Saarland University

Faculty of Mathematics and Computer Science

Bachelor's Thesis

The Undecidability Of First-Order Logic over Small Signatures

Author
Johannes Hostert

Advisors
Dr. Andrej Dudenhefner
Dominik Kirst

Reviewers
Prof. Dr. Gert Smolka
Dr. Andrej Dudenhefner

Submitted: 30th September 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, 30th September, 2021

Abstract

In 1936, Church and Turing independently negatively answered the Entscheidungsproblem, showing that validity, satisfiability, and provability are undecidable for first-order logic. It was proven shortly afterwards by Kalmár and others that this result holds even when formulas are restricted to just a single binary relation. In 1950, Trakhtenbrot showed this signature is also the smallest one for which finite first-order logic satisfiability is undecidable.

Over the last few decades, there has been interest in mechanizing mathematical proofs in interactive theorem provers. Since then, a significant amount of mathematics was mechanized, including the mentioned undecidability results for the first-order problems.

In this thesis, we present and mechanize a new proof of the undecidability of these first-order problems in the minimal signature. Our proof aims to be more feasible and more direct compared to previous mechanizations, by starting at a variant of Diophantine constraints, a problem shown undecidable in 1970 by Matiyasevitch.

While minimizing the signature, we also investigate the minimal class of logical connectives required for the aforementioned problems to be undecidable. In particular, we present the first mechanized proof that first-order validity and provability are undecidable for formulas with a single binary relation in the forall-implicative fragment without negation.

Acknowledgements

I want to thank my advisors Andrej Dudenhefner and Dominik Kirst for their outstanding advice and support during this project. In particular, I am thankful for the insightful discussions we had each week, for the extremely helpful feedback given by them, and for entrusting me with this project to begin with.

I would like to thank Prof. Smolka for allowing me to write this thesis at his group. I also thank him and Andrej for reviewing this thesis. Furthermore, I am thankful to Prof. Bläser for mentoring me during my Bachelor's studies.

Finally, I am thankful to my friends and family for supporting me, especially during the last few months. I am especially thankful to Lorenz and Niklas for proof-reading my thesis.

Contents

Chapter 1

Introduction

In the early 20th century, many mathematicians became increasingly interested in formalizing the entirety of mathematics on a uniform foundation. In doing so, logicians invented several logics and deduction systems, aiming to give a syntactical definition to mathematical truth. In particular, first-order logic became influential. In 1928, Hilbert and Ackermann posed the Entscheidungsproblem [13], asking for a general and effective procedure for deciding whether a given statement in first-order logic is valid. In 1936, Church [2] and Turing [29] independently showed that such a general procedure does not exist – validity, satisfiability and provability for first-order logic are undecidable.

Parallel to this, logicians had already begun investigating "reductions" of first-order formulas, which involved minimizing formulas into a smaller version of first-order logic while preserving validity. In particular, one could reduce the number and arity of predicate symbols, or the number of the allowed logical connectives – i.e., by restricting oneself to a smaller signature or logical fragment. Alongside this, there also was interest in finite model theory, which (among other things) involved restricting the semantics of first-order logic.

Seminal results by Kalmár ([16], for small signatures) and Trakhtenbrot ([28], for small signatures on finite models) gave sharper descriptions of the problems related to the Entscheidungsproblem: They are undecidable as soon as there is a single binary predicate, even when one only considers formulas using a heavily restricted set of logical connectives, in some cases not even involving falsity. For finite models, the same results hold, except that one may not necessarily restrict the use of falsity. Since then, several different strategies for proving these results have been found.

Related to these developments was the search for a foundation of mathematics. Nowadays, axiomatic (Zermelo-Fraenkel) set theory is considered the "dominant" foundation. Alternative theories have been proposed, most importantly type theory [30, 23]. Type theory bases all mathematics on types instead of sets. Along with

this, intuitionistic or constructive logic, which does not assume the law of excluded middle, has become increasingly important, since such a logic is much more natural in type theory.

Type theory was hugely influential in the development of proof assistants like Coq [27], Agda [25], or Lean [4], which are computer programs that can check mathematical proofs for correctness. Such proof assistants are often based on type-theoretic mathematics because type theory often offers a natural implementation as a computer program. Mechanizing a proof in such an assistant allows one to be very sure about its correctness, but this often involves significant work since the proof has to be given completely, including lemmas usually considered trivial.

For computability theory, there are several different approaches for mechanization in a proof assistant: First, one can follow the usual development of classical computability theory and mechanize Turing machines or equivalent models of computation. However, this is often undesirable due to the large amount of specificity required for the mechanization. A different, *synthetic* approach by Bauer [1] building on work by Richman [26] involves recognizing that a change of one's foundation of mathematics to one only admitting computable functions can abstract from having to specify a concrete model of computation. Type theory in particular is such a computable foundation. This approach has been adapted by Forster et al. [5] to also mechanize undecidability proofs.

A significant collection of undecidability proofs for many problems has been collected in the Coq Library of Undecidability Proofs [6], including the mentioned results for small signatures [17] and finite models [18]. However, the existing mechanizations of these results are rather involved, proceeding using ZF (a first-order set theory), or by a mechanized chain of quotients and signature compressions.

1.1 Contributions

In this thesis, we contribute alternative proofs of the undecidability of validity, satisfiability, provability, and finite satisfiability for small signatures, which we suggest is more feasible compared to the existing approaches. Our approach starts by picking a special Diophantine constraints problem UDPC which is easily axiomatizable in first-order logic. In general, it is undecidable whether a system of Diophantine equations has a solution [24], a result which has also been mechanized [20]. While this approach allows for a shorter proof compared to previous mechanizations, it relies on the undecidability of Diophantine equations, which has been very hard to prove historically.

Besides showing the undecidability of the mentioned problems for small signatures, we also show undecidability for the small logical fragment. In particular, we show that it suffices to allow only forall quantifiers, implication, and falsity in

first-order formulas for the mentioned problems to become undecidable, and for some problems, even falsity is unnecessary. As far as we know, we present the first mechanization of the undecidability of validity and provability for the minimal signature and minimal logical fragment.

The definition of the problem UDPC is not due to me. My contributions are showing the undecidability of UDPC by reducing from UDC, the inductive characterization of UDPC and the remaining reductions to VAL, PRV, FSAT and so on. For the VAL reduction, I was initially given a first-order axiomatization, only contributing the proof that this reduction function is correct. Later, I further modified that reduction function into the current version of Chapter 5, and then transformed it into the version of Chapters 6 and 7. For FSAT and Chapter 8, I found and verified the reduction function presented there. The mechanization of the double negation translation of Chapter 9 is also due to me.

1.2 Outline

This thesis is structured as follows: In Chapter 2, we introduce our type-theoretic setting. In Chapter 3, we further introduce basic concepts like first-order logic and synthetic computability theory. In Chapter 4, we present our source problem UDPC, which allows for elegantly short reduction. Chapter 5 gives a first proof of the undecidability of validity over small signatures, which is refined in Chapter 6 to only require a small logical fragment. Chapter 7 extends the results to show that natural deduction systems for first-order logic are also undecidable, concluding that satisfiability, validity and provability are undecidable in the mentioned small cases. Continuing in Chapter 8, we show that finite satisfiability (i.e. satisfiability restricted to finite models) is undecidable for small signatures, which is then extended a small logical fragment in Chapter 9. Chapter 10 summarizes our results, details insights on the Coq mechanization, and compares them to the existing literature and mechanizations.

Additionally, we include a complete mechanization of our results in the Coq proof assistant. In the electronic version of this thesis, most lemmas, theorems, and statements are annotated with a link leading to the corresponding lemma in the mechanization.

Chapter 2

On Intuitionistic Type Theory

To begin, we first introduce basic concepts of type theory for readers unfamiliar with the topic.

Type theory is an alternative foundation of mathematics. Usually, set theory is considered the foundation of mathematics, which means that all theorems, definitions, and proofs are defined and done in the language of set theory. In type theory, however, we have types and their elements as the primitives. Our particular variant of type theory is the Calculus of inductive Constructions [3] (CIC), as implemented in the Coq proof assistant [27].

This type theory is dependent and features a single type $\mathbb{P}$ of propositions, and an infinite hierachy of type universes $\mathbb{T}_1 : \mathbb{T}_2 : \mathbb{T}_3 : \ldots$. (Indices are omitted wherever possible.) The internal language of CIC actually is a functional programming language, used to write definitions, propositions, and proofs. Propositions are represented by types (specifically, of type $\mathbb{P}$), while proofs are programs of said type, following the Curry-Howard isomorphism [15]. The fundamental statement is that of type assignment: $x : T$ describes that x is of type T. For example, $1 : \mathbb{N} : \mathbb{T}_1$ means that 1 is an element of type $\mathbb{N}$, the type of natural numbers. $\mathbb{N}$ itself has type $\mathbb{T}_1$, the (first) type of types. Both 1 and $\mathbb{N}$ can furthermore be defined using the language of CIC, as is demonstrated shortly.

To continue this example, a statement like $\forall xy : \mathbb{N}.x + y =_{\mathbb{N}} y + x$ has type $\mathbb{P}$, making it a proposition. The statement $x =_{\mathbb{N}} y : \mathbb{P}$ also is a type, specifically the type of proofs that $x : \mathbb{N}$ and $y : \mathbb{N}$ are identical. $+$ is a function of type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$, which is defined in the internal language of CIC. If we now wanted to prove that statement, we would have to find a term P in the internal language of CIC inhabiting that type. Such a term P would be a function taking two numbers and yielding a proof that these two given numbers are equal.[1] We then say that $P : (\forall xy : \mathbb{N}, x + y = y + x)$,

[1] The actual program proceeds by induction, case elimination, rewriting using $=$ and the definition of $+$, all of which can be done using the internal language of CIC.

or that $\forall xy : \mathbb{N}, x + y = y + x$ is inhabited. If we consider it a proposition, we simply say that $\forall xy : \mathbb{N}, x + y = y + x$ holds.

In order to check whether a proof is valid, we can check whether it has the required type. This type checking mechanism is implemented in the Coq proof assistant software, allowing us to have our proofs checked by a computer. However, this requires us to carry out all our proofs with the utmost level of rigor, forcing us to formally prove every utility lemma, including seemingly trivial ones. We consider this desirable, since we can be certain our proof is valid.

In the next section, we formally define the types representing propositions. It turns out that CIC, using these definitions, implements **intuitionistic logic**. This means that the law of excluded middle $\mathsf{LEM} := \forall P : \mathbb{P}.P \vee \neg P$ does not hold, as well as several equivalent statements like double negation elimination $\mathsf{DN} := \forall P : \mathbb{P}.\neg\neg P \to P$. Intuitionism is often considered "natural" in type theory, in particular because LEM, when interpreted as a computer program, denotes a program which "decides", for any proposition P, whether or not P is true. Such a program, of course, does not exist (by the very results investigated in this thesis) and so type theorists do not assume LEM or DN. This kind of logic is also called **constructive**, since all existence proofs (i.e. proofs of statements with an existential quantifier) must be done by explicitly constructing a witness – the common approach of showing that non-existence is contradictory implicitly relies on DN.

Often, it is insightful to compare this constructive approach to one where LEM holds. Since LEM is consistent and independent in CIC, we could assume it as an axiom, and change the properties of our type theory. We call such a type theory **classical meta-theory**, in contrast to the intuitionistic meta-theory we use by default.

To summarize, we try to re-implement mathematics by defining a special "programming language" along with a type system where types are powerful enough to denote propositions or arbitrary mathematical constructions. Programs of such types are then understood as proofs. The typing system is implemented in the proof assistant Coq, which can then mechanically verify our proofs, allowing us to be certain that our proofs are correct. The proof assistant also comes with a standard library of types, which we re-iterate in this chapter.

We refer to the practice of implementing such proofs in the Coq proof assistant as **mechanization**. Most proofs presented in this thesis are mechanized in Coq. Thus, we often only give a high-level overview of the proof ideas, while embedding a link to the actual mechanization in the digital version of this thesis. We encourage the reader to refer to this mechanization for more specific proof details.

2.1 The Types of CIC

Types in CIC are defined inductively, by giving a collection of constructors which can be used to construct points of said type. For example, $\mathbb{N}$, the type of natural numbers, has the constructors $0 : \mathbb{N}$ representing 0, and $S : \mathbb{N} \to \mathbb{N}$ representing the successor. An inductive type is the **smallest** type admitting these constructors, making $\mathbb{N}$ the smallest type having 0 and a successor function, which makes this definition of $\mathbb{N}$ analogous to the definition implicit in Peano Arithmetic. Smallest means that we can perform proofs by induction on that type. We sometimes denote such a type using a **BNF**, i.e. $n : \mathbb{N} ::= 0 \,|\, S\, n$. For certain types, in particular inductive predicates, we sometimes give the constructors in **inference rule notation**. For $\mathbb{N}$, this might look like the following:

$$0\ \frac{}{0 : \mathbb{N}} \qquad\qquad \mathsf{S}\ \frac{n : \mathbb{N}}{S\, n : \mathbb{N}}$$

This mechanism is strong enough to define all "base" types we need for this thesis. Additionally, we have the fundamental type of dependent functions $\forall x : A.T\, x$. Its inhabitants are functions taking an argument of type A, and yielding a result of type $T\, x$, where T can be defined using $x : A$. A special case is the type of non-dependent functions $A \to B$, where B does not depend on the argument. To define functions and construct proofs, we need a functional programming language, which is the λ-calculus, extended with mechanisms for induction and recursion on inductive types. Note that types also are terms in CIC, hence $\forall(x : A).T\, x$ also is part of its syntax.

Using this, we can define additional types needed for the remainder of this thesis. We have $\mathbb{0} : \mathbb{T}$, which is empty (it has no constructors), and $\mathbb{1} : \mathbb{T}$, which has a single constructor $\star : \mathbb{1}$. For $a : A, b : B$, we furthermore have types $A \times B : \mathbb{T}$ and $A + B : \mathbb{T}$, where $A \times B$ is the type of pairs (a, b), and $A + B$ is the sum type, or disjoint union, representing either an element of A or an element of B, constructed using either $i_1\, a : A + B$ or $i_2\, b : A + B$. Even stronger, we have dependent pairs $\sum_{x:A} T\, x$, where the definition of T can use $x : A$. Inhabitants of this type are tuples (x, y) where y is assigned a type $T\, x$ depending on x.

There is an important distinction between types living in $\mathbb{T}$, and types living in $\mathbb{P}$, the type of propositions, namely that propositions are "computationally irrelevant". For example, if we write a program $\mathbb{1} + \mathbb{1} \to \mathbb{N}$, the program can case-analyze the input of type $\mathbb{1} + \mathbb{1}$ when computing the output. If, however, we consider a "mirror version" $\mathbb{1} \vee \mathbb{1}$, where $A \vee B : \mathbb{P}$ is defined similar to $A + B$ except that it lives in $\mathbb{P}$, then the program may not case-analyze the input in general. There are multiple exceptions to this rule, the most important one is that this case-analysis is allowed if the output is also "computationally irrelevant", or if the input is syntactically irrelevant.

We define propositions mirroring our previous types, in particular $\bot$ for $\mathbb{0}$, $\top$ for $\mathbb{1}$, $A \wedge B$ for $A \times B$, $A \vee B$ for $A + B$ and $\exists x : A.T\,x$ for $\sum_{x:A} T\,x$. These types correspond to the well-known logical connectives suggested by the notation. As mentioned, we consider such a proposition "true" when it is inhabited.

We already adressed that in our meta-theory neither LEM nor DN hold. In this thesis, we do not assume LEM, DN, or any other axioms unless explicitly mentioned. All our main theorems are proven without assuming any axioms.

2.2 Derived Notation

We already introduced $\mathbb{N}$, the type of natural numbers. $\mathscr{L}(A)$ for $A : \mathbb{T}$ is the type of finite lists over elements of A, constructed from either the empty list $[]_A : \mathscr{L}(A)$, or by prepending a value $x : A$ to an already existing list $l : \mathscr{L}(A)$, denoted as $(x :: l) : \mathscr{L}(A)$. We often write down lists as $[x_1, x_2, \ldots x_n]$, denoting the list $x_1 :: x_2 :: \cdots :: x_n :: []$. We write $a \in l$ if $a : A$ is an element of the list $l : \mathscr{L}(A)$, and $l_1 \subseteq l_2$ if $\forall a \in l_1.a \in l_2$. While this notation mirrors set notation, note that lists are finite ordered collections, whereas sets are unordered collections where elements can occur at most once. We also omit the index $[]_A$ whenever possible.

An important function on lists is $map : \forall AB : \mathbb{T}.(A \to B) \to \mathscr{L}(A) \to \mathscr{L}(B)$, which applies a function $f : A \to B$ to every element of a list. Furthermore, we need $concat : \forall A : \mathbb{T}.\mathscr{L}(\mathscr{L}(A)) \to \mathscr{L}(A)$, which concatenates a list of lists into a single list. We also abbreviate $map\ f\ l$ as $[f\,a \mid a \in l]$. The composition $concat\ (map\ f\ l)$ is usually known as *flatmap* to functional programmers.

The type $\mathbb{B} ::= \texttt{tt} \mid \texttt{ff}$ models "booleans" representing either true or false. They are a datatype and not used to assign "truth" to propositions.

To continue, we have $\mathscr{O}(A)$, the type of optionals over A. The inhabitants are either $\emptyset : \mathscr{O}(A)$, representing the absence of an $a : A$, or $\lceil a \rceil$, representing the presence of such an $a : A$. Thus, $\mathscr{O}(A)$ has one more inhabitant than A (for finite A).

For $n : \mathbb{N}$, we also use D^n to denote lists of length n. In type theory, lists of fixed length n, where n is carried in the type, are known as **vectors**.

To better describe types, we introduce the following properties:

Definition 2.1 (Properties of types) *A type $T : \mathbb{T}$ is:*

- ***empty*** *iff it is not inhabited, i.e. if $T \to \bot$ is inhabited.*
- ***listable*** *or* ***finite*** *iff there is a list $l : \mathscr{L}(T)$ such that $\forall a : T, a \in l$.*
- ***countable*** *iff there is an injective relation $R : T \to \mathbb{N} \to \mathbb{P}$.*
- ***countably infinite*** *if there is a bijective relation $R : T \to \mathbb{N} \to \mathbb{P}$*

We note that $\mathbb{N}$ and $\mathscr{L}(A)$ are not listable (for nonempty A), while $\mathbb{0}, \mathbb{1}, \mathbb{B}, \mathscr{O}(A)$, $A \times B, A + B$ are if A and B are. $\mathbb{N}$ is countably infinite, as are lists over A if A is not empty and countable. $\mathbb{0}$ is empty.

Chapter 3

Preliminaries

3.1 Synthetic Undecidability Theory

Here, we introduce basic concepts of synthetic undecidability theory, following the approach of Forster et al. [5].

Usually, computability theory is defined by introducing an explicit notion of computation, usually the Turing machine. One can then prove the usual results, like the existence of an universal Turing machine or the fact that the halting problem is uncomputable.

In CIC, this approach is also possible. Since we want to have our proofs checked by a computer, this approach involves formally specifying every Turing machine (or equivalent concrete model of computation) used to show a function is computable, which involves significant work. A closer look at our meta-theory, however, reveals that every function definable in CIC is computable, since CIC is intuitionistic and hence admits models where every function is computable. The proof assistant Coq, which implements a syntactic model of CIC, may[1] be such a model, as it allows one to type-check and execute functions defined in CIC. The peculiarities of CIC's typing system even ensure that all functions defined in the internal language are total, i.e. they always halt.

Thus, since we are already implementing our proofs as programs, it is desirable to actually implement the programs needed for our (un)decidability proofs in the same programming language. When we do so, we know our program is actually computable, and also give a fully rigorous implementation. So, as our goal is to verify correctness for all programs used in (un)decidability proofs, this approach suffices. To define decidability, we first need to define problems.

A **problem** on A is a predicate of type $A \to \mathbb{P}$, for $A : \mathbb{T}$. For example, $prime : \mathbb{N} \to \mathbb{P}$, the predicate describing primality, is a problem on $\mathbb{N}$. By $\overline{P}$, we denote the

[1] this is suspected but not formally proven

complementary problem, defined by $\overline{P}\, a := \neg(P\, a)$. Note that $\overline{\overline{P}} \leftrightarrow P$ does not hold in general, since LEM does not hold. We call $a : A$ a **yes-instance** of some problem P on A if $P\, a$, and a **no-instance** if $\neg P\, a$.

Definition 3.1 (Decidability) *A proposition $p : \mathbb{P}$ is **decidable** if there exists a function $f : p + \neg p$. A problem $P : A \to \mathbb{P}$ on A is decidable if $P\, a$ is decidable for all a.*

Note that f takes no arguments. Since we used $p + \neg p$ and not $p \vee \neg p$, the resulting object is not computationally irrelevant and can be used in further computations, as we would expect from a program "deciding" certain properties. This in particular implies that, even if we take LEM as an axiom, our definition remains meaningful, since LEM is defined using the computationally irrelevant types. The type $p + \neg p$ can be understood as "either a proof of p, or a proof of $\neg p$", and a program can later inspect which of these alternatives is present. Equivalently, we could define f to merely output a boolean $b : \mathbb{B}$, and then require that $f = \mathtt{tt} \Leftrightarrow p$. However, the first definition is easier to work with. This definition allows us to define an additional important property of types:

Definition 3.2 (Discreteness) *A type $A : \mathbb{T}$ is **discrete** iff for all $x, y : A$, $x = y$ is decidable.*

Most types introduced so far are discrete, including $\mathbb{B}, \mathbb{N}$ and lists over discrete types.

To define undecidability, we consider how we usually show that a problem is undecidable: by reducing from another undecidable problem. Remember that when developing computability theory, one usually first shows that the halting problem is undecidable using a diagonal argument, and then reduces from it to show other problems undecidable. Specifying a reduction, in particular a many-one reduction, usually requires specifying a computable reduction function, and then showing that this function preserves yes- and no-instances. As explained, we want to write that function in our existing meta-language, rather than by defining a model of computation and specifying the function within it. Thus, we define many-one reducibility as follows:

Definition 3.3 (Many-one reduction) *A problem $P : A \to \mathbb{P}$ **many-one reduces** to a problem $Q : B \to \mathbb{P}$ iff there is a function $f : A \to B$ such that $\forall a : A. P\, a \leftrightarrow Q\, (f\, a)$.*

*Often, we show the equivalence by showing both directions separately. Then, we call the step $P\, a \to Q\, (f\, a)$ **preservation** and the opposite direction **reflection**.*

Armed with this definition, we can now verify reductions without reference to a constructed model of computation because we implicitly use our meta-theory as a

model of computation. However, we still need a suitable definition of undecidability. While the obvious definition "P is undecidable if it is not decidable" transports along our definition of many-one reducibility, it is unusable - many problems can not be shown undecidable.

For example, the statement "there is no decider for the halting problem for Turing machines" is independent of CIC – it holds in some models, while being invalid in others. In particular, while all functions definable in type theory are computable, a model might contain functions which are not definable in that internal language, and hence can be uncomputable. Thus, there are models containing "deciders" for the halting problem. As a consequence, we need a different definition. Before we can state this definition, we need to first define semi-decidability. A problem is semi-decidable if yes-instances can be found by an unbound search:

Definition 3.4 (Semi-Decidability) *A problem $P : A \to \mathbb{P}$ on A is **semi-decidable** iff there exists a function $f : \mathbb{N} \to A \to \mathbb{B}$ such that for all $a : A$, $P\,a \leftrightarrow \exists n : \mathbb{N}. f\,n\,a = \mathtt{tt}$. Such a function f is called a **semi-decider** for P. If $\overline{P}$ is semi-decidable, P is **co-semi-decidable**.*

Definition 3.5 (Undecidability) *A problem $P : A \to \mathbb{P}$ on A is **undecidable** iff, assuming it were decidable, the halting problem for Turing machines would be co-semi-decidable.*

If the semi-decidability of P implies the semi-decidability of the halting problem, P is non-semi-decidable.

This definition captures the usual meaning of undecidability, and in particular is preserved many-one reductions:

Lemma 3.6 (Many-one reduction soundness) *If $P : A \to \mathbb{P}$ is an undecidable problem on A, and $Q : B \to \mathbb{P}$ a problem on B, and P many-one reduces to Q, then Q is undecidable.*

We stress that the definition of undecidability deviates from the one originally used by Forster et al. [5]: There, a problem P is undecidable if decidability of P implies the decidability of the halting problem, whereas here, it suffices to imply the co-semi-decidability of the halting problem.

Using these basic notions, many problems can be shown undecidable. This thesis can be understood as part of a larger program, aiming to mechanize undecidability proofs for many problems. Many such proofs are part of the Coq Library of Undecidability Proofs [6]. This library already mechanizes reductions from the halting problem to several other problems from which it is easier to reduce, since they do not

involve arguing the formal semantics of a Turing machine. In particular, the library contains a mechanization of the undecidability of Diophantine equations [20].

These definitions have some noteworthy consequences which might seem surprising for a classical computability theorist: If a problem P is semi-decidable (witnessed by f) and co-semi-decidable (witnessed by $\overline{f}$), it is not necessarily decidable. This is precisely because our logic is constructive - for it to be decidable, we would have to give a terminating function. In a classical meta-theory, we can simply step-wise probe f and $\overline{f}$ until either returns `tt`, which must happen eventually since either $P\,a$ or $\overline{P}\,a$ must hold by LEM.[2]

To re-iterate, a proof that some problem P is undecidable (understood as by Definition 3.5) becomes meaningful only when we consider the meta-mathematics of intuitionistic CIC. Then, we know that a proof of the undecidability of P contains a function deriving a co-semi-decider for the halting problem from a decider for P. If we consider a model of type theory where all functions are executable, this function also is, and hence the halting problem is co-semi-decidable, which is a contradiction. This concrete approach is based on ideas from synthetic computability theory by Bauer [1] and was first used by Forster et al. [5]. It is the basis of the Coq Library of Undecidability Proofs [6].

3.2 First-Order Logic

First-order logic (FOL) is a formal logic. It has quantifiers, subject to an important restriction: They only quantify over the individuals of some "domain of discourse", and not predicates or functions on that domain.

To define FOL formally, we first need to define signatures, which is used to fix the admissible "primitive" formulas and terms.

Definition 3.7 (Signature) *A signature is a pair of types $(\mathcal{F}, \mathcal{P})$, along with two functions $|f|_{\mathcal{F}} : \mathbb{N}$ and $|p|_{\mathcal{P}} : \mathbb{N}$ for $f \in \mathcal{F}, p \in \mathcal{P}$, called the **arity** of f and p respectively. We omit the indices since they are clear from the context. We denote the type of all signatures as Σ.*

Example 3.8 (Signature of PA) *The signature of Peano arithmetic $\Sigma_{\mathsf{PA}} : \Sigma$ is the tuple $(\mathcal{F}_{\mathsf{PA}}, \mathcal{P}_{\mathsf{PA}})$, where $\mathcal{F}$ contains the elements $\dot{0}, \dot{S}, \dot{+}, \dot{*}$, and $\mathcal{P}$ contains the element $\dot{=}$. We have $|\dot{0}| = 0, |\dot{S}| = 1, |\dot{+}| = |\dot{*}| = |\dot{=}| = 2$.*

As mentioned, we use signatures to define the "atomic" symbols usable in FOL. The signature of PA, the first-order theory of natural numbers, thus contains the elementary symbols necessary to describe the natural numbers.

[2] actually, MP (a weaker axiom than LEM that is also consistent and independent) suffices.

We now define the actual terms and formulas of first-order logic. Terms can be understood as denoting a value (i.e. an element of the "domain of discourse"), while formulas denote a statement (i.e. something which may be "true"). In general, we assume a countably infinite, discrete type $\mathcal{V}$, the set of **variables**. Suitable examples are the type of finite lists of characters, i.e. strings, or the natural numbers.

Definition 3.9 (Syntax of FOL) *Given a signature $(\mathcal{F}, \mathcal{P}) : \Sigma$, we define the syntax of FOL terms and formulas. Terms are given by the following BNF, with $v : \mathcal{V}$*

$$\begin{aligned} e_1, e_2 : \mathit{Term} ::= & \; v \\ & \mid f(e_1, \ldots, e_n) \quad f : \mathcal{F}, |f| = n \end{aligned}$$

This allows us to define FOL*, the type of formulas:*

$$\begin{aligned} \varphi, \psi : \mathsf{FOL} ::= & \; \dot{\bot} \\ & \mid \varphi \mathbin{\dot{\wedge}} \psi \\ & \mid \varphi \mathbin{\dot{\vee}} \psi \\ & \mid \varphi \mathbin{\dot{\to}} \psi \\ & \mid \dot{\forall} x.\varphi \\ & \mid \dot{\exists} x.\varphi \\ & \mid p(e_1, \ldots, e_n) \quad p : \mathcal{P}, |p| = n \end{aligned}$$

We denote $\dot{\neg}\varphi$ as $\varphi \mathbin{\dot{\to}} \dot{\bot}$ and $\varphi \leftrightarrow \psi$ as $\varphi \to \psi \wedge \psi \to \varphi$. Additionally, we may denote $p(e_1, \ldots, e_n)$ and $f(e_1, \ldots, e_n)$ in infix notation if $n = 2$.

Formally, these definitions denote inductive types where each possible syntactic category is a constructor, taking the subformulas as arguments.

Example 3.10 (FOL formulas in PA) *$\dot{S}(\dot{S}(\dot{0}))$ is a term over PA, as is $\dot{S}\, x \mathbin{\dot{+}} (y \mathbin{\dot{*}} \dot{0})$.*

Formulas of FOL over PA include $x \doteq x, \dot{\forall} x.\dot{\forall} y.x \mathbin{\dot{+}} y \doteq y \mathbin{\dot{+}} x$, or $\dot{\forall} x.(x \doteq \dot{0} \mathbin{\dot{\vee}} \dot{\exists} y.x \doteq \dot{S}y)$.

Often, we omit the dot above a symbol when giving a FOL formula, as long as it is clear from the context that the formula should be understood as a FOL formula, instead of as a statement in the meta-theory.

Since our formulas feature variables and binders for them (namely $\dot{\forall}$ and $\dot{\exists}$), we need to define free and bound variables:

Definition 3.11 (Free variables) *A variable $v : \mathcal{V}$ is **free** in $\varphi :$ FOL if it occurs in φ and the occurrence is not bound. An occurrence of x is **bound** if the formula ψ x occurs in is part of $\dot{\forall} x.\psi$ or $\dot{\exists} x.\psi$. If it is bound, then it is **bound by** the innermost such quantifier.*

Example 3.12 *In $\dot{\forall} x.(x \doteq \dot{0} \mathrel{\dot{\vee}} \dot{\exists} y.x \doteq \dot{S}y)$, there are no free variables, since x is bound by the $\dot{\forall}$-quantifier and y is bound by the $\dot{\exists}$-quantifier. In $\dot{\forall} x.\dot{\forall} x.x \doteq y$, x is bound by the innermost quantifier, while y occurs freely.*

We consider two formulas α-equivalent if they are equal up to the naming of bound variables. For example, $\dot{\forall} x.\dot{\forall} y.x \doteq y$ is α-equivalent to $\dot{\forall} y.\dot{\forall} x.y \doteq x$, since we just consistently renamed x to y and vice-versa. They are not α-equivalent to $\dot{\forall} y.\dot{\forall} x.x \doteq y$, since the latter has a different "binding structure". α-renaming describes the process of changing variables names in an α-equivalence preserving way.

An alternative notion for working with variable names and α-equivalence are de Brujin indices: In de Brujin form, instead of using named binders and named variables, a variable is instead a number denoting the amount of quantifiers between it and its binding site. This notation is much harder to read, yet it makes mechanizing FOL much easier, and hence is used in the Coq formalization. We note that in de Brujin form, α-equivalent terms are equal.

Example 3.13 (de Brujin indexing) *The formula $\dot{\forall} x.\dot{\forall} y.x \doteq y$ is represented in de Brujin notation by $\dot{\forall}\dot{\forall}.1 \doteq 0$. Note that the quantifiers no longer explicitly bind variables: y is now written as 0 since the quantifier that used to bind y was nearest to the usage of y. x became 1, since the quantifier that used to bind x is the second-nearest.*

An important operation is **capture-free substitution**, denoted by $\varphi[e/x]$ for $\varphi : \mathsf{FOL}, e : \mathit{Term}, x : \mathcal{V}$. It replaces all free occurrences of x in φ by e, such that the free variables of e remain free at the substitution points. This may involve first α-renaming the bound variables in φ such that free variables of e are not captured. In de Brujin notation, no renaming is necessary, but indices in e might need to be adjusted.

Example 3.14 (Capture-free substitution) $(\dot{\forall} x.x \doteq y)[\dot{0}/y] = \dot{\forall} x.x \doteq \dot{0}$, *since we replace y by $\dot{0}$.* $(\dot{\forall} x.x \doteq y)[\dot{0}/x] = \dot{\forall} x.x \doteq y$, *however, since x does not occur freely. Importantly,* $(\dot{\forall} x.x \doteq y)[x/y] = \dot{\forall} x'.x' \doteq x$, *since the x substituted in must now occur freely. Hence, we must α-rename the bound x to something else (here x') to avoid capture.*

Chapter 4

Uniform Diophantine Pair Constraints

The ultimate goal of this thesis – showing the undecidability of various first-order problems in their minimal version in an efficient way – will be accomplished by reducing from a source problem which has several interesting properties. The source problem is a certain kind of Diophantine constraint problem, in which all constraints are defined using the relation $\wr$[1] on pairs of natural numbers. The specific properties of this relation are key to making all following undecidability proofs feasible.

Definition 4.1 ($\wr$) *The relation $\wr$ on $\mathbb{N}^2 \times \mathbb{N}^2$ is defined as*

$$(a, b) \wr (c, d) := a + b + 1 = c \wedge b^2 + b = d + d$$

This relation is easy to characterize axiomatically, and this will be possible almost without reference to $+$ or $\cdot$, by re-using the relation itself on "smaller" values. It is nontheless powerful enough to express all possible Diophantine constraints.

4.1 Defining Equations for $\wr$

To start, we can express the successor relation $x = y + 1$ using just 0 and $\wr$:

Lemma 4.2 (Base; Characterizing equation 1) $(a, 0) \wr (c, 0)$ *iff* $c = a + 1$.

The next characterization is of very high significance, since it captures most of "the essence" behind $\wr$ while only referring to $\wr$ itself:

Lemma 4.3 (Step; Characterizing equation 2)

$$(a, b) \wr (c, d) \wedge b \neq 0 \text{ iff } (a, b') \wr (c', d') \wedge (d', b') \wr (d, d') \wedge (b', 0) \wr (b, 0) \wedge (c', 0) \wr (c, 0)$$

[1] This relation was found by Andrej Dudenhefner while trying to find Diophantine constraints that allow for easy axiomatization in FOL or related logics. Dominik Kirst later helped apply it to FOL specifically.

Expressed this way, we can see that this equation defines $\wr$ using only itself in an inductive/recursive way. However, the property is better understood by rewriting with Lemma 4.2, and substituting b, c:

$$(a, b'+1)\wr(c'+1, d) \textit{ iff } (a, b')\wr(c', d') \wedge (d', b')\wr(d, d')$$

This characterization can be found by realizing that $\wr$ encodes a Gaussian sum in the pairs' right-hand components. In particular, if $(0, b)\wr(1, d)$, then $d = \sum_{i=1}^{b} i$. Thus, if we want to solve $(a, b'+1)\wr(c'+1, d)$ for d given $(a, b')\wr(c', d')$, we need to set $d := d' + b' + 1$, which is encoded as $(d', b')\wr(d, d')$ in Lemma 4.3.

Up until Chapter 8, we only use a slightly weaker version of Lemma 4.3:

Lemma 4.4 (Weak Step)

$$(a, b)\wr(c, d) \textit{ if } (a, b')\wr(c', d') \wedge (d', b')\wr(d, d') \wedge (b', 0)\wr(b, 0) \wedge (c', 0)\wr(c, 0)$$

A third characterizing equation is also only needed for the second part of this thesis.

Lemma 4.5 (Tieback; Characterizing equation 3) $(a, 0)\wr(c, d)$ *only if* $d = 0$.

The significance of Lemma 4.5 and the strong version of Lemma 4.3 is that without them, we would not be able to prove the following lemma. In fact, we could only show that $(a, b)\wr(c, d)$ implies $(a, b)\mathcal{R}(c, d)$, but not vice-versa. Since this weaker property is sufficient for the next few chapters, we defer until Section 8.2 for a detailed discussion of this.

Theorem 4.6 (Soundness and Completeness)
Any relation $\mathcal{R}$ on $\mathbb{N}^2 \times \mathbb{N}^2$ satisfying Lemmas 4.2, 4.3 and 4.5 is equivalent to $\wr$.

If $\mathcal{R}$ only satisfies Lemmas 4.2 and 4.4, then completeness holds, while soudness does not necessarily do so anymore: only $(a, b)\ \mathcal{R}\ (c, d) \leftarrow (a, b)\wr(c, d)$ can be proven.

We remark that $\wr$ can alternatively be defined as an inductive predicate:

Definition 4.7 (Inductive $\wr$)

$$\text{Base}\ \frac{}{(a, 0)\wr(a+1, 0)}$$

$$\text{Step}\ \frac{(a, b')\wr(c', d') \qquad (d', b')\wr(d, d') \qquad (b', 0)\wr(b, 0) \qquad (c', 0)\wr(c, 0)}{(a, b)\wr(c, d)}$$

Note that our constructors mirror the Lemmas 4.2 and 4.4. Yet, this relation satisfies all three characterizing equations. While this seems surprising at first, there is a deeper reason behind this: As explained in Chapter 3, an inductively defined

relation can be understood as the intersection of all relations satisfying the properties induced by the constructors – in our case, these are Lemmas 4.2 and 4.4. Hence, it is the "smallest" relation satisfying these properties as the constructors are **the only** methods for proving this relation. As it is defined using Lemmas 4.2 and 4.4, it is complete for ≀ (Theorem 4.6), and since it is the smallest relation satisfying these axioms, it must be at least as strong as ≀. Hence, the inductive definition of ≀ is equivalent[2]. To re-iterate, we were able to find a relation equivalent to ≀ by requiring that it satisfies the weak set of defining equations (Lemma 4.2, Lemma 4.4), and additionally requiring that it is the "smallest" such relation. This concept is crucial to the reductions outlined in the following chapters.

Lemma 4.8 *≀ is the smallest relation satisfying Lemmas 4.2 and 4.4.*

Thinking about our characterizing equations this way allows us to realize that they can be understood as constructors and destructors (in the sense that they allow us to "construct" derivation trees, as well as case-analyze them), allowing us to construct proofs of $(a, b) \wr (c, d)$ from other proofs of ≀ where a, b, c, d are smaller. In particular, the first characterizing equation requires c to be the immediate successor of a, and the second one has b', the immediate predecessor of b, used where previously b was used. This property allows one to easily use this relation in proofs by natural induction on the first or second argument of the left-hand pair. This "structurality" (along the left-hand pair) is another crucial property of ≀ and its axiomatization, albeit harder to pin down formally.

Another property that is helpful in the following chapters is the irreflexivity of ≀:

Lemma 4.9 $\forall p : \mathbb{N}^2.\neg(p \wr p)$

Havin understood ≀, we can use it to define the decision problem UDPC used as the reduction source in the remaining chapters of this thesis: The problem will be a list of equations in several variables, where all equations are defined as using ≀. If such an equation has a solution, it will be a yes-instance of UDPC. Formally, we define:

Definition 4.10 (Uniform Diophantine Pair Constraints)
*A **uniform Diophantine pair constraint** is a tuple $((x, y), (z, w))$, with x, y, z, w elements of some discrete, countably infinite set of variables $\mathcal{V}$. We say that an assignment $\rho : \mathcal{V} \to \mathbb{N}$ **satisfies** a uniform Diophantine constraint $((x, y), (z, w))$ iff $(\rho\, x, \rho\, y) \wr (\rho\, z, \rho\, w)$.*

*We call a list of such tuples an **uniform Diophantine pair constraint collection**.*

Problem 4.11 (UDPC) UDPC *is the following decision problem:*

$$\mathsf{UDPC}\,(h : \mathscr{L}\left(\mathcal{V}^2 \times \mathcal{V}^2\right)) := \exists \rho. \forall ((x, y), (z, w)) \in h. \rho \text{ satisfies } ((x, y), (z, w))$$

[2]This is merely an intuition, the actual proof is straightforward but boring.

As mentioned before, this problem is a Diophantine constraints problem, by which we mean problems asking for (the existence of) a solution to a list of equations over $\mathbb{N}$, with various requirements imposed on the shape of the equations. Another such problem is the following, where we use a slightly different relation as the basis of allowed equations.

Definition 4.12 (Uniform Diophantine Constraints)
*A **uniform Diophantine constraint** is a triple (x, y, z), with x, y, z elements of some discrete, countably infinite set of variables $\mathcal{V}$. We say that an assignment $\rho : \mathcal{V} \to \mathbb{N}$ **satisfies** a uniform Diophantine constraint (x, y, z) iff*

$$1 + \rho\, x + (\rho\, y)^2 = \rho\, z$$

*We call a list of such triples a **uniform Diophantine constraint collection**.*

Problem 4.13 (UDC) UDC *is the following decision problem:*

$$\mathsf{UDC}\,(l : \mathscr{L}\left(\mathcal{V}^3\right)) := \exists\rho.\forall(x, y, z) \in l.\rho \text{ satisfies } (x, y, z)$$

Theorem 4.14 UDC *is undecidable, semi-decidable and non-co-semi-decidable.*

Theorem 4.14 is already mechanized in the Coq Library of Undecidability Proofs [6][3], and serves as the starting point for all reductions in this thesis. In particular, we use it to show UDPC undecidable. Satisfiability of Diophantine constraints in general was first shown undecidable in 1970 [24], and this is already mechanized in Coq [20]. Since the undecidability of UDC was shown by reducing from general Diophantine equations, UDC can express all other Diophantine constraint problems.

4.2 Showing UDPC Undecidable

In order to show UDPC undecidable, we many-one reduce from our source problem UDC. For this, we need a computable reduction function F^{UDPC}:

$$\begin{aligned}
\mathit{code} &: \mathcal{V}^3 \to \mathscr{L}\left(\mathcal{V}'^2 \times \mathcal{V}'^2\right)\\
F^{\mathsf{UDPC}} &: \mathscr{L}\left(\mathcal{V}^3\right) \to \mathscr{L}\left(\mathcal{V}'^2 \times \mathcal{V}'^2\right)\\
\mathit{code}\,(x, y, z) &:= [((y_1, y_1), (y_2, y_4)), ((y_3, y_0), (y_2, y_1)), ((y_3, xx_0), (z_0, x_1))]\\
F^{\mathsf{UDPC}}(l) &:= \mathit{concat}\,[\mathit{code}\, p \,|\, p \in l]
\end{aligned}$$

The reduction function F^{UDPC} is computable, since we implemented it in our type-theortic meta-language. In the remainder of this chapter, we will simply use F instead of F^{UDPC}. F encodes a collection of uniform constraints by generating three new pair constraints. Those are created from a new collection of variables

[3] Andrej Dudenhefner also found it, and mechanized its undecidability.

$\mathcal{V}' := \mathcal{V} \times \{0, 1, 2, 3, 4\}$. This can be understood as creating a five new variables v_0, v_1, v_2, v_3, v_4 for each old variable v, where formally the new variable v_i denotes the pair (v, i). Since $\mathcal{V}$ and $\mathcal{V}'$ are in computable bijection, we can later translate back to the original $\mathcal{V}$. These new variables v_i are later assigned values solely depending on the value of v as follows:

i	v_i
0	v
1	$\frac{v^2+v}{2}$
2	$v^2 + v + 1$
3	v^2
4	$\frac{v_1^2+v_1}{2}$

We remark that $\frac{v^2+v}{2}$ is always a natural number, because $v^2 + v$ is always even. We now need to show that F fulfills the reduction properties, that is, for some h an instance of UDC, both $\mathsf{UDC}\, h \to \mathsf{UDPC}(F(l))$ and $\mathsf{UDPC}(F(l)) \to \mathsf{UDC}\, l$ must hold.

Lemma 4.15 (Preservation) $\mathsf{UDC}\, l \to \mathsf{UDPC}(F(l))$

Proof Here, we are given a solution ρ satisfying the constraint collection l. We construct a new solution ρ' assigning values to variables v_i according to ρ and the table above. Showing that this solution fulfills the pair constraints given by $F(l)$ is straightforward. □

Lemma 4.16 (Reflection) $\mathsf{UDPC}(F(l)) \to \mathsf{UDC}\, l$

Proof Here, we are given a solution ρ for the pair constraint collection $F(l)$, and we need to construct a solution ρ' for l. The actual solution is given as $\rho'\, v = \rho\, v_0$. It remains to show that this solution is valid. This follows by basic arithmetics after unfolding the definition of $\wr$ in the three pair constraints generated for each original uniform constraint. □

Lemma 4.17 UDC *is many-one reducible to* UDPC*.*

Proof F^{UDPC} is a reduction function by Lemma 4.15, Lemma 4.16. □

Theorem 4.18 UDPC *is undecidable.*

Proof By Lemma 4.17 and Theorem 4.14. □

We further remark that, as a consequence of Lemma 4.17, UDPC can express all Diophantine constraints, and hence is as expressive as UDC.

Chapter 5

Validity

Usually, when modeling a concept in first-order logic, one is guided by an already existing mathematical structure whose properties one aims to cast into formal first-order formulas. For example, Peano arithmetic is the first-order theory of natural numbers, i.e. it aims to allow reasoning about natural numbers in first-order logic. In this case, given a FOL formula in the signature of PA, we are able to directly interpret this as a statement about the natural numbers. Formally, we refer to the natural numbers as a model for PA. The precise definition of a model is as follows:

Definition 5.1 (Tarski Model, Satisfaction) *Given a signature $(\mathcal{F}, \mathcal{P}) : \Sigma$ of function and relation symbols and a type D, an **interpretation** I is a collection of functions and relations on D such that:*

- *For each $f : \mathcal{F}$, where f is a function symbol, there is a function $f_I : D^{|f|} \to D$, the **interpretation** of f.*
- *For each $p : \mathcal{P}$, where p is a relation symbol, there is a relation p_I on $D^{|f|}$, the **interpretation** of p.*

*A (**Tarski**) **model** $M = (D, I)$ is a pair consisting of a type D and an interpretation I for the symbols of a given signature $(\mathcal{F}, \mathcal{P}) : \Sigma$. Such a model (along with an environment ρ defining the free variables) **satisfies** some formula φ if $M \vDash_\rho \varphi$ with $\vDash$ defined inductively as follows:*

$$
\begin{aligned}
[\![\dot{x}]\!]_\rho &:= \rho\, \dot{x} & [\![f(e_1, \ldots, e_n)]\!]_\rho &:= f_i([\![e_1]\!]_\rho, \ldots, [\![e_n]\!]_\rho) \\
M \vDash_\rho p(e_1, \ldots, e_n) &:= p_i([\![e_1]\!]_\rho, \ldots, [\![e_n]\!]_\rho) & M \vDash_\rho \dot{\bot} &:= \bot \\
M \vDash_\rho \dot{\forall}\dot{x}.\varphi &:= \forall y : D. M \vDash_{\rho[y \mapsto \dot{x}]} \varphi & M \vDash_\rho \dot{\exists}\dot{x}.\varphi &:= \exists y : D. M \vDash_{\rho[y \mapsto \dot{x}]} \varphi \\
M \vDash_\rho \varphi \dot{\wedge} \psi &:= M \vDash_\rho \varphi \wedge M \vDash_\rho \psi & M \vDash_\rho \varphi \dot{\vee} \psi &:= M \vDash_\rho \varphi \vee M \vDash_\rho \psi \\
M \vDash_\rho \varphi \dot{\to} \psi &:= M \vDash_\rho \varphi \to M \vDash_\rho \psi
\end{aligned}
$$

To continue our example of PA and $\mathbb{N}$, we can formally define $\mathbb{N}$ as a model of PA:

Example 5.2 *Let $\Sigma_{PA} = (\{\dot{0}, \dot{S}, \dot{+}, \dot{\times}\}, \{\doteq\})$ be a signature. The interpretation I is defined in the obvious way, i.e. the interpretation of $\dot{+}$ is $+$ etc.*

Now, we can show that $\mathbb{N} \vDash_\rho \dot{\forall}\dot{x}\dot{y}.\dot{x} \dot{+} \dot{y} \doteq \dot{y} \dot{+} \dot{x}$, since unfolding $\vDash$ and $[\![\cdot]\!]$ transforms this into $\forall ab : \mathbb{N}.a + b = b + a$.

It should be noted that PA admits many models, and the one presented here is just one of them. In particular, by the Löwenheim-Skolen theorem [22], we can find models of every infinite cardinality. In a set-theoretic meta-theory, we even find models where we keep $\mathbb{N}$ as the domain of discourse, but have different interpretations[1]. Yet, we consider the above model "special" since it guides us in finding formulas and reasoning about truth in PA, which is not surprising, given that PA was created to formalize the natural numbers. Such special models are colloquially called "standard models". It should be noted that a standard model is just a particular model we as mathematicians working with FOL consider "intuitive", it is not necessarily special in a formal way. However, standard models are often characterized by some higher-order property: Here, $\mathbb{N}$ additionally is the smallest (computable) model of PA.

As noted, though, there usually are many models for a given theory, and it often is of interest whether a formula is not just satisfied by a certain special model, but by all possible models. This motivates the following decision problems:

Problem 5.3 (Validity, Satisfiability)
The decision problems $\mathsf{VAL}, \mathsf{SAT}$ *are decision problems on FOL formulas, defined as follows:*

$$\mathsf{VAL}\,\varphi := \forall M \rho.M \vDash_\rho \varphi$$
$$\mathsf{SAT}\,\varphi := \exists M \rho.M \vDash_\rho \varphi$$

Historically, VAL is known as the "Entscheidungsproblem" [13], and was among the first problems shown undecidable, a result independently obtained by Church and Turing in 1936 [2, 29], as mentioned in Chapter 1. In a classical meta-theory, VAL and $\overline{\mathsf{SAT}}$ as well as $\overline{\mathsf{VAL}}$ and SAT are many-one reducible to each other with the reduction function $F\,\varphi := \neg\varphi$, which then suffices to show that both are undecidable if either is. In our intuitionistic meta-theory, this does not hold – in fact, it requires LEM. Yet, our definition of undecidability based on the co-semi-decidability of the halting problem is strong enough for the previously mentioned result to still apply, but more canonical definitions of synthetic undecidability [1, 5] do not. However, we have to prove this by directly reducing from $\overline{\mathsf{UDPC}}$.

[1] In our meta-theory, this is not possible because these interpretations are uncomputable, and hence can not be defined.

With the general Entscheidungsproblem shown undecidable, special cases of it gained attention. Already in 1915 [22], it was proven that for a signature with only unary function and relation symbols, the Entscheidungsproblem was decidable. In 1937, Kalmár [16] completed a program of finding a syntax compression and reduction chain, by showing that an arbitrary FOL formula over an arbitrary signature can be transformed into an equivalent formula using only a single binary relation, and no functions. The result characterizes the decidable and undecidable fragments of VAL:

Theorem 5.4 ((Un-)decidability of VAL**)** *The problems* VAL *and* $\overline{\mathsf{SAT}}$ *for formulas of a fixed signature* $(\mathcal{F}, \mathcal{P}) : \Sigma$ *are*

1. *decidable, if* $\forall p \in \mathcal{P}.|p| \leq 1$ *and* $\forall f \in \mathcal{F}.|f| \leq 1$
2. *decidable, if* $\forall p \in \mathcal{P}.|p| = 0$
3. *undecidable, if* $\exists p \in \mathcal{P}.|p| \geq 2$
4. *undecidable, if* $\exists f \in \mathcal{F}.|f| \geq 2$ *and* $\exists p \in \mathcal{P}.|p| \geq 1$.

Proof

1. By Löwenheim [22].
2. FOL here degenerates into boolean[2] logic, which is trivially decidable.
3. By Kalmár [16]. It suffices to assume $|p| = 2$.
4. Similar to 3. □

It should be noted that these results hold for general formulas. There are always special cases (like $\bot$, which is a formula in any signature) which are trivially decidable, the undecidability results merely state that an algorithm does not exist in the general case.

We note that a signature with just a single binary relation is not just "small", but indeed **the** minimal relation for which VAL is undecidable: If there were fewer relations, FOL would degenerate into boolean logic and Theorem 5.4, case 2 would apply. Further, if the relations in the signature had lower arity, they would be unary and decidability would follow by Theorem 5.4, case 1.

In this chapter, we give a new proof of case 3, aiming at a short and easily understandable version, which does neither require prior model-theoretic work in ZF or PA, nor a sequence of syntax compression steps. For this, we reduce from UDPC,

[2] In particular, there are no atomic propositions besides $\bot$

since it allows an elegantly short axiomatization in FOL. The following chapters will yield even sharper versions of that undecidability result. However, since these are less approachable, we first present this version as a simpler variant.

The general undecidabilty of VAL is already mechanized in Coq [5], and case 3 of Theorem 5.4 was mechanized in [17] by transforming formulas of ZF set theory into a signature with a single binary predicate $\in$.

Hence, we now fix a concrete signature $(\emptyset, \{\wr\}) : \Sigma$, where $\wr$ is our single binary relation, and proceed to work within this signature unless explicitly mentioned. This includes that we, from now on, consider VAL and SAT and related problems to be defined on formulas over this signature.

5.1 Reducing from UDPC

For our reduction, we are given a constraint set $h : \mathcal{V}^2 \times \mathcal{V}^2$, and have to construct a formula $F^{\mathsf{VAL}}\, h$ such that $F^{\mathsf{VAL}}\, h$ is valid in all models if and only if h had a solution – formally, $\mathsf{UDPC}\, h \leftrightarrow \mathsf{VAL}\, (F^{\mathsf{VAL}}\, h)$. For this, we construct a first-order formalization of $\wr$, which later allows us to translate concrete constraints into FOL. We proceed by first constructing a standard model, which not only guides our understanding of the FOL formalization, but is also featured prominently in the reflection step of the reduction. While we heavily motivate the following definitions by interpreting them in the standard model, they nonetheless are to be understood as a general formalization of $\wr$ in FOL, which captures just enough properties of the standard model to replay the following construction in any model.

Our standard model is defined over the type $D := \mathbb{N} + \mathbb{N}^2$, that is, an element of the model is either a natural number x, or a pair (a, b). We define our interpretation I by defining the interpretation of $l \wr r$ as follows:

l \ r	$y : \mathbb{N}$	$(c, d) : \mathbb{N}^2$
$x : \mathbb{N}$	$x = y$	$x = c$
$(a, b) : \mathbb{N}^2$	$y = b$	$(a, b) \wr (c, d)$

The interpretation of $\wr$ on pairs is unsurprising, as we want to formalize $\wr$ in FOL. Furthermore, we need to be able to describe the component of pairs, motivating the interpretation of $(a, b) \wr y$ and $x \wr (c, d)$, which can be understood as projections. The interpretation of $x \wr y$ seems useful for constraining two elements of the model to be equal. However, we do not actually use this in the formula. The only case where both sides of $\wr$ are numbers is the case $x \wr x$, where both sides are already identical. Hence, many interpretations for $\wr$ are admissible as well, as long as $x \wr x$ holds for $x : \mathbb{N}$, i.e. $\wr$ is reflexive on such elements. This becomes relevant in later chapters, where we change this aspect of the interpretation to allow encoding various other

properties.

In order to formalize all properties of $\wr$, working on the level of $\wr\!\!\wr$ itself is infeasible. Instead, we define increasingly complex shorthands referring increasingly complex predicates while keeping our formulas readable. We start by defining N and P':

$$N\,k := k \wr\!\!\wr k$$
$$P'\,k := \neg N\,k$$

In our standard model, the predicate $N\,k$ constraints k to be a natural number, while $P'\,k$ constraints k to be the opposite, i.e. a pair. This is because equality $=$ is reflexive, while $\wr$ is not by Lemma 4.9. Here, in the definition of N, we find the aforementioned single use of $x \wr\!\!\wr y$ where x, y are both numbers, which as mentioned already has x identical to y since they must be the same term.

To continue with the construction of syntactic sugar, we need a predicate P which does not only constrain its argument to a pair, but to a specific pair with fixed components. We define:

$$P\,k\,l\,r := P'\,k \wedge N\,l \wedge N\,r \wedge l \wr\!\!\wr p \wedge p \wr\!\!\wr r$$

Again, it is easy to verify that $P\,k\,l\,r$ in our standard model is equivalent to $k = (l, r)$.

Given this, we can define a predicate R, which is used to model $\wr$ itself. While this might seem unnecessary as we have $\wr$ already in the interpretation of $\wr\!\!\wr$, we have no means of explicitly constructing a pair given four elements, hence requiring the following syntactic sugar:

$$R\,a\,b\,c\,d := \exists pq.P\,p\,a\,b \wedge P\,q\,c\,d \wedge p \wr\!\!\wr q$$

With this, we have all syntactic sugar necessary to build the axioms actually formalizing $\wr$. For this, it suffices to turn the characterizations behind Lemmas 4.2 and 4.4 into axioms. The ones defined in Lemmas 4.3 and 4.5 are not necessary, which becomes apparent once we have formalized the reduction. We restate Lemmas 4.2 and 4.4 here:

$(a, 0) \wr (c, 0)$ iff $c = a + 1$
$(a, b) \wr (c, d)$ if $(a, b') \wr (c', d') \wedge (d', b') \wr (d, d') \wedge (b', 0) \wr (b, 0) \wedge (c', 0) \wr (c, 0)$

To turn the first characterization into a FOL axiom, we need to realize that this characterization relates $\wr$ to the already existing linear order on natural numbers. In FOL, we would have to model this linearly increasing "chain" along with this

property. To do this, we create an axiom which allows us to find, given some k, a "successor" element k' for which $R\,k\,0\,k'\,0$. Thus we define Axiom 1 as

$$A_1^{\mathsf{VAL}} := \forall k.N\,k \to \exists k'.R\,k\,\dot{0}\,k'\,\dot{0}$$

This axiom may appear very similar to the "successor axiom" of Peano arithmetic, because it fulfills the same function, allowing us to construct a "chain" of elements in our model resembling the natural numbers. In our standard model, of course, these elements are the natural numbers themselves.

The attentive reader might realize that we introduced $\dot{0}$ here, which has not been previously defined. Usually, one would define $\dot{0}$ as a zero-argument function symbol, so that it can be used in formulas. This is not possible in our case since, in order to achieve the minimal result, we require a signature without function symbols. Instead, we use a trick: We proceed to treat $\dot{0}$ as if it was a function of arity 0. Later, when finalizing our reduction function, we make sure to add an outermost $\forall$-quantifier, binding $\dot{0}$. Since we are reducing from VAL, we are later allowed to instantiate an arbitrary element of the standard model, just as if $\dot{0}$ was a function of arity 0, since the formula would be valid for all interpretations of that symbol.

To continue constructing our axioms, we might consider whether we also need the other axioms of Peano arithmetic in order to complete our reduction function. It turns out that we only need a single further axiom of PA[3], namely the one stating that $\dot{0}$ is a number. This allows us to write down axiom 2. It should be noted that further characterization of $\dot{0}$ is not necessary. In particular, we do not need an axiom constraining $\dot{0}$ to be the smallest number, since this is not necessary for the axiomatization.

$$A_2^{\mathsf{VAL}} := N\,\dot{0}$$

Finally, we need to turn the second characterizing equation of $\wr$ into an axiom. This is straightforward:

$$A_3^{\mathsf{VAL}} := \forall abcdb'c'd'.R\,a\,b'\,c'\,d' \wedge R\,d'\,b'\,d\,d' \wedge R\,b'\,\dot{0}\,b\,\dot{0} \wedge R\,c'\,\dot{0}\,c\,\dot{0} \to R\,a\,b\,c\,d$$

These three axioms are sufficient to formalize $\wr$ to the degree necessary to construct

[3] In some presentations of PA, this axiom is implicit. Here, it needs to be explicit.

a reduction function:

$$\begin{aligned} F^{\mathsf{VAL}}, \mathit{code} : \mathscr{L}\left(\mathcal{V}^2 \times \mathcal{V}^2\right) &\to \mathsf{FOL} \\ F^{\mathsf{VAL}}\, h &:= \dot\forall \dot 0.A_1^{\mathsf{VAL}} \mathrel{\dot\to} A_2^{\mathsf{VAL}} \mathrel{\dot\to} A_3^{\mathsf{VAL}} \mathrel{\dot\to} \dot\bigexists_{v\in\mathcal{V}(h)} \mathit{code}\, h \\ \mathit{code}\, [] &:= \dot\top \\ \mathit{code}\, (((a,b),(c,d)) :: h) &:= R\, a\, b\, c\, d \mathrel{\dot\wedge} \mathit{code}\, h \end{aligned}$$

The reduction function has $\dot 0$ outermostly quantified, which, as mentioned before, allows us to use $\dot 0$ like a function of arity 0 in the remainder. We then add the axioms as guard conditions around the remainder of the reduction function. Then, we encode the constraints: Encoding a single constraint is done using R, and we require that all constraints hold at the same time, hence using $\dot\wedge$. The $\dot\exists_{v\in\mathcal{V}(h)}$-construction binds a FOL variable for each variable occuring in the constraint collection h. This is best understood by considering an example:

Example 5.5 *For $h = [((x,x),(y,x)),((x,y),(z,y))]$ a constraint collection, we have:*

$$F^{\mathsf{VAL}}\, h = \dot\forall \dot 0, A_1^{\mathsf{VAL}} \mathrel{\dot\to} A_2^{\mathsf{VAL}} \mathrel{\dot\to} A_3^{\mathsf{VAL}} \mathrel{\dot\to} \dot\exists xyz.R\, x\, x\, y\, x \mathrel{\dot\wedge} R\, x\, y\, z\, y \mathrel{\dot\wedge} \dot\top$$

Note that the existential quantifier binds x, y, z, which are the variables used in h.

To proceed with our reduction, we note that since F^{VAL} was implemented in the meta-language of our type theory, it is computable.

Now, we need to show that $\mathsf{VAL}\,(F^{\mathsf{VAL}}(h)) \leftrightarrow \mathsf{UDPC}\, h$. We start with the reflection step, since it is easier.

5.2 Reduction Reflection

Lemma 5.6 (Reflection) $\mathsf{VAL}\,(F^{\mathsf{VAL}}(h)) \to \mathsf{UDPC}\, h$

Proof For this reduction part, we are given a proof that $F^{\mathsf{VAL}}\, h$ is valid in all models, and have to find a solution ρ to our constraint problem h. Since $F^{\mathsf{VAL}}\, h$ is valid in all models, it is satisfied by the standard model for an arbitrary environment (which does not matter, since $F^{\mathsf{VAL}}\, h$ does not have free variables).

When interpreting $F^{\mathsf{VAL}}\, h$ in our standard model, we specialize $\dot 0$ with $0 : \mathbb{N}$, and since the axioms all hold in our standard model, we now know that $\dot\exists_{v\in\mathcal{V}(h)} \mathit{code}\, h$. Eliminating this existential quantifier yields a value for each variable, which we use to construct ρ, the solution to our constraint set. Showing that ρ is a constraint solution is straightforward by the definition of code, which states that the variables to which ρ now encodes a solution satisfy the constraints. Thus, we can conclude $\mathsf{UDPC}\, h$. □

5.3 Reduction Preservation

This reduction step is much more involved than the previous one. Before, we could simply use the proof of $\mathsf{VAL}\,(F^{\mathsf{VAL}}\,h)$ and work in a concrete model. Now, we have to work in an arbitrary model $M = (D, I)$ (along with an arbitrary environment, which we again ignore because $F^{\mathsf{VAL}}\,h$ does not have free variables), and show that $M \vDash F^{\mathsf{VAL}}\,h$ assuming h has a solution. We are thus given a solution ρ to h, as $\mathsf{UDPC}\,h$ is given, and we can further assume M satisfies our axioms, so that $M \vDash \exists_{v \in \mathcal{V}(h)}$ *code* h remains to be shown. We can assume that M satisfies our axioms because they guard the remainder of the reduction function: Since our model is a Tarski model, if we want to show $M \vDash \varphi \to \psi$, we can assume $M \vDash \varphi$ to derive $M \vDash \psi$. Hence we can assume $M \vDash A_i^{\mathsf{VAL}}$ for $1 \leq i \leq 3$.

This means that our model M satisfies the axioms. Apart from this, the model is arbitrary, and we cannot assume that it satisfies any further properties. Hence, we are now restricted to using only the axioms, and the fact that ρ is a solution to h. To begin this proof, we start by finding the elements in our model corresponding to natural numbers. For this, we define a data structure called "chain":

Definition 5.7 (Chain) *A **chain** up to $m : \mathbb{N}$ is a function $f : \mathbb{N} \to D$ such that*

- $f\,0 = \dot{0}$
- *For all $n < m$, we have $R\,(f\,n)\,\dot{0}\,(f\,(n+1))\,\dot{0}$.*

*If $fn = d$, then d **represents** n.*

The function f defining a chain yields the representation for numbers n in the model D. Since we construct it inductively, we only construct chains up to some upper bound m. This upper bound is later chosen as the highest number in ρ, the solution to h. The first property of a chain defines that the chain starts at $\dot{0}$, which can be understood as requiring that the $\dot{0}$ actually represents 0, which is necessary for the other axioms to "work properly". The second one requires the chain to be consistent with the ordering of the natural numbers: If $n + 1 = m$, then $f\,m$ must be the successor of $f\,n$ in the model.

Note that in the definition of a chain, we require $f\,0 = \dot{0}$, which is not well-formed, since $f\,0$ is an element of our model D, while $\dot{0}$ is part of the syntax of FOL terms. Formally, the statement should be $f\,0 = [\![\dot{0}]\!]$. However, since the remainder of this proof involves working within the model, we now start to omit $[\![\cdot]\!]$ and $\vDash$ when describing points in and properties of the model.

The defining properties of a chain suffice for showing that all the elements contained in a chain actually are numbers:

Lemma 5.8 *Given a chain f up to m and $n \leq m$, then $N\,(fn)$.*

Proof For 0, use A_2^{VAL}. Otherwise, by definition of R. □

In order to give a proof of $M \vDash \exists_{v \in \mathcal{V}(h)}$ *code* h, we need to first construct a chain:

Lemma 5.9 (Chain construction) *Given m, there exists a chain f up to m.*

Proof We proceed by induction on m:

- $m = 0$: The chain is given by $f\, n := \dot{0}$. This satisfies both properties, using Lemma 5.8.
- $m = m' + 1$, where f' is a chain up to m' by induction. We apply A_1^{VAL} to $f'\, m'$ and get d such that $R\,(f'\, m')\, 0\, d\, 0$. We can now define f:

$$f\, n := \begin{cases} d & n = m \\ f'\, n & \text{otw.} \end{cases}$$

  Chain property 1 is shown by induction, and property 2 also is, except for $n = m'$, where it is satisfied because A_1^{VAL} gave us d which already fulfills the required property. Lemma 5.8 is also needed. □

We now use Lemma 5.9 to build a chain f up to $\max_{v \in \mathcal{V}(h)} \rho v$. Then, we can proceed to prove $M \vDash \exists_{v \in \mathcal{V}(h)}$ *code* h by instantiating the existential quantifiers generated for each variable as follows: For a variable v, chose $f\,(\rho\, v)$[4]. The remaining goals are now of shape $R\,(f\,(\rho x))\,(f\,(\rho y))\,(f\,(\rho z))\,(f\,(\rho w))$, for all $((x, y), (z, w)) \in h$. Finally, this can be shown using another lemma:

Lemma 5.10 (Chain steps) *If $(a, b) \wr (c, d)$, $a, b, c, d < m$ and f is a chain up to m, then $M \vDash R\,(f\, a)\,(f\, b)\,(f\, c)\,(f\, d)$.*

Proof by induction on b, with a, c, d quantified:

- $b = 0$: Then $d = 0, c = a + 1$. Since $a < m$ and f is a chain, we are done by chain property 1 and 2.
- $b = b' + 1$. By Lemma 4.3, we have c', d' such that

$$(a, b') \wr (c', d') \wedge (d', b') \wr (d, d') \wedge (b', 0) \wr (b, 0) \wedge (c', 0) \wr (c, 0)$$

  By applying the IH, we get $R\, a\, b'\, c'\, d' \wedge R\, d'\, b'\, d\, d'$. Using an argument similar to that of the induction base case $b = 0$, we are able to further show $R\, b'\, \dot{0}\, b\, \dot{0} \wedge R\, c'\, \dot{0}\, c\, \dot{0}$. Thus we can conclude using axiom A_3^{VAL}. □

[4] Rememer that the variables bound by the existential quantifier are the same ones used for defining h, and hence are bound to values by ρ

With this lemma, we can conclude the complete proof of $M \vDash F^{\mathsf{VAL}}\, h$ for an arbitrary M.

Lemma 5.11 (Preservation) $\mathsf{UDPC}\, h \to \mathsf{VAL}\, (F^{\mathsf{VAL}}(h))$

Proof We have ρ, a solution to h, and the fact that our model fulfills the axioms A^{VAL}_{1-3}. Thus, we can build a chain f up to $\max_{v \in \mathcal{V}(h)} \rho v$ by Lemma 5.9. This chain allows us to give witnesses to the $\exists_{v \in \mathcal{V}(h)}$-construction, namely $f(\rho\, v)$ for given $v : \mathcal{V}$. We conclude by Lemma 5.10 for the remaining goals created by $code$. □

5.4 Conclusion

Our reduction is now complete, VAL is undecidable if the signature has just a single binary relation.

Lemma 5.12 UDPC *is many-one reducible to* VAL, *even when restricted to formulas with a single binary predicate.*

Proof F^{VAL} is a reduction function by Lemma 5.6, Lemma 5.11. □

Theorem 5.13 VAL *is undecidable, even when restricted to formulas with a single binary predicate.*

Proof By Lemma 5.12 and Theorem 4.18. □

Lemma 5.14 UDPC *is many-one reducible to* $\overline{\mathsf{SAT}}$, *even when restricted to formulas with a single binary predicate.*

Proof Using $F'\, \varphi := \neg(F^{\mathsf{VAL}}\, \varphi)$ as reduction function. □

Lemma 5.15 $\overline{\mathsf{SAT}}$ *is undecidable, even when restricted to formulas with a single binary predicate.*

Proof By Lemma 5.14 and Theorem 5.13. □

5.4.1 Remarks on The Models of $F^{\mathsf{VAL}}(h)$

We only formalized the characterizing equations Lemma 4.2, Lemma 4.4 of $\wr$ as first-order axioms in our model. Due to Theorem 4.6, any relation $\mathcal{R}$ satisfying these axioms can be shown complete but not sound – we can show that $(a, b)\mathcal{R}(c, d)$ must hold if $(a, b) \wr (c, d)$ while the other direction may not. In this reduction, this is sufficient, since for Lemma 4.16, we are given concrete solutions $(a, b) \wr (c, d)$, which we have to reflect into our model.

For Lemma 4.15, we had to show the inverse direction, which we did by specializing with the finite model. On a more abstract level, this works because we are reducing towards VAL: For a formula φ to be valid, it has to be satisfied by all models, and

hence also by a model where "the fewest" statements are true. While an arbitrary relation R satisfying Lemma 4.2, Lemma 4.4 can relate numbers not related by $\wr$, if we consider the intersection of all such relations, the resulting relation is equivalent to $\wr$ by Lemma 4.8. The problem VAL now requires a formula to hold in all models, which means that a formula is only valid if it is in the intersection of all sets of true statements for each model. Hence, R, our direct first-order translation of $\wr$ only needs to satisfy Lemmas 4.2 and 4.4 – since we are reducing towards VAL, we later insert a model where R is the smallest relation satisfying these axioms. This model is precisely the standard model, since there, R is interpreted as $\wr$. This implies that our standard model actually has a special property, namely that it is the "strongest" model.

If we just consider a single arbitrary model of $F^{\mathsf{VAL}}\ h$, we can see that $F^{\mathsf{VAL}}\ h$ contains elements corresponding to the natural numbers, as well as elements resembling their pairs. It may contain multiple elements representing a single number, spurious elements which behave unlike any pair or number, or numbers behaving similar to non-standard numbers known from the non-standard models of PA. Furthermore, as mentioned, the relation $R\,a\,b\,c\,d$, while true whenever a, b, c, d represent numbers from a single chain satisfying $\wr$, it may also be true for numbers for which $\wr$ is not. Indeed, we can easily imagine a model where $R\,a\,b\,c\,d$ is always true.

5.4.2 Remarks on The Coq Mechanization

In Coq, we model FOL with de Bruijn indices, and UDPC also uses indices as variable names, which make mechanizing $F\ h$ rather tedious since all formulas need to be given in de Brujin form. However, F^{VAL}, we can "re-use" the indices defining h when constructing $\exists_{v \in \mathcal{V}(h)}$, so that we just generate $1 + \max\ h$ existential quantifiers, which is much simpler than properly pairing FOL and UDPC variables. This re-use later requires subtle $\exists$-instantiation helpers, so that we can properly inject the correct values into the variables bound by $\exists$. Furthermore, we have to track the index of $\dot{0}$, which is outermostly quantified, so that it is referred to correctly. Furthermore, one has to re-state the notational shorthands as statements in M, something we glossed over here by simply re-using the notation. For instance, $N\ k$ as a FOL formula and $N\ k$ as a statement about $k : M$ are actually different, and this is explicit in Coq. One then writes several elimination helpers to ease transitioning from one to the other, which is straightforward but tedious.

We considered using the tools developed at this chair [14] in order to ease writing formulas with de Bruijn indices. However, this is infeasible for several reasons. First, our reduction formula "re-uses" the indices used to define h, and transitioning from them to named binders to de Bruijn indices makes mechanizing the reduction even more tedious not just because there is an additional translation step, but also since one has to be careful to not shadow identically named variables. Furthermore,

the mentioned tools are only designed for formulas with a static binding structure, whereas we have a variable number of $\forall$-quantifiers, parameterized by the indices in h.

Chapter 6

Minimizing The Logical Fragment

In the previous section, we showed VAL and SAT undecidable for a minimal version, where there is just a single binary predicate. However, there is more than one axis along which one can minimize a formula, and characterizing VAL or SAT into decidable and undecidable fragments along these axes is also highly interesting.

In this section, we proceed to analyze whether VAL and SAT are undecidable when minimizing the logical fragment, that is, the number of logical connectives we can use, in a way similar to how we minimized the number of atomic predicates required for FOL.

Until now, we defined FOL with falsity $\bot$, the binary connectives $\wedge, \vee$ and $\to$, and the two quantifiers $\forall$ and $\exists$. These connectives define the "full logical fragment". A particularly interesting restricted set of logical connectives is the following:

Definition 6.1 (Forall-implicative fragment) *If a formula uses only $\bot$, the logical connective $\to$ and the $\forall$-quantifier, as well as symbols from the signature, it is said to be within the $\forall, \to, \bot$-**fragment**.*

Definition 6.2 (Negation) *If a formula in the $\forall, \to, \bot$-fragment does additionally not contain $\bot$, it is said to be the $\forall, \to$-**fragment**. In general, a formula without $\bot$ is in a fragment **without negation**.*

Example 6.3 *The formula $\bot \to \bot$ is in the $\forall, \to, \bot$-fragment (with negation), while $\forall ab.a \between b \to b \between a$ is in the $\forall, \to$-fragment. The second formula is also in the $\forall, \to, \bot$-fragment, since the fragments only limit the number of usable symbols, without requiring that all usable symbols actually are used.*

In this section, we investigate whether VAL and $\overline{\text{SAT}}$ in the aforementioned minimal case are still undecidable when restricted to the $\forall, \to, \bot$-fragment, with or without negation.

6.1 Results in Classical Logic

6.1.1 Forall-Implicative Fragment

In classical logic, VAL and SAT are easily shown undecidable in the reduced logical fragment by performing a double negation translation. The translation $(\cdot)^N$, first used by Gödel [12] and Gentzen [10][1] is sometimes specified as follows:

Definition 6.4 (Double negation translation)

$$
\begin{aligned}
(\bot)^N &:= \bot & (p(e_1, \ldots, e_n))^N &:= \neg\neg p(e_1, \ldots, e_n) \\
(\varphi \wedge \psi)^N &:= \neg((\varphi)^N \to \neg(\psi)^N) & (\varphi \vee \psi)^N &:= \neg(\varphi)^N \to \neg\neg(\psi)^N \\
(\varphi \to \psi)^N &:= (\varphi)^N \to (\psi)^N & & \\
(\forall x.\varphi)^N &:= \forall x.(\varphi)^N & (\exists x.\varphi)^N &:= \neg(\forall x.\neg(\varphi)^N)
\end{aligned}
$$

We remind that $\neg\varphi$ is a shorthand as $\varphi \to \bot$, which is expressible in the $\forall, \to, \bot$-fragment.

Different authors use different versions of this translation, because they aim to reduce into differently defined fragments. This particular translation has the property that if φ were provable using a classical proof system, then $(\varphi)^N$ would be provable using an intuitionistic proof system. This result in particular has been also mechanized by Forster et al. [5]. Thus, if our meta-theory was classical, we could simply reduce from VAL over the full fragment to VAL over the reduced fragment, using $(\cdot)^N$ as our reduction function. However, since our meta-theory is intuitionistic, this does not work, because $(\varphi)^N$ does not necessarily imply φ, especially not if φ is undecidable. In order to work around, this we first have to explore ways to eliminate $\bot$.

6.2 $\bot$ Elimination

The Friedman A-Translation is another transformation of FOL formulas, which was originally used by Friedman [8] to prove certain conservativity properties between PA and Heyting arithmetic (i.e. intuitionistic PA). On a high level, one replaces $\bot$ with some formula B and adjoins this to all atomic predicates.

Definition 6.5 (Friedman translation)

$$
\begin{aligned}
(\bot)^B &:= B & (p(e_1, \ldots, e_n))^B &:= B \vee p(e_1, \ldots, e_n) \\
(\varphi \wedge \psi)^B &:= (\varphi)^B \wedge (\psi)^B & (\varphi \vee \psi)^B &:= (\varphi)^B \vee (\psi)^B \\
(\varphi \to \psi)^B &:= (\varphi)^B \to (\psi)^B & & \\
(\forall x.\varphi)^B &:= \forall x.(\varphi)^B & (\exists x.\varphi)^B &:= \exists x.(\varphi)^B
\end{aligned}
$$

[1] Gentzen's translation embeds into the $\forall, \wedge, \bot$-fragment.

It can be shown that $(\varphi)^{\psi}$ is equivalent to $\varphi \vee \psi$, as long as the atomic predicates fulfill certain properties (notably, they must be decidable), and as long as no free variables of ψ are captured.

6.2.1 Translations in Intuitionistic Logic

In classical logic, as well as in our intuitionistic meta-theory, we can quickly see that SAT for formulas without falsity is decidable, because any formula without $\bot$ is satisfied by a trivial model consisting of a single element $\star$, where all predicates are always interpreted as true.

As mentioned, we can easily translate our reduction function into the $\forall, \to, \bot$-fragment when working in a classical meta-theory. Eliminating $\bot$ becomes possible for VAL.

In our iniuitionistic logic, we can not apply these results directly. However, we can still modify our actual reduction formula in ways inspired by these two translations, in order to still transfer it into the $\forall, \to$-fragment.

6.3 Translating Our Reduction

We now proceed to adapt $F^{\mathsf{VAL}}\, h$ into the $\forall, \to, \bot$-fragment. For this, we will rewrite F^{VAL} such that the overall structure remains the same, while locally eliminating $\bot$ and unwanted logical connectives. We end up with the following new notational shorthands, which replace the old shorthands from the last chapter.

$$
\begin{aligned}
\bot_w &:= c_1 \between c_2 \\
\bot_s &:= \forall ab.a \between b \\
\neg_w \varphi &:= \varphi \to \bot_w \\
N\, k &:= k \between k \\
P'\, k &:= N\, k \to \bot_s \\
P\, k\, l\, r\, \psi &:= P'\, k \to N\, l \to N\, r \to l \between p \to p \between r \to \psi \\
R\, p\, q\, a\, b\, c\, d\, \psi &:= P\, p\, a\, b\, (P\, q\, c\, d\, (p \between q \to \psi))
\end{aligned}
$$

Mirroring the Friedman translation, we define a replacement for $\bot$. However, we actually define two replacements: $\bot_s$ and $\bot_w$. The reason is that $\bot_s$ is interpreted as $\bot$ (actual falsity) in our standard model. $\bot_w$, however, is not interpreted as falsity in the standard model, instead, it becomes an interpretation which allows us to escape a double-negated context, as becomes apparent shortly. This kind of translation was already used by Forster et al. ([5], see Definition 3.15) to reduce from classical to intuitionistic provability. It is likely that all usages of $\bot_s$ could be replaced by $\bot_w$, however, we do not do so since it only makes mechanizing the reduction harder.

Using $\bot_w$, we also define $\neg_w$ mirroring the original $\neg$. We use strong falsity to define P', because using strong falsity eases working with the statement P' in the standard

model, where we can actually show that $P'\,p$ is interpreted to mean that p is a pair.

For P, we add a another argument ψ. This is an artefact of an implicit double negation translation – previously, we usually had P appear as a condition of an implication. Because P was defined as the conjunction of several statements, we now build an implication chain which has all these statements as preconditions. This implication chain is attached to a consequence ψ, which is then implied by the preconditions. In total, our new statement $P\,n\,l\,r\varphi$ is equivalent (modulo translations) to the old statement $P\,n\,l\,r \to \varphi$.

A similar construction happens for R, which also has an additional argument φ used in the same way. Furthermore, R previously bound p, q using existential quantifiers. Now, we have to translate away existential quantifiers, so we remove them from R and require that p, q are aready appropiately quantified and merely passed into R. This allows us to eliminate existential quantifiers in preconditions by turning them into $\forall$-quantifiers. For example, a formula like $(\exists p, p \rrarr \dot{0}) \to \psi$ is equivalent to $\forall p, p \rrarr \dot{0} \to \psi$ (assuming p is not free in ψ), which is a formula in the $\forall, \to$-fragment. Because we want to avoid as many double negations as possible, we aim to use these "alternative" translations as often as possible.

$$
\begin{aligned}
A_1^{\mathsf{VAL}_\perp} &:= \forall k.N\,k \to \neg_w \forall pqk'.R\,p\,q\,k\,\dot{0}\,k'\,\dot{0}\,\perp_w \\
A_2^{\mathsf{VAL}_\perp} &:= N\,0 \\
A_3^{\mathsf{VAL}_\perp} &:= \forall abcdb'c'd'p_1 \ldots p_8.R\,p_1\,p_2\,a\,b'\,c'\,d' \\
&\qquad (R\,p_3\,p_4\,d'\,b'\,d\,d' \\
&\qquad (R\,p_5\,p_6\,b'\,0\,b\,0 \\
&\qquad (R\,p_7\,p_8\;\,c'\,0\,c\,0 \\
&\qquad (\neg_w \forall p_9 p_{10}.R\,p_9\,p_{10}\,a\,b\,c\,d\,\perp_w))))
\end{aligned}
$$

Here, we define the necessary axioms. Axiom 2 remains unchanged. Axiom 1 undergoes a slight change because it now has to quantify p, q used for R. Here, we can not avoid a double negation, as we have to translate an $\exists$-quantifier on the right-hand side of an implication. The changes for axiom 3 are similar, except that we now additionally need to quantify the 10 pairs which previously were implicitly quantified in R. For eight of them, we can avoid introducing a double negation. These ten pairs $p_1 \ldots p_{10}$ were previously quantified inside the old R. Now, the new R requires us to quantify them seperately, and to pass them to the R. This has the benefit of allowing us to reduce the amount of double negations required. Apart

from adapting to the old syntactic sugar, the axioms remain similar.

$$F^{\bot}, \mathit{code} : \mathscr{L}\left(\mathcal{V}^2 \times \mathcal{V}^2\right) \to \mathsf{FOL}$$
$$F^{\bot}\, h := \forall \dot{0}\, c_1\, c_2 . A_1^{\mathsf{VAL}_{\bot}} \to A_2^{\mathsf{VAL}_{\bot}} \to A_3^{\mathsf{VAL}_{\bot}} \to \neg_w \bigvee_{v \in \mathcal{V}(h)} \mathit{code}\, h$$
$$\mathit{code}\, [] := \bot_w$$
$$\mathit{code}\, (((a,b),(c,d)) :: h) := \neg_w(\forall p_1 p_2 . R\, p_1\, p_2\, a\, b\, c\, d\, \bot_w) \to \mathit{code}\, h$$

The reduction function on a large scale still works by prefixing the axioms to an $\exists_{v \in \mathcal{V}(h)}$-construction. Of course, we must now replace the $\exists$ with a $\forall$-quantifier over the same variables. When translating the constraints in h, we can again be clever and avoid additional double negations.

Apart from that, we have two new outermostly quantified constants, c_1 and c_2, which are used similarly to $\dot{0}$. They are used to define $\bot_w$, which is later interpreted as $\mathsf{UDPC}\, h$ in our standard model, allowing us to "escape" a double-negated context once, and is thus crucial to allow this proof to be valid in our intuitionistic meta-theory.

We now change the previous standard model, adjusting the interpretation of $x \wr y$ for $x, y : \mathbb{N}$:

l \ r	$y \in \mathbb{N}$	$(c,d) \in \mathbb{N}^2$
$x \in \mathbb{N}$	$x = y \vee (x = 0 \wedge y = 1 \wedge \underline{\mathsf{UDPC}\, h})$	$x = c$
$(a,b) \in \mathbb{N}^2$	$y = b$	$(a,b) \wr (c,d)$

By exploiting additional encoding space left undefined by $F^{\bot}\, h$, we are able to fit $\mathsf{UDPC}\, h$ into the model, where it becomes the interpretation of $\bot_w$, as mentioned. This underlined addition for $0 \wr 1$ is the only change.

In the actual reduction, we still need to show $\mathsf{VAL}\,(F^{\bot}(h)) \leftrightarrow \mathsf{UDPC}\, h$. The reflection step (Lemma 6.6) needs only slight changes, yet these are the most significant. The preservation step (Lemma 6.10) remains the same on the large scale. Of course, we must adapt to the introduced double negations. While this is easy in theory, it often leads to unintuitive, double-negated goals, making the proof hard to understand in practice.

6.3.1 Reduction Reflection

Lemma 6.6 (Reflection) $\mathsf{VAL}\,(F^{\bot}(h)) \to \mathsf{UDPC}\, h$

Proof Previously, to prove this step, we instantiated our standard model into the proof that $F\, h$ is valid in all models, the remaining extraction of a solution witnessing

UDPC h then was straightforward. Now, we are working with a double-negated, negation-eliminated formula. If we had not eliminated $\bot$ by replacing it with $\bot_w$, we would not be able to actually extract a solution from $\neg_w \forall_{v \in \mathcal{V}(h)}\, code\, h$, because we only have a proof that the non-existence of a solution is contradictory, which is not equivalent to an actual solution in intuitionistic logic, as UDPC h is not decidable. However, by now instantiating $c_1 := 0, c_1 := 1$, we have $(M \vDash \bot_w) \Leftrightarrow \mathsf{UDPC}\, h$, which is our goal in this situation. Thus, we are able to escape the double-negated context, and it remains to show that $\forall_{v \in \mathcal{V}(h)}\, code\, h$, which again is straightforward. □

The mentioned "escape" is very subtle, so we explicitly explain it now: At that point, we know that in our standard model, $\left(\forall_{v \in \mathcal{V}(h)}\, code\, h\right) \to \bot_w$. Since in our standard model, $\bot_w \Leftrightarrow \mathsf{UDPC}\, h$ holds by definition, we can apply this so that $\forall_{v \in \mathcal{V}(h)}\, code\, h$ remains to be shown. To prove this, we first introduce the $\forall$-quantified variables. Now $code\, h$ remains to be shown. Notice that this is just an implication chain where the consequent is $\bot_w$ again, and thus is equivalent to UDPC h. The antecedents simply require the variables previously bound by $\forall$ fulfill the constraints of h. Thus, we can recover a solution for h from this.

We strongly suggest the reader inspect the mechanized version of this proof for further details.

6.3.2 Reduction Preservation

The changes for this part of the reduction are minor – adjusting the proof to the double-negated statements is almost mechanical, yet still tedious. In particular, the intermediary goals one has to prove now are unintuitive, since the entire context is double-negated. We shortly state the the double-negated versions of the key lemmas from the previous reduction in Section 5.3:

Lemma 6.7 (Lemma 5.9 for the minimal fragment)
Let $m : \mathbb{N}$. To show $\bot_w$, it suffices to show $\forall f.(f$ is a chain up to $m) \to \bot_w$.

Lemma 6.8 (Lemma 5.10 for the minimal fragment)
If $(a, b) \wr (c, d)$, $a, b, c, d < m$ and f is a chain up to m, then showing $M \vDash \bot_w$ requires showing $M \vDash \forall pq.R\, p\, q\, (f\, a)\, (f\, b)\, (f\, c)\, (f\, d)\, \bot_w$

The proofs of these lemmas closely mirror the ones for their original versions, except that now significant parts of them work in double-negated contexts.

Furthermore, the utility lemma that previously allowed instantiating $\exists_{v \in \mathcal{V}(h)}\, \varphi$ is now changed to allow specializing $\forall_{v \in \mathcal{V}(h)}\, \varphi$, which now occurs as a hypothesis instead of inside the goal:

Lemma 6.9 ($\forall$ specialization)
If $\sigma : \mathcal{V} \to \mathcal{M}$ and $\mathcal{M} \vDash_\rho \bigforall_{v \in l} \varphi$ for some $l : \mathscr{L}(\mathcal{V})$, then $\mathcal{M} \vDash_{\rho'} \varphi$, where

$$\rho'\, v := \begin{cases} \sigma\, v & v \in l \\ \rho\, v & \textit{otw.} \end{cases}$$

In total, we are able to conclude that the new formula is still valid:

Lemma 6.10 (Preservation) $\mathsf{UDPC}\, h \to \mathsf{VAL}\, (F^\bot(h))$

6.4 Conclusion

In summary, we are able to additionally show that VAL is undecidable even if the formulas are required to be in the forall-implicative logical fragment without negation. Our old reduction function could be adapted using well-known translations. We thus get sharper versions of the lemmas proved in the previous chapters:

Lemma 6.11 UDPC *is many-one reducible to* VAL*, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Proof Similar to Lemma 5.12, except for the changes outlined in this chapter. □

Theorem 6.12 VAL *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Proof By Lemma 6.11 and Theorem 4.18. □

Lemma 6.13 UDPC *is many-one reducible to* $\overline{\mathsf{SAT}}$*, even when restricted to formulas with a single binary predicate over the forall-implicative logical fragment.*

Proof Using $F'\, \varphi := \neg(F\, \varphi)$ as reduction function. □

Lemma 6.14 $\overline{\mathsf{SAT}}$ *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative logical fragment.*

Proof By Lemma 6.13 and Theorem 6.12. □

We remark that the the reduction function for $F^\bot$ uses just a single $\bot$, which implies that SAT becomes undecidable as soon as the allowed formulas contain just a single $\bot$.

We note that these results are minimal along several axes. Due to Theorem 5.4, having a single binary predicate is minimal. Furthermore, the $\forall, \to$-fragment is also minimal (though not necessarily the minimum, since the $\forall, \wedge$-fragment might also be): If one further disallowed the use of the $\forall$ quantifier, the resulting formulas

would be formulas of predicate logic, which is decidable. Disallowing the use of the $\wedge$ logical connective would cause the resulting formula to just be a single predicate with an optional quantifier prefix, and thus validity would be trivial since it is never valid. Thus the $\forall, \rightarrow$-fragment is a minimal fragment for VAL to be undecidable. For SAT, as mentioned, the $\forall, \rightarrow, \bot$-fragment is necessary, for similar reasons.

A different minimization axis is the quantifier prefix. We do not consider minimizing the quantifier prefix, and suspect that the translation presented here actually enlarges the prefix, compared to the version of Chapter 5.

We further remark that while the translation performed here is suspected to hold in general, this fact can not be easily shown since it is part of the meta-theory of our mathematical foundation.

Note that the mechanized version of the outlined proofs instead uses results from the next chapter. Hence, some lemmas do not exist in the mechanization, or have a different proof.

Chapter 7

Provability

In the previous chapters, we proved VAL and $\overline{\mathsf{SAT}}$ undecidable. These problems are defined on models, where one interpretes the FOL formulas as statements about these models and then considers whether they are true according to the specific rules and definition of each model. The problem VAL arises naturally if one considers a formula "true" if it is valid in all models.

However, this definition of truth is somewhat problematic, because it defines truth by referring to the inner workings of various models. Historically, formal deduction systems were invented to define "truth" in a formal way, which does not refer to models at all, instead defining "truth" as a property of a given FOL formula. These proof systems define the structure of formal proofs by tightly specifying the allowed deduction rules. Furthermore, proof systems are finitistic, and thus allow expressing "truth" in comparatively weak meta-theories.

The proof system we are going to use for first-order logic is defined as an inductive predicate. There are multiple different formulations of proof systems, importantly Hilbert-style and natural deduction system. We use a natural deduction system, following Forster et al. [5], mostly because this is already mechanized in the library.

Definition 7.1 (Proof system) *We define $\vdash$, a relation relating lists of FOL formulas with FOL formulas. $A \vdash \varphi$, for $A : \mathscr{L}(\mathsf{FOL})$ a **list of hypotheses** and $\varphi : \mathsf{FOL}$ the **consequence**, denotes that A **proves** φ. $\vdash$ is defined as an inductive predicate, hence the smallest system admitting the following rules:*

$$\mathsf{Ctx}\ \frac{\varphi \in A}{A \vdash \varphi} \qquad \bot_{\mathsf{E}}\ \frac{A \vdash \bot}{A \vdash \varphi} \qquad \to_{\mathsf{E}}\ \frac{A \vdash \psi \qquad A \vdash \psi \to \varphi}{A \vdash \varphi}$$

$$\forall_{\mathsf{I}}\ \frac{A \vdash \varphi[y/x] \qquad y \text{ free in } A, \varphi}{A \vdash \forall x.\varphi} \qquad \forall_{\mathsf{E}}\ \frac{A \vdash \forall x.\varphi}{A \vdash \varphi[e/x]} \qquad \to_{\mathsf{I}}\ \frac{\psi :: A \vdash \varphi}{A \vdash \psi \to \psi}$$

Remember that $\varphi[e/x]$ is the capture-free substitution of e for x in φ.

The relation $\vdash_c$ additionally admits the following deduction rule:

$$\textsf{Peirce}\ \frac{}{A \vdash_c ((\varphi \to \psi) \to \varphi) \to \varphi}$$

*The proof system $\vdash_c$ is called **classical**. To disambiguate, $\vdash$ is sometimes referred to as **intuitionistic**. A formula φ is (**classically**) **provable** if $[] \vdash_{(c)} \varphi$.*

The deduction rules for an intuitionistic deduction system model those of our intuitionistic meta-theory. The additional deduction rule Peirce adds Peirce's law as an axiom to the deduction system, which is equivalent to LEM or double negation elimination. Note that the deduction system we gave is only useful for the $\forall, \to, \bot$-fragment. While the remaining deduction rules are mostly straightforward, we omit them here because they are not relevant to the later reduction.

To understand this proof system, it is best to consider an example.

Example 7.2 (Deduction for $(\forall xy.x \between y) \to \forall x.x \between x$))

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall xy.x \between y \in [(\forall xy.x \between y)]}\,\textsf{Ctx}}{[(\forall xy.x \between y)] \vdash \forall xy.x \between y}\,\forall_{\textsf{E}}}{[(\forall xy.x \between y)] \vdash \forall y.z \between y}\,\forall_{\textsf{E}}}{[(\forall xy.x \between y)] \vdash z \between z}\,\forall_{\textsf{I}}\quad z \text{ free in } \forall xy.x \wr y, x \wr x}{[(\forall xy.x \between y)] \vdash \forall x.x \between x}\,\to_{\textsf{I}}}{[] \vdash (\forall xy.x \between y) \to \forall x.x \between x}$$

For a proof system to be useful, we expect it to have certain properties. The most important one is **soundness**: The proof system should only prove "true" statements, which here means that all provable formulas should be valid. It turns out that our deduction system is sound:

Lemma 7.3 (Soundness of $\vdash$)
If $[] \vdash \varphi$, then $M \vDash_\rho \varphi$ for all M, ρ.

We note that $\vdash_c$ is not sound, since it would imply that LEM holds for all first-order expressible properties in M, which it does not, since our meta-theory is intuitionistic. It is sound for all models which "behave classically", i.e. all models where LEM holds, as would be the case if our meta-theory is classical. In particular, soundness of $\vdash_c$ is equivalent to LEM.

Besides soundness, proof systems typically have other basic properties like "weakening", which allow us deduce $B \vdash \varphi$ from $A \vdash \varphi$ if $B \supseteq A$. In general, this property and other related ones are often used during formal deductions in order to re-order or otherwise adjust the given hypothesis into a nicer representation. Other properties like cut elimination [9] allow converting proofs into normal forms, which can

be further analyzed to extract additional information. Such analysis is not necessary for this thesis.

Another key property of proof systems is **completeness**: The proof system should prove all "true" statements. However, not all proof systems are complete, and finding a complete proof system is often a significant result. For a classical meta-theory, $\vdash_c$ is complete, as first shown by Gödel:

Lemma 7.4 (Gödel's completeness theorem) *In a classical meta-theory, if $M \vDash_\rho \varphi$ for all M, ρ, then $[] \vdash_c \varphi$.*

Proof Due to Gödel [11]. Gödel's proof uses a Hilbert-style deduction system. □

In our intuitionistic meta-theory, this does not hold. Until now, we have only considered Tarski models, for which the intuitionistic deduction system can not be shown complete. Kripke models [19] give a semantics to FOL formulas for which intuitionistic completeness can be shown (in a classical meta-theory), based on an additional "reachability relation". Forster et al. [7] showed that the following properties hold in our meta-theory:

Lemma 7.5 (Completeness in intuitionistic meta-theory)
For φ a FOL formula, $\mathcal{M}_T$ a Tarski model and $\mathcal{M}_K$ a Kripke model, we have

$$\mathcal{M}_T \vDash \varphi \Rightarrow \neg\neg([] \vdash_c \varphi)$$
$$\mathcal{M}_K \vDash \varphi \Rightarrow \neg\neg([] \vdash \varphi)$$

Note that for $\mathcal{M}_K$, $\vDash$ denotes Kripke semantics.

We do not further define Kripke semantics here. We note that from every Tarski model, we can construct a Kripke model such that the formulas satisfied by that Kripke model under Kripke semantics are precisely the formulas satisfied by the original Tarski model. Also $\vdash$ is sound for Kripke semantics, while $\vdash_c$ is not (even when one assumes LEM). While we previously defined validity and satisfiability for Tarski models, we can adjust them to Kripke semantics, by defining the following decision problems:

Definition 7.6 (Kripke problems)

$$\mathsf{kVAL}\,\varphi := \forall \mathcal{M}_K \rho. \mathcal{M}_K \vDash_\rho \varphi$$
$$\mathsf{kSAT}\,\varphi := \exists \mathcal{M}_K \rho. \mathcal{M}_K \vDash_\rho \varphi$$

Besides these new decision problems based on Kripke semantics, the deduction system naturally defines its own problem:

Problem 7.7 (PRV) *The decision problem* PRV *is defined as follows:*

$$\mathsf{PRV}\,\varphi := [] \vdash \varphi$$

Additionally, $\mathsf{PRV}_c\,\varphi := [] \vdash_c \varphi$ *is defined analogous.*

In a classical meta-theory, PRV_c is trivially undecidable, since it is equivalent to VAL. Thus the same minimality considerations discussed in the previous chapters apply. In our intuitionistic meta-theory, this proof is not that simple, and we must instead perform a more complicated reduction in order to show PRV undecidable. We do this now, by reducing from UDPC.

In order to show PRV undecidable, we re-use significant parts of the VAL reduction defined in Lemma 6.11. In particular, the reduction function stays the same. So, it now remains to show that $\mathsf{PRV}\,(F^{\bot}\,h) \leftrightarrow \mathsf{UDPC}\,h$. We again start with the reflection step, since it is shorter.

7.1 Reduction Reflection

Lemma 7.8 (Reflection) $\mathsf{PRV}\,(F^{\bot}(h)) \to \mathsf{UDPC}\,h$.

Proof Immediate using Lemmas 6.6 and 7.3. □

7.2 Reduction Preservation

We want to prove $\mathsf{UDPC}\,h \to \mathsf{PRV}\,(F^{\bot}(h))$. The following proof is analogous to Lemma 6.10 on a large scale. On the small scale, however, there are significant differences.

To begin, we are no longer working within an arbitrary yet fixed model. Instead, we are constructing an abstract proof in our deduction system. To be precise, we have to derive $[] \vdash F\,h$. We then introduce our axioms, turning the statement into $A_1^{\mathsf{VAL}_{\bot}}, A_2^{\mathsf{VAL}_{\bot}}, A_3^{\mathsf{VAL}_{\bot}}, \forall_{v \in \mathcal{V}(h)}\, code\, h \vdash \bot_w$. In general, since we did a double-negation translation, most of the following proof formally derives $\bot_w$ from the given assumptions.

The particularities of our deduction system forbid us from using any higher-order constructs within our proof. In particular, we need to adjust our definition of a chain. Previously, it was defined as a function which yields elements of the model, for which we also required certain properties. Now, instead of elements, we would yield free variables. However, we can no longer "mix" the data and its properties: All properties these variables fulfill must be part of the list of hypotheses A in $A \vdash \varphi$. This forces us to define our chain in two parts:

Definition 7.9 (Chain for provability)
A ***proto-chain up to*** n *is a list* $\mathscr{L}(\mathcal{V}^3)$ *of length* k*. We canonically name the constituents*

of each pair (n, l, r)*. Every proto-chain* c *has a* ***head number*** $head\, c$*, which is* $\dot{0}$ *for* $[]$ *and* n *for* $[(n, l, r), \ldots]$*.*

The ***chain hypotheses*** $hyp\, k\, c : \mathscr{L}(\mathsf{FOL})$ *for a proto-chain* c *up to* k *are defined inductively:*

$$\begin{aligned} hyp\, 0\, [] :=& [N\, \dot{0}] \\ hyp\, (k' + 1)\, ((n, l, r) :: cr) :=& N\, n :: P'\, l :: P'\, r \\ & ::l \between (head\, cr) :: \dot{0} \between l \\ & ::r \between n :: \dot{0} \between r \\ & ::l \between r :: hyp\, k'\, cr \end{aligned}$$

For c *a proto-chain and* A *a list of hypotheses, if* $hyp\, c \subseteq A$*, we say that* c *is a* ***chain in*** A*.*

A variable v ***represents*** *a number* $m \leq k$ *if* $Q\, k\, c\, m = v$*, where* Q *is defined inductively:*

$$\begin{aligned} Q\, 0\, []\, 0 &:= \dot{0} \\ Q\, k\, ((n, l, r) :: c)\, k &:= n \\ Q\, k\, ((n, l, r) :: c)\, m &:= Q\, (k - 1)\, c\, m \end{aligned}$$

To recapitulate, our chain was used to find the elements of the model representing natural numbers. Now, we use it to find variables representing natural numbers instead. To encode that these variables actually represent natural numbers, we require that the chain hypotheses hold. For a chain $(n, l, r) :: cr$, the chain hypotheses generated for (n, l, r) exactly require that r is the pair $(n, \dot{0})$, l is the pair $(head\ cr)$ and that $l \between r$. This are precisely the conditions imposed by the chain for validity, defined in Definition 5.7. More formally, the chain hypotheses generated just for a single pair (n, l, p) are equivalent to the antecedents generated by $R\, l\, r\, (head\ cr)\, \dot{0}\, n\, \dot{0}\, \psi$.

Furthermore, the function Q looks up the variable representing a natural number in the chain. It searches from the back, i.e. the variable representing the number 1 is the furthest back (0 is always represented by $\dot{0}$). Defined this way, it is easier to prove the chain construction lemma, while the lookup function becomes a bit more tedious to define.

The reason we need a list of multiple hypotheses opposed to just a single one is that our original definition of a chain is not first-order expressible. Also, we have eliminated $\wedge$ from our fragment, so we can not use it in our deductive proof. Hence we need to keep all properties required for a chain as seperate entries in the list of hypotheses, making working with our chain rather tedious. We considered alternative representations of the chain, but found that they do not significantly alter the proof, since managing the chain hypotheses is responsible for most of the complexity.

This two-faced definition requires additional utility lemmas, for example, to show that if c is a chain in A, then any prefix of c also is a chain in A. These lemmas are tedious to prove since they require one specifying the right hypotheses in A, but since they are not particularly interesting, we do not give them here, and instead refer to the Coq mechanization.

While the definition of a chain has changed significantly, the semantic properties have not, so we still expect to prove a variation of Lemma 6.7 for provability:

Lemma 7.10 (Lemma 6.7 for PRV)
Let A be a list of hypotheses including $A_{1-3}^{\mathsf{VAL}_\perp}$. If c is a proto-chain up to k, and there is a proof of hyp $k\,c + A \vdash \perp_w$, then there is a proof of $A \vdash \perp_w$.

Proof Similar to Lemma 6.7. □

We can adapt Lemma 6.8 in a similar way:

Lemma 7.11 (Lemma 6.8 for PRV)
Let A be a list of hypotheses including $A_{1-3}^{\mathsf{VAL}_\perp}$. If $(a,b) \wr (c,d)$, $a,b,c,d < k$ and c is a chain up to k in A, then, to construct a proof $A \vdash \perp_w$, it suffices to construct a proof $(\neg\forall pq.R\,p\,q\,(Q\,k\,a)\,(Q\,k\,b)\,(Q\,k\,c)\,(Q\,k\,d)\,\perp_w) :: A \vdash \perp_w$.

Proof Similar to Lemma 6.8. □

Put simply, we can add the hypothesis that the representations of a, b, c, d are related by R to the already existing list of hypotheses when proving $\perp_w$. Then, the formula $(\neg\forall pq.R\,p\,q\,(Q\,k\,a)\,(Q\,k\,b)\,(Q\,k\,c)\,(Q\,k\,d)\,\perp_w)$ can be understood as holding in general in this double-negated context.

This completes the most significant part of the reduction. To continue, we again need several utility lemmas which allow us to manipulate the $\forall_{v\in\mathcal{V}(h)}$-construction, so that we can specialize all bound variables at once without having to explicitly track multiple freshness conditions. With these, we are then able to conclude our reduction, similar to Lemma 6.11.

Lemma 7.12 (Preservation) $\mathsf{UDPC}\,h \to \mathsf{PRV}\,(F^\perp(h))$.

Proof Using Lemmas 7.10 and 7.11, as outlined above. □

7.3 Conclusion

Lemma 7.13 UDPC *is many-one reducible to* PRV*, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Proof $F^\perp$ is a reduction function by Lemmas 7.8 and 7.12. □

Theorem 7.14 PRV *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Proof By Lemma 7.13 and Theorem 4.18. □

Theorem 7.15 *Assuming* LEM, PRV_c *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Proof $F^\perp$ is a reduction function by Lemmas 7.8 and 7.12 and classical soundness. Undecidability follows by Theorem 4.18. □

Note that we used Lemma 6.6 (validity reflection) to prove Lemma 7.8 (provability reflection), by using soundness and instantiating the standard model. Similarly, we can now give an alternate proof of Lemma 6.10 (validity preservation) using Lemma 7.12 (provability preservation) and soundness. This should not be too surprising, since we have performed the same construction, except within the abstract deduction system as opposed to within an arbitrary model. Even further, the abstract proof does not only re-prove the previous results for VAL and SAT, but also establishes similar results for Kripke semantics:

Lemma 7.16 UDPC *is many-one reducible to* kVAL, *even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation.*

Theorem 7.17 kVAL *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative fragment without negation..*

Lemma 7.18 $\overline{\mathsf{UDPC}}$ *is many-one reducible to* kSAT, *even when restricted to formulas with a single binary predicate over the forall-implicative fragment* (*with negation*).

Theorem 7.19 kSAT *is undecidable, even when restricted to formulas with a single binary predicate over the forall-implicative fragment* (*with negation*).

7.3.1 Remarks on The Coq Mechanization

Formalizing this reduction in Coq was the most tedious, compared to all other mechanizations in this thesis. As before, we are mechanizing another formal logic in Coq, which requires a very high level of formal precision and care. Now, we additionally model first-order provability as an inductive predicate, which further increases the level of formality required. In particular, compared to Coq's powerful system of notations and it's many-sorted, infinite-order meta-theory, working in the abstract deduction system is way less comfortable, since it lacks all of these features. In Coq, there are many "higher-order" tactics used to manipulate the current proof state. In the deduction system, we must explicitly use the low-level deduction rules, which makes certain operations very tedious: For example, to

specialize $\forall$-quantifiers in a hypothesis requires multiple uses of the $\to_{\mathsf{I},\mathsf{E}}$-laws, as well as weakening, in order to even start manipulating the assumptions.

Furthermore, while Coq terms are specified using named binders,[1] we have to again carefully track de Bruijn indices. In particular, working with the mechanized deduction rules defined in Definition 7.1 often involve index shifts, especially for the $\forall_{\mathsf{I}}$-rules. While there are additional lemmas which keep the amount of index shifts manageable, this still requires significant care.

In the Coq mechanization, we use a slightly different definition of proto-chains up to n. There, we model them as an indexed-inductive data type with n as the index argument. Working with this index-inductive data type requires the textbook inversion operators.

Recently, tools supposed to ease working within our deduction system [14] have been developed. However, these tools could not be used here, since they were not designed to work with "encapsulated" lists of hypotheses, as would be needed for $hyp\ k\ c$. The tools could be modified to support such constructions, and we suggest doing so, since such tools would have made working with the deduction system significantly easier.

[1] they use de Bruijn indices internally, but this is not visible to the user

Chapter 8

Finite Satisfiability

In Chapter 5, we defined the decision problem SAT, which asked whether there exists a (Tarski) model in which a given formula is satisfied, and showed it undecidable. Afterwards, we considered how one could sharpen the result by restricting the admissible formulas. In this chapter, we change the requirements for the model - notably, we restrict ourselves to finite models. This gives rise to the decision problem FSAT, or *finite satisfiability*:

Definition 8.1 (Finite Satisfiability in Classical Logic)
*A formula $\varphi : \mathsf{FOL}$ over a signature $(\mathcal{F}, \mathcal{P}) : \Sigma$ is **finitely satisfied** by a model $\mathcal{M} = (D, \mathcal{I})$ if D is finite[1] and there exists an environment ρ such that $\mathcal{M} \vDash_\rho \varphi$. It is **finitely satisfiable** if there exists an $\mathcal{M}$ finitely satisfying φ.*

Finite model theory is the branch of logic which concerns itself with such finite models and their application. A major result is the undecidability of FSAT, shown first by Trakhtenbrot in 1950[28]. While Trakhtenbrot only showed this for arbitrary formulas, it was already known [16] that this result can be reduced to the same minimal version, where there is only a single binary relation (and no function symbol). Trakhtenbrot's original proof of the general undecidability is by reducing from μ-recursive functions, constructing a formula finitely satisfiable iff the original function has a root via structural recursion on the definition of the original function. A more modern version of the proof is given in Libkin's textbook on Finite Model Theory [21], which also outlines applications of and reductions using FSAT as a source problem. Trakhtenbrot's result has been mechanized in Coq by Kirst and Larchey-Wendling [18], including the undecidability over the minimal signature, using a syntax compression chain.

In this chapter, we mechanize a reduction showing FSAT undecidable in the minimal case, by reducing from UDPC.

[1] assuming the classical definition of finite, i.e. $|D| \in \mathbb{N}$

8.1 Finite Model Theory in Constructive Meta-Theory

Since we work in a constructive meta-theory, we need to be careful when trying to capture the definition of FSAT. Importantly, in a classical, set-theoretic meta-theory we assume any finite model to be given "as a table": Since the domain is finite, we consider all atomic relations decidable and all atomic functions computable. As such, a finite model is completely determined by the number of elements in its domain, and the (truth) values of the various function and relation symbols applied to these elements. Furthermore, since the domain is finite, we can decide whether any first-order formula holds in a fixed model, since finite quantification is decidable. In order to capture these properties, it is not sufficient to merely pose that the type D representing the domain is finite, we must also require that any atomic relation is decidable.

This requirement might seem surprising, since we are trying to establish the undecidability of FSAT. However, this definition is crucial since otherwise we would not be able to compute with values from our finite models, which is necessary for this reduction. Furthermore, this property is desirable because we would otherwise be able to construct finite models with exhibit surprising behavior. Consider, for example, a variation of the *trivial model*:

Example 8.2 (Pathological trivial model) *Let $M := \mathbb{1}$, the type with exactly one element $\star$. To define the interpretation of $\wr$, it suffices to interpret $\star \wr \star$. We define $\star \wr \star := \forall P : \mathbb{P}, P \vee \neg P$, i.e. excluded middle.*

Without the decidability restriction for $\wr$, this model would be acceptable. This has several downsides - the first being that even in a fixed model, we are unable to prove either $\star \wr \star$ or $\neg(\star \wr \star)$, since XM is independent of our type theory – a highly unintuitive result for a finite model theorist. Furthermore, this model is a counterexample to the assumption that finite models are (finitely) enumerable: In particular, we would expect there to only be two trivial models, the one where $\star \wr \star$ is true, and the one where $\star \wr \star$ is false. This model, however, is a counter-example, and we are not even able to enumerate all models of size 1. Since such computation and enumeration arguments are a fundamental part of finite model theory, we are forced to add the decidability restriction to all atomic predicates, if we hope to establish even basic results from the literature.

Since we already implicitly consider all functions to be computable, we do not need to add an explicit function computability restriction. In summary, we adjust Definition 8.1:

Definition 8.3 (Finite Satisfiability) *A formula $\varphi : \mathsf{FOL}$ over a signature Σ is **finitely satisfied** by a model $\mathcal{M} = (D, \mathcal{I})$ if D is listable, all atomic relations in $\mathcal{I}$ are decidable and*

*there exists an environment ρ such that $\mathcal{M} \vDash_\rho \varphi$. It is **finitely satisfiable** if there exists an $\mathcal{M}$ finitely satisfying φ. Such an M is called a **finite model**.*

Using this definition, we can now prove that for a fixed finite model, satisfaction is decidable:

Lemma 8.4 (Satisfaction in fixed model) *Given M a finite model, $M \vDash_\rho \varphi$ is decidable for all ρ, φ.*

This gives rise to our decision problem FSAT:

Problem 8.5 (FSAT) *A decision problem on first-order formulas, defined as follows:*

$$\mathsf{FSAT}(\varphi) := \exists M. M \text{is a finite model for } \varphi$$

This definition is due to Kirst and Larchey-Wendling [18], who used it to mechanize the aforementioned undecidability results in Coq: They mechanized a reduction from Post's correspondence problem to show FSAT over an arbitrary signature undecidable. Afterwards, they mechanized a chain of signature compression steps to sharpen this result to the minimal version where the signature only contains a single binary relation. While this signature compression chain is important on its own, it introduces significant complexity when trying to understand the actual proof of the undecidability of FSAT in the minimal version. Our proposed reduction, however, immediately has already-minimal FSAT as the target problem. We thus hope to give a more feasible proof of this result.

Apart from these undecidability results, they also mechanized the decidability of finite satisfication in a fixed model, and the decidability of FSAT without binary relation symbols, as well as further basic theorems of Finite Model Theory.

8.2 Construction of the Reduction Function

As mentioned, our reduction takes UDPC as a source problem. However, the reduction itself is composed "inversely" compared to the previous reduction: Before, we constructed a formula, and had to show that it holds in an arbitrary model assuming the input constraint set had a solution. Now, we are given an arbitrary model satisfying our formula from which we have to extract a solution to our constraint set. This requires us to change the reduction function, and all of the axioms. By keeping our changes minimal, we are able to re-use most of the syntactic sugar.

While we need new axioms, we do not need to find new axioms from scratch. Careful consideration of the problem at hand shows that, when one understands the original axioms as "constructors" of the proof (in the sense of constructive logic), the kind of axioms we require now can be understood as "eliminators".

We use this duality, explicitly noted first by Kirst and Larchey-Wendling [18], to transform our previous axioms. The inductive characterization of $\wr$ (Definition 4.7) gives the required constructors. In summary, we need the full version of all three characterizing equations (Lemmas 4.2, 4.3 and 4.5).

The first axiom to consider is $\forall k.N\,k \to \exists k'.(k,0) \wr (k',0)$[2]. As a constructor, this axiom allows us to construct successors in an arbitrary model. The "inverse" of that axiom would be an axioms which allows us to find the predecessor of any number. However, such an axiom would be unsound, since 0 does not have a predecessor. Taking this into account, the actual transformation is $\forall k'.N\,k' \to k' \not\equiv \dot{0} \to \exists k.(k,\dot{0}) \wr (k',\dot{0})$. We proceed to abbreviate this as A_1^{FSAT}.

The attentive reader might notice $\equiv$, which is a previously undefined symbol. We define $a \equiv b$ to denote that a and b are *first-order indistinguishable*:

Definition 8.6 (First-Order Indistinguishability) *Given a model $\mathcal{M}$ of some first-order theory, two elements a, b of that model are first-order indistinguishable if for all first-order contexts $\varphi[\cdot]$, $\varphi[a] \leftrightarrow \varphi[b]$.*

While first-order indistinguishability is not immediately first-order expressible, it can be expressed in special cases like ours, because we only have a single relation symbol. (In fact, this generalizes to arbitrary finite signatures.) By letting $\equiv$ be syntactic sugar defined as follows, we can trivially show that a and b are first-order indistinguishable iff $a \equiv b$:

$$a \equiv b := \forall k.a \wr k \leftrightarrow b \wr k \land k \wr a \leftrightarrow k \wr b$$

We note that $\equiv$ is an equivalence relation and decidable for a fixed model, since finite quantification is decidable. We proceed to treat it as "the" equality in a given model, since actual equality between elements in a model is too fine to be meaningful, as two different elements might have the same extensional behavior. We could make this explicit by, from now on, only considering the quotient of our model modulo $\equiv$, as has been done by Kirst and Larchey-Wendlin [18], but we choose not to, since this would add significant overhead to the mechanization.

With this definition out of the way, we are able to transform the other axiom, ending up with this axiom, which we are from now on abbreviating as A_2^{FSAT}:

$$\begin{aligned}&\forall abcd.(a,b) \wr (c,d) \to b \not\equiv \dot{0} \to \\ &\exists b'c'd'.(b',\dot{0}) \wr (b,\dot{0}) \land (c',\dot{0}) \wr (c,\dot{0}) \land (a,b') \wr (c',d') \land (d',b') \wr (d,d') \land d' < d\end{aligned}$$

The attentive reader might again notice $<$, which we explain shortly. Apart from this, this axiom allows us to deconstruct $(a,b) \wr (c,d)$ into four new pair relations,

[2] remember $(a,0) \wr (b,0)$ denotes $b = a + 1$

mirroring the previous axiom which allowed us to show $(a, b) \wr (c, d)$ by first showing these four pair relations. Again, we had to introduce the guard condition $b \not\equiv \dot{0}$. This, however, made our axiom sublty weaker than the original one we tried to invert. Previously, when constructing an arbitrary relation $(a, b) \wr (c, d)$, we used the relations constructed by the first axiom (of shape $(a, \dot{0}) \wr (a', \dot{0})$). Using our new axiom, we want to destruct any arbitrary relation until we eventually reach this shape. However, just using the axiom shown above, we would only be able to eventually derive relations of shape $(a, \dot{0}) \wr (a', d)$, where $d \equiv \dot{0}$ is not necessarily true. The solution is adding characterizing equation three ("tieback", Lemma 4.5), which is encoded as the new axiom A_3^{FSAT}:

$$\forall aa'd.(a, \dot{0}) \wr (a', d) \to d \equiv \dot{0}$$

We have now successfully transformed all axioms necessary to describe $\wr$. Yet, we still require some more axioms: In the previous reduction, we used induction on natural numbers to find their representative in the model. Now, we have to do induction on elements of the model to find the natural number they represent. While before, this was just plain natural induction, we now have to do induction on elements of the model, since we do not yet know which number a given element of the model represents. Since the model is arbitrary, we must find a well-founded relation on the elements of the model to do this induction. This seems impossible at first glance, because well-foundedness is not first-order expressible. However, using a well-known fact about relations on finite types, already mechanized by e.g. Kirst and Larchey-Wendling [18], we can induce a well-founded relation into our model:

Fact 8.7 (Well-founded relations for finite types)
Let D be a listable type and $\prec$ be a transitive, irreflexive relation on D. Then $\prec$ is well-founded.

Since our induction mirrors complete (natural) induction, we have to find a relation $<$ (mirroring $<_{\mathbb{N}}$) to be the transitive closure of the successor relation. Again, we work around the fact that closure operations are not first-order expressible, by instead defining $<$ seperately, and relating it to the successor relation using an additional axiom. We thus define the following syntactic sugar and axioms:

$$\begin{aligned}
a \leq b &:= N\, a \wedge N\, b \wedge a \wr b \\
a < b &:= a \leq b \wedge a \not\equiv b \\
A_4^{\mathsf{FSAT}} &:= \forall lr.(l, \dot{0}) \wr (r, \dot{0}) \to l < r \wedge \forall k.k < l \to k \leq r \\
A_5^{\mathsf{FSAT}} &:= \forall abc.a < b \to b < c \to a < c
\end{aligned}$$

We note that this is sufficient to show $<$ is transitive and irreflexive. Axiom 4 makes $<$ a closure of the successor relation, while also requiring that successive numbers are

"close" – for any a, there is no number between a and its successor. This axiom also strengthens the successor relation in subtle ways, which we discuss in Section 8.5.1

These five axioms are sufficient to close the reduction. This allows us to now write down the actual reduction function F:

$$
\begin{aligned}
&F^{\mathsf{FSAT}}, \mathit{code} : \mathscr{L}(\mathcal{V}^2 \times \mathcal{V}^2) \to \mathsf{FOL} \\
&F^{\mathsf{FSAT}}\, h := \exists \dot{0} m. A_1^{\mathsf{FSAT}} \wedge A_2^{\mathsf{FSAT}} \wedge A_3^{\mathsf{FSAT}} \wedge A_4^{\mathsf{FSAT}} \wedge A_5^{\mathsf{FSAT}} \wedge \bigdoublebarwedge_{v \in \mathcal{V}(h)} \mathit{code}\, h \\
&\quad \mathit{code}\, [] := \top \\
&\quad \mathit{code}\, (((a,b),(c,d)) :: hs) := (a,b) \wr (c,d) \wedge \mathit{code}\, hs \wedge a,b,c,d \leq m
\end{aligned}
$$

This function is computable, since we have implemented it in Coq. It is defined analogous to our previous reduction function, by posing $\dot{0}$ as a constant and requiring that the axioms are satisfied. We need an additional constant, m, bounding the variables representing the ones from the constraint problem from above. Notice that in the prefix, we turned $\forall$ into $\exists$ and $\to$ into $\wedge$, which is necessary since we are reducing to (finite) satisfiability, not validity.

We proceed to now show that this function defines a reduction by first showing that $\mathsf{UDPC}\, h \to \mathsf{FSAT}\, (F^{\mathsf{FSAT}}(h))$ in Section 8.3, then that $\mathsf{FSAT}\, (F^{\mathsf{FSAT}}(h)) \to \mathsf{UDPC}\, h$ in Section 8.4.

8.3 Reduction Preservation

For this step, we are given a solution ρ to the constraint problem h, i.e. an assignment $\mathcal{V} \to \mathcal{N}$ assigning each variable in h a value such that all the constraints encoded by h hold. From this, we need to show that $F(h)$ is finitely satisfiable, which we do by constructing a finite model. The actual finite model is a prefix of the standard model we constructed previously, except for one change in the interpretation of $\wr$.

Given the solution ρ, let $m := 1 + \max_{v \in \mathcal{V}(h)} \rho(v)$. We denote by $\mathbb{N}_{\leq m}$ the type of natural numbers up to (and including) m. We set our domain $D := \mathbb{N}_{\leq m} + \mathbb{N}^2_{\leq m}$, so that each element of the domain is either a natural number not larger than m, or the pair of two such numbers. The interpretation of $l \wr r$, which is then allows us to define our complete model $\mathcal{M} = (D, I)$, is given as follows:

l \ r	$y \in \mathbb{N}_{\leq m}$	$(c,d) \in \mathbb{N}^2_{\leq m}$
$x \in \mathbb{N}_{\leq m}$	$x \leq y$	$x = c$
$(a,b) \in \mathbb{N}^2_{\leq m}$	$y = b$	$(a,b) \wr (c,d)$

Recall the definition of $N\, m := m \wr m$. While before, we had the interpretation of $\wr$ for two numbers be $x = y$, so the change to $x \leq y$ does not impact the definition of

$N\,m$, because both are reflexive. The change is nontheless significant, since in the previous formula, the only occurance of $a \between b$ with a, b numbers, was in N, where $a = b$. However, since we now encode $\leq$ using $\between$, there are actually cases where e.g. $0 \between 1$ can occur. Before, this was impossible by construction of $F^{\bot}$, and gave us additional encoding space for the Friedman translation, where we chose $0 \between 1$ as the translation for $\bot$, as we were able to give it special treatment in our model. This encoding method is now no longer possible.

We now need to show that this model actually is a finite model. It is finite, since $|D| = (m+1) + (m+1)^2$. Furthermore, $l \between r$ is decidable for fixed l, r since $\mathbb{N}$ is a discrete type (allowing us to also decide $\wr$), and since $l \leq_{\mathbb{N}} r$ is also trivially decidable.

It now remains to show that our finite model satisfies $F^{\mathsf{FSAT}}(h)$. To show that the axioms hold is mostly straightforward, once one has shown that $a \equiv b \Leftrightarrow a = b$ in our model. The latter fact seems straightforward, but the proof is rather subtle, especially if either side of the $\between$ is a pair and the other is a number.

Showing that the $\exists$-construction is satisfied also is straightforward, because we choose $\rho(v)$ for each v. Each $\rho(v)$ is in $\mathbb{N}_{\leq m}$, since it is smaller than m by construction. Thus we have shown that $F^{\mathsf{FSAT}}(h)$ is finitely satisfied, concluding the simpler half of the reduction:

Lemma 8.8 (Preservation) $\mathsf{UDPC}\, h \to \mathsf{FSAT}\,(F^{\mathsf{FSAT}}(h))$.

Proof As outlined above. □

8.4 Reduction Reflection

In this part, we have to show that the constraints encoded by h have a solution. We are given an arbitrary, finite model $\mathcal{M} = (D, I)$ satisfying $F(h)$. By unfolding $F(h)$, we know that this model satisfies the axioms. Now, we want to extract a solution to h from this model. Before we can do this, we first have to prove a handful of utility lemmas. Afterwards, we introduce the concept of a *chain* and use it to extract a solution to the constraint set.

To start, we first need to show some auxiliary lemmas. In particular, we are able to show that $a < b \leq c \Rightarrow a < c$, as well as $a \leq b < c \Rightarrow a < c$, by case analysis on $b \equiv a$ (or c respectively). By Fact 8.7, y is well-founded. Using well-founded induction along $<$, we can show $\forall k.\neg(k < \dot{0})$ and $\forall k.k \geq \dot{0}$, which seem equivalent, but are not necessarily, since we have not been able to actually derive totality for $\leq$. Totality for $\leq$ is not necessary to formalize this reduction, and we discuss the model-theoretic implications of this in Section 8.5.1. We are able to show $\leq$ antisymmetric, though. Additionally, by the definition of $N\,k$, we have $N\,k \Rightarrow k \leq k$.

8.4.1 Chains

In order to extract a solution to our constraint set h, we first need to extract the number corresponding to a certain element of our model. To do this, we build a data structure called *chain*, a mirror version of the identically-named construction we defined in the previous reduction (compare Definition 7.9 or Definition 5.7), by iteratively applying the predecessor axiom A_1^{FSAT} until we reach 0. The chain has all properties necessary to extract numbers representing elements of D.

Definition 8.9 (Chain)
*A function $f : D \to \mathscr{O}(\mathbb{N})$ is called a **chain up to** $m : D$ **representing** $n : \mathbb{N}$ iff*

1. $\forall d : D.d \leq m \Leftrightarrow f\, m \neq \emptyset$
2. $\forall k : \mathbb{N}.\, (\exists d : D.d \leq m \wedge f\, d = \lceil n \rceil) \Rightarrow k \leq n$
3. $f\, m = [n]$
4. $f\, \left(\dot{0}\right) = \lceil 0 \rceil$
5. $\forall (d_l\, d_r : D)(k_l\, k_r : \mathbb{N}).f\, d_l = \lceil k_l \rceil \wedge f\, d_r = \lceil k_r \rceil \Rightarrow \left(S\, k_l = k_r \Leftrightarrow (d_l, \dot{0}) \wr (d_r, \dot{0})\right)$
6. $\forall d d'\, f.d \neq \emptyset \Rightarrow (f\, d = f\, d' \Leftrightarrow d \equiv d')$

*We call m and n the **upper bound of** f, and say that some m' **represents** some n' iff $f\, m' = [n']$.*

The chain encodes a number for some elements $m : D$. Since not all elements are the representation of some number, f can also yield $\emptyset$. The properties can be understood as follows:

1. This property requires a chain up to m to actually yield a number for every $m' \leq m$. Additionally, it poses that the chain only yields numbers for such inputs.
2. The chain only yields numbers less than its upper bound.
3. This property just encodes that m actually represents n.
4. This property can be thought of as the starting condition for building the chain. Initially, we want $\dot{0}$ to be represented by $0 : \mathbb{N}$.
5. This property is crucial since it encodes that the numbers represented by the chain "are canonical". In particular, the elements of D representing two numbers are successors if and only if the represented numbers themselves are. Remember that $S\, k_l = k_r$ just means $k_l + 1 = k_r$.
6. The $\Leftarrow$-direction just poses that f is injective (where not $\emptyset$), so that every number has a unique representative (up to $\equiv$).

The $\Rightarrow$-direction is necessary since we want to avoid constructing an explicit quotient modulo $\equiv$, and poses that f does not distinguish first-order indistinguishable inputs.

Before we can work with such a chain, we need to construct one:

Lemma 8.10 (Chain construction)
Given $m : D$ such that $N\, m$, there exists n, f such that f is a chain up to m representing n.

Proof by well-founded induction using $<$ on m. We have two cases:

- $m \equiv \dot{0}$. In this case, we pose f as follows. The chain properties are easily shown in this case.

$$f k = \begin{cases} \lceil 0 \rceil & k \equiv \dot{0} \\ \emptyset & \text{otherwise} \end{cases}$$

- $m \not\equiv \dot{0}$. We know m has a predecessor m' by A_1^{FSAT}. Applying the induction hypothesis to m' yields f', a chain representing m' up to n'. Our new chain f, given as follows, is up to m representing $n' + 1$.

$$f k = \begin{cases} \lceil n' + 1 \rceil & k \equiv m \\ f'\, k & \text{otherwise} \end{cases}$$

  Proving that f satisfies properties 1-6 is mostly straightforward, although it involves a lot of case distinctions. Doing so makes crucial use the fact that f' is a chain, as well as the auxiliary lemmas mentioned before. □

Note that we could easily construct a chain having just properties 1,2,3, or 4, as these are proven independently. Showing property 6 needs property 1 and 2, while showing property 5 needs 1, 2, 3 and 6. Properties 1, 4, 5 and 6 are also needed later on.

8.4.2 Using Chains

Once we are able to construct a chain, we are able to use this chain to extract solutions to the constraints encoded in h.

Lemma 8.11 (Constraint recovery)
If f is a chain up to m representing n, and if $(a, b) \wr (c, d)$ where a, b, c, d are all $\leq m$, then we find a_r, b_r, c_r, d_r such that $f\, a = \lceil a_r \rceil, f\, b = \lceil b_r \rceil, f\, c = \lceil c_r \rceil, f\, d = \lceil d_r \rceil$ and $(a_r, b_r) \wr (c_r, d_r)$. In other words, $\wr$ on D transports to $\wr$ on $\mathbb{N}$ for the represented numbers.

Proof by well-foundedness of $<$ on b, with a, c, d quantified.
We note that $f\, a, \ldots f\, d \neq \emptyset$ by chain property 1. We again consider two cases:

- $b \equiv 0$. In this case, by A_3^{FSAT}, we find $d \equiv 0$. So we are given $(a,0) \wr (c,0)$, which fits chain property 5 of f. Since $(x,0) \wr (x+1,0)$, we can conclude using properties 4 and 6.
- $b \not\equiv 0$. We can apply A_2^{FSAT}. We further apply the induction hypothesis to $(d',b') \wr (d,d')$ and $(a,b') \wr (c',d')$, which is possible especially since $d < d'$, which we needed to explicitly add to A_2^{FSAT}. The remainder is straightforward by the defining properties of $\wr$ and chain property 5, similar to case 1. □

This concludes most of the heavy lifting required for this reduction, and we can now conclude.

Lemma 8.12 (Reflection) $\mathsf{FSAT}(F^{\mathsf{FSAT}}(h)) \to \mathsf{UDPC}h$

Proof We need to first build a chain f up to m using Lemma 8.10, where m is existentially quantified in F. After this, we are able to extract the numbers making up the solution to h by looking at the elements encoded in the big $\exists$-quantifier, and looking up the represented numbers in f. Afterwards, it remains to show that these numbers actually are solutions to h, which can easily be done using Lemma 8.11. □

8.5 Finalizing the Reduction

Our reduction is now complete:

Lemma 8.13 UDPC *is many-one reducible to* FSAT*, even when restricted to formulas with a single binary predicate.*

Proof By Lemma 8.8, Lemma 8.12. □

Lemma 8.14 FSAT *is undecidable, even when restricted to formulas with a single binary predicate.*

The problem FVAL, or finite validity, is the finite variant of VAL, requiring finite models similarly to how FSAT does.

In classical logic, we would now be able to conclude that finite validity is undecidable, because, assuming LEM, we can show that a formula φ is finitely valid iff $\neg\varphi$ is finitely satisfiable. In our meta-theory, we have to introduce double negations, so we can only show that $\mathsf{FVAL}\ (\dot{\neg}\varphi) \Leftrightarrow \neg\mathsf{FSAT}\ \varphi$. Yet, we can still reduce from $\overline{\mathsf{UDPC}}$ in this particular case. Since our definition of undecidability is by the co-semi-decidability of the halting problem, we can still conclude that FVAL is undecidable.

Lemma 8.15 $\overline{\mathsf{UDPC}}$ *is many-one reducible to* FVAL*, even when restricted to formulas with a single binary predicate.*

Lemma 8.16 FVAL *is undecidable, even when restricted to formulas with a single binary predicate.*

Proof We assume a decider for FVAL. We can thus semi-decide FVAL and also $\overline{\mathsf{UDPC}}$ (Lemma 8.15) and thus $\overline{\mathsf{UPC}}$ (Lemma 4.17), arriving at an semi-decider for $\overline{\mathsf{UDC}}$. Therefore, UDC is co-semi-decidable. Contradiction with Theorem 4.14. □

As a corollary, FVAL is not effectively axiomatizable, since one usually requires an effective deduction system to be sound, complete and enumerable/semi-decidable. However, this would imply the semi-decidability of FVAL, which would imply a co-semi-decider for the halting problem.

8.5.1 Remarks on the Finite Models of $F^{\mathsf{FSAT}}(h)$

While we aimed to characterize the finite models induced by the axioms to closely match our standard model, we did not chose to make the axioms strong enough to show canonicity, i.e. that all models are isomorphic to the standard model. In particular, in an arbitrary model, numbers (that is, elements d for which $N\,d$ is true) might not be arranged in a line. While a chain construction can be used to show that there is a linear "chain" from any d to 0, it is not necessarily the case that there is only one element d representing a given number, and the multiple numbers need not be related by $<$. Any number, while only having one predecessor, might have multiple successors which are neither first-order indistinguishable nor related by $\leq$. Thus, $\leq$ may not necessarily be total, and trichotomy may not hold. This explains the need for m, which bounds all elements representing numbers making up the solution to h from above, since this fixes them onto the unique chain descending from m to 0, and thus fixes a particular walk down the "successor tree" along which the elements of D behave more like natural numbers, allowing us to show Lemma 8.11. While we have not actually constructed a model where successors are not unique, we suspect it to exist.

Showing that the predecessors of a given number are unique is fairly straightforward using A_4^{FSAT} and antisymmetry of $\leq$: Given two predecessors of a number, the only way for both to fulfill the closeness property is if both are equal.

8.6 Remarks on the Coq Mechanization

In the Coq mechanization of the standard model, we construct an explicit list to show D finite. We need to be careful in our modeling of $\mathbb{N}_{\leq m}$ – we choose an index-inductive datatype with a single constructor by defining $\mathbb{N}_{\leq m} := fN : \forall n.n \leq_{\mathbb{N}} m \to \mathbb{N}_{\leq m}$, and use the fact that $\leq_{\mathbb{N}}$ has derivation uniqueness, that is, for fixed a, b, all proofs of $a \leq b$ are equal.

For the reflection step (Section 8.4), we have to do a lot of ground work before we can get started proving the lemmas iterated here. Importantly, we have to derive deciders for $\equiv$, just given a decider for $\wr$. We also have to show that $\equiv$ is a congruence relation for all of our syntactic sugar. Again, actually closing the reduction requires more work since we have to carefully extract the $\exists$-quantified elements representing the numbers making up the solution to h, since in Coq this embedding is implemented by re-using the de Brujin indices used to define h. Keeping this consistent during the actual implementation of F, as well as the $\exists$-elimination later on requires careful housekeeping.

Mechanizing this reduction takes ~ 750 lines of Coq code. This count excludes utility code shared with other reductions from previous chapters, as well as the proof of Fact 8.7, which is taken from the mechanization by Kirst and Larchey-Wendling [18].

Chapter 9

Minimizing Logical Connectives for FSAT

As before, we do not only want to show FSAT undecidable for the minimal signature, but also want to minimize the number of logical connectives. For this, we reconsider the approaches already discussed in Chapter 6.

9.1 Double Negation Translation

Previously, we defined a double negation translation in Definition 6.4. Then, we were not able to directly use this translation, since it was only valid under LEM, because we work in an intuitionistic meta-theory. For FSAT, we required all models to be finite and have decidable predicates. Since finite quantification is decidable, we can construct a decider for finite satisfaction in a fixed model:

Lemma 9.1 (Decidability of finite satisfaction) *Given a finite model $\mathcal{M}$, an environment $\rho : \mathcal{V} \to \mathcal{M}$ and a formula φ, $M \vDash_\rho \varphi$ is decidable.*

Proof All atomic predicates of $\mathcal{M}$ are decidable. Quantified formulas are decidable since finite quantification is decidable and $\mathcal{M}$ is finite. □

Thus, we are able to verify that the translation $(\cdot)^N$ of Definition 6.4 preserves satisfaction by finite models.

Lemma 9.2 *Given a finite model $\mathcal{M}$, an environment $\rho : \mathcal{V} \to \mathcal{M}$ and a formula φ, $(M \vDash_\rho \varphi) \Leftrightarrow (M \vDash_\rho (\varphi)^N)$.*

Proof By induction on φ, using Lemma 9.1 for $\vee, \exists$. □

Using this translation, we are able to reduce any FOL formula into one that is equivalent under FSAT and within the $\forall \to \bot$-fragment. We are able to prove the following reduction.

Lemma 9.3 FSAT *is many-one reducible to* FSAT *over the $\forall, \to, \bot$-fragment.*

Proof The function $(\cdot)^N$ fulfills the reduction properties by Lemma 9.2. □

Theorem 9.4 FSAT *is undecidable, even when restricted to formulas with a single binary predicate over the* $\forall, \to, \bot$*-fragment.*

Proof By Lemma 9.3 and Lemma 8.14 □

9.2 $\bot$ Elimination, Revisited

In Chapter 6, we already showed that SAT is decidable for formulas in a fragment without negation. The precise model used for this proof was the "trivial" model, which only has one element. Thus, it is a finite model, since additionally all relations are interpreted as true, which is trivially decidable.

Definition 9.5 (Trivial model) *For any signature, the* ***trivial model*** $\mathcal{M} := \mathbb{1}$ *has exactly one inhabitant,* $\star$*. All functions are trivially interpreted to yield* $\star$*, while all predicates are interpreted as* $\top$*. Since* $\top$ *is decidable and* $\mathbb{1}$ *is finite, this model is an admissible finite model.*

Lemma 9.6 FSAT *is decidable for formulas without* $\bot$*.*

Proof Every formula without $\bot$ is satisfied by the trivial model. □

9.3 Finite Validity

For finite validity, we can do the same reduction:

Lemma 9.7 FVAL *is many-one reducible to* FVAL *over the* $\forall, \to, \bot$*-fragment.*

Proof The function $(\cdot)^N$ fulfills the reduction properties by Lemma 9.2. □

Theorem 9.8 FVAL *is undecidable, even when restricted to formulas with a single binary predicate over the* $\forall, \to, \bot$*-fragment.*

Proof By Lemma 9.7 and Lemma 8.16. Note that our reduction starts at $\overline{\text{UDPC}}$. □

Unlike for FSAT, where satisfiability for fomulas in the $\forall, \to$-fragment is decidable, for FVAL we conjecture we are able to eliminate $\bot$ using an approach similar to the one from Chapter 6.

However, finding such a reduction is hard: FVAL actually is only co-semi-decidable. UDPC, however, is only semi-decidable. So we can not reduce from UDPC to FVAL, only from $\overline{\text{UDPC}}$. Furthermore, a Friedman-style translation likely requires additional encoding space in the interpretation of $\wr$. Unlike before, we have filled all available space, since $n \wr m$ for $n, m : \mathbb{N}_{\leq m}$ is now interpreted as $n \leq m$. This interpretation is necessary for the standard model to satisfy our reduction formula, and hence a significant change in the formula constructed by the reduction function, as well as the standard model, is likely required. Hence this remains further work. Formally, we conjecture:

Conjecture 9.9 FVAL *is undecidable, even when restricted to formulas with a single binary predicate over the* $\forall, \to$*-fragment.*

Chapter 10

Conclusion

We have discussed the undecidability of the variants of the Entscheidungsproblem present when investigating first-order logic in an intuitionistic framework. In short, we found new proofs that validity (asking whether a formula is valid in all models), satisfiability (whether a formula is valid in at least one model) and intuitionistic provability (whether a formula is provable according to some deduction system) are undecidable even when restricted to a single binary relation, while also investigating which restrictions on the logical connectives are possible.

To be precise, we have shown that validity, intuitionistic provability and Kripke validity are undecidable even when further restricted to formulas of the $\forall, \rightarrow$-fragment (without negation), beyond just having a single binary relation. This was shown in Theorem 6.12 for valdity and Theorem 7.14 for provability and Kripke validity. For satisfiability, Kripke satisfiability as well as satisfiability and validity restricted to finite models, we only showed that they are undecidable for formulas restricted to the $\forall, \rightarrow, \bot$-fragment and to having only a single binary relation. This was shown in Lemma 6.14 for satisfiability, Theorem 7.19 for Kripke satisfiability, Lemma 8.14 for finite satisfiability, and Lemma 8.16 for finite validity. For finite validity, we conjecture (see Conjecture 9.9) that this result could be sharpened to the fragment without negation. For all variants of satisfiability, the $\forall, \rightarrow, \bot$-fragment is already minimal.

10.1 The Coq Nechanization

The complete Coq mechanization can be found at

`https://nilsvital.de/coq/indexpage.html`

To be precise, we host a fork of the Coq Library of Undecidability Proofs [6], which includes our proofs. The electronic version of this thesis includes references to the Coq mechanization for each specific lemma. The particular files of interest are:

- `FOL.minFOL_undec` – File summarizing the entire thesis

- `FOL.Reductions.H10UPC_to_FOL_minimal` – Results of Chapter 6, Chapter 7
- `FOL.Reductions.H10UPC_to_FSAT` – Results of Chapter 8, Chapter 9
- `FOL.Util.DoubleNegation` – Mechanization of Lemma 9.2
- `DiophantineConstraints.H10UPC` – Definition of ≀. Originally due to Andrej Dudenhefner and Dominik Kirst
- `DiophantineConstraints.Util.H10UPC_facts` – Some results of Chapter 4
- `DiophantineConstraints.Reductions.H10UC_SAT_to_H10UPC_SAT` – Undecidability of UDPC
  - `DiophantineConstraints.H10UPC_undec` actually states the undecidability

Note that in Coq, the problem UDPC is know as `H10UPC_SAT`. The relation ≀ is called `h10upc_sem_direct`.

The proof of Theorem 7.14 (undecidability of provability in the minimal fragment and signature) takes about 900 lines of Coq code. From that proof, the undecidability of validity, satisfiability, Kripke validity and Kripke satisfiability follow with only a few lines of code each. This proof involves a lot of work in double-negated contexts, which makes it somewhat hard to understand without knowing the "untranslated" version of Chapter 5. However, it is still more direct than a proof based on ZF, which first has to prove a lot of basic constructions regarding axiomatic set theory, before attempting to reduce an undecidable problem into a set-theoretic formulation. Of course, the proof requires the undecidability of Diophantine equations, which we discuss shortly.

For finite satisfiability, our proof also takes about 1200 lines of Coq code when including the double negation of Chapter 9. Here, we did not need to perform a manual translation into the $\forall, \to, \bot$-fragment, since this translation can be shown to hold in general, as finite models behave classically.

A recurring theme in the Coq mechanization of FOL is the use of de Brujin indices, which are used because they allow for an easier mechanization. Especially, the meta-theory of FOL deduction systems is easier to mechanize. However, giving concrete FOL terms is harder than on paper, since indices have to be computed before entering the term. Verifying that a formula constructed using indices is correct is easier, since we can circumvent the issues introduced by shadowing and aliasing.

It should be noted that we do not present a mechanization of the "simple" version of Chapter 5, since that chapter proves strictly weaker results than Chapter 7. During

the development of the reduction ultimately outlined in Chapter 6, we did develop a "simpler" version similar to that of Chapter 5, however, this version also uses a different standard model and very ad-hoc construction of the reduction formula. We still include Chapter 5 for didactical reasons, since the reduction of the later chapters are best understood by comparing them to to the "simple" version performed in that chapter. Similarly, the presented proof of Lemma 6.10 also is elided, since it can easily be followed from Lemma 7.12.

The Coq proofs themselves, while not straightforward, are nontheless usually kept simple, avoiding more advanced type theoretic notions like index-inductive elimination. There are a few exceptions: First, we rely on the uniqueness of proofs of $<$ to construct our finite model in Section 8.3. This result is often a textbook example for working with indexed-inductive predicates, and thus already part of the Coq standard library. Furthermore, the chains defined in Definition 7.9 are defined similarly to a vector, so that working with them requires a modest amount of index-inductive elimination and inversion principles.

10.2 Comparsion to Existing Work

Compared to the paper by Kirst and Larchey-Wendling [18], our undecidability proof is significantly shorter (ignoring the fact that we start at Diophantine equations). Their proof performs explicit quotient construction and signature compression, which requires several thousand lines of Coq code, whereas our reduction not only immediately targets the minimal signature, but also inlines the quotient, avoiding an explicit construction of this. However, while we just proved the undecidability of finite satisfiability for that minimal case, they proved that any FOL formula over a discrete signature can be reduced to one in the minimal fragment while preserving finite satisfiability, which is a much more general result.

The properties of the relation $\wr$ were critical in keeping this reduction short. Since $\wr$ has a very simple axiomatization, working with it in FOL was very feasible. Especially, one avoids having to first prove basic results about the underlying theory, as was the case with ZF. Additionally, the axiomatization is already formulated using just pairs and the relation itself, avoiding the need for a complex reduction into a signature with just a single binary relation.

We expect that $\wr$ and UDPC can be used as a reduction source for other problems, especially since it not only has an easily approachable definition (unlike Turing machines, which involve dealing with an operational semantics), but also a short, elegant, effective axiomatization, which makes it especially promising for proving similar results for other logics.

However, while our particular reductions from UDPC (or UDC) to FOL problems were simple, showing that our Diophantine constraints problem itself is undecidable

is very hard. In particular, this result relies on the undecidability of Diophantine equations. Diophantine equations were first shown undecidable in 1970 by Matiyasevitch [24] after a program spanning several decades.[1] On the other hand, this shows the utility of collecting undecidability proofs in a library like the Coq Library of Undecidability Proofs [6]: Since the undecidability of Diophantine equations is already mechanized, it can be used as a source problem for other reductions while being certain that it actually is an undecidable problem. Further, collecting existing undecidable problems allows for finding increasingly simpler reductions, or for showing increasingly complex goals. Also, first-order logic was already mechanized in the library, which freed us from mechanizing the syntax and deduction system ourselves. Thus, the library greatly accelerated this project and made it possible to be in the scope of a Bachelor's thesis at all.

Furthermore, when comparing with the previous mechanization of the undecidability result for validity by Kirst and Hermes [17], we note that our result also restricts the logical fragment to the $\forall, \rightarrow$-fragment, which theirs does not. We thus believe this to be the first mechanization of the undecidability of various first-order problems not only for the minimal signature, but also for the minimal logical fragment.

We shall also compare our work to the original, non-mechanized undecidability proofs we already mentioned. Notably, Turing [29] shows that validity for a system similar to FOL is undecidable by constructing a formula provable if and only if a given Turing machine ever outputs 0. Hence, he many-one reduces from a slight variation of the halting problem. The formula he constructs uses conjunction, implication and a signature with more than one binary relation, hence it is not minimal. Turing does not explicitly consider validity and satisfiability, however, since he works in a classical meta-theory, the undecidability of these problems immediately follows.

Kalmár's work [16] is part of a broader program by Kalmár and others to find a signature compression which, starting at an arbitrary FOL formula, yields a FOL formula in the minimal signature, that is, with only a single binary relation. This paper contributes a crucial part of this chain, reducing from a formula with arbitrarily many binary relations to a formula with just one such relation, resting on previous work by Kalmár and others which already reduced the arbitrary case to that case. Notably, Kalmár also reduces the formula to a simpler quantifier prefix, something which we have not considered. Kalmár's result is much stronger than our result, since it allows one to signature-compress an arbitrary FOL formula, whereas we have only shown that validity and related problems are undecidable for a small signature. We also have not considered the quantifier prefix for this result, while

[1] Finding a process to determine whether Diophantine equation solutions exist was posed by Hilbert in 1900, as problem 10 of his famous list of problems.

Kalmár's result also has as a corollary that our FOL problems are undecidable for formulas with such a quantifier prefix.

Compared to Trakhtenbrot's original proof of the undecidability of finite satisfiability [28], we have a somewhat different approach. Trakhtenbrot reduces from μ-recursive functions, an alternative Turing-complete model of computation. His reduction builds a formula which is finitely satisfiable if and only if the given μ-recursive function has a root, i.e. its image contains 0. His actual reduction function is surprisingly elegant, since it proceeds by plain structural recursion on the AST of the given μ-function.

Since Trakhtenbrot's original proof, many alternative variations have been given, like the one by Libkin [21], which directly reduces from Turing machines. Libkin also claims that this reduction can be easily transformed into targeting the minimal signature, but leaves this as an exercise to the reader.

10.3 Open Problems

As mentioned, showing that finite validity is undecidable for a fragment without negation remains open. We conjecture (Conjecture 9.9) that a reduction from $\overline{\mathsf{UDPC}}$ (since finite validity is co-semi-decidable, where UDPC is semi-decidable) is possible using the axiomatization presented in this thesis.

Similarly, in Chapter 7, we only showed that provability defined over the intuitionistic proof system is undecidable. Such a result immediately extends to provability for the classical proof system as long as one assumes LEM. We suspect that one might weaken the used axioms here, yet this too remains open. A technique based on a translation from classical provability to intuitionstic provability defined by Forster et al. [5] might be successful here.

To continue, one may want to show that FOL is undecidable even when restricted to a certain quantifier prefix, like in the work by Kalmár [16]. A cursory analysis of our formula for validity defined in Section 6.3 reveals that it seems to be Π_6^{0}[2], which certainly is not the minimal prefix. Reducing the quantifier prefix is likely to require significant work, and thus also remains open.

[2]The notation Π_6^0 means that the formula can be converted to one where all quantifiers are at the front of the overall formula, where that prefix has shape $\forall\exists\forall\exists\forall\exists$. The actual prefix might repeat individual quantifiers, but they must appear in that order.

Bibliography

[1] Andrej Bauer. First Steps in Synthetic Computability Theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006. ISSN 1571-0661. doi: https://doi.org/10.1016/j.entcs.2005.11.049. URL `https://www.sciencedirect.com/science/article/pii/S1571066106001861`. Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXI).

[2] Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 1936. doi: 10.2307/2269326.

[3] Thierry Coquand and Christine Paulin. Inductively defined types. In Per Martin-Löf and Grigori Mints, editors, *COLOG-88*, pages 50–66, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. ISBN 978-3-540-46963-6.

[4] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean Theorem Prover (System Description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6.

[5] 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. doi: 10.1145/3293880.3294091. URL `https://www.ps.uni-saarland.de/extras/fol-undec/`.

[6] Yannick Forster, Dominique, Larchey-Wendling, Andrej, Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, and Gert Smolka. A Coq Library of Undecidable Problems. CoqPL 20, 2020.

[7] Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness Theorems for First-Order Logic Analysed in Constructive Type Theory. In *Symposium on Logical Foundations Of Computer Science (LFCS 2020), January 4-7, 2020, Deerfield Beach, Florida, U.S.A.*, Jan 2020.

[8] Harvey Friedman. Classically and intuitionistically provably recursive functions. In Gert H. Müller and Dana S. Scott, editors, *Higher Set Theory*, pages 21–27, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-540-35749-0.

[9] Gerhard Gentzen. Untersuchungen Über Das Logische Schließen. I. *Mathematische Zeitschrift*, 35:176–210, 1935.

[10] Gerhard Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936. URL `http://eudml.org/doc/159839`.

[11] K. Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. In K. Berka and L. Kreiser, editors, *Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik (vierte Auflage)*, pages 305–315. Akademie-Verlag, Berlin, 1986.

[12] Kurt Gödel. Zur intuitionistischen Arithmetik und Zahlentheorie – Ergebnisse eines Mathematischen Kolloquiums. pages 493–565, 1928-1933.

[13] David Hilbert and Wilhelm Ackermann. *Grundzüge der Theoretischen Logik*. Springer Verlag, 1928.

[14] Johannes Hostert, Mark Koch, and Dominik Kirst. A Toolbox for Mechanised First-Order Logic. In *The Coq Workshop 2021*, 2021.

[15] W. A. Howard. The formulae-as-types notion of construction. 1969.

[16] László Kalmár. Zurückführung des Entscheidungsproblems auf den Fall von Formeln mit einer einzigen, binären, Funktionsvariablen. *Compositio Mathematica*, 4:137–144, 1937. URL `http://www.numdam.org/item/CM_1937__4__137_0/`.

[17] Dominik Kirst and Marc Hermes. Synthetic Undecidability and Incompleteness of First-Order Axiom Systems in Coq. In *Interactive Theorem Proving - 12th International Conference, ITP 2021, Rome, Italy*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021.

[18] Dominik Kirst and Dominique Larchey-Wendling. Trakhtenbrot's Theorem in Coq. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 79–96, Cham, 2020. Springer International Publishing. ISBN 978-3-030-51054-1.

[19] Saul A. Kripke. Semantical Analysis of Intuitionistic Logic I. In J.N. Crossley and M.A.E. Dummett, editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92–130. Else-

vier, 1965. doi: https://doi.org/10.1016/S0049-237X(08)71685-9. URL https://www.sciencedirect.com/science/article/pii/S0049237X08716859.

[20] Dominique Larchey-Wendling and Yannick Forster. Hilbert's Tenth Problem in Coq. 4th International Conference on Formal Structures for Computation and Deduction, 2019. doi: 10.4230/LIPIcs.FSCD.2019.27. URL http://drops.dagstuhl.de/opus/volltexte/2019/10534.

[21] Leonid Libkin. *Elements of Finite Model Theory*. Springer, August 2004. ISBN 3540212027.

[22] L. Löwenheim. Über Möglichkeiten im Relativkalkül. *Mathematische Annalen*, 76:447–470, 1915. URL http://eudml.org/doc/158703.

[23] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. Elsevier, 1975. doi: https://doi.org/10.1016/S0049-237X(08)71945-1. URL https://www.sciencedirect.com/science/article/pii/S0049237X08719451.

[24] Yuri V Matiyasevich. Enumerable sets are Diophantine. *Doklady Akademii Nauk SSSR*, 191:279–282, 1970.

[25] Ulf Norell. *Dependently Typed Programming in Agda*, pages 230–266. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-04652-0. doi: 10.1007/978-3-642-04652-0_5. URL https://doi.org/10.1007/978-3-642-04652-0_5.

[26] Fred Richman. Church's thesis without tears. *Journal of Symbolic Logic*, 48(3): 797–803, 1983. doi: 10.2307/2273473.

[27] The Coq Development Team. The coq proof assistant. Jan 2021. doi: 10.5281/zenodo.4501022.

[28] Boris Trakhtenbrot. The Impossibility of an Algorithm for the Decidability Problem on Finite Classes. In *Proceedings of the USSR Academy of Sciences*, 1950.

[29] Alan M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936. URL http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf.

[30] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1925–1927.