FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Extraktion von Programmen aus Beweisen

mit Friedmans A-Übersetzung und Realisierbarkeit

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Computational Logic

eingereicht von

Mariëlle Rietdijk
Matrikelnummer 1635309

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Prof. Alexander Leitsch

Wien, 1. Juni 2018

Mariëlle Rietdijk	Alexander Leitsch

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Extracting programs from proofs

using Friedman's A-translation and realizability

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computational Logic

by

Mariëlle Rietdijk
Registration Number 1635309

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Alexander Leitsch

Vienna, 1st June, 2018

Mariëlle Rietdijk

Alexander Leitsch

Erklärung zur Verfassung der Arbeit

Mariëlle Rietdijk
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Juni 2018

Mariëlle Rietdijk

Acknowledgements

I would first of all like to thank my supervisor Alexander Leitsch, for suggesting the topic of this thesis that allowed me to explore both the areas of logic and computation, which I very much enjoyed, as well as for all his patience and time for our regular meetings and careful proofreading of all the content of this thesis. I am thankful as well to the people of the GAPT-group, for their availability to answer all questions that popped up while working on the implementation that complements this thesis, and their valuable suggestions for it.

I feel very grateful and honored for having been awarded the Helmut Veith stipend during my stay in Vienna and would hereby like to express my gratitude towards everyone who contributed to this.

This thesis marks the end of my master studies of almost three years. I am grateful to my family as well as to my friends for all their support during this time. I would like to use this opportunity especially to thank the friends I have met through this master, for the great moments we shared but also for backing me up and helping me out in lesser moments. You have made this master an experience I would never have wanted to miss in my life. Lastly, I would like to thank Dave, for helping me whenever possible, and supporting and always being there for me whenever I needed it the most.

Kurzfassung

Das Thema dieser Dissertation basiert auf der starken Verbindung zwischen mathematischen Beweisen und Programmen, in dem Sinne, dass Beweise einen Rechengehalt haben, zusätzlich zur bloßen Vermittlung der Wahrheit ihrer Schlussfolgerung. Insbesondere beschreiben wir eine Methode zum Extrahieren von Programmen aus gegebenen klassischen arithmetischen Beweisen erster Ordnung von Π_2-Sätzen. Dies geschieht mit Hilfe einer Beweisübersetzung, die klassische Beweise in intuitionistische Beweise umwandelt, und einer Realisierbarkeitsinterpretation, die es erlaubt, intuitionistische Zeugen für Sätze aus ihren Beweisen in Form von Programmen zu extrahieren. Wir werden Kolmogorovs negative Übersetzung und Friedman's A-Übersetzung verwenden, um die Beweise zu transformieren und Realisatoren nach Kreisels modifizierter Realisierbarkeit zu extrahieren.

Wir interessieren uns speziell für diese intuitionistischen Zeugen für Theoreme, weil sie Programme in Bezug auf eine Spezifikation sind, die in der Schlussfolgerung des Beweises ausgedrückt wird. Eine logische Formel A mit den freien Variablen $x_1, \ldots, x_r$ und y kann als Spezifikation verstanden werden, wobei ein Programm, das diese Spezifikation erfüllt, eine Funktion f berechnet, so dass für alle möglichen Eingabewerte $n_1, \ldots, n_r$, der Satz $A[n_1/x_1] \ldots [n_r/x_r][f(n_1, \ldots, n_r)/y]$ wahr ist. Die in dieser Arbeit beschriebene Methode extrahiert Programme, die solchen Spezifikationen A entsprechen, aus Beweisen, die die Existenz einer Funktion für A implizieren, also aus Beweisen von Sätzen $\forall x_1 \ldots \forall x_r \exists y A$. Insbesondere bietet uns die Realisierbarkeitsinterpretation eine Möglichkeit, Programme aus konstruktiven Beweisen zu extrahieren, und die Beweistransformation ermöglicht uns, diese Extraktion auch auf klassische Beweise auszuweiten.

Diese Extraktion von Programmen aus klassischen Beweisen von Π_2 -Formeln wird im GAPT-System implementiert, einem an der TU Wien entwickelten Beweistheorie-Framework, das sich auf Beweistransformationen konzentriert. Das GAPT-System enthält unter anderem Schnittstellen zu Theorembeweisern, Deskolemisierung von Beweisen und Übersetzungen zwischen Beweiskalkülen. Mit Hilfe dieser Funktionalitäten, die bereits in GAPT vorhanden sind, zielt die in dieser Arbeit diskutierte Implementierung darauf ab, zur Programmsynthese in GAPT beizutragen - die Aufgabe, automatisch ein Programm zu konstruieren, das eine gegebene Spezifikation erfüllt.

Abstract

The topic of this thesis is based on the strong connection between mathematical proofs and programs, in the sense that proofs have computational content, additional to merely conveying the truth of their conclusion. In particular, we will describe a method for extracting programs from given classical, first-order arithmetical proofs of Π_2 theorems. This is done with the help of a proof translation that transforms classical proofs to intuitionistic proofs, and a realizability interpretation that allows to extract intuitionistic witnesses for theorems from their proofs, in the form of programs. We will use Kolmogorov's negative translation and Friedman's A translation to transform the proofs and extract realizers according to Kreisel's modified realizability.

We are specifically interested in these intuitionistic witnesses for theorems because they are programs with respect to a specification expressed in the conclusion of the proof. A logical formula A with free variables $x_1, \ldots, x_r$ and y can be understood as a specification, where a program satisfying this specification computes a function f such that for all possible input values $n_1, \ldots, n_r$, the sentence $A[n_1/x_1] \ldots [n_r/x_r][f(n_1, \ldots, n_r)/y]$ is true. The method described in this thesis extracts programs satisfying such specifications A from proofs that imply the existence of a function for A, that is, from proofs of sentences $\forall x_1 \ldots \forall x_r \exists y A$. Specifically, the realizability interpretation provides us with a way of extracting programs from constructive proofs, and the proof transformation allows us to extend this extraction to classical proofs as well.

This extraction of programs from classical proofs of Π_2 formulas is implemented in the GAPT system, a proof theory framework developed at TU Wien, that focuses on proof transformations. The GAPT system contains, among other things, interfaces to theorem provers, deskolemization of proofs, and translations between proof calculi. With the help of these functionalities that are already present in GAPT, the implementation discussed in this thesis aims to contribute to program synthesis in GAPT - the task to automatically construct a program that satisfies a given specification.

Contents

CHAPTER 1

Introduction

This thesis falls within the intersection of the areas of computability theory and logic, specifically proof theory, where mathematical proofs are analyzed on behalf of their computational content. This relation between proof theory and computability theory is most prominent in the scope of the Curry-Howard correspondence, which results from the observation that a proof of a formula can be directly understood as a construction of a typed program, and vice versa. An introduction to this topic can be found in [rU06], where the idea for this thesis was found as well.

The connection between programs and proofs is however not only found in the syntactic correspondence of proof calculi and type systems, but comes already quite naturally when considering intuitionistic instead of classical logic. As opposed to the classical variant, intuitionistic logic emphasizes the need for constructions to witness or justify the truth of formulas, which can be understood as programs. The shape of such constructions for composed formulas is described by the Brouwer-Heyting-Kolmogorov (BHK) interpretation of intuitionistic logic. Different formal representations of constructions following the BHK interpretation are known, such as realizability interpretations for first-order intuitionistic logic combined with the Peano axioms, called Heyting arithmetic. Kleene introduced in [Kle45] the first realizability interpretation, where the witnessing constructions, called realizers, take the shape of (Gödel numbers of) recursive functions. A variant of realizability that is first described by Kreisel in [Kre62], where realizers are programs, is discussed in this thesis. Translations from proofs in classical logic to proofs in intuitionistic logic, of which we will describe one in this thesis, generalize this connection between proofs and programs.

In this thesis we describe a method of extracting algorithmic content from classical proofs of Π_2 theorems in arithmetic, i.e. theorems of the form $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$, where A is quantifier-free. The method is based on a proof translation and a variant of realizability, as mentioned above. We will extract this content from proofs that are presented in a natural deduction proof calculus for Peano arithmetic (classical first-order logic with

the Peano axioms), extended with definitions for all primitive recursive functions. The first step of this extraction method is a transformation of proofs of Π_2 theorems in Peano arithmetic, to proofs of those theorems in Heyting arithmetic, using Kolmogorov's negative translation, and Friedman's A-translation. The second step consists of extracting realizers in the form of programs for those theorems from their constructive proofs, which, as explained above, can be understood as justifications of the intuitionistic truth of the theorems. The realizers that we extract (referred to as modified realizers, or m-realizers) are terms of Gödel's system **T** - an extension of the simply typed lambda calculus with primitive recursion for finite types. In summary, we will describe a method to extract constructive witnesses in the form of programs from classical proofs of Π_2 theorems.

The reason that we are particularly interested in m-realizers of classical Π_2 theorems is because an m-realizer for a Π_2 sentence $\forall x_1 \dots \forall x_r \exists y A$ defines a program that satisfies the specification A. Specifications in this sense are Σ_1 formulas, with free variables $x_1, \dots, x_r$ and y, which are placeholders for input values and an output value respectively. The formula A expresses the desire for a program that computes a function f such that for all input values $n_1, \dots, n_r$, the proposition $A[n_1/x_1] \dots [n_r/x_r][f(n_1, \dots, n_r)/y]$ is true. We thus express in A how the output of the desired program should relate to its input. Specifically, we do not say anything about how the output is obtained from the input, but merely that it relates to the input in a certain way. The specification basically describes a number of pairs, namely those pairs $n_1, \dots, n_r, m$ such that $A[n_1/x_1] \dots [n_r/x_r][m/y]$ is true. The function f that is computed by the constructed program, should be a subset of those allowed pairs.

From this perspective, the method for program extraction from Π_2 sentences that is described before, which we will discuss in depth in the thesis, thus allows us to extract programs satisfying Σ_1 specifications A from classical proofs of propositions $\forall x_1 \dots \forall x_r \exists y A$. That is, we reduce the search for a program for a given specification A to finding a proof for $\forall x_1 \dots \forall x_r \exists y A$, i.e a proof for the proposition that A defines a function over the natural numbers. The algorithm described in this thesis then automatically produces a program that satisfies A.

This program extraction method will be implemented in the GAPT system - a proof theory framework developed at TU Wien. The implementation is part of a long-range program in GAPT, which aims to automatically construct programs for given specifications, to which we refer to as program synthesis. The first step in this program is to formulate a specification A for a program as described above. GAPT offers built-in, and interfaces to (inductive) theorem provers which can be used to obtain a proof for $\forall x_1 \dots \forall x_r \exists y A$ in a specific proof calculus (such as the resolution calculus). This is the only place in the program where some user interaction may be required, as inductive theorem proving cannot be fully automated in practice. The next step is to transform this proof into another in the proof calculus for which the program extraction is defined. This is done with the help of deskolemization (as theorem provers typically give skolemized proofs) and translations between different proof calculi (from resolution calculus to sequent calculus, and from sequent calculus to natural deduction). Then the program extraction described

in this thesis, i.e. the transformation of a classical to an intuitionistic proof combined with the extraction of an m-realizer from this proof, can be applied to obtain a program satisfying the specification A.

1.1 Related work

As mentioned above, the method described in this thesis is aimed towards a program synthesis program in the GAPT system. Research in the field of program synthesis arises from the general objective in computer science of obtaining programs that are formally proven to be correct with respect to specifications, i.e. verified programs. There are many ways to obtain verified programs. The field of formal verification for example is an extensive research field in which one aims to prove the correctness of existing programs, and systems in general. The topic of this master's thesis rather falls within the area of program synthesis, because we start with only a specification and no program.

The aim of program synthesis is to automatically generate a program that satisfies the intention of a user, expressed as a specification. The deductive approach of program synthesis that we consider, namely to extract a program from a proof of the specification, is one of many approaches to program synthesis, which are all described in [GPS17]. After the development of automated theorem provers, the idea emerged to use theorem provers to construct a proof and from this proof extract a program. This idea is described for example in [MW71]. There are however many other approaches towards program synthesis, that differ in the type of specifications (such as examples and demonstrations) and the derivation of programs (e.g. machine learning techniques).

The approach of program synthesis that we discuss here relies on reducing the problem of finding a program for a specification to finding a proof for a theorem. If we can express a specification as a sentence in a logic for which a Curry-Howard correspondence is known, then finding a program for this specification is the same as finding a constructive proof for the sentence. A Curry-Howard isomorphism namely identifies types of programs with theorems and their proofs. Types are usually understood as not as expressive as specifications. However, the theory of dependent types allows the type to be sufficiently expressive that it serves as a specification for a program. When we express our specifications for programs as dependent types, then we reduce the problem of finding a program for this type, i.e. specification, to finding a constructive proof for a theorem. There are many systems that implement some variant of dependently typed languages, such as the Coq proof assistant [coq], where a proof assistant helps to find a formal proof for the specification. Dependent type systems can also be used for formal verification, where a program is type-checked for a specification, which corresponds to proof-checking from the logical point of view.

The theoretical foundations of the program extraction considered in this thesis, namely the extraction of modified realizers from constructive proofs and a proof translation of Π_2 from classical to intuitionistic logic are known already for several decades, and this implementation is therefore not the first one. For example, Minlog, as described

in [BBS+98], extracts programs from proofs based on modified realizability and uses a proof translation from classical to intuitionistic proofs based on Friedman's A-translation as well. The focus of the implementation of these methods in GAPT is however to use the built-in provers and interfaces to external theorem provers to obtain a proof, instead of defining it in the system itself as in Minlog.

Although the motivation of implementing a method for program extraction from proofs as described in this thesis comes from program synthesis, the extraction of content from formal proofs, not necessarily restricted to the extraction of programs, is a research field on its own. The suggestion that a proof may contain more information than merely the truth of its conclusion is due to Kreisel. The idea behind this is that a proof not merely shows that a statement is true, but also shows why the theorem is true. Originally, Kreisel called this research field 'proof unwinding'. A survey of research inspired by the extraction of additional content form proofs can be found in [Fef96].

A modern variant of this research field led by Ulrich Kohlenbach, referred to as proof mining, is described in the book [Koh08]. Although it is not always possible to extract computable witnesses from classical proofs, one could still try to extract other information, such as bounds for possible witnesses. This research has led to, for example, the extraction of new computable bounds from proofs in analysis. Other examples of research in proof mining can be found in the area of proof normalization, as described in for example [Lei15], where the removal of lemma's through cut-elimination reveals computational content that was previously hidden. In [Ili17] other directions for proof mining are suggested, found in the application of new theories from the programming languages research area.

1.2 Structure of the thesis

This document aims to explain the details of the methods that are implemented to obtain a program extraction algorithm from classical Π_2 proofs, and how this indeed serves the purpose of extracting programs that satisfy the initially defined specification. The input of the program extraction is discussed in the section **Proof calculus**, where we will give a detailed description of the form of the proofs that serve as the input of the implementation. Next, in section **Proof transformation**, the first part of the implementation will be described, namely the transformation of classical proofs of Π_2 theorems to intuitionistic proofs of those theorems. Then in the chapter **Program extraction**, we will describe the extraction of programs from the intuitionistic arithmetical proofs. In the following section **Inductive data types**, we will address the extension towards inductive data types that is implemented in GAPT. Finally, we will suggest some improvements that can be made to the algorithm, taking into account the theoretical limitations of this method, in the section **Conclusion and future work**.

CHAPTER 2

Proof calculus

In this section we will describe a natural deduction proof calculus for first-order arithmetic in which the proofs that are the inputs for the program extraction are presented. In short, we will use a natural deduction proof calculus for Peano arithmetic, extended with definitions for all primitive recursive functions. We will first give the details of the proof calculus for Peano arithmetic, and then explain how exactly we extend the calculus with these definitions. In the chapter following this one, whenever we mention Peano or Heyting arithmetic, we refer to this extended version.

2.1 Natural deduction calculus for Peano arithmetic

We will first define the notion of substitution that we use. When applying a substitution $[t/x]$ to a formula A, all free occurrences of the variable x in A are replaced by the term t. Whenever a variable occurring in the term t already occurs as a bound variable in A, we will consider an alpha-equivalent variant of A for which this is not the case, and apply the substitution to this formula instead. This is to prevent that the formula gets new bound variables after applying the substitution. For example, the outcome of $(\forall x P(x, y))[x/y]$ will not be $\forall x P(x, x)$, where the quantifier now binds both variables as opposed to before applying the substitution, but $\forall z P(z, x)$ (or any other alpha-equivalent expression).

As has already been mentioned, we will use natural deduction as the proof calculus to present proofs in Peano arithmetic. The inference rules of the presentation that we use, operate on expressions which we refer to as judgments. A judgment is of the form $\Gamma \vdash A$, where A is a formula, which we call the succedent of the judgment, and Γ is a multi set of formulas, which we call the antecedent of the judgment. The intended meaning is that A is a consequence of Γ, i.e. if all the formulas in the antecedent are provable, then the succedent is provable. We say that a judgment $\Gamma \vdash A$ is provable if there is a proof with $\Gamma \vdash A$ as its conclusion.

The logical core of the natural deduction proof calculus consists of elimination and introduction rules for connectives and the equality predicate, the logical axiom schema, and an excluded middle rule to capture classical reasoning in addition to the aforementioned constructive rules. Furthermore, the calculus contains two structural rules, namely the weakening and contraction rules. As the antecedent of the judgment is a multi set, there is no need for an additional exchange rule. The Peano axioms concerning equality are theorems of this logic, due to the equality elimination and introduction rule. We could equivalently define the calculus without these rules, but with the equality axioms explicitly added to the calculus instead. The induction scheme of the Peano axioms is captured by a single induction rule, which again is equivalent to the calculus without this rule, but with all instances of the induction scheme as axioms. The remaining Peano axioms, which are the two axioms describing the natural numbers as an inductive data type, and the four axioms defining addition and multiplication as primitive recursive functions, are simply axioms in the calculus itself as well.

In the following, let A, B, C be formulas, and Γ, Π be multisets of formulas. The variable α denotes an eigenvariable.

Logical axiom schema

$$\frac{}{A \vdash A}\,\text{ax}$$

Structural rules

$$\frac{\Gamma \vdash B}{A, \Gamma \vdash B}\,\text{w} \qquad \frac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}\,\text{c}$$

Propositional rules

Elimination rules

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge\text{e1} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\,\wedge\text{e2}$$

$$\frac{\Gamma \vdash A \to B \qquad \Pi \vdash A}{\Gamma, \Pi \vdash B}\,\to\text{e}$$

$$\frac{\Gamma \vdash A \vee B \qquad \Pi, A \vdash C \qquad \Delta, B \vdash C}{\Gamma, \Pi, \Delta \vdash C}\,\vee\text{e}$$

Introduction rules

$$\frac{\Gamma \vdash A \qquad \Pi \vdash B}{\Gamma, \Pi \vdash A \wedge B}\,\wedge\text{i}$$

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \to B}\,\to\text{i}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee\text{i1} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee\text{i2}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\,\bot\mathrm{e}$$

Quantifier rules

Elimination rules

$$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]}\,\forall\mathrm{e}$$

$$\frac{\Gamma \vdash \exists x A \qquad A[\alpha/x], \Pi \vdash B}{\Gamma, \Pi \vdash B}\,\exists\mathrm{e}$$

Introduction rules

$$\frac{\Gamma \vdash A[\alpha/x]}{\Gamma \vdash \forall x A}\,\forall\mathrm{i}$$

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}\,\exists\mathrm{i}$$

Equality rules

Elimination rule

$$\frac{\Gamma \vdash t_1 = t_2 \qquad \Pi \vdash A[t_1/x]}{\Gamma, \Pi \vdash A[t_2/x]}\,=\mathrm{e}$$

Introduction rule

$$\frac{}{\vdash t = t}\,=\mathrm{i}$$

Excluded middle

$$\frac{\Gamma, A \vdash B \qquad \Pi, \neg A \vdash B}{\Gamma, \Pi \vdash B}\,\mathrm{em}$$

Peano axioms

Induction

$$\frac{\Gamma \vdash A[0/x] \qquad A[\alpha/x], \Pi \vdash A[s(\alpha)/x]}{\Gamma, \Pi \vdash A[t/x]}\,\mathrm{ind}$$

Remaining Peano axioms

$$\frac{}{\vdash \forall x \neg(s(x) = 0)}\,\mathrm{nat1}$$

$$\frac{}{\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))}\,\mathrm{nat2}$$

$$\frac{}{\vdash \forall x(x + 0 = x)}\,{+}0$$

$$\frac{}{\vdash \forall x \forall y(x + s(y) = s(x + y))}\,{+}s$$

$$\frac{}{\vdash \forall x(x \cdot 0 = 0)}\,{\cdot}0$$

$$\frac{}{\vdash \forall x \forall y(x \cdot s(y) = (x \cdot y) + x)}\,{\cdot}s$$

Note that the calculus does not have any rules for the negation. The reason for this is that a negated formula $\neg A$ is equivalent to the formula $A \to \bot$. We can therefore consider the connective $\neg$ as an abbreviation. When using introduction and elimination rules for the negation, we simply refer to the corresponding rules for the implication, emphasizing that the formula B in these rules (the consequent of the implication) is $\bot$.

2.2 Extension with primitive recursive definitions

As mentioned before, we will extend Peano arithmetic with definitions for all primitive recursive functions. This means that for every r-ary primitive recursive function f, we will have an r-ary function symbol $\hat{f}$ in the language, and we will extend the calculus with axioms such that $\hat{f}$ represents the function f. This means that $f(n_1, \dots, n_r) = m$ if and only if we can prove in the extended calculus that $\vdash \hat{f}(\bar{n_1}, \dots, \bar{n_r}) = \bar{m}$, where a numeral $\bar{n}$ for a number n is a term $s(\dots s(0)\dots)$, where $s(\dots)$ occurs n times.

The function symbols 0 and s are already part of the language of arithmetic, and we will therefore not add new symbols $\hat{0}$ and $\hat{s}$ to the language. The Peano axioms guarantee that they represent the primitive recursive constant 0, and the successor function s that returns each natural numbers' successor. For all other primitive recursive symbols f, we add a new function symbol $\hat{f}$ to the language of arithmetic. We guarantee that this symbol $\hat{f}$ represents the function f by adding axioms to calculus, depending on how f is defined. For each n-ary projection function P^i_n, which returns its i-th argument, where $1 \leq i \leq n$, we add the following axiom:

$$\frac{}{\vdash \hat{P}^i_n(x_1, \dots, x_n) = x_i}\ \hat{P}^i_n$$

For each function f that is defined as the composition of n-ary primitive recursive g, and n m-ary primitive recursive functions $h_1, \dots, h_n$, we add its definition to the calculus as an axiom:

$$\frac{}{\vdash \hat{f}(x_1, \dots, x_m) = \hat{g}(\hat{h_1}(x_1, \dots, x_m), \dots, \hat{h_n}(x_1, \dots, x_m))}\ \hat{f}$$

For each function f that is defined by primitive recursion with the help of n-ary function g and $n+2$-ary function h, the following two axioms are added to the calculus:

$$\frac{}{\vdash \hat{f}(0, x_1, \dots, x_n) = \hat{g}(x_1, \dots, x_n)}\ \hat{f}0$$

$$\frac{}{\vdash \hat{f}(s(x), x_1, \dots, x_n) = \hat{h}(x_1, \dots, x_n, x, \hat{f}(x, x_1, \dots, x_n))}\ \hat{f}s$$

In the following, we will simply write f, instead of $\hat{f}$ when referring to the function symbol that represents a primitive recursive function f.

Note that the primitive recursive definition for addition is:

$$+(0, y) = P_1^1(y)$$

$$+(s(x), y) = s(P_3^3(y, x, +(x, y)))$$

which results in axioms that are different from the Peano axioms concerning the addition symbol, as projection functions are used to define addition as a primitive recursive function. However, in the extended calculus we can prove that these axioms are equivalent to the axioms of Peano arithmetic concerning the symbol $+$. The same goes for the multiplication function $\cdot$. In general, defining axioms using projection functions are equivalent to axioms that do not use projection functions, where occurrences of $P_n^i(x_1, \ldots, x_n)$ are simply replaced by x_i. It is therefore unnecessary to include the identifiers and axioms for the projection functions in the calculus, whenever we rewrite all axioms to equivalent ones that do not use them.

As another example, consider the exponentiation function, defined as follows:

$$\exp(x, 0) = s(0)$$

$$\exp(x, s(y)) = P_3^1(x, y, \exp(x, y)) \cdot P_3^3(x, y, \exp(x, y))$$

Note that the second equation of the definition is equivalent to the simpler axiom $\exp(x, s(y)) = x \cdot \exp(x, y)$. Therefore, the following axioms are added to the proof calculus:

$$\frac{}{\vdash \exp(x, 0) = s(0)}\ \exp\ 0 \qquad\qquad \frac{}{\vdash \exp(x, s(y)) = x \cdot \exp(x, y)}\ \exp\ s$$

The extension of Peano and Heyting arithmetic with definitions for primitive recursive functions is a conservative extension. This means that whenever a judgment $\Gamma \vdash A$ with no occurrences of function symbols that are not in the original formulation of Peano and Heyting arithmetic is provable in the extended calculus, then it is provable in the original calculus as well. This implies that even when we use the additional axioms to prove the result, as long as the conclusion does not have any occurrences of function symbols apart from 0, s, $+$ and $\cdot$, the judgment will be provable in the original arithmetic too.

We can interpret definitions for primitive recursive functions as term rewrite rules, by simply reading an equality sign as the rewrite symbol $\Rightarrow$. The term rewrite system $\mathcal{R}$ that arises from this is strongly normalizing and has the Church-Rosser property, which implies that every term has a unique normal form. Observe that in this rewrite system, every closed term rewrites to a numeral. As rewriting is a specific form of equality inference, and therefore is simulated in the proof calculus by the equality rules, we have

that whenever a term t_1 rewrites to a term t_2, we can prove $\vdash t_1 = t_2$. Specifically, if a closed term rewrites to a numeral, then we can prove that this is the case.

It often happens while constructing a proof that we want to replace a term t_1 in a formula A by a term t_2, such that t_1 and t_2 have the same normal form in the rewrite system $\mathcal{R}$. To do this, we need a proof π of $\Gamma \vdash t_1 = t_2$ (which always exists, as explained above), and then we apply the equality elimination rule:

$$\frac{\overset{\pi_1}{\vdash t_1 = t_2} \qquad \Gamma \vdash A[t_1/x]}{\Gamma \vdash A[t_2/x]} =\text{e}$$

The rewrite rule is a convenient shortcut for this sort of inferences. It is defined as:

$$\frac{\Gamma \vdash A[t_1/x]}{\Gamma \vdash A[t_2/x]}\,\mathcal{R}$$

where t_1 and t_2 rewrite to the same term in $\mathcal{R}$.

We use the rewrite rule as an abbreviation, but we could in fact equivalently define the extended calculus with the help of just this rewrite rule, instead of adding all the definitions of primitive recursive functions as axioms. The advantage of our current approach is that we do not need an external rewrite system in order to represent the primitive recursive functions - the proof calculus on itself is sufficient.

CHAPTER 3

Proof transformation

By removing the excluded middle rule from Peano arithmetic, we obtain the intuitionistic version of arithmetic, called Heyting arithmetic. As we cannot in general simulate the excluded middle rule using the other rules of the proof calculus, due to its non-constructive nature, the theorems of Heyting arithmetic are a strict subset of the theorems of Peano arithmetic. This is however not the case for Π_2 theorems, which will be shown in this chapter. The result is not only interesting in the scope of program extraction, but has many significant corollaries, such as the direct result that a computable function is provably total in Peano arithmetic if and only if it is provably total in Heyting arithmetic. The Π_2-conservativity result was first shown by Kreisel in [Kre58], but was later proven again by Friedman in [Fri77], who gave the idea for an explicit proof transformation of proofs for Π_2 sentences. We will follow the method of proving Π_2 conservativity suggested by Friedman, by showing how a any classical proof of a theorem of the form $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$, where A is quantifier-free, can be transformed into an intuitionistic proof of the same theorem.

We will start by showing that any proof of a theorem $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$ in Peano arithmetic can be transformed into a proof of a theorem $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q (t = 0)$, for some term t, in Peano arithmetic, and how any proof of this theorem $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q (t = 0)$ in Heyting arithmetic can be transformed to a proof in Heyting arithmetic of the original theorem $\forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$. This normal form for quantifier-free formulas is crucial for the method of proof transformation of Friedman. We then continue with a negative translation (specifically, we use Kolomogorov's translation) that embeds proofs of classical arithmetic into intuitionistic arithmetic. In the last section we describe Friedman's A-translation, whose aim is to remove this embedding from the resulting constructive proofs.

3.1 Normal form for quantifier-free formulas

In this section we will show that every quantifier-free formula is equivalent to an equation of the form $t = 0$, in both Peano and Heyting arithmetic, which we call its normal form. This property results from the extension of the calculus with axioms for primitive recursive functions, and is in fact the reason for introducing this extension in the first place. The idea is to simulate the behavior of propositional connectives in primitive recursive functions. This is possible because the characteristic functions of these connectives are primitive recursive and, within arithmetic, the classical meaning of these connectives, which is captured by the characteristic functions, corresponds to the their intuitionistic meaning.

In the main theorem of this section we will use a specific case of the classical principle of the excluded middle. We will show that the excluded middle principle holds for formulas of the form $t = 0$, for any term t, with the help of the following lemma that implies that the disjunction is not needed in Heyting arithmetic:

Lemma 1. *For all formulas A and B, $\Gamma \vdash A \vee B$ if and only if $\Gamma \vdash \exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))$ in Heyting arithmetic.*

Proof. Let $P(x)$ be an abbreviation for $((x = 0) \to A) \wedge (\neg(x = 0) \to B)$.

Assume that π_1 is a proof for $\Gamma \vdash A \vee B$. The following shows that we then have a proof for $\Gamma \vdash \exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))$

$$\dfrac{\begin{array}{ccc}\pi_1 & \pi_{1L} & \pi_{1R} \\ \Gamma \vdash A \vee B & A \vdash \exists x P(x) & B \vdash \exists x P(x)\end{array}}{\Gamma \vdash \exists x P(x)}$$

where π_{1L} is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \vdash A}\,\text{ax}}{A, 0 = 0 \vdash A}\,\text{wkn}}{A \vdash (0 = 0) \to A}\,{\to}\text{i} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\neg(0 = 0) \vdash \neg(0 = 0)}\,\text{ax} \quad \dfrac{}{\vdash 0 = 0}\,{=}\text{i}}{\neg(0 = 0) \vdash \bot}\,{\neg}\text{e}}{\neg(0 = 0) \vdash B}\,{\bot}\text{e}}{\vdash \neg(0 = 0) \to B}\,{\to}\text{i}}{A \vdash ((0 = 0) \to A) \wedge (\neg(0 = 0) \to B)}\,{\wedge}\text{i}}{A \vdash \exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))}\,{\exists}\text{i}$$

and π_{1R} is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \forall x \neg(s(x) = 0)}\,\text{nat1}}{\vdash \neg(s(0) = 0)}\,\forall\text{e} \qquad \dfrac{}{s(0) = 0 \vdash s(0) = 0}\,\text{ax}}{s(0) = 0 \vdash \bot}\,\neg\text{e}}{s(0) = 0 \vdash A}\,\bot\text{e}}{\vdash (s(0) = 0) \to A}\,\to\text{i} \qquad \dfrac{\dfrac{\dfrac{}{B \vdash B}\,\text{ax}}{B, \neg(s(0) = 0) \vdash B}\,\text{wkn}}{B \vdash \neg(s(0) = 0) \to B}\,\to\text{i}$$

$$\dfrac{\dfrac{\vdash (s(0) = 0) \to A) \wedge (\neg(s(0) = 0) \to B)}{B \vdash ((s(0) = 0) \to A) \wedge (\neg(s(0) = 0) \to B)}\,\wedge\text{i}}{B \vdash \exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))}\,\exists\text{i}$$

Assume that π_2 is a proof for $\Gamma \vdash \exists x((x = 0 \to A) \wedge (\neg(x = 0) \to B))$. The following shows that we have proof for $\Gamma \vdash A \vee B$

$$\dfrac{\Gamma \vdash \exists x P(x) \qquad \dfrac{\dfrac{\vdash P(0) \to (A \vee B) \qquad \dfrac{P(x) \to (A \vee B) \vdash P(s(x)) \to (A \vee B)}{\vdash P(x) \to (A \vee B)} \qquad \dfrac{}{P(x) \vdash P(x)}\,\text{ax}}{P(x) \vdash A \vee B}\,\to\text{e}}{\Gamma \vdash A \vee B}\,\exists\text{e}$$

where π_B is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{P(0) \vdash P(0)}\,\text{ax}}{P(0) \vdash (0 = 0) \to A}\,\wedge\text{e1} \qquad \dfrac{}{\vdash 0 = 0}\,=\text{i}}{P(0) \vdash A}\,\to\text{e}}{P(0) \vdash A \vee B}\,\vee\text{i1}}{\vdash P(0) \to (A \vee B)}\,\to\text{i}$$

and π_I is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(s(x)) \vdash P(s(x))}\,\text{ax}}{P(s(x)) \vdash \neg(s(x) = 0) \to B}\,\wedge\text{e2} \qquad \dfrac{\dfrac{}{\vdash \forall x \neg(s(x) = 0)}\,\text{nat1}}{\vdash \neg(s(x) = 0)}\,\forall\text{e}}{P(s(x)) \vdash B}\,\to\text{e}}{P(x) \to (A \vee B), P(s(x)) \vdash B}\,\text{wkn}}{P(x) \to (A \vee B), P(s(x)) \vdash A \vee B}\,\vee\text{i2}}{P(x) \to (A \vee B) \vdash P(s(x)) \to (A \vee B)}\,\to\text{i}$$

□

Lemma 2. *For all terms t, $(t = 0) \vee \neg(t = 0)$ is a theorem in Heyting arithmetic*

Proof. The result follows easily from Lemma 1. □

We will use this lemma in the proof of the following theorem, which shows how primitive recursive functions can simulate the propositional connectives.

Theorem 1. *For all quantifier-free formulas A, there is a term t such that $\Gamma \vdash A$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash t = 0$ in respectively Peano or Heyting arithmetic.*

Proof. The proof is by induction on the structure of A. We will show the specific term t for which the theorem holds for each induction case, but will only show a proof of it for the case where A is an implication, to demonstrate the usage of Lemma 1.

- Let A be $\bot$. One can easily show that $\Gamma \vdash \bot$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash s(x) = 0$ in Peano or Heyting arithmetic with the help of the axiom $\forall x \neg(s(x) = 0)$.
- Let A be an equation $t_1 = t_2$. Consider the predecessor function p, which reverses the effect of the s constructor, and the subtraction operator $\dot{-}$ over the natural numbers based on that:

$$p(0) = 0$$
$$p(s(x)) = x$$

$$x \dot{-} 0 = x$$
$$x \dot{-} s(y) = p(x \dot{-} y)$$

  With the help of these functions, we can define the difference function as follows:

$$\text{diff}(x, y) = (x \dot{-} y) + (y \dot{-} x)$$

  Given that we are working in the extension of Peano and Heyting arithmetic concerning primitive recursive functions, we assume that function symbols and definitions for these functions are contained in the calculus and rewrite system, as discussed in the previous chapter.

  We then have that $\Gamma \vdash t_1 = t_2$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash \text{diff}(x, y) = 0$ in respectively Peano or Heyting arithmetic.

- Let A be $A_1 \wedge A_2$. We then have that $\Gamma \vdash A_1 \wedge A_2$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash t_1 + t_2 = 0$ in respectively Peano or Heyting arithmetic.

- Let A be $A_1 \vee A_2$. We then have that $\Gamma \vdash A_1 \wedge A_2$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash t_1 \cdot t_2 = 0$ in respectively Peano or Heyting arithmetic.
- Let A be $A_1 \to A_2$. Consider the complement of the signum function, defined as follows:

$$\overline{\mathrm{sgn}}(0) = s(0)0$$
$$\overline{\mathrm{sgn}}(s(y)) = 0$$

We then have that $\Gamma \vdash A_1 \to A_2$ in Peano or Heyting arithmetic if and only if $\Gamma \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0$ in respectively Peano or Heyting arithmetic. We will show here only the left to right direction. Let π_1 be a proof for $\Gamma \vdash A_1 \to A_2$ in Peano or Heyting arithmetic.

Note that from the logical axiom, instantiated with the formula $t_1 = 0$, we get a proof π^{I1} of $t_1 = 0 \vdash A_1$ by the induction hypothesis. With the help of the this proof, we obtain the following:

$$\frac{\begin{array}{cc}\pi_1 & \pi^{I1} \\ \Gamma \vdash A_1 \to A_2 & t_1 = 0 \vdash A_1\end{array}}{\Gamma, t_1 = 0 \vdash A_2} \to\mathrm{e}$$

Then by the induction hypothesis, we obtain a proof π^{I2} for $\Gamma, t_1 = 0 \vdash t_2 = 0$. Let π_{L2} be a proof obtained from Lemma 2 where t is t_1. The following proof then shows that $\Gamma \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0$ in respectively Peano or Heyting arithmetic.

$$\frac{\begin{array}{ccc}\pi_{L2} & \pi_L & \pi_R \\ \vdash (t_1 = 0) \vee \neg(t_1 = 0) & \Gamma, t_1 = 0 \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0 & \Gamma, \neg(t_1 = 0) \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0\end{array}}{\Gamma \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0} \vee\mathrm{e}$$

where π_L is:

$$\frac{\dfrac{\begin{array}{cc}\pi^{I2} & \\ \Gamma, t_1 = 0 \vdash t_2 = 0 & \dfrac{}{\vdash 0 = 0}\,{=}\mathrm{i}\end{array}}{\Gamma, t_1 = 0 \vdash 0 = t_2}\,{=}\mathrm{e} \qquad \dfrac{\dfrac{}{\vdash 0 = 0}\,{=}\mathrm{i}}{\vdash \overline{\mathrm{sgn}}(t_1) \cdot 0 = 0}\,\mathcal{R}}{\Gamma, t_1 = 0 \vdash \overline{\mathrm{sgn}}(t_1) \cdot t_2 = 0}\,{=}\mathrm{e}$$

The proof π_R is then easily obtained with the help of the theorems $\forall x \forall y(x \cdot y = y \cdot x)$ and $\forall x(\neg(x = 0) \to \exists y(x = s(y)))$ of Heyting arithmetic, which state that multiplication is associative and every natural number that is not 0 is the successor of a number.

□

s Note that the theorem, in combination with Lemma 2, implies that for quantifier-free formulas A, $A \vee \neg A$ is a theorem in Heyting arithmetic.

The following corollary is the reason why we are interested in Theorem 1. It implies that we can reduce our problem of showing conservativity for Π_2 sentences to showing conservativity for Π_2 sentences in a normal form such that the quantifier free part of the sentence is always of the form $t = 0$ for some term t.

Corollary 1. *For quantifier-free A, there is a term t such that if $\vdash \forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$ in Peano arithmetic then $\vdash \forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q (t = 0)$ in Peano arithmetic, and if $\vdash \forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q (t = 0)$ in Heyting arithmetic, then $\vdash \forall x_1 \dots \forall x_r \exists y_1 \dots \exists y_q A$ in Heyting arithmetic*

Note that it would have been possible to simplify the problem of Π_2 conservativity even more, by reducing the problem to Π_2 sentences of the form $\forall x \exists y (t = 0)$, as opposed to an arbitrary amount of universal and existential quantifiers. Every prenex sentence is namely equivalent to a prenex sentence with just single consecutive existential and universal quantifiers, which is how the result is usually shown in the literature. The key to this normal form is a primitive recursive, bijective pairing function. We have decided not to use this normal form however, because it does not make the transformation significantly simpler and the effort of introducing another proof transformation therefore does not pay off in this case. As opposed to the normal form discussed in this section, collapsing sequences of the same quantifiers to a single quantifier is not crucial in order to be able to apply Friedman's translation.

3.2 Negative translation: Kolmogorov

Double negation translations, also known as negative translations, are translations of formulas that allow to interpret classical logic in intuitionistic logic, or even minimal logic. Formally, this means that a negative translation is a function f that maps formulas A to classically equivalent formulas $f(A)$, such that if $f(A)$ is provable in classical logic, then it is provable in intuitionistic logic as well. These translations thus show that classical logic can be simulated in intuitionistic logic. One could say that intuitionistic logic is weaker than classical logic, because it lacks the excluded middle rule, but negative translations allow one to see intuitionistic logic as stronger or more expressive than classical logic, as it includes classical reasoning, but makes finer distinctions.

The importance of negative translations not only lies in the scope of showing conservativity of Π_2 theorems in arithmetic, but has far reaching consequences. An important consequence, and the reason why Gödel and Gentzen, independently of each other and Kolmogorov, came up with such translations, is to show that the consistency of Peano arithmetic is reduced to the consistency of Heyting arithmetic. That is, if $\vdash \bot$ in Peano

arithmetic, then $\vdash \bot$ in Heyting arithmetic as well. The reason is that, through a negative translation, one obtains a proof of $\vdash \neg\neg\bot$ in Heyting arithmetic, when given a proof of $\vdash \bot$ in Peano arithmetic. It can be easily seen that $\vdash \neg\neg\bot$ in Heyting arithmetic, then $\vdash \bot$ as well.

Among many different translations, we will use Kolmogorovs's translation to embed Peano arithmetic in Heyting arithmetic, mainly because of its simplicity. Although a good amount of literature can be found about the number of negations that are added to a formula in its interpretation in intuitionistic logic, the complexity of the translation itself in terms of the size of the resulting proofs, seems less often investigated, although in [BL17] it is shown that the Kolmogorov translation for sequent calculus can be computed in polynomial time.

The Kolmogorov translation k applied to a formula A simply puts a double negation in front of every subformula, including the formula A itself. The following definition formalizes this idea.

Definition 1. *The function k for formulas is inductively defined as follows:*

- $k(A) := \neg\neg A$, *where A is atomic*
- $k(A \wedge B) := \neg\neg(k(A) \wedge k(B))$
- $k(A \vee B) := \neg\neg(k(A) \vee k(B))$
- $k(A \to B) := \neg\neg(k(A) \to k(B))$
- $k(\exists x A) := \neg\neg\exists x k(A)$
- $k(\forall x A) := \neg\neg\forall x k(A)$

If we write $k(\Gamma)$, where Γ is $\{C_1, \ldots, C_k\}$, then we mean $\{k(C_1), \ldots k(C_k)\}$.

The following two lemmas state some properties about the function k, that basically boil down to the observation that k does not affect the terms of a formula. We will not prove the lemmas, but this can be done easily by induction on the structure of formulas.

Lemma 3. *For every formula A, the free variables in A are the free variables in $k(A)$.*

Lemma 4. *For all terms t, $k(A[t/x]) = k(A)[t/x]$.*

Note that for every formula A, $k(A)$ begins with a double negation $\neg\neg$. This means that for every formula A, $k(A) = \neg\neg A^k$, where $\cdot^k$ is like the function k but does not put $\neg\neg$ before the formula itself. Formally $\cdot^k$ is defined inductively as follows:

- $A^k := A$, where A is atomic
- $(A_1 \wedge A_2)^k := k(A_1) \wedge k(A_2)$

- $(A \vee B)^k := k(A) \vee k(B)$
- $(A \to B)^k := k(A) \to k(B)$
- $(\exists x A)^k := \exists x k(A)$
- $k(\forall x A) := \forall x k(A)$

The following lemma show that we can use the function k and $\cdot^k$ interchangeably.

Lemma 5. *For every formula A, $k(A) := \neg\neg A^k$.*

Proof. By induction over the structure of A. □

Before showing the proof of the main theorem of this section, we will prove two helpful lemmas. The first lemma basically states that we may always put two negations in front of a formula, and the second lemma states that we can always remove two negations, given that there is a third one. The intuition behind is that a double negated formula $\neg\neg A$ corresponds to the classical truth of the formula A. The first lemma then corresponds to the idea that whenever a formula is constructively true, it is classically true as well, and the second lemma to the idea that if a formula is classically false, it must be constructively false as well (or in other words, no witness can be found for a false formula).

Lemma 6. *For all formula A, $\vdash A \to \neg\neg A$ in Heyting arithmetic.*

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg A \vdash \neg A}\,\mathrm{ax} \quad \dfrac{}{A \vdash A}\,\mathrm{ax}}{\neg A, A \vdash \bot}\,\neg\mathrm{e}}{A \vdash \neg\neg A}\,\neg\mathrm{i}}{\vdash A \to \neg\neg A}\,\to\mathrm{i}$$

□

Lemma 7. *For all formula A, $\vdash \neg\neg\neg A \to \neg A$ in Heyting arithmetic.*

Proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg\neg A \vdash \neg\neg\neg A}\,\mathrm{ax} \quad \dfrac{\dfrac{\dfrac{}{\neg A \vdash \neg A}\,\mathrm{ax} \quad \dfrac{}{A \vdash A}\,\mathrm{ax}}{\neg A, A \vdash \bot}\,\neg\mathrm{e}}{A \vdash \neg\neg A}\,\neg\mathrm{i}}{\neg\neg\neg A, A \vdash \bot}\,\neg\mathrm{e}}{\neg\neg\neg A \vdash \neg A}\,\neg\mathrm{i}}{\vdash \neg\neg\neg A \to \neg A}\,\to\mathrm{i}$$

□

The following theorem shows that classical arithmetic can be simulated in intuitionistic arithmetic, when we embed the formulas in Kolmogorov's translation k.

Theorem 2. *If $\Gamma \vdash A$ in Peano arithmetic, then $k(\Gamma) \vdash k(A)$ in Heyting arithmetic*

Proof. We will show this by induction over proofs in Peano arithmetic. For the cases of the logical axiom schema, and the two structural rules, the result follows immediately as an instance of the same rule. The proofs for the introduction and elimination rules roughly follow the same pattern as the proofs of Lemma 6 and Lemma 7, respectively.

$\wedge$ elimination 1

$$\dfrac{\overset{\pi}{\Gamma \vdash A \wedge B}}{\Gamma \vdash A}\,\wedge\text{e1}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(A \wedge B)$. According to the definition of k, $k(A \wedge B) := \neg\neg(k(A) \wedge k(B))$. According to Lemma 6, $k(A) := \neg\neg A^k$. The following proof shows that $k(\Gamma) \vdash k(A)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\overset{\pi'}{k(\Gamma) \vdash \neg\neg(\neg\neg A^k \wedge k(B))} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\neg\neg A^k \wedge k(B) \vdash \neg\neg A^k \wedge k(B)}\,\text{ax}}{\neg\neg A^k \wedge k(B) \vdash \neg\neg A^k}\,\wedge\text{e1} \qquad \dfrac{}{\neg A^k \vdash \neg A^k}\,\text{ax}}{\neg\neg A^k \wedge k(B), \neg A^k \vdash \bot}\,\neg\text{e}}{\neg A^k \vdash \neg(\neg\neg A^k \wedge k(B))}\,\neg\text{i}}{k(\Gamma), \neg A^k \vdash \bot}\,\neg\text{e}}{k(\Gamma) \vdash \neg\neg A^k}\,\neg\text{i}$$

The case for the second conjunction elimination rule is similar.

$\wedge$ introduction

$$\dfrac{\overset{\pi_1}{\Gamma \vdash A} \qquad \overset{\pi_2}{\Pi \vdash B}}{\Gamma, \Pi \vdash A \wedge B}\,\wedge\text{i}$$

The induction hypothesis states that π_1' and π_2' are proofs in Heyting arithmetic for $k(\Gamma) \vdash k(A)$ and $k(\Pi) \vdash k(B)$ respectively. According to the definition of k, $k(A \wedge B) := \neg\neg(k(A) \wedge k(B))$. The following proof shows that $k(\Gamma), k(\Pi) \vdash k(A \wedge B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{}{\neg(k(A)\wedge k(B))\vdash\neg(k(A)\wedge k(B))}\,\text{ax}\qquad\dfrac{\overset{\pi_1'}{k(\Gamma)\vdash k(A)}\qquad\overset{\pi_2'}{k(\Pi)\vdash k(B)}}{k(\Gamma),k(\Pi)\vdash k(A)\wedge k(B)}\,\wedge\text{i}}{\dfrac{k(\Gamma),k(\Pi),\neg(k(A)\wedge k(B))\vdash\bot}{k(\Gamma),k(\Pi)\vdash\neg\neg(k(A)\wedge k(B))}\,\neg\text{i}}\,\neg\text{e}$$

$\vee$ elimination

$$\dfrac{\overset{\pi_1}{\Gamma\vdash A\vee B}\qquad\overset{\pi_2}{\Pi,A\vdash C}\qquad\overset{\pi_3}{\Delta,B\vdash C}}{\Gamma,\Pi,\Delta\vdash C}\,\vee\text{e}$$

The induction hypothesis states that π_1', π_2' and π_3' are proofs in Heyting arithmetic for $k(\Gamma)\vdash k(A\vee B)$, $k(\Pi),k(A)\vdash k(C)$ and $k(\Delta),k(B)\vdash k(C)$ respectively. According to the definition of k, $k(A\vee B):=\neg\neg(k(A)\vee k(B))$. According to Lemma 6, $k(C):=\neg\neg C^k$. The following proof shows that $k(\Gamma),k(\Pi),k(\Delta)\vdash k(C)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\overset{\pi_1'}{k(\Gamma)\vdash\neg\neg(k(A)\vee k(B))}\qquad\dfrac{\overset{\pi}{k(\Pi),k(\Delta),\neg C^k,k(A)\vee k(B)\vdash\bot}}{k(\Pi),k(\Delta),\neg C^k\vdash\neg(k(A)\vee k(B))}\,\neg\text{i}}{k(\Gamma),k(\Pi),k(\Delta),\neg C^k\vdash\bot}\,\neg\text{i}}{k(\Gamma),k(\Pi),k(\Delta)\vdash\neg\neg C^k}\,\neg\text{i}$$

where π is:

$$\dfrac{\dfrac{}{k(A)\vee k(B)\vdash k(A)\vee k(B)}\,\text{ax}\qquad\overset{\pi_A}{k(\Pi),k(A),\neg C^k\vdash\bot}\qquad\overset{\pi_B}{k(\Delta),k(B)\neg C^k\vdash\bot}}{k(\Pi),k(\Delta),\neg C^k,k(A)\vee k(B)\vdash\bot}\,\vee\text{e}$$

and π_A and π_B are:

$$\dfrac{\overset{\pi_2'}{k(\Pi),k(A)\vdash\neg\neg C^k}\qquad\dfrac{}{\neg C^k\vdash\neg C^k}\,\text{ax}}{k(\Pi),k(A),\neg C^k\vdash\bot}\,\neg\text{e}$$

$$\dfrac{\dfrac{\overset{\pi'_3}{k(\Delta), k(B) \vdash \neg\neg C^k} \qquad \dfrac{}{\neg C^k \vdash \neg C^k}\,\text{ax}}{k(\Delta), k(B), \neg C^k \vdash \bot}\,\neg\text{e}$$

$\vee$ introduction 1

$$\dfrac{\overset{\pi}{\Gamma \vdash A}}{\Gamma \vdash A \vee B}\,\vee\text{i1}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(A)$. According to the definition of k, $k(A \vee B) := \neg\neg(k(A) \vee k(B))$. The following proof shows that $k(\Gamma) \vdash k(A \vee B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{\overset{\pi}{k(\Gamma) \vdash k(A)}}{k(\Gamma) \vdash k(A) \vee k(B)}\,\vee\text{i1} \qquad \dfrac{}{\neg(k(A) \vee k(B)) \vdash \neg(k(A) \vee k(B))}\,\text{ax}}{k(\Gamma), \neg(k(A) \vee k(B)) \vdash \bot}\,\neg\text{e}}{k(\Gamma) \vdash \neg\neg(k(A) \vee k(B)}\,\neg\text{i}$$

The case for the second disjunction introduction elimination rule is similar.

$\to$ elimination

$$\dfrac{\overset{\pi_1}{\Gamma \vdash A \to B} \qquad \overset{\pi_2}{\Pi \vdash A}}{\Gamma, \Pi \vdash B}\,\to\text{e}$$

The induction hypothesis states that π'_1 and π'_2 are proofs in Heyting arithmetic for $k(\Gamma) \vdash k(A \to B)$ and $k(\Pi) \vdash k(A)$ respectively. According to the definition of k, $k(A \to B) := \neg\neg(k(A) \to k(B))$. According to Lemma 6, $k(B) := \neg\neg B^k$. The following proof shows that $k(\Gamma), k(\Pi) \vdash k(B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\overset{\pi'_1}{k(\Gamma) \vdash \neg\neg(k(A) \to \neg\neg B^k)} \qquad \dfrac{\overset{\pi}{k(\Pi), \neg B^k, k(A) \to \neg\neg B^k \vdash \bot}}{k(\Pi), \neg B^k \vdash \neg(k(A) \to \neg\neg B^k)}\,\neg\text{i}}{k(\Gamma), k(\Pi), \neg B^k \vdash \bot}\,\neg\text{e}}{k(\Gamma), k(\Pi) \vdash \neg\neg B^k}\,\neg\text{i}$$

where π is:

$$\dfrac{\dfrac{\dfrac{}{k(A) \to \neg\neg B^k \vdash k(A) \to \neg\neg B^k}\,\mathrm{ax} \qquad \overset{\pi_2'}{k(\Pi) \vdash k(A)}}{k(\Pi), k(A) \to \neg\neg B^k \vdash \neg\neg B^k}\,{\to}\mathrm{e} \qquad \dfrac{}{\neg B^k \vdash \neg B^k}\,\mathrm{ax}}{k(\Pi), \neg B^k, k(A) \to \neg\neg B^k \vdash \bot}\,\neg\mathrm{e}$$

$\to$ introduction

$$\dfrac{\overset{\pi}{A, \Gamma \vdash B}}{\Gamma \vdash A \to B}\,{\to}\mathrm{i}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(A), k(\Gamma) \vdash k(B)$. According to the definition of k, $k(A \to B) := \neg\neg(k(A) \to k(B))$. The following proof shows that $k(\Gamma) \vdash k(A \to B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{\overset{\pi'}{k(A), k(\Gamma) \vdash k(B)}}{k(\Gamma) \vdash k(A) \to k(B)}\,{\to}\mathrm{i} \qquad \dfrac{}{\neg(k(A) \to k(B)) \vdash \neg(k(A) \to k(B))}\,\mathrm{ax}}{k(\Gamma), \neg(k(A) \to k(B)) \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma) \vdash \neg\neg(k(A) \to k(B))}\,\neg\mathrm{i}$$

$\bot$ elimination

$$\dfrac{\overset{\pi}{\Gamma \vdash \bot}}{\Gamma \vdash A}\,\bot\mathrm{e}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(\bot)$. According to the definition of k, $k(\bot) := \neg\neg\bot$. The following proof shows that $k(\Gamma) \vdash k(A)$ in Heyting arithmmetic.

$$\dfrac{\dfrac{\overset{\pi'}{k(\Gamma) \vdash \neg\neg\bot} \qquad \dfrac{\dfrac{}{\bot \vdash \bot}\,\mathrm{ax}}{\vdash \neg\bot}\,\neg\mathrm{i}}{k(\Gamma) \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma) \vdash k(A)}\,\bot\mathrm{e}$$

$\forall$ elimination

$$\dfrac{\begin{array}{c}\pi\\ \Gamma \vdash \forall x A\end{array}}{\Gamma \vdash A[t/x]}\,\forall\mathrm{e}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(\forall x A)$. According to the definition of k, $k(\forall x A) := \neg\neg\forall x \neg\neg A^k$. According to Lemma 5, $k(A[t/x]) := k(A)[t/x]$, and then by Lemma 6, $k(A)[t/x] := (\neg\neg A^k)[t/x]$. Note that we also get that $k(A[t/x]) := \neg\neg(A[t/x])^k$, by Lemma 6. Therefore $(\neg\neg A^k)[t/x] := \neg\neg(A[t/x])^k$, which explain what happens in the topmost negation elimination in the following proof that shows that $k(\Gamma) \vdash k(A[t/x])$ in Heyting arithmetic.

$$\dfrac{\begin{array}{c}\pi'\\ k(\Gamma) \vdash \neg\neg\forall x \neg\neg A^k\end{array} \qquad \dfrac{\dfrac{\dfrac{}{\forall x \neg\neg A^k \vdash \forall x \neg\neg A^k}\,\mathrm{ax}}{\forall x \neg\neg A^k \vdash (\neg\neg A^k)[t/x]}\,\forall\mathrm{e} \qquad \dfrac{}{\neg(A[t/x])^k \vdash \neg(A[t/x])^k}\,\mathrm{ax}}{\forall x \neg\neg A^k, \neg(A[t/x])^k \vdash \bot}\,\neg\mathrm{e}}{\neg(A[t/x])^k \vdash \neg\forall x \neg\neg A^k}\,\neg\mathrm{i}}{k(\Gamma), \neg(A[t/x])^k \vdash \bot}\,\neg\mathrm{e}$$
$$\dfrac{}{k(\Gamma) \vdash \neg\neg(A[t/x])^k}\,\neg\mathrm{i}$$

$\forall$ introduction

$$\dfrac{\begin{array}{c}\pi\\ \Gamma \vdash A[\alpha/x]\end{array}}{\Gamma \vdash \forall x A}\,\forall\mathrm{i}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(A[\alpha/x])$. According to the definition of k, $k(\forall x A) := \neg\neg\forall x k(A)$. According to Lemma 5, $k(A[\alpha/x]) := k(A)[\alpha/x]$. The following proof shows that $k(\Gamma) \vdash k(\forall x A)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{}{\neg\forall x k(A) \vdash \neg\forall x k(A)}\,\mathrm{ax} \qquad \dfrac{\begin{array}{c}\pi'\\ k(\Gamma) \vdash k(A)[\alpha/x]\end{array}}{k(\Gamma) \vdash \forall x k(A)}\,\forall\mathrm{i}}{k(\Gamma), \neg\forall x k(A) \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma) \vdash \neg\neg\forall x k(A)}\,\neg\mathrm{i}$$

By Lemma 4 we get that the eigenvariable condition posed by the $\forall$ introduction rule is satisfied.

∃ elimination

$$\dfrac{\dfrac{\pi_1}{\Gamma \vdash \exists x A} \qquad \dfrac{\pi_2}{A[\alpha/x], \Pi \vdash B}}{\Gamma, \Pi \vdash B}\,\exists\mathrm{e}$$

The induction hypothesis states that π_1' and π_2' are proofs in Heyting arithmetic for $k(\Gamma) \vdash k(\exists x A)$ and $k(A[\alpha/x]), k(\Pi) \vdash k(B)$ respectively. According to the definition of k, $k(\exists x A) := \neg\neg\exists x k(A)$. According to Lemma 6, $k(B) := \neg\neg B^k$, and according to Lemma 5, $k(A[\alpha/x]) := k(A)[\alpha/x]$. The following proof shows that $k(\Gamma), k(\Pi) \vdash k(B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\pi_1'}{k(\Gamma) \vdash \neg\neg\exists x k(A)} \qquad \dfrac{\dfrac{\overset{\pi}{k(\Pi), \neg B^k, \exists x k(A) \vdash \bot}}{k(\Pi), \neg B^k \vdash \neg\exists x k(A)}\,\neg\mathrm{i}}{k(\Gamma), k(\Pi), \neg B^k \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma), k(\Pi) \vdash \neg\neg B^k}\,\neg\mathrm{i}$$

where π is:

$$\dfrac{\dfrac{\dfrac{}{\exists x k(A) \vdash \exists x k(A)}\,\mathrm{ax} \qquad \overset{\pi_2'}{k(A)[\alpha/x], k(\Pi) \vdash \neg\neg B^k}}{\exists x k(A), k(\Pi) \vdash \neg\neg B^k}\,\exists\mathrm{e} \qquad \dfrac{}{\neg B^k \vdash \neg B^k}\,\mathrm{ax}}{k(\Pi), \neg B^k, \exists x k(A) \vdash \bot}\,\neg\mathrm{e}$$

By Lemma 4 we get that the eigenvariable condition posed by the ∃ elimination rule is satisfied.

∃ introduction

$$\dfrac{\overset{\pi}{\Gamma \vdash A[t/x]}}{\Gamma \vdash \exists x A}\,\exists\mathrm{i}$$

The induction hypothesis states that π' is a proof in Heyting arithmetic for $k(\Gamma) \vdash k(A[t/x])$. According to the definition of k, $k(\exists x A) := \neg\neg\exists x k(A)$. According to Lemma 5, $k(A[t/x]) := k(A)[t/x]$. The following proof shows that $k(\Gamma) \vdash k(\exists x A)$ in Heyting arithmetic.

$$\dfrac{\dfrac{}{\neg\exists x k(A) \vdash \neg\exists x k(A)}\,\text{ax} \qquad \dfrac{\dfrac{\overset{\pi'}{k(\Gamma) \vdash k(A)[t/x]}}{k(\Gamma) \vdash \exists x k(A)}\,\exists\text{i}}{k(\Gamma), \neg\exists x k(A) \vdash \bot}\,\neg\text{e}}{k(\Gamma) \vdash \neg\neg\exists x k(A)}\,\neg\text{i}$$

= elimination

$$\dfrac{\overset{\pi_1}{\Gamma \vdash t_1 = t_2} \qquad \overset{\pi_2}{\Pi \vdash A[t_1/x]}}{\Gamma, \Pi \vdash A[t_2/x]}\,=\text{e}$$

The induction hypothesis states that π_1' and π_2' are proofs in Heyting arithmetic for $k(\Gamma) \vdash k(t_1 = t_2)$ and $k(\Pi) \vdash k(A[t_1/x])$ respectively. According to the definition of k, $k(t_1 = t_2) := \neg\neg(t_1 = t_2)$. According to Lemma 6, $k(A[t_1/x]) := \neg\neg(A[t_1/x])^k$. Note that $k(A[t_2/x]) := \neg\neg(A[t_2/x])^k$ as well. The following proof shows that $k(\Gamma), k(\Pi) \vdash k(A[t_2/x])$ in Heyting arithmetic.

$$\dfrac{\dfrac{\overset{\pi_1'}{k(\Gamma) \vdash \neg\neg(t_1 = t_2)} \qquad \dfrac{\dfrac{\overset{\pi_2'}{k(\Pi) \vdash \neg\neg(A[t_1/x])^k} \qquad \dfrac{\overset{\pi}{t_1 = t_2, \neg(A[t_2/x])^k \vdash \neg(A[t_1/x])^k}}{k(\Pi), \neg(A[t_2/x])^k, t_1 = t_2 \vdash \bot}\,\neg\text{i}}{k(\Pi), \neg(A[t_2/x])^k \vdash \neg(t_1 = t_2)}\,\neg\text{e}}{k(\Gamma), k(\Pi), \neg(A[t_2/x])^k \vdash \bot}\,\neg\text{e}}{k(\Gamma), k(\Pi) \vdash \neg\neg(A[t_2/x])^k}\,\neg\text{i}$$

where π is:

$$\dfrac{\dfrac{\dfrac{}{t_1 = t_2 \vdash t_1 = t_2}\,\text{ax} \qquad \dfrac{}{\vdash t_1 = t_1}\,=\text{i}}{t_1 = t_2 \vdash t_2 = t_1}\,=\text{e} \qquad \dfrac{}{\neg(A[t_2/x])^k \vdash \neg(A[t_2/x])^k}\,\text{ax}}{t_1 = t_2, \neg(A[t_2/x])^k \vdash \neg(A[t_1/x])^k}\,=\text{e}$$

= introduction

$$\dfrac{}{\vdash t = t}\,=\text{i}$$

According to the definition of k, $k(t = t) := \neg\neg(t = t)$. Let π be the proof of Lemma 7, instantiated with the formula $t = t$. The following proof shows that $\vdash k(t = t)$ in Heyting arithmetic.

$$\dfrac{\overset{\pi}{\vdash (t = t) \to \neg\neg(t = t)} \qquad \dfrac{}{\vdash t = t}\,{=}\mathrm{i}}{\vdash \neg\neg(t = t)}\,{\to}\mathrm{e}$$

Induction

$$\dfrac{\overset{\pi_1}{\Gamma \vdash A[0/x]} \qquad \overset{\pi_2}{A[\alpha/x], \Pi \vdash A[s(\alpha)/x]}}{\Gamma, \Pi \vdash A[t/x]}\,\mathrm{ind}$$

The induction hypothesis states that π_1' and π_2' are proofs in Heyting arithmetic for $k(\Gamma) \vdash k(A[0/x])$, and $k(A[\alpha/x]), k(\Pi) \vdash k(A[s(\alpha)/x])$ respectively. By observing that $k(A[t/x]) := k(A)[t/x]$ for any term t, according to Lemma 5, we immediately obtain an instance of the induction rule:

$$\dfrac{\overset{\pi_1'}{k(\Gamma) \vdash k(A)[0/x]} \qquad \overset{\pi_1'}{k(A)[\alpha/x], k(\Pi) \vdash k(A)[s(\alpha)/x]}}{k(\Gamma), k(\Pi) \vdash k(A)[t/x]}\,\mathrm{ind}$$

By Lemma 4 we get that the eigenvariable condition is still satisfied.

Natural numbers

$$\dfrac{}{\vdash \forall x \neg(s(x) = 0)}\,\mathrm{nat1}$$

$$\dfrac{}{\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))}\,\mathrm{nat2}$$

We will not show the cases for the nat1 and nat2 axioms. One can obtain proofs for $\vdash k(\forall x \forall y((s(x) = s(y)) \to (x = y)))$ and $\vdash k(\neg\forall x(s(x) = 0))$ in Heyting arithmetic, with the help of Lemma 5 and Lemma 6.

Primitive recursive definitions

Any axiom for a primitive recursive definition will be of the form $\vdash \forall x_1 \ldots \forall x_r(t_1 = t_2)$, for some terms t_1 and t_2, with free variables $x_1, \ldots, x_r$. We will not show how a proof for $\vdash k(\forall x_1 \ldots \forall x_r(t_1 = t_2))$ looks like, but it can be easily seen that this is a result of Lemma 5.

Excluded middle

$$\dfrac{\overset{\pi_1}{\Gamma, A \vdash B} \qquad \overset{\pi_2}{\Pi, \neg A \vdash B}}{\Gamma, \Pi \vdash B}\,\mathrm{em}$$

The induction hypothesis states that π_1' and π_2' are proofs in Heyting arithmetic for $k(\Gamma), k(A) \vdash k(B)$ and $k(\Pi), k(\neg A) \vdash k(B)$ respectively. According to the definition of k, $k(\neg A) := \neg\neg\neg k(A)$. According to Lemma 6, $k(A) := \neg\neg A^k$. The following proof shows that $k(\Gamma), k(\Pi) \vdash k(B)$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overset{\pi}{k(\Pi), \neg\neg\neg A^k \vdash \neg\neg B^k} \qquad \dfrac{}{\neg B^k \vdash \neg B^k}\,\mathrm{ax}}{k(\Pi), \neg\neg\neg A^k, \neg B^k \vdash \bot}\,\neg\mathrm{e}}{k(\Pi), \neg B^k \vdash \neg\neg\neg\neg A^k}\,\neg\mathrm{i} \qquad \dfrac{\dfrac{\dfrac{\overset{\pi_1'}{k(\Gamma), \neg\neg A^k \vdash \neg\neg B^k} \qquad \dfrac{}{\neg B^k \vdash \neg B^k}\,\mathrm{ax}}{k(\Gamma), \neg\neg A^k, \neg B^k \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma), \neg B^k \vdash \neg\neg\neg\neg A^k}\,\neg\mathrm{i}}{k(\Gamma), k(\Pi), \neg B^k \vdash \bot}\,\neg\mathrm{e}}{k(\Gamma), k(\Pi) \vdash \neg\neg B^k}\,\neg\mathrm{i}$$

where π is obtained from the proof π_2' of $k(\Pi), \neg\neg\neg\neg\neg A^k \vdash \neg\neg B^k$ and Lemma 8. □

3.3 Friedman's A-translation

As mentioned before, Friedman's translation was introduced to give a purely syntactical proof of Peano arithmetic being a conservative extension of Heyting arithmetic for Π_2 sentences. Specifically, the only step in showing Π_2 conservativity for which no syntactical argument was known is a proof of the closure of Heyting arithmetic under Markov's rule, which states that $\vdash \forall x \neg\neg \exists y(f(x, y) = 0)$ whenever $\vdash \forall x \exists y(f(x, y) = 0)$. In this section we will show how the proof transformation reduces to a syntactical proof for Markov's rule and how the Friedman translation solves this.

First note that for any formula A we can easily go from a proof for $\vdash \forall x_1 \ldots \forall x_r A$ to a proof for $\vdash A$, by applying the $\forall$ elimination rule r times, choosing the quantified variable itself as the term to be substituted. Similarly, we can go from a proof of $\vdash A$ to a proof of $\vdash \forall x_1 \ldots \forall x_r A$, by applying the $\forall$ introduction rule r times to the conclusion - which, because of the empty antecedent, satisfies the eigenvariable condition as required. We therefore reduce our problem of transforming a proof for $\forall x_1 \ldots \forall x_r \exists y_1 \ldots \exists y_q(t = 0)$ to transforming a proof for $\exists y_1 \ldots \exists y_q(t = 0)$.

Consider a proof in Peano arithmetic of $\exists y_1 \ldots \exists y_q(t = 0)$. By Kolmogorov's translation we then obtain a proof of $k(\exists y_1 \ldots \exists y_q(t = 0))$ in Heyting arithmetic. According to the

definition of k, this means that we have a proof for $\neg\neg\exists y_1 \dots \neg\neg\exists y_q \neg\neg(t = 0)$. We can transform this into a proof for $\neg\neg\exists y_1 \dots \exists y_q (t = 0)$, with the help of the following lemma:

Lemma 8. *For any formula A, $\neg\neg\exists y_1 \dots \neg\neg\exists y_r \neg\neg A \vdash \neg\neg\exists y_1 \dots \exists y_r A$ in Heyting arithmetic.*

Proof. We will show this by induction over the amount of occurrences of an existential quantifier followed by a double negation. We will first rename the bound variables such that we count the quantified variables from right to left instead from left to right. This results in an α-equivalent formula, as we are merely renaming bound variables. Precisely, we will show that $\neg\neg\exists y_r \dots \neg\neg\exists y_1 \neg\neg A \vdash \neg\neg\exists y_r \dots \exists y_1 A$.

The base case is trivial. The following proof shows that $\neg\neg A \vdash \neg\neg A$:

$$\frac{}{\neg\neg A \vdash \neg\neg A}\,\text{ax}$$

The induction hypothesis states that we have a proof π in Heyting arithmetic for $\neg\neg\exists y_r \dots \neg\neg\exists y_1 \neg\neg A \vdash \neg\neg\exists y_r \dots \exists y_1 A$. We will present a proof, that is made with the help of π, for the judgment $\neg\neg\exists y_{r+1} \neg\neg\exists y_r \dots \neg\neg\exists y_1 \neg\neg A \vdash \neg\neg\exists y_{r+1} \exists y_r \dots \exists y_1 A$. Let B be an abbreviation for $\exists y_r \neg\neg \dots \exists y_1 \neg\neg A$ and let C be an abbreviation for the formula $\exists y_r \dots \exists y_1 A$. We thus need to show a proof for $\neg\neg\exists y_{r+1} \neg\neg B \vdash \neg\neg\exists y_{r+1} C$, with the help of a proof π for $\neg\neg B \vdash \neg\neg C$. Consider first the following proof π_1:

$$\dfrac{\dfrac{}{\exists y_{r+1}\neg\neg B \vdash \exists y_{r+1}\neg\neg B}\,\text{ax} \qquad \dfrac{\dfrac{\pi}{\neg\neg B \vdash \neg\neg C} \qquad \dfrac{\dfrac{\dfrac{}{\neg\exists y_{r+1} C \vdash \neg\exists y_{r+1} C}\,\text{ax} \qquad \dfrac{\dfrac{}{C \vdash C}\,\text{ax}}{C \vdash \exists y_{r+1} C}\,\exists\text{i}}{\neg\exists y_{r+1} C, C \vdash \bot}\,\neg\text{e}}{\neg\exists y_{r+1} C \vdash \neg C}\,\neg\text{i}}{\neg\neg B, \neg\exists y_{r+1} C \vdash \bot}\,\neg\text{e}}{\exists y_{r+1}\neg\neg B, \neg\exists y_{r+1} C \vdash \bot}\,\exists\text{e}$$

Note that the eigenvariable condition imposed by the $\exists$ elimination rule is satisfied, because the variable y_{r+1} does not occur free in $\neg\exists y_{r+1} C$. The following proof then concludes the lemma:

$$\dfrac{\dfrac{\dfrac{}{\neg\neg\exists y_{r+1}\neg\neg B \vdash \neg\neg\exists y_{r+1}\neg\neg B}\,\text{ax} \qquad \dfrac{\dfrac{\pi_1}{\neg\exists y_{r+1} C, \exists y_{r+1}\neg\neg B \vdash \bot}}{\neg\exists y_{r+1} C \vdash \neg\exists y_{r+1}\neg\neg B}\,\neg\text{i}}{\neg\neg\exists y_{r+1}\neg\neg B, \neg\exists y_{r+1} C \vdash \bot}\,\neg\text{e}}{\neg\neg\exists y_{r+1}\neg\neg B \vdash \neg\neg\exists y_{r+1} C}\,\neg\text{i}$$

□

We now have a proof for $\neg\neg\exists y_1 \dots \exists y_q(t = 0)$. What is left is a proof transformation resulting in a proof for $\vdash \exists y_1 \dots \exists y_q(t = 0)$. This requires the closure of Heyting arithmetic under Markov's rule and is indeed where Friedman's A-translation comes into play.

Friedman's translation of formulas is a surprisingly simple trick, that simply replaces every atom by a disjunction of that atom and any arbitrary chosen formula. Considering an arbitrary formula φ, the translation $\cdot^\varphi$ applied to a formula A is the result of replacing every atomic subformula B of A by $B \vee \varphi$. The variables in φ are not supposed to be bound by quantifiers in the formula A in this process, or in other words, whenever we consider e.g. $(\forall x A)^\varphi$, we must assume that x is not free in φ.

It is important to keep in mind that $(\neg A)^\varphi$ is not equal to $\neg A^\varphi$, but instead to $A^\varphi \to \bot^\varphi$ as $\neg A$ is an abbreviation of the formula $A \to \bot$. As $\bot$ is an atom as well, $\bot^\varphi$ is equal to $\bot \vee \varphi$, which is equivalent to simply φ. We can thus replace every occurrence of $\bot^\varphi$ by φ, and therefore every occurrence of $(\neg A)^\varphi$ by $A^\varphi \to \varphi$.

Our aim is to show that whenever $\Gamma^\varphi \vdash A$ is provable, then $\Gamma^\varphi \vdash A^\varphi$ is provable as well. In order to do this, we will need the following lemma:

Lemma 9. *For all formulas A and φ, $\varphi \vdash A^\varphi$ in Heyting arithmetic.*

Proof. By induction over the structure of A. □

In the following we will refer to a proof for $\varphi \vdash A^\varphi$ as $\pi[\varphi, A]$.

Theorem 3. *For any formula φ, if $\Gamma^\varphi \vdash A$ in Heyting arithmetic, then $\Gamma^\varphi \vdash A^\varphi$ in Heyting arithmetic.*

Proof. We will show this by induction over proofs in Heyting arithmetic. For most cases, the result follows immediately as an instance of the same rule. This is because the translation $\cdot^\varphi$ only affects the atoms of the formula that it is applied on. We will only present those cases for which something else needs to happen to obtain a proof of $\Gamma^\varphi \vdash A^\varphi$. It should be noted however that whenever a rule requires an eigenvariable condition, and this eigenvariable is present in the formula φ, the eigenvariable condition could be invalidated in the proof of the translated proof. This is however easily fixed by simply substituting the eigenvariable for a fresh eigenvariable that does not occur in φ in all subproofs that the rule is applied on.

$\bot$ elimination

$$\dfrac{\begin{matrix}\pi \\ \Gamma \vdash \bot\end{matrix}}{\Gamma \vdash A}\ \bot\mathrm{e}$$

The induction hypothesis states that π' is a proof of $\Gamma^\varphi \vdash \bot^\varphi$. According to the definition of $\cdot^\varphi$, $\bot^\varphi := \varphi$. The following proof shows that $\Gamma^\varphi \vdash A^\varphi$ in Heyting arithmetic.

$$\dfrac{\dfrac{\begin{array}{c}\pi[\varphi, A]\\ \varphi \vdash A^{\varphi}\end{array}}{\vdash \varphi \to A^{\varphi}} \to\text{i} \qquad \begin{array}{c}\pi'\\ \Gamma^{\varphi} \vdash \varphi\end{array}}{\Gamma^{\varphi} \vdash A^{\varphi}} \to\text{e}$$

= elimination

$$\dfrac{\begin{array}{c}\pi_1\\ \Gamma \vdash t_1 = t_2\end{array} \qquad \begin{array}{c}\pi_2\\ \Pi \vdash A[t_1/x]\end{array}}{\Gamma, \Pi \vdash A[t_2/x]} =\text{e}$$

The induction hypothesis states that π_1 and π_2' are proofs for $\Gamma^{\varphi} \vdash (t_1 = t_2)^{\varphi}$ and $\Pi^{\varphi} \vdash (A[t_1/x])^{\varphi}$ respectively.

Note that for any formula A and term t, $(A[t/x])^{\varphi}$ is not necessarily the same as $A^{\varphi}[t/x]$, if we don't have a guarantee that x does not occur in φ. In the case that φ contains occurrences of x, the substitution $[t/x]$ also works on φ in the case of $A^{\varphi}[t/x]$, but will leave φ unchanged in the case of $(A[t/x])^{\varphi}$.

According to the definition of $\cdot^{\varphi}$, $(t_1 = t_2)^{\varphi} := (t_1 = t_2) \vee \varphi$. Let y be a variable that does not occur yet in A or φ. Then $(A[t_1/x])^{\varphi} := A[y/x]^{\varphi}[t_1/y]$, and $(A[t_2/x])^{\varphi} := A[y/x]^{\varphi}[t_2/y]$. The following proof shows that $\Gamma^{\varphi}, \Pi^{\varphi} \vdash (A[t_2/x])^{\varphi}$ in Heyting arithmetic.

$$\dfrac{\begin{array}{c}\pi_1'\\ \Gamma^{\varphi} \vdash (t_1 = t_2) \vee \varphi\end{array} \qquad \dfrac{\dfrac{}{t_1 = t_2 \vdash t_1 = t_2}\,\text{ax} \qquad \begin{array}{c}\pi_2'\\ \Pi^{\varphi} \vdash A[y/x]^{\varphi}[t_1/y]\end{array}}{\Pi^{\varphi}, t_1 = t_2 \vdash A[y/x]^{\varphi}[t_2/y]} =\text{e} \qquad \begin{array}{c}\pi[\varphi, A[t_1/x]]\\ \varphi \vdash A[y/x]^{\varphi}[t_2/y]\end{array}}{\Gamma^{\varphi}, \Pi^{\varphi} \vdash A[y/x]^{\varphi}[t_2/y]} \vee\text{e}$$

= introduction

$$\dfrac{}{\vdash t = t} =\text{i}$$

According to the definition of $\cdot^{\varphi}$, $(t = t)^{\varphi} := (t = t) \vee \varphi$. The following proof shows that $\vdash (t = t)^{\varphi}$ in Heyting Arithmetic.

$$\dfrac{\dfrac{}{\vdash t = t} =\text{i}}{\vdash (t = t) \vee \varphi} \vee\text{i}$$

Induction

$$\dfrac{\overset{\pi_1}{\Gamma \vdash A[0/x]} \qquad \overset{\pi_2}{A[\alpha/x], \Pi \vdash A[s(\alpha)/x]}}{\Gamma, \Pi \vdash A[t/x]}\,\text{ind}$$

The induction hypothesis states that π_1' and π_2' are proofs for $\Gamma^\varphi \vdash (A[0/x])^\varphi$ and $(A[\alpha/x])^\varphi, \Pi^\varphi \vdash (A[s(\alpha)/x])^\varphi$ respectively.

Let y be a fresh variable that does not occur yet in A or in φ. Note that $(A[t/x])^\varphi$ is the same formula as $A[y/x]^\varphi[t/y]$. As y does not occur in φ, the formula φ is left unchanged, as in $(A[t/x])^\varphi$. As y does not occur yet in A, all free occurrences of y in $A[y/x]$ are exactly on the same places as the free occurrences of x in A itself. Therefore the $A[y/x][t/y]$ and $A[t/x]$ have the same effect. We therefore get an instance of the induction rule again, but with a different substituted variable:

$$\dfrac{\overset{\pi_1'}{\Gamma^\varphi \vdash A[y/x]^\varphi[0/y]} \qquad \overset{\pi_2'}{A[y/x]^\varphi[\alpha/y], \Pi^\varphi \vdash A[y/x]^\varphi[s(\alpha)/y]}}{\Gamma^\varphi, \Pi^\varphi \vdash A[y/x]^\varphi[t/y]}\,\text{ind}$$

Natural numbers

$$\dfrac{}{\vdash \forall x \neg(s(x) = 0)}\,\text{nat1}$$

According to the definition of $\cdot^\varphi$, $(\forall x \neg(s(x) = 0))^\varphi := \forall x((s(x) = 0) \vee \varphi) \to \varphi)$. The following proof shows that $\vdash (\forall x \neg(s(x) = 0))^\varphi$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(s(x) = 0) \vee \varphi \vdash (s(x) = 0) \vee \varphi}\,\text{ax} \qquad \overset{\pi_1}{s(x) = 0 \vdash \varphi} \qquad \dfrac{}{\varphi \vdash \varphi}\,\text{ax}}{(s(x) = 0) \vee \varphi \vdash \varphi}\,\vee\text{e}}{\vdash (s(x) = 0) \vee \varphi) \to \varphi}\,\to\text{i}}{\vdash \forall x((s(x) = 0) \vee \varphi) \to \varphi)}\,\forall\text{i}$$

where π_1 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \forall x \neg(s(x) = 0)}\,\text{nat1}}{\vdash \neg(s(x) = 0)}\,\forall\text{e} \qquad \dfrac{}{s(x) = 0 \vdash s(x) = 0}\,\text{ax}}{s(x) = 0 \vdash \bot}\,\neg\text{e}}{s(x) = 0 \vdash \varphi}\,\bot\text{e}$$

$$\frac{}{\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))}\,\text{nat2}$$

According to the definition of $\cdot^\varphi$, $(\forall x \forall y((s(x) = s(y)) \to (x = y)))^\varphi := \forall x \forall y(((s(x) = s(y)) \vee \varphi) \to ((x = y) \vee \varphi))$. The following proof shows that $\vdash (\forall x \forall y((s(x) = s(y)) \to (x = y)))^\varphi$ in Heyting arithmetic.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(s(x) = s(y)) \vee \varphi \vdash (s(x) = s(y)) \vee \varphi}\,\text{ax} \qquad \pi_1 \quad s(x) = s(y) \vdash (x = y) \vee \varphi \qquad \dfrac{\dfrac{}{\varphi \vdash \varphi}\,\text{ax}}{\varphi \vdash (x = y) \vee \varphi}\,\vee\text{i2}}{(s(x) = s(y)) \vee \varphi \vdash (x = y) \vee \varphi}\,\vee\text{e}}{\vdash ((s(x) = s(y)) \vee \varphi) \to ((x = y) \vee \varphi)}\,\to\text{i}}{\vdash \forall y(((s(x) = s(y)) \vee \varphi) \to ((x = y) \vee \varphi))}\,\forall\text{i}}{\vdash \forall x \forall y(((s(x) = s(y)) \vee \varphi) \to ((x = y) \vee \varphi))}\,\forall\text{i}$$

where π_2 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))}\,\text{nat2}}{\vdash \forall y((s(x) = s(y)) \to (x = y))}\,\forall\text{e}}{\vdash (s(x) = s(y)) \to (x = y)}\,\forall\text{e} \qquad \dfrac{}{s(x) = s(y) \vdash s(x) = s(y)}\,\text{ax}}{s(x) = s(y) \vdash x = y}\,\to\text{e}}{s(x) = s(y) \vdash (x = y) \vee \varphi}\,\vee\text{i1}$$

Primitive recursive definitions

Assume that the proof for $\Gamma \vdash A$ is a defining axiom. Any axiom for a primitive recursive definition of f will be of the form $\vdash \forall x_1 \ldots \forall x_r(t_1 = t_2)$, for some terms t_1 and t_2, with free variables $x_1, \ldots, x_r$. A proof for $\vdash (\forall x_1 \ldots \forall x_r(t_1 = t_2))^\varphi$ is obtained by applying the $\forall$ elimination r times to the axiom $\vdash \forall x_1 \ldots \forall x_r(t_1 = t_2)$, then $\vee$ introduction with φ and universally close the formula again with applications of the $\forall$ introduction rule.

□

Theorem 3, with $\exists y_1 \ldots \exists y_q(t = 0)$ as formula φ, allows us to transform a proof for $\vdash \neg\neg\exists y_1 \ldots \exists y_q(t = 0)$ to a proof of $\vdash (\neg\neg\exists y_1 \ldots y_q(t = 0))^{\exists y_1 \ldots \exists y_q(t=0)}$, i.e. a proof of $\vdash (\exists y_1 \ldots \exists y_q((t = 0) \vee \exists y_1 \ldots \exists y_q(t = 0)) \to \exists y_1 \ldots \exists y_q(t = 0)) \to \exists y_1 \ldots \exists y_q(t = 0)$, by definition of $\cdot^\varphi$. As soon as we have a proof for $\vdash \exists y_1 \ldots \exists y_q((t = 0) \vee \exists y_1 \ldots \exists y_q(t = 0)) \to \exists y_1 \ldots \exists y_q(t = 0)$ in Heyting arithmetic, we will then have, by an application of the $\to$ elimination rule, have the desired proof for $\vdash \exists y_1 \ldots \exists y_q(t = 0)$, which finishes the proof transformation from Peano to Heyting arithmetic for Π_2 theorems. The following lemma provides us with this last step.

Lemma 10. *For all t, $\vdash \exists y_1 \dots \exists y_q((t = 0) \vee \exists y_1 \dots \exists y_q(t = 0)) \to \exists y_1 \dots \exists y_q(t = 0)$ in Heyting arithmetic*

Proof. Note that we obtain a proof for $t = 0 \vdash \exists y_1 \dots \exists y_q(t = 0)$ by repeatedly applying the $\exists$ introduction rule to the logical axiom instantiated with $t = 0$. We will prove that for all sentences A, whenever we have a proof π for $t = 0 \vdash A$, then for all t, $\vdash \exists y_1 \dots \exists y_q((t = 0) \vee A) \to A$ in Heyting arithmetic, which gives indeed the desired result, when A is instantiated with $\exists y_1 \dots \exists y_q(t = 0)$.

We prove this by induction over the number of existential quantifiers. First we will rename the bound variables, similar as in the proof of Lemma 7, such that we can count quantifiers from right to left instead from left to right. In other words, we will show a proof for $\vdash \exists y_q \dots \exists y_1((t = 0) \vee A) \to A$.

The base case, where q is 0, is proven as follows:

$$\dfrac{\dfrac{\dfrac{}{t = 0 \vee A \vdash t = 0 \vee A}\,\text{ax} \qquad \overset{\pi}{\underset{t = 0 \vdash A}{}} \qquad \dfrac{}{A \vdash A}\,\text{ax}}{(t = 0) \vee A \vdash A}\,\vee\text{e}}{\vdash ((t = 0) \vee A) \to A}\,\to\text{i}$$

The induction hypothesis gives a proof π^I of $\vdash \exists y_q \dots \exists y_1((t = 0) \vee A) \to A$. We use it to show that $\vdash \exists y_{q+1} \exists y_q \dots \exists y_1((t = 0) \vee A) \to A$. Let B be an abbreviation for $\exists y_q \dots \exists y_1((t = 0) \vee A)$ in the following proof:

$$\dfrac{\dfrac{\dfrac{}{\exists y_{q+1} B \vdash \exists y_{q+1} B}\,\text{ax} \qquad \dfrac{\overset{\pi_I}{\underset{\vdash B \to A}{}} \qquad \dfrac{}{B \vdash B}\,\text{ax}}{B \vdash A}\,\to\text{e}}{\exists y_{q+1} \exists y_q \dots \exists y_1 B \vdash A}\,\exists\text{e}}{\vdash \exists y_{q+1} B \to A}\,\to\text{i}$$

We know that the eigenvariable condition imposed by the $\exists$ elimination rule is satisfied, by the assumption that A is a sentence.

□

CHAPTER 4

Program extraction

This chapter describes the extraction of programs from proofs in Heyting arithmetic. We will start by discussing system **T**, an extension of the simply typed lambda calculus which will serve as the programming language for the extraction. Then we discuss the definition of modified realizability, a relation between sentences of arithmetic and terms of system **T**, that basically describes how we can see the terms of system **T** as witnesses for the intuitionistic truth of sentences. The idea that system **T** suffices as a language that justifies all theorems of Heyting arithmetic, and can thus be seen as a model for Heyting arithmetic, is supported by the main theorem of this chapter, where we show that all theorems of Heyting arithmetic are m-realized by some program in system **T**.

We are specifically interested in m-realizers for theorems of the form $\forall x_1 \dots \forall x_r \exists y A$, where A is a Σ_1 specification, because they describe a function satisfying the specification A, which we will prove as well in this chapter. The precise extraction algorithm of an m-realizer from a proof is shown by the proof of the main theorem that states that all theorems are m-realizable. This implies that whenever we have a proof for $\forall x_1 \dots \forall x_r \exists y A$ in Heyting arithmetic, we can extract a program from this proof that satisfies the specification A. In the final section we show two examples of proofs for specifications and their extracted programs in system **T**.

4.1 System T

System **T** was invented by Gödel in [Göd80] to prove the consistency of arithmetic. It is an extension of the simply typed lambda calculus (with the type of the natural numbers as its only base type) that models recursive computation over the natural numbers for terms of any type.

The types of system **T** are defined by the following grammar:

$$t_1, t_2 ::= \text{nat} \mid t_1 \to t_2 \mid t_1 \wedge t_2$$

The only base type is nat, and furthermore, we have the usual function type of the simply typed lambda calculus, and additionally a product type, which will be the type of pairs.

The terms of system **T** are defined by the grammar:

$$M_1, M_2 ::= x^t \mid \lambda x^t.M \mid M_1 M_2 \mid \langle M_1, M_2 \rangle \mid \pi_1(M_1) \mid \pi_2(M_2) \mid 0 \mid \mathrm{s} \mid \mathrm{R}_t$$

where x is a variable and t is any type.

The types of abstractions and applications are determined in the usual way, where $M : t$ means that term M has type t:

- If $M : t_2$, then $\lambda x^{t_1}.M : t_1 \to t_2$
- If $M_2 : t_1 \to t_2$ and $M_1 : t_1$, then $M_2 M_1 : t_2$.

In a similar sense, the types of pairs and projections are determined by the following rules:

- If $M_1 : t_1$ and $M_2 : t_2$, then $\langle M_1, M_2 \rangle : t_1 \wedge t_2$
- If $M : t_1 \wedge t_2$, then $\pi_1(M) : t_1$
- If $M : t_1 \wedge t_2$, then $\pi_2(M) : t_2$

The constants 0 and s can be understood as zero and successor, and therefore are of the fixed types nat and nat $\to$ nat, respectively. As in the language of arithmetic, 0 and s allow to define numerals $\bar{n}$, i.e. terms $\mathrm{s} \ldots \mathrm{s}0$ where, s occurs n times.

We will refer to closed terms of the product type as pairs, and terms of the function type as functions.

As is shown by the grammar above, system **T** has a recursor constant for every type t. For any term t, the type of the recursor constant R_t is $(\mathrm{nat} \to t \to t) \to t \to \mathrm{nat} \to t$.

Note that all typed terms have a unique type, determined by the type of its variables and the above mentioned type rules. It is however clear that not every term is typable, because of the typing rules for the application and projections that are only defined for the function and product type. System **T** consists only of those terms that can be assigned a type according to these rules. Although not every application MN or projection $\pi_1(M)$, $\pi_2(M)$, where M, N are terms of system **T** will be a term of system **T**, we will typically not justify explicitly all the applications and projections that we make in the rest of this document and assume to be a term of system **T**.

Computation in system **T** is captured not only by the beta-reduction rule, but also by reduction rules for projections and recursors. Beta-reduction is expressed by the following rewrite rule:

$$(\lambda x^t.M)N \Rightarrow M[N/x^t]$$

Here we use the same notion of substitution as we have defined before for logical formulas. A bound variable in this case is a variable x^t that occurs within the scope of a lambda λx^t, i.e. a variable x^t occurring within the term M in an abstraction $\lambda x^t.M$.

The following two rewrite rules for projections on pairs express the projection on the left and right element of the pair, respectively:

$$\pi_1(\langle M_1, M_2\rangle) \Rightarrow M_1$$
$$\pi_2(\langle M_1, M_2\rangle) \Rightarrow M_2$$

The reduction rules for applications to the recursors represent the aforementioned idea of recursive computation over the natural numbers:

$$\mathrm{R}_t M_1 M_2 0 \Rightarrow M_1$$
$$\mathrm{R}_t M_1 M_2 (\mathrm{s}N) \Rightarrow M_2 N (\mathrm{R}_t M_1 M_2 N)$$

We will write $M_1 =_{\mathbf{T}} M_2$ whenever the term M_1 and M_2 rewrite to each other according to the rewrite rules of system **T**.

Although system **T** is more expressive than the simply typed lambda calculus, due to the recursors and their reduction rules, it still is strongly normalizing (in contrast to the classic lambda calculus) and has the Church-Rosser property. Proofs of these properties of system **T** can be found in for example [GTL89]. Seen as a programming language, it means that all programs written in system **T** terminate, with a unique output. This property plays a crucial role in extracting realizers from proofs, which will be discussed in the next sections.

As mentioned before, system **T** was introduced by Gödel to give a proof of consistency for Peano arithmetic, in combination with a negative translation. By Gödel's second incompleteness theorem, such a consistency proof cannot be formalized in Peano arithmetic itself, unless Peano arithmetic is inconsistent. The proof of strong normalization of system **T** is the part of his consistency proof that uses methods that transcend the techniques that are found in Peano arithmetic, such as induction over the natural numbers. All other parts of the proof can be encoded into Gödel numbers and proven as a result of the Peano axioms, including the proof transformation that from Peano to Heyting arithmetic. The difficulty in the proof of strong normalization of system **T** lies in the strong normalization of applications to the recursors, because we have a recursor of every type of the system **T**, which makes it impossible to prove this by merely induction over the natural numbers. The proof of strong normalization of the rewrite system arising from the primitive recursive definition discussed in the first chapter is easier to show, because it corresponds to the strong normalization of system **T** with only the recursor $\mathrm{R}_{\mathrm{nat}}$.

We can use system **T** to define functions over the natural numbers. Formally, a function $f : \mathbb{N}^k \to \mathbb{N}$ is definable in system **T** by a closed term F if and only if $F\bar{n_1} \dots \bar{n_k} =_T$

$\overline{f(n_1, \ldots, n_k)}$, for all $n_1, \ldots, n_k$. Note that this implies that F is of the type $\text{nat}^k \to \text{nat}$. Many functions are definable in system **T**. For example, the term

$$\text{plus} := \lambda x^{\text{nat}} \lambda y^{\text{nat}}.\text{R}_{\text{nat}} x^{\text{nat}} (\lambda z^{\text{nat}} \lambda u^{\text{nat}}.\text{s} u^{\text{nat}}) y$$

defines the addition function - when applied to two numerals, the application normalizes to their sum. Similarly, the term

$$\text{mult} := \lambda x^{\text{nat}} \lambda y^{\text{nat}}.\text{R}_{\text{nat}} 0 (\lambda z^{\text{nat}} \lambda u^{\text{nat}}.\text{plus} x^{\text{nat}} u^{\text{nat}}) y^{\text{nat}}$$

defines the multiplication function. The following lemma shows that not just $+$ and $\cdot$, but all primitive recursive are definable in system **T**, with just the recursor of type nat.

Lemma 11. *All primitive recursive functions are definable in system* ***T***

Proof. The proof is by induction over the primitive recursive functions. We will show for each case the term of system **T** that defines it, but leave out the proof that this term indeed defines the function.

- The constant function 0 is definable by the constant 0 of system **T**.
- The successor function is defined by the successor constant s of system **T**.
- The n-ary projection function P_n^i is defined by $\lambda x_1^{\text{nat}} \ldots \lambda x_n^{\text{nat}}.x_i^{\text{nat}}$.
- Let f be the composition of an n-ary primitive recursive function g and n m-ary primitive recursive functions $h_1, \ldots, h_n$. The induction hypothesis states that g and $h_1, \ldots, h_n$ are defined by the terms G and $H_1, \ldots H_n$ of system **T**, respectively. Then f is defined by:

$$\lambda x_1^{\text{nat}} \ldots \lambda x_m^{\text{nat}}.G(H_1 x_1^{\text{nat}} \ldots x_m^{\text{nat}}) \ldots (H_n x_1^{\text{nat}} \ldots x_m^{\text{nat}})$$

- Let f be defined by primitive recursion with the help of n-ary primitive recursive g and $n+2$-ary primitive recursive h. The induction hypothesis states that g and h are defined by the terms G and H of system **T**, respectively. Then f is defined by:

$$\lambda x_1^{\text{nat}} \ldots \lambda x_n^{\text{nat}} \lambda x_{n+1}^{\text{nat}}.\text{R}_{\text{nat}} (G x_1^{\text{nat}} \ldots x_n^{\text{nat}}) (\lambda y^{\text{nat}} \lambda z^{\text{nat}}.H x_1^{\text{nat}} \ldots x_n^{\text{nat}} y^{\text{nat}} z^{\text{nat}}) x_{n+1}^{\text{nat}}$$

□

However, much more functions than the primitive recursive functions can be defined in system **T**. With the help of $\text{R}_{\text{nat}\to\text{nat}}$, for example, the Ackermann function can be defined. In fact, the functions that are definable in system **T** correspond exactly to the functions that are provable total in Peano arithmetic. A proof of this result can be found in [rU06] and [GTL89]. The set of provably total functions is a strict, but very rich subset of the computable functions, that contains most functions that are commonly considered.

Additionally to the terms of system **T** that we just described, we will add another typed term to system **T**, namely $\mathrm{i} : 1$, that does not influence any of the above results about system **T**. This term is supposed to represent the empty program, and it thus seems like a useless addition to a programming language. We merely add this because later when we discuss modified realizability, it will be useful to express a program that has computational meaning. In particular, we will use it to realize equations.

If this term occurs in a term of system **T**, it intuitively does not add any meaning to it. For example, the pair $\langle M, \mathrm{i}\rangle$, i.e. a pair of term M and the empty program, has the same meaning as simply the term M. The same goes for other terms involving $\mathrm{i} : 1$. This gives rise to the idea of normalizing a term of system **T** with respect to $\mathrm{i} : 1$, i.e. removing occurrences $\mathrm{i} : 1$ from terms. After normalization, we thus either end up with a term that is free from occurrences of $\mathrm{i} : 1$, or is the empty program itself. The function $|\cdot|$, that operates on both types and terms of system **T**, formalizes this idea of preserving only the actual informative part of a term.

For a type t, we obtain a type $|t|$ that is free of occurrences 1 or equal to 1 in the following way:

- $|t_1 \wedge t_2| = \begin{cases} |t_1| & \text{if } |t_2| = 1 \\ |t_2| & \text{if } |t_1| = 1 \\ |t_1| \wedge |t_2| & \text{otherwise} \end{cases}$
- $|t_1 \to t_2| = \begin{cases} 1 & \text{if } |t_2| = 1 \\ |t_2| & \text{if } |t_1| = 1 \\ |t_1| \to |t_2| & \text{otherwise} \end{cases}$

Note that 1 in types behaves the same as $\top$ in propositional formulas.

The term $|M|$ is the normal form of a term M with respect to i. Obviously, the terms 0, s and i stay as they are. Note that we map variables of type 1 to the term i, because i is the only possible term of this type.

- $|\mathrm{R}_t| = \begin{cases} \mathrm{i} & \text{if } |t| = 1 \\ \mathrm{R}_{|t|} & \text{otherwise} \end{cases}$
- $|x^t| = \begin{cases} \mathrm{i} & \text{if } |t| = 1 \\ x^{|t|} & \text{otherwise} \end{cases}$
- $|\lambda x^t M| = \begin{cases} \mathrm{i} & \text{if } M : t' \text{ and } |t'| = 1 \\ |M| & \text{if } |t| = 1 \\ \lambda x^{|t|} |M| & \text{otherwise} \end{cases}$

- $|MN| = \begin{cases} \mathrm{i} & \text{if } M : t_1 \to t_2 \text{ and } |t_2| = 1 \\ |M| & \text{if } M : t_1 \to t_2,\ |t_1| = 1 \text{ and } |t_2| \neq 1 \\ |M||N| & \text{otherwise} \end{cases}$
- $|\langle M, N\rangle| = \begin{cases} |M| & \text{if } N : t \text{ and } |t| = 1 \\ |N| & \text{if } M : t \text{ and } |t| = 1 \\ \langle |M||N| \rangle & \text{otherwise} \end{cases}$
- $\pi_1(M) = \begin{cases} \mathrm{i} & \text{if } M : t_1 \wedge t_2 \text{ and } |t_1| = 1 \\ |M| & \text{if } M : t_1 \wedge t_2,\ |t_2| = 1 \text{ and } |t_1| \neq 1 \\ \pi_1(|M|) & \text{otherwise} \end{cases}$
- $\pi_2(M) = \begin{cases} \mathrm{i} & \text{if } M : t_1 \wedge t_2 \text{ and } |t_2| = 1 \\ |M| & \text{if } M : t_1 \wedge t_2,\ |t_1| = 1 \text{ and } |t_2| \neq 1 \\ \pi_2(|M|) & \text{otherwise} \end{cases}$

It follows that if we have a term $M : t$, then $|M| : |t|$, i.e. the function preserves typability and therefore the resulting term is again a term of system **T**. If a term T defines a function $f : \mathbb{N}^k \to \mathbb{N}$, i.e. if program T computes f, then its normal form $|T|$ computes the function f as well. This implies that normalization on programs is harmless. It just removes redundancies from programs, without changing their input-output behavior.

4.2 Modified realizability

Modified realizability is a definition that connects the language of system **T** to sentences of arithmetic. As explained before, it defines how terms of system **T** can be understood as constructions that serve as witnesses for formulas, which is done by Kleene's realizability in a similar sense for Gödel numbers of computable functions. Our interest in m-realizers for sentences however, is explained by Theorem 4, which implies that an m-realizer for a sentence $\forall x_1 \ldots \forall x_r \exists y$ that states that a specification A describes a total function, defines a function that satisfies A.

In the following definition of modified realizability we will use the notation $\bar{n}$ to denote the numeral for the natural number n, in both the language of arithmetic and system **T**. It will always be clear from the context to which of the two formalisms $\bar{n}$ is referring to.

Definition 2. *Let A be a sentence and let M be a closed term of system **T**. The following conditions describe when M m-realizes A:*

- *i m-realizes $t_1 = t_2$ iff t_1 and t_2 rewrite to the same numeral.*
- *M m-realizes $B_1 \wedge B_2$ iff $\pi_1(M)$ m-realizes B_1 and $\pi_2(M)$ m-realizes B_2.*

- *M m-realizes $B_1 \vee B_2$ iff $\pi_1(M) = 0$ and $\pi_1(\pi_2(M))$ m-realizes B_1 or $\pi_1(M) = s(0)$ and $\pi_2(\pi_2(M))$ m-realizes B_2.*
- *M m-realizes $B_1 \to B_2$ iff MN m-realizes B_2 whenever N m-realizes B_1.*
- *M m-realizes $\exists x B$ iff there is an n such that $\pi_1(M) \Rightarrow_T \bar{n}$ and $\pi_2(M)$ m-realizes $B[\bar{n}/x]$.*
- *M m-realizes $\forall x B$ iff $M\bar{n}$ m-realizes $B[\bar{n}/x]$ for all n.*
- *No term m-realizes $\bot$.*

We refer to the term that m-realizes a sentence as its m-realizer.

The definition of an m-realizer for a disjunction differs from the other connectives in the sense that there is no unique construction in system **T** that serves as realizer for the disjunction, as is the case for the other connectives. The reason for this is that the disjunction is actually not needed in Heyting arithmetic since every disjunction $A \vee B$ is equivalent to the formula $\exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))$, shown by Lemma 1. An m-realizer for $A \vee B$ is thus an m-realizer for $\exists x(((x = 0) \to A) \wedge (\neg(x = 0) \to B))$, normalized according to the function $|\cdot|$.

The definition of modified realizability can be adjusted in such a way that all m-realizers are normalized, i.e. either equal to i : 1 of free of i : 1. This can be done by building the normalization with respect to i : 1 into the definition of terms of system **T**, by identifying terms with their normalized forms. This is actually the approach taken in [rU06], but it makes the definition of modified realizability harder to understand, which is why we have decided to provide the normalization as an independent operation on terms of system **T**, which can be used afterwards to obtain a better program.

Before we show that every provable sentence is m-realizable, we will discuss some helpful definitions and lemma's. We will first define a function $\flat$ that maps formulas to types, and show that whenever a sentence A is m-realizable, it is realizable by a term of the type $\flat(A)$.

Definition 3. *The function $\flat$ maps formulas of arithmetic to types of the system **T**, in the following way:*

- $\flat(t_1 = t_2) := 1$
- $\flat(B_1 \wedge B_2) := \flat(B_1) \wedge \flat(B_2)$
- $\flat(B_1 \to B_2) := \flat(B_1) \to \flat(B_2)$
- $\flat(B_1 \vee B_2) := nat \wedge (\flat(B_1) \wedge \flat(B_2))$
- $\flat(\exists x B) := nat \wedge \flat(B)$

- $\flat(\forall x B) := nat \to \flat(B)$
- $\flat(\bot) = 1$.

Lemma 12. *Let A be a formula, with free variables $x_1, \ldots, x_r$. For all $n_1, \ldots, n_r$, if M is m-realizable, then $A[\bar{n_1}/x_1] \ldots [\bar{n_r}/x_r]$, then there is an m-realizer of the type $\flat(A)$.*

Proof. The proof is by induction over the structure of formula A. We will not show the proof, but note that the formula $\bot$ is never m-realized by definition, and therefore the type of its possible m-realizer could be anything, so we just choose i : 1. Note also that whenever a term M m-realizes B_1, we have that $\langle 0, \langle M, N\rangle\rangle$ m-realizes $B_1 \vee B_2$, where N can be any term. We can then just choose an arbitrary term of the type $\flat(B_2)$.

The definition of $\flat$ for the disjunction indeed corresponds to the normalized type $\flat(existsx(((x = 0) \to A) \wedge (\neg(x = 0) \to B)))$. □

Note that the type of the m-realizer is independent of the terms occurring in the sentence it realizes. This implies that we can compute already the type $\flat(A)$ of an m-realizer for some closed instance of a formula A, without knowing what exactly this instance will be.

Let t be a term in the language of arithmetic, with free variables $x_1, \ldots, x_r$. We will show that the term t corresponds to a term T in system **T** such that for all natural numbers $n_1, \ldots, n_r$, the term $t[\bar{n_1}/x_1] \ldots [\bar{n_r}/x_r]$ rewrites to $\bar{n}$ in the rewrite system $\mathcal{R}$ if and only if $(\lambda x_1 \ldots \lambda x_n.T)\bar{n_1} \ldots [\bar{n_r}] =_T \bar{n}$, for all n. This means that the rewriting behavior described by $\mathcal{R}$ of every arithmetical term can be completely represented in system **T**.

We will first define what such a term T in system **T** for an arithmetical term t looks like. As we have already shown in Lemma 11 that all primitive recursive functions are definable by system **T**, the ideas that we obtain a term T from t by simply replacing the variables $x_1, \ldots, x_r$ by $x_1^{\text{nat}}, \ldots, x_r^{\text{nat}}$, and replacing all function symbols f by terms F in system **T** that define f. Formally, T is defined as follows:

Definition 4. *Let t be a term in the language of arithmetic. We define a corresponding term T in the system **T** inductively over the structure of term t:*

- *Let t be a variable x. Then $T := x^{nat}$.*
- *Let t be $f(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms. Note that the function symbol f is a function symbol for a primitive recursive function. Let F be the term in system **T** that defines this primitive recursive function, following Lemma 11 of the previous section. Then $T := FT_1 \ldots T_n$, where $T_1 \ldots T_n$ are the corresponding terms in system **T** for terms $t_1, \ldots, t_n$.*

The following lemma states that T behaves as desired.

Lemma 13. *$t[\bar{n_1}/x_1]\dots[\bar{n_r}/x_r]$ rewrites to $\bar{n}$ if and only if $(\lambda x_1 \dots \lambda x_n.T)\bar{n_1}\dots[\bar{n_r}] =_T \bar{n}$ for all n.*

Proof. By induction over de structure of t. □

We will now proof that all theorem of Heyting arithmetic are m-realizable. We want to make use of the recursive definition of proofs in the natural deduction calculus for Heyting arithmetic, so we can prove the statement by induction. However, the proof calculus provides us with an inductive definition of provable judgments, not of theorems specifically. We will therefore prove a more general statement. Instead of showing that every theorem A of Heying arithmetic if m-realizable by a term M, we will prove the following theorem.

Theorem 4. *Let $\Gamma \vdash A$ be a provable judgment with free variables $x_1, \dots, x_r$, and where Γ is $\{C_1, \dots, C_k\}$. If $\Gamma \vdash A$ is provable. Then there is term M such that for all natural numbers $n_1, \dots, n_r$, if $M_i[\bar{n_1}/x_1^{nat}]\dots[\bar{n_r}/x_r^{nat}]$ m-realizes $C_i[\bar{n_1}/x_1]\dots[\bar{n_r}/x_r]$ for all i such that $1 \leq i \leq k$, then $M[M_1/y_1^{\flat(C_1)}]\dots[M_k/y_k^{\flat(C_k)}][\bar{n_1}/x_1^{nat}]\dots[\bar{n_r}/x_r^{nat}]$ m-realizes $A[\bar{n_1}/x_1^{nat}]\dots[\bar{n_r}/x_r^{nat}]$.*

It follows that if A is a theorem of Heyting arithmetic, i.e. $\vdash A$ is provable, and A does not contain any free variables, then it is m-realized by a term M of system **T**. The m-realizability of all theorems is a corollary of the theorem, where the antecedent is empty and there are no free variables.

Corollary 2. *All theorems of Heyting arithmetic are m-realizable.*

In the proof of Theorem 4 that follows, whenever we want to say that M is such that for all natural numbers $n_1, \dots, n_r$, if $M_i[\bar{n_1}/x_1^{\text{nat}}]\dots[\bar{n_r}/x_r^{\text{nat}}]$ m-realizes $C_i[\bar{n_1}/x_1]\dots[\bar{n_r}/x_r]$ for all i such that $1 \leq i \leq k$, then $M[M_1/y_1^{\flat(C_1)}]\dots[M_k/y_k^{\flat(C_k)}][\bar{n_1}/x_1^{\text{nat}}]\dots[\bar{n_r}/x_r^{\text{nat}}]$ m-realizes $A[\bar{n_1}/x_1^{\text{nat}}]\dots[\bar{n_r}/x_r^{\text{nat}}]$, we will simply say that M m-realizes $\Gamma \vdash A$. We will refer to The variable $y_i^{\flat(C_i)}$ as the placeholder for an m-realizer for $C_i[\bar{n_1}/x_1^{\text{nat}}]\dots[\bar{n_r}/x_r^{\text{nat}}]$.

Proof. The proof is by induction over provable judgments. The base cases correspond to axioms of the calculus, and the inductive cases tot the proof rules. We will only show the m-realizers, and leave out the proof that they indeed are m-realizers for the axioms and conclusions of proof rules.

Logical axiom

$$\frac{}{A \vdash A}\ \text{ax}$$

Let $y^{\flat(A)}$ be the placeholder for an m-realizer for A. The term

$$y^{\flat(A)}$$

m-realizes $A \vdash A$.

Weakening

$$\frac{\Gamma \vdash B}{A, \Gamma \vdash B}\,\mathrm{w}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash B$. The term

$$M$$

m-realizes $A, \Gamma \vdash B$.

Contraction

$$\frac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}\,\mathrm{c}$$

The induction hypothesis states that there is a term M that m-realizes $A, A, \Gamma \vdash B$. Let $y^{\flat(A)}$ and $z^{\flat(A)}$ be the placeholders for m-realizers for A. The term

$$M[y^{\flat(A)}/z^{\flat(A)}]$$

m-realizes $A, \Gamma \vdash B$.

$\wedge$ elimination 1

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\,\wedge\mathrm{e1}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash A \wedge B$. The term

$$\pi_1(M)$$

m-realizes $\Gamma \vdash A$.

$\wedge$ elimination 2

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge\text{e2}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash A \wedge B$.

The term

$$\pi_2(M)$$

m-realizes $\Gamma \vdash A$.

$\wedge$ introduction

$$\frac{\Gamma \vdash A \qquad \Pi \vdash B}{\Gamma, \Pi \vdash A \wedge B} \wedge\text{i}$$

The induction hypothesis states that there is a term M_1 that m-realizes $\Gamma \vdash A$, and a term M_2 that m-realizes $\Pi \vdash B$. The term

$$\langle M_1, M_2 \rangle$$

m-realizes $\Gamma, \Pi \vdash A \wedge B$.

$\vee$ elimination

$$\frac{\Gamma \vdash A \vee B \qquad \Pi, A \vdash C \qquad \Delta, B \vdash C}{\Gamma, \Pi, \Delta \vdash C} \vee\text{e}$$

The induction hypothesis states that there are terms M_1, M_2 and M_3 that m-realize $\Gamma \vdash A \vee B$, $\Pi, A \vdash C$, and $\Delta, B \vdash C$ respectively. Let $y^{\flat(A)}$ and $y^{\flat(B)}$ be the placeholders for m-realizers for A and B, respectively. The term

$$\mathrm{R}_{\flat(C)} M_2[\pi_2(M_1)/y^{\flat(A)}]\lambda z_1 \lambda z_2 . M_3[\pi_2(M_1)/y^{\flat(B)}]\pi_1(M_1)$$

m-realizes $\Gamma, \Pi, \Delta \vdash C$.

As showed in Lemma 1, the disjunction is not needed in Heyting Arithmetic and can be understood as an abbreviation. This means that also the rules concerning the disjunction are superfluous, and they can be understood as shortcuts for proofs without the disjunction. This explains the occurrence of the recursor in the realizer for $\Gamma, \Pi, \Delta \vdash C$, as the second half of the proof of Lemma 1 uses induction, which means that the induction rule is used in the shortcut that the $\vee$ elimination rule abbreviates. The realizer can be understood as an if-else expression, specifically as 'if $\pi_1(M_1) =_T 0$ then $M_2[\pi_2(M_1)/y^{\flat(A)}]$, else $M_3[\pi_2(M_1)/y^{\flat(B)}]$'.

$\vee$ introduction 1

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \vee\text{i1}$$

The induction hypothesis states that there is term M that m-realizes $\Gamma \vdash A$. Let N be any closed term of type $\flat(B)$. The term

$$\langle 0, \langle M, N \rangle \rangle$$

m-realizes $\Gamma \vdash A \vee B$

$\vee$ introduction 2

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \vee\text{i2}$$

The induction hypothesis states that there is term M that m-realizes $\Gamma \vdash B$. Let N be any closed term of type $\flat(A)$. The term

$$\langle s(0), \langle N, M \rangle \rangle$$

m-realizes $\Gamma \vdash A \vee B$

$\to$ elimination

$$\frac{\Gamma \vdash A \to B \qquad \Pi \vdash A}{\Gamma, \Pi \vdash B} \to\text{e}$$

The induction hypothesis states that there is a term M_1 that m-realizes $\Gamma \vdash A \to B$, and a term M_2 that m-realizes $\Pi \vdash A$. The term

$$(M_1 M_2)$$

m-realizes $\Gamma, \Pi \vdash B$.

$\to$ introduction

$$\frac{A, \Gamma \vdash B}{\Gamma \vdash A \to B} \to\text{i}$$

The induction hypothesis states that there is a term M that m-realizes $A, \Gamma \vdash B$. Let $y^{\flat(A)}$ be the placeholder for an m-realizer for A. The term

$$\lambda y^{\flat(A)}.M$$

m-realizes for $\Gamma \vdash A \to B$.

$\bot$ elimination

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot\mathrm{e}$$

The induction hypothesis states that there is a term M that m-realizes m-realizes $\Gamma \vdash \bot$. The term

$$N$$

m-realizes $\Gamma \vdash A$, where N is any term of the type $\flat(A)$.

$\forall$ elimination

$$\frac{\Gamma \vdash \forall x A}{\Gamma \vdash A[t/x]}\ \forall\mathrm{e}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash \forall x A$. Let T be the term in system $\mathbf{T}$ that corresponds to t. The term

$$(MT)$$

m-realizes $\Gamma \vdash A[t/x]$.

$\forall$ introduction

$$\frac{\Gamma \vdash A[\alpha/x]}{\Gamma \vdash \forall x A}\ \forall\mathrm{i}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash A[\alpha/x]$. The term

$$\lambda\alpha^{\mathrm{nat}}.M$$

m-realizes $\Gamma \vdash \forall x A$.

$\exists$ elimination

$$\frac{\Gamma \vdash \exists x A \qquad A[\alpha/x], \Pi \vdash B}{\Gamma, \Pi \vdash B}\ \exists\mathrm{e}$$

The induction hypothesis states that there is a term M_1 that m-realizes $\Gamma \vdash \exists x A$, and a term M_2 that m-realizes $A[\alpha/x], \Pi \vdash B$. Let $y^{\flat(A)}$ be the placeholder for an m-realizer for $A[\alpha/x]$. The term

$$M_2[\pi_2(M_1)/y^{\flat(A)}][\pi_1(M_1)/\alpha^{\mathrm{nat}}]$$

m-realizes $\Gamma, \Pi \vdash B$.

$\exists$ introduction

$$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A}\ \exists\mathrm{i}$$

The induction hypothesis states that there is a term M that m-realizes $\Gamma \vdash A[t/x]$. Let T be the term in system $\mathbf{T}$ that corresponds to t. The term

$$\langle T, M \rangle$$

m-realizes $\Gamma \vdash \exists x A$.

$=$ elimination

$$\frac{\Gamma \vdash t_1 = t_2 \qquad \Pi \vdash A[t_1/x]}{\Gamma, \Pi \vdash A[t_2/x]}\ =\mathrm{e}$$

The induction hypothesis states that there is a term M_1 that m-realizes $\Gamma \vdash t_1 = t_2$, and a term M_2 that m-realizes $\Pi \vdash A[t_1/x]$. The term

$$M_2$$

m-realizes $\Gamma, \Pi \vdash A[t_2/x]$.

$=$ introduction

$$\frac{}{\vdash t = t}\ =\mathrm{i}$$

The term

$$\mathrm{i}$$

m-realizes $\vdash t = t$.

Induction

$$\frac{\Gamma \vdash A[0/x] \qquad A[\alpha/x], \Pi \vdash A[s(\alpha)/x]}{\Gamma, \Pi \vdash A[t/x]}\ \mathrm{ind}$$

The induction hypothesis states that there is a term M_1 that m-realizes $\Gamma \vdash A[0/x]$, and a term M_2 that m-realizes $A[\alpha/x], \Pi \vdash A[s(\alpha)/x]$. Let $y^{\flat(A)}$ be the placeholder for a m-realizer for $A[\alpha/x]$, and let T be the term in system $\mathbf{T}$ that corresponds to t. The term

$$\mathrm{R}_{\flat(A)} M_1 (\lambda \alpha^{\mathrm{nat}} \lambda y^{\flat(A)}.M_2) T$$

m-realizes $\Gamma, \Pi \vdash [t/x]$.

Natural numbers

$$\frac{}{\vdash \forall x \neg(s(x) = 0)}\ \text{nat1}$$

The term

$$\lambda x^{\text{nat}} \lambda y^{1}.\text{i}$$

m-realizes $\vdash \forall x \neg(s(x) = 0)$.

$$\frac{}{\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))}\ \text{nat2}$$

The term

$$\lambda x^{\text{nat}} \lambda y^{\text{nat}} \lambda z^{1}.\text{i}$$

m-realizes $\vdash \forall x \forall y((s(x) = s(y)) \to (x = y))$.

Primitive recursive definitions

Consider any axiom for a primitive recursive definition. It will be of the form $\vdash \forall x_1 \dots \forall x_r(t_1 = t_2)$, for some terms t_1 and t_2, with free variables $x_1, \dots, x_r$.

The term

$$\lambda x_1^{\text{nat}} \dots \lambda x_r^{\text{nat}}.\text{i}$$

m-realizes $\vdash \forall x_1 \dots \forall x_r(t_1 = t_2)$.

□

Note that the consistency of Heyting arithmetic is a consequence of Theorem 4, as $\bot$ is not realizable, and therefore also the consistency of Peano arithmetic, as we have explained in the second chapter of this thesis.

As explained above, we are mainly interested in m-realizers for Π_2 sentences, because the realizer for a provable sentence $\forall x_1 \dots \forall x_r \exists y A$ contains a program that satisfies the specification A. Specifically, the following theorem implies that whenever M m-realizes $\forall x_1 \dots \forall x_r \exists y A$, then the program $\lambda x_1^{\text{nat}} \dots \lambda x_r^{\text{nat}} \pi_1(M x_1^{\text{nat}} \dots x_r^{\text{nat}})$ is correct with respect to A.

Theorem 5. *If M m-realizes $\forall x_1 \dots \forall x_r \exists y A$, then $\lambda x_1^{nat} \dots \lambda x_r^{nat} \pi_1(M x_1^{nat} \dots x_r^{nat})$ is a program that satisfies the specification A.*

Proof. In other words, according to what it means for a program to satisfy a specification, we want to show that if M m-realizes $\forall x_1 \dots \forall x_r \exists y A$, then for all natural numbers $n_1, \dots, n_r$, the sentence $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][(\lambda x_1^{\text{nat}} \dots x_r^{\text{nat}}.\pi_1(M x_1^{\text{nat}} \dots x_r^{\text{nat}})\bar{n_1} \dots \bar{n_r})/y]$ is true, i.e., by beta-reduction, the sentence $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][\pi_1(M\bar{n_1} \dots \bar{n_r})/y]$ is true.

Assume M m-realizes $\forall x_1 \dots \forall x_r \exists y A$. Then, according to the definition of m-realizability of universally quantified formula, for all $n_1, \dots, n_r$, the term $M\bar{n_1} \dots \bar{n_r}$ m-realizes $\exists y A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r]$. Then, according to the definition of m-realizability of existentially quantified formula, for all $n_1, \dots, n_r$, there is an n, such that $\pi_1(M\bar{n_1} \dots \bar{n_r}) = \bar{n}$ and $\pi_2(M\bar{n_1} \dots \bar{n_r})$ m-realizes $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][\bar{n}/y]$. In other words, for all $n_1, \dots, n_r$, $\pi_2(M\bar{n_1} \dots \bar{n_r})$ m-realizes $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][\pi_1(M\bar{n_1} \dots \bar{n_r})/y]$.

Consider arbitrary natural numbers $n_1, \dots, n_r$. We have shown that $\pi_2(M\bar{n_1} \dots \bar{n_r})$ m-realizes the sentence $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][\pi_1(M\bar{n_1} \dots \bar{n_r})/y]$, but we need to show that the $A[\bar{n_1}/x_1] \dots [\bar{n_r}/x_r][\pi_1(Mn_1 \dots \bar{n_r})/y]$ is true to obtain the desired result. For this purpose we will show that for all quantifier-free sentences A, if A is m-realizable, then it is true.

In general it is not the case that an m-realizable formula is always true. However, it is well-known that Heyting arithmetic is complete for Σ_1 sentences, and luckily the property still holds for the extension of Heyting arithmetic with primitive recursive function that we consider. This implies that the classical and intuitionistic meaning of Σ_1 sentences agree, and therefore, the meaning of a Σ_1 sentence A being provable in both Peano and Heyting arithmetic, can be understood as A being true in the standard model.

Knowing that Heyting arithmetic is thus complete for quantifier-free sentences, we can show that a quantifier-free sentence is true, whenever it is m-realizable. For the purpose of obtaining a contradiction, assume that A is m-realizable, but not true. By completeness we get that $\neg A$ is provable and then by Theorem 4 we get that $\neg A$ is m-realizable. By definition of m-realizability, this means that A cannot be m-realizable, which is in contradiction with the initial assumption that A is m-realizable.

□

Note that if M m-realizes $\forall x_1 \dots \forall x_r \exists y A$, we have that $\lambda x_1^{\text{nat}} \dots \lambda x_r^{\text{nat}} \pi_1(M x_1^{\text{nat}} \dots x_r^{\text{nat}})$ is the same as the term $\lambda x_1^{\text{nat}} \dots \lambda x_r^{\text{nat}} \pi_1(|M| x_1^{\text{nat}} \dots x_r^{\text{nat}})$. We can therefore just look at the normalized realizer of a theorem in order to obtain a program for a specification.

4.3 Examples

We will first demonstrate the extraction of m-realizers and programs from proofs with the help of a simple example. Consider the specification $y = x + s(0)$, where x denotes the input, and y the output value. The following proof shows that there indeed exists a functions that increments the input with one.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash s(x) = s(x)}\,=\!\text{i}}{\vdash s(x) = x + s(0)}\,\mathcal{R}}{\vdash \exists y(y = x + s(0))}\,\exists\text{i}}{\vdash \forall x \exists y(y = x + s(0))}\,\forall\text{i}$$

We start the extraction of an m-realizer from this proof by extracting a realizer for the leaf of the prooftree, and then recursively construct the realizer for all the subproofs, following the algorithm described in the previous section. We end up with a realizer $\lambda x^{\mathrm{nat}}.\langle s(x^{\mathrm{nat}}),\mathrm{i}\rangle$ for the conclusion $\forall x \exists y(y = x + s(0))$ in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\text{i m-realizes } s(x) = s(x)}\,=\mathrm{i}}{\text{i m-realizes } s(x) = x + s(0)}\,\mathrm{d}}{\langle s(x^{\mathrm{nat}}),\mathrm{i}\rangle \text{ m-realizes } \exists y(y = x + s(0))}\,\exists\mathrm{i}}{\lambda x^{\mathrm{nat}}.\langle s(x^{\mathrm{nat}}),\mathrm{i}\rangle \text{ m-realizes } \forall x \exists y(y = x + s(0))}\,\forall\mathrm{i}$$

Then according to Theorem 4, the term $\lambda x^{\mathrm{nat}}.\pi_1((\lambda x^{\mathrm{nat}}.\langle s(x^{\mathrm{nat}}),\mathrm{i}\rangle)x^{\mathrm{nat}})$ is a program that satisfies the specification $y = x + s(0)$. This term reduces to the simpler term $\lambda x^{\mathrm{nat}}.s(x^{\mathrm{nat}})$, for which it can easily seen that it is indeed a program that, given a natural number, outputs its successor.

Consider now the more interesting specification $x(= 2 \cdot y) \vee (x = (2 \cdot y) + 1)$, where 2 and 1 are abbreviations for the numerals $s(s(0))$ and $s(0)$. The variable x denotes the input and y the output value. It is a specification that requires a program that divides the input x by two. Let A be $\exists y((x = 2 \cdot y) \vee (x = 2 \cdot y + 1))$. Consider the following proof, that shows that there exists indeed such a function:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 = 0}\,=\mathrm{i}}{\vdash 0 = 2 \cdot 0}\,\mathrm{d}}{\vdash (0 = 2 \cdot 0) \vee (0 = (2 \cdot 0) + 1)}\,\vee\mathrm{i}}{\vdash \exists y((0 = 2 \cdot y) \vee (0 = (2 \cdot y) + 1))}\,\exists\mathrm{i} \qquad \begin{matrix}\pi_{\mathrm{I}}\\ A \vdash \exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))\end{matrix}}{\vdash \exists y((x = 2 \cdot y) \vee (x = (2 \cdot y) + 1))}\,\mathrm{ind}}{\vdash \forall x \exists y((x = 2 \cdot y) \vee (x = (2 \cdot y) + 1))}\,\forall\mathrm{i}$$

The proof π_{I} that follows is the induction case of the proof. We will proof this with the help of $\exists$ elimination:

$$\dfrac{\dfrac{}{A \vdash A}\,\mathrm{ax} \qquad \begin{matrix}\pi'_I\\ (x = 2 \cdot z) \vee (x = (2 \cdot z) + 1) \vdash \exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))\end{matrix}}{A \vdash \exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))}\,\exists\mathrm{e}$$

Let B be $\exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))$. The following proof π'_I shows that we will make a proof by cases through $\vee$ elimination.

$$\dfrac{\dfrac{}{(x = 2 \cdot z) \vee (x = (2 \cdot z) + 1) \vdash B}\,\text{ax} \qquad \overset{\pi_1}{x = 2 \cdot z \vdash B} \qquad \overset{\pi_2}{x = (2 \cdot z) + 1 \vdash B}}{(x = 2 \cdot z) \vee (x = (2 \cdot z) + 1) \vdash B}\,\vee\text{e}$$

The following proof π_1 shows that from $x = 2 \cdot z$, we can find a value y such that $(s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1)$, namely the value z itself.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x = y \vdash x = y}\,\text{ax} \quad \dfrac{}{\vdash s(x) = s(x)}\,=\text{i}}{x = y \vdash s(x) = s(y)}\,=\text{e}}{\vdash (x = y) \to (s(x) = s(y))}\,\to\text{i}}{\vdash \forall y((x = y) \to (s(x) = s(y)))}\,\forall\text{i}}{\vdash (x = 2 \cdot z) \to (s(x) = s(2 \cdot z))}\,\forall\text{e}}{\vdash (x = 2 \cdot z) \to (s(x) = (2 \cdot z) + 1)}\,\mathcal{R} \qquad \dfrac{}{x = 2 \cdot z \vdash x = 2 \cdot z}\,\text{ax}}{x = 2 \cdot z \vdash s(x) = (2 \cdot z) + 1}\,\to\text{e}}{x = 2 \cdot z \vdash (s(x) = 2 \cdot z) \vee (s(x) = (2 \cdot z) + 1)}\,\vee\text{i}}{x = 2 \cdot z \vdash \exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))}\,\exists\text{i}$$

The following proof π_2 shows the other case of the proof π'_{I}, which states that if $x = (2 \cdot z) + 1$, we can also find a value y such that $\exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))$, namely the value $s(z)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x = y \vdash x = y}\,\text{ax} \quad \dfrac{}{\vdash s(x) = s(x)}\,=\text{i}}{x = y \vdash s(x) = s(y)}\,=\text{e}}{\vdash (x = y) \to (s(x) = s(y))}\,\to\text{i}}{\vdash \forall y(((x = y) \to (s(x) = s(y)))}\,\forall\text{i}}{\vdash (x = (2 \cdot z) + 1) \to (s(x) = s((2 \cdot z) + 1))}\,\forall\text{e}}{\vdash (x = (2 \cdot z) + 1) \to (s(x) = 2 \cdot s(z))}\,\mathcal{R} \qquad \dfrac{}{x = (2 \cdot z) + 1 \vdash x = (2 \cdot z) + 1}\,\text{ax}}{x = (2 \cdot z) + 1 \vdash s(x) = 2 \cdot s(z)}\,\to\text{e}}{x = (2 \cdot z) + 1 \vdash (s(x) = 2 \cdot s(z)) \vee (s(x) = (2 \cdot s(z)) + 1)}\,\vee\text{i}}{x = (2 \cdot z) + 1 \vdash \exists y((s(x) = 2 \cdot y) \vee (s(x) = (2 \cdot y) + 1))}\,\exists\text{i}$$

The term that is extracted from this proof by applying the algorithm described in the previous section is a very complicated term that gives little insight in what the program computes. Applying normalization with respect to the empty program that we have discussed in the section about system $\mathbf{T}$, results in the following simpler term M:

$$\lambda x^{\mathrm{nat}} \mathrm{R}_\tau \langle 0, 0\rangle (\lambda y^{\mathrm{nat}} \lambda z^\tau . \mathrm{R}_\tau \langle \pi_1(z^\tau), \mathrm{s}0\rangle (\lambda v^{\mathrm{nat}} \lambda w^\tau . \langle \mathrm{s}\pi_1(z^\tau), 0\rangle)(\pi_2(z^\tau)))x^{\mathrm{nat}}$$

where τ is nat $\wedge$ nat.

The program that computes the division by two, i.e. the function that satisfies the specification $(x = 2 \cdot y) \vee (x = (2 \cdot y) + 1)$, is the function $\lambda x^{\mathrm{nat}}.\pi_1(Mx^{\mathrm{nat}})$ according to Theorem 4. The outcome of this function applied to a numeral $\bar{n}$ will be the first element of the outcome of the function M applied to $\bar{n}$. The function computes the division of two of a number n by increasing the outcome every second recursive call of the first recursor, starting with outcome 0.

Note that the extracted program uses a recursor of a product type. The function that divides numbers by two is however a primitive recursive function, which implies that it can be defined in system **T** by a term that uses only recursors of type nat. The reason that the algorithm gives as this term, is because of the shape of the proof that we extracted it from. If we would have pushed the application of the induction rules more upwards, such that we apply only induction on equations, we would have obtained a different term, that uses only recursors of the type nat.

CHAPTER 5

Conclusion and future work

In this thesis we have shown that we can obtain programs for Σ_1 specifications by reducing the search for a satisfying program to the search for a proof of the proposition that the specification defines a total function. In other words, given that we have a proof in Peano arithmetic of $\forall x_1 \dots \forall x_r \exists y A$, for a specification A, we can automatically extract a program that computes a function f such that $A[n_1/x_1] \dots [n_r/x_r][f(n_1, \dots, n_r)/y]$ is true. We will discuss the theoretical limitations of this approach, and when we know what is theoretically (im)possible to do, we will suggest possible improvements within this scope.

The described algorithm starts with a proof of the totality of the relation that is defined by a specification A, i.e. a proof of $\forall x_1 \dots \forall x_r \exists y A$. Such a proof does not necessarily exist for every specification A for which a total, computable function would be satisfactory. All these functions are definable by some formula A, but we cannot always prove the totality of the relation defined by A, even though it is in fact total. This means that there are specifications for which satisfying programs exist, but we will not be able to extract them, because there is no proof to begin with. We can however not expect to solve this by strengthening the proof system in some way, as Gödel's incompleteness shows that incompleteness already takes place on the level of Π_1 formulas, for any proof system that captures a basic amount of arithmetical reasoning. Any method of extracting programs from proofs of totality will therefore only work for those specifications that describe so-called *provably total functions*. However, as mentioned already in relation to system **T**, whose definable functions are exactly the provably total functions, these functions are a very rich subset of the computable functions, of up to high complexity, and include most of the commonly considered functions.

As the algorithms are tailored towards Π_2 sentences, an obvious idea for extending this method is to aim to make this work for sentences of higher complexity. This would allow one to define specifications in a richer language than the language of Σ_1 formulas. However, Σ_1 formulas are already sufficient to define a wide range of relations. All

computable functions are definable by a Σ_1 formula, with the help of Kleene's primitive recursive T-predicate (specifically, for all computable functions there exists a number e such that for all n, $f(n) = m$ iff $\exists x(T(e, \bar{n}, x) \wedge U(x) = \bar{m})$), and therefore all provably total functions as well.

Besides there being no significant reason in terms of expressiveness to extend the algorithm to sentences different than Π_2, it is also not possible to actually do this. The main reason for this is that the proof translation from proofs in Peano arithmetic to proofs in Heyting arithmetic cannot be extended to sentences of arbitrary high complexity. For example, the sentence $\forall x \exists y \forall z(T(x, x, y) \vee \neg T(x, x, z))$, where T is Kleene's T-predicate, is easily provable with the help of the excluded middle rule, but is not a theorem of Heyting arithmetic. Deciding beforehand whether a classically proven theorem would be intuitionistically provable as well is not an option either, because deciding whether a classical theorem can be proved constructively is an undecidable problem.

Significant improvements can be made to this program extraction method in the direction of improving the quality of the extracted program. This can be achieved by post-processing the program in order to obtain for example a more efficient program with the same input-output behavior. A more natural approach to this however is to transform the proof from which the term is extracted. The form of the proof determines the form extracted function, so one could aim to transform the constructive proof in such a way that its extracted program will be guaranteed to have favorable properties. An elegant proof does not necessarily allow for the extraction of an efficient program. An example of such an approach that is implemented in the Minlog is described in chapter 7 in the book [SW12].

Bibliography

[BBS+98] Benl, Berger, Schwichtenberg, Seisenberger, and Zuber. *Proof Theory at Work: Program Development in the Minlog System*, pages 41–71. Springer Netherlands, Dordrecht, 1998.

[BL17] Matthias Baaz and Alexander Leitsch. On the complexity of translations from classical to intuitioistic proofs. In *IfCoLog Journal of Logics and their Applications*, volume 4 of *4*, pages 901—-938, 2017.

[coq] The coq proof assistent. `https://coq.inria.fr/`. Accessed: 3 June 2018.

[Fef96] Solomon Feferman. Kreisel's unwinding program. In Piergiorio Odifreddi, editor, *Kreiseliana: About and Around Georg Kreisel*, pages 247–273. A K Peters, 1996.

[Fri77] Harvey Friedman. Classically and intuitionistically provably recursive functions. In G.H. Müller and D.S. Scott, editors, *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–27. Springer-Verlag Berlin Heidelberg, 1977.

[Göd80] Kurt Gödel. On a hitherto unexploited extension of the finitary standpoint. *Journal of Philosophical Logic*, 9(2):133–142, May 1980.

[GPS17] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119, 2017.

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[Ili17] Danko Ilik. Perspectives for proof unwinding by programming languages techniques. *IfColog Journal of Logics and their Applications (FLAP)*, 4(10):3487–3508, 11 2017.

[Kle45] S. C. Kleene. On the interpretation of intuitionistic number theory. *The Journal of Symbolic Logic*, 10(4):109–124, 1945.

[Koh08] Ulrich Kohlenbach. Applied proof theory: Proof interpretations and their use in mathematics. 2008.

[Kre58] Georg Kreisel. Mathematical significance of consistency proofs. *J. Symb. Log.*, 23(2):155–182, 1958.

[Kre62] G. Kreisel. On weak completeness of intuitionistic predicate logic. *The Journal of Symbolic Logic*, 27(2):139–158, 1962.

[Lei15] Alexander Leitsch. On proof mining by cut-elimination. In D. Delahaye and B.Woltzenlogel Paleo, editors, *All about proofs, proofs for all*, volume 55 of *Logic and Foundations*, pages 173–200. College Publications, 2015.

[MW71] Zohar Manna and Richard J. Waldinger. Toward automatic program synthesis. *Commun. ACM*, 14(3):151–165, 1971.

[rU06] Morten Heine B. Sørensen and Pawel Urzyczyn. Lectures on the Curry-Howard isomorphism. *Studies in logic and the foundations of mathematics*, 149, 2006.

[SW12] Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and Computations*. Cambridge University Press, New York, NY, USA, 1st edition, 2012.