does not require full classicality, but only Markov's principle. Moreover, there is the additional benefit that the reduction also gives us undecidability results for many other problems related to second-order logic along the way.

5.2 Undecidability via Reduction from Hilbert's Tenth Problem

Hilbert's tenth problem (H_{10}) is about deciding whether Diophantine equations $p = q$ have a solution in the natural numbers. This problem is undecidable [42] which has already been verified up to the halting problem for Turing machines as part of the Coq Library of Undecidability Proofs [37, 12]. The following reduction is largely based on the one by Kirst and Hermes [31].

Definition 5.3 (Diophantine Equations) *Polynomials consist of constants, variables, addition and multiplication:*

$$p, q : \mathsf{poly} ::= \mathsf{num}\ n \mid \mathsf{var}\ x \mid \mathsf{add}\ p\ q \mid \mathsf{mul}\ p\ q \qquad (n, x : \mathbb{N})$$

Evaluation $\langle\!\langle p \rangle\!\rangle_\alpha$ *of polynomials under a variable assignment* $\alpha : \mathbb{N} \to \mathbb{N}$ *is defined by*

$$\langle\!\langle \mathsf{num}\ n \rangle\!\rangle_\alpha := n \qquad \langle\!\langle \mathsf{add}\ p\ q \rangle\!\rangle_\alpha := \langle\!\langle p \rangle\!\rangle_\alpha + \langle\!\langle q \rangle\!\rangle_\alpha$$
$$\langle\!\langle \mathsf{var}\ x \rangle\!\rangle_\alpha := \alpha\ x \qquad \langle\!\langle \mathsf{mul}\ p\ q \rangle\!\rangle_\alpha := \langle\!\langle p \rangle\!\rangle_\alpha \cdot \langle\!\langle q \rangle\!\rangle_\alpha$$

A Diophantine equation $p = q$ *has a solution if there is an* α *such that* $\langle\!\langle p \rangle\!\rangle_\alpha = \langle\!\langle q \rangle\!\rangle_\alpha$.

H_{10} is well suited to reduce into arithmetic since the problem can easily be expressed as a formula. We start with encoding polynomials as terms by defining a translation function η:

Definition 5.4 (Translation) *We define* $\eta : \mathsf{poly} \to \mathfrak{T}$ *by*

$$\eta\,(\mathsf{num}\ n) := \nu(n) \qquad \eta\,(\mathsf{add}\ p\ q) := \eta\,p \oplus \eta\,q$$
$$\eta\,(\mathsf{var}\ i) := x_i \qquad \eta\,(\mathsf{mul}\ p\ q) := \eta\,p \otimes \eta\,q$$

where ν *is recursively defined by* $\nu(0) = O$ *and* $\nu(n+1) = S(\nu(n))$.

A Diophantine equation can now be encoded by binding all variables with existential quantifiers:

Definition 5.5 *Let* n *be a bound on the variables occurring in* $p = q$. *We define* $\varphi_{p,q} := \exists^n\ \eta\,p \equiv \eta\,q$ *where* $\exists^0\,\psi := \psi$ *and* $\exists^{Sn} := \exists\,\exists^n\,\psi$.

Fact 5.6 $\varphi_{p,q}$ *is closed for all* p, q.

Notice that the resulting formula $\varphi_{p,q}$ is first-order, since no second-order quantifiers or variables are involved. The following fact characterizes the behaviour of the $\exists^n$ operation:

Fact 5.7 *Let* $\mathcal{M}$ *be a model,* φ *a formula, and* $n : \mathbb{N}$ *with* $\sup(\varphi) \leqslant n$. *Then* $\mathcal{M} \vDash \exists^n \varphi$ *iff there is an environment* ρ *with* $\mathcal{M}, \rho \vDash \varphi$.

Proof We show both directions by induction on n. We only discuss the general intuition:

- → If $\mathcal{M} \vDash \exists^n \varphi$, then in particular $\mathcal{M}, \rho \vDash \mathcal{M}$ for some ρ (e.g. $\rho = \langle \lambda x.\, O^{\mathcal{I}}, \lambda x n \boldsymbol{v}.\, \top \rangle$). Hence, there exist $d_1, ..., d_n$ such that $\mathcal{M}, d_1 \cdot ... \cdot d_n \cdot \rho \vDash \varphi$.
- ← If there is ρ with $\mathcal{M}, \rho \vDash \varphi$, then the existential quantifiers can be satisfied by $\rho_i\,0, ..., \rho_i\,(n-1)$ and the remaining environment $\rho' = \langle \lambda x.\, \rho_i\,(x+n), \rho_p \rangle$, such that $\mathcal{M}, \rho' \vDash \exists^n \varphi$. Since $\exists^n \varphi$ is closed, we can switch to any other environment and have $\mathcal{M} \vDash \exists^n \varphi$. □

The evaluation of the encoded term in the standard model coincides with the evaluation of the polynomial:

Lemma 5.8 *Let* p *by a polynomial,* α *an assignment and* ρ_p *a predicate environment. Then* $[\![\eta\ p]\!]^{\mathbb{N}}_{(\alpha,\rho_p)} = \langle\!\langle p \rangle\!\rangle_\alpha$.

Proof By induction on p. □

Thus, we can reduce H_{10} to validity in the standard model:

Lemma 5.9 (Undecidability of $\mathbb{N}$) $p = q$ *has a solution iff* $\mathbb{N} \vDash \varphi_{p,q}$.

Proof We show both directions separately:

- $\rightarrow$ Let α be a solution to $p = q$. By Fact 5.7 it suffices to give an environment ρ with $\mathbb{N}, \rho \vDash \eta\, p \equiv \eta\, q$ which reduces to $[\![\eta\ p]\!]^{\mathbb{N}}_{\rho} = [\![\eta\ q]\!]^{\mathbb{N}}_{\rho}$. By Lemma 5.8 this holds for the environment (α, ρ_p) for arbitrary ρ_p.
- $\leftarrow$ If $\mathbb{N} \vDash \varphi_{p,q}$, we have ρ with $\mathbb{N}, \rho \vDash \eta\, p \equiv \eta\, q$ by Fact 5.7. Thus, ρ_i is a solution to $p = q$ by Lemma 5.8. □

5.3 Incompleteness

Using the undecidability result, we can refute enumerability via Post's theorem:

Lemma 5.10 *Assuming* MP*, enumerability of all closed, first-order formulas that hold in* $\mathbb{N}$ *implies decidability of* H_{10}*:*

Proof Suppose $\lambda\varphi : \mathfrak{F}_1.\,\mathsf{closed}\ \varphi \wedge \mathbb{N} \vDash \varphi$ is enumerable. In order to show decidability of H_{10} it suffices to show decidability of $\lambda pq.\,\mathbb{N} \vDash \varphi_{p,q}$ by Lemma 5.9. By Post's theorem (Fact 2.7) it suffices to show that this problem is bi-enumerable:

- Enumerability of $\lambda pq.\,\mathbb{N} \vDash \varphi_{p,q}$ follows from the assumption since $\varphi_{p,q}$ is first-order and closed.
- Enumerability of $\lambda pq.\,\neg\mathbb{N} \vDash \varphi_{p,q}$ is equivalent to $\lambda pq.\,\mathbb{N} \vDash \dot\neg\varphi_{p,q}$ and follows again by the assumption. □

However, if second-order logic were deduction-complete, we could enumerate all first-order truths in $\mathbb{N}$:

Lemma 5.11 *The existence of a sound, complete, and enumerable second-order deduction system for the signature* Σ_{PA_2} *implies the enumerability of all closed first-order formulas that hold in* $\mathbb{N}$*.*

Proof By categoricity, a closed first-order formula φ holds in $\mathbb{N}$ iff it is valid in PA_2. Since a complete deduction system enumerates all valid formulas, it also enumerates all closed first-order ones. As it is decidable whether formulas are closed and first-order, this yields the enumerator we were looking for. □

Thus, we have deduction-incompleteness:

Theorem 5.12 (Incompleteness) *Assuming* MP, *the existence of a sound, complete, and enumerable deduction system for* $\mathfrak{F}_2^{F}$ *in* Σ_{PA_2} *implies the decidability of* H_{10}.

Proof Immediate by Lemma 5.10 and Lemma 5.11. □

In particular, the system $\vdash$ introduced in Chapter 3 is incomplete:

Corollary 5.13 (Incompleteness of $\vdash$)

1. *Assuming* MP, *completeness of* $\vdash_i$ *implies decidability of* H_{10}.
2. *Assuming* LEM, *completeness of* $\vdash_c$ *implies decidability of* H_{10}.

Proof Follows from Theorem 5.16 since $\vdash$ is sound and enumerable. Soundness in $\vdash_c$ requires LEM, subsuming MP needed for Theorem 5.16. □

5.4 Extending Incompleteness to Arbitrary Signatures

We can extend Theorem 5.12 to arbitrary signatures since second-order logic allows us to encode finite signatures and axiomatisations into formulas themselves. For this task, it is very useful to have function quantifiers available since in this case the functions and predicates symbols can directly be replaced with variables that are quantified at the beginning:

Lemma 5.14 (Signature Embedding in $\mathfrak{F}_2^{F}$) *Let* $\varphi : \mathfrak{F}_2^{F}(\Sigma_{\mathsf{PA}_2})$ *be a formula and* Σ' *an arbitrary signature. Then there is a formula* $\varphi' : \mathfrak{F}_2^{F}(\Sigma')$ *and theory* PA_2' *in this signature such that*

1. $\mathsf{PA}_2 \vDash \varphi$ *iff* $\vDash \dot\forall\, f_O\, f_S\, f_\oplus\, f_\otimes.\, \dot\forall\, P_\equiv.\, \mathsf{PA}_2' \mathrel{\dot\rightarrow} \varphi'$,
2. φ *is satisfiable in* PA_2 *iff* $\dot\exists\, f_O\, f_S\, f_\oplus\, f_\otimes.\, \dot\exists\, P_\equiv.\, \mathsf{PA}_2' \mathrel{\dot\wedge} \varphi'$ *is satisfiable.*

Proof We obtain φ' and PA_2' by replacing function and predicate symbols with variables referencing the corresponding quantifier at the beginning. We only discuss (1), as (2) is proven in a similar way.

→ We have to show $\mathcal{M}', f_O \cdot f_S \cdot f_\oplus \cdot f_\otimes \cdot P_\equiv \cdot \rho \vDash \mathsf{PA}_2' \mathrel{\dot\rightarrow} \varphi'$ for all $\mathcal{M}'$, f_O, f_S, $f_\oplus$, $f_\otimes$, $P_\equiv$, and ρ. We construct a model $\mathcal{M}$ of PA_2 by using the domain of $\mathcal{M}'$ and interpreting O using f_O, S using f_S, and so on. We have $\mathcal{M} \vDash \mathsf{PA}_2$ because we can assume $\mathcal{M}', f_O \cdot f_S \cdot f_\oplus \cdot f_\otimes \cdot P_\equiv \cdot \rho \vDash \mathsf{PA}_2'$. Since φ is valid in PA_2, this yields $\mathcal{M}, \rho \vdash \varphi$ which finally gives us $\mathcal{M}', f_O \cdot f_S \cdot f_\oplus \cdot f_\otimes \cdot P_\equiv \cdot \rho \vDash \varphi'$.

← We get a model $\mathcal{M}$ of PA_2 and construct a model $\mathcal{M}'$ in Σ' by choosing an arbitrary interpretation. Then, we instantiate f_O, f_S, etc. in the assumption with the symbol interpretations of $\mathcal{M}$, yielding $\mathcal{M}', O^{\mathcal{I}} \cdot S^{\mathcal{I}} \cdot \oplus^{\mathcal{I}} \cdot \otimes^{\mathcal{I}} \cdot \equiv^{\mathcal{I}} \cdot \rho \vDash \mathsf{PA}_2' \mathrel{\dot\rightarrow} \varphi'$. This suffices, since the interpretations satisfy PA_2. □

This allows us to extend Lemma 5.11 to arbitrary signatures:

Lemma 5.15 *The existence of a sound, complete, and enumerable deduction system for $\mathfrak{F}_2^\mathsf{F}$ in any signature implies the enumerability of all closed first-order formulas that hold in $\mathbb{N}$.*

Proof By categoricity, a closed first-order formula φ holds in $\mathbb{N}$ iff it is valid in PA_2. Using Lemma 5.14 this is the case iff $\dot\forall\, f_O\ f_S\ f_\oplus\ f_\otimes.\,\dot\forall\, P_\equiv.\,\mathsf{PA}_2' \mathbin{\dot\to} \varphi'$ is second-order valid. Hence, the deduction system yields the enumerator we were looking for. □

Theorem 5.16 (Incompleteness for $\mathfrak{F}_2^\mathsf{F}$) *Assuming* MP*, the existence of a sound, complete, and enumerable deduction system for $\mathfrak{F}_2^\mathsf{F}$ in any signature implies the decidability of* H_{10}.

Proof Immediate by Lemma 5.10 and Lemma 5.15. □

Obtaining the same result for $\mathfrak{F}_2$ would be more tedious since the embedding would need to simulate function symbols by predicate variables. For example, the formula $S(x \oplus y) \equiv z$ would be turned into $\dot\exists a.\, P_\oplus(x, y, a) \mathbin{\dot\wedge} \dot\exists b.\, P_S(a, b) \wedge P_\equiv(b, z)$. Also, all previous results would need to be rephrased in terms of a relational interpretation of $\oplus$, $\otimes$ and so on. While possible in principle, we have not mechanized this as part of the thesis.

5.5 Further Undecidability Results

The H_{10} reduction can not only be used to obtain deduction-incompleteness, but also yields undecidability of other problems related to second-order logic. For example, since $\mathbb{N}$ is the only model of PA_2 because of categoricity, we know that validity in PA_2 is also undecidable:

Corollary 5.17 (Undecidability of PA_2) $p = q$ *has a solution iff* $\mathsf{PA}_2 \vDash \varphi_{p,q}$.

Proof Follows by Lemma 5.9, Lemma 4.8, and the fact that $\varphi_{p,q}$ is closed. □

While categoricity made this proof very convenient, it is not strictly necessary to obtain this result. Kirst and Hermes work in a first-order setting were categoricity is not available. They define a homomorphism $\mu : \mathbb{N} \to \mathcal{M}$ for arbitrary models of PA_1 to transport a solution α in the natural numbers to the model domain. However, categoricity allows us to obtain a solution to $p = q$ from any model that satisfies $\varphi_{p,q}$, not only $\mathbb{N}$. This is not possible in the first-order case.

Fact 5.18 $\mathcal{M} \vDash \varphi_{p,q}$ *for any model* $\mathcal{M}$ *of* PA_2 *implies that* $p = q$ *has a solution.*

Proof If $\mathcal{M} \vDash \varphi p, q$, we also have $\mathbb{N} \vDash \varphi_{p,q}$ by categoricity which yields a solution to $p = q$ by Lemma 5.9. □

This allows us to also obtain undecidability of satisfiability:

Corollary 5.19 (Undecidability of PA_2 satisfiability) *$p = q$ has a solution iff there is a model $\mathcal{M}$ of PA_2 and ρ with $\mathcal{M}, \rho \vDash \varphi_{p,q}$.*

Proof We show both directions separately:

- $\rightarrow$ If $p = q$ has a solution, then the standard model satisfies $\mathbb{N}, \rho \vDash \varphi_{p,q}$ for any ρ by Lemma 5.9.
- $\leftarrow$ Follows by Fact 5.18. □

Using the previous embedding idea, we can extend those results to undecidability of general validity and satisfiability in the empty signature using function quantifiers.

Corollary 5.20 (Undecidability of Second-order Logic)

1. *$p = q$ has a solution iff $\dot\forall f_O\, f_S\, f_\oplus\, f_\otimes .\, \dot\forall P_\equiv .\, \mathsf{PA}_2' \dot\rightarrow \varphi'_{p,q}$ is valid in the empty signature.*
2. *$p = q$ has a solution iff $\dot\exists f_O\, f_S\, f_\oplus\, f_\otimes .\, \dot\exists P_\equiv .\, \mathsf{PA}_2' \dot\wedge \varphi'_{p,q}$ is satisfiable in the empty signature.*

Like undecidability in $\mathbb{N}$, those results can also be extended to refute enumerability:

Theorem 5.21 (Enumerability) *Assuming MP, enumerability of the following problems implies decidability of H_{10}:*

1. *Validity in PA_2*
2. *Satisfiability in PA_2*
3. *Second-order validity in the empty signature*
4. *Second-order satisfiability in the empty signature*

Proof (3) and (4) follow from (1) and (2) using the signature embedding.

1. By Corollary 5.17 it suffices to show that $\lambda pq.\, \mathsf{PA}_2 \vDash \varphi_{p,q}$ is decidable. By Post's theorem (Fact 2.7) it suffices to show bi-enumerability:
   - $\lambda pq.\, \mathsf{PA}_2 \vDash \varphi_{p,q}$ is enumerable if validity in PA_2 is enumerable.
   - φ is not valid in PA_2 iff $\dot\neg\varphi$ is valid in PA_2, since if $\varphi_{p,q}$ does not hold in all models of PA_2, it holds in none of them because of categrocity (Lemma 4.8). Thus, co-enumerability of the problem also follows from enumerability of validity in PA_2.
2. Similar to (1) using the fact that φ is not satisfiable in PA_2 iff $\dot\neg\varphi$ is satisfiable in PA_2 because of categoricity. □

Chapter 6

Henkin Semantics

Henkin introduced the notion of *general models*, now known as *Henkin models*, in 1950 to show completeness for the theory of types [26]. Type theory, being more expressive than second-order logic, suffered from the same incompleteness problems for standard semantics we discussed in the previous chapter. However, his newly introduced semantics allowed him to obtain a completeness result that also scales down to higher-order and especially second-order logic. Roughly, the idea is that second-order quantifiers no longer range over *all* predicates of a given domain, but only over a some subset of them that is provided by the model.

6.1 Definition of Henkin Semantics

A Henkin model $\mathcal{H}$ consists of the same components as a model in standard semantics, except that it also specifies a universe $\mathbb{U}$ that contains the predicates that the second-order quantifiers should range over. However, we do not want this universe to be chosen completely arbitrary. For example, we do not want it to be empty. More concretely, we require that it should at least contain all properties that are second-order definable. Otherwise, the comprehension axiom of $\vdash$ would not be sound. Hence, we want Henkin models to satisfy $\mathcal{H} \vDash \mathsf{Compr}^n_\varphi$ for all φ and n. This way we guarantee that all second-order definable predicates are contained in $\mathcal{H}$.

Remark 6.1 (Terminology) *Interestingly, the terminology varies a bit in the literature. While Väänänen [67] considers a model only "Henkin" if it satisfies comprehension and calls them "general models" otherwise, Shapiro [56] always speaks of "Henkin models" and calls them "faithful" if comprehension holds. Leivant [38] proceeds similarly to Väänänen but uses the term "Henkin-prestructure". We will use the terminology of Väänänen, so a general model is considered a Henkin model if it has comprehension.*

Using this intuition, we can now formally define Henkin semantics:

Definition 6.2 (Henkin Semantics) *A Henkin model $\mathcal{H}$ consists of a domain $D : \mathbb{T}$, an interpretation function $\mathcal{I}$ for function and predicate symbols and a set of relations $\mathbb{U}_n$:*

$(D^n \to \mathbb{P}) \to \mathbb{P}$ for each $n : \mathbb{N}$. The predicate interpretations should be included in $\mathbb{U}$, in other words $\mathbb{U}_n\ \mathcal{P}^{\mathcal{I}}_n$ for all n-ary predicate symbols $P_n : \mathcal{P}_\Sigma$.

The model must satisfy $\mathcal{H} \vDash_H \mathsf{Compr}^n_\varphi$ for all φ and n, where $\vDash_H$ is defined by

$$\mathcal{H}, \rho \vDash_H \dot\forall^n_p\ \varphi := \forall P : D^n \to \mathbb{P}.\ \mathbb{U}_n\ P \to \mathcal{H}, P \cdot \rho \vDash_H \varphi$$
$$\mathcal{H}, \rho \vDash_H \dot\exists^n_p\ \varphi := \exists P : D^n \to \mathbb{P}.\ \mathbb{U}_n\ P \wedge \mathcal{H}, P \cdot \rho \vDash_H \varphi$$

The remaining cases and term evaluation $[\![\cdot]\!]^{\mathcal{H}}_\rho$ are defined in the same way as for standard semantics. We say a formula φ is valid in $\mathcal{H}$, written $\mathcal{H} \vDash_H \varphi$, if $\mathcal{H}, \rho \vDash_H \varphi$ for all environments ρ with $\mathbb{U}_n\ (\rho_p\ x\ n)$ for all $x, n : \mathbb{N}$. We call such environments Henkin environments. In the remainder of this thesis we will mostly work with Henkin semantics. Therefore, we will write $\vDash$ instead of $\vDash_H$ if it is clear from the context that we are in the Henkin setting.

Henkin semantics can also be generalized to the fragment $\mathfrak{F}^F_2$. In this case, a model also contains a set of function universes $\mathbb{F}_n : (D^n \to D) \to \mathbb{P}$ and must satisfy function comprehension. In the remainder of this thesis, we will however only focus on $\mathfrak{F}_2$.

By looking at the definition of satisfiability above, we can see that Henkin semantics agree with the standard semantics if $\mathbb{U}_n\ P$ for all $P : D^n \to \mathbb{P}$.[1] A model in standard semantics can therefore be seen as a Henkin model that accepts all possible predicates over the domain. This is also the reason why models in standard semantics are sometimes called *full models*. They are "full" in the sense that $\mathbb{U}$ contains every predicate.

Fact 6.3 *Let $\mathcal{M}$ be a model in standard semantics. Then we can also interpret M as a Henkin model with $\mathbb{U}_n\ P := \top$. It holds that $\mathcal{M}, \rho \vDash \varphi$ iff $\mathcal{M}, \rho \vDash_H \varphi$ for all ρ and φ.*

Proof By induction on φ with ρ generalized. □

Therefore, we can see that $\mathcal{T} \vDash_H \varphi \to \mathcal{T} \vDash \varphi$. The converse does not hold, since there are many more Henkin models than there are models for standard semantics, so there are more models available that could potentially make φ invalid and disprove $\mathcal{T} \vDash_H \varphi$. This also gives a rough intuition why Henkin semantics could be (deduction) complete: We hope that the valid but unprovable formulas for standard semantics are invalidated by the additional Henkin models.

On the other hand, completeness means that we need to give up on the idea of categoricity of infinite structures since categoricity yields incompleteness as seen in Chapter 5. We also see why this might be the case: There are now many more models that can potentially be non-isomorphic and behave differently compared

[1] Comprehension is trivially satisfied by such a model.

to each other. It seems like categoricity and completeness "pull in opposite directions" [52] and we cannot have both at the same time. So in a way, Henkin semantics trade some of the expressivity of second-order logic and its power to categorically describe infinite structures for the ability to have a complete proof system.

6.2 Soundness of the Deduction System

The deduction system we presented in Chapter 3 is sound for Henkin Semantics:

Theorem 6.4 (Soundness for Henkin Semantics) *Let* T *be a theory and* φ *a formula. Then*

1. $\mathcal{T} \vdash_i \varphi \to \mathcal{T} \vDash_H \varphi$
2. $\mathcal{T} \vdash_c \varphi \to \mathcal{T} \vDash_H \varphi$ *under* LEM

Proof Similar to the proof of Theorem 3.23. The comprehension axiom is sound since every Henkin model $\mathcal{H}$ satisfies $\mathcal{H} \vDash \mathsf{Compr}^n_\varphi$ by definition. □

In the next chapters, we will show that $\vdash$ is in fact complete, if we interpret it in Henkin semantics.

Chapter 7

Henkin Completeness: Translation to First-Order Logic

The completeness theorem, originally due to Gödel [20] and simplified by Henkin [25], is one of the hallmarks of first-order logic. It states that the natural deduction system $\vdash_1$ is complete for first-order Tarski semantics. This theorem has already been mechanized in Coq, for example Forster, Kirst, and Wehr [15] show:

Theorem 7.1 (Forster, Kirst and Wehr [15]: Completeness of $\vdash_1$) $\mathcal{T} \vDash_1 \varphi$ *implies* $\mathcal{T} \vdash_1 \varphi$ *for all* $\mathcal{T}$ *and* φ *under* LEM.

In Chapter 6 we claimed that Henkin semantics allow for a similar result to be obtained for second-order logic. While the usual Henkin style completeness proof used for first-order logic can also be adapted to the second-order case [56], we want to follow a different strategy in this thesis. Interestingly, the switch to Henkin semantics not only yields completeness, but also brings many of the other (in)famous properties of first-order logic like compactness or the Löwenheim-Skolem theorems with it [56]. As it turns out, second-order logic interpreted in Henkin semantics actually reduces to first-order logic [41, 64, 43]. By this we mean that given a second-order formula φ, we can construct a first-order formula $\varphi^\star$, such that

1. φ is valid in Henkin semantics iff $\varphi^\star$ is valid in first-order Tarski semantics. We call this *semantic reduction*.
2. φ is provable in the second-order natural deduction system iff $\varphi^\star$ is provable in the first-order system. We call this *deductive reduction*.

This does not come as a surprise if we consider Lindström's theorem [39] from abstract model theory: First-order logic is the strongest logic that has both the compactness property and the (downward) Löwenheim-Skolem theorem. Thus, second-order logic with Henkin semantics cannot be more expressive than first-order logic, which is exactly what is captured by this reduction.

$$\begin{tikzcd}
\mathcal{T} \vDash_2 \varphi \arrow[d, "(1)"'] & \mathcal{T} \vdash_2 \varphi \\
\mathcal{T}^\star \vDash_1 \varphi^\star \arrow[r, "\text{Theorem 7.1}"] & \mathcal{T}^\star \vdash_1 \varphi^\star \arrow[u, "(2)"']
\end{tikzcd}$$

Figure 7.1: Outline of how completeness of $\vdash_2$ can be derived using the reduction.

As illustrated in Figure 7.1, we can use this reduction to transport the completeness theorem from first-order logic to second-order logic.[1] Note, that we write $\vdash_2$ and $\vDash_2$ for the second-order natural deduction system and Henkin validity to make the distinction between first- and second-order derivations and the different semantics clearer.

This chapter will mostly focus on the intuition behind the reduction (Section 7.1) and the definition of the actual translation function (Section 7.2). We verify the semantic reduction property (1) in Chapter 8 and the deductive one (1) in Chapter 9. Finally, we obtain completeness and further meta-theoretic properties in Chapter 10.

7.1 Intuition

To begin with, we want to remark that there are different flavours of first-order logic. For us, the distinction between mono-sorted and many-sorted first-order logic is of particular interest. The version we introduced in Chapter 3 is the mono-sorted one. That means that in formulas like $\dot\forall xy.\,\varphi$ the variables x and y always range over the same universe of objects. On the other hand, many-sorted first-order logic introduces the notion of sorts that are used to annotate the universes the quantifiers range over. For example, in the many-sorted formula $\dot\forall x^{\mathcal{S}_1}.\dot\forall y^{\mathcal{S}_2}\,\varphi$, the variable x belongs to the sort $\mathcal{S}_1$ and y belongs to $\mathcal{S}_2$. Interestingly, this feature allows for a straightforward embedding of second-order logic, for example as presented in [41]:

Example 7.2 (Reduction to Many-Sorted First-Order Logic) *We use a sort $\mathcal{I}$ for individuals and sorts $\mathcal{P}_n$ for predicates of arity n. Then we can represent the second-order formula $\dot\forall P.\,\dot\exists Q.\,\dot\forall x.\,P(x) \dot\leftrightarrow \dot\neg Q(x)$ as the many-sorted formula*

$$\dot\forall P^{\mathcal{P}_1}.\,\dot\exists Q^{\mathcal{P}_1}.\,\dot\forall x^{\mathcal{I}}.\,\mathsf{App}_1(P,x) \dot\leftrightarrow \dot\neg\mathsf{App}_1(Q,x)$$

We simply turned the quantifiers into first-order ones and annotated them with the correct

[1] The figure also shows that already the forwards direction of the semantic reduction (1) and the backwards direction of (2) suffice to obtain completeness of $\vdash_2$. However, we will show both directions respectively.

sort. Since first-order logic does not have a notion of predicate variable applications, we added the custom symbol App *to the signature that represents this operation.*

In this example we used the fact that first-order sorts can capture the different kinds of quantifiers we have in second-order logic. However, in this thesis we want to focus on the mono-sorted case because it is the more traditional formalism and many of its properties (including completeness [15]) have already been mechanized. On top of that, the reduction to mono-sorted first-order logic represents the stronger result. Luckily, it is also well known that the many-sorted case can be reduced to the mono-sorted one [41]. The basic idea is that sorts can be simulated by custom predicates that guard the quantifiers. But as it turns out, this folklore technique is more difficult to work with than it might seem at first glance. For example, consider Van Dalen's approach, who uses this idea to extend the reduction from Example 7.2 to mono-sorted first-order logic [64]:

Example 7.3 (Van Dalen's Reduction to Mono-Sorted First-Order Logic)
We can represent the formula from Example 7.2 as the following mono-sorted first-order formula:

$$\dot\forall P.\ \mathsf{isPred}_1(P) \mathbin{\dot\rightarrow} \dot\exists Q.\ \mathsf{isPred}_1(Q) \mathbin{\dot\wedge} \dot\forall x.\ \mathsf{isIndi}(x) \mathbin{\dot\rightarrow} (\mathsf{App}_1(P,x) \mathbin{\dot\leftrightarrow} \dot\neg\mathsf{App}_1(Q,x))$$

The symbols isPred_1 *and* isIndi *assert that* P *and* Q *should be unary predicates and that* x *should be an individual.*

While proving the semantic reduction property for this translation is straightforward, showing the deductive part is challenging. Van Dalen mentions that there is a "tedious but routine proof" [64], but he only gives a very brief sketch. Nour and Raffalli investigated this claim and "were not able to end his proof" [43]. They point out the problem, that this translation is not surjective. This means that the formulas occurring in a proof of $\varphi^\star$ do not necessarily have the shape $\psi^\star$. Therefore, it is not obvious how to translate such a proof in the first-order system to the second-order one (at least we are not aware of any proposed solution to this problem and also were not able to come up with one on our own).

Luckily, Nour and Raffalli suggest a slightly simpler reduction that avoids this issue by staying closer to the original structure of the second-order formula [43]. Essentially, they get rid of the isIndi and isPred symbols from the previous translation:

Example 7.4 (Nour and Raffalli's Reduction to Mono-Sorted First-Order Logic)
Nour and Raffalli represent the formula from Example 7.2 as the following mono-sorted first-order formula:

$$\dot\forall P.\ \dot\exists Q.\ \dot\forall x.\ (\mathsf{App}_1(P,x) \mathbin{\dot\leftrightarrow} \dot\neg\mathsf{App}_1(Q,x))$$

Here, P*,* Q*, and* x *represent individuals and predicates of all all arities at the same time. The semantics of the* App *symbol then interprets them differently based on their position in the arguments: As* P *and* Q *are the first argument they are interpreted as predicates, whereas* x *will be interpreted as an individual.*

Importantly, Nour and Raffalli work in a variant of second-order logic without function quantifiers. While the semantic reduction can be extended to also work in $\mathfrak{F}_2^F$, doing the same for the deductive part is challenging and would be very difficult to mechanize (see Remark 9.6). Thus, the mechanization and the following presentation on paper are both only concerned with the fragment $\mathfrak{F}_2$.

7.2 Translation Function to First-Order Logic

The first step in formalizing the reduction is to define the translation function to first-order logic described in Example 7.4. For this, we first fix a signature Σ in which we will work for the remainder of this chapter. We also assume that Σ is discrete and enumerable.

As illustrated in the previous examples, the translation requires us to replace applications of predicate variables with corresponding first-order primitives. Therefore, we extend the signature with custom symbols that represents this operation:

Definition 7.5 (Extended Signature) *We obtain the extended signature* Σ_+ *by adding* $(n+1)$*-ary predicate symbols* App_n *for all* $n : \mathbb{N}$*, representing the application of* n*-ary predicate variables.*

Notice that this leads to an infinite blow-up of the signature, however we keep discreteness and enumerability:

Fact 7.6 Σ_+ *is discrete and enumerable.*

Another important step that is not very evident in the example is how to turn the second-order variables into first-order ones. This is crucial because individual and predicate variables of different arities are completely independent from each other in second-order logic. But now, we need to map all of them into the single variable space provided by mono-sorted first-order logic. This was easy in our informal example, since we only needed to choose a unique name for each variable. However, implementing this idea in our de Bruijn encoding is more challenging. Essentially, we need to keep track of which first-order de Bruijn index a given second-order variable corresponds to. To achieve this, we use functions

$$\pi_i : \mathbb{N} \to \mathbb{N} \qquad\qquad \pi_p : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

that return the corresponding first-order index for each variable. An individual variable x_i then gets turned into the first-order variable $\pi_i\ i$ and a predicate variable

p_i^n gets turned into $\pi_p\, i\, n$. We carry those functions throughout the whole translation process and update them on each quantifier. For example, when we turn an individual quantifier into a first-order one, we want that the individual variable 0 is still bound to this quantifier. However, all other variables need to be shifted in order to make space for the new one. We write this transformation as $\uparrow_i \pi$. Similarly, the transformation for a predicate quantifier of arity n will be written as $\uparrow_p^n \pi$. The formal definition of those operations are given by

$$\begin{aligned} (\uparrow_i \pi_i)\, 0 &:= 0 \\ (\uparrow_i \pi_i)\, (Sx) &:= S\, (\pi_i\, x) \\ (\uparrow_i \pi_p)\, x\, n &:= Sx \end{aligned} \qquad \begin{aligned} (\uparrow_p^n \pi_i)\, x &:= Sx \\ (\uparrow_p^n \pi_p)\, 0\, n &:= 0 \\ (\uparrow_p^n \pi_p)\, (Sx)\, n &:= S\, (\pi_p\, x\, n) \\ (\uparrow_p^n \pi_p)\, x\, m &:= m \qquad\qquad \text{if } m \neq n \end{aligned}$$

Now we have all ingredients to define our translation function:

Definition 7.7 (Translation) *The translation function $\_^\star_\pi : \mathfrak{F}_2(\Sigma) \to \mathfrak{F}_1(\Sigma_+)$ is recursively defined by:*

$$\begin{aligned} \bot^\star_\pi &:= \bot \\ (p_i^n\, \boldsymbol{v})^\star_\pi &:= \mathsf{App}_n(x_{\pi_p\, i\, n} :: \boldsymbol{v}^\star_\pi) \\ (\mathcal{P}\, \boldsymbol{v})^\star_\pi &:= \mathcal{P}\, \boldsymbol{v}^\star_\pi \end{aligned} \qquad \begin{aligned} (\varphi \mathbin{\dot\Box} \psi)^\star_\pi &:= \varphi^\star_\pi \mathbin{\dot\Box} \psi^\star_\pi \\ (\dot\nabla\, \varphi)^\star_\pi &:= \dot\nabla\, \varphi^\star_{\uparrow_i \pi} \\ (\dot\nabla_p^n\, \varphi)^\star_\pi &:= \dot\nabla\, \varphi^\star_{\uparrow_p^n \pi} \end{aligned}$$

where the $\_^\star_\pi$ notation is extended to terms and vectors of terms by:

$$(x_i)^\star_\pi := x_{\pi_i\, i} \qquad\qquad (f\, \boldsymbol{v})^\star_\pi := f\, \boldsymbol{v}^\star_\pi$$

The remaining question is what the initial values for π should be when we start the translation. A first observation is that this initial choice only effects the free variables in the formula since the $\uparrow$ operations fix the values for bound variables. Therefore, the choice of π would not matter if we restricted ourselves to closed formulas:

Lemma 7.8 $\varphi^\star_\pi = \varphi^\star_{\pi'}$ *for all closed formulas φ and arbitrary π and π'.*

Proof It suffices to show $(\forall x.\ \sup(\varphi) \not\leq x \to \pi_i\, x = \pi'_i\, x) \to (\forall x n.\, \sup_p^n(\varphi) \not\leq x \to \pi_p\, x\, n = \pi'_i\, x\, n) \to \varphi^\star_\pi = \varphi^\star_{\pi'}$, by induction on φ. □

However, since we also want to work with open formulas, choosing an arbitrary function would not be adequate. For the translation to go through, we need to make sure that each free individual and predicate variable gets mapped to a unique first-order variable. We opt for using the natural pairing function to achieve this:

Definition 7.9 *We define $\pi_i^0\, x := \langle 0, x\rangle$ and $\pi_p^0\, x\, n := \langle 1, \langle x, n\rangle\rangle$ and write $\varphi^\star$ for $\varphi^\star_{\pi^0}$.*

Now that the translation function is defined, we can continue with the verification of the reduction.

Chapter 8

Henkin Completeness: Semantic Reduction Property

In this chapter, we verify the semantic part of the reduction. We want to show that validity of a formula φ in Henkin semantics agrees with the validity of $\varphi^\star$ in first-order Tarski semantics. As discussed in the introduction of Chapter 7, it would technically suffice to prove only one direction of this to obtain completeness. That is, we need to be able to translate first-order models $\mathcal{M}$ into Henkin models $\mathcal{M}^\diamond$. Nour and Raffalli only proof this result. However, we are also interested in proving the other direction, such that we end up with a "real" semantic reduction. That is, we also want to translate Henkin models $\mathcal{H}$ into first-order models $\mathcal{H}^\star$.

We begin with this latter direction in Section 8.1, as we believe it is instructive in understanding the idea behind Nour and Raffali's translation. In Section 8.2, we show the converse direction and combine both results in Section 8.3.

8.1 Translating Henkin Models into First-Order Models

Suppose we have a Henkin model $\mathcal{H}$ that consists of a domain D, a predicate universe $\mathbb{U}$ and a symbol interpretation $\mathcal{I}$. We also assume that $\mathcal{H}$ is not empty, i.e. there is a $d_0 : D$. By comprehension, we also get $P_0^n : D^n \to \mathbb{P}$ with $\mathbb{U}_n\ P_0^n$ for each n.

Recall from the initial motivation that first-order variables represent individuals and predicates at the same time. Therefore, the translated first-order domain $D^\star$ should contain both the individuals and the predicates of $\mathcal{H}$:

Definition 8.1 *We define* $D^\star$ *as an inductive type by*

$$\mathsf{fromIndi} : D \to D^\star \qquad\qquad \mathsf{fromPred}_n : \forall P.\, \mathbb{U}_n\ P \to D^\star$$

We will leave out the proof argument of $\mathsf{fromPred}_n$ *on paper.*

The problem we face now is that an object in $D^\star$ is either an individual or a predicate, but not both at the same time like our reduction requires. Thankfully, the Henkin model is not empty, so we can define that each predicate simultaneously represents the dummy value d_0 and each individual represents all the dummy predicates P_0^n. To this end we define the following interpretation functions:

Definition 8.2 *We define* $\mathsf{toIndi} : D^\star \to D$ *and* $\mathsf{toPred}_n : D^\star \to (D^n \to \mathbb{P})$ *by*

$$\begin{aligned} \mathsf{toIndi}\,(\mathsf{fromIndi}\,d) &:= d \\ \mathsf{toIndi}\,(\mathsf{fromPred}_n\,\_) &:= d_0 \end{aligned} \qquad \begin{aligned} \mathsf{toPred}_n\,(\mathsf{fromIndi}\,\_) &:= P_0^n \\ \mathsf{toPred}_n\,(\mathsf{fromPred}_n\,P) &:= P \\ \mathsf{toPred}_n\,(\mathsf{fromPred}_m\,\_) &:= P_0^n \quad \textit{if}\ n \neq m \end{aligned}$$

Definition 8.3 *The first-order symbol interpretation* $\mathcal{I}^\star$ *is given by*

$$\begin{aligned} \mathcal{F}^{\mathcal{I}^\star}\,\boldsymbol{v} &:= \mathsf{fromIndi}\,(\mathcal{F}^{\mathcal{I}}\,(\mathsf{toIndi}\,\boldsymbol{v})) \\ \mathcal{P}^{\mathcal{I}^\star}\,\boldsymbol{v} &:= \mathcal{P}^{\mathcal{I}}(\mathsf{toIndi}\,\boldsymbol{v}) \\ \mathsf{App}_n^{\mathcal{I}^\star}\,(d :: \boldsymbol{v}) &:= \mathsf{toPred}_n\,d\,(\mathsf{toIndi}\,\boldsymbol{v}) \end{aligned}$$

The last step is to translate Henkin environments ρ into first-order environments $\rho^\star$. Here, we need to make sure that $\rho^\star$ maps the free variables in a translated formula $\varphi^\star$ to the same values as ρ does in φ. Recall that the free variables are translated according to the initial π functions. Therefore, $\rho^\star$ needs to reverse π_i^0 and π_p^0 as defined in 7.7:

Definition 8.4 *Let* ρ *be a Henkin environment. We define* $\rho^\star$ *by*

$$\rho^\star\,n := \begin{cases} \mathsf{fromIndi}\,(\rho_i\,x) & \textit{if}\ n = \langle 0, x\rangle \\ \mathsf{fromPred}_m\,(\rho_p\,x\,m) & \textit{if}\ n = \langle S\,\_, \langle x, m\rangle\rangle \end{cases}$$

Lemma 8.5 *For all Henkin environments* ρ *it holds that*

1. $\forall x.\,\mathsf{toIndi}\,(\rho^\star\,(\pi_i^0\,x)) = \rho_i\,x$
2. $\forall xn.\,\mathsf{toPred}_n(\rho^\star\,(\pi_p^0\,x\,n)) = \rho_p\,x\,n$

Proof Immediate by the correctness of the pairing function. □

Now that we have all components of our translated model $\mathcal{H}^\star$, we can show that it agrees with the initial model $\mathcal{H}$:

Lemma 8.6 *For all Henkin environments* ρ *and second-order formulas* φ *it holds that*

$$\mathcal{H}^\star, \rho^\star \vDash_1 \varphi^\star \;\leftrightarrow\; \mathcal{H}, \rho \vDash_2 \varphi.$$

Proof We show the stronger claim

$$\begin{aligned}\forall \rho_1\, \rho_2 \pi.\ &(\forall x.\, \mathsf{toIndi}\,(\rho_1\ (\pi_i\, x)) = \rho_{2,i}\, x)\\ &\to\ (\forall x n.\, \mathsf{toPred}_n\,(\rho_1\ (\pi_p\, x\, n)) = \rho_{2,p}\, x\, n)\\ &\to\ (\mathcal{H}^\star, \rho_1 \vDash_1 \varphi^\star_\pi \leftrightarrow \mathcal{H}, \rho_2 \vDash_2 \varphi)\end{aligned}$$

by induction on φ, which suffices according to Lemma 8.5:

- If $\varphi = \mathcal{P}\, \boldsymbol{v}$, we have $\varphi^\star_\pi = \mathcal{P}\, \boldsymbol{v}^\star_\pi$ and need to show $\mathcal{P}^{\mathcal{I}}\ (\mathsf{toIndi}\ \llbracket \boldsymbol{v}^\star_\pi \rrbracket^{\mathcal{H}^\star}_{\rho_1}) \leftrightarrow \mathcal{P}^{\mathcal{I}}\ \llbracket \boldsymbol{v} \rrbracket^{\mathcal{H}}_{\rho_2}$. It suffices to prove

  $$\forall \rho_1\, \rho_2 \pi.\ (\forall x.\, \mathsf{toIndi}\,(\rho_1\ (\pi_i\, x)) = \rho_{2,i}\, x) \to \mathsf{toIndi}\ \llbracket t^\star_\pi \rrbracket^{\mathcal{H}^\star}_{\rho_1} = \llbracket t \rrbracket^{\mathcal{H}}_{\rho_2}$$

  for all terms t by induction on t. This implies $\mathsf{toIndi}\ \llbracket \boldsymbol{v}^\star_\pi \rrbracket^{\mathcal{H}^\star}_{\rho_1} = \llbracket \boldsymbol{v} \rrbracket^{\mathcal{H}}_{\rho_2}$.

- If $\varphi = p^n_i\, \boldsymbol{v}$, we have $\varphi^\star_\pi = \mathsf{App}_n(x_{\pi_p\, i\, n} :: \boldsymbol{v}^\star_\pi)$ and need to show

  $$\mathsf{toPred}_n\ (\rho_1 (\pi_p\, i\, n))\ (\mathsf{toIndi}\ \llbracket \boldsymbol{v}^\star_{\pi_i, \pi_p} \rrbracket^{\mathcal{H}^\star}_{\rho_1}) \leftrightarrow \rho_{2,p}\, i\, n\ \llbracket \boldsymbol{v} \rrbracket^{\mathcal{H}}_{\rho_2}.$$

  By assumption we have $\mathsf{toPred}_n\ (\rho_1 (\pi_p\, i\, n)) = \rho_{2,p}\, i\, n$ and again $\mathsf{toIndi}\ \llbracket \boldsymbol{v}^\star_\pi \rrbracket^{\mathcal{H}^\star}_{\rho_1} = \llbracket \boldsymbol{v} \rrbracket^{\mathcal{H}}_{\rho_2}$ like in the previous case.

- If $\varphi = \dot\forall_i\, \psi$, we have $\varphi^\star_\pi = \dot\forall\, \psi^\star_{\uparrow_i \pi}$ and need to show

  $$\left(\forall a : D^\star.\, \mathcal{H}^\star, a \cdot \rho_1 \vDash_1 \psi^\star_{\uparrow_i \pi}\right) \leftrightarrow (\forall d : D.\, \mathcal{H}, d \cdot \rho_2 \vDash_2 \psi)$$

  - $\rightarrow$ Let $d : D$. From the assumption we get $\mathcal{H}^\star, (\mathsf{fromIndi}\ d) \cdot \rho_1 \vDash_1 \psi^\star_{\uparrow_i \pi}$ which is equivalent to $\mathcal{H}, d \cdot \rho_2 \vDash_2 \psi$ by the inductive hypothesis. The preconditions of the inductive hypothesis hold since the updated functions $\uparrow_i \pi_i$ and $\uparrow_i \pi_p$ match the updated environments $(\mathsf{fromIndi}\ d) \cdot \rho_1$ and $d \cdot \rho_2$.

  - $\leftarrow$ Let $a : D^\star$. If $a = \mathsf{fromIndi}\, d$, then instantiate the assumption with d. If $a = \mathsf{fromPred}_n\, P$ then instantiate with d_0. Conclude again using the inductive hypothesis.

- If $\varphi = \dot\forall^n_p\, \psi$, we have $\varphi^\star_\pi = \dot\forall\, \psi^\star_{\uparrow^n_p \pi}$ and need to show

  $$\left(\forall a : D^\star.\, \mathcal{H}^\star, a \cdot \rho_1 \vDash_1 \psi^\star_{\uparrow^n_p \pi}\right) \leftrightarrow (\forall P : D^n \to \mathbb{P}.\, \mathbb{U}_n\, P \to \mathcal{H}, P \cdot \rho_2 \vDash_2 \psi)$$

  - $\rightarrow$ Let $P : D^n \to \mathbb{P}$. Instantiate the assumption with $a := \mathsf{fromPred}_n\, P$ and conclude using the inductive hypothesis.

  - $\leftarrow$ Let $a : D^\star$. If $a = \mathsf{fromPred}_n\, P$, then instantiate the assumption with P. Since $\mathbb{U}_n\, P$, we can finish with the inductive hypothesis. If $a = \mathsf{fromIndi}\, d$ or $a = \mathsf{fromPred}_m\, P$ with $m \neq n$, then instantiate with P^n_0 and conclude again with the inductive hypothesis as $\mathbb{U}_n\, P^n_0$ by assumption.

The remaining cases are straightforward or are proven in a similar way. □

8.2 Translating First-Order Models into Henkin Models

Next, we verify the converse direction. Suppose we have a first-order model $\mathcal{M}$ that consists of a domain D and a symbol interpretation I. Now, we want to turn this into a Henkin model $\mathcal{M}^\diamond$.

Since each object in D represents, among other things, an individual we can choose the same domain for our Henkin model:

Definition 8.7 *We set* $D^\diamond := D$.

We also need to specify a predicate universe for the Henkin model. Here, we take the set of predicates induced by the interpretation of the App symbol:

Definition 8.8 *We set* $\mathbb{U}^\diamond_n\, P := \exists d : D.\, \forall \boldsymbol{v}.\, P\, v \leftrightarrow \mathsf{App}^{\mathcal{I}}_n\,(d :: \boldsymbol{v})$.

In other words, a predicate is contained in the Henkin model if $\mathcal{M}$ contains an object whose application extensionally behaves the same as the predicate. At this point we can also nicely see why the reduction would fail for full semantics. Recall that we can interpret models in standard semantics as Henkin models that contain all predicates. Therefore, $\mathbb{U}^\diamond_n$ would need to accept every predicate, which is not always the case, since we have no guarantee that $\mathcal{M}$ actually contains all of them.

Another important point is that we want our Henkin model to fulfill comprehension. By looking at the definition of $\mathbb{U}^\diamond_n$, we can see that this is only the case if $\mathcal{M}$ has comprehension through the App symbol. Unfortunately this is not always the case. If we do not restrict $\mathcal{M}$ to have comprehension, $\mathcal{M}^\diamond$ will only yield a general model. However, before tackling this problem, we first want to finish the translation.

For the symbol interpretation, we can simply reuse the interpretation from $\mathcal{M}$:

Definition 8.9 *The symbol interpretation* $\mathcal{I}^\diamond$ *is given by* $\mathcal{P}^{\mathcal{I}^\diamond}\, \boldsymbol{v} := \mathcal{P}^{\mathcal{I}}\, \boldsymbol{v}$ *and* $\mathcal{F}^{\mathcal{I}^\diamond}\, \boldsymbol{v} := \mathcal{F}^{\mathcal{I}}\, \boldsymbol{v}$.

Finally, we need to translate a first-order environment ρ into a second-order environment $\rho^\diamond$:

Definition 8.10 *Given a first-order environment* ρ *we define* $\rho^\diamond := (\rho^\diamond_i, \rho^\diamond_p)$ *by*

$$\rho^\diamond_i\, x := \rho\,(\pi^0_i\, x) \qquad\qquad \rho^\diamond_p\, x\, n := \lambda \boldsymbol{v}.\, \mathsf{App}^{\mathcal{I}}_n(\rho\,(\pi^0_p\, x\, n) :: \boldsymbol{v})$$

Lemma 8.11 $\rho^\diamond$ *is a Henkin environment, i.e.* $\mathbb{U}^\diamond_n(\rho^\diamond_p\, x\, n)$ *for all* x *and* n.

Proof We need to find an object $d : D$ whose predicate application behaves the same as $\rho^\diamond_p\, x\, n$. Simply choose $d = \rho\,(\pi^0_p\, x\, n)$. □

Now, we can prove the analogue of Lemma 8.6:

Lemma 8.12 *For all first-order environments* ρ *and second-order formulas* φ *it holds that*

$$\mathcal{M}, \rho \vDash_1 \varphi^\star \leftrightarrow \mathcal{M}^\diamond, \rho^\diamond \vDash_2 \varphi.$$

Proof We show the stronger claim

$$\begin{aligned}&\forall \rho_1\, \rho_2 \pi.\, (\forall x.\, \rho_{2,i}\, x = \rho_1\, (\pi_i\, x))\\ &\quad \to\; (\forall x n \boldsymbol{v}.\, \rho_{2,p}\, x\, n\, \boldsymbol{v} \leftrightarrow \mathsf{App}^{\mathcal{I}}_n (\rho_1 (\pi_p\, x\, n) :: \boldsymbol{v}))\\ &\quad \to\; (\mathcal{M}, \rho_1 \vDash_1 \varphi^\star_\pi \leftrightarrow \mathcal{M}^\diamond, \rho_2 \vDash_2 \varphi)\end{aligned}$$

by induction on φ, which trivially suffices:

- If $\varphi = \mathcal{P}\,\boldsymbol{v}$, we have $\varphi^\star_\pi = \mathcal{P}\,\boldsymbol{v}^\star$ and need to show $\mathcal{P}^{\mathcal{I}}\, [\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1} \leftrightarrow \mathcal{P}^{\mathcal{I}}\, [\![\boldsymbol{v}]\!]^{\mathcal{M}^\diamond}_{\rho_2}$. It suffices to prove

  $$\forall \rho_1\, \rho_2 \pi.\, (\forall x.\, \rho_{2,i}\, x = \rho_1\, (\pi_i\, x)) \to \mathsf{toIndi}\, [\![t^\star_\pi]\!]^{\mathcal{M}}_{\rho_1} = [\![t]\!]^{\mathcal{M}^\diamond}_{\rho_2}$$

  for all terms t by induction on t. This implies $[\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1} = [\![\boldsymbol{v}]\!]^{\mathcal{M}^\diamond}_{\rho_2}$.

- If $\varphi = p^n_i\, \boldsymbol{v}$, we have $\varphi^\star_\pi = \mathsf{App}_n(x_{\pi_p\, i\, n} :: \boldsymbol{v}^\star)$ and need to show

  $$\mathsf{App}^{\mathcal{I}}_n\, (\rho_1 (\pi_p\, i\, n) :: [\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1}) \leftrightarrow \rho_{2,p}\, x\, n\, [\![\boldsymbol{v}]\!]^{\mathcal{M}^\diamond}_{\rho_2}.$$

  By assumption we have $\mathsf{App}^{\mathcal{I}}_n\, (\rho_1 (\pi_p\, i\, n) :: [\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1}) \leftrightarrow \rho_{2,p}\, i\, n\, [\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1}$ and again $[\![\boldsymbol{v}^\star_\pi]\!]^{\mathcal{M}}_{\rho_1} = [\![\boldsymbol{v}]\!]^{\mathcal{M}^\diamond}_{\rho_2}$ like in the previous case.

- If $\varphi = \dot\forall_i\, \psi$, we have $\varphi^\star_\pi = \dot\forall\, \psi^\star_{\uparrow_i \pi}$ and need to show

  $$\left(\forall d : D.\, \mathcal{M}, d \cdot \rho_1 \vDash_1 \psi^\star_{\uparrow_i \pi_i}\right) \leftrightarrow (\forall d : D^\diamond.\, \mathcal{M}^\diamond, d \cdot \rho_2 \vDash_2 \psi)$$

  → Let $d : D^\diamond$. Use the inductive hypothesis on the assumption instantiated with d. The preconditions of the inductive hypothesis hold, since the updated functions $\uparrow_i\, \pi_i$ and $\uparrow_i\, \pi_p$ match the updated environments $d \cdot \rho_1$ and $d \cdot \rho_2$.

  ← Let $d : D$. Conclude again using the inductive hypothesis.

- If $\varphi = \dot\forall^n_p\, \psi$, we have $\varphi^\star_\pi = \dot\forall\, \psi^\star_{\uparrow^n_p \pi}$ and need to show

  $$\left(\forall d : D.\, \mathcal{M}, d \cdot \rho_1 \vDash_1 \psi^\star_{\uparrow^n_p \pi}\right) \leftrightarrow (\forall P : (D^\diamond)^n \to \mathbb{P}.\, \mathbb{U}^\diamond_n\, P \to \mathcal{M}^\diamond, P \cdot \rho_2 \vDash_2 \psi)$$

  → Let $P : (D^\diamond)^n \to \mathbb{P}$ with $\mathbb{U}^\diamond_n\, P$. This means there is a $d : D$ whose predicate application behaves the same as P, i.e. $P\, \boldsymbol{v} \leftrightarrow \mathsf{App}^{\mathcal{I}}_n\, (d :: \boldsymbol{v})$ for all $\boldsymbol{v}$. Instantiate the assumption with d and conclude with the inductive hypothesis.

  ← Let $d : D$. Instantiate the assumption with $P := \lambda \boldsymbol{v}.\, \mathsf{App}^{\mathcal{I}}_n\, (d :: \boldsymbol{v})$. P is trivially contained in $\mathbb{U}^\diamond_n$ so that we can finish using the inductive hypothesis.

The remaining cases are straightforward or follow in a similar way. □

8.3 Final Semantic Reduction

Now we can translate Henkin models into first-order ones and vice versa. However, there is still the problem that the second translation only works for first-order models that have comprehension. To enforce this, we can define a theory that encodes this property and then only consider first-order models that satisfy this theory.

Definition 8.13 *Let $\mathcal{C}$ be the second-order theory that contains the formulas $\dot{\forall}\,\mathsf{Compr}^n_\varphi$ for all n and φ, where the $\dot{\forall}$ operation adds enough universal quantifiers such that the resulting formula is closed.*

Fact 8.14 *$\mathcal{C}$ is enumerable if Σ is enumerable.*

Proof Follows from the enumerability of formulas and numbers. □

Notice that $\mathcal{C}$ is a second-order theory. But if we translate it into a first-order one, we get exactly the property we want:

Lemma 8.15 *Let $\mathcal{M}$ by a first-order model and ρ_1 a first-order environment with $\mathcal{M}, \rho \vDash_1 \mathcal{C}^\star$. Then $\mathcal{M}^\diamond \vDash_2 \mathsf{Compr}^n_\varphi$ for all n and φ.*

Proof We get $\mathcal{M}^\diamond, \rho_1^\diamond \vDash_2 \dot{\forall}\mathsf{Compr}^n_\varphi$ by Lemma 8.12. Given an arbitrary Henkin environment ρ_2, we can instantiate the quantifiers with the first values from ρ_2 and replace $\rho_1^\diamond$ with the remaining part of ρ_2, yielding $\mathcal{M}^\diamond, \rho_2 \vDash_2 \mathsf{Compr}^n_\varphi$. □

This leads us to our final semantic reduction:

Theorem 8.16 (Semantic Reduction) *For all second-order theories $\mathcal{T}$ and second-order formulas φ it holds that*

$$\mathcal{T} \vDash_2 \varphi \;\leftrightarrow\; (\mathcal{T}, \mathcal{C})^\star \vDash_1 \varphi^\star.$$

Proof We show both directions separately:

- $\rightarrow$ Let $\mathcal{M}$ be a first-order model and ρ a first-order environment with $\mathcal{M}, \rho \vDash_1 (\mathcal{T}, \mathcal{C})^\star$. We need to prove $\mathcal{M}, \rho \vDash_1 \varphi^\star$. By Lemma 8.12 it suffices to show $\mathcal{M}^\diamond, \rho^\diamond \vDash_2 \varphi$. Since $\rho^\diamond$ is a Henkin environment according to Lemma 8.11 and $\mathcal{M}^\diamond$ satisfies comprehension thanks to Lemma 8.15, we can use the assumption and only need to prove $\mathcal{M}^\diamond, \rho^\diamond \vDash_2 \mathcal{T}$. By Lemma 8.12 it suffices to show $\mathcal{M}, \rho \vDash_1 \mathcal{T}^\star$ which holds by assumption.
- $\leftarrow$ Let $\mathcal{H}$ be a Henkin model and ρ a Henkin environment with $\mathcal{H}, \rho \vDash_2 \mathcal{T}$. We need to prove $\mathcal{H}, \rho \vDash \varphi$. $\mathcal{H}$ is not empty, since ρ_i contains values from the domain. By Lemma 8.6 it therefore suffices to show $\mathcal{H}^\star, \rho^\star \vDash_1 \varphi^\star$. Now we can use the assumption and only need to prove $\mathcal{H}^\star, \rho^\star \vDash_1 (\mathcal{T}, \mathcal{C})^\diamond$. By Lemma 8.6 it again suffices to show $\mathcal{H}, \rho \vDash_2 \mathcal{T}, \mathcal{C}$. Finally, $\mathcal{H}, \rho \vDash_2 \mathcal{T}$ holds by assumption

and $\mathcal{H}, \rho \vDash_2 \mathcal{C}$ holds because Henkin models satisfy comprehension in any environment. □

Interestingly, if we were only interested in refuting the result from Chapter 5 for Henkin semantics, that is to show that there exists a deduction system that is sound, complete and enumerable, we would be finished at this point. Consider the following deduction system:

Definition 8.17 *We define* $\mathcal{T} \vdash_2' \varphi := (\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$.

Fact 8.18 $\vdash_2'$ *is sound, complete and enumerable under* LEM*.*

Proof Follows from soundness, completeness and enumerability of $\vdash_1$ with Theorem 8.16. □

However, although fulfilling the requirements put forth in Chapter 5, $\vdash_2'$ can hardly be considered a feasible deduction system. The main goal of is still to show that the natural deduction system $\vdash_2$ is complete. To achieve this using the reduction we need to at least also show $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star \to \mathcal{T} \vdash_2 \varphi$. This will be addressed in the next chapter.

Chapter 9

Henkin Completeness: Deductive Reduction Property

After showing that second-order logic semantically reduces to first-order logic in Chapter 8, we want to obtain the same property for the deduction system, that is

$$\mathcal{T} \vdash_2 \varphi \;\leftrightarrow\; (\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star.$$

We begin with a brief overview of the proof strategy.

9.1 Overview

The backwards direction of the deductive reduction is the hardest part of the proof and will require multiple intermediate steps. The forwards direction is easier and again not strictly necessary to get completeness. While Nour and Raffalli show $\mathcal{T} \vdash_2 \varphi \to (\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$ via an induction on the second-order derivation, it is also possible to obtain this direction by soundness of $\vdash_2$ and completeness of $\vdash_1$:

Lemma 9.1 $\mathcal{T} \vdash_2 \varphi \to (\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$ *under* LEM.

Proof Assume $\mathcal{T} \vdash_2 \varphi$. By soundness, we have $\mathcal{T} \vDash_2 \varphi$ which is equivalent to $(\mathcal{T}, \mathcal{C})^\star \vDash_1 \varphi^\star$ by Theorem 8.16. Completeness of $\vdash_1$ (Theorem 7.1) finally gives us $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$ under LEM. □

Proving the converse direction $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star \to \mathcal{T} \vdash_2 \varphi$ is more difficult. This is also the point where the main idea of Nour and Raffalli comes in. They define a backwards translation function $\_^\diamond : \mathfrak{F}_1(\Sigma_+) \to \mathfrak{F}_2(\Sigma)$ that turns arbitrary first-order formulas in the extended signature back into second-order formulas of the original signature. Then, they prove the following results:

1. $\varphi^\diamond$ turns first-order derivations into second-order ones ($A \vdash_1 \varphi \to A^\diamond \vdash_2 \varphi^\diamond$).

2. $\varphi^{\star\diamond}$ is logically equivalent to φ ($\vdash_2 \varphi^{\star\diamond} \dot\leftrightarrow \varphi$).

Using (1) they can turn $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$ into $(\mathcal{T}, \mathcal{C})^{\star\diamond} \vdash_1 \varphi^{\star\diamond}$ which is equivalent to $\mathcal{T}, \mathcal{C} \vdash_2 \varphi$ according to (2). As $\vdash_2$ proves comprehension, they finally get $\mathcal{T} \vdash_2 \varphi$. This way they get around needing to do a direct induction on the initial derivation $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$, which might only work in a cut-free deduction system. Note that their approach works, because their translation function $\_^\star$ keeps the structure of the formulas unchanged, allowing for $\_^\diamond$ to be defined in a streamlined way. This is unlike the folklore translation presented in Example 7.4, where the isIndi and isPred symbols would be difficult to handle.

We will first define the backwards translation function in Section 9.2 and then show a property similar to (1) in Section 9.3. After an intermediate step in Section 9.4, we show property (2) in Section 9.5 and finish the deductive reduction.

9.2 Backwards Translation Function to Second-Order Logic

The difficulty in defining the backwards translation is that variables in our first-order formulas represent individuals and predicates of all arities at the same time. Therefore, a variable could technically be used for all of those things at once in the same formula. Consider for example the following formula:

$$\varphi := \dot\forall x.\, \mathsf{App}_0(x) \mathbin{\dot\land} \mathsf{App}_1(x, x)$$

Here, x is used as an individual as well as a nullary and unary predicate. Of course, our forwards translation would never produce such formulas, but the backwards translation needs to handle it nonetheless.

The intuitive idea to solve this, is to split up the different usages of x into an individual variable x and predicate variables X_0 and X_1 that each get bound by their own second-order quantifier. Then we can replace the App symbols with the actual second-order atomic formulas:

$$\varphi^\diamond := \dot\forall X_1 X_0.\, \dot\forall x.\, X_0 \mathbin{\dot\land} X_1(x)$$

This always works, because there are only finitely many variables occurring in a formula, so we can always add enough quantifiers such that all different usages of them are bound.

Similarly to the forwards translation, we also need to think about how to implement $\_^\diamond$ in our de Bruijn encoding. Luckily, the usage of de Bruijn indices makes the backwards translation actually easier to define. In our example formula φ, x corresponds to the de Bruijn index 0. Since the translated variables x, X_0, and X_1 all live in their own scope and count de Bruijn indices independently of each other, they all still correspond to the index 0. Therefore, x_i can stay x_i and $\mathsf{App}_n\,(x_i :: \boldsymbol{v})$ is turned into $p^n_i\, \boldsymbol{v}$. This also means that the backwards translation on terms is the identity function.

Remark 9.2 (Free Variables) *A concern one might have at this point is that free variables, especially the ones translated by _*, might be treated incorrectly by this approach (see Example 9.23). However, we will address this problem later on and present a solution to extend the result to free variables (Section 9.5). Therefore we can pretend for now that all of the formulas are closed.*

So in general, in order to define $(\dot\forall\varphi)^\diamond$ for some formula φ, we first calculate $\varphi^\diamond$ and then need to add enough quantifiers such that all usages of the de Bruijn index 0 are covered. An observation that eases the mechanization is that we can always add the individual quantifier $\dot\forall$, even if the individual variable 0 is not used in $\varphi^\diamond$, because unused quantified variables do not change the meaning of the formula. Similarly, we can add arbitrarily many predicate quantifiers of increasing arity as long as the ones that are actually used are present. Over-approximating this number of quantifiers makes the mechanization simpler since we do not need to bother with precisely finding out for which arities the index 0 is unbound in $\varphi^\diamond$. Instead, we find a bound on the arities of predicate variables that occur anywhere in the formula and simply add quantifiers for each arity below this bound:

Definition 9.3 (Arity Bounds) *We say* b *is a bound on the arities occuring in* φ*, if all predicate variables in* φ *have a smaller arity than* b*. We write this as* $\sup|\varphi| \leqslant b$ *which is defined by* $\sup|p_i^n\,\boldsymbol{v}| \leqslant b := n < b$ *and the remaining cases being obvious.*

We write $\sup|\varphi|$ *for the lowest arity bound on phi. It is defined by* $\sup|p_i^n\,\boldsymbol{v}| := Sb$ *and* $\sup|\mathcal{P}\,\boldsymbol{v}| := 0$*. The remaining cases are defined in the obvious way.*

Fact 9.4 $\sup|\varphi|$ *is correct, i.e. it computes an arity bound for all* φ*.*

Proof By induction on φ. □

Definition 9.5 (Predicate Closing Operation) *We define* $\dot\forall_p^{<n}\,\varphi$ *recursively by*

$$\dot\forall_p^{<0}\,\varphi := \varphi \qquad\qquad \dot\forall_p^{<Sn}\,\varphi := \dot\forall_p^{n}\,\dot\forall_p^{<n}\,\varphi.$$

We also define $\dot\exists_p^{<n}\,\varphi$ *in the same way and write* $\dot\nabla_p^{<n}$ *as a placeholder for both cases.*

Now, we can formally define $(\forall\varphi)^\diamond := \dot\forall_p^{<\sup|\varphi^\diamond|}\,\dot\forall\,\varphi^\diamond$.

Remark 9.6 (Function Quantifiers) *We want to illustrate why the approach of Nour and Raffalli would not directly work for function quantifiers. If we look the at the formula*

$$\psi := \dot\forall x.\,\mathsf{funcApp}_1(x,x) \equiv y$$

we could similarly translate it to

$$\psi^\diamond := \dot\forall\,F_1.\,\dot\forall\,x.\,F_1(x) \equiv y.$$

But in a derivation it could happen that y *is substituted with* $\mathsf{funcApp}_2(x, x, x)$. *In the original formula, this would result in* $\forall x.\, \mathsf{funcApp}_1(x, x) \equiv \mathsf{funcApp}_2(x, x, x)$, *in which all occurrences of* x *are still bound to the single first-order quantifier at the beginning. However, if we correspondingly substituted* y *with* $F_2(x, x, x)$ *in* $\psi^\diamond$, *we would have that* F_2 *is unbound. So the problem with function quantifiers is that substitutions can introduce applications of function variables with arities that did not occur in the formula before and therefore are not bound, which changes the meaning of the statement.*

This problem could potentially be fixed by finding a bound on the arity of substitutions occurring in the derivation $A \vdash_1 \varphi$ *we are interested in and adding quantifiers until all of them are covered. But this approach would be very tedious to mechanize as it introduces a dependency between the derivation and the translation function. Therefore, we only consider* $\mathfrak{F}_2$ *for the translation.*

The only non-trivial case of the backwards translation still missing is what happens if the first argument of App is not a variable, but an application $\mathcal{F}\,\boldsymbol{v}$ of some function symbol $\mathcal{F} : \mathcal{F}_\Sigma$. This case is generally problematic, as functions only return individuals and their output should not be interpreted as a predicate. Nour and Raffalli treat this as an error case and simply translate it into the formula $\dot\perp$. For reasons discussed later (see Remark 9.17), we slightly diverge from their approach and add a special predicate symbol $\dot\perp_n$ to the signature that represents falsity. We then define $(\mathsf{App}_n(t :: \boldsymbol{v}))^\diamond := \dot\perp_n\, \boldsymbol{v}^\diamond$, if t is not a variable. This also means that the backwards translation does not end up in the initial signature, but the initial signature extended with those falsity symbols:

Definition 9.7 *We obtain the extended signature* $\Sigma_\perp$ *by adding* n*-ary predicate symbols* $\dot\perp_n$ *for each* $n : \mathbb{N}$.

In Section 9.4 we present a technique to go from $\Sigma_\perp$ back to Σ.

Remark 9.8 (Choice of Error Representation) *One might wonder why we bother with encoding the error cases in the formula itself. The usual approach would be to instead use an option type for the backwards translation function and handle errors this way. However, consider that erroneous formulas can appear in a proof* $\vdash_1 \varphi$, *even if* φ *itself is valid. This would make it impossible to prove property (1) with an ordinary induction on the derivation. One possible solution might be to obtain a cut-free derivation, but we choose to stay with the approach by Nour and Raffalli, such that* $\varphi^\diamond$ *is defined on all formulas* φ.

In summary, we define the backwards translation as follows:

Definition 9.9 (Backwards Translation Function) *The backwards translation func-*

tion $\_^\diamond : \mathfrak{F}_1(\Sigma_+) \to \mathfrak{F}_2(\Sigma_\perp)$ *is recursively defined by*

$$\begin{aligned} \perp^\diamond &:= \dot{\perp} & (\mathcal{P}\boldsymbol{v})^\diamond &:= \mathcal{P}\boldsymbol{v} \\ (\mathsf{App}_n\,(x_i :: \boldsymbol{v}))^\diamond &:= p_i^n\,\boldsymbol{v} & (\varphi\,\dot{\Box}\,\psi)^\diamond &:= \varphi^\diamond\,\dot{\Box}\,\psi^\diamond \\ (\mathsf{App}_n\,(\mathcal{F}\boldsymbol{v} :: \_))^\diamond &:= \dot{\perp}_n\,\boldsymbol{v} & (\dot{\nabla}\varphi)^\diamond &:= \dot{\nabla}_p^{<\,\mathrm{sup}\,|\varphi|^\diamond}\,\dot{\nabla}\,\varphi^\diamond \end{aligned}$$

9.3 Translating First-Order Derivations

In this section we want to show that our backwards translation function allows us to turn first-order derivations into second-order ones, that is

$$A \vdash_1 \varphi \to A^\diamond \vdash_2^\perp \varphi^\diamond.$$

We write $\vdash_2^\perp$ to highlight the fact that the backwards translation extends the signature with falsity symbols that can also occur in the derivation.[1]

The proof will be via an induction on the first-order derivation $A \vdash_1 \varphi$. While most cases are trivial, the challenge lies in the quantifier rules. Since the proof is fairly technical, we first want to give some intuition on how the quantifier cases are handled:

Example 9.10 (All Elimination) *Consider the first-order formula* $\dot\forall x.\,\varphi$ *with*

$$\varphi := \mathsf{App}_1(x, y)\,\dot\wedge\,\mathsf{App}_2(x, y, y)\,\dot\wedge\,\mathsf{P}\,(x).$$

The translation $(\dot\forall x.\,\varphi)^\diamond$ *is given by*

$$(\dot\forall x.\,\varphi)^\diamond = \dot\forall X_2 X_1 X_0.\,\dot\forall x.\,X_1(y)\,\dot\wedge\,X_2(y, y)\,\dot\wedge\,\mathsf{P}(x).$$

Now, suppose a first-order derivation instantiates x *with a term* t*, resulting in*

$$\varphi[t] = \mathsf{App}_1(t, y) \wedge \mathsf{App}_2(t, y, y) \wedge \mathsf{P}\,(t).$$

To simulate this in a second-order derivation, we would need to instantiate all variables x *got turned into by the translation, namely* X_2*,* X_1*,* X_0*, and* x*, resulting in a substitution of shape* $(\varphi^\diamond)[?]_p^2\,[?]_p^1\,[?]_p^0\,[?]$*. The key insight is, that we want those substitutions to fulfill*

$$(\varphi^\diamond)[?]_p^2\,[?]_p^1\,[?]_p^0\,[?] \;=\; (\varphi[t])^\diamond$$

such that the inductive hypothesis can be used. Recall, that the formula $(\varphi[t])^\diamond$ *evaluates to depends on the shape of* t*:*

$$\begin{aligned} (\varphi[t])^\diamond &= Z_1(y)\,\dot\wedge\,Z_2(y, y)\,\dot\wedge\,\mathsf{P}\,(t^\diamond) && \text{if } t \text{ is a variable } z \\ (\varphi[t])^\diamond &= \dot\perp_1(y)\,\dot\wedge\,\dot\perp_2(y, y)\,\dot\wedge\,\mathsf{P}\,(t^\diamond) && \text{if } t \text{ is not a variable} \end{aligned}$$

Therefore, we should substitute X_i *with* Z_i *if* t *is a variable z, and with* $\dot\perp_i$ *otherwise. The individual* x *should be replaced with* $t^\diamond$*. Performing those instantiations allows us to deduce* $A^\diamond \vdash_2^\perp (\varphi[t])^\diamond$ *from* $A^\diamond \vdash_2^\perp (\dot\forall x.\,\varphi)^\diamond$*.*

[1] In Section 9.4 we develop a technique to remove those symbols.

Next, we want to transport this intuition from the example to our formal de Bruijn encoding. We again start with a first-order formula $\dot\forall\varphi$ where the quantifier will be instantiated with a term t. Translating $\dot\forall\varphi$ will split up the quantifier into many different second-order quantifiers:

$$(\dot\forall\varphi)^\diamond = \dot\forall_p^{<n}\,\dot\forall\,\varphi^\diamond = \dot\forall_p^{n-1}\dots\dot\forall_p^{0}\,\dot\forall\,\varphi^\diamond \qquad \text{where } n = \sup|\varphi|$$

To simulate the instantiation with t, we again apply the all elimination rule multiple times resulting in substitutions of shape

$$(\varphi^\diamond)[?]_p^{n-1}\dots[?]_p^{0}\,[?]$$

To allow for more compact notation, we define an operation $\varphi[\sigma]_p^{<n}$ to combine those substitutions:

Definition 9.11 (Combining Substitutions) *We write $\varphi[\sigma]_p^{<n}$ for a predicate substitution $\sigma : \mathbb{N} \to \forall n.\,\mathfrak{P}_n$ that is simultaneously applied to all variables with arities smaller than n. Similarly, $\varphi[\sigma]_p^{\geqslant n}$ describes a substitution that is applied to all variables with arities greater or equal to n. Finally, we write $\varphi[\sigma]_p$ for a substitution that affects variables of all arities.*

This allows us to write $(\varphi^\diamond)[?]_p^{<n}[?]$. To fill in the blanks, we can use the intuition from the example and look at whether t is a variable or not. To make this more general, we define the backwards translation of first-order substitutions into second-order ones:

Definition 9.12 (Backwards Translation of Substitutions) *We turn first-order substitutions $\sigma : \mathbb{N} \to \mathfrak{T}$ into a second-order individual substitutions $\sigma_i^\diamond : \mathbb{N} \to \mathfrak{T}$ and predicate substitutions $\sigma_p^\diamond : \mathbb{N} \to \forall n.\,\mathfrak{P}_n$ with*

$$\sigma_i^\diamond\,x := (\sigma\,x)^\diamond \qquad\qquad \sigma_p^\diamond\,x\,n := \begin{cases} p_i^n & \textit{if } \sigma\,x = x_i \\ \dot\bot_n & \textit{otherwise} \end{cases}$$

Notice how $\sigma_p^\diamond$ effects predicates of all arities. Using this, we can turn the first-order substitution of t into the second-order substitutions we were looking for:

$$(\varphi^\diamond)[(t)_p^\diamond]_p^{<n}\,[(t)_i^\diamond]$$

Finally, recall that n is an arity bound of φ. This means that φ does not contain any predicate variables with arities greater or equal to n. Therefore, the bounded substitution $[\cdot]_p^{<n}$ effects all variables anyway, so we can just as well drop the bound:

Fact 9.13 *Let b be an arity bound of a formula φ. It holds that*

1. $\varphi[\sigma]_p^{<b} = \varphi[\sigma]_p$.
2. $\varphi[\sigma]_p^{\geqslant b} = \varphi$.

Proof It suffices to prove $\forall \sigma \tau. (\forall x n. n < b \to \sigma\, x\, n = \tau\, x\, n) \to \varphi[\sigma]_p = \varphi[\tau]_p$ by induction on φ. □

Thus, we have $(\varphi^\diamond)[(t)_p^\diamond]_p^{<n}[(t)_i^\diamond] = (\varphi^\diamond)[(t)_p^\diamond]_p[(t)_i^\diamond]$. The main task now is to verify that this really simulates the substitution of t, in other words, we have to show

$$(\varphi[t])^\diamond = (\varphi^\diamond)[(t)_p^\diamond]_p[(t)_i^\diamond],$$

as this allows us to use the inductive hypothesis. To achieve this, we will prove the more general lemma $\forall \sigma. (\varphi[\sigma])^\diamond = (\varphi^\diamond)[\sigma_p^\diamond]_p[\sigma_i^\diamond]$, stating that we can move arbitrary substitutions out of the backwards translation by splitting it up into predicate and individual substitutions.

But first, we need some additional lemmas to characterize the behaviour of the predicate closing operation under substitutions:

Fact 9.14 (Substitution of Closing Operation) *Writing $p_0 \cdot \uparrow\sigma$ as notation for the substitution $\lambda x n. (p_0^n \cdot \uparrow^n \sigma)\, x\, n$ it holds that:*

1. $(\dot\forall_p^{<n}\, \varphi)[\sigma] = \dot\forall_p^{<n}\, \varphi[\sigma]$
2. $(\dot\forall_p^{<n}\, \varphi)[\sigma]_p = \dot\forall_p^{<n}\, \varphi[p_0 \cdot \uparrow\sigma]_p^{<n}[\sigma]_p^{\geqslant n}$
3. $\sup|\varphi| \leqslant b \to (\dot\forall_p^{<b}\, \varphi)[\sigma]_p = \dot\forall_p^{<b}\, \varphi[p_0 \cdot \uparrow\sigma]_p$

The same properties also hold for $\dot\exists_p^{<n}$.

Proof We show (1) and (2) by induction on n with σ generalized. (3) follows from (2) with Fact 9.13. □

Fact 9.15 (Arity Bounds of Substituted Formulas)

1. $\sup|\varphi[\sigma]| = \sup|\varphi|$
2. $\sup|\varphi[\sigma]_p| = \sup|\varphi|$

Proof By induction on φ with σ generalized. □

Now, we can prove our substitution lemma:

Lemma 9.16 *We can move first-order substitutions out of the backwards translation function by splitting them up into an individual and a predicate substitution:*

$$(\varphi[\sigma])^\diamond = (\varphi^\diamond)[\sigma_p^\diamond]_p[\sigma_i^\diamond]$$

Proof By induction on φ with σ generalized. We only discuss the interesting cases:

- If $\varphi = \mathsf{App}_n\,(t :: \boldsymbol{v})$, the translation depends on whether t is a variable:
  - If t is not a variable, the claim reduces to $\dot\bot_n\,\boldsymbol{v} = \dot\bot_n\,\boldsymbol{v}$.
  - If $t = x_i$ then we have to show $(\mathsf{App}_n(\sigma\,i :: \boldsymbol{v}[\sigma]))^\diamond = (\sigma_p^\diamond\,i\,n)\,\boldsymbol{v}$ which again depends on whether $\sigma\,i$ is a variable:
    - If $\sigma\,i$ is not a variable, we have $(\mathsf{App}_n(\sigma\,i :: \boldsymbol{v}[\sigma]))^\diamond = \dot\bot_n\,\boldsymbol{v}$, but also $\sigma_p^\diamond\,i\,n = \dot\bot_n$.
    - If $\sigma\,i = x_j$, we have $(\mathsf{App}_n(\sigma\,i :: \boldsymbol{v}[\sigma]))^\diamond = p_j^n\,\boldsymbol{v}$, but also $\sigma_p^\diamond\,i\,n = p_j^n$.

    We can see that the substitution $\sigma_p^\diamond$ matches the behaviour of $\_^\diamond$ with regards to the falsity predicate.
- If $\varphi = \dot\forall\varphi$, we have to show

$$\dot\forall_p^{<\sup|\varphi[x_0\,\cdot\uparrow\sigma]^\diamond|}\,\dot\forall\,\varphi[x_0\,\cdot\uparrow\sigma]^\diamond = \left(\dot\forall_p^{<\sup|\varphi^\diamond|}\,\dot\forall\,\varphi^\diamond\right)[\sigma_p^\diamond]_p\,[\sigma_i^\diamond].$$

  By Fact 9.14 we have

$$\left(\dot\forall_p^{<\sup|\varphi^\diamond|}\,\dot\forall\,\varphi^\diamond\right)[\sigma_p^\diamond]_p\,[\sigma_i^\diamond] = \dot\forall_p^{<\sup|\varphi^\diamond|}\,\dot\forall\,\varphi^\diamond[p_0\,\cdot\uparrow\sigma_p^\diamond]_p\,[x_0\,\cdot\uparrow\sigma_i^\diamond].$$

  Also, $\sup|\varphi[x_0\,\cdot\uparrow\sigma]^\diamond| = \sup|\varphi^\diamond|$ by the inductive hypothesis and Fact 9.15. It only remains to show $\varphi[x_0\,\cdot\uparrow\sigma]^\diamond = \varphi^\diamond[p_0\,\cdot\uparrow\sigma_p^\diamond]_p\,[x_0\,\cdot\uparrow\sigma_i^\diamond]$. By the inductive hypothesis we have $\varphi[x_0\,\cdot\uparrow\sigma]^\diamond = \varphi^\diamond[(x_0\,\cdot\uparrow\sigma)_p^\diamond]_p\,[(x_0\,\cdot\uparrow\sigma)_i^\diamond]$ which extensionally behaves the same as the substitutions $p_0\,\cdot\uparrow\sigma_p^\diamond$ and $x_0\,\cdot\uparrow\sigma_i^\diamond$. □

Remark 9.17 (Necessity of the Falsity Predicates) *The* App *case in this proof illustrates why we do not use Nour and Raffalli's approach of replacing invalid formulas with* $\dot\bot$. *Imagine that we have a formula* $\varphi := \mathsf{App}_n\,(x_i :: \boldsymbol{v})$. *If we applied a substitution* σ *to* φ *that replaces* x_i *with a term that is not a variable, we would trigger the error case upon translation. Nour and Raffalli would then set* $\varphi[\sigma]^\diamond = \dot\bot$. *However, if we first translated* φ *to* $\varphi^\diamond = p_i^n\,\boldsymbol{v}$ *we would have no chance to turn this into* $\dot\bot$ *through a second-order substitution: No matter what we substituted afterwards, we would still end up with an atomic formula. Therefore, Lemma 9.16 can only be proven if we have an error value that can get introduced by a substitution.*

Nour and Raffalli solve this by allowing substitutions that replace whole atomic formulas. They would write this as $\varphi^\diamond[x := \lambda\boldsymbol{v}.\,\dot\bot] = \dot\bot$. *But since substitutions like this are not available in our setting, this would not work directly. On the other hand, it is probably possible to simulate those substitutions as*

$$\psi[x := \lambda\boldsymbol{v}.\,P\,\boldsymbol{v}] \;\sim\; \dot\exists P'.\,(\dot\forall x_1 ... x_n.\,P'(x_1, ..., x_n) \mathrel{\dot\leftrightarrow} P(x_1, ..., x_n)) \mathrel{\dot\land} \psi[x := P']$$

and obtain P' through the comprehension rule. This is essentially an inlined version of the argument we will use in Section 9.4 to remove the falsity symbols from derivations. The benefit of splitting it up the way we did, is that it allows us to prove Lemma 9.16 in this general way and slightly reduce the complexity of the main proof.

Next, we want to give some intuition on the treatment of the introduction of universal quantifiers:

Example 9.18 (All Introduction) *Consider again the first-order formula $\dot\forall\varphi$ with*

$$\varphi = \mathsf{App}_1(x, y) \mathbin{\dot\wedge} \mathsf{App}_1(x, y, y) \mathbin{\dot\wedge} P(x).$$

Recall that the translation $(\dot\forall\varphi)^\diamond$ of is given by

$$(\dot\forall\varphi)^\diamond = \dot\forall X_2 X_1 X_0. \dot\forall x. X_1(y) \mathbin{\dot\wedge} X_2(y, y) \mathbin{\dot\wedge} P(x).$$

While the first-order proof of φ introduces the single variable x, the second-order proof of $\varphi^\diamond$ needs to apply the introduction rule multiple times to introduce all quantified variables x got turned into. The remaining obligation $X_1(y) \mathbin{\dot\wedge} X_2(y, y) \mathbin{\dot\wedge} P(x)$ should be handled by the inductive hypothesis.

While this seems straightforward, performing the same argument in our de Bruijn encoding is more tricky. There, introducing variables leads to shifts in the context, which can be problematic. For example, after introducing the quantifiers, we would be left with

$$(A^\diamond)[\uparrow]_p^{<3}\,[\uparrow] \vdash_2^\perp \varphi^\diamond$$

However, the inductive hypothesis would be

$$(A[\uparrow])^\diamond \vdash_2^\perp \varphi^\diamond.$$

Importantly, we have $(A^\diamond)[\uparrow]_p^{<3}\,[\uparrow] \neq (A[\uparrow])^\diamond$ since A can contain predicates with arities greater than 2. One possible solution would be to apply the substitution $[\uparrow]_p^{\geqslant 3}$ using WEAKS$_p$ *which would not effect $\varphi^\diamond$, but make $(A^\diamond)[\uparrow]_p^{<3}\,[\uparrow]\,[\uparrow]_p^{\geqslant 3} = (\uparrow A)^\diamond$.*

Instead, we chose to use the named version of all introduction from Lemma 3.21 that does not introduce shifts. We substitute the formula with a fresh index i and leave the context untouched:

$$A \vdash_2^\perp (\varphi^\diamond)[p_i]_p^{<3}\,[x_i]$$

We can then finish the proof by bringing the inductive hypothesis in a similar shape.

The following fact shows how the predicate closing operation can be proven and what can be derived from it. As discussed in the example, we use named variants of all introduction and existential elimination:

Fact 9.19 (Derivations with Closing Operation) *The following rules are admissible for* $\vdash_2$*:*

$$\textsc{AllI}^{\star}_{p}\ \frac{A \vdash \varphi[p_i]_p \qquad p_i \notin A, \varphi \qquad \sup|\varphi| \leqslant b}{A \vdash \dot{\forall}^{<b}_{p} \varphi}$$

$$\textsc{AllE}^{\star}_{p}\ \frac{A \vdash_2 \dot{\forall}^{<b}_{p} \qquad \sup|\varphi| \leqslant b}{A \vdash_2 \varphi[P]_p}$$

$$\textsc{ExI}^{\star}_{p}\ \frac{A \vdash_2 \varphi[P]_p \to A \qquad \sup|\varphi| \leqslant b}{A \vdash_2 \dot{\exists}^{<b}_{p} \varphi}$$

$$\textsc{ExE}^{\star}_{p}\ \frac{A \vdash_2 \dot{\exists}^{<b}_{p} \varphi \qquad \varphi[p_i]_p :: A \vdash_2 \psi \qquad p_i \notin A, \varphi, \psi \qquad \sup|\varphi| \leqslant b}{A \vdash_2 \psi}$$

where $P : \forall n.\, \mathfrak{P}_n$ *is a family of predicates for each arity and* p_i *inidcates the predicate family* $\lambda n.\, p^n_i$.

Proof We discuss the different rules separately.

1. Since b is an arity bound it suffices to show $A \vdash_2 \varphi[p_i]^{<b}_p \to A \vdash_2 \dot{\forall}^{<b}_p \varphi$ according to Fact 9.13. We perform an induction on b with φ generalized. If $b = 0$, we have $\varphi[p_i]^{<0}_p = \varphi$. If $b = Sn$, we use the ($\textsc{AllI}'_p$) rule and have to show $A \vdash_2 (\dot{\forall}^{<n}_p \varphi)[p_i]_p$. Follows by Fact 9.14 and the inductive hypothesis.
2. Since b is an arity bound, we can assume $A \vdash_2 \varphi[P]^{<b}_p \to A \vdash_2 \dot{\forall}^{<b}_p \varphi$ according to Fact 9.13. We perform an induction on b with φ generalized. If $b = 0$, we have $\varphi[P]^{<0}_p = \varphi$. If $b = Sn$, we use the ($\textsc{AllE}_p$) rule on the assumption to obtain $A \vdash_2 (\dot{\forall}^{<n}_p \varphi)[P]_p$. Then $A \vdash_2 \varphi[P]^{<Sn}_p$ follows by Fact 9.14 and the inductive hypothesis.
3. Similar to (2).
4. Similar to (1). □

Now, we can finally perform the translation of the first-order derivation:

Theorem 9.20 $A \vdash_1 \varphi \to A^{\diamond} \vdash^{\perp}_2 \varphi^{\diamond}$ *for all* A *and* φ*.*

Proof We perform an induction on the first-order derivation $A \vdash_1 \varphi$. We only discuss the cases of all introduction and elimination. The existential quantifier rules are similar and the remaining cases are trivial.

(AllI) In the case of all introduction we get the inductive hypothesis $(A[\uparrow])^\diamond \vdash_2^\perp \varphi^\diamond$ and have to show $A^\diamond \vdash_2^\perp (\dot\forall\varphi)^\diamond$. We start by removing the shift from the inductive hypothesis. For this we use (WeakS$_p$) and (WeakS) which gives us

$$(A[\uparrow])^\diamond[(x_i)_p^\diamond]_p\,[(x_i)_i^\diamond] \vdash_2^\perp (\varphi^\diamond)[(x_i)_p^\diamond]_p\,[(x_i)_i^\diamond].$$

Using Lemma 9.16 we can simplify this to

$$A^\diamond \vdash_2^\perp (\varphi^\diamond)[(x_i)_p^\diamond]_p\,[(x_i)_i^\diamond].$$

By definition, the goal reduces to $A^\diamond \vdash_2^\perp \dot\forall_p^{<\sup|\varphi|}\dot\forall\,\varphi^\diamond$. Using (AllI$_p^\star$) and (AllI$'$) it then suffices to show $A^\diamond \vdash_2^\perp (\varphi^\diamond)[p_i]_p\,[x_i]$ for an index i that is unused in $A^\diamond$ and $\varphi^\diamond$. Since $(x_i)_p^\diamond$ corresponds to the substitution $[p_i]_p$, and $(x_i)_i^\diamond$ to substituting x_i, we are finished.

(AllE) In the case of all elimination we get the inductive hypothesis $A^\diamond \vdash_2^\perp (\dot\forall\varphi)^\diamond$ and have to show $A^\diamond \vdash_2^\perp (\varphi[t])^\diamond$. If t is a variable x_i, then set $P := \lambda n.\, p_i^n$. Otherwise, set $P := \lambda n.\, \dot\perp_n$. Using (AllE$_p^\star$) with $P := (t)_p^\diamond$ as well as (AllE) on the inductive hypothesis gives us $A^\diamond \vdash_2 (\varphi^\diamond)[(t)_p^\diamond]_p\,[(t)_i^\diamond]$ which is equal to our goal by Lemma 9.16. □

9.4 Removing the Falsity Predicates

Next, we want to turn derivations $\vdash_2^\perp$ in the signature $\Sigma_\perp$ into derivations in the initial signature Σ. The basic idea is to define a function $\_^\perp : \mathfrak{F}_2(\Sigma_\perp) \to \mathfrak{F}_2(\Sigma)$ that replaces each application of a falsity predicate with the formula $\dot\perp$.

Definition 9.21 (Falsity Symbol Removal) *We recursively define the function* $\_^\perp : \mathfrak{F}_2(\Sigma_\perp) \to \mathfrak{F}_2(\Sigma)$ *by*

$$\begin{aligned} \perp^\perp &:= \dot\perp & (\varphi \mathbin{\dot\Box} \psi)^\perp &:= \varphi^\perp \mathbin{\dot\Box} \psi^\perp \\ (\dot\perp_n \boldsymbol{v})^\perp &:= \dot\perp & (\dot\forall\,\varphi)^\perp &:= \dot\forall\,\varphi^\perp \\ (\mathcal{P}\boldsymbol{v})^\perp &:= \mathcal{P}\boldsymbol{v} & (\dot\forall_p^n\,\varphi)^\perp &:= \dot\forall_p^n\,\varphi^\perp \end{aligned}$$

Then, we can turn $A \vdash_2^\perp \varphi$ into a derivation $A^\perp \vdash_2 \varphi^\perp$ by replacing each occurrence of the falsity symbol in the derivation with an application of the (Compr) rule.

Lemma 9.22 (Derivations without Falsity Symbol) *For all* $\varphi : \mathfrak{F}_2(\Sigma_\perp)$ *and contexts* A *it holds that*

$$A \vdash_2^\perp \varphi \to A^\perp \vdash_2 \varphi^\perp.$$

Proof By induction on the derivation $A \vdash_2^\perp \varphi$. We only discuss the case of predicate all elimination. There we get a predicate P of arity n as well as the inductive hypothesis $A^\perp \vdash_2 \dot\forall_p^n\,\varphi^\perp$ and have to show $A^\perp \vdash_2 (\varphi[P]_p^n)^\perp$.

- If P is a variable or a symbol we can show that the goal is equivalent to $A^{\bot} \vdash_2 (\varphi^{\bot})[\mathsf{P}]^n_p$. This follows from the inductive hypothesis using the $(\textsc{AllE}_p)$ rule.
- If $\mathsf{P} = \dot{\bot}_n$, we obtain a variable p^n_f with

$$A^{\bot} \vdash_2 \dot{\forall} x_1, ..., x_n.\, p^n_f\,(x_1, ..., x_n) \dot{\leftrightarrow} \dot{\bot}$$

using the comprehension scheme and the nameless (ExE'_p) rule. We can show $\vdash_2 (\varphi[\mathsf{P}]^n_p)^{\bot} \dot{\leftrightarrow} (\varphi^{\bot})[p^n_f]^n_p$ such that it suffices to prove $A^{\bot} \vdash_2 (\varphi^{\bot})[p^n_f]^n_p$ which again follows from the inductive hypothesis using the $(\textsc{AllE}_p)$ rule.

The remaining cases are proven in a similar way or are straightforward. □

9.5 Combining Forwards and Backwards Translation

In this section, we want to show that $\varphi^{\star\diamond\bot}$ is equivalent to φ. To illustrate how the different functions interact, we first want to look at an example:

Example 9.23 (Forwards and Backwards Translation) *Consider the second-order formula $\varphi : \mathfrak{F}_2(\Sigma)$ given by*

$$\varphi := \dot{\forall} x.\, \mathsf{Q}(x, x) \dot{\rightarrow} \dot{\exists} \mathsf{P}.\, \mathsf{P}(x).$$

Translating this to first-order logic results in a formula $\varphi^{\star} : \mathfrak{F}_1(\Sigma_+)$ that looks like

$$\varphi^{\star} = \dot{\forall} x.\, \mathsf{App}_2(q, x, x) \dot{\rightarrow} \dot{\exists} p.\, \mathsf{App}_1(p, x).$$

Applying the backwards translation gives us the formula $\varphi^{\star\diamond} : \mathfrak{F}_2(\Sigma_{\bot})$ with

$$\varphi^{\star\diamond} = \dot{\forall} X_2 X_1 X_0.\, \dot{\forall} x.\, \mathsf{Q}(x, x) \dot{\rightarrow} \dot{\exists} \mathsf{P}_1 \mathsf{P}_0.\, \dot{\exists} p.\, \mathsf{P}_1(x).$$

A first observation is that the backwards translation did not trigger the error case. In fact, we can convince ourselves that any formula $\varphi^{\star}$ produced by the forwards translation is well-formed, i.e. has variables as the first argument of the App *symbol. Therefore, $\varphi^{\star\diamond\bot}$ will look exactly like $\varphi^{\star\diamond}$ except that it lives in the original signature, in other words $\varphi^{\star\diamond\bot} : \mathfrak{F}_2(\Sigma)$.*

Also notice how the variable x got split up into multiple second-order variables but only the original variable x is bound. Similarly, P got split up, but only the unary predicate variable P_1 is bound, corresponding to the original variable P in φ. We can see that in effect, translating to first-order logic and back only adds a bunch of unused quantifiers. Therefore it should be intuitive that the resulting formula is equivalent to φ.

One might even have hope to show $\varphi^{\star\diamond\bot} = \varphi$ by more carefully defining $\_^{\diamond}$ such that the number of quantifiers are not over-approximated. But besides being very unpleasant to mechanize, Nour and Raffalli claim that this would not work in general. Consider the formula $\dot{\forall} \mathsf{R}_3.\, \varphi$ and notice that the predicate variable R_3 is not used in φ. After turning this into a first-order quantifier $\dot{\forall} r.\, \varphi^{\star}$, the backwards translation has no way of knowing that

r *was a ternary relation variable as it is unused. Thus we cannot end up with the initial formula in general.*

Finally, we need to point out that the translation with named binders above is not entirely accurate. Notice how we translated the unbound variable Q *into the first-order variable* q *and then conveniently back into* Q*. In the actual translation, the situation is different. There,* Q *is represented by some de Bruijn index* i*. The first-order translation uses its* π^0 *functions to replace* i *with* $\pi^0_p\, i\, 2$*. Since the backwards translation preserves the indices, what we end up with no longer represents the same variable. This is the issue hinted at before in Remark 9.2. To solve it, we need to modify the backwards translation such that it turns* $\pi^0_p\, i\, 2$ *back into* i*.*

The key to solving the problems with free variables is the insight that we can simulate the initial choice of π functions with a substitution:

Lemma 9.24 (Simulate π through Substitutions) *For all* φ *and* π *it holds that*

1. $(\varphi^\star_\pi)^\diamond = (\varphi^\star_{\lambda x.x,\pi_p})^\diamond[\lambda i.\, x_{\pi_i\, i}]$
2. $(\varphi^\star_\pi)^\diamond = (\varphi^\star_{\pi_i,\lambda x n.x})^\diamond[\lambda i n.\, p^n_{\pi_p\, i\, n}]_p$

Proof Both properties are shown by induction on φ. We only discuss two cases of (2) since the other ones are proven similarly or are straightforward:

- If $\varphi = p^n_i\, \boldsymbol{v}$, we have to show
$$p^n_{\pi_p\, i\, n}\, \boldsymbol{v}^\star_{\pi_i} = (p^n_i\, \boldsymbol{v}^\star_{\pi_i})[\lambda x n.\, p^n_{\pi_p\, x\, n}]_p.$$
As the substitution replaces p^n_i with $p^n_{\pi_p\, i\, n}$ this holds trivially.

- If $\varphi = \dot\forall\, \psi$, we have to show
$$\dot\forall_p^{<\sup|(\psi^\star_{\uparrow_i \pi})^\diamond|}\, \dot\forall\, (\psi^\star_{\uparrow_i \pi})^\diamond$$
$$||$$
$$\left(\dot\forall_p^{<\sup|(\psi^\star_{\uparrow_i \pi, \uparrow_i(\lambda x n.x)})^\diamond|}\, \dot\forall\, (\psi^\star_{\uparrow_i \pi_i, \uparrow_i(\lambda x n.x)})^\diamond\right) [\lambda x n.\, p^n_{\pi_p\, x\, n}]_p$$
Using Lemma 9.14 we move the substitution inwards, resulting in
$$\dot\forall_p^{<\sup|(\psi^\star_{\uparrow_i \pi_i, \uparrow_i(\lambda x n.x)})^\diamond|}\, \dot\forall\, (\psi^\star_{\uparrow_i \pi_i, \uparrow_i(\lambda x n.x)})^\diamond[p_0\ \cdot \uparrow(\lambda x n.\, p^n_{\pi_p\, x\, n})]_p$$
Since $\sup|(\psi^\star_{\uparrow_i \pi_i, \uparrow_i \pi_p})^\diamond| = \sup|(\psi^\star_{\uparrow_i \pi_i, \uparrow_i(\lambda x n.x)})^\diamond|$, we only need to show
$$(\psi^\star_{\uparrow_i \pi_i, \uparrow_i \pi_p})^\diamond = (\psi^\star_{\uparrow_i \pi_i, \uparrow_i(\lambda x n.x)})^\diamond[p_0\ \cdot \uparrow(\lambda x n.\, p^n_{\pi_p\, x\, n})]_p.$$
Using the inductive hypotheses, we can turn $\uparrow_i\ \pi_p$ and $\uparrow_i\ (\lambda x n.x)$ into substitutions. We can show that the resulting substitutions behave the same on both sides. We refer to the Coq mechanization for further detail. □

The fact that the initial π^0 functions can be represented through a single substitution also means that we can "undo" their effect by substituting the inverse functions $(\pi^0_i)^{-1}$ and $(\pi^0_p)^{-1}$.

Definition 9.25 (Inverse π^0 Functions) *We define* $(\pi^0_i)^{-1}, (\pi^0_p)^{-1} : \mathbb{N} \to \mathbb{N}$ *by*

$$(\pi^0_i)^{-1}\ a := x \quad \textit{for } a = \langle \_, x \rangle \qquad\qquad (\pi^0_p)^{-1}\ a := x \quad \textit{for } a = \langle \_, \langle x, \_ \rangle\rangle$$

Fact 9.26 *The inverse functions are correct:*

1. $\forall x.\, (\pi^0_i)^{-1}\ (\pi^0_i\ x) = x$
2. $\forall x n.\, (\pi^0_p)^{-1}\ (\pi^0_p\ x\, n) = x$

Proof Immediate by the correctness of the natural pairing function. □

We modify the backwards translation function to apply this substitution:

Definition 9.27 (Extended Backwards Translation for Free Variables) *We define* $\_^{\blacklozenge} : \mathfrak{F}_1(\Sigma_+) \to \mathfrak{F}_2(\Sigma_\perp)$ *by*

$$\varphi^{\blacklozenge} := (\varphi^{\diamond})[\lambda x n.\, p^n_{(\pi^0_p)^{-1}\,x}]_p\ [\lambda x.\, x_{(\pi^0_i)^{-1}\,x}].$$

Fact 9.28 $\varphi^{\star\blacklozenge} = (\varphi^{\star}_{\lambda x.x,\lambda x n.x})^{\diamond}$ *for all* φ.

Proof Follows from Lemma 9.24 and Fact 9.25. □

Therefore, it suffices to show that $(\varphi^{\star}_{\lambda x.x,\lambda x n.x})^{\diamond\perp}$ is equivalent to φ:

Lemma 9.29 $\vdash_2 (\varphi^{\star}_{\lambda x.x,\lambda x n.x})^{\diamond\perp} \dot\leftrightarrow \varphi$ *for all* φ.

Proof We prove the stronger claim

$$\forall \pi_i \pi_p.\, (\forall x.\, \pi_i\, x = x) \to (\forall x n.\, \pi_p\, x\, n = x) \to\ \vdash_2 (\varphi^{\star}_{\pi_i,\pi_p})^{\diamond\perp} \leftrightarrow \varphi$$

by induction on φ:

- If $\varphi = \mathcal{P}\,\boldsymbol{v}$, we have to show $\vdash_2 \mathcal{P}\,\boldsymbol{v}^{\star}_{\pi_i} \leftrightarrow \mathcal{P}\,\boldsymbol{v}$. It suffices to prove $\forall \pi.\, (\forall x.\, \pi_i\, x = x) \to t^{\star}_{\pi} = t$ for all t by induction on t.
- If $\varphi = p^n_i\,\boldsymbol{v}$, we have to show $p^n_{\pi_p\, i\, n}\,\boldsymbol{v}^{\star}_{\pi} \leftrightarrow p^n_i\,\boldsymbol{v}$. By assumption we have $p^n_{\pi_p\, i\, n} = p^n_i$ and $\boldsymbol{v}^{\star}_{\pi_i} = \boldsymbol{v}$ like in the previous case.
- If $\varphi = \dot\forall\,\psi$, we have to show

  $$\vdash_2 \left(\dot\forall_p^{<\sup|(\psi^{\star}_{\uparrow_i\pi})^{\diamond}|}\,\dot\forall\,(\psi^{\star}_{\uparrow_i\pi})^{\diamond}\right)^{\perp} \leftrightarrow \dot\forall\,\psi.$$

  We can show that the falsity translation can be moved inwards:

  $$\vdash_2 \left(\dot\forall_p^{<\sup|(\psi^{\star}_{\uparrow_i\pi})^{\diamond}|}\,\dot\forall\,(\psi^{\star}_{\uparrow_i\pi})^{\diamond\perp}\right) \leftrightarrow \dot\forall\,\psi$$

- $\rightarrow$ Recall from the intuition in Example 9.23 that all the quantified predicate variables are unused and only the original individual quantifier is bound. Here, we can also see why this is the case: The forwards translation in the assumption uses $\uparrow_i \pi_p$ which shifts every predicate variable. But this also means that no variable with index 0 can remain. Thus, the predicate quantifiers are unused.

  Therefore, we can instantiate them with an arbitrary value, for example the index 0. This is done using $(\textsc{AllE}^\star_p)$, which gives us $\dot\forall\,(\psi^\star_{\uparrow_i \pi})^{\diamond\perp}[p_0]_p$, which is equal to $\dot\forall\,((\psi^\star_{\uparrow_i \pi})^{\diamond}[p_0]_p)^{\perp}$.

  But this substitution now "undoes" the predicate shifts. Using Lemma 9.24, we can easily show that

  $$(\varphi^\star_{\uparrow_i \pi})^{\diamond}[P]_p = (\varphi^\star_{(\uparrow_i \pi_i),\pi_p})^{\diamond}.$$

  This leaves us with $\dot\forall\,(\psi^\star_{(\uparrow_i \pi_i),\pi_p})^{\diamond\perp}$. Since $\pi_i\, x = x$ for all x implies $\uparrow_i \pi_i\, x = x$, we can use the inductive hypothesis to show that this is equivalent to $\dot\forall\,\psi$.

- $\leftarrow$ We can use $(\textsc{AllI}^\star_p)$ to introduce the predicate quantifiers, which leaves us to prove $\dot\forall\,(\psi^\star_{\uparrow_i \pi})^{\diamond\perp}[p_i]_p$ for a fresh index i. Again, the substitution reverts the shift so that it suffices to prove $\dot\forall\,(\psi^\star_{(\uparrow_i \pi_i),\pi_p})^{\diamond\perp}$, which by the inductive hypothesis is equivalent to $\dot\forall\,\psi$.

The cases for predicate and existential quantifiers are handled similarly The other cases are straightforward. □

Corollary 9.30 $\vdash_2 \varphi^{\star\blacklozenge\perp} \leftrightarrow \varphi$ *for all* φ.

Proof Follows from Fact 9.28 and Lemma 9.29. □

We can also extend the result from section 9.3 to the updated backwards translation:

Corollary 9.31 $A \vdash_1 \varphi \rightarrow A^{\blacklozenge\perp} \vdash_2 \varphi^{\blacklozenge\perp}$ *for all* φ *and* A.

Proof Follows from Lemma 9.22, (WeakS) and Theorem 9.20. □

This finally gives us the backwards direction of our deductive reduction:

Theorem 9.32 (Deductive Reduction) $\mathcal{T} \vdash_2 \varphi \;\leftrightarrow\; (\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$ *for all* $\mathcal{T}$ *and* φ *under* LEM.

Proof $\rightarrow$ This direction has been proven in Lemma 9.1.

$\leftarrow$ Assume $(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star$. By Corollary 9.31 we get $(\mathcal{T}, \mathcal{C})^{\star\blacklozenge\perp} \vdash_1 \varphi^{\star\blacklozenge\perp}$ and obtain $(\mathcal{T}, \mathcal{C}) \vdash_2 \varphi$ by Corollary 9.30. Since $\vdash_2 \mathcal{C}$, we get $\mathcal{T} \vdash_2 \varphi$. □

$$\begin{tikzcd}
\mathcal{T} \vDash_2 \varphi \arrow[dd, leftrightarrow, "\text{Theorem 8.16}"'] & & \mathcal{T} \vdash_2 \varphi \arrow[ll, "\text{Theorem 6.4}"'] \arrow[ddddll, leftrightarrow, "\text{Theorem 9.32 (LEM)}"'] \\
 & & \\
(\mathcal{T}, \mathcal{C})^\star \vDash_1 \varphi^\star \arrow[dd, dashed, "\text{Theorem 7.1 (LEM)}"'] & & \mathcal{T}, \mathcal{C} \vdash_2 \varphi \arrow[uu, "\vdash_2 \text{ proves } \mathcal{C}"'] \\
 & & \\
(\mathcal{T}, \mathcal{C})^\star \vdash_1 \varphi^\star \arrow[r, "\text{Corollary 9.31}"'] & (\mathcal{T}, \mathcal{C})^{\star\blacklozenge} \vdash_2^\perp \varphi^{\star\blacklozenge} \arrow[r, "\text{Lemma 9.22}"'] & (\mathcal{T}, \mathcal{C})^{\star\blacklozenge\perp} \vdash_2 \varphi^{\star\blacklozenge\perp} \arrow[uu, "\text{Corollary 9.30}"']
\end{tikzcd}$$

Figure 9.1: Summary of the reduction results.

9.6 Summary

Figure 9.1 summarizes the results regarding the reduction. While we focused on completeness, it is also possible to interpret the reduction in the context of synthetic computability theory. Since our translation $\_^\star$ is an ordinary function, we can view Theorem 8.16 and Theorem 9.32 as many-one reductions from first- to second-order validity (in Henkin semantics) and provability.

Overall, we believe that the semantic reduction in Chapter 8 is quite elegant and concise, whereas the deductive part in this chapter is fairly complicated and significantly more difficult to mechanize. If one would only be interested in obtaining completeness, it would probably be easier to formalize the usual Henkin-style completeness proof. However, the reduction approach is more powerful as it gives us the deeper insight that second-order logic with Henkin semantics is essentially the same as first-order logic. Additionally, the reduction not only yields completeness, but as we will see in the next chapter, it also allows us to transport many other meta-theoretic properties from the first-order setting.

Chapter 10

Consequences of the Reduction

Now that the reduction is fully verified we can use it to obtain second-order Henkin completeness from the first-order completeness theorem (Section 10.1). While the focus of the previous chapters lied on completeness, it can also be used to transport many other properties from first-order logic. As an example we will derive the model existence properties for Henkin semantics that we refuted for standard semantics in Chapter 4 (i.e. compactness in Section 10.2 and Löwenheim-Skolem in Section 10.3).

10.1 Completeness

We derive the completeness of $\vdash_2$ from the completeness of $\vdash_1$:

Theorem 10.1 *If $\vdash_1$ is complete, then so is $\vdash_2$ for Henkin semantics.*

Proof Follows by Theorem 8.16 and Theorem 9.32. □

Using the mechanization of Forster, Kirst, and Wehr [15], we could then obtain completeness. However, we have not merged the developments as part of this thesis. Therefore this corollary is not mechanized:

Corollary 10.2 (Completeness) *$\vdash_2$ is complete under* LEM.

10.2 Compactness

Compactness follows as a classical consequence of completeness:

Theorem 10.3 (Compactness by Completeness) *If first-order logic is complete, then second-order logic with Henkin semantics is compact under* LEM.

Proof Let $\mathcal{T}$ be a second-order theory. Assume that all finite contexts $A \subseteq \mathcal{T}$ have a model. Since we assume LEM we can argue classically. Suppose there were no model of $\mathcal{T}$. Then $\mathcal{T} \vDash_2 \dot{\bot}$ and by completeness (Theorem 10.1) $\mathcal{T} \vdash_2 \dot{\bot}$. Hence, there is also a finite context $A \subseteq \mathcal{T}$ with $A \vDash_2 \dot{\bot}$ any by soundness $A \vdash_2 \dot{\bot}$. This is not possible, since by assumption $A \subseteq \mathcal{T}$ has a model. □

This is essentially the same proof that allowed us to conclude infinitary incompleteness in Theorem 4.13. However, we can also directly derive Henkin compactness from first-order compactness, only requiring the semantic part of the reduction:

Lemma 10.4 (Model Existence Reduction) *Let $\mathcal{T}$ be a second-order theory. Then*

1. *If $\mathcal{T}$ has a Henkin model then $\mathcal{T}^\star$ has a first-order model.*
2. *If $(\mathcal{T} \cup \mathcal{C})^\star$ has a first-order model then $\mathcal{T}$ has a Henkin model.*

Proof We discuss both properties:

1. Let $\mathcal{H}$ and ρ be a Henkin model and environment with $\mathcal{H}, \rho \vDash_2 \mathcal{T}$. Then $\mathcal{H}^\star, \rho^\star \vDash_1 \mathcal{T}^\star$ by Lemma 8.6.
2. Let $\mathcal{M}$ and ρ be a first-order model and environment with $\mathcal{M}, \rho \vDash_1 (\mathcal{T} \cup \mathcal{C})^\star$. Then $\mathcal{M}^\diamond, \rho^\diamond \vDash_2 \mathcal{T} \cup \mathcal{C}$ by Lemma 8.12 and thus $\mathcal{M}^\diamond, \rho^\diamond \vDash_2 \mathcal{T}$.

Theorem 10.5 (Compactness by Reduction) *If first-order logic is compact, then so is second-order logic with Henkin semantics.*

Proof Let $\mathcal{T}$ be a second-order theory. Assume that all finite contexts $A \subseteq \mathcal{T}$ have a Henkin model. To show that $\mathcal{T}$ has a Henkin model it suffices to show that $(\mathcal{T} \cup \mathcal{C})^\star$ has a first-order model by Lemma 10.4. By first-order compactness it is enough to verify that every finite $A^\star \subseteq (\mathcal{T} \cup \mathcal{C})^\star$ has a first-order model. We split this context into $A^\star = (A_\mathcal{T})^\star +\!\!+ (A_\mathcal{C})^\star$ with $A_\mathcal{T} \subseteq \mathcal{T}$ and $A_\mathcal{C} \subseteq \mathcal{C}$. By assumption, we know that $A_\mathcal{T}$ has a Henkin model $\mathcal{H}$ which gives us a first-order model $\mathcal{H}^\star$ using Lemma 10.4. Since by construction, $\mathcal{H}^\star$ satisfies comprehension it is also a model of $(A_\mathcal{T})^\star +\!\!+ (A_\mathcal{C})^\star = A^\star$. □

10.3 Löwenheim-Skolem

Instead of separating the Löwenheim-Skolem theorems into an upward and downward direction, we use a version that combines both of them into one statement. That is, we say a logic has the Löwenheim-Skolem property if every theory with an infinite model has models of every infinite cardinality. This is equivalent to the more common upward and downward formulation.

Theorem 10.6 (Löwenheim-Skolem) *If first-order logic has the Löwenheim-Skolem property, then so does second-order logic with Henkin semantics.*

Proof Let $\mathcal{T}$ be a second-order theory with an infinite Henkin model $\mathcal{H}$. We need to show that there is also a model for an arbitrary cardinality. It suffices to find a first-order model $\mathcal{M}$ of $(\mathcal{T} \cup \mathcal{C})^\star$ with this cardinality since the Henkin model $\mathcal{M}^\star$ preserves this cardinality. Thus, by the first-order Löwenheim-Skolem property it suffices to show that $(\mathcal{T} \cup \mathcal{C})^\star$ has some infinite first-order model. Since $\mathcal{H}^\star$ is infinite and also satisfies $(\mathcal{T} \cup \mathcal{C})^\star$, we are finished. □

We used this combined statement since it allows for an easier proof. Showing the upwards direction on its own using the reduction would be more complicated. Turning a Henkin model $\mathcal{H}$ into a first-order model $\mathcal{H}^\star$ can increase the cardinality, since the translated domain $\mathsf{D}^\star$ also contains the predicate universe $\mathbb{U}$ (see Definition 8.1). Thus, one might need to "go down" first in order to end up with the desired cardinality. Our approach sidesteps this issue and produces a result just as strong.

Finally, this result also indicates that PA_2 is not categorical for Henkin semantics. Also it becomes clear that Henkin semantics does no permit a categorical axiomatisation of the natural numbers in general.

Chapter 11

Discussion

We conclude this thesis with a brief summary of our results in Section 11.1 and discuss related (Section 11.2) and future work (Section 11.3).

11.1 Summary

We have formalized various well-known results regarding two different semantics of second-order logic that behave very differently on a meta-logical level. On the one hand, we have the standard Tarski semantics that harnesses the full power of the second-order quantifiers by allowing them to range over any property on the domain of discourse. This results in enough expressive power to categorically axiomatise the natural numbers as seen in Chapter 4. We showed that $\mathbb{N}$ is the only model of second-order Peano arithmetic, demonstrating that second-order standard semantics is capable of uniquely describing infinite structures, unlike its first-order counterpart. This lead us to conclude that second-order logic with standard semantics is neither compact, nor does it satisfy the upward Löwenheim-Skolem property. In Chapter 5, we used the categoricity result to show that any sound and enumerable deduction system must be incomplete, including the natural deduction system introduced in Chapter 3. We made use of a computability argument using a reduction from the decidability of Diophantine equations which also yielded undecidability of satisfiability and validity in second-order logic and PA_2.

On the opposite end of the spectrum lie Henkin semantics, where quantifiers do not range over all properties, but only over a subset of them specified by the model. In Chapters 7, 8, and 9 we showed that Henkin semantics make second-order logic collapse to first-order logic. In particular, this means that completeness, compactness, and the Löwenheim-Skolem theorems can be transported from first- to second-order logic as seen in Chapter 10. The cost of obtaining those results is that Henkin semantics sacrifice the expressivity of second-order quantification, no longer permitting a categorical axiomatisation of the natural numbers.

Overall, we have seen that categoricity and completeness pull in different directions; one cannot have both at the same time. Choosing standard semantics allows for categorical axiomatizations of infinite structures at the cost of completeness and meta-theoretic results, whereas Henkin semantics recover completeness and first-order meta-theory at the cost of expressive power and categoricity.

We also investigated the role of function quantifiers in second-order logic. In the classical set-theoretic meta-theory, they are simply a notational convenience and can be reduced to predicates and vice versa. The situation is more interesting in our setting of constructive type theory, where one must choose between interpreting functions as type-theoretic functions or total and functional relations. The former approach is more natural and allows for easier mechanization, while the latter is more faithful to the traditional set theoretic foundation. We observed that this choice is indeed relevant as the isomorphism in Chapter 4 is not necessarily computable, making the translation of type theoretic functions impossible. Of course, altering the meta-theory by assuming unique choice would resolve those differences. All in all, we believe that while functions quantifiers are useful in certain instances (for example see the discussion around embedding signatures into formulas in Lemma 5.14), in particular the relational semantics make them difficult to handle and complicate the development overall. Therefore, we chose not to focus on them on paper in this thesis.

11.2 Related work

Mechanized first- and higher-order logic While we are not aware of any previous mechanizations of second-order logic, there have been many formalizations concerned with first- or higher-order logics in various theorem provers like Isabelle/HOL [22, 23, 35], Mizar [4], Lean [65], and Coq [15, 32, 28, 3]. Our work is largely based on the Coq formalization of first-order logic developed in [15] and [32]. We adopted their major design decisions of using a syntax with de Bruijn binders parametrized over an arbitrary signature. The second-order natural deduction introduced in Chapter 3 is also an extension of the first-order system presented in [15]. We also want to point to the work by Kirst and Smolka [33, 34] in which they investigate second-order ZF set theory in Coq, though not via an embedded second-order logic, showing that the theory is categorical for equipotent models.

Synthetic computability theory The development of synthetic computability for the type theory of Coq is due to Forster, Kirst, and Smolka [14]. Their work lays the foundation for our synthetic analysis of incompleteness and undecidability carried out in Chapter 5. Crucially, the synthetic undecidability of Hilbert's tenth problem has been verified by Larchey-Wendling and Forster [37] as part of the Coq Library of Undecidability Proofs [12], which serves as the starting point for our own reductions. The undecidability results mechanized in this thesis are also intended as a contribution to this library.

Incompleteness The synthetic account of Gödel's incompleteness theorem used in Chapter 5 was developed by Kirst and Hermes [31]. The initial reduction from Hilbert's tenth problem we use to establish undecidability of satisfiability in $\mathbb{N}$ is drawn from their work. They also initially proposed the idea of using a synthetic computability argument to obtain incompleteness results circumventing the more tedious mechanization of explicit independent Gödel/Rosser sentences.

Sources Finally, this work is based on Shapiro's [56] account of second-order logic and the method used in Chapter 7 to reduce second- to first-order logic is due to Nour and Raffalli [43].

11.3 Future Work

Throughout this thesis, we have used PA_2 to demonstrate the ability of second-order logic to characterize infinite structures. We want to remark that second-order arithmetic and subsystems of it are also frequently used in the programme of reverse mathematics with some noteworthy connections to PA_1. Restricting induction to first-order definable properties and comprehension to first-order properties with second-order parameters results in a system called arithmetical comprehension (ACA_0). Interestingly, ACA_0 is a conservative extension of PA_1, i.e. it proves the same first-order statements as PA_1 does [57]. Investigating those subsystems and their conservativity properties would give further insight into the relationship between first- and second-order Peano arithmetic.

Apart from the natural numbers, it would also be worthwhile to explore other structures. For example, as Shapiro shows, the categoricity of PA_2 can be used to prove that second-order analysis is categorical [56]. This serves as an example of a theory that only permits uncountable models, refuting the downward Löwenheim-Skolem theorem. It would be interesting to investigate whether Shapiro's construction also works in our setting where the isomorphisms are not computable. Another example of uncountable structures is second-order ZF set theory. Kirst and Smolka already provide a mechanization of this fact in Coq [33] that could be transported to our second-order logic backbone.

With regards to categoricity, there is also the notion of so called *internal categoricity* due to Väänänen and Wang [66, 68]. It draws from the idea that second-order logic is able to express its own categoricity. Consider a theory $\mathcal{T}$ depending on a single predicate symbol $\mathcal{P}$. We can construct a second-order formula

$$\mathsf{Categ}(\mathcal{T}) := \dot{\forall} \mathsf{D}_1 \mathsf{D}_2 \mathsf{P}_1 \mathsf{P}_2 .\, \mathcal{T}(\mathsf{P}_1)^{\mathsf{D}_1} \dot{\rightarrow} \mathcal{T}(\mathsf{P}_2)^{\mathsf{D}_2} \dot{\rightarrow} \dot{\exists} \cong .\, \mathsf{Iso}(\dot{\cong}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{P}_1, \mathsf{P}_2)$$

where $\mathcal{T}(\mathsf{P}_1)^{\mathsf{D}_1}$ replaces the symbol $\mathcal{P}$ with the variable P_1 and guards all quantifiers with the domain predicate D_1. Then, $\mathcal{T}$ is categorical iff $\vDash \mathsf{Categ}(\mathcal{T})$. The remarkable insight is that this statement is provable in many cases (despite incompleteness), for example we have $\vdash \mathsf{Categ}(\mathsf{PA}_2)$. This means that although being a meta-theoretical concept, categoricity can be written and proven at the object level, without depending on any external set theory (or type theory in our case). Additionally, we know that $\vdash \mathsf{Categ}(\mathcal{T})$ is equivalent to validity of $\mathsf{Categ}(\mathcal{T})$ in Henkin semantics, yielding a notion of categoricity that holds for full models and *inside* Henkin models. Väänänen interprets this as "a bridge between full semantics and Henkin semantics" [67]. This would be a very interesting subject to tackle in a future formalization.

On the mechanization side it would be useful to develop further tooling to assist future formalization efforts in second-order logic. For example, while we did not need to derive concrete statements in the deduction system, this would certainly be necessary when verifying $\vdash \mathsf{Categ}(\mathsf{PA}_2)$. Constructing such derivation by hand rule by rule would be very tedious and made even more complicated by the de Bruijn encoding. This could be alleviated by extending the first-order proof mode developed by Hostert, Koch and Kirst [29] to second-order logic.

While our mechanization is currently stand-alone we plan on making it compatible with and integrating parts of it into the Coq Library of Undecidability Proofs. In particular, this would make the reduction presented in Chapter 7 more accessible. For example, a future mechanization of the first-order Löwenheim-Skolem theorems based on the definitions in the library would immediately yield the same property for second-order Henkin semantics via the argument in Chapter 10. We believe that, in principle, the reduction can be used to transport similar first-order properties that might be formalized in the future.

We also plan on merging the relevant first-order completeness proof by Forster, Kirst, and Wehr [15] into our development to obtain a fully mechanized version of Corollary 10.2. In their work, Forster, Kirst, and Wehr analyse the constructiveness of first-order completeness and particularly focus on the $\dot\forall, \dot\rightarrow, \dot\bot$-fragment. While the extension to full first-order logic used in this thesis requires LEM, they showed that completeness for the restricted fragment is equivalent to Markov's principle. It might be worthwhile to investigate whether our treatment of second-order logic and in particular the reduction to first-order logic can be altered to work in this fragment. However, the dependency of comprehension on the existential quantifier might serve as an obstacle for this task.

Finally, one could also investigate second-order intuitionistic Kripke models. Interestingly, Nour and Raffalli [43] extend their reduction to Kripe models which allows them to additionally transport intuitionistic completeness from second-order logic. They claim that the resulting second-order Kripke models are similar to Prawitz's adaptation of Beth's models [48].

Appendix A

Coq Mechanization

Working in a proof assistant turned out to be very helpful in the development of this thesis, in particular with verifying the complex deductive reduction in Chapter 9. We remark on some interesting aspects of the mechanization and give a brief overview of its structure.

A.1 Remarks on the Mechanization

The mechanization is completely self-contained and follows the structure presented in this thesis. We do not assume any global axioms. The usage of LEM and MP is local to the theorems as remarked throughout the thesis. Thus, all results where not stated otherwise are fully constructive.

Our formalization is closely related to the first-order design developed in [15] and has similar features. As explained in Section 3.1, we parameterize the syntax over a signature Σ. To avoid having to specify Σ for every formula, we define it as a type class, allowing the appropriate signature to be automatically inferred based on the context. We use an inductive type `form` for formulas and `term` for terms. Those types also incorporate function quantifiers and variables such that most results and notions are directly formalized in terms of the $\mathfrak{F}_2^F$ and $\mathfrak{T}_2^F$ fragment. We use predicates `first_order` and `func_free` to restrict to $\mathfrak{F}_1$ and $\mathfrak{F}_2$ when necessary. The only exception is the first-order reduction. Here, we use the formalization of first-order logic from the Coq Library of Undecidability proof in anticipation of easing compatibility.

Coq is not able to derive a useful induction principle for terms because of the nested occurrence of the inductive vector type. We manually prove the following induction scheme

$$\forall P : \mathfrak{T}_2^F \to \mathbb{P}.\, (\forall i.\, P\, x_i) \to (\forall \mathcal{F} \boldsymbol{v}.\, P\, \boldsymbol{v} \to P\, (\mathcal{F}\, \boldsymbol{v})) \to (\forall i n.\, P\, \boldsymbol{v} \to P\, (f_i^n\, \boldsymbol{v})) \to \forall t.\, P\, t$$

where $P\, \boldsymbol{v}$ denotes that P holds for all components of the vector $\boldsymbol{v}$.

Furthermore, the mechanization treats second-order substitution slightly different than presented in Definition 3.7. Instead of annotating a substitution $\varphi[\sigma]_p^n$ with an arity n and using $\sigma : \mathbb{N} \to \mathfrak{P}_n$ which is more readable on paper, we employ substitutions that affect variables of all arities at once. Hence, the formalization uses $\sigma : \mathbb{N} \to \forall n. \mathfrak{P}_n$. Restricting σ to a single arity can then be obtained by:

$$\varphi[\sigma]_p^n := \varphi\,[\lambda i m.\,\mathsf{if}\ \ulcorner m = n \urcorner\ \mathsf{then}\ \sigma\, i\, m\ \mathsf{else}\ p_i^m\,]_p$$

This simplifies the definition of arity bounded substitution in Definition 9.11 to:

$$\varphi[\sigma]_p^{<n} := \varphi\,[\lambda i m.\,\mathsf{if}\ \ulcorner m < n \urcorner\ \mathsf{then}\ \sigma\, i\, m\ \mathsf{else}\ p_i^m\,]_p$$

A.2 Structure of the Mechanization

We reuse some external code from the Coq Library of Undecidability proofs, in particular related to the definitions of synthetic computability presented in Chapter 2 and the formalization of first-order logic, amounting to 1157 lines of the preliminaries overall. The table below summarizes the lines of code associated with each chapter of this thesis.

Chapter	**Specification**	**Proofs**
2. Preliminaries	706	879
3. Syntax, Semantics & Deduction	1224	2424
4. Categoricity of PA_2	554	1089
5. Incompleteness & Undecidability	185	399
6. Henkin Semantics	145	347
7. Completeness via Reduction	400	978
8. Consequences of Reduction	48	160
Total	3262	6276

The mechanization also contains an extension of the semantic reduction from Chapter 8 to the full $\mathfrak{F}_2^F$ fragment, as well as the semantic part of Van Dalens reduction from Example 7.3.

Bibliography

[1] The coq proof assistant. URL `https://coq.inria.fr/`. Accessed August 19 2021.

[2] Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. ISSN 1571-0661. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[3] Chad E. Brown. A semantics for intuitionistic higher-order logic supporting higher-order abstract syntax. 2014.

[4] Marco Caminati. Basic first-order model theory in mizar. *Journal of Formalized Reasoning*, 3, 01 2010.

[5] Thierry Coquand and Gérard Huet. *The calculus of constructions*. PhD thesis, INRIA, 1986.

[6] Thierry Coquand and Bassel Mannaa. The independence of markov's principle in type theory. 02 2016.

[7] Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012.

[8] Nicolaas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972. ISSN 1385-7258.

[9] Richard Dedekind. *Was sind und was sollen die Zahlen? / von Richard Dedekind*. Vieweg, Braunschweig, 1888. doi: 10.24355/dbbs.084-200902200100-1.

[10] Andrej Dudenhefner. Undecidability of Semi-Unification on a Napkin. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, volume 167 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:16, Dagstuhl, Germany, 2020.

Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-155-9. doi: 10.4230/LIPIcs.FSCD.2020.9.

[11] Ronald Fagin. Generalized first-order spectra, and polynomial. time recognizable sets. *SIAM-AMS Proc.*, 7, 01 1974.

[12] Y. Forster, Dominique, Larchey-Wendling, Andrej, Dudenhefner, Edith Heiter, Dominik Kirst, F. Kunze, and G. Smolka. A coq library of undecidable problems. 2019.

[13] Yannick Forster and Dominique Larchey-Wendling. Certified undecidability of intuitionistic linear logic via binary stack machines and minsky machines. CPP 2019, page 104–117, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362221.

[14] Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in coq, with an application to the entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019, page 38–51, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362221.

[15] Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness theorems for first-order logic analysed in constructive type theory: Extended version. *Journal of Logic and Computation*, 31(1):112–151, 01 2021. ISSN 0955-792X.

[16] Gottlob Frege. Begriffsschrift, eine der arithmetischen nachgebildete formelsprache des reinen denkens. *Halle: Verlag L. Nebart. Preface and Part*, 1:47–78, 1879.

[17] Gottlob Frege. *Grundlagen der Arithmetik: studienausgabe mit dem Text der Centenarausgabe*, volume 366. Felix Meiner Verlag, 1988.

[18] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39:176–210, 1935.

[19] Gerhard Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, 39:405–431, 1935.

[20] Kurt Gödel. *Über die vollständigkeit des logikkalküls*. na, 1929.

[21] Kurt Gödel. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik*, 38(1):173–198, 1931.

[22] John Harrison. Formalizing basic first order model theory. In Jim Grundy and Malcolm Newey, editors, *Theorem Proving in Higher Order Logics*, pages

153–170, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. ISBN 978-3-540-49801-8.

[23] John Harrison. Towards self-verification of hol light. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the third International Joint Conference, IJCAR 2006*, volume 4130 of *Lecture Notes in Computer Science*, pages 177–191, Seattle, WA, 2006. Springer-Verlag.

[24] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[25] Leon Henkin. The completeness of the first-order functional calculus. *The journal of symbolic logic*, 14(3):159–166, 1949.

[26] Leon Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, 1950.

[27] Hugo Herbelin. An intuitionistic logic that proves markov's principle. *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 50–56, 2010.

[28] Hugo Herberlin, Sunyoung Kim, and Gyesik Lee. Formalizing the meta-theory of first-order predicate logic. *Journal of the Korean Mathematical Society*, 54:1521–1536, 01 2017.

[29] Johannes Hostert, Mark Koch, and Dominik Kirst. A toolbox for mechanised first-order logic. *The Coq Workshop*, 2021.

[30] Neil Immerman. *Descriptive complexity*. Springer Science & Business Media, 2012.

[31] Dominik Kirst and Marc Hermes. Synthetic undecidability and incompleteness of first-order axiom systems in coq. In *ITP*, 2021.

[32] Dominik Kirst and Dominique Larchey-Wendling. Trakhtenbrot's theorem in coq: A constructive approach to finite model theory. *International Joint Conference on Automated Reasoning*, 2020.

[33] Dominik Kirst and Gert Smolka. Categoricity results for second-order zf in dependent type theory. In *ITP*, 2017.

[34] Dominik Kirst and Gert Smolka. Categoricity results and large model constructions for second-order zf in dependent type theory. *Journal of Automated Reasoning*, 63:415–438, 2018.

[35] Ramana Kumar, Rob Arthan, Magnus O. Myreen, and Scott Owens. Self-formalisation of higher-order logic. *Journal of Automated Reasoning*, 56(3):221–259, 2016.

[36] Boris A. Kushner. Markov's constructive analysis; a participant's view. *Theoretical Computer Science*, 219(1):267–285, 1999. ISSN 0304-3975.

[37] Dominique Larchey-Wendling and Yannick Forster. Hilbert's Tenth Problem in Coq. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:20, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-107-8. doi: 10.4230/LIPIcs.FSCD.2019.27.

[38] Daniel Leivant. Higher order logic. 02 1994.

[39] Per Lindström. On extensions of elementary logic. *Theoria*, 35(1):1–11, 1969.

[40] Leopold Löwenheim. Über möglichkeiten im relativkalkül. *Mathematische Annalen*, 76(4):447–470, 1915.

[41] Maria Manzano. *Extensions of first-order logic*, volume 19 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.

[42] Yuri Matiyasevich. *Martin Davis and Hilbert's Tenth Problem*, pages 35–54. Springer International Publishing, Cham, 2016. ISBN 978-3-319-41842-1.

[43] Karim Nour and Christophe Raffalli. Simple proof of the completeness theorem for second-order classical and intuitionistic logic by reduction to first-order mono-sorted logic. *Theoretical computer science*, 308(1-3):227–237, 2003.

[44] Russell O'Connor. Essential incompleteness of arithmetic verified by coq. In *Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics*, TPHOLs'05, page 245–260, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3540283722.

[45] 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*, pages 328–345, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. ISBN 978-3-540-47586-6.

[46] Lawrence C. Paulson. A mechanised proof of gödel's incompleteness theorems using nominal isabelle. *Journal of Automated Reasoning*, 55:1–37, 2015.

[47] Giuseppe Peano. *Arithmetices principia: Nova methodo exposita*. Fratres Bocca, 1889.

[48] Dag Prawitz. Some results for intuitionistic logic with second order quantification rules. In A. Kino, J. Myhill, and R.E. Vesley, editors, *Intuitionism and Proof Theory: Proceedings of the Summer Conference at Buffalo N.Y. 1968*, vol-

ume 60 of *Studies in Logic and the Foundations of Mathematics*, pages 259–269. Elsevier, 1970.

[49] Willard V. Quine. *Philosophy of logic*. Harvard University Press, 1986.

[50] Stephen Read. Completeness and categoricity: Frege, gödel and model theory. *History and Philosophy of Logic*, 18(2):79–93, 1997.

[51] Fred Richman. Church's thesis without tears. *The Journal of Symbolic Logic*, 48 (3):797–803, 1983.

[52] Marcus Rossberg. First-order logic , second-order logic , and completeness. 2004.

[53] Barkley Rosser. Extensions of some theorems of gödel and church. *Journal of Symbolic Logic*, 1(3):87–91, 1936.

[54] Bertrand Russell and Alfred N. Whitehead. Principia mathematica vol. i. 1910.

[55] Natarajan Shankar. *Metamathematics, Machines and Gödel's Proof*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

[56] Stewart Shapiro. *Foundations without foundationalism: A case for second-order logic*, volume 17. Clarendon Press, 1991.

[57] Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic. Cambridge University Press, 2 edition, 2009.

[58] Gert Smolka. Computational type theory and interactive theorem proving with coq. `https://www.ps.uni-saarland.de/~smolka/drafts/icl2020.pdf`, 2020. Accessed July 30 2021.

[59] Simon Spies and Yannick Forster. Undecidability of higher-order unification formalised in coq. CPP 2020, page 143–157, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370974.

[60] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. Autosubst 2: Reasoning with multi-sorted de bruijn terms and vector substitutions. CPP 2019, page 166–180, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362221.

[61] Alfred Tarski. The semantic conception of truth: and the foundations of semantics. *Philosophy and Phenomenological Research*, 4(3):341–376, 1944.

[62] Alfred Tarski and Robert L. Vaught. Arithmetical extensions of relational systems. *Compositio Mathematica*, 13:81–102, 1958.

[63] Amin Timany and Matthieu Sozeau. Cumulative Inductive Types In Coq. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for*

Computation and Deduction (FSCD 2018), volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-077-4.

[64] Dirk Van Dalen. *Logic and structure*, volume 3. Springer, 1994.

[65] Luiz Viana. Proving the consistency of logic in lean. pages 1–8, 08 2020.

[66] Jouko Väänänen. Second-order logic and set theory. *Philosophy Compass*, 10 (7):463–478, 2015.

[67] Jouko Väänänen. Second-order and higher-order logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[68] Jouko Väänänen and Tong Wang. Internal categoricity in arithmetic and set theory. *Notre Dame Journal of Formal Logic*, 56, 01 2012.

[69] Ernst Zermelo. Über grenzzahlen und mengenbereiche: Neue untersuchungen über die grundlagen der mengenlehre. *Fundamenta mathematicae*, 16, 1930.