On the Constructive Content of Proofs

Monika Seisenberger

München 2003

On the Constructive Content of Proofs

Monika Seisenberger

Dissertation
an der Fakultät für Mathematik und Informatik
der Ludwig–Maximilians–Universität München

vorgelegt von
Monika Seisenberger
März 2003

Erstgutachter: Prof. Dr. H. Schwichtenberg
Zweitgutachter: Prof. Dr. W. Buchholz
Tag der mündlichen Prüfung: 10. Juli 2003

Abstract

This thesis aims at exploring the scopes and limits of techniques for extracting programs from proofs. We focus on constructive theories of inductive definitions and classical systems allowing choice principles. Special emphasis is put on optimizations that allow for the extraction of realistic programs.

Our main field of application is infinitary combinatorics. Higman's Lemma, having an elegant non-constructive proof due to Nash-Williams, constitutes an interesting case for the problem of discovering the constructive content behind a classical proof. We give two distinct solutions to this problem. First, we present a proof of Higman's Lemma for an arbitrary alphabet in a theory of inductive definitions. This proof may be considered as a constructive counterpart to Nash-Williams' minimal-bad-sequence proof. Secondly, using a refined A-translation method, we directly transform the classical proof into a constructive one and extract a program. The crucial point in the latter is that we do not need to avoid the axiom of classical dependent choice but directly assign a realizer to its translation.

A generalization of Higman's Lemma is Kruskal's Theorem. We present a constructive proof of Kruskal's Theorem that is completely formalized in a theory of inductive definitions.

As a practical part, we show that these methods can be carried out in an interactive theorem prover. Both approaches to Higman's Lemma have been implemented in Minlog.

Zusammenfassung

Ziel der vorliegenden Arbeit ist es, die Reichweiten und Grenzen von Techniken zur Extraktion von Programmen aus Beweisen zu erforschen. Wir konzentrieren uns dabei auf konstruktive Theorien Induktiver Definitionen und klassische Systeme mit Auswahlprinzipien. Besonderes Gewicht liegt auf Optimierungen, die zur Extraktion von realisischen Programmen führen.

Unser Hauptanwendungsgebiet ist die unendliche Kombinatorik. Higmans Lemma, ein Satz mit einem eleganten klassischen, auf Nash-Williams zurückgehenden Beweis, ist ein interessantes Fallbeispiel für die Suche nach dem konstruktiven Gehalt in einem klassischen Beweis. Wir zeigen zwei unterschiedliche Lösungen zu dieser Problemstellung auf. Zunächst präsentieren wir einen induktiven Beweis von Higmans Lemma für ein beliebiges Alphabet, der als konstruktives Pendant zum klassischen Beweis angesehen werden kann. Als zweiten Ansatz verwandeln wir mit Hilfe der verfeinerten A-Übersetzungsmethode den klassischen Beweis in einen konstruktiven und extrahieren ein Programm. Der entscheidende Punkt ist hierbei, dass wir einen direkten Realisierer für das übersetzte Auswahlaxiom verwenden.

Die Verallgemeinerung von Higmans Lemma führt zu Kruskals Satz. Wir geben einen konstruktiven Beweis von Kruskals Theorem, der vollständig auf den Induktiven Definitionen basiert.

Der praktische Teil der Arbeit befasst sich mit der Ausführbarkeit dieser Methoden und Beweise in dem Beweissystem MINLOG.

Acknowledgements

I wish to express my sincere thanks to Professor Helmut Schwichtenberg for supervising this thesis and for providing and supporting a very lively scientific environment. His constructive view of mathematical logic has widely influenced this thesis. I thank him for his scientific advice, his patient guidance and also the time he offered to me, not at least with extending the Minlog system according to my needs and wishes.

I am grateful to my colleagues in the Munich working group and in the Graduiertenkolleg 'Logik in der Informatik' for their interest and cooperation during all the years. In particular, I want to thank Professor Wilfried Buchholz for his advice concerning prooftheoretical questions, Felix Joachimski for his continuous help in both mathematical and other matters, Laura Crosilla for many stimulating discussions and her support, not only in presenting this thesis, and Stefan Berghofer for his interest in implementations of constructive proofs of Higman's Lemma.

Many others have helped me with their comments and suggestions. I am especially thankful to Professor Robert Constable for an encouraging discussion on the subject in Marktoberdorf, 2001, and to Daniel Fridlender for sharing his knowledge on constructive proofs of Higman's Lemma with me.

I am indebted to Professor John Tucker and the members of the Computer Science Department of the University of Wales Swansea for integrating me into a very strong theory group. Special thanks go to my colleagues Al-Fathiatul Habibah Abdul Rahman, Marco Mazzucco and Will Harwood who patiently answered all my questions concerning the English language.

My gratitude and my love go to Ulrich Berger. I thank him for his encouragement, his inspiring ideas and his overall support. Finally, I would like to thank my friends and my two families for their benevolence and their support to finish this work.

I gratefully acknowledge the support of the Graduiertenkolleg 'Logik in der Informatik' sponsored by the Deutsche Forschungsgemeinschaft (DFG) and the funding of the Engineering and Physical Sciences Research Council (EPSRC).

Contents

1 Introduction **1**

1.1 Constructive content of proofs 1

1.2 Higman's Lemma and Kruskal's Theorem 3

1.3 Aims and results of the thesis 5

1.4 Outline of the contents 6

2 Computational Content of Proofs using Inductive Definitions **9**

2.1 Extended Heyting Arithmetic 9

2.2 Inductive Definitions 15

2.3 Program extraction 16

3 Computational Content of Classical Proofs **26**

3.1 Refined A-translation 26

3.2 Computational content of classical dependent choice 31

3.3 An example for using external realizers 35

4 Higman's Lemma and Kruskal's Theorem **39**

4.1 Nash-Williams' minimal-bad-sequence proof 39

4.2 Equivalent characterizations of well quasiorderings 41

4.3 On constructive proofs of Higman's Lemma and Kruskal's Theorem . . . 45

5 An inductive version of Nash-Williams' proof of Higman's Lemma **49**

5.1 Basic definitions 49

5.2 The analogy between the classical and the constructive proof 50

5.3 Forests 52
5.4 The proof of Higman's Lemma 53
5.5 Formalization in MINLOG 57

6 *A*-translation of Nash-Williams' proof of Higman's Lemma 61
6.1 An equivalent formulation of classical dependent choice 61
6.2 Formalization of Nash-Williams' proof 64
6.3 Discussion of the extracted program 67

7 An inductive proof of Kruskal's Theorem 69

8 Conclusion and further work 78

References 81

A Implementations in the MINLOG system 89
A.1 *A*-translation using an external realizer for dependent choice 89
A.2 Inductive Definitions, an example 93
A.3 Higman's Lemma for a 0/1 alphabet 96
A.4 Higman's Lemma for a finite alphabet 103

1 Introduction

The idea that a constructive proof yields a program has been attractive to mathematicians and computer scientists for many years. It is succinctly referred to as the proofs-as-programs paradigm and originates in proof theoretical methods for analyzing constructive theories. More recently, the programming aspect of these methods has become more important, most notably, in view of a new methodology for the development of secure software. This thesis aims at exploring and comparing the scope and limits of these techniques as well as utilizing them for the synthesis of realistic programs.

1.1 Constructive content of proofs

In order to convey an intuition of the connection between a constructive proof and a program extracted from it, we recall the so called Brouwer-Heyting-Kolmogorov interpretation (e.g. [TvD88]) which assigns a constructive meaning to each logical connective and quantifier. According to the Brouwer-Heyting-Kolmogorov interpretation, for instance, a proof of an existence statement of the form $\exists x A$ is given by a witness a and a proof of $A(a)$, and a proof of a universal statement $\forall x A$ is given by a construction or method transforming an arbitrary individual a into a proof of $A(a)$. Hence, in the case of a statement of the form $\forall x \exists y A(x, y)$, a proof would yield, in principle, a 'program' which for a given input a produces an output b such that $A(a, b)$ holds. A difficulty in this interpretation, though, is that it is quite vague, since it does not specify what amounts to such a construction or method.

There are several methods to make the constructive content behind a proof more explicit. In classical proof theory, for instance, we may use cut elimination. Essentially, given a, say, arithmetical proof of an existence statement, we can first remove all applications of induction by embedding the proof into an infinitary system, further eliminate all cuts, and then may easily read off an instance of the existence statement. This method has led to many celebrated results in proof theory; however as a programming tool, it seems to be quite inefficient.

In 1945, Kleene [Kle45] introduced another technique, recursive realizability, which connects Brouwer's Intuitionism and the theory of general recursive functions [Kle60]. This method has been successful in proof theory as well, but also turned out to be fruitful with respect to obtaining programs. Kleene's realizability assigns to each statement A a new statement n `realizes` A where n is a number coding all information related to the realization of existential (and disjunctive) statements occurring in a proof of A. Therefore it may be seen as an instance of the Brouwer-Heyting-Kolmogorov-interpretation[1].

A third method we briefly want to address is Gödel's Dialectica interpretation [Göd58]

[1] According to Kleene, though, both Heyting's proof interpretation and Kolmogorov's problem interpretation, 'failed to help' him to his goal [Kle60].

which is applicable to both constructive and classical logic, but is not as direct as the realizability interpretation. Recent applications of the Dialectica interpretation, e.g., in approximation theory, have been given by Kohlenbach, see for instance, [KO03].

In this thesis we will focus on the realizability interpretation. We work with a variant due to Kreisel which is based on Heyting arithmetic in finite types and is known as modified realizability [Kre62, Tro73, Tro98]. Beside the lack of a coding (which is not needed when working in a typed system), the essential difference is that Kleene realizability produces partial functions as realizers whereas in modified realizability all realizers are total.

Realizability as a programming method becomes more realistic when supported by a machine. In a theorem prover proofs can be checked automatically and the extracted programs, by construction, are provably correct. Probably the first and still one of the most important realizations of the proofs-as-programs paradigm [BC85] is the Nuprl system [CAB+86]. Other successful theorem provers supporting program extraction are Coq [BBC+97] and the PX-system [Hay90]. Very recently the extraction mechanism has been included in the Isabelle system [NPW02] (see [Ber03]). The proof system we are working with, the Minlog system, differs from most of these systems in that it is not built on constructive type theory. Minlog is based on first order natural deduction and is intended to reason about computable functionals in finite types using minimal rather than classical or intuitionistic logic. For a detailed description of the Minlog system we refer to [BBS+98]. It is one of the aims of this thesis to explore the feasibility of larger case studies in Minlog.

Since its introduction, realizability has been analyzed in manifold respects, e.g., the logics, the strength of the systems and the fields of application. Originally proposed for Heyting arithmetic, it has been investigated in Martin-Löf type theory, in second order systems, and intuitionistic and constructive set theory (for the latter see [Bee85, McC84, Cro00]).

Proof theoretically very weak systems have been examined, for instance, in [Hof99, AH02, ABHS03], the idea being to restrict the logic in such a way that the extracted programs are of a certain sub-recursive complexity. Our investigations are located at the other end: we try to explore the limits of realizability by applying it to logically 'strong' systems. Our first extension comprises generalized inductive definitions, our second classical arithmetic using choice principles.

Inductive definitions form a thoroughly investigated powerful proof tool (e.g., [Acz77, BFPS81, Mat98]) and are used in the development of large parts of mathematics. Elegant applications may be found, for instance, in proving termination in the context of term rewriting theory (e.g., [Buc95] or [Per99]) but also in combinatorics, as we will see later.

Inductive definitions together with program extraction have also been investigated for the calculus of constructions [PM89, PMW93] and for second order systems, e.g. in [Par92, MP02]. We present a version that fits the Minlog system, i.e., it comprises

program extraction for Heyting arithmetic extended by inductive types and inductive definitions.

With regard to classical logic, there exist roughly two types of methods for getting hold of the constructive content of (a restricted class of) proofs, either by giving classical proofs a direct computational interpretation (compare e.g. [FFKD87, KP90, BS95a]), or by translating the classical proof into a constructive one. Although apparently different, these methods are closely related in many cases (see e.g. [Mur91, Mur93]). It is unknown which of the two methods is better suited for applications. In this thesis we will concentrate on the translation method, which is mostly referred to as Friedman's A-translation, but was independently introduced by Dragalin [Dra79]. The method was further investigated in [Lei85, TvD88] and made applicable to larger problems by refinements in [BS95b, BBS02]. The refined A-translation is implemented in the MINLOG system, for applications, see for instance [BS96, BSS01].

Choice principles play a crucial role in classical mathematics. From a constructive point of view, the combination of the excluded middle and choice principles turns out to be problematic, the main reason being that their negative translations fail to be intuitionistic consequences of the original axioms [BBC98]. With respect to extracting programs, this problem may be overcome by directly assigning realizers to the translated choice axioms. Such realizers were proposed by Beradi, Bezem and Coquand in [BBC98] and Berger and Oliva in [BO03]. We will use this approach to extract computational content of proofs using classical choice principles.

Our main field of application is infinitary combinatorics. We will explore the computational content of classical proofs and also give some new constructive proofs. The main examples are Higman's Lemma and Kruskal's Theorem.

1.2 Higman's Lemma and Kruskal's Theorem

Higman's Lemma [Hig52] and and Kruskal's Theorem [Kru60] are two well-known theorems in infinitary combinatorics that comprise statements of the form 'for every infinite sequences of words, trees respectively, we can find two elements in the sequence such that the first one is 'embeddable' in the second one. In order to characterize embeddability and formulate the two theorems we need some more technical definitions.

Definition. Let A^* be the set of finite sequences, also called words, with elements in a set A, often called alphabet, equipped with a binary relation $\leq_A$ on A. A sequence $[a_1, \ldots, a_n]$ is embeddable in $[b_1, \ldots, b_m]$ if there is a strictly increasing map $f\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_i \leq_A b_{f(i)}$ for all $i \in \{1, \ldots n\}$. Moreover, let $T(A)$ be the set of finite trees with labels in A. A tree is embeddable into another one if there exists a one to one map on them such that (1) the infima of nodes are respected and (2) the label of each node is less or equal to that of its image. We denote the embeddability relation on words over A by $\leq_{A^*}$, that on trees by $\leq_{T(A)}$.

Higman's Lemma and Kruskal's Theorem are usually formulated in terms of well quasiorders.

Definition. Let $(Q, \leq)$ be a quasiorder (i.e., $\leq$ is reflexive and transitive). Then, $(Q, \leq)$ is a well quasiorder (wqo) if every infinite sequence in Q is 'good', i.e.,

$$\forall (q_i)_{i<\omega} \exists i, j. i < j \wedge q_i \leq q_j.$$

Proposition 1.1 (Higman's Lemma).

If $(A, \leq_A)$ is a well quasiorder,
then so is the set $(A^*, \leq_{A^*})$ of finite sequences in A.

Proposition 1.2 (Kruskal's Theorem).

If $(A, \leq_A)$ is a well quasiorder,
then so is the set $(T(A), \leq_{T(A)})$ of finite trees.

Higman's Lemma and Kruskal's theorem are of interest in proof theory, computer science and mathematics alike.

Both theorems have an elegant, classical proof due to Nash-Williams [NW63] using the so-called minimal-bad-sequence argument. This proof (see chapter 4) will play a special role in our thesis. More direct proofs of Higman's Lemma have been given in [dJP77, Sch79, SS85, MR90, RS93, CF94, CTB94] either using ordinal notation systems or inductive definitions. In [Vel00] an intuitionistic proof for relations not required to be decidable is provided; this proof has been transformed in [Fri97] into a type theoretic proof which only uses inductive definitions. Direct proofs for Kruskal's theorem can be found in [Sch79, RW93, Has94, Vel00]. In chapter 4 we shall discuss these proofs in more detail and classify them.

The proof-theoretic strength of Higman's Lemma is that of Peano Arithmetic, i.e., ϵ_0, as was shown in [Gir87] using the constructive proof in [SS85]. The proof in [SS85] is based on the determination of the maximal ordertype, that is, the height of the tree of all bad sequences, which for Higman's Lemma is ω^ω [dJP77, Sch79]. Kruskal's theorem yields an example of a very natural statement not provable in Peano Arithmetic - not even in ATR_0 [Sim85]. Its proof theoretic strength is $\vartheta(\Omega^\omega)$ [RW93], an ordinal between Γ_0 and the Bachmann-Howard ordinal $\vartheta(\epsilon_{\Omega+1})$.

In term rewriting theory, Higman's Lemma and Kruskal's Theorem are used to prove termination of string rewriting systems and term rewriting systems respectively. The orders whose termination is covered by these two theorems are called simplification orders. They form an important class since the criterion of being a simplification order can be checked syntactically. A constructive proof, e.g., as given in [CTB94], moreover yields a bound for the longest possible bad sequence. In the case of Higman's Lemma the reduction length, expressed in terms of the Hardy hierarchy, H, assuming a finite

alphabet A, is as follows. If we have a bad sequence $(t_i)_{i<n}$, fulfilling the condition $|t_i| \leq |t_0| + k \times i$, where k is a constant and $|t|$ denotes the size of t, then the length n of the sequence is bound by $\Phi(|t_0|)$ where Φ is an elementary function in $H_{\omega^{\omega^{|A|}}}$ [CTB94, Tou97]. For Kruskal's theorem, using the size restriction $|t_i| \leq |t_0| + k \times i^2$, similar results hold: the bound is $H_{\vartheta\Omega^\omega}$ [Wei94]. Both bounds are essentially optimal since there are term rewriting systems which 'reach' these bounds [Tou02, Lep01].

Finally, both theorems are valuable in graph theory since they are special cases of the graph minor theorem, the analogous theorem for graphs where instead of an embedding we have the so-called graph minor relation. (A graph is a minor of another if the first can be obtained from the second by deleting and contracting edges.) The main ingredient for the proof of the graph minor theorem is a generalization of Kruskal's theorem, the so-called extended Kruskal theorem or Kruskal's theorem with gap condition for which up to now only a classical proof is known.

1.3 Aims and results of the thesis

The general problem addressed by this thesis may be summarized by the question: 'How to find effective solutions to problems known to be solvable', or, in more technical terms, 'how to find computational content in proofs of $\forall\exists$-statements.' This task is particularly interesting when the proof given for the $\forall\exists$-statement is non-constructive.

Higman's Lemma, having a very elegant classical proof, constitutes an important example of such a theorem, the relevant question now being: What is the computational content behind its classical proof? We already know several constructive proofs of Higman's Lemma. By a closer inspection we see that the combinatorial idea behind these proofs is different from the one behind the classical proof (cf. chapter 4). Therefore the second question arises of how we can obtain a constructive proof whose computational content corresponds to that of the classical proof. In this thesis we present two distinct solutions to this question:

- We present a proof of Higman's Lemma for an arbitrary alphabet in a theory of inductive definitions that may be considered as a constructivization of the minimal-bad-sequence proof due to Nash-Williams.
- Using A-translation, we directly transform the classical proof of Higman's Lemma into a constructive one and extract the program.

The first approach has its origins in a proof for Higman's Lemma restricted to a two letter alphabet given by Coquand and Fridlender [CF94].

The second approach has been pursued by Murthy [Mur90] and resulted in a rather large program; the program we obtain will be shorter, mainly for the following reasons. First, we apply a refined version of the A-translation and secondly, we assign computational

content directly to the axiom of classical dependent choice, hence we do not need to reformulate the classical proof in order to avoid the choice principle.

On the theoretical side, two extensions of the program extraction method have been necessary to achieve these goals.

- We extend the realizability interpretation from Heyting arithmetic in finite types to strictly positive inductive definitions in such a way that it exactly fits the MINLOG system.
- We extend the A-translation mechanism to allow assumptions with an external realizer, as will be exemplified by the translation of classical dependent choice and provide concrete examples.

Both extensions are supported by the MINLOG system. We explore the feasibility of these methods by means of our examples and

- give an implementation of both approaches to Higman's Lemma and at each case extract a program.

As to our second main example, Kruskal's theorem, the situation is slightly different. Rathjen and Weiermann [RW93] have presented a constructive proof of Higman's Lemma using ordinal notations. Hasegawa [Has94] has given a proof, using a system of algebras, similar to ordinal notations. Furthermore, Veldman has provided an intuitionistic proof in [Vel00]. However, there is no direct constructive proof of Kruskal's theorem (as was expressed in [Per99]) which allows for a straightforward formalization. As a solution to this problem

- we present a constructive proof of Kruskal's theorem that only uses inductive definitions.

This proof might be seen as the counterpart to the proofs of Higman's Lemma in [RS93, Fri93] (cf. the classification of proofs given in section 4.3). In analogy to Higman's Lemma, the questions that remain for the future are, how an inductive proof of Kruskal's theorem which directly corresponds to the classical proof can be obtained, and whether it is possible to apply the A-translation method to Kruskal's theorem. We believe that the second question can be answered positively, even in the case of the Extended Kruskal Theorem.

1.4 Outline of the contents

We conclude this introduction with an overview of the subsequent chapters.

The thesis has three main parts, two theoretical and one practical. The first part comprises program extraction and its extensions, the second contains new proofs of Higman's Lemma and Kruskal's Theorem and the third part, mainly the appendix, is about the formalization in the MINLOG system.

In Chapter 2 we describe program extraction from constructive proofs and extend the realizability interpretation to strictly positive inductive definitions in such a way that it can be formalized in the theorem prover MINLOG. Particular emphasis thereby lies in an optimization which avoids redundant variables in the extracted program.

Chapter 3 is devoted to program extraction from classical proofs allowing assumptions with external realizers. We give a short presentation of the refined A-translation method [BBS02] and combine it with results of [BO03]. Finally, we add a first example in which we use a realizer for the negative translation of dependent choice to underpin this method.

Chapter 4 gives an overview on the classical and constructive proofs of Higman's Lemma and Kruskal's Theorem. To this end we compare several methods of proving a set well quasiordered and show how the constructive proofs are related. The chapter concludes with a classification of all known proofs.

Chapter 5 aims at a proof of Higman's Lemma for an arbitrary well quasiordered alphabet in a theory of inductive definitions that uses the same combinatorial idea as the non-constructive proof of Nash-Williams.

In Chapter 6, we study a variant of the axiom of dependent choice and use it for the implementation and A-translation of the classical proof of Higman's lemma. We briefly discuss the extracted program and give an alternative animation of the realizer for the axiom of dependent choice by directly adding an optimized scheme program.

A new constructive proof of Kruskal's Theorem in a theory of inductive definitions is presented in chapter 7.

In the appendix we show the implementations and the extracted programs for the examples given in the chapters 3, 4 and 5.

2 Computational Content of Proofs using Inductive Definitions

Our first chapter is devoted to the computational content of constructive proofs. First of all, we shall fix the formal system we are working in. We start with a description of Heyting Arithmetic, extended by inductive types and inductive definitions, and, explain the mechanism of extracting programs. The main emphasis thereby lies on the treatment of inductive definitions, where we distinguish between inductive definitions with and without computational content. A further optimization consists in allowing two sorts of quantifiers, the usual ones and those carrying no computational content, to avoid redundant variables in the programs. This thesis is essentially self-contained; however, we assume some familiarity with proofs presented in a Curry-Howard style and with realizability interpretations and directly develop the system as it is needed for this thesis. The treatment of inductive types is inspired by [Ber95] and [Ben98]. The optimization of allowing quantifiers with no computational content was first suggested in [Ber93].

2.1 Extended Heyting Arithmetic

Our term calculus is HA^{μ} which is an extension of Heyting Arithmetic in finite types, HA^{ω} (see [Tro73] or [TvD88]), by inductively defined types, also called free algebras.

Types and Terms

Definition. Types are generated from inductive types, denoted μ, via $\times$ and $\to$, that is, if ρ and σ are types, then so are $\rho \times \sigma$ and $\rho \to \sigma$; in short: types are

$$\mu \,|\, \rho \times \sigma \,|\, \rho \to \sigma.$$

A new inductive type μ is introduced by the following equation:

$$\begin{aligned} \mu \;=\; & c_1\big(\vec{\rho}_1,\, \vec{\sigma_{11}} \to \mu,\, \ldots,\, \vec{\sigma_{1m_1}} \to \mu\big) \\ & \qquad + \cdots + \\ & c_n\big(\vec{\rho}_n,\, \vec{\sigma_{n1}} \to \mu,\, \ldots,\, \vec{\sigma_{nm_n}} \to \mu\big) \end{aligned}$$

where for all i, j such that $1 \le i \le n,\ 1 \le j \le m_i,\ 0 \le m_i, n$, $\vec{\rho}_i$ and $\vec{\sigma_{ij}}$ are lists of types built from previously defined types only. Then, μ is the type whose elements are generated from the constructors

$$c_i : \vec{\rho}_i \to \overrightarrow{\vec{\sigma}_i \to \mu} \to \mu.$$

Conventions. $\vec{\rho}$ denotes a list of types: $\rho_1, \ldots, \rho_{|\vec{\rho}|}$. Moreover, we use the abbreviation $\vec{\rho} \to \sigma$ for the type $\rho_1 \to \cdots \to \rho_{|\vec{\rho}|} \to \sigma$.

Remark. 1. The notion of an inductive type can be extended by assigning names to the destructors of the inductive types, as well. Thus, in

$$\begin{aligned}\mu \;=\;& c_1\big(\vec{p_1} : \vec{\rho_1},\ q_{11} : \vec{\sigma_{11}} \to \mu, \ldots, q_{1m_1} : \vec{\sigma_{1m_1}} \to \mu\big)\\ &+ \cdots +\\ & c_n\big(\vec{p_n} : \vec{\rho_n},\ q_{n1} : \vec{\sigma_{n1}} \to \mu, \ldots, q_{nm_n} : \vec{\sigma_{nm_n}} \to \mu\big)\end{aligned}$$

we, in addition, fix the destructors

$$\begin{aligned} p_{ij} :&\ \ \mu \to \rho_{ij}\\ q_{ij} :&\ \ \mu \to (\vec{\sigma_{ij}} \to \mu).\end{aligned}$$

2. Note also that these definitions can be extended to simultaneously defined types in the obvious way.

Examples. 1. $\mathsf{boole} = \mathsf{true} + \mathsf{false}$

2. $\mathsf{nat} = 0 \ + \ \mathsf{Succ}(\mathsf{Pred} : \mathsf{nat})$

3. $\mathsf{tsil}\,\alpha$ is the type for reverse lists over the type α, i.e.,

$$\mathsf{tsil}\,\alpha = \mathsf{Lin} + \mathsf{Snoc}(\mathsf{Lead} : \mathsf{tsil}\,\alpha, \mathsf{Last} : \alpha)$$

with constructors $\mathsf{Lin} : \mathsf{tsil}\,\alpha$ and $\mathsf{Snoc} : \mathsf{tsil}\,\alpha \to \alpha \to \mathsf{tsil}\,\alpha$ and destructors $\mathsf{Lead} : \mathsf{tsil}\,\alpha \to \mathsf{tsil}\,\alpha$ and $\mathsf{Last} : \mathsf{tsil}\,\alpha \to \alpha$.

Definition. Terms are built from typed variables and constants (including constructors and destructors) via λ-abstraction, application, pairing and projection, that is, terms are

$$x \mid c \mid \lambda x t \mid st \mid \langle s, t\rangle \mid \pi_i(t).$$

For each ground type μ and type τ we have a recursion operator $R_{\mu,\tau}$ which allows us to define functions from μ to τ via recursion on the structure of μ. That is, if

$$\vec{\rho_i} \to \overrightarrow{\vec{\sigma_i} \to \mu} \to \mu$$

is the type of the i-th constructor c_i of μ, then the i-th step type δ_i is

$$\vec{\rho_i} \to \overrightarrow{\vec{\sigma_i} \to \mu} \to \overrightarrow{\vec{\sigma_i} \to \tau} \to \tau$$

and the recursion operator has the type

$$\mathsf{R}_{\mu,\tau} : \delta_1 \to \cdots \to \delta_n \to \mu \to \tau.$$

Analogously, we also have a case distinction operator

$$C_{\mu,\tau} : \delta_1 \to \cdots \to \delta_n \to \mu \to \tau$$

where the i-th step type δ_i simplifies to $\vec{\rho_i} \to \overrightarrow{\vec{\sigma_i} \to \mu} \to \tau$.

Conventions.

1. We use x, y, z, u, v, w for variables and r, s, t for terms. The type information can be written as follows: t^ρ or $t : \rho$. For sake of readability, it is often omitted.
2. $\mathrm{FV}(t)$ comprises the free variables of t. $t[x/s]$ denotes substitution of the variable x by the term s in the term t, renaming variables if necessary.

Conversions. The conversion rules are

$$\begin{array}{rcl} (\lambda x t)s & \mapsto & t[x/s] \\ (\lambda x t)x & \mapsto & t, \quad x \notin \mathrm{FV}(t) \\ \pi_i(\langle t_0, t_1\rangle) & \mapsto & t_i,\ i = 0, 1 \\ \langle \pi_0(t), \pi_1(t)\rangle & \mapsto & t \end{array}$$

With regard to the recursion operator, assuming that $\vec{t}$ consists of parameter arguments ${t_1}^P, \ldots, {t_m}^P$ and recursive arguments ${t_1}^R, \ldots, {t_n}^R$, we have the conversion rule

$$\mathsf{R}_{\mu,\tau}\vec{s}(c_i\vec{t}) \mapsto s_i\vec{t}(\mathsf{R}_{\mu,\tau}\vec{s} \circ {t_1}^R) \ldots (\mathsf{R}_{\mu,\tau}\vec{s} \circ {t_n}^R)$$

where $r^{\sigma\to\tau} \circ t^{\vec{\rho}\to\sigma} := \lambda\vec{y}^{\vec{\rho}}.(r(t\vec{y}))$.

The analogous rule for the case distinction operator is

$$C_{\mu,\tau}\vec{s}(c_i\vec{t}) \ \mapsto \ s_i\vec{t}.$$

Remark. 1. We identify terms with the same normal form. 2. We also could have an iterative variant of the conversion rule for the recursion operator, i.e.,

$$\mathsf{R}_{\mu,\tau}\vec{s}(c_i\vec{t}) \mapsto s_i(\mathsf{R}_{\mu,\tau}\vec{s} \circ {t_1}^R) \ldots (\mathsf{R}_{\mu,\tau}\vec{s} \circ {t_n}^R)$$

where we do not use the parameters $\vec{t}$ as arguments of the step terms s_i. However, for our purposes the recursive version is appropriate, as we will see later, since it exactly matches with the strengthened induction principle for inductive definitions.

Formulas

We define formulas as well as the type of a formula relative to a given set of predicate symbols and a type assignment on this set. The type of a formula will be the type of the program extracted from a proof of that formula.

In the definition of formulas we introduce two sorts of quantifiers, the usual quantifiers $\forall, \exists$ and quantifiers $\forall^{\mathsf{nc}}$, $\exists^{\mathsf{nc}}$ carrying **n**o computational **c**ontent. In proofs, the use of $\forall^{\mathsf{nc}}$ is only allowed if an additional condition is fulfilled. The formulation of this condition also refers to the type of a formula.

Definition. Let $\mathcal{P}$ be a set of predicate symbols, each of a fixed arity $\vec{\rho} = \rho_1, \ldots, \rho_n$. We always assume $\mathcal{P}$ to contain a nullary predicate symbol $\bot$ and a predicate symbol atom of arity boole. Formulas are built from atomic formulas $P(\vec{t})$ $(P \in \mathcal{P})$ via implication, conjunction and quantification. Hence formulas are

$$P(\vec{t}) \mid A \to B \mid A \wedge B \mid \forall x^\rho A \mid \forall^{\mathsf{nc}} x^\rho A \mid \exists x^\rho A \mid \exists^{\mathsf{nc}} x^\rho A.$$

where in $P(\vec{t})$ we assume that P is of arity $\vec{\rho}$ and the terms $\vec{t} = t_1, \ldots, t_n$ are of types $\rho_1, \ldots, \rho_n$ respectively.

Conventions and remarks.

1. We use A, B, C, D for formulas, P, Q for predicates and I for inductively defined predicates (cf. next section). $A[x/t]$ denotes the substitution of the variable x by the term t. We write $P : \vec{\rho}$ if the predicate P has arity $\vec{\rho}$.
2. If A is a formula, then $\{\vec{x} \mid A\}$ is a predicate, also called comprehension term. We identify $\{\vec{x} \mid A\}\vec{t}$ with $A[\vec{x}/\vec{t}]$.
3. The predicate symbol atom transforms a boolean term t into an atomic formula. We will carefully distinguish between $F := \mathsf{atom}\ \mathsf{false}$ and the predicate symbol $\bot$ when explaining the mechanism of transforming classical proofs into constructive ones (cf. next chapter).
4. We use the notation $\forall^{(\mathsf{nc})}, \exists^{(\mathsf{nc})}$ if both quantifiers can be used.

Definition (The type of a formula). To every formula A we inductively assign an object $\tau(A)$ that is either a type or the symbol $*$. The assignment is relative to a given (partial) type assignment τ_0 for predicates.

$$\tau(P(\vec{t})) := \begin{cases} \tau_0(P) & \text{if } P \in \mathcal{P} \text{ with assigned } \tau_0(P) \\ * & \text{otherwise.} \end{cases}$$

$$\tau(A \to B) := \begin{cases} \tau(B) & \text{if } \tau(A) = *, \\ * & \text{if } \tau(B) = *, \\ \tau(A) \to \tau(B) & \text{otherwise.} \end{cases}$$

$$\tau(A_0 \wedge A_1) := \begin{cases} \tau(A_i) & \text{if } \tau(A_{1-i}) = *, \\ \tau(A_0) \times \tau(A_1) & \text{otherwise.} \end{cases}$$

$$\tau(\forall x^\rho A) := \begin{cases} * & \text{if } \tau(A) = *, \\ \rho \to \tau(A) & \text{otherwise.} \end{cases}$$

$$\tau(\exists x^\rho A) := \begin{cases} \rho & \text{if } \tau(A) = *, \\ \rho \times \tau(A) & \text{otherwise.} \end{cases}$$

For formulas with a quantifier containing no computational content the obvious definition is

$$\begin{array}{lcl} \tau(\forall^{\mathsf{nc}} x^\rho\, A) & := & \tau(A) \\ \tau(\exists^{\mathsf{nc}} x^\rho\, A) & := & \tau(A) \end{array}$$

A formula is called computationally meaningful or said to have computational content if $\tau(A) \neq *$.

Definition (Invariant formulas). Formulas without the (computationally meaningful) existential quantifier $\exists$, which only contain predicates without computational content, are called invariant formulas.

Proofs

Proofs are presented as lambda terms via the Curry-Howard correspondence and are defined simultaneously with the set FA of free assumptions. We first give a rough idea of what is so special about the quantifier $\forall^{\mathsf{nc}}$. Logically, $\forall^{\mathsf{nc}}$ behaves similar to the usual quantifier $\forall$, albeit the behavior will change when it comes to program extraction. The main idea is the following: Suppose we conclude

$$\frac{A}{\forall x A}$$

and, beside the usual variable condition, the variable x is not free in any term t used in the derivation d of A, (that is, in $(\forall^-)$ or $(\exists^+)$). Then it is easy to see that the subprogram corresponding to the proof $d : A$ will not depend on x. So, while extracting the program, we may omit the variable x completely and in order to bookmark that we are allowed to do this, we 'assign' the label 'nc' to the quantifier.

Formally, this is done by determining a set CV of 'computationally relevant variables', i.e., variables occurring free in a (sub)derivation, and formulating a strengthened variable condition for the $\forall^{\mathsf{nc}}$-rule.

Similar conditions would be necessary in the elimination rule for the $\exists^{\mathsf{nc}}$-quantifier. However, since we treat the existential quantifier via axioms, the only difference is that the $\exists^{\mathsf{nc}}$-axioms refer to the $\forall^{\mathsf{nc}}$-quantifier.

Definition. Proofs are

$$\begin{array}{l} u^A \mid c^A \ (c \text{ an axiom}) \mid (\lambda u^A d^B)^{A\to B} \mid (d^{A\to B} e^A)^B \mid \\ (\langle d^A, e^B\rangle)^{A\wedge B} \mid (\pi_0(d^{A\wedge B}))^A \mid \pi_1((d^{A\wedge B}))^B \mid \\ (d^{\forall x A} t)^{A(t)} \mid (d^{\forall^{\mathsf{nc}} x A} t)^{A(t)} \mid \\ (\lambda x d)^{\forall x A},\ x \notin \mathrm{FV}(C) \text{ for } u^C \in \mathrm{FA}(d) \mid \\ (\lambda x d)^{\forall^{\mathsf{nc}} x A},\ x \notin \mathrm{CV}(d) \cup \mathrm{FV}(C) \text{ for } u^C \in \mathrm{FA}(d) \end{array}$$

where for a given derivation $d : A$, $\mathrm{CV}(d)$ and $\mathrm{FA}(d)$ are defined as follows.

If $\tau(A) \neq *$, then

(ass)	$\mathrm{FA}(u) := \{u\}$	$\mathrm{CV}(u) := \emptyset$
(ax)	$\mathrm{FA}(c) := \emptyset$	$\mathrm{CV}(c) := \emptyset$
$(\to^+)$	$\mathrm{FA}(\lambda u.d) := \mathrm{FA}(d)\backslash\{u\}$	$\mathrm{CV}(\lambda u.d) := \mathrm{CV}(d)$
$(\to^-)$	$\mathrm{FA}(de) := \mathrm{FA}(d) \cup \mathrm{FA}(e)$	$\mathrm{CV}(de) := \mathrm{CV}(d) \cup \mathrm{CV}(e)$
$(\wedge^+)$	$\mathrm{FA}(\langle d, e\rangle) := \mathrm{FA}(d) \cup \mathrm{FA}(e)$	$\mathrm{CV}(\langle d, e\rangle) := \mathrm{CV}(d) \cup \mathrm{CV}(e)$
$(\wedge^-)$	$\mathrm{FA}(\pi_i(d)) := \mathrm{FA}(d)$	$\mathrm{CV}(\pi_i(d)) := \mathrm{CV}(d)$
$(\forall^+)$	$\mathrm{FA}((\lambda x.d)^{\forall x A}) := \mathrm{FA}(d)$	$\mathrm{CV}((\lambda x.d)^{\forall x A}) := \mathrm{CV}(d)\backslash\{x\}$
$(\forall^{\mathsf{nc}+})$	$\mathrm{FA}((\lambda x.d)^{\forall^{\mathsf{nc}} x A}) := \mathrm{FA}(d)$	$\mathrm{CV}((\lambda x.d)^{\forall^{\mathsf{nc}} x A}) := \mathrm{CV}(d)$
$(\forall^-)$	$\mathrm{FA}(d^{\forall x A} t) := \mathrm{FA}(d)$	$\mathrm{CV}(d^{\forall x A} t) := \mathrm{CV}(d) \cup \mathrm{FV}(t)$
$(\forall^{\mathsf{nc}-})$	$\mathrm{FA}(d^{\forall^{\mathsf{nc}} x A} t) := \mathrm{FA}(d)$	$\mathrm{CV}(d^{\forall^{\mathsf{nc}} x A} t) := \mathrm{CV}(d)$

Otherwise, i.e., if $\tau(A) = *$, we set $\mathrm{CV}(d^A) := \emptyset$ and $\mathrm{FA}(d^A)$ is defined as above.

It remains to provide the axioms. First, the existential quantifier is treated by the axioms:

$(\exists^+_{x,A})$	$\forall x.A \to \exists x A$	
$(\exists^-_{x,A,B})$	$\exists x A \to (\forall x.A \to B) \to B,$	$(x \notin \mathrm{FV}(B))$
$(\exists^{\mathsf{nc}+}_{x,A})$	$\forall^{\mathsf{nc}} x.A \to \exists^{\mathsf{nc}} x A$	
$(\exists^{\mathsf{nc}-}_{x,A,B})$	$\exists^{\mathsf{nc}} x A \to (\forall^{\mathsf{nc}} x.A \to B) \to B,$	$(x \notin \mathrm{FV}(B))$

Second, we have the logical axioms Truth: T and, depending on our logic, the efq-axioms $F \to A$ and $\bot \to A$ for any formula A.

Third, we need axioms for equality, such as reflexivity, transitivity, symmetry and compatibility, all written with the quantifier $\forall^{\mathsf{nc}}$, e.g.,

(Compat) $\quad \forall^{\mathsf{nc}} \vec{x}, x, y.x = y \to P(x) \to P(y).$

Fourth, for each algebra, μ, we have axioms for case distinction and induction, denoted by $\mathsf{Cases}_{\mu,A}$ and $\mathsf{Ind}_{\mu,A}$. For simplicity, here we only communicate the axiom for case distinction for the type boole which will be needed, at some places, later

$(\mathsf{Cases}_{\mathsf{boole},A})$ $\quad A(\mathsf{true}) \to A(\mathsf{false}) \to \forall p^{\mathsf{boole}}.\, A(p).$

Finally, in the next section we will introduce inductive definitions; giving rise to introduction and elimination axioms.

The set of axioms may be further extended by the user.

Remark. The quantifier $\forall^{\mathsf{nc}}$ has been introduced in [Ber93] in order to extract the normalization-by-evaluation algorithm from Tait's normalization proof for the simply typed lambda calculus. We have adopted the idea of avoiding redundant variables since, in particular, it seems to be adequate for the introduction and elimination axioms for inductive definitions (see next section).

2.2 Inductive Definitions

Introduction and Elimination Axioms

Usually, a (strictly positive) inductive definition I is given by n rules where the i-th rule is of the form

$$\frac{A_i \wedge \bigwedge\!\!\!\bigwedge_j \forall \vec{y_{ij}}.\, \overrightarrow{B_{ij}} \to I(\vec{s_{ij}})}{I(\vec{t_i})}$$

where A_i and $\overrightarrow{B_{ij}}$ are arbitrary formulas, $\vec{t_i}$ and $\vec{s_{ij}}$ are terms with possibly free variables $\vec{x_i}$ respectively $\vec{x_i}$ and $\vec{y_{ij}}$. The notation $\overrightarrow{A} \to B$ is used for $A_1 \to \ldots \to A_m \to B$. In this presentation, we prefer to work with axioms instead of rules.

Definition (Inductively defined predicate). An inductively defined predicate $I : \rho_1, \ldots, \rho_l$ is introduced by n closure axioms, $K_1[I], \ldots, K_n[I]$,$1 \leq n$, (also called introduction axioms), where

$$K_i[I] := \forall \vec{x_i}, \forall^{\mathsf{nc}} \vec{x_i}'. A_i \to \overrightarrow{\forall \vec{y_{ij}}, \forall^{\mathsf{nc}} \vec{y_{ij}}'.\, \overrightarrow{B_{ij}} \to I(\vec{s_{ij}})} \to I(\vec{t_i}).$$

Given a predicate P, let $K_i[P]$ be the formula which is obtained by replacing the predicate I in $K_i[I]$ by P. Then, the induction principle (also called elimination axiom) is

$$K_1[P] \to \ldots \to K_n[P] \to \forall^{\mathsf{nc}} \vec{z}. I(\vec{z}) \to P(\vec{z}).$$

The strengthened induction principle

It can easily be proven that this induction principle can be strengthened: instead of the closure $K_i[P]$ we only have to show this closure for $\{\vec{x} | I(\vec{x}) \wedge P(\vec{x})\}$ which is logically equivalent to

$$K_i[I, P] := \forall \vec{x_i}. \overrightarrow{A_i} \to \overrightarrow{\forall \vec{y_i}.\, \overrightarrow{B_i} \to I(\vec{s_i})} \to \overrightarrow{\forall \vec{y_i}.\, \overrightarrow{B_i} \to P(\vec{s_i})} \to P(\vec{t_i}).$$

The induction principle then reads

$$\mathsf{Ind}_{I,P} := K_1[I, P] \to \ldots \to K_n[I, P] \to \forall^{\mathsf{nc}} \vec{z}. I(\vec{z}) \to P(\vec{z}).$$

We will mainly work with this strengthened principle; it trivially implies the simple one, and on the program side it exactly matches the recursion operator given in the preceeding section whereas the simple induction principle corresponds to iteration.

The type of an inductive definition

In the last section we have defined the type of each formula assuming an already given type assignment τ_0 for the predicates. Here, we show how the type assignment for a new (inductive) predicate symbol should be chosen. In case, we have an inductive definition in which the formulas A_i and B_{ij} in the closure axioms are invariant (i.e., do not contain $\exists$ or a predicate with computational content), we may decide whether or not this inductive definition should have computational content. In all other cases we assign an inductive datatype whose constructors have the same type as the closure axioms.

Definition (The type of an inductively defined predicate). In the case of an inductive predicate I with computational content, given by the axioms $K_1[I], \ldots, K_n[I]$ where

$$K_i[I] := \forall \vec{x}_i^{\vec{\rho}_i}, \forall^{\mathsf{nc}} \vec{x}_i'^{\vec{\rho}_i'}. \overrightarrow{A_i} \to \overrightarrow{\forall \vec{y}_i^{\vec{\sigma}_i}, \forall^{\mathsf{nc}} \vec{y}_i'^{\vec{\sigma}_i'}. \overrightarrow{B_i} \to I(\vec{s}_i)} \to I(\vec{t}_i),$$

we set

$$\tau_0(I) := \mu,$$

where μ is either inductively defined by

$$\begin{aligned} \mu \;=\; & c_1\big(\vec{\rho}_1, \tau(\overrightarrow{A_1}), \overrightarrow{\vec{\sigma}_1 \to \tau(\overrightarrow{B_1}) \to \mu}\big) \\ & + \cdots + \\ & c_n\big(\vec{\rho}_n, \tau(\overrightarrow{A_n}), \overrightarrow{\vec{\sigma}_n \to \tau(\overrightarrow{B_n}) \to \mu}\big) \end{aligned}$$

with new constructors $c_1, \ldots, c_n$ or it is an existing inductive type with constructors of the same type. Here, we have written $\tau(\vec{A})$ for $\tau(\vec{A}_1), \ldots, \tau(\vec{A}_{|\vec{A}|})$ and $\tau(\vec{B}) \to \mu$ for $\tau(\vec{B}_1) \to \cdots \to \tau(\vec{B}_{|\vec{B}|}) \to \mu$.

For an inductive predicate without computational content the obvious definition is

$$\tau_0(I) := *.$$

2.3 Program extraction

We start with defining the extracted term along the definition of a proof. In the next subsection we give the definition of realizability as well as a correctness proof.

The interesting case is of course that of an inductive definition, occurring in proofs via the introduction and elimination axioms. Since on the program side an inductive definition corresponds to an inductive datatype, namely the generation tree of the inductive definition, the natural realizers of the introduction axioms are the constructors of this datatype. The elimination axiom, i.e., the induction principle, is realized by recursion.

The Extracted Program

Definition. a) Given a derivation d of a computationally meaningful formula A, we inductively define the extracted program $[\![d]\!]$ of type $\tau(A)$. We assume that for every assumption variable u^A there is a unique object variable $x_u^{\tau(A)}$ assigned to it.

$$
\begin{array}{rcl}
[\![u^A]\!] & := & x_A^{\tau(A)} \\[2ex]
[\![\lambda u^A d]\!] & := & \begin{cases} [\![d]\!] & \text{if } \tau(A) = *, \\ \lambda x_u^{\tau(A)} [\![d]\!] & \text{otherwise.} \end{cases} \\[2ex]
[\![d^{A\to B} e]\!] & := & \begin{cases} [\![d]\!] & \text{if } \tau(A) = *, \\ [\![d]\!][\![e]\!] & \text{otherwise.} \end{cases} \\[2ex]
[\![\langle d_0^{A_0}, d_1^{A_1}\rangle]\!] & := & \begin{cases} [\![d_i]\!] & \text{if } \tau(A_{1-i}) = * \\ \langle [\![d_0]\!], [\![d_1]\!]\rangle & \text{otherwise.} \end{cases} \\[2ex]
[\![\pi_i(d^{A_0\wedge A_1})]\!] & := & \begin{cases} [\![d]\!] & \text{if } \tau(A_{1-i}) = * \\ \pi_i [\![d]\!] & \text{otherwise.} \end{cases} \\[2ex]
[\![(\lambda x d)^{\forall x A}]\!] & := & \lambda x [\![d]\!], \\
[\![d^{\forall x A} t]\!] & := & [\![d]\!] t. \\[2ex]
[\![(\lambda x d)^{\forall^{\mathsf{nc}} x A}]\!] & := & [\![d]\!], \ ^{2} \\
[\![d^{\forall^{\mathsf{nc}} x A} t]\!] & := & [\![d]\!].
\end{array}
$$

It remains to provide realizers for the axioms. The extracted terms for the $\exists$-axioms are

$$
[\![\exists^+_{x^\rho, A}]\!] := \begin{cases} \lambda x^\rho x & \text{if } \tau(A) = *, \\ \lambda x^\rho \lambda y^{\tau(A)} \langle x, y\rangle & \text{otherwise.} \end{cases}
$$

$$
[\![\exists^-_{x^\rho, A, B}]\!] := \begin{cases} \lambda x^\rho \lambda f^{\rho\to\tau(B)} f x & \text{if } \tau(A) = *, \\ \lambda z^{\rho\times\tau(A)} \lambda f^{\rho\to\tau(A)\to\tau(B)} f \pi_0(z) \pi_1(z) & \text{otherwise.} \end{cases}
$$

In the case of an existential quantifier without computational content these definitions simplify to

$$
[\![\exists^{\mathsf{nc}+}_{x, A}]\!] := \lambda y^{\tau(A)}.y,
$$

$$
[\![\exists^{\mathsf{nc}-}_{x^\rho, A(x), B}]\!] := \begin{cases} \lambda y^{\tau(B)}.y & \text{if } \tau(A) = *, \\ \lambda z^{\tau(A)} \lambda y^{\tau(A)\to\tau(B)}.yz & \text{otherwise.} \end{cases}
$$

[2] This is a sound definition since, by the CV-lemma, see below, and the strengthened variable condition, we know that $x \notin \mathrm{FV}([\![d]\!])$. Strictly speaking the CV-lemma has to be proven simultaneously with this definition.

Next, given an inductively defined predicate I, we need realizers for the closure axioms $K_1[I], \ldots, K_n[I]$ and the induction principle $\mathsf{Ind}_{I,P}$. Assume that we have assigned an algebra μ with constructors $c_1, \ldots, c_n$ to this predicate, i.e., that we are dealing with a predicate with computational content. Then we set

$$[\![K_i[I]]\!] := c_i$$

and the induction principle corresponds to recursion on μ, more precisely,

$$[\![\mathsf{Ind}_{I,P}]\!] := \mathsf{R}_{\mu,\tau(P)}.$$

Finally, for a formula A with $\tau(A) \neq *$, the efq axioms $F \to A$ and $\bot \to A$ are realized by a canonical inhabitant of the type $\tau(A)$. The compatibility axiom, $\forall^{\mathsf{nc}}\vec{x}, x, y.x = y \to A(x) \to A(y)$, is realized by $\lambda x^{\tau(A)}x$.

Moreover, induction and case distinction on an inductive datatype μ, $\mathsf{Ind}_{\mu,A}$ and $\mathsf{Cases}_{\mu,A}$, correspond to recursion on this datatype, $\mathsf{R}_{\mu,\tau(A)}$, case distinction, $C_{\mu,\tau(A)}$, respectively.

b) In the case $d : A$ where A has no computational content, i.e., if $\tau(A) = *$, we set $[\![d]\!] = \epsilon$ where ϵ is a new symbol.

Lemma 2.1 (CV-lemma). $\mathrm{FV}([\![d]\!]) \subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d).\}$

Proof. If $\tau(A) = *$, then we have $[\![d]\!] = \epsilon$ and $\mathrm{FV}([\![d]\!]) = \emptyset$. Therefore we assume $\tau(A) \neq *$ and proceed by induction on the structure of d. *Case:* u:

$$\mathrm{FV}([\![u]\!]) = \mathrm{FV}(x_u) = \{x_u\} = \{x_u | u \in \mathrm{FA}(u)\}.$$

Case: $\lambda u^A d^B$: $\mathrm{FV}([\![\lambda u^A d^B]\!]) =$

$$\begin{array}{lll}
= \mathrm{FV}(\lambda x_{u^A}[\![d]\!]) & & [\text{by def } [\![]\!]] \\
= \mathrm{FV}([\![d]\!])\backslash\{x_{u^A}\} & & [\text{by def FV}] \\
\subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d)\}\backslash\{x_{u^A}\} & & [\text{by ih}] \\
= \mathrm{CV}(\lambda ud) \cup \{x_u | u \in \mathrm{FA}(\lambda ud)\} & & [\mathrm{CV}(\lambda ud) = \mathrm{CV}(d), \\
 & & \mathrm{FA}(\lambda ud) = \mathrm{FA}(d)\backslash\{u\}]
\end{array}$$

Case: $d^{A\to B}e^B$: $\mathrm{FV}([\![de]\!]) =$

$$\begin{array}{lll}
= \mathrm{FV}([\![d]\!]) \cup \mathrm{FV}([\![e]\!]) & & [\text{by def } [\![]\!], \mathrm{FV}] \\
\subseteq \mathrm{CV}(d) \cup \mathrm{CV}(e) \cup \{x_u | u \in (\mathrm{FA}(d) \cup \mathrm{FA}(e))\} & & [\text{by ih}] \\
= \mathrm{CV}(de) \cup \{x_u | u \in \mathrm{FA}(de)\} & & [\text{by def CV, FA}].
\end{array}$$

Case: $(\lambda x d)^{\forall x A}$: $\mathrm{FV}([\![(\lambda x d)^{\forall x A}]\!]) =$

$$\begin{array}{ll}
= \mathrm{FV}(\lambda x [\![d]\!]) & [\text{by def } [\![]\!]] \\
= \mathrm{FV}([\![d]\!]) \backslash \{x\} & [\text{by def FV}] \\
\subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d)\} \backslash \{x\} & [\text{by ih}] \\
= \mathrm{CV}((\lambda x d)^{\forall x A}) \cup \{x_u | u \in \mathrm{FA}(\lambda x d)\} & [\mathrm{CV}((\lambda x d)^{\forall x A}) = \mathrm{CV}(d) \backslash \{x\} \\
 & \mathrm{FA}((\lambda x d)^{\forall x A}) = \mathrm{FA}(d)]
\end{array}$$

Case: $(\lambda x d)^{\forall^{\mathsf{nc}} x A}$: $\mathrm{FV}([\![(\lambda x d)^{\forall^{\mathsf{nc}} x A}]\!])$

$$\begin{array}{ll}
= \mathrm{FV}([\![d]\!]) & [\text{by def } [\![]\!]] \\
\subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d)\} & [\text{by ih}] \\
= \mathrm{CV}((\lambda x d)^{\forall^{\mathsf{nc}} x A}) \cup \{x_u | u \in \mathrm{FA}(\lambda x d)\} & [\mathrm{CV}((\lambda x d)^{\forall^{\mathsf{nc}} x A}) = \mathrm{CV}(d) \\
 & \mathrm{FA}((\lambda x d)^{\forall^{\mathsf{nc}} x A}) = \mathrm{FA}(d)]
\end{array}$$

Case: $d^{\forall x A} t$: $\mathrm{FV}([\![d^{\forall x A} t]\!]) =$

$$\begin{array}{ll}
= \mathrm{FV}([\![d]\!]) \cup \mathrm{FV}(t) & [\text{by def } [\![]\!], \mathrm{FV}] \\
\subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d)\} \cup \mathrm{FV}(t) & [\text{by ih}] \\
= \mathrm{CV}(d^{\forall x A} t) \cup \{x_u | u \in \mathrm{FA}(d^{\forall x A} t)\} & [\mathrm{CV}(d^{\forall x A} t) = \mathrm{CV}(d) \cup \mathrm{FV}(t), \\
 & \mathrm{FA}(d^{\forall x A} t) = \mathrm{FA}(d)]
\end{array}$$

Case: $d^{\forall^{\mathsf{nc}} x A} t$: $\mathrm{FV}([\![d^{\forall^{\mathsf{nc}} x A} t]\!]) =$

$$\begin{array}{ll}
= \mathrm{FV}([\![d]\!]) & [\text{by def } [\![]\!]] \\
\subseteq \mathrm{CV}(d) \cup \{x_u | u \in \mathrm{FA}(d)\} & [\text{by ih}] \\
= \mathrm{CV}(d^{\forall^{\mathsf{nc}} x A} t) \cup \{x_u | u \in \mathrm{FA}(d^{\forall^{\mathsf{nc}} x A} t)\} & [\mathrm{CV}(d^{\forall^{\mathsf{nc}} x A} t) = \mathrm{CV}(d), \\
 & \mathrm{FA}(d^{\forall^{\mathsf{nc}} x A} t) = \mathrm{FA}(d)]
\end{array}$$

All remaining cases can be proven easily by means of the induction hypothesis. □

The Correctness of the Program

Definition (Modified Realizability). For every formula A we define a formula r **mr** A where r is either a term of type $\tau(A)$ or the symbol ϵ depending on whether or not A is computationally meaningful.

We assume that for each predicate $P : \rho_1, \ldots, \rho_n$ with computational content we have enriched our language by a predicate $\tilde{P}$ of arity $\tau_0(P), \rho_1, \ldots, \rho_n$.

$$r\ \mathbf{mr}\ P(\vec{t}) \quad := \quad \begin{cases} \tilde{P}(r,\vec{t}) & \text{if } P(\vec{t}) \text{ has computational content} \\ P(\vec{t}) & \text{otherwise} \end{cases}$$

$$r\ \mathbf{mr}\ (A \to B) \quad := \quad \begin{cases} \epsilon\ \mathbf{mr}\ A \to r\ \mathbf{mr}\ B & \text{if } \tau(A) = *, \\ \forall x.x\ \mathbf{mr}\ A \to \epsilon\ \mathbf{mr}\ B & \text{if } \tau(A) \neq * = \tau(B), \\ \forall x.x\ \mathbf{mr}\ A \to rx\ \mathbf{mr}\ B & \text{otherwise.} \end{cases}$$

$$r\ \mathbf{mr}\ (A_0 \wedge A_1) \quad := \quad \begin{cases} r\ \mathbf{mr}\ A_{1-i} \wedge\ \epsilon\ \mathbf{mr}\ A_i & \text{if } \tau(A_i) = *, \\ \pi_0(r)\ \mathbf{mr}\ A_0 \wedge \pi_1(r)\ \mathbf{mr}\ A_1 & \text{otherwise.} \end{cases}$$

$$r\ \mathbf{mr}\ \forall x A \quad := \quad \begin{cases} \forall x.\epsilon\ \mathbf{mr}\ A & \text{if } \tau(A) = *, \\ \forall x.rx\ \mathbf{mr}\ A, & \text{otherwise.} \end{cases}$$

$$r\ \mathbf{mr}\ \exists x\, A \quad := \quad \begin{cases} \epsilon\ \mathbf{mr}\ A[x/r] & \text{if } \tau(A) = *, \\ \pi_1(r)\ \mathbf{mr}\ A[x/\pi_0(r)] & \text{otherwise.} \end{cases}$$

In the case of quantifiers without computational content we set

$$\begin{array}{lcl} r\ \mathbf{mr}\ \forall^{\mathsf{nc}} x\, A & := & \forall x.\ r\ \mathbf{mr}\ A \\ r\ \mathbf{mr}\ \exists^{\mathsf{nc}} x\, A & := & \exists x.\ r\ \mathbf{mr}\ A. \end{array}$$

Finally, we communicate how the predicate $\tilde{I}$ for an inductive predicate I with computational content is to be introduced. If I is given by n closure axioms of the form

$$K[I] = \forall \vec{x}, \forall^{\mathsf{nc}} \vec{x}'. \overrightarrow{A} \to \overrightarrow{\forall \vec{y}, \forall^{\mathsf{nc}} \vec{y}'.\ \overrightarrow{B} \to I(\vec{s})} \to I(\vec{t}),$$

then $\tilde{I}$ has to be inductively defined by n analogous axioms $\tilde{K}[\tilde{I}]$ of the form

$$\begin{array}{ll} \forall \vec{x}, \forall^{\mathsf{nc}} \vec{x}', \forall \vec{u}. & \vec{u}\ \mathbf{mr}\ \overrightarrow{A} \to \\ & (\forall f_1, \forall \vec{y_1}, \forall^{\mathsf{nc}} \vec{y_1}', \forall \vec{v_1}.\ \vec{v_1}\ \mathbf{mr}\ \overrightarrow{B_1} \to \tilde{I}(f_1 \vec{y_1} \vec{v_1}, \vec{s_1})) \to \\ & \vdots \\ & (\forall f_m, \forall \vec{y_m}, \forall^{\mathsf{nc}} \vec{y_m}', \forall \vec{v_m}.\ \vec{v_m}\ \mathbf{mr}\ \overrightarrow{B_m} \to \tilde{I}(f_m \vec{y_m} \vec{v_m}, \vec{s_m})) \to \\ & \tilde{I}(c\vec{x}\vec{u}\vec{f}, \vec{t}) \end{array}$$

Remark 2.2. 1. Looking at the closure axioms of $\tilde{I}$ we see that only invariant formulas are involved, so we may declare $\tilde{I}$ to be a predicate without computational content.

2. In the cases $r\ \mathbf{mr}\ \forall^{\mathsf{nc}} xA$ and $r\ \mathbf{mr}\ \exists^{\mathsf{nc}} xA$, x is supposed not to be free in r. Note also that in $r\ \mathbf{mr}\ \forall^{\mathsf{nc}} xA$ the realizer r might be ϵ, depending on whether or not $\tau(A) = *$.

Lemma 2.1. *a)* $(r\ \mathbf{mr}\ A)[x/t] = r[x/t]\ \mathbf{mr}\ A[x/t]$.

b) $\epsilon\ \mathbf{mr}\ A = A$ *if* A *is invariant.*

Proof. Straightforward. □

Proposition 2.3 (Soundness Theorem). If d is a proof of a formula A, then we can derive $[\![d]\!]$ **mr** A from the assumptions $\{\bar{u}\colon x_u \textbf{ mr } C \mid u^C \in \mathrm{FA}(d)\}$, where $x_u := \epsilon$ if u^C is an assumption variable for a formula C without computational content.

Proof. By induction on the structure of d. We concentrate on the axioms concerning inductive definitions and the logical axioms for the quantifiers carrying no computational content.

Case: Closure axiom of an inductive predicate I. Sub-case: I has computational content. Let

$$K[I] = \forall \vec{x}, \forall^{\mathsf{nc}} x'.\, \overrightarrow{A} \to \overrightarrow{\forall \vec{y}, \forall^{\mathsf{nc}} \vec{y}'.\, \overrightarrow{B_j} \to I(\vec{s_j})} \to I(\vec{t})$$

be the i-th closure axiom of an inductively defined predicate, and let c be the i-th constructor of the associated algebra μ. Then we have to show

$$c \textbf{ mr } K[I].$$

For simplicity, we assume that all formulas, $\overrightarrow{A}$, $\overrightarrow{B}_j$, have computational content; the other cases are analogous, but simpler. By the definition of **mr**, c **mr** $K[I]$ expands to

$$\begin{array}{lll}
\forall \vec{x}, \forall^{\mathsf{nc}} \vec{x}', & \forall \vec{u}. & \vec{u} \textbf{ mr } \overrightarrow{A} \to \\
& \forall f_1. & f_1 \textbf{ mr } (\forall \vec{y_1}, \forall^{\mathsf{nc}} \vec{y_1}'.\overrightarrow{B_1} \to I(\vec{s_1})) \to \\
& & \vdots \\
& \forall f_m. & f_m \textbf{ mr } (\forall \vec{y_m}, \forall^{\mathsf{nc}} y_m'.\overrightarrow{B_m} \to I(\vec{s_m})) \to \\
& & c\vec{x}\vec{u}\vec{f} \textbf{ mr } I(\vec{t})
\end{array}$$

which is equivalent to

$$\begin{array}{lll}
\forall \vec{x}, \forall^{\mathsf{nc}} \vec{x}', & \forall \vec{u}. & \vec{u} \textbf{ mr } \overrightarrow{A} \to \\
& \forall f_1. & (\forall \vec{y_1}, \forall^{\mathsf{nc}} \vec{y_1}', \forall \vec{v_1}.\vec{v_1} \textbf{ mr } \overrightarrow{B_1} \to f_1\vec{y_1}\vec{v_1} \textbf{ mr } I(\vec{s_1})) \to \\
& & \vdots \\
& \forall f_m. & (\forall \vec{y_m}, \forall^{\mathsf{nc}} \vec{y_m}', \forall \vec{v_m}.\vec{v_m} \textbf{ mr } \overrightarrow{B_m} \to f_m\vec{y_m}\vec{v_m} \textbf{ mr } I(\vec{s_m})) \to \\
& & c\vec{x}\vec{u}\vec{f} \textbf{ mr } I(\vec{t})
\end{array}$$

But this is the i-th closure axiom for $\tilde{I}$.

Sub-case: I is a predicate without computational content. We have to show $[\![K[I]]\!]$ **mr** $K[I]$, that is, by definition of $[\![]\!]$, ϵ **mr** $K[I]$. Since by assumption $K[I]$ is supposed to be invariant, we know by Lemma 2.1 that ϵ **mr** $K[I] = K[I]$, so we can reduce our goal to $K[I]$ which is an axiom.

Case: $\mathsf{Ind}_{I,P}$. Sub-case: I is an inductive predicate with computational content. The aim is to show

$$\mathsf{R}_{\mu,\tau(P)}\ \mathbf{mr}\ K_1[I,P] \to \ldots \to K_n[I,P] \to \forall^{\mathsf{nc}}\vec{z}.\, I(\vec{z}) \to P(\vec{z}).$$

Here, we prove the case in which in the closure axioms the type τ of all formulas is different from $*$; the other cases are analogous. To ease readability, we subdivide the proof in numbered steps.

1. Assume that we have $\vec{w}, \vec{z}$, and w such that

$$\begin{array}{rl} w_i\ \mathbf{mr}\ K_i[I,P] & \text{for } 1 \le i \le n \\ w\ \mathbf{mr}\ I(\vec{z}) & \text{i.e., } \tilde{I}(w,\vec{z}) \end{array}$$

and show

$$\mathsf{R}_{\mu,\tau(P)} w_1 \ldots w_n\, w\ \ \mathbf{mr}\ P(\vec{z}) =: \tilde{P}(w,\vec{z}).$$

2. Using the (strengthened) induction principle for $\tilde{I}$

$$\mathsf{Ind}_{\tilde{I},\tilde{P}} := \tilde{K}_1[\tilde{I},\tilde{P}] \to \ldots \to \tilde{K}_n[\tilde{I},\tilde{P}] \to \forall^{\mathsf{nc}} w,\vec{z}.\, \tilde{I}(w,\vec{z}) \to \tilde{P}(w,\vec{z})$$

it suffices to show $\tilde{K}_i[\tilde{I},\tilde{P}]$ for each i, (i.e., again suppressing the index i):

$$\begin{array}{rl} \forall\vec{x}, \forall^{\mathsf{nc}}\vec{x}', \forall\vec{u},\vec{f}. & \vec{u}\ \mathbf{mr}\ \overrightarrow{A} \to \\ & (\forall\vec{y_1}, \forall^{\mathsf{nc}}\vec{y_1}', \forall\vec{v_1}.\, \vec{v_1}\ \mathbf{mr}\ \overrightarrow{B_1} \to \tilde{I}(f_1\vec{y_1}\vec{v_1}, \vec{s_1})) \to \\ & \vdots \\ & (\forall\vec{y_m}, \forall^{\mathsf{nc}}\vec{y_m}', \forall\vec{v_m}.\, \vec{v_m}\ \mathbf{mr}\ \overrightarrow{B_m} \to \tilde{I}(f_m\vec{y_m}\vec{v_m}, \vec{s_m})) \to \\ & (\forall\vec{y_1}, \forall^{\mathsf{nc}}\vec{y_1}, \forall\vec{v_1}.\, \vec{v_1}\ \mathbf{mr}\ \overrightarrow{B_1} \to \tilde{P}(f_1\vec{y_1}\vec{v_1}, \vec{s_1})) \to \\ & \vdots \\ & (\forall\vec{y_m}, \forall^{\mathsf{nc}}\vec{y_m}', \forall\vec{v_m}.\, \vec{v_m}\ \mathbf{mr}\ \overrightarrow{B_m} \to \tilde{P}(f_m\vec{y_m}\vec{v_m}, \vec{s_m})) \to \\ & \tilde{P}(c\vec{x}\vec{u}\vec{f}, \vec{t}) \end{array}$$

3. Fix $\vec{x}, \vec{x}', \vec{u}$ and $\vec{f}$ such that

   (a) $\vec{u}\ \mathbf{mr}\ \overrightarrow{A}$

   (b) $\forall\vec{y_j}, \forall^{\mathsf{nc}}\vec{y_j}', \forall\vec{v_j}.\, \vec{v_j}\ \mathbf{mr}\ \overrightarrow{B_j} \to \tilde{I}(f_j\vec{y_j}\vec{v_j}, \vec{s_j})$ for all j, $1 \le j \le m$.

   (c) $\forall\vec{y_j}, \forall^{\mathsf{nc}}\vec{y_j}', \forall\vec{v_j}.\, \vec{v_j}\ \mathbf{mr}\ \overrightarrow{B_j} \to \tilde{P}(f_j\vec{y_j}\vec{v_j}, \vec{s_j})$ for all j, $1 \le j \le m$.

and show $\tilde{P}(c\vec{x}\vec{v}\vec{f}, \vec{t})$, i.e.,

$$\mathsf{R}_{\mu,\tau(P)}\vec{w}(c\vec{x}\vec{v}\vec{f})\ \mathbf{mr}\ P(\vec{t}).$$

4. Applying the conversion rule for the recursion operator we obtain

$$\begin{array}{ll} w_i\vec{x}\vec{v}\vec{f} & (\lambda\vec{y_1}, \vec{v_1}^{\tau(\overrightarrow{B_1})}.\, \mathsf{R}_{\mu,\tau(P)}\vec{w}(f_1\vec{y_1}\vec{v_1})) \\ & \vdots \\ & (\lambda\vec{y_m}, \vec{v_m}^{\tau(\overrightarrow{B_m})}.\, \mathsf{R}_{\mu,\tau(P)}\vec{w}(f_m\vec{y_m}\vec{v_m}))\ \mathbf{mr}\ P(\vec{t}). \end{array}$$

5. This can be proven by using the assumption w_i **mr** $K_i[I,P]$ (i.e., suppressing the index i almost everywhere but not in w_i),

$$\begin{array}{lll} \forall\vec{x},\forall\vec{x}', & \forall\vec{v}. & \vec{v}\ \mathbf{mr}\ \overrightarrow{A} \to \\ & \forall\vec{f}. & \vec{f}\ \mathbf{mr}\ \overrightarrow{\forall\vec{y_j},\forall^{\mathsf{nc}}\vec{y_j}'.\, \overrightarrow{B_j} \to I(\vec{s_j})} \to \\ & \forall\vec{g}. & \vec{g}\ \mathbf{mr}\ \overrightarrow{\forall\vec{y_j},\forall^{\mathsf{nc}}\vec{y_j}'.\, \overrightarrow{B_j} \to P(\vec{s_j})} \to w_i\vec{x}\vec{u}\vec{f}\vec{g}\ \mathbf{mr}\ P(\vec{t}) \end{array}$$

with $\vec{x}, \vec{x}', \vec{v}, \vec{f}$ and

$$\lambda\vec{v_1}^{\tau(\overrightarrow{B_1})}\, \mathsf{R}_{\mu,\tau(P)}\vec{w}(f_1\vec{y_1}\vec{v_1}),\ \ldots,\ \lambda\vec{v_m}^{\tau(\overrightarrow{B_m})}\, \mathsf{R}_{\mu,\tau(P)}\vec{w}(f_m\vec{y_m}\vec{v_m}).$$

6. It remains to show

(a') $\vec{v}\ \mathbf{mr}\ \overrightarrow{A}$,

(b') $\lambda\vec{y_j}, \vec{v_j}^{\tau(\overrightarrow{B_j})}\, f_j\vec{y_j}\vec{v_j}\ \mathbf{mr}\ \forall\vec{y_j},\forall^{\mathsf{nc}}\vec{y_j}.\, \overrightarrow{B_j} \to I(\vec{s_j})$,

(c') $\lambda\vec{y_j}, \vec{v_j}^{\tau(\overrightarrow{B_j})}\, \mathsf{R}_{\mu,\tau(P)}w_1\ldots w_n(f_j\vec{y_j}\vec{v_j})\ \mathbf{mr}\ \forall\vec{y_j},\forall^{\mathsf{nc}}\vec{y_j}.\, \overrightarrow{B_j} \to P(\vec{s_j})$,

for all j, $1 \leq j \leq m$, which hold by (a), (b) and (c).

Sub-case: I is an inductive predicate without computational content.

We have to show

$$[\![\mathsf{Ind}_{I,P}]\!]\ \mathbf{mr}\ \mathsf{Ind}_{I,P},$$

i.e., by definition of $[\![\,]\!]$,

$$\epsilon\ \mathbf{mr}\ K_1[I,P] \to \ldots \to K_n[I,P] \to \forall^{\mathsf{nc}}\vec{z}.I(\vec{z}) \to P(\vec{z})$$

and by definition of **mr**

$$\epsilon\ \mathbf{mr}\ K_1[I,P] \to \ldots \to \epsilon\ \mathbf{mr}\ K_n[I,P] \to \forall\vec{z}.\epsilon\ \mathbf{mr}\ I(\vec{z}) \to \epsilon\ \mathbf{mr}\ P(\vec{z}).$$

Now, we distinguish two cases: if P is a predicate without computational content, we know that $K_i[I,P]$ is invariant. As

$$K_i[I,P] = \forall\vec{x}.\overrightarrow{A} \to \overrightarrow{\forall\vec{y}.\, \overrightarrow{B} \to I(\vec{s})} \to \overrightarrow{\forall\vec{y}.\, \overrightarrow{B} \to P(\vec{s})} \to P(\vec{t})$$

we have $\epsilon\ \mathbf{mr}\ K_i[I,P] = K_i[I,P], \epsilon\ \mathbf{mr}\ I(\vec{z}) = I(\vec{z})$, and $\epsilon\ \mathbf{mr}\ P(\vec{z}) = P(\vec{z})$ and the goal can be reduced to

$$K_1[I,P] \to \ldots \to K_n[I,P] \to \forall^{\mathsf{nc}}\vec{z}.I(\vec{z}) \to P(\vec{z}),$$

i.e., to the axiom itself.

Otherwise, if P has computational content, we set $Q(\vec{x}) := \epsilon\ \mathbf{mr}\ P(\vec{x})$ and our goal can be proven using

$$K_1[I,Q] \to \ldots \to K_n[I,Q] \to \forall^{\mathsf{nc}}\vec{z}.I(\vec{z}) \to Q(\vec{z}).$$

Case: $(\forall^{\mathsf{nc}+})$. We have to show

$$[\![(\lambda x d)^{\forall^{\mathsf{nc}} x A}]\!]\ \mathbf{mr}\ \forall^{\mathsf{nc}} x A,$$

i.e., by the definition of $[\![]\!]$ and **mr**,

$$\forall x.\, [\![d^A]\!]\ \mathbf{mr}\ A.$$

Note that by means of the CV-lemma and the strengthened variable condition, we know that $x \notin \mathrm{FV}([\![d^A]\!])$. Now, we are almost done, since by induction hypothesis we have a proof of $[\![d]\!]\ \mathbf{mr}\ A$.

Case: $(\forall^{\mathsf{nc}-})$. We show $[\![d^{\forall^{\mathsf{nc}} x A} t]\!]\ \mathbf{mr}\ A[x/t]$, that is, by the definition of $[\![]\!]$, $[\![d^{\forall^{\mathsf{nc}} x A}]\!]\ \mathbf{mr}\ A[x/t]$. By ih, let $\mu(d)$ be a proof of

$$[\![d^{\forall^{\mathsf{nc}} x A}]\!]\ \mathbf{mr}\ \forall x A,$$

i.e., $\forall x'.[\![d^{\forall^{\mathsf{nc}} x A}]\!]\ \mathbf{mr}\ A[x/x']$. An instantiation with t yields $[\![d^{\forall x A}]\!]\ \mathbf{mr}\ A[x/t]$.

Case: $(\exists^{\mathsf{nc}+}_{\vec{x},x,A})$. Sub-case $\tau(A) \neq *$. We need to show

$$\lambda y^{\tau(A)}.y\ \mathbf{mr}\ \forall^{\mathsf{nc}}\vec{x},x.A \to \exists^{\mathsf{nc}} x A,$$

where $\vec{x}$ denotes the free parameters. By the definition of **mr** it suffices to show

$$\forall \vec{x}, x, y.y\ \mathbf{mr}\ A \to y\ \mathbf{mr}\ \exists^{\mathsf{nc}} x A,$$

i.e.,

$$\forall \vec{x}, y, x.y\ \mathbf{mr}\ A \to \exists x.y\ \mathbf{mr}\ A$$

which is the axiom $\exists^{+}_{\vec{x}',x,\ y\ \mathbf{mr}\ A}$ where the free parameters $\vec{x}'$ comprise $\vec{x}$ and y.

Sub-case: $\tau(A) \neq *$. Show

$$\epsilon\ \mathbf{mr}\ \forall^{\mathsf{nc}}\vec{x},x.A \to \exists^{\mathsf{nc}} x A.$$

By definition of **mr** it suffices to show

$$\forall \vec{x}, x.\epsilon\ \mathbf{mr}\ A \to \epsilon\ \mathbf{mr}\ \exists^{\mathsf{nc}} x A,$$

that is,

$$\forall \vec{x}, x.\epsilon\ \mathbf{mr}\ A \to \exists x.\epsilon\ \mathbf{mr}\ A$$

which again is an axiom.

Case: $(\exists^{\mathsf{nc}-}_{\vec{x},x,A,B})$. We start with the case $\tau(A) \neq * \neq \tau(B)$ and show that it can be reduced to the axiom $\exists^{-}_{\vec{x},z,y,x,z\ \mathbf{mr}\ A,yz\ \mathbf{mr}\ B}$.

$$\begin{aligned}
& \lambda z, y.yz\ \mathbf{mr}\ \forall^{\mathsf{nc}}\vec{x}.\exists^{\mathsf{nc}}xA \to \forall^{\mathsf{nc}}x(A \to B) \to B, \\
\Leftrightarrow\ & \forall \vec{x}, z, y.z\ \mathbf{mr}\ \exists^{\mathsf{nc}}xA \to y\ \mathbf{mr}\ \forall^{\mathsf{nc}}x(A \to B) \to yz\ \mathbf{mr}\ B \\
\Leftrightarrow\ & \forall \vec{x}, z, y.(\exists x.z\ \mathbf{mr}\ A) \to (\forall x, z'.z'\ \mathbf{mr}\ A \to yz'\ \mathbf{mr}\ B) \to yz\ \mathbf{mr}\ B \\
\Rightarrow\ & \forall \vec{x}, z, y.(\exists x.z\ \mathbf{mr}\ A) \to (\forall x.z\ \mathbf{mr}\ A \to yz\ \mathbf{mr}\ B) \to yz\ \mathbf{mr}\ B.
\end{aligned}$$

In the case $\tau(A) = *$ we reduce the goal to the axiom $\exists^{-}_{\vec{x},z,x,\epsilon\ \mathbf{mr}\ A,y\ \mathbf{mr}\ B}$.

$$\begin{aligned}
& \lambda y.y\ \mathbf{mr}\ \forall^{\mathsf{nc}}\vec{x}.\exists^{\mathsf{nc}}xA \to \forall^{\mathsf{nc}}x(A \to B) \to B, \\
\Leftrightarrow\ & \forall \vec{x}, y.\epsilon\ \mathbf{mr}\ \exists^{\mathsf{nc}}xA \to y\ \mathbf{mr}\ \forall^{\mathsf{nc}}x(A \to B) \to y\ \mathbf{mr}\ B, \\
\Leftrightarrow\ & \forall \vec{x}, y.(\exists x.\epsilon\ \mathbf{mr}\ A) \to (\forall x.\epsilon\ \mathbf{mr}\ A \to y\ \mathbf{mr}\ B) \to y\ \mathbf{mr}\ B.
\end{aligned}$$

For a proof of the remaining cases we refer to the standard literature, see for example, [TvD88]. As a presentation also treating the axiom cases, we recommend [Sch03]. □

Remark 2.4. We presented a version of realizability for inductive definitions allowing the use of both quantifiers, $\forall$ and $\forall^{\mathsf{nc}}$. It might be the case that one would like to swap from one quantifier to the other. To this extend, we suggest an axiom (or lemma), here formulated for the example of the natural numbers,

$$\forall nA \leftrightarrow (\forall^{\mathsf{nc}}n.N(n) \to A)$$

where $N(n)$ is the inductive predicate saying that n is a natural number, defined by the closure axioms $N(0)$ and $\forall n.N(n) \to N(n+1)$. The axiom has a natural realizer, the identity, thus we may use it and extract programs in the usual way.

One might object that, to avoid redundancies, we could have restricted ourselves to the $\forall^{\mathsf{nc}}$ quantifier, and viewed the other quantifiers just as abbreviations in the sense of the axiom. We prefer to allow both quantifiers, first, for reasons of convenience, and, secondly, to be as general as possible.

3 Computational Content of Classical Proofs

This chapter is about program extraction from classical proofs. Later, we want to make use of the classical proof of Higman's Lemma due to Nash-Williams and extract a program from this proof. To this end, we shall first explain the method, often called A-translation, of transforming a classical proof into a constructive one. Subsequently, we will show how to deal with the axiom of dependent choice (DC) used in Nash-Williams' proof. In doing so, we shall follow [BO03]. We recall the result of [BO03] and place it in the context of [BBS02], which is an extension of the A-translation mechanism, now built-in in the MINLOG system. Our exposition is slightly more general than [BBS02] since we are working in HA^μ which has a more general type system.

3.1 Refined A-translation

The formal system

We briefly fix the formal framework and then sketch the idea behind the A-translation method, where we in particular stress the role of the so-called definite and goal formulas. The main statement of this section is concisely formulated in Proposition 3.3.

In this chapter we work in Heyting Arithmetic HA^μ. Let X be a distinguished nullary predicate symbol with assigned type $\tau_0(X) := \nu$ where ν is some unspecified but fixed type. Then by the definition of realizability we have

$$r \ \mathbf{mr}\ X = \tilde{X}(r)$$

where $\tilde{X}$ is a new predicate symbol of arity ν. In addition, we denote by D^X the formula D where $\bot$ is substituted by X. If $\vec{D} = D_1, \ldots, D_n$, then $\vec{D}^X$ stands for $D_1^X, \ldots, D_n^X$.

HA^μ may be enriched by additional axioms, Δ; whereby each new axiom $C \in \Delta$ should fulfil the requirement $C[\bot/A] \in \Delta$ for an arbitrary formula A.

Finally, by $\mathsf{HA}^\mu \vdash_m$ we mean derivability in HA^μ without the use of $\mathsf{efq}_A : \bot \to A$ for any A. (Note that the axiom $F \to A$ is still allowed.) Conversely, we use the notion $\vdash_i$ to stress that all efq axioms are allowed.

The idea of the refined translation

Assume that we have a proof

$$\mathsf{HA}^\mu \vdash_m \vec{D} \to (\forall \vec{y}.\vec{G} \to \bot) \to \bot \quad (*)$$

where D and G are arbitrary formulas. Then it also holds

$$\mathsf{HA}^\mu \vdash_m \vec{D}^X \to (\forall \vec{y}.\vec{G}^X \to X) \to X$$

since we are in minimal logic. If we now know that

(1) $\mathsf{HA}^\mu \vdash D_i \to D_i^X$.

(2) $\mathsf{HA}^\mu \vdash \forall \vec{y}.\overrightarrow{G}^X \to X$,

then by putting $X := \exists \vec{y}.G_1 \wedge \ldots \wedge G_m$ we obtain a proof of

$$\mathsf{HA}^\mu \vdash \overrightarrow{D} \to \exists y.G_1 \wedge \ldots \wedge G_m.$$

Up to now we have explained the so-called Friedman-trick (the replacement of $\bot$ by the existence formula). It remains the question of which formulas we may use in $(*)$ such that (1) and (2) hold. This is the part where usually double negation comes into play.

Let us start with a proof of

$$\forall \vec{x}_1 C_1 \to \ldots \to \forall \vec{x}_n C_n \to \exists^{\mathsf{cl}} y.B \quad (**)$$

where C_i and B are decidable (see below for a definition) and $\exists^{\mathsf{cl}}$ is used as abbreviation for $\neg\forall\neg$. Then, by Gödel's negative translation,

$$\forall \vec{x}_1 C_1^{\neg\neg} \to \ldots \to \forall \vec{x}_n C_n^{\neg\neg} \to \exists^{\mathsf{cl}} y.B^{\neg\neg}$$

is provable in minimal logic and it can be easily shown that (1) and (2) hold for $D_i \equiv C_i^{\neg\neg}$ and $G \equiv B^{\neg\neg}$.

The problem here is that double negating all (atomic and existential) subformulas is, as we will see later, very expensive with regard to the extracted programs. A refined way, therefore, was proposed in [BS95b] where only subformulas with so-called 'critical' relation symbols are double negated.

In [BBS02], a different way is chosen in order to leave the situation as general as possible. We define classes of formulas, i.e. definite formulas D and goal formulas G and show that (1) and (2) hold for these classes. The user is free to apply the method of [BS95b] or, if the problem is more complex, to find his own way to obtain the required situation of definite and goal formulas.

Definite and Goal formulas

We start with the definition of definite and goal formulas (introduced in [BBS02] and slightly extended here) which is based on the definition of (ir)relevant formulas and decidable formulas.

Definition (Decidable formulas). A formula is called decidable if it is built from atoms $\mathsf{atom}\,(t)$ using propositional connectives and boolean quantifiers only.

Lemma 3.1. *(Case distinction on decidable formulas). For decidable formulas D we have*

$$\vdash_i (D \to A) \to (\neg D \to A) \to A.$$

Proof. By induction on D. □

Definition (Relevant and irrelevant formulas). (Ir)relevant formulas are defined inductively by the clauses

- $\bot$ is relevant,
- if C is (ir)relevant and B is arbitrary, then $B \to C$ is (ir)relevant,
- if, C_0 and C_1 are (ir)relevant, then $C_0 \wedge C_1$ is (ir)relevant.
- if C is (ir)relevant, then $\forall x C$ is (ir)relevant.

Remark 3.1. We have extended the definition of (ir)relevant formulas with respect to the connective $\wedge$. For a formula without any conjunction it still holds that it is irrelevant if and only if it is not relevant. The definition of definite and goal formulas could be generalized in the same way. However, since, later, we require a formula to be either relevant or irrelevant, this would involve either an extra treatment (in the proofs) for formulas which are none of both or some further double negations for these formulas in order to make them relevant. There is a more important generalization in the definition of goal formulas where the requirement 'quantifier free' has been replaced by 'decidable' in order to allow inductive predicates.

Definition (Definite and goal formulas). We inductively define definite formulas D and goal formulas G. Here, P is a predicate without computational content.

$$\begin{array}{rcll} D & := & P(\vec{t}) \mid G \to D & \text{provided } D \text{ irrelevant} \Rightarrow G \text{ irrelevant} \\ & & \mid \forall x D. & \\ G & := & P(\vec{t}) \mid D \to G & \text{provided } D \text{ irrelevant} \Rightarrow D \text{ decidable} \\ & & \mid \forall x G & \text{provided } G \text{ irrelevant,} \end{array}$$

Lemma 3.2. *For definite formulas D and goal formulas G we have*

(1) $\mathsf{HA}^\mu \vdash (\neg D \to X) \to D^X$ *for D relevant,*
(2) $\mathsf{HA}^\mu \vdash D \to D^X$,
(3) $\mathsf{HA}^\mu \vdash G^X \to G$ *for G irrelevant,*
(4) $\mathsf{HA}^\mu \vdash G^X \to (G \to X) \to X$.

Proof. Analogous to [BBS02], Lemma 3.1. The condition 'D irrelevant $\Rightarrow$ D decidable' is used to prove the statement (4) and it is necessary in order to apply case distinction (Lemma 3.1) which is formulated for decidable formulas. □

Lemma 3.3. *For goal formulas $\overrightarrow{G} = G_1, \ldots, G_n$ we have*

$$\mathsf{HA}^\mu \vdash (\overrightarrow{G} \to X) \to \overrightarrow{G}^X \to X.$$

Proof. By Lemma 3.2(4) we have proofs s_i for ${G_i}^X \to (G_i \to X) \to X, 1 \le i \le m$. Then the assertion follows by

$$\dfrac{\dfrac{G_1^X \to (G_1 \to X) \to X \qquad G_1^X \qquad \dfrac{\dfrac{\dfrac{\dfrac{\overrightarrow{G} \to X \quad G_1 \ \ldots \ G_m}{X}}{G_m \to X}}{X}}{\vdots}}{G_1 \to X}}{X}}{(\overrightarrow{G} \to X) \to \overrightarrow{G^X} \to X}$$

□

Remark 3.2. 1. By looking at the preceeding display of the proof we easily see that, if $s_1, \ldots, s_m$ are the realizers of $(G_j \to X) \to G_j^X \to X$, $1 \le j \le m$, given by Lemma 3.2(4), then

$$\lambda u^\nu, \lambda \vec{w}^{\tau(\overrightarrow{G}^X)}.\ s_1 w_1(\ldots(s_m w_m u))$$

is a realizer for $(\overrightarrow{G} \to X) \to \overrightarrow{G}^X \to X$, in case $\tau(X) = \nu$ and G_j^X is computational meaningful for all j. (Otherwise the terms w_j having an index j such that $G_j^X = *$ are simply missed out.)

2. For definite $\overrightarrow{D}$, Lemma 3.2 and Lemma 3.3 immediately allow for a transformation of a proof $\mathsf{HA}^\mu \vdash_m \overrightarrow{D}^X \to (\forall \vec{y}.\overrightarrow{G}^X \to X) \to X$ into an intuitionistic proof

$$\mathsf{HA}^\mu \vdash \overrightarrow{D} \to \exists y.G_1 \wedge \ldots \wedge G_m.$$

Program extraction from classical proofs

We now recall, in a slightly streamlined form, the main result of [BBS02].

Proposition 3.3. Let Δ be an axiom system,

$D_1, \ldots, D_k$ arbitrary formulas,

$D_{k+1}, \ldots, D_n$ definite formulas,

$G_1, \ldots, G_m$ goal formulas and

$\vec{y} = y_1^{\nu_1}, \ldots, y_l^{\nu_l}$.

Moreover, assume that we have terms $t_1, \ldots, t_k$ such that

$$\mathsf{HA}^\mu + \Delta \vdash t_i \ \mathbf{mr}\ D_i^X, \quad 1 \le i \le k,$$

and

$$\mathsf{HA}^\mu \vdash_m d : \overrightarrow{D} \to (\forall \vec{y}.\, \overrightarrow{G} \to \bot) \to \bot.$$

Then, there are terms $\vec{r}$ such that

$$\mathsf{HA}^\mu + \Delta \vdash D_{k+1}, \ldots, D_n \to (G_1 \wedge \ldots \wedge G_m)[\vec{y}/\vec{r}]$$

Proof. 1. By Lemma 3.2 (2) we have HA^μ-proofs of $D_i \to D_i^X$ for all i such that $k+1 \le i \le n$, hence, by the soundness theorem, there are terms $t_{k+1}, \ldots, t_n$ such that $t_i\ \mathbf{mr}\ (D_i \to D_i^X)$, that is, since definite formulas are invariant,

$$\mathsf{HA}^\mu \vdash D_i \to t_i \ \mathbf{mr}\ D_i^X \text{ for all } k+1 \le i \le n.$$

2. Following Lemma 3.3 we have a realizer s' such that

$$\mathsf{HA}^\mu \vdash s' \ \mathbf{mr}\ ((\overrightarrow{G} \to X) \to \overrightarrow{G}^X \to X)$$

and therefore

$$\mathsf{HA}^\mu \vdash (x \ \mathbf{mr}\ \forall \vec{y}.\overrightarrow{G} \to X) \ \to \ s(x) \ \mathbf{mr}\ \forall \vec{y}.\, \overrightarrow{G}^X \to X$$

where x is a new variable and $s(x) := \lambda \vec{y}.\, s'(x\vec{y})$.

3. If we substitute X for $\bot$ in our given derivation d we obtain

$$\mathsf{HA}^\mu \vdash_m d^X : \overrightarrow{D}^X \to (\forall \vec{y}.\, \overrightarrow{G}^X \to X) \to X$$

which implies, by soundness,

$$\mathsf{HA}^\mu \vdash_m [\![d^X]\!] \ \mathbf{mr}\ (\overrightarrow{D}^X \to (\forall \vec{y}.\, \overrightarrow{G}^X \to X) \to X),$$

i.e.,

$$\mathsf{HA}^\mu \vdash_m \ \forall \vec{u}.\, \vec{u} \ \mathbf{mr}\ \overrightarrow{D}^X \to \forall v.\, v \ \mathbf{mr}\ (\forall \vec{y}.\, \overrightarrow{G}^X \to X) \to [\![d^X]\!]\vec{u}v \ \mathbf{mr}\ X.$$

Instantiating $\vec{u}$ and v with $\vec{t}$ and $s(x)$, we obtain

$$\mathsf{HA}^\mu + \Delta \vdash D_{k+1}, \ldots, D_n \to (x \ \mathbf{mr}\ \forall \vec{y}.\overrightarrow{G} \to X) \ \to \ [\![d^X]\!]\vec{t}s(x) \ \mathbf{mr}\ X.$$

4. Now, set $\nu := \nu_1 \times \cdots \times \nu_l$ and

$$X := \exists z^\nu G(z)$$

where $G(z) := (G_1 \wedge \ldots \wedge G_m)[\vec{y}/(z)_1, \ldots, (z)_l]$ and $(z)_i$ is the i-th projection of z. Then, $z\ \mathbf{mr}\ X = G(z)$. Setting $x := \lambda \vec{y}.\, \langle y_1, \ldots, y_l \rangle$ we have $x\ \mathbf{mr}\ \forall \vec{y}.\overrightarrow{G} \to X$ and therefore, by 3.,

$$\mathsf{HA}^\mu + \Delta \vdash D_{k+1}, \ldots, D_n \to [\![d^X]\!]\vec{t}s(x) \ \mathbf{mr}\ X$$

Finally by putting $r_i := ([\![d^X]\!]\vec{t}s(x))_i$ we end up with

$$\mathsf{HA}^\mu + \Delta \vdash D_{k+1}, \ldots, D_n \to (G_1 \wedge \ldots \wedge G_m)[\vec{y}/\vec{r}].$$

□

3.2 Computational content of classical dependent choice

In this section we give a realizer for (the negative translation of) the axiom of classical dependent choice

$$(\mathrm{DC}) : \forall n^{\mathsf{nat}} \forall x^{\rho} \exists^{\mathsf{cl}} y^{\rho} A_n(x,y) \to \exists f^{\mathsf{nat}\to\rho}.\ f(0) = x_0 \wedge \forall n A_n(f(n), f(n+1))$$

where x_0 is some fixed term of type ρ.

The problem with the A-translation method and the axiom of classical dependent choice is that the latter is neither a definite formula, nor can be transformed into one by the usual double negation applications. Furthermore, A-translation, applied to a proof using DC would yield a proof depending on DC^X. This would not be problematic if DC^X were an instance of the axiom of dependent choice. (Note this is usually the case with respect to the other axioms, e.g., induction.) But unfortunately, this is not the case, and DC^X can also not be derived from DC. The proposed solution to this problem is to directly give computational content to DC^X which, finally, allows us to extract programs from classical proofs using DC.

Notation

Unlike the rest of the thesis, in this chapter the variables s and t are allowed to denote finite sequences. Furthermore, we use

$[x_0, \ldots, x_{n-1}]$	for the finite list with elements $x_0, \ldots, x_{n-1}$
$\lvert t \rvert$	for the length of t
$(t)_k$	for the k-th element of t
$s\#t$	for the concatenation of s and t
$s\#\alpha$	for the concatenation of s and the infinite sequence α
$\bar{\alpha}n$	for $[\alpha(0), \ldots, \alpha(n-1)]$

If S is a predicate of arity ρ^*, then for $\alpha : \mathsf{nat} \to \rho$ we write

$$\alpha \in S \iff \forall n S(\bar{\alpha}n).$$

We work in $\mathsf{HA}^{\mu} + \Delta$ where Δ comprises the following three axioms.

Modified bar recursion at type ρ

The defining equation for modified bar recursion is

$$\Psi(Y, H, s) \stackrel{\tau}{=} Y(s\#\lambda k.H(k, s, \lambda x.\Psi(Y, H, s * x)))$$

where the type τ is not allowed to contain arrow types and, if τ is an inductive type, μ, then $\vec{\sigma}$ must be empty in each constructor type $\vec{\rho} \to \overrightarrow{\vec{\sigma} \to \mu} \to \mu$. The restriction on τ

guarantees that the following principle of continuity holds in the model of the continuous functionals (see Remark 3.4 below).

Principle of continuity

$$\forall F^{(\mathsf{nat}\to\rho)\to\tau}, \alpha^{\mathsf{nat}\to\rho}\exists n\forall\beta(\bar{\alpha}n = \bar{\beta}n \to F(\alpha) = F(\beta).$$

Principle of relativized quantifier free bar induction

$$\frac{\begin{array}{c}\forall\alpha \in S\exists n P(\bar{\alpha}n)\\ \forall s \in S.\forall x(S(s*x) \to P(s*x)) \to P(s)\\ S([])\end{array}}{P([])}$$

Remark 3.4. Modified bar recursion was investigated in [BO03] and compared with Spector's bar recursion [Spe62] and a variant due to Kohlenbach [Koh90]. In particular, it was shown that all three versions of bar recursion have the continuous functionals and the partial continuous functionals as models. As bar recursion is total in these models, it follows from Plotkin's adequacy result [Plo77] that extending the term rewriting system of HA^μ by bar recursion (viewed as a rewrite rule from left to right) does not destroy termination.

Remark 3.5. 1. To get some familiarity with the principle of relativized bar induction we first of all compare it with the usual principle of bar induction.

$$\frac{\begin{array}{c}\forall\alpha\exists n Q(\bar{\alpha}n)\\ \forall s.Q(s) \to P(s)\\ \forall s.\forall x P(s*x) \to P(s)\end{array}}{P([])}$$

From this principle we obtain the principle of relativized bar induction by putting $Q \equiv P$ and relativizing all finite sequences to S.

2. Second, we give a classical proof of the principle of relativized bar induction to underpin that this is a 'reasonable' axiom.

Proof. Assume $\neg P([])$. Then, because of the second and third premise, there must be an x_0 such that $S([x_0]$, but $\neg P([x_0]))$. Following this pattern we generate an infinite sequence $\alpha := [x_0, x_1, \ldots]$ such that $\forall n S(\bar{\alpha}n)$, but $\neg P(\bar{\alpha}n)$ contradicting the first clause $\forall\alpha\exists n P(\bar{\alpha}n)$ of the axiom.

Remark 3.6. Finally, we give a hint of the roles of these axioms in the main proof. Bar recursion is used to define the realizer Ψ for dependent choice. In order to compute $\Psi_{G,Y}([])$ we need to compute $\Psi_{G,Y}$ for a singleton list, then for a list with two elements, and so on. Now, the principle of continuity tells us that we only have to compute Ψ for all sequences up to a certain length and the principle of relativized bar induction shows us how this length could stepwise be reduced to length 0.

The following lemma is based on the simple fact that for a relevant formula A, $\bot \to A$ is provable in minimal logic.

Lemma 3.4. *Let A be a relevant formula (with possibly free variables). Then there is a (closed) term H such that $\vdash_m H$ **mr** $(X \to A^X)$.*

Proof. Let $\nu := \tau(X)$. $\mathsf{Ind}(A)$. *Case:* Atomic formula. Then $A^X \equiv X$ and we set $H := \lambda x^\nu.x$. *Case:* $A \to B$. By ih we have an H' such that H' **mr** $(X \to B)$. Then we let

$$H := \begin{cases} \lambda x^\nu.H'x & \text{if } \tau(A^X) = *, \\ \lambda x^\nu, y^{\tau(A)}.H'x & \text{otherwise.} \end{cases}$$

Case: $A_0 \wedge A_1$. Then, A_0 and A_1 are relevant. By ih we have terms H_i such that H_i **mr** $(X \to A_i)$ for $i \in \{0,1\}$. We put $H := \lambda x^\nu.\langle H_0 x, H_1 x\rangle$. *Case:* $\forall y^\rho A$. By ih we have a term H' such that H' **mr** $(X \to A)$. We set $H := \lambda x^\nu, y^\rho.\, H'x$. □

Proposition 3.7. Let A be a relevant formula and assume x_0 to be of type ρ. Moreover let DC be the formula

$$\forall n^{\mathsf{nat}} \forall x^\rho \exists^{\mathsf{cl}} y^\rho A_n(x,y) \to \exists^{\mathsf{cl}} f^{\mathsf{nat}\to\rho}.\ f(0) = x_0^\rho \wedge \forall n A_n(f(n), f(n+1)).$$

Then DC^X is realizable in $\mathsf{HA}^\mu + \Delta$.

Proof. By unfolding the classical existential quantifier and substituting X for $\bot$ we obtain DC^X:

$$(\forall n, x^\rho.\ (\forall y^\rho.\ A_n(x,y)^X \to X) \to X) \to \\ (\forall f.\ f(0) = x_0^\rho\ \wedge\ \forall n A_n(f(n), f(n+1))^X \to X) \to X.$$

Now, let $\nu := \tau(X)$ and $\sigma := \tau(A_n(x,y)^X)$. and assume that we have realizers G, Y such that

$$G^{\mathsf{nat}\to\rho\to(\rho\to\sigma\to\nu)\to\nu}\ \mathbf{mr}\ \forall n \forall x^\rho.\ (\forall y^\rho.\ A_n(x,y)^X \to X) \to X$$
$$Y^{(\mathsf{nat}\to\rho)\to(\mathsf{nat}\to\sigma)\to\nu}\ \mathbf{mr}\ \forall f.\ f(0) = x_0^\rho \wedge \forall n\, A_n(f(n), f(n+1))^X \to X.$$

It suffices to find a realizer for X. Let, from now on, β be a variable of type $\mathsf{nat} \to \rho \times \sigma$ and t be a variable of type $(\rho \times \sigma)^*$. We define

$$\Psi_{G,Y}(t) := \tilde{Y}(t \# \lambda n.\langle 0^\rho, H(G(|t|, ([x_0]\#(\pi_0 \circ t))_{|t|}, \lambda y^\rho, z^\sigma.\Psi_{G,Y}(t * \langle y, z\rangle)))\rangle)$$

where π_0 and π_1 are left and right projection of pairing $\langle\cdot,\cdot\rangle$, while

$$\tilde{Y}(\beta) := Y([x_0]\#(\pi_0 \circ \beta), \pi_1 \circ \beta),$$

and H is the closed term, given by Lemma 3.4, such that

$$\forall n, x, y(H\ \mathbf{mr}\ (X \to A_n(x,y)^X)).$$

Next, let

$$\begin{aligned} S(t) &:= \forall i < |t|(\pi_1((t)_i)\ \mathbf{mr}\ A_i(([x_0]\#(\pi_0 \circ t))_i, (\pi_0 \circ t)_i)^X) \\ P(t) &:= \Psi_{G,Y}(t)\ \mathbf{mr}\ X. \end{aligned}$$

Using quantifier free bar induction relativized to S we show $P([\,])$, thus $\lambda G, Y.\Psi_{G,Y}([\,])$ is the realizer we are looking for.

1. Assume that we have a $\beta \in S$, i.e., $\forall n S(\bar{\beta}n)$. We have to show $\exists n P(\bar{\beta}n)$. Let

$$\begin{aligned} f &:= [x_0]\#(\pi_0 \circ \beta) \\ \gamma &:= \pi_1 \circ \beta, \end{aligned}$$

then we have

$$\begin{aligned} S(\bar{\beta}n) &\equiv \forall i < n.\, \pi_1(\beta(i))\ \mathbf{mr}\ A_i(([x_0]\#(\pi_0 \circ \bar{\beta}n))_i, (\pi_0 \circ \bar{\beta}n)_i)^X \\ &\equiv \forall i < n.\, \gamma(i)\ \mathbf{mr}\ A_i(f(i), f(i+1))^X, \\ \forall n S(\bar{\beta}n) &\equiv \forall n.\, \gamma(n)\ \mathbf{mr}\ A_n(f(n), f(n+1))^X \\ &\equiv \gamma\ \mathbf{mr}\ \forall n A_n(f(n), f(n+1))^X. \end{aligned}$$

By the definition of Y we obtain $Yfg\ \mathbf{mr}\ X$, hence $\tilde{Y}(\beta)\ \mathbf{mr}\ X$. Furthermore, by the principle of continuity we know that in order to compute $\tilde{Y}$ we only need to look at finitely many values in the sequence β, i.e., there exists an n such that $\tilde{Y}(\beta) = \tilde{Y}(\bar{\beta}n\#\lambda n.\mathsf{any}(n))$, where $\mathsf{any}(n)$ are any terms of type $\rho \times \sigma$. In particular, we have $\tilde{Y}(\beta) = \Psi_{G,Y}(\bar{\beta}n)$. Hence we have $\Psi(\bar{\beta}n)\ \mathbf{mr}\ X$, i.e., $P(\bar{\beta}n)$.

2. We show $\forall t \in S.\forall q(S(t * q) \to P(t * q)) \to P(t)$. Let $t \in S$ where t is of the form $t = [\langle x_1, z_0\rangle, \ldots, \langle x_n, z_{n-1}\rangle]$ and assume $\forall q(S(t * q) \to P(t * q))$, i.e.,

$$\forall x_{n+1}, z_n.t \in S \wedge z_n\ \mathbf{mr}\ A_n(x_n, x_{n+1})^X \to \Psi_{G,Y}(t * \langle x_{n+1}, z_n\rangle)\ \mathbf{mr}\ X,$$

that is,

$$t \in S \to \lambda x_{n+1}, z_n.\Psi_{G,Y}(t * \langle x_{n+1}, z_n\rangle)\ \mathbf{mr}\ \forall x_{n+1}.A_n(x_n, x_{n+1})^X \to X.$$

Setting $v := \lambda x_{n+1}, z_n.\Psi_{G,Y}(t * \langle x_{n+1}, z_n\rangle)$ and using the realizer G we end up with

$$G(n, x_n, v)\ \mathbf{mr}\ X.$$

On the other hand, by Lemma 3.4, we have a closed term H such that

$$\forall m, x, y.\, H\ \mathbf{mr}\ (X \to A_m(x, y)^X),$$

hence we may conclude $\forall m, x, y.\, H(G(n, x_n, v))$ **mr** $A_m(x, y)^X$. Now, let

$$\begin{aligned} w &:= H(G(n, x_n, v) \\ f &:= [x_0, \ldots, x_n]\#\lambda n.0 \\ \gamma &:= [z_0, \ldots, z_{n-1}]\#\lambda n.w \end{aligned}$$

and recall that

$$\begin{aligned} &\forall i < n.\gamma(i) \textbf{ mr } A_i(f(i), f(i+1))^X && [\text{since } t \in S] \\ &\gamma(n) \textbf{ mr } A_n(f(n), f(n+1))^X && [\text{since } w \textbf{ mr } A_n(x_n, 0)^X] \\ &\forall m > n.\gamma(m) \textbf{ mr } A_m(f(m), f(m+1))^X && [\text{since } w \textbf{ mr } A_n(0, 0)^X] \end{aligned}$$

i.e., γ **mr** $\forall n A_n(f(n), f(n+1))^X$. Again we have

$$\begin{aligned} &Yf\gamma \textbf{ mr } X && [\text{by the definition of } Y \text{ and } \textbf{ mr }] \\ &\Psi_{G,Y}(t) \textbf{ mr } X && [\text{since } Yf\gamma = \tilde{Y}(t\#\lambda n.\langle 0, w\rangle) = \Psi_{G,Y}(t)], \end{aligned}$$

hence $P(t)$.

3. $S([])$. Trivial. □

Remark 3.8. 1. The only restriction in proposition 3.7 is that $A_n(x, y)$ must be a relevant formula. But similar to the question of 'how to obtain definite and goal formulas', this can be achieved by double negating the atomic formulas occurring in the formula A.

2. From a computational point of view the functional Ψ that realizes DC^X is highly inefficient although we know that the functional Y only looks at finitely many entries of its argument. Albeit, since these arguments are constant from a certain point on, they should not be computed anew each time but be stored after being computed the first time (cf. section 6.3).

3.3 An example for using external realizers

We now give an example which demonstrates the A-translation method when an axiom, here the axiom of classical dependent choice, with an external realizer is allowed. The example is motivated by our application of the A-translation method to Higman's lemma and will occur as a lemma in section 6.2. The use of dependent choice could be avoided in this example. However, the example serves as a test example for Higman's Lemma where dependent choice will be essential. A discussion of the same example, without using dependent choice, has been given by Constable and Murthy [CM91]. Our example has been formalized in Minlog and its implementation can be found in the appendix A.1.

The example. Using the axiom of classical dependent choice we show the lemma saying that every infinite boolean valued sequence has an infinite constant subsequence

$$\forall h^{\mathsf{nat}\to\mathsf{boole}}\exists^{\mathsf{cl}} e^{\mathsf{nat}\to\mathsf{nat}}, a^{\mathsf{boole}}\forall k.\ e(k) < e(k+1)\ \wedge\ h(e(k+1)) = a$$

Given such a constant subsequence we clearly obtain the simple corollary

$$\forall h\exists^{\mathsf{cl}} i, j.\ i < j\ \wedge\ h(i) = h(j)$$

where i and j are obtained by taking $e(1)$ and $e(2)$.

We will extract a program from this latter statement.

Informal proof of the lemma. Assume we are given an infinite sequence $h : \mathsf{nat} \to \mathsf{boole}$. Then we argue by classical case distinction. In case the infinite sequence eventually becomes constant with value T, i.e., there is an n such that $\forall m.n < m \to h(m) = T$, we simply put

$$e := \lambda k.\ n + k, \quad a := T$$

Otherwise, for all n there is an m such that $n < m\ \wedge\ h(m) = F$. Applying the axiom of dependent choice with $A(n, m) :\equiv n < m\ \wedge\ h(m) = F$, we obtain an e such that

$$\forall k.\ e(k) < e(k+1)\ \wedge\ h(e(k+1)) = F,$$

hence we are done by using this e and $a := F$.

Formalization. In order to formalize these proofs in a setting suitable for A-translation some changes are necessary. We recall the requirements: (1) The proof is to be carried out in minimal logic. (2) Only definite assumptions are used (and of course the goal formula should be a goal formula in the formal sense). (3) DC is applied to a relevant formula.

In our informal proof, we used classical case distinction. However, since the goal is a negated one, the case distinction on a formula A can be simulated in minimal logic by a cut of the formula $\neg A$. Next, we implicitly used the assumption

$$((a = T) \to \bot) \to a = F$$

which is neither provable without the logical Efq-axiom nor is a definite formula. It can be transformed into a definite formula (and also be proven without Efq) when we double negate the statement $a = F$. Finally, we have to use (DC) with $A(n, m) :\equiv \neg\neg(n < m\ \wedge\ h(m) = F)$. Therefore the statement we are actually proving is

$$\forall h^{\mathsf{nat}\to\mathsf{boole}}\exists^{\mathsf{cl}} e^{\mathsf{nat}\to\mathsf{nat}}, a^{\mathsf{boole}}\forall k.\ \neg\neg(e(k) < e(k+1)\ \wedge\ h(e(k+1)) = a)$$

using the (definite) assumption

$$((a = T) \to \bot) \to (a = F \to \bot) \to \bot.$$

Realizing dependent choice. Although a general treatment would be possible, for simplicity, we define the realizer for the specific instance of (DC) as it is used in the example. Thus, the types are $\rho \equiv \mathsf{nat}, \nu \equiv \mathsf{nat} \times \mathsf{nat}$, and $\sigma \equiv \nu \to \nu$. We use the algebra of reverse lists: $\mathsf{tsil}\,\alpha$ with the constructors `Lin`, displayed as `[]`, and `Snoc`, displayed as infix operator `::`. Moreover we have constants for the length, `Lh`: $\mathsf{tsil}\,\alpha \to \mathsf{nat}$ and for projection, displayed infix as underscores.

Here, we restrict ourselves to presenting the realizer; the information about missing types and constants can be read off from the definitions in the preceeding chapter or from the demo file in the appendix. The realizer for the axiom of dependent choice is

```
(lambda (G) (lambda (Y) (Psi G Y [])))
```

where for `Psi` we have the rewrite rule

```
(add-computation-rule
 (pt "Psi G Y t")
 (pt "Tilde Y
            ([n][if (n < (Lh t))
                    (t__n)
                    (0@
                      H (G [if (Lh t = 0)
                               0
                               (left (t__(Pred (Lh t))))]
                           ([y,z] (Psi G Y (t::(y@z))))))])"))
```

which relies on the realizer $H : \nu \to \nu \to \nu$ for ex-falso-quodlibet. In this case H is just the term $\lambda p_1, p_2 . p_1$.

The extracted program. The program obtained by the MINLOG system is

```
(lambda (h0)
  (((Psi
      (lambda (n1)
        (if ((h0 (Succ n1))= True)
            (if ((h0 (Succ (Succ n1))) = True)
                (lambda (f2) ((Succ n1) @ (Succ (Succ n1))))
                (if ((h0 (Succ (Succ n1))) = False)
                    (lambda (f2)
                      ((f2 (Succ (Succ n1))) (lambda (p3) p3)))
                    (lambda (f2) (0 @ 0))))
            (if ((h0 (Succ n1)) = False)
                (lambda (f2) ((f2 (Succ n1)) (lambda (p3) p3)))
                (lambda (f2) (0 @ 0))))))
    (lambda (e1)
```

```
    (lambda (g2)
      ((g2 0) ((g2 1) ((e1 1) @ (e1 2)))))))
(Lin nat@@(nat@@nat=>nat@@nat))))
```

where $f : \mathsf{nat} \to (\nu \to \nu) \to \nu$, $g : \mathsf{nat} \to \nu \to \nu$ are new variables and `@` is used for the display of pairs. Applying the term rewriting rule for `Psi` we end up with a program that consists of groundterms only and uses 12 case distinctions. We discuss the behavior of the program by means of some sample sequences given by their initial segments.

$$FFFFF \;\Rightarrow\; 1,2 \qquad FTFTF \;\Rightarrow\; 2,4$$
$$TFTFT \;\Rightarrow\; 1,3 \qquad FTFTT \;\Rightarrow\; 3,4.$$

In general, the first element is never considered since the axiom of dependent choice only yields a sequence that is constant after the first index. Further, the behavior is not symmetric and two indices i and j with $h(i) = h(j) = T$ are only found if there are no other letters in between.

An alternative version of the example. As an optimal algorithm to this problem we would expect an algorithm that only considers the first three elements. Indeed such an algorithm can be obtained by a reformulation of the statement as follows:

$$\forall h^{\mathsf{nat}\to\mathsf{boole}} \exists^{\mathsf{cl}} e^{\mathsf{nat}\to\mathsf{nat}} \forall k.\ \neg\neg(e(k) < e(k+1)\ \wedge\ h(e(k)) = h(e(k+1))).$$

This formulation allows for a symmetric proof but requires an assumption (or lemma) of the form

$$\forall a, b, c.\ (a = b \to \bot) \to (a = c \to \bot) \to (b = c \to \bot) \to \bot$$

whose treatment through the A-translation gives rise to exactly three case distinctions stirring the according extracted program.

Generalizations. 1. A natural generalization appears when instead of asking for a pair of indices, we look for a finite, increasing list of given length, on which h is constant:

$$\forall h^{\mathsf{nat}\to\mathsf{boole}}, n^{\mathsf{nat}} \exists^{\mathsf{cl}} ns^{\mathsf{tsil}\,\mathsf{nat}}.|ns| = n\ \wedge\ \mathsf{Inc}\,ns\ \wedge\ \mathsf{Const}\,h\,ns,$$

the predicates Inc and Const being defined in the obvious way. It turns out that this example could be treated in better way when using a version of the axiom of dependent choice as discussed in chapter 6.1.

2. Ramsey's Theorem. For an arbitrary $(A, \leq_A)$ we may show

$$\forall h^{\mathsf{nat}\to A} \exists^{\mathsf{cl}} e^{\mathsf{nat}\to\mathsf{nat}} \forall k.\ e(k) < e(k+1)\ \wedge\ h(e(k)) \leq_A h(e(k+1)$$

which is a version of Ramsey's Theorem (see section 4.1). The proof is similar to our example albeit it requires two applications of the axiom of dependent choice.

4 Higman's Lemma and Kruskal's Theorem

Having examined in detail the theory we will now focus on its applications. This chapter is introductory in character, but also presents essential terms used in subsequent chapters. It refers to Nash-Williams' classical proof of Higman's Lemma and Kruskal's Theorem, gives different formulations of a well quasiorder and a sketch of their equivalences, and concludes with an overview on constructive proofs of Higman's Lemma and Kruskal's theorem.

4.1 Nash-Williams' minimal-bad-sequence proof

Well quasiorders

Traditionally, Higman's Lemma and Kruskal's Theorem are formulated in terms of well quasiorders. We recall the definition of a well quasiorder given in the introduction. Let, in this section, Q be a set with a relation $\leq_Q$ (or $\leq$ in short if the dependency is clear).

Definition. We call an infinite sequence $(q_i)_{i<\omega}$ good if $\exists i, j.\ i < j\ \wedge\ q_i \leq q_j$, otherwise it is called bad. Let $(Q, \leq)$ be a quasiorder (i.e., $\leq$ is reflexive and transitive). Then, $(Q, \leq)$ is a well quasiorder, in short $\mathsf{Wqo}\, Q$, if every infinite sequence in Q is good.

Remark 4.1. 1. It is easy to see that $(Q, \leq)$ is a well quasiorder if and only if there is neither a strictly decreasing infinite sequence in Q nor an infinite subset of pairwise incomparable elements.

2. For historical reasons we require well quasiorders to be transitive. However, transitivity, will not play any role in our constructive proofs (This is important since there might be cases in which proving transitivity is expensive.) We have used transitivity in the classical proof; however, it could easily be avoided. Moreover, also reflexivity need not to be presupposed since one can prove $q \leq q$ once one has the property $\forall (q_i)_{i<\omega} \exists i, j.\ i < j \wedge q_i \leq q_j$ by simply applying it to the constant sequence $(q)_i$.

3. In our thesis quasiorders are required to be decidable.

We start with a lemma, which is usually attributed to Ramsey [Ram30]. Interestingly, there are two versions of Ramsey's theorem. The second involves the cartesian product of two orders together with the product order.

Ramsey's Theorem

Lemma 4.1. *Let $(Q, \leq)$ be a well quasiordering. Then, every infinite sequence $(q_i)_{i<\omega}$ has a weakly increasing infinite subsequence, i.e., there exists $(\kappa_i)_{i<\omega}$ such that $\forall i, j.\ i < j \to q_{\kappa_i} \leq q_{\kappa_j}$.*

Proof. Let $(q_i)_{i<\omega}$ be an infinite sequence. We call an element q_m terminal if there is no $n > m$ such that $q_m \leq q_n$. The number of terminal elements must be finite, since otherwise we would have an infinite bad subsequence. Therefore there is an M such that q_M is terminal, and q_n is not terminal for all $n > M$. Now, starting with q_{M+1} and using the axiom of dependent choice, we can build our weakly increasing infinite sequence. □

Lemma 4.2. $\mathsf{Wqo}\, P \wedge \mathsf{Wqo}\, Q \to \mathsf{Wqo}\, P \times Q$.

Proof. Assume that we are given an infinite sequence $(\langle p_i, q_i\rangle)_{i<\omega}$. By Lemma 4.1 we obtain an infinite weakly increasing subsequence $(p_{\kappa_i})_{i<\omega}$ (with respect to $\leq_P$) and by the well quasiorderedness of Q we know that for $(q_{\kappa_i})_{i<\omega}$ there are indices $i < j$ such that $q_{\kappa_i} \leq_Q q_{\kappa_j}$. □

Higman's Lemma

Definition. On the set A^* of finite sequences in a given alphabet A with a quasiorder $\leq_A$ we define an embeddability relation, denoted by $\leq_{A^*}$, as follows: A sequence $[a_1, \ldots, a_n]$ is embeddable in $[b_1, \ldots, b_m]$ if there is a strictly increasing map $f\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_i \leq_A b_{f(i)}$, for all $i \in \{1, \ldots n\}$.

Proposition 4.2 (Higman's Lemma).

> If $(A, \leq_A)$ is a well quasiorder,
> then so is the set $(A^*, \leq_{A^*})$ of finite sequences in A.

Proof. Let A be a well quasiorder and assume for contradiction that there is a bad sequence in A^*. Among all infinite bad sequences we choose a 'minimal' bad sequence as follows: Choose a word w_0 such that w_0 starts an infinite bad sequence, but any (proper) initial segment of w_0 does not. Now assuming that we have already determined $w_0, \ldots, w_n$, we choose w_{n+1} such that there is an infinite bad sequence that starts with $w_0, \ldots, w_{n+1}$ but no sequence starting with $w_0, \ldots, w_n, v$, in case v is an initial segment of w_{n+1}. Assuming the axiom of dependent choice this process yields an infinite sequence $(w_i)_{i<\omega}$ which we call minimal. Since $(w_i)_{i<\omega}$ is bad, we know $w_i \neq [\,]$ for all i. Therefore we may define words v_i and letters a_i such that $w_i = v_i{*}a_i$ for all i. Using Ramsey's theorem, i.e., Lemma 4.1, and the fact that our alphabet A is a well quasiorder, there is an infinite subsequence $a_{\kappa_0} \leq_A a_{\kappa_1} \leq_A \cdots$ of the sequence $(a_i)_{i<\omega}$. This also determines a corresponding sequence $w_0, \ldots, w_{\kappa_0 - 1}, v_{\kappa_0}, v_{\kappa_1}, \ldots$. The sequence $w_0, \ldots, w_{\kappa_0 - 1}, v_{\kappa_0}, v_{\kappa_1}, \ldots$. must be bad (otherwise $(w_i)_{i<\omega}$ would have been good), hence contradicts the choice of the minimal bad sequence. □

Kruskal's Theorem

Definition. Let $T(A)$ be the set of finite trees with labels in A. A tree is embeddable into another if there exists a one to one map on them such that, (1), infima of nodes are respected and, (2), the label of each node is less or equal to that of its image. We denote the embeddability relation by $\leq_{T(A)}$.

Proposition 4.3 (Kruskal's Theorem).

> If $(A, \leq_A)$ is a well quasiorder,
> then so is the set $(T(A), \leq_{T(A)})$ of finite trees.

Proof. Assume for contradiction that there is an infinite bad sequence of trees. In analogy to Higman's Lemma we select an infinite minimal bad sequence $(t_i)_{i<\omega}$ such that for all i $t_0, \ldots, t_i$ starts an infinite bad sequence, but $t_0, \ldots, t_{i-1}, u$, in case u is a subtree of t_i, does not. Now, we assume that in this sequence each tree t_i consists of a root a_i and immediate subtrees $u_{i0}, \ldots, u_{ik_i-1}$. Let S be the collection of all immediate subtrees occurring in $(t_i)_{i<\omega}$. If there were a bad sequence in S, there would be a bad sequence $(u_{n_i m_i})_{i<\omega}$ where $0 \leq m_i \leq k_{n_i-1}$ and $n_i < n_j$ for all $i < j < \omega$. Then, also $t_0, \ldots, t_{n_0-1}, u_{n_0 m_0}, u_{n_1 m_1}, \ldots$ would be bad, contradicting the choice of the minimal bad sequence. Therefore S is a well quasiorder, and so, thanks to Higman's Lemma, is S^*. Now, by Ramsey's theorem, we know that $A \times S^*$ is a well quasiorder. In particular for the sequence $((a_i, [u_{i0}, \ldots, u_{ik_i-1}]))_{i<\omega}$ we can find i and j such that the i-th element is 'embeddable' in the j-th element. Therefore, looking at the corresponding trees, we have $t_i \leq_{T(A)} t_j$, contradicting the badness of chosen sequence. □

4.2 Equivalent characterizations of well quasiorderings

From a constructive point of view, the problem with the preceeding definition of a well quasiorder is, that, constructively, we can not inspect infinitely many arguments in a sequence. However, we may use the fact that 'good' is a so-called open property, meaning that an infinite sequence has this property if and only if there is a finite initial segment which has this property. Therefore, instead of talking about infinite sequences, we may argue about finite sequences, e.g., in a theory of inductive definitions as described below.

Definition (good/bad for finite sequences). We call a finite sequence $[q_0, \ldots, q_{n-1}]$ with elements in Q good if there exist $i < j < n$ such that $q_i \leq_Q q_j$; otherwise we called it bad. We also use the predicate Good_Q (or Good without the parameter) in order to express that the sequence is good. $\mathrm{Bad}(Q)$ denotes the set of all finite bad sequences.

Notation. Let Q^* be the set of finite sequences $[q_0, \ldots, q_{n-1}], n \geq 0$ with elements q_i in Q. We make the variables ps, qs to denote elements in Q^* and use the following notations: $qs \# qs'$ for the concatenation of qs and qs' and $qs * q$ for $qs \# [q]$,

An inductive characterization of a well quasiorder

We inductively define a predicate called Bar_Q (short Bar, if the dependency is clear) which contains all finite sequences that are either good or which, being extended by one element finitely often, eventually become good. Explained in terms of infinite sequences, $\mathsf{Bar}\, qs$ holds if all infinite sequences starting with qs are good. $\mathsf{Bar}\,[\,]$, therefore, is an inductive characterization of a well quasiorder as will explicitly be shown in Lemma 4.3. For convenience, we present the introduction axioms for the inductive definition not as axioms as it was proposed in chapter 2 but, in a mathematically more legible style, as rules.

Definition. We inductively define a set $\mathsf{Bar} \subseteq Q^*$ via the following rules:

$$\frac{\mathsf{Good}\, qs}{\mathsf{Bar}\, qs} \qquad \frac{\forall q\, \mathsf{Bar}\, qs{*}q}{\mathsf{Bar}\, qs}$$

Lemma 4.3. *Let $(Q, \leq)$ be a (decidable) quasiorder. Then*

$$\mathsf{Bar}\,[\,] \iff \mathsf{Wqo}(Q).$$

Proof. "$\Longrightarrow$". We show, more generally,

$$\forall qs.\, \mathsf{Bar}\, qs \to \forall (q_i)_{i<\omega},\, \forall n.\, [q_0, \ldots, q_{n-1}] = qs \to \exists i < j.\, q_i \leq q_j$$

by induction on Bar. The statement then follows by letting $qs = [\,]$.

1. $\mathsf{Good}\, qs$. Assume that there are an infinite sequence $(q_i)_{i<\omega}$ and a number n such that $[q_0, \ldots, q_{n-1}] = qs$. Since qs is good, there are $i < j$ such that $(qs)_i \leq (qs)_j$ and therefore also such that $q_i \leq q_j$.

2. We have the induction hypothesis:

$$\forall q,\, \forall (q_i)_{i<\omega},\, \forall n.\, [q_0, \ldots, q_{n-1}] = qs{*}q \to \exists i < j.\, q_i \leq q_j.$$

Assume that an infinite sequence $(q_i)_{i<\omega}$ and an n such that $[q_0, \ldots, q_{n-1}] = qs$ are given. Then, our goal follows by using the induction hypothesis with q_{n+1}.

"$\Longleftarrow$". This direction is an instance of Brouwer's axiom of bar induction which is usually considered to be intuitionistically acceptable. We also give a classical proof: assume that $\mathsf{Bar}\,[\,]$ does not hold. Then, by the definition of Bar there is an element $q_0 \in Q$ such that $\neg\mathsf{Bar}\,[q_0]$. By iteration (using the Axiom of Dependent Choice) we get a sequence $(q_i)_{i<\omega}$, such that for all n the following holds: $\neg\mathsf{Bar}\,[q_0, \ldots, q_{n-1}]$ and $\forall i < j \leq n\, q_i \not\leq q_j$. This contradicts Q being a well quasiorder. □

Remark 4.4. The definition of Bar and the "$\Longrightarrow$" direction of this proof have been formalized in MINLOG (See appendix A.2). While Bar is an inductive predicate with computational content, for simplicity, Good is formalized as an inductive definition without computational content. Hence, we prove that every infinite sequence has a good initial segment.

In order to motivate our second inductive characterization of a well quasiorder by means of an accessibility definition, we first look at the tree of bad sequences.

The tree of bad sequences

Definition. On the set $\mathrm{Bad}(Q)$ of bad sequences in Q we define a relation $\ll_Q$ by

$$qs' \ll_Q qs :\leftrightarrow qs' = qs * q \text{ for some } q \in Q.$$

Remark 4.5. $(\mathrm{Bad}(Q), \ll_Q)$ forms a tree with the empty sequence as root; therefore this set is often called the tree of bad sequences. $\ll_Q$ is a partial order on $\mathrm{Bad}(Q)$ since it is irreflexive, antisymmetric and transitive. For a set Q it can easily be shown that to be well quasiordered is equivalent to the fact that there is no infinite decreasing sequence (wrt $\ll_Q$) in $\mathrm{Bad}(Q)$, i.e.,

$$(Q, \leq_Q) \text{ is wqo} \iff (\mathrm{Bad}(Q), \ll_Q) \text{ is well-founded.}$$

The well-foundedness of $\mathrm{Bad}(Q)$ implies an induction principle usually phrased 'induction along the well-founded tree of bad sequences.'

A second inductive characterization

Definition. The accessible part (also called the well-founded part) of the relation $\ll_Q \subseteq \mathrm{Bad}(Q) \times \mathrm{Bad}(Q)$ is inductively given by the rule

$$\frac{\forall qs'.\ qs' \ll_Q qs \ \to\ \mathsf{acc}_{\ll_Q} qs'}{\mathsf{acc}_{\ll_Q} qs}$$

Lemma 4.4. $\mathsf{acc}_{\ll_Q} [] \iff (Q, \leq)$ *is a wqo.*

Proof. This is similar to the proof of Lemma 4.3. □

We present two lemmas (see also [Sei01b]) which illustrate some basic properties of this second characterization of a well quasiorder. We will refer to them in chapter 7.

Definition. For $qs \in \mathrm{Bad}(Q)$ let

$$Q_{qs} := \{q \in Q : qs * q \in \mathrm{Bad}(Q)\}.$$

$\leq_{Q_{qs}}$ is the relation $\leq_Q$ restricted to Q_{qs}.

Lemma 4.5. $(\forall q \in Q.\, \mathsf{acc}_{\ll_{Q_{[q]}}} []) \to \mathsf{acc}_{\ll_Q} []$.

Proof. Assume $\forall q \in Q\, \mathsf{acc}_{\ll_{Q_{[q]}}}[]$. By definition of $\mathsf{acc}_{\ll_Q}$ we need to show $\forall q \in Q\, \mathsf{acc}_{\ll_Q}[q]$. Let $q \in Q$. By assumption we have $\mathsf{acc}_{\ll_{Q_{[q]}}}[]$. Then $\mathsf{acc}_{\ll_Q}[q]$ may be obtained from the following more general assertion (by setting $ps = [q]$ and $qs = []$):

$$\forall ps, qs.\, \mathsf{acc}_{\ll_{Q_{ps}}} qs \to \mathsf{acc}_{\ll_Q} ps \# qs.$$

We prove this assertion by induction on the predicate acc.

$\mathsf{Ind}(\mathsf{acc}_{\ll_{Q_{ps}}})$. Fix ps, qs and assume $\mathsf{ih}\colon \forall qs' \ll_{Q_{ps}} qs\, \mathsf{acc}_{\ll_Q} ps \# qs'$. Again by definition of $\mathsf{acc}_{\ll_Q}$ it suffices to show $\mathsf{acc}_{\ll_Q} ps \# qs * q$ for an arbitrary $q \in Q_{ps\#qs}$. This follows by the induction hypothesis since $qs * q \ll_{Q_{ps}} qs$ and $(ps \# qs) * q = ps \# (qs * q)$. □

Definition. Let $(P, \leq_P)$, $(Q, \leq_Q)$ be quasiorders. A quasi embedding from $(P, \leq_P)$ to $(Q, \leq_Q)$ is an injective map $f\colon P \to Q$, that does not create any new $\leq$–relations; i.e. for all $p_1, p_2 \in P$:

$$f(p_1) \leq_Q f(p_2) \to p_1 \leq_P p_2.$$

Lemma 4.6. *Let $f\colon P \to Q$ be a quasi embedding. Then we have*

$$\mathsf{acc}_{\ll_Q}[] \to \mathsf{acc}_{\ll_P}[].$$

Proof. We show $\forall qs.\, \mathsf{acc}_{\ll_Q} qs \to \forall ps.\, f(ps) = qs \to \mathsf{acc}_{\ll_P} ps$, where $f(ps)$ means that the quasi embedding f is applied component wise to ps, i.e., $f([p_1, \ldots, p_n]) = [f(p_1), \ldots, f(p_n)]$. Clearly, $\mathsf{acc}_{\ll_Q}[] \to \mathsf{acc}_{\ll_P}[]$ follows by $ps = qs = []$.

$\mathsf{Ind}(\mathsf{acc}_{\ll_Q})$. Fix qs and assume $\mathsf{ih}\colon \forall qs' \ll_Q qs, \forall ps.\, f(ps) = qs' \to \mathsf{acc}_{\ll_P} ps$. Let ps s.t. $f(ps) = qs$. We have to show $\mathsf{acc}_{\ll_P} ps$, i.e. $\forall p \in P_{ps}\, \mathsf{acc}_{\ll_P} ps * p.\ p \in P_{ps}$ satisfies $(ps)_i \not\leq_P p$ for all $i \leq |ps|$ and, since f is a quasi embedding, it follows $f((ps)_i) \not\leq_Q f(p)$, i.e. $(qs)_i \not\leq_Q f(p)$. Hence $qs * f(p) \ll_Q qs$, and by ih, applied to $qs * f(p)$ and $ps * p$, we obtain $\mathsf{acc}_{\ll_P} ps * p$. □

The characterizations of a well quasiorder, presented up to now, will be essential for the later chapters. In order to be able to compare various proofs at the end of this chapter, we give some further characterizations.

Reifications

Definition. A reification of a quasi order $(Q, \leq)$ into a well ordering $(\sigma, <)$ is a map

$$r : \mathrm{Bad}(Q) \to \sigma,$$

such that for all $qs * q \in \mathrm{Bad}(Q)$: $r(qs * q) < r(qs)$.

Remark 4.6. $(Q, \leq)$ wqo $\iff$ there is a reification of $(Q, \leq)$ into a well ordering σ.

The maximal order type of a well quasiordering

Definition. We call the height of the tree of bad sequences $\mathrm{Bad}(Q)$ the maximal order type of the well quasiorder Q. It is denoted by $|\mathrm{Bad}(Q)|$.

Remark 4.7. 1. There is a an alternative characterization of the ordertype of a well quasiorder as supremum of the ordertypes of all extensions of $(Q, \leq)$ to a linear order. This was the characterization originally used by de Jongh/Parikh [dJP77] and Schmidt [Sch79]. If the ordertype of such an extension of $(Q, \leq)$ is α, then we have

$$\alpha \leq |\mathrm{Bad}(Q)|.$$

2. If there is a reification of Q into a well order with order type β, then

$$|\mathrm{Bad}(Q)| \leq \beta.$$

4.3 On constructive proofs of Higman's Lemma and Kruskal's Theorem

There are several constructive proofs of Higman's Lemma. However, as will be shown by analyzing the underlying proof principles and constructions, most of them have the same computational content. To this end we give a sample proof of $\mathsf{Wqo}\, A \to \mathsf{Wqo}\, A^*$ without referring to any particular characterization of a well quasiorder. We conclude this chapter by presenting a classification containing all proofs also including those of Kruskal's theorem.

The underlying construction

We start by explaining the underlying construction. We write $A \subseteq B$ in order to express that A can be considered as a subset of B. More precisely $A \subseteq B$ means that there exists a quasi embedding from A to B. Now, the combinatorial idea is that for a given word $v = [a_1, \ldots, a_n]$ the 'space' available to extend the singleton sequence $[v]$ in a bad way – we denote this space by $(A)^*_{[v]}$ – can be considered as a subset of another set consisting of disjoint unions and cartesian products of some known sets. By induction hypothesis, this set can be proven to be well quasiordered. Formally, this idea is covered by the following lemma.

Lemma 4.7. $(A)^*_{[[a_1,\ldots,a_n]]} \subseteq \bigcup\{(A_{[a_1]})^* \times A \times \cdots \times A \times (A_{[a_l]})^* : l < n\}$.

Proof. Assume $w \in A^*_{[[a_1,\ldots,a_n]]}$, i.e., $v = [a_1, \ldots, a_n] \not\leq w$. Then there is an l, $0 \leq l < n$ such that $[a_1, \ldots, a_l] \leq_{A^*} w$, but $[a_1, \ldots, a_{l+1}] \not\leq_{A^*} w$. In addition, there are $b_1, \ldots, b_l$ and $w_1, \ldots, w_{l-1}$ such that $a_i \leq b_i$ for all i, $1 \leq i \leq l$ and $w_i \in A_{[a_i]}$ for all i, $1 \leq i \leq l+1$. Then w can be identified with $(w_1, b_1, \ldots, b_l, w_{l+1})$ □

The essentials of the proof

Suppose that using any of our particular characterizations of a well quasiorder we have proven the statements:

(1) $(\forall q\ \mathsf{Wqo}\, Q_{[q]}) \to \mathsf{Wqo}\, Q$
(2) $P \subseteq Q \to \mathsf{Wqo}\, Q \to \mathsf{Wqo}\, P$
(3) $\mathsf{Wqo}\, P \wedge \mathsf{Wqo}\, Q \to \mathsf{Wqo}\, P \cup Q$
(4) $\mathsf{Wqo}\, P \wedge \mathsf{Wqo}\, Q \to \mathsf{Wqo}\, P \times Q$.

Then the proof of Higman's Lemma

$$\mathsf{Wqo}\, A \to \mathsf{Wqo}\, A^*$$

is as follows. Assume $\mathsf{Wqo}\, A$; then by induction hypothesis, since, e.g., $[a]$ is 'lower' than $[]$ in the well-founded tree of bad sequences, we have for all $a \in A$:

(5) $\mathsf{Wqo}\, A_{[a]} \to \mathsf{Wqo}\, {A_{[a]}}^*$.

By (1) it suffices to prove $\forall v\ \mathsf{Wqo}\, A_{[v]}$. Hence let $v = [a_1, \ldots, a_n]$ and by Lemma 4.7 and (2) we are done once we have shown

$$\mathsf{Wqo}\,(\bigcup\{(A_{[a_1]})^* \times A \times \cdots \times A \times (A_{[a_l]})^* : l < n\}).$$

But this assertion holds by (3), (4) and (5).

The assertions (1) and (2) are already proven, using an inductive characterization (compare Lemma 4.5 and Lemma 4.6.) The inductive proofs for (3) and (4) are straightforward. Note that a classical proof of (4) has been given in Lemma 4.2.

There is one part in this general argument where we have been a bit sloppy, that is when the induction principle is applied: in the case of the inductive characterization of a wellquasiorder (using the acc-notation) (5) is not directly given. In this case, Higman's Lemma should first be generalized to $\forall as.\ \mathsf{acc}_{\ll_A} as \to \mathsf{acc}_{\ll_{(A_{as})^*}} []$.

We conclude this section with a classification of all proofs of Higman's Lemma and Kruskal's theorem, mentioned in the introduction, thereby showing how the content of the next three chapters fits in the landscape of proofs.

Table 1: Classification of proofs of Higman's Lemma and Kruskal's Theorem

	Higman's Lemma [Hig52]	Kruskal's Thm [Kru60]
Classical	Nash-Williams [NW63]	Nash-Williams [NW63]
A-translation	Murthy [Mur90], [a] S., chapter. 6, [b]	
ID (Bar)	Coquand, Fridlender [CF94], [c] S., chapter 5, [b]	
Max. ordertype	deJongh, Parikh [dJP77]	Schmidt [Sch79]
Reifications	Schütte, Simpson [SS85]	Rathjen, Weiermann [RW93] (Hasegawa [Has94])
ID (acc)	Fridlender [Fri93], [d] Richman, Stolzenberg [RS93] (Murthy, Russell [MR90]) (Cichon, Tahhan Bittar [CTB94])	S., chapter. 7
Intuitionistic	Veldman [Vel00]	Veldman [Vel00]
ID (Bar)	Fridlender [Fri97], [d]	

[a] Formalized in Nuprl; the A-translated version of this proof has been implemented by Herbelin in Coq [Her94].
[b] Formalized in Minlog.
[c] Higman's Lemma for a two letter alphabet has been implemented in Minlog and by Berghofer in Isabelle [Ber03].
[d] Formalized in Alf.

In table 1 in each case, the left column gives a hint on the method or the characterization of a well quasiorder used in the according proofs. With respect to the combinatorial idea,

we mainly distinguish three types of proofs. In the table these three types are separated by a double line in the tabular.

The first group comprises the classical proof and their constructive counterparts which form the topic of the next two chapters. Note that, among the classical proofs, we are only mentioning the minimal-bad-sequence proof; an overview of other classical proofs may be found in [Fri97]. The second group is the one which make use of the construction explained in this section. Although not completely constructively formalized, we include the proofs given in [dJP77, Sch79] since they use the same construction. Finally, proofs that do not require decidability for the relation on the alphabet have been given by Veldman, using intuitionistic methods. Fridlender [Fri97] has transformed the proof for Higman's Lemma into a proof using inductive definitions. This proof as well as the analysis of the constructive proofs of Higman's Lemma in the second group have been investigated in our diploma thesis [Sei98], and therefore will not be considered in detail in this thesis.

Some of the proofs have also been formalized in the theorem prover - this information has been included in form of footnotes.

5 An inductive version of Nash-Williams' proof of Higman's Lemma

We have seen the short and elegant proof of Higman's Lemma due to Nash-Williams using the so-called Minimal-Bad-Sequence-Argument. The objective of this section is to present a proof of Higman's Lemma that uses the same combinatorial idea as Nash-Williams' classical proof, but which is constructive. For the case of a two letter alphabet such a proof was given by Coquand and Fridlender [CF94]. Using more flexible structures, we present a proof that works for an arbitrary well quasiordered alphabet.

An earlier version of our proof has been published in [Sei01a]. Here, it is improved with respect to the computational content which only can be read off when using a 'positive' formulation of a well quasiorder (cf. remark 5.3).

The proof is based on the inductive characterization of a well quasiorder via the predicate Bar which was introduced in section 4.2. Thus the statement we are going to prove in this section is

$$\mathsf{Bar}_A\,[] \to \mathsf{Bar}_{A^*}\,[].$$

5.1 Basic definitions

We assume $(A, \leq_A)$ to be a set with a reflexive and transitive, decidable relation[3].

Notation. We use

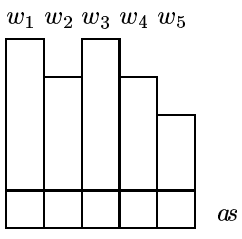
$a, b, \ldots$	for letters, i.e., elements of a A,
$as, bs, \ldots$	for finite sequences of letters, i.e., elements of A^*,
$v, w, \ldots$	for words, i.e., elements of A^*, [4]
$vs, ws, \ldots$	for finite sequences of words, i.e., elements of A^{**} ,
$vss \ldots$	for elements in A^{***} .

Definition (Higman embedding). The embedding relation on A^* can be inductively described by the following rules:

$$\frac{}{[] \leq_{A^*} []} \qquad \frac{v \leq_{A^*} w}{v \leq_{A^*} w*a} \qquad \frac{v \leq_{A^*} w,\ a \leq_A b}{v*a \leq_{A^*} w*b}.$$

[3] Whereas transitivity is only required for historical reasons, but is not used in our proof, decidability plays an essential role.

[4] Although of the same kind, we distinguish between finite sequences (of letters), as, and words, w, because they will play different roles, as is illustrated in the picture on the right.



Definition. For a finite sequence ws of non-empty words let $\mathsf{lasts}\, ws$ denote the finite sequence consisting of the end-letters of the words of ws, that is,[5]

$$\mathsf{lasts}\,[w_0{*}a_0, \ldots, w_{n-1}{*}a_{n-1}] = [a_0, \ldots, a_{n-1}],\ n \geq 0.$$

If ws contains an empty word, for simplicity, we set $\mathsf{lasts}\, ws := []$.

Definition. Good and Bad are used to express that a finite sequence is good, bad respectively. Furthermore, we use the notion $\mathsf{good}(as, a)$ if there is an element in as, say the i-th one, such that $(as)_i \leq_A a$. $\mathsf{bad}(as, a)$ stands for $\neg\mathsf{good}(as, a)$.

Further, $\mathsf{bseq}(as)$ determines the 'first' bad subsequence occurring in as:

$$\begin{aligned} \mathsf{bseq}([]) &= [] \\ \mathsf{bseq}(as{*}a) &= \begin{cases} \mathsf{bseq}(as){*}a & \text{if } \mathsf{bad}(\mathsf{bseq}(as), a), \\ \mathsf{bseq}(as) & \text{otherwise.} \end{cases} \end{aligned}$$

Finally, we recall the definition of the predicate Bar_A (see section 4.2).

Definition. We inductively define a set $\mathsf{Bar}_A \subseteq A^*$ via the following rules:

$$\frac{\mathsf{Good}\, as}{\mathsf{Bar}_A\, as} \qquad \frac{\forall a\, \mathsf{Bar}_A\, as{*}a}{\mathsf{Bar}_A\, as}.$$

The preceeding two definitions should be understood for arbitrary relations, not only for our fixed $(A, \leq_A)$. They are also used for $(A^*, \leq_{A^*})$.

5.2 The analogy between the classical and the constructive proof

In order to motivate further definitions we first want to highlight the idea behind the constructive proof. This is best done by showing the connection between the classical and the constructive proof. To this extend we shortly recall the structure of Nash-Williams' minimal-bad-sequence proof (see section 4.1) and show how the main steps are captured by the inductive proof.

The steps of Nash-Williams' proof:

1. In order to show '$\mathsf{Wqo}\,(A)$ implies $\mathsf{Wqo}\,(A^*)$', assume for contradiction that there is a bad sequence of words.

[5] In our picture we have $\mathsf{last}[w_1, \ldots, w_5] = as$.

2. Among all infinite bad sequences we choose (using classical dependent choice) a minimal bad sequence, i.e., a sequence $(w_i)_{i<\omega}$, such that, for all n, $w_0, \ldots, w_n$ starts an infinite bad sequence, but $w_0, \ldots, w_{n-1}, v$, where v is an initial segment of w_n, does not.

3. Since for all i $w_i \neq [\,]$, let $w_i = v_i{*}a_i$. By Ramsey's theorem and the fact that our alphabet A is a well quasiorder, there exists an infinite subsequence $a_{\kappa_0} \leq_A a_{\kappa_1} \leq_A \cdots$ of the sequence $(a_i)_{i<\omega}$. This also determines a corresponding sequence $w_0, \ldots, w_{\kappa_0-1}, v_{\kappa_0}, v_{\kappa_1}, \ldots$.

4. The sequence $w_0, \ldots, w_{\kappa_0-1}, v_{\kappa_0}, v_{\kappa_1}, \ldots$ must be bad (otherwise also $(w_i)_{i<\omega}$ would be good), but this contradicts the minimality in 2.

In the constructive proof these steps correspond to

1. Prove inductively '$\mathsf{Bar}_A\,[\,] \to \mathsf{Bar}_{A^*}\,[\,]$'.

2. The minimality argument will be replaced by structural induction on words.

3. Given a sequence $ws = [w_0, \ldots, w_n]$ s.t. $w_i = v_i{*}a_i$, we are interested in all subsequences $a_{\kappa_0} \leq_A \cdots \leq_A a_{\kappa_l}$ of maximal length[6] and their corresponding sequences $w_0, \ldots, w_{\kappa_0-1}, v_{\kappa_0}, \ldots, v_{\kappa_l}$. The sequences $[a_{\kappa_0}, \ldots, a_{\kappa_l}]$ form a forest. In the proof these sequences will be computed by the procedure forest which takes ws as input and yields a forest labeled by pairs in $A^{**} \times A$. In the produced forest the right-hand components of each path form such an ascending subsequence $[a_{\kappa_0}, \ldots, a_{\kappa_l}]$ and the left-hand component of each label with a right-hand component a_{κ_i} consists of the sequence $w_0, \ldots, w_{\kappa_0-1}, v_{\kappa_0}, \ldots, v_{\kappa_i}$. If we extend the sequence ws by a word $v{*}a$, then in the existing forest either new nodes, possibly at several places, are inserted, or a new singleton tree with node $\langle ws{*}v, a\rangle$ is added. Now the informal idea of the inductive proof is: if in $\mathsf{forest}\,ws$ new nodes can not be inserted infinitely often (without ending up with a good left-hand component in a node) and if also new trees can not be added infinitely often, then ws can not be extended badly infinitely often. Formally, this will be captured by the statement: $\forall ws.\, \mathsf{Bar}_A\,\mathsf{bseq}(\mathsf{lasts}\,ws) \to \mathsf{Barforest}\,\mathsf{forest}\,ws \to \mathsf{Bar}_{A^*}\,ws$.

4. The first part of item 4 corresponds to Lemma 5.1.

The formal definition of the procedure forest and the inductive definition of the predicate $\mathsf{Barforest}$ expressing that the forest 'behaves in a wellfounded way' will be given in the next section.

[6] By maximal length we mean that we only look at those subsequences which are ascending, but not contained in other ones, for instance our chosen subsequences of $[1, 4, 3, 0, 3]$ are $[1, 4]$, $[1, 3, 3]$ and $[0, 3]$.

5.3 Forests

Definition. We use

t	for elements in $T(A^{**} \times A)$, i.e., trees labeled by pairs in $A^{**} \times A$,
f, ts	for elements in $(T(A^{**} \times A))^*$, i.e., forests.

The tree with root $\langle ws, a\rangle$ and list of subtrees ts is written $\langle ws, a\rangle ts$. We use the destructors left and right for pairs and the destructors root and $\mathsf{subtrees}$ for trees, hence $\mathsf{root}\,\langle ws, a\rangle ts = \langle ws, a\rangle$ and $\mathsf{subtrees}\,\langle ws, a\rangle ts = ts$. For better readability we set:

$$
\begin{aligned}
&\mathsf{newtree}\,\langle ws, a\rangle && := \langle ws, a\rangle[\,],\\
&\mathsf{roots}\,[t_1, \ldots, t_n] && := [\mathsf{root}\,t_1, \ldots, \mathsf{root}\,t_n],\\
&\mathsf{lefts}\,[\langle vs_1, a_1\rangle, \ldots, \langle vs_n, a_n\rangle] && := [vs_1, \ldots, vs_n],\\
&\mathsf{rights}\,[\langle vs_1, a_1\rangle, \ldots, \langle vs_n, a_n\rangle] && := [a_1, \ldots, a_n].
\end{aligned}
$$

We now come to the formal definition of the forest of a finite sequence of words. Note that forest is intended to be a partial function, i.e., is intended to be only defined for a sequence of non-empty words. However, it turns out that it is convenient to put $\mathsf{forest}\,(ws * [\,]) := [\,]$ and it simplifies the formalization.

Definition. Let $ws \in A^{**}$ be a sequence of words. Then $\mathsf{forest}\,ws \in (T(A^{**} \times A))^*$ is recursively defined by:

$$
\begin{aligned}
&\mathsf{forest}\,[\,] && = [\,],\\
&\mathsf{forest}\,ws * [\,] && = [\,],\\
&\mathsf{forest}\,ws{*}(w{*}a) && = \begin{cases} \mathsf{insertforest}(\mathsf{forest}\,ws, w, a) & \text{if } \mathsf{good}(\mathsf{bseq}(\mathsf{lasts}\,ws), a)\\ (\mathsf{forest}\,ws){*}\mathsf{newtree}\,\langle ws{*}w, a\rangle & \text{otherwise,}\end{cases}
\end{aligned}
$$

where

$$
\mathsf{insertforest}(f, w, a) = \mathrm{map}\left(\lambda t \left[\begin{array}{ll} \text{if} & \mathsf{right}\,(\mathsf{root}\,t) \leq_A a\\ & \mathsf{inserttree}(t, w, a)\\ & t\end{array}\right]\right)\ f
$$

and

$$
\begin{aligned}
&\mathsf{inserttree}(\langle vs, a'\rangle ts, w, a) =\\
&\qquad \begin{cases} \langle vs, a'\rangle\mathsf{insertforest}(ts, w, a) & \text{if } \mathsf{good}(\mathsf{rights}\,(\mathsf{roots}\,ts), a),\\ \langle vs, a'\rangle(ts * \mathsf{newtree}\,\langle vs{*}w, a\rangle) & \text{otherwise.}\end{cases}
\end{aligned}
$$

Definition. Let $f \in (T(A^{**} \times A))^*$. Then $\mathsf{Goodforest}\, f$ (read 'f is a good forest') holds if there is a label in f such that the left-hand side of this label is good. On $(T(A^{**} \times A))^*$ we define a relation $\prec$ by

$$f' \prec f :\leftrightarrow\ f' \neq f\ \wedge\ \exists w, a.\ f' = \mathsf{insertforest}(f, w, a)$$

Finally, we inductively define the predicate $\mathsf{Barforest} \subseteq (T(A^{**} \times A))^*$ via the following rules

$$\frac{\mathsf{Goodforest}\, f}{\mathsf{Barforest}\, f} \qquad \frac{\forall f'.\, f' \prec f\ \rightarrow\ \mathsf{Barforest}\, f'}{\mathsf{Barforest}\, f}.$$

5.4 The proof of Higman's Lemma

Lemma 5.1. *Let ws be a sequence of words. Then*

i) $\mathsf{Goodforest}\,\mathsf{forest}\, ws \rightarrow \mathsf{Good}\, ws$.

ii) $\mathsf{bseq}(\mathsf{lasts}\ ws) = \mathsf{rights}(\mathsf{roots}\ (\mathsf{forest}\ ws))$.

Proof. This follows from the construction of forest. □

Lemma 5.2. *i)* $\mathsf{Barforest}\,[\,]$.

ii) $\forall f, t.\ \mathsf{Barforest}\, f \wedge \mathsf{Barforest}\,[t] \rightarrow \mathsf{Barforest}\, f * t$.

Proof. i) $\mathsf{Barforest}\,[\,]$ follows from the second rule of the definition of $\mathsf{Barforest}$, using Efq.

ii) This assertion holds since $\mathsf{insertforest}$ is defined by a map operation. Formally, we prove

$$\forall f_1.\ \mathsf{Barforest}\, f_1 \rightarrow \forall f_2.\ \mathsf{Barforest}\, f_2 \rightarrow \mathsf{Barforest}\, f_1 \# f_2.$$

by induction on $\mathsf{Barforest}\, f_1$ and $\mathsf{Barforest}\, f_2$. In order to show $\mathsf{Barforest}\, f_1 \# f_2$ let f' such that $f' \prec f_1 \# f_2$ and show $\mathsf{Barforest}\, f'$.

Case 1: $f' = f_1 \# f_2'$ such that $f_2' \prec f_2$. Then $\mathsf{Barforest}\, f_1 \# f_2'$ follows by the second induction hypothesis.

Case2: $f' = f_1' \# f_2'$ such that $f_1' \prec f_1$ and $f_2' \prec f_2$ (or $f_2' = f_2$). In this case $\mathsf{Barforest}\, f_1' \# f_2'$ can be obtained by

$$\mathsf{ih}_1 :\ \forall f_1'.\, f_1' \prec f_1 \rightarrow \forall f_2.\, \mathsf{Barforest}\, f_2 \rightarrow \mathsf{Barforest}\, f_1' \# f_2,$$

using the strengthening of the second induction hypothesis $\forall f_2'.\, f_2' \prec f_2 \rightarrow \mathsf{Barforest}\, f_2'$ (in the case $f_2' \prec f_2$). □

The next lemma tells us that a forest consisting of only one tree, in which we continue to insert new nodes by $\mathsf{insertforest}$ operations, eventually becomes good.

Lemma 5.3. *Assume* $\mathsf{Bar}_A\,[\,]$. *Then*

$$\forall ws.\,\mathsf{Bar}_{A^*}\,ws \to \forall a.\,\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws, a\rangle].$$

Proof. $\mathsf{Ind}_1(\mathsf{Bar}_{A^*})$: 1.1. $\mathsf{Good}\,ws$. Then we have $\mathsf{Goodforest}\,[\mathsf{newtree}\,\langle ws, a\rangle]$ since the left-hand-side of the root label is good, i.e., $\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws, a\rangle]$. 1.2. Assume

$$\mathsf{ih}_1 : \forall w, a.\,\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws{*}w, a\rangle].$$

Let $a \in A$. Instead of proving $\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws, a\rangle]$ we show more generally that this assertion holds for all t such that $\mathsf{root}\,t = \langle ws, a\rangle$ and (a) $\mathsf{subtrees}\,t$ is in $\mathsf{Barforest}$, and (b) $\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t))$ is in Bar_A. We do this by main induction on (b) and side induction on (a), i.e., formally we prove

$$\begin{array}{ll}
\forall as. & \mathsf{Bar}_A\,as \to \\
\forall ts. & \mathsf{Barforest}\,ts \to \\
\forall t. & \mathsf{root}\,t = (ws, a) \to \mathsf{subtrees}\,t = ts \to \\
 & \mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t)) = as \to \mathsf{Barforest}\,[t].
\end{array}$$

$\mathsf{Ind}_2(\mathsf{Bar}_A)$. 2.1. $\mathsf{Good}\,as$. Assume that there is a t such that

$$\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t)) = as.$$

Since by construction $\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t))$ for any t is bad, this leads to a contradiction and the result follows by ex-falso-quodlibet. 2.2. $\mathsf{Bar}_A\,as$ is obtained by the second rule. We fix an as, assume ih_2 and have to show $\forall ts.\,\mathsf{Barforest}\,ts \to \forall t.\,\mathsf{root}\,t = (ws, a) \to \mathsf{subtrees}\,t = ts \to \mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t)) = as \to \mathsf{Barforest}\,[t]$.

$\mathsf{Ind}_3(\mathsf{Barforest})$. 3.1. Fix an ts such that $\mathsf{Goodforest}\,ts$. Then for any t such that $\mathsf{subtrees}\,t = ts$, $\mathsf{Goodforest}\,ts$ implies $\mathsf{Goodforest}\,[t]$, i.e., $\mathsf{Barforest}\,[t]$. 3.2. Fix ts and assume $\mathsf{ih}_{3a} : \forall ts'.\,ts' \prec ts \to \mathsf{Barforest}\,ts'$ and

$$\begin{array}{lll}
\mathsf{ih}_{3b}: & \forall ts'. & ts' \prec ts \to \\
 & \forall t. & \mathsf{root}\,t = (ws, a) \to \mathsf{subtrees}\,t = ts' \to \\
 & & \mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t)) = as \to \mathsf{Barforest}\,[t].
\end{array}$$

Fix t such that $\mathsf{root}\,t = \langle ws, a\rangle$, $\mathsf{subtrees}\,t = ts$, and $\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t)) = as$. Then, we have to prove $\mathsf{Barforest}\,[t]$. Unfolding the definition of $\mathsf{Barforest}$ it suffices to show $\mathsf{Barforest}\,[t']$ where $t' = \mathsf{inserttree}(t, w, a') \neq t$ for some $w \in A^*$ and a' such that $a < a'$. We prove the assertion by case distinction on the definition of $\mathsf{inserttree}$.

Case 1. $t' = \langle ws, a\rangle\,(ts * \mathsf{newtree}\,\langle ws{*}w, a'\rangle)$ for some w and a' such that $\mathsf{bad}(as, a')$. Then we have

$$\begin{array}{ll}
\mathsf{root}\,t' & = \langle ws, a\rangle, \\
\mathsf{subtrees}\,t' & = ts * \mathsf{newtree}\,\langle ws{*}w, a'\rangle, \\
\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t')) & = as{*}a'.
\end{array}$$

We may apply ih_2 to $as{*}a', ts * \mathsf{newtree}\,\langle ws{*}w, a'\rangle$ and t' and conclude $\mathsf{Barforest}\,[t']$ once we have proven

$$\mathsf{Barforest}\,ts * \mathsf{newtree}\,\langle ws{*}w, a'\rangle.$$

By Lemma 5.2, using $\mathsf{Barforest}\,ts$ which holds by ih_{3a}, it suffices to show

$$\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws{*}w, a'\rangle].$$

But this follows by ih_1.

Case 2. $t' = \langle ws, a\rangle\,\mathsf{insertforest}(ts, w, a')$ where a' such that $\mathsf{good}(as, a')$. In this case we have

$$\begin{aligned}
&\mathsf{root}\,t' && = \langle ws, a\rangle,\\
&\mathsf{subtrees}\,t' && = \mathsf{insertforest}(ts, w, a'),\\
&\mathsf{rights}\,(\mathsf{roots}\,(\mathsf{subtrees}\,t')) && = as.
\end{aligned}$$

Then $[t'] \prec [t]$ implies $\mathsf{subtrees}\,t' \prec \mathsf{subtrees}\,t$ and by ih_{3b}, applied to $\mathsf{subtrees}\,t'$ and t', we end up with $\mathsf{Barforest}\,[t']$.

Now, the proof of the general assertion is completed, and we may put $as = []$, $f = []$ and $t = \mathsf{newtree}\,\langle ws, a\rangle$. Since we have $\mathsf{Bar}_A\,[]$ by assumption and $\mathsf{Barforest}\,[]$ by Lemma 5.2, we obtain $\mathsf{Bar}_{A^*}\,ws \to \mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws, a\rangle]$. □

Proposition 5.1 (Higman's Lemma). $\mathsf{Bar}_A\,[] \to \mathsf{Bar}_{A^*}\,[]$.

Proof. Assume $\mathsf{Bar}_A\,[]$. We show more generally

$$\begin{aligned}
&\forall as. \quad \mathsf{Bar}_A\,as \to\\
&\forall f. \quad \mathsf{Barforest}\,f \to\\
&\forall ws. \quad \mathsf{bseq}(\mathsf{lasts}\,ws) = as \to \mathsf{forest}\,ws = f \to \mathsf{Bar}_{A^*}\,ws.
\end{aligned}$$

$\mathsf{Ind}_1(\mathsf{Bar}_A)$. 1.1. $\mathsf{Good}\,as$. Then, the result follows by ex-falso-quodlibet since for any ws, $\mathsf{bseq}(\mathsf{lasts}\,ws)$ is bad. 1.2. Let $as \in A^*$ and assume $\mathsf{ih}_1\colon \forall a\,\forall f.\,\mathsf{Barforest}\,f \to \forall ws.\,\mathsf{bseq}(\mathsf{lasts}\,ws){=}as{*}a \to \mathsf{forest}\,ws{=}f \to \mathsf{Bar}_{A^*}\,ws$.

$\mathsf{Ind}_2(\mathsf{Barforest})$. 2.1. $\mathsf{Goodforest}\,f$. Then, by Lemma 5.1, i), for any ws such that $\mathsf{forest}\,ws = f$, we obtain $\mathsf{Good}\,ws$ and hence $\mathsf{Bar}_{A^*}\,ws$. 2.2. Fix an f and assume $\mathsf{ih}_{2a}\colon \forall f'.\,f' \prec f \to \mathsf{Barforest}\,f'$ and $\mathsf{ih}_{2b}\colon \forall f'.\,f' \prec f \to \forall ws.\,\mathsf{bseq}(\mathsf{lasts}\,ws) = as \to \mathsf{forest}\,ws = f' \to \mathsf{Bar}_{A^*}\,ws$. Assume that we have ws such that $\mathsf{bseq}(\mathsf{lasts}\,ws) = as$ and $\mathsf{forest}\,ws = f$. In order to prove $\mathsf{Bar}_{A^*}\,ws$, we fix a word w and show $\mathsf{Bar}_{A^*}\,ws{*}w$ by induction on the structure of w:

$\mathsf{Ind}_3(w)$. 3.1. $\mathsf{Bar}_{A^*}\,ws{*}[]$ holds since the empty word is embeddable in any word. 3.2. Assume that we have a word of form $w{*}a$. We show $\mathsf{Bar}_{A^*}\,ws{*}(w{*}a)$ by case analysis on whether or not $\mathsf{good}(as, a)$.

Case 1. $\mathsf{bad}(as, a)$. Then we have

$$\begin{aligned}\mathsf{bseq}(\mathsf{lasts}\,(ws{*}(w{*}a))) &= as{*}a,\\ \mathsf{forest}\,(ws{*}(w{*}a)) &= f * \mathsf{newtree}\,\langle ws{*}w, a\rangle.\end{aligned}$$

By $\mathsf{ih_{2a}}$ and $\mathsf{ih_3}$, we have $\mathsf{Barforest}\,f$ and $\mathsf{Bar}_{A^*}\,ws{*}w$. Hence, by Lemma 5.3, applied to $ws{*}w$ and a, we obtain $\mathsf{Barforest}\,[\mathsf{newtree}\,\langle ws{*}w, a\rangle]$ and by Lemma 5.2 we may conclude

$$\mathsf{Barforest}\,f * \mathsf{newtree}\,\langle ws{*}w, a\rangle.$$

Now, we are able to apply $\mathsf{ih_1}$ (to a, $f * \mathsf{newtree}\,\langle ws{*}w, a\rangle$ and $ws{*}(w{*}a)$) and end up with $\mathsf{Bar}_{A^*}\,ws{*}(w{*}a)$.

Case 2. $\mathsf{good}(as, a)$. In this case, it follows

$$\begin{aligned}\mathsf{bseq}(\mathsf{lasts}\,(ws{*}(w{*}a))) &= as,\\ \mathsf{forest}\,(ws{*}(w{*}a)) &= \mathsf{insertforest}(f, w, a).\end{aligned}$$

Moreover, by 5.1, ii) and the definition of $\mathsf{forest}\,(ws{*}(w{*}a))$, we know that at least one node has been inserted into $\mathsf{forest}\,ws$, hence

$$\mathsf{insertforest}(f, w, a) \prec f.$$

In this situation, we may apply $\mathsf{ih_{2b}}$ (to $\mathsf{insertforest}(f, w, a)$ and $ws{*}(w{*}a)$) and conclude $\mathsf{Bar}_{A^*}\,ws{*}(w{*}a)$.

This completes the proof of the general assertion. Now, by putting $as = [\,]$, $f = [\,]$ and $ws = [\,]$ and the fact that $\mathsf{Barforest}\,[\,]$ always holds (cf. Lemma 5.2) we obtain $\mathsf{Bar}_A\,[\,] \to \mathsf{Bar}_{A^*}\,[\,]$. □

Remark 5.2. Note that there is a certain freedom in modelling the construction behind the Nash-Williams proof. We discuss some alternatives in order to contribute to a wider understanding. All choices, bar one, have no essential influence on the resulting program.

First of all, we may define forests by distinguishing cases only in the definition of insertforest. Then, the definition of forests in the step case amounts to

$$\mathsf{forest}\,(ws * (w * a)) = \mathsf{insertforest}(\mathsf{forest}\,ws, ws, w, a)$$

where, however, we now need the additional argument ws, and, accordingly, the definition of $\prec$ has to be modified to

$$f' \prec f :\leftrightarrow \mathsf{roots}\,(f') = \mathsf{roots}\,(f) \wedge \exists ws, w, a.f' = \mathsf{insertforest}(f, ws, w, a).$$

Concerning the generation of forests, a second decision is to store sequences of words instead of words in the left-hand-sides of the labels. Conversely, we may add new nodes

in form of $\mathsf{newtree}\,\langle w, a\rangle$, but, in order to show that a forest is good, it is desirable to have stored somewhere all earlier occurred words that are of interest.

A choice which changes the behavior of the resulting program is how often a new given node should be inserted. Our decision to insert wherever reasonable is influenced by the idea that the extracted program should yield some of the first indices fulfilling the desired property.

Finally, one may dislike the $\exists$-quantifier in the definition of $\mathsf{Barforest}$. An alternative definition is

$$\frac{\forall w, a.\, \mathsf{insertforest}(f, w, a) \neq f \to \mathsf{Barforest}\,\mathsf{insertforest}(f, w, a)}{\mathsf{Barforest}\; f}.$$

Remark 5.3. In order to make the computational content behind the inductive proof visible, it is essential to use a 'positive' formulation of a well quasiorder, that is, a definition using two rules, as was pointed out, e.g., in [Fri93]. Having a proof of $\mathsf{Bar}_{A^*}\, ws$ implies that the proof yields the information whether $\mathsf{Bar}_{A^*}\, ws$ was obtained by the first rule or by the second. In the first case the result can be read off, in the second we continue with looking at a proof of $\mathsf{Bar}_{A^*}\, ws * w$ for some w. It might happen that ws is good, but our proof of $\mathsf{Bar}_{A^*}\, ws$ was obtained by the second rule, so we continue with the search. If we used a definition consisting of only one rule, i.e., an acc-notion as we did in [Sei01a], $\mathsf{Bar}_{A^*}\, ws$ would correspond to the statement 'for all w such that $\mathsf{bad}(ws, w)$ implies $\mathsf{Bar}_{A^*}\, ws * w$' where the test whether or not $\mathsf{bad}(ws, w)$ results in a brute-force search; it is not given by the proof itself.

5.5 Formalization in MINLOG

In MINLOG we have formalized two proofs, first, the proof of Higman's Lemma for a two letter alphabet due Coquand and Fridlender in [CF94], and, secondly, the proof given in a preceeding section, instantiated to a finite alphabet. Both implementations can be found in the appendix.

In the case of a finite alphabet instead of an arbitrary alphabet, the proof given in the preceeding section simplifies with respect to the structure of the forests. They now only consist of non-branching trees and, therefore, reduce to folders. Furthermore, we let $a \leq_A b := a = b$. Then $\mathsf{bseq}(as)$ yields the sequence of all letters, occurring in as, without repetitions. For a better understanding of the implementation, we briefly recall this special case of our proof.

Definition. Let $ws \in A^{**}$ be a sequence of words. Then $\mathsf{folder}\, ws \in A^{***}$ is defined recursively by:

$$\begin{array}{ll}
\mathsf{folder}\,[\,] & = [\,],\\
\mathsf{folder}\,ws * [\,] & = [\,],\\
\mathsf{folder}\,ws{*}(w{*}a) & = \begin{cases} \mathsf{insertfolder}(\mathsf{folder}\,ws, w, i) & \text{if } a{=}(\mathsf{bseq}(\mathsf{lasts}\,ws))_i\\ (\mathsf{folder}\,ws) * (ws{*}w) & \text{otherwise,}\end{cases}
\end{array}$$

where

$$\mathsf{insertfolder}([vs_0, \ldots, vs_n], w, i) = [vs_0, \ldots, vs_i{*}w, \ldots, vs_n].$$

Definition. We inductively define a predicate $\mathsf{Bars} \subseteq A^{***}$ via the following rules

$$\frac{i < |vss| \wedge \mathsf{Good}\,(vss)_i}{\mathsf{Bars}\,vss} \qquad \frac{\forall w, i.i < |vss| \to \mathsf{Bars}\,(\mathsf{insertfolder}(vss, w, i))}{\mathsf{Bars}\,vss}.$$

Next, we reformulate Lemma 5.1 and Lemma 5.2. Lemma 5.3 becomes trivial.

Lemma 5.4. *Let ws be a sequence of words. Then*

i) $\mathsf{Good}\,(\mathsf{folder}\,ws)_i \to \mathsf{Good}\,ws$.

ii) $|\mathsf{bseq}(\mathsf{lasts}\,ws)| = |\mathsf{folder}\,ws|$.

Proof. Both follows from the construction of folder. □

Lemma 5.5. *i)* $\mathsf{Bars}\,[\,]$. *ii)* $\mathsf{Bar}_{A^*}\,us \wedge \mathsf{Bars}\,vss \to \mathsf{Bars}\,vss{*}vs$.

Proof. ii) $\mathsf{Ind}_1(\mathsf{Bar})$. $\mathsf{Ind}_2(\mathsf{Bars})$. □

Proposition 5.4 (Higman's Lemma for a finite alphabet). $\mathsf{Bar}_{A^*}\,[\,]$.

Proof. We show more generally

$$\begin{array}{ll}
\forall as. & \mathsf{Bar}_A\,as \to\\
\forall vss. & \mathsf{Bars}\,vss \to\\
\forall ws. & \mathsf{bseq}(\mathsf{lasts}\,ws) = as \to \mathsf{folder}\,ws = vss \to \mathsf{Bar}_{A^*}\,ws.
\end{array}$$

$\mathsf{Ind}_1(\mathsf{Bar}_A)$. 1.1. $\mathsf{Good}\,as$. Then, the goal follows by ex-falso-quodlibet since, for any ws, $\mathsf{bseq}(\mathsf{lasts}\,ws)$ is bad. 1.2. Let $as \in A^*$ and assume

$$\mathsf{ih}_1 : \forall a\,\forall vss.\,\mathsf{Bars}\,vss \to \forall ws.\,\mathsf{bseq}(\mathsf{lasts}\,ws){=}as{*}a \to \mathsf{folder}\,ws{=}vss \to \mathsf{Bar}_{A^*}\,ws.$$

$\mathsf{Ind}_2(\mathsf{Bars})$. 2.1. $\exists i < |vss|.\,\mathsf{Good}\,(vss)_i$. Then, by Lemma 5.4, i), for any ws such that $\mathsf{folder}\,ws = vss$, we obtain $\mathsf{Good}\,ws$, hence $\mathsf{Bar}_{A^*}\,ws$. 2.2. Fix vss and assume $\mathsf{ih}_{2\mathsf{a}} : \forall v, i.\mathsf{Bars}\,\mathsf{insertfolder}(vss, v, i)$ and $\mathsf{ih}_{2\mathsf{b}} : \forall v, i, ws.\ \mathsf{bseq}(\mathsf{lasts}\,ws) = as \to \mathsf{folder}\,ws = \mathsf{insertfolder}(vss, v, i) \to \mathsf{Bar}_{A^*}\,ws$. Now, assume that we have ws such that $\mathsf{bseq}(\mathsf{lasts}\,ws) =$

as and $\mathsf{folder}\, ws = \mathsf{insertfolder}(vss, v, i)$. In order to prove $\mathsf{Bar}_{A^*}\, ws$, we fix a word w and show $\mathsf{Bar}_{A^*}\, ws{*}w$ by induction on the structure of w.

$\mathsf{Ind}_3(w)$. 3.1. $\mathsf{Bar}_{A^*}\, ws{*}[]$ holds since the empty word is embeddable in any word. 3.2. Assume that we have a word of form $w{*}a$. We show $\mathsf{Bar}_{A^*}\, ws{*}(w{*}a)$ by case analysis on whether or not $\mathsf{bad}(as, a)$.

Case 1. $\mathsf{bad}(as, a)$. Then, we obtain

$$\begin{aligned} \mathsf{bseq}(\mathsf{lasts}\,(ws{*}(w{*}a))) &= as{*}a, \\ \mathsf{folder}\,(ws{*}(w{*}a)) &= vss{*}(ws{*}w). \end{aligned}$$

By ih_{2a} and ih_3, we have $\mathsf{Bars}\, vss$ and $\mathsf{Bar}_{A^*}\, ws{*}w$ and thanks to Lemma 5.5 we may conclude

$$\mathsf{Bars}\, vss{*}(ws{*}w).$$

Now, we are able to apply ih_1 (to a, $vss{*}(ws{*}w)$ and $ws{*}(w{*}a)$) and end up with $\mathsf{Bar}_{A^*}\, ws{*}(w{*}a)$.

Case 2. $\mathsf{good}(as, a)$. Let i such that $(as)_i = a$. In this case, we have

$$\begin{aligned} \mathsf{bseq}(\mathsf{lasts}\,(ws{*}(w{*}a))) &= as, \\ \mathsf{folder}\,(ws{*}(w{*}a)) &= \mathsf{insertfolder}(vss, w, i). \end{aligned}$$

Here we may apply ih_2 (to w, i and $ws{*}(w{*}a)$) and conclude $\mathsf{Bar}_{A^*}\, ws{*}(w{*}a)$.

This completes the proof of the general assertion. Now, let $as = [], vss = []$ and $ws = []$, then from $\mathsf{Bars}\,[]$ and $\mathsf{Bar}_A\,[]$ which holds for finite alphabet we obtain $\mathsf{Bar}_{A^*}\,[]$. □

Remark 5.5. In the appendices A.3 and A.4, in each case, we have extracted a program that for a given infinite sequence of words yields a good initial segment (uniquely determined by its length). Instead of an extensive discussion of the extracted programs we simply refer to the implementation. Note that the formalization in A.4, although more general concepts involving, is not much longer than that in A.3. The reason for this is mainly that the program extraction mechanism in MINLOG allows for using unproven assumptions (as long as these assumptions are computationally meaningless). An example for such an assumption is lemma 5.4 whose correctness is obvious from the construction but which is indeed elaborate to prove.

Also note that the general proof in this section does not directly yield the Coquand/Fridlender proof when restricted to a 0/1 alphabet, but a proof organized in a different way. Nevertheless, both proofs, the Coquand/Fridlender proof [CF94] and the restricted general proof, result in programs which behave in the same way since the underlying combinatorial idea is still the same. Furthermore, when applied to the sequence

```
[0 0], [1], [1 0], [ ], ...
```

they yield a good initial segment whose length is greater than 3, showing that the shortest good initial sequence is not always found and therefore the algorithms differ from a simple search algorithm.

6 A-translation of Nash-Williams' proof of Higman's Lemma

Whilst in the preceeding section we have given a proof of Higman's Lemma that can be considered as an inductive analogue to the classical proof, in this section we are interested in a direct transformation of the classical Nash-Williams proof into a constructive one and extract a program.

The main ingredient of the classical proof is the minimal bad sequence argument which can be proven conveniently using an alternative version of the axiom of dependent choice. This second version, called DC-seq, can be derived from the ordinary DC, discussed in section 3.2, as we will show in Lemma 6.1. In the implementation on Higman's Lemma, however, we prefer to directly provide a realizer for this new version (in the same style as it has been done for DC) thereby omitting an additional recursion in the program. The main difference between the two versions of the axiom of dependent choice is that DC yields an infinite sequence in which only the n-th and $n+1$-th element are connected, whereas in the sequence obtained by DC-seq the $n+1$-th element is related to all previously computed elements up to index n.

6.1 An equivalent formulation of classical dependent choice

Let DC-seq be the scheme

$$B([]) \to (\forall xs^{\rho^*}.\ B(xs) \to \exists^{\mathsf{cl}} x^{\rho}.\ B(xs * x)) \to \exists^{\mathsf{cl}} g^{\mathsf{nat}\to\rho} \forall n.\ B(\bar{g}n).$$

Lemma 6.1. $\mathsf{HA}^{\omega} \ \vdash_i \ DC \to DC\text{-}seq$.

Proof. Assume, (1), $B([\,])$ and, (2), $\forall xs.\ B(xs) \to \exists^{\mathsf{cl}} x.\ B(xs * x)$. We have to show $\exists^{\mathsf{cl}} g.\ \forall B(\bar{g}n)$. From (2), using $\mathsf{efq}_B : \bot \to B$, we can derive

$$\forall xs \exists^{\mathsf{cl}} x. B(xs) \to B(xs * x).$$

In the following, for a ys of form $xs * x$ we use the notations $\mathsf{lead}(ys) := xs$ and $\mathsf{last}(ys) := x$. Thereby, we easily obtain

$$\forall xs \exists^{\mathsf{cl}} ys.\ \mathsf{lead}(ys) = xs \wedge (B(xs) \to B(ys))$$

By DC for type ρ^*, $x_0 := []$ and $A(xs, ys) :\equiv \mathsf{lead}(ys) = xs \wedge (B(xs) \to B(ys))$, we have

$$\exists^{\mathsf{cl}} f. f(0) = [] \wedge \forall n. \mathsf{lead}(f(n+1)) = f(n) \wedge (B(f(n)) \to B(f(n+1))). \quad (*)$$

Now, we set

$$g := \lambda n. \mathsf{last}(f(n+1))$$

and we show that $\forall n.B(\bar{g}n)$. Indeed, we prove more generally

$$\forall n.B(\bar{g}n) \ \wedge \ \bar{g}n = f(n)$$

by induction on n. $n = 0 : B([])$ holds by (1) and $\bar{g}0 = [] = f(0)$ by $(*)$.
$n \to n+1$: By ih, we have $B(f(n))$, hence by $(*)$, $B(\bar{g}(n+1))$. We conclude $\bar{g}(n+1) = (\bar{g}n) * g(n) \stackrel{\mathsf{ih}}{=} f(n) * \mathsf{last}(f(n+1)) \stackrel{(*)}{=} \mathsf{lead}(f(n+1)) * \mathsf{last}(f(n+1)) = f(n+1)$. □

Remark 6.1. 1. We used efq_B in the proof. This could be avoided when B is a formula which 'ends' with $\bot$, i.e., is relevant (this includes that in a conjunction both subformulas are relevant). 2. Note that the direction DC-seq $\to$ DC also holds. Therefore, indeed, both formulations of dependent choice are equivalent.

Proposition 6.2. Let B be relevant. Let Δ is the axiom system defined in section 3.2 and DC-seq be the formula

$$B([]) \to (\forall xs^{\rho^*}.B(xs) \to \exists^{\mathsf{cl}} x^{\rho}.B(xs * x)) \to \exists^{\mathsf{cl}} g^{\mathsf{nat}\to\rho}.\forall n B(\bar{g}n).$$

Then DC-seq^X is realizable in $\mathsf{HA}^\omega + \Delta$.

Proof. Similar to Berger and Oliva's proof, presented in 3.2. By unfolding the classical existential quantifier and substituting X for $\bot$ we build DC-seq^X:

$$\begin{array}{l} B([])^X \to \\ (\forall xs^{\rho^*}.B(xs)^X \to (\forall x^{\rho}.B(xs * x)^X \to X) \to X) \to \\ (\forall g^{\mathsf{nat}\to\rho}.\forall n B(\bar{g}n)^X \to X) \to X. \end{array}$$

Let $\nu := \tau(X), \sigma := \tau(B^X)$ and assume that we have realizers G_0, G, Y such that

$$\begin{array}{lll} G_0^{\rho} & \mathbf{mr} \ B([])^X & (1) \\ G^{\rho^*\to\sigma\to(\rho\to\sigma\to\nu)\to\nu} & \mathbf{mr} \ \forall xs^{\rho^*}.B(xs)^X \to (\forall x^{\rho}.B(xs*x)^X \to X) \to X & (2) \\ Y^{(\mathsf{nat}\to\rho)\to(\mathsf{nat}\to\sigma)\to\nu} & \mathbf{mr} \ \forall g.\forall n B(\bar{g}n)^X \to X & (3) \end{array}$$

It suffices to find a realizer for X. Let β be a variable of type $\mathsf{nat} \to \rho \times \sigma$ and t be a variable of type $(\rho \times \sigma)^*$. Moreover, we extend the notation $\bar{\beta}n$ to finite sequences, i.e., given an n such that $0 \le n \le |t|$, we let $\bar{t}n := [t_0, \ldots, t_{n-1}]$. We define, using bar recursion,

$$\Psi(t) := \tilde{Y}(t\#\lambda n.\langle [], H(G(\pi_0 \circ t, ([G_0]\#(\pi_1 \circ t))_{|t|}, \lambda x^{\rho}, z^{\sigma}.\Psi(t * \langle x, z\rangle)))\rangle)$$

where Ψ may depend on G_0, G and Y,

$$\tilde{Y}(\beta) := Y(\pi_0 \circ \beta, [G_0]\#(\pi_1 \circ \beta)),$$

and H is the closed term given by Lemma 3.4 such that

$$\forall xs.\ H\ \mathbf{mr}\ (X \to B(xs)^X).$$

Next, let

$$\begin{array}{lcl} S(t) & := & \forall i \leq |t|.\ ([G_0]\#(\pi_1 \circ t))_i\ \mathbf{mr}\ B((\overline{\pi_0 \circ t})i)^X \\ P(t) & := & \Psi(t)\ \mathbf{mr}\ X. \end{array}$$

Using quantifier free bar induction relativized to S (see section 3.2) we show $P([\,])$.

1. Assume that we have a $\beta \in S$, i.e., $\forall n S(\bar{\beta}n)$. Then we have to show $\exists n P(\bar{\beta}n)$. Let

$$\begin{array}{lcl} g & := & \pi_0 \circ \beta \\ \gamma & := & [G_0]\#(\pi_1 \circ \beta). \end{array}$$

Then

$$\begin{array}{lcl} S(\bar{\beta}n) & \equiv & \forall i \leq n.\ ([G_0]\#(\pi_1 \circ \beta))_i\ \mathbf{mr}\ B((\overline{\pi_0 \circ \beta})i)^X \\ & \equiv & \forall i \leq n.\gamma(i)\ \mathbf{mr}\ B(\bar{g}i)^X. \end{array}$$

Hence

$$\begin{array}{lcl} \forall n S(\bar{\beta}n) & \equiv & \forall n.\gamma(n)\ \mathbf{mr}\ B(\bar{g}n)^X \\ & \equiv & \gamma\ \mathbf{mr}\ \forall n B(\bar{g}n)^X. \end{array}$$

By (3) we obtain $Yg\gamma\ \mathbf{mr}\ X$, hence $\tilde{Y}(\beta)\ \mathbf{mr}\ X$. Furthermore, by the principle of continuity we know that in order to compute $\tilde{Y}$ we only need to look at finitely many values in the sequence β, i.e., there exists an n such that $\tilde{Y}(\beta) = \tilde{Y}(\bar{\beta}n\#\lambda n.\mathsf{any}(n))$ where $\mathsf{any}(n)$ are any terms. In particular, we have

$$\tilde{Y}(\beta) = \Psi_{G,Y}(\bar{\beta}n)$$

and hence end up with $\Psi(\bar{\beta}n)\ \mathbf{mr}\ X$, i.e., $P(\bar{\beta}n)$.

2. Show $\forall t \in S.\forall q(S(t * q) \to P(t * q)) \to P(t)$. Let $t \in S$ where t is of form $t = [\langle x_0, z_0\rangle, \ldots, \langle x_{n-1}, z_{n-1}\rangle]$ and set $xs := [x_0, \ldots, x_{n-1}]$. Now assume $\forall q(S(t{*}q) \to P(t{*}q))$, i.e., for all x_n, z_n

$$(\forall i \leq n+1.([G_0, z_0, \ldots, z_n])_i\ \mathbf{mr}\ B(\overline{[x_0, \ldots, x_n]}i)^X) \to \Psi(t * \langle x_n, z_n\rangle)\ \mathbf{mr}\ X.$$

Set $u := ([G_0, z_0, \ldots, z_{n-1}])_n$, $xs := [x_0, \ldots, x_{n-1}]$ then, in particular, we have

$$u\ \mathbf{mr}\ B(xs)^X \to \lambda x_n, z_n.\Psi(t * \langle x_n, z_n\rangle)\ \mathbf{mr}\ \forall x_n.B(xs * x_n)^X \to X.$$

Now, by (2), applied to xs,u and $v := \lambda x_n, z_n.\Psi(t * \langle x_n, z_n\rangle)$ we obtain

$$G(xs, u, v)\ \mathbf{mr}\ X.$$

By Lemma 3.4, we have a closed term H such that

$$\forall ys.\ H\ \mathbf{mr}\ (X \to B(ys)^X),$$

hence

$$\forall ys.\ H(G(xs,u,v))\ \mathbf{mr}\ B(ys)^X.$$

Now, let

$$\begin{array}{lcl} w & := & H(G(xs,u,v)) \\ g & := & [x_0,\ldots,x_{n-1}]\#\lambda n.[\,] \\ \gamma & := & [G_0,z_0,\ldots,z_{n-1}]\#\lambda n.w. \end{array}$$

We recall:

$$\begin{array}{ll} \forall i \le n.\ \gamma(i)\ \mathbf{mr}\ B(\bar{g}i)^X & [\text{since } t \in S] \\ \forall m > n.\ \gamma(m)\ \mathbf{mr}\ B(\bar{g}m)^X & [\text{since } w\ \mathbf{mr}\ B(\overline{(xs\#\lambda n.[\,])}m)^X] \end{array}$$

i.e.,

$$\gamma\ \mathbf{mr}\ \forall n B(\bar{g}n)^X.$$

Again we have

$$\begin{array}{ll} Yg\gamma\ \mathbf{mr}\ X & [\text{by def } Y \text{ and } \mathbf{mr}\ ] \\ \Psi_{G,Y}(t)\ \mathbf{mr}\ X & [Yg\gamma] = \tilde{Y}(t\#\lambda n.\langle[\,],w\rangle) = \Psi_{G,Y}(t)], \end{array}$$

hence $P(t)$.

3. $S([\,])$ follows from (1).

□

6.2 Formalization of Nash-Williams' proof

We give a formalization of the classical proof of Higman's Lemma presented in section 4.1 where, for simplicity, we restrict ourselves to a two letter alphabet, i.e., the booleans with the equality as quasiorder. We shall use both versions of dependent choice: (DC-seq) for proving the minimal bad sequence argument and (DC) for the lemma stating that every infinite boolean valued sequence has a constant subsequence (cf. section 3.3). Higman's Lemma for an arbritrary alphabet could be proven by using Ramsey's theorem instead of the constant subsequence lemma. The proof of Ramsey's theorem is similar to that of the lemma.

The example has been implemented in the Minlog system. Here, we sketch the formalized classical proof. The full implementation of the example may be found in the Minlog repository.

Types. Beside nat and boole we have word := tsil boole and seq := tsil (tsil boole) where, we recall, $\mathsf{tsil}\,\alpha$ is the type for reverse lists over the type α, i.e.,

$$\mathsf{tsil}\,\alpha = \mathsf{Lin} + \mathsf{Snoc}(\mathsf{Lead} : \mathsf{tsil}\,\alpha, \mathsf{Last} : \alpha)$$

with constructors $\mathsf{Lin} : \mathsf{tsil}\,\alpha$ and $\mathsf{Snoc} : \mathsf{tsil}\,\alpha \to \alpha \to \mathsf{tsil}\,\alpha$ and destructors $\mathsf{Lead} : \mathsf{tsil}\,\alpha \to \mathsf{tsil}\,\alpha$ and $\mathsf{Last} : \mathsf{tsil}\,\alpha \to \alpha$. We usually display Lin as $[\,]$ and $\mathsf{Snoc}(xs, x)$ as $xs * x$ (or as $xs :: x$ in MINLOG).

Variables.

$i, j, k, l, n, m :$	nat	$e :$	$\mathsf{nat} \to \mathsf{nat}$
$a, b, c :$	boole	$h :$	$\mathsf{nat} \to \mathsf{boole}$
$u, v, w, :$	word	$f, f_{\min}, g :$	$\mathsf{nat} \to \mathsf{word}$
$vs, ws :$	seq		

Constants. $\mathsf{Init} : (\mathsf{nat} \to \alpha) \to \mathsf{nat} \to \mathsf{tsil}\,\alpha$, where we abbreviate $\mathsf{Init} f n$ by $\bar{f} n$, as usual, computes an initial segment of length n of a given infinite sequence.

$\mathsf{Isinit} : (\mathsf{nat} \to \alpha) \to \mathsf{tsil}\,\alpha \to \mathsf{boole}$ decides whether a given finite sequence is an initial segment of an infinite sequence, i.e. $\mathsf{Isinit}(f, xs) := (\bar{f}|xs| = xs)$.

There are two relations on words we are interested in: the proper (transitive) initial segment relation $\prec$: word $\to$ word $\to$ boole

$$\begin{array}{lcl} v \prec [\,] & := & \mathsf{false} \\ v \prec w{*}a & := & [\mathsf{if}\ (v = w)\ \mathsf{true}\ (v \prec w)], \end{array}$$

where $=$ is the equality on words, and the Higman embedding, $\leq^*$: word $\to$ word $\to$ boole

$$\begin{array}{lcl} [\,] \leq^* w & := & \mathsf{true} \\ v{*}a \leq^* [\,] & := & \mathsf{false} \\ v{*}a \leq^* w{*}b & := & [\mathsf{if}\ (a = b)\ (v \leq^* w)\ (v{*}a \leq^* w)] \end{array}$$

The axiom of dependent choice. We use both variants of dependent choice:

$$\begin{array}{ll} \mathrm{DC}: & \forall x^\rho \exists^{\mathsf{cl}} y^\rho A(x,y) \to \exists^{\mathsf{cl}} f^{\mathsf{nat}\to\rho}.\ f(0) = x_0^\rho \wedge \forall n A(f(n), f(n+1)) \\ \mathrm{DC\text{-}seq}: & B([\,]) \to (\forall xs^{\rho^*}.B(xs) \to \exists^{\mathsf{cl}} x^\rho.B(xs * x)) \to \exists^{\mathsf{cl}} g^{\mathsf{nat}\to\rho} \forall n.B(\bar{g}n). \end{array}$$

Proposition 6.3 (Higman's Lemma).

$$\forall f^{\mathsf{nat}\to\mathsf{word}} \exists^{\mathsf{cl}} i, j.i < j \wedge f(i) \leq^* f(j).$$

Proof. Assume that there is an infinite bad sequence, i.e., let f be such that $\forall i, j.i < j \to f(i) \leq^* f(j) \to \bot$ and show $\bot$. In a first step, from the existence of a bad sequence we

derive that there is a 'minimal' bad sequence. We use the abbreviation $P(ws)$ to express that ws starts an infinite bad sequence. $B(ws)$ will be useful to express the minimality condition.

$$\begin{aligned} P(ws) &:= \exists^{\mathsf{cl}} f.\mathsf{Isinit}(f, ws) \wedge \forall i, j.i < j \to \neg f(i) \leq^* f(j) \\ B(ws) &:= P(ws) \wedge \forall vs, v.ws = vs * v \to \forall u.u \prec v \to \neg P(vs * u). \end{aligned}$$

CLAIM 1. $\exists^{\mathsf{cl}} f_{\min} \forall n B(\overline{f_{\min}} n)$.

This will be proven using DC-seq for B. Note that B is a relevant formula. We have to show

(a) $B([])$, which is true since by assumption there is a bad sequence, and

(b) $\forall vs.\ B(vs) \to \exists^{\mathsf{cl}} v.\ B(vs * v)$.

Assume vs such that $B(vs)$. Then $\exists^{\mathsf{cl}} v B(vs * v)$ can be proven by the minimum principle

$$\exists^{\mathsf{cl}} v P(vs * v) \to \exists^{\mathsf{cl}} v.\ P(vs * v) \wedge \forall u.\ u \prec v \to \neg P(vs * u).$$

The premise $\exists^{\mathsf{cl}} v P(vs * v)$ follows from $B(vs)$ and the conclusion is exactly what we want. The minimum principle itself is proven inductively. Now, we may assume that we have an $f_{\min}$ with $\forall n B(\overline{f} n)$.

CLAIM 2. $f_{\min}$ is a bad sequence.

Assume i, j such that $i < j$ and $f_{\min}(i) \leq^* f_{\min}(j)$. By claim 1, in particular we have $\forall n P(\overline{f_{\min}} n)$, which applied to $j + 1$ yields that there is a bad sequence f starting with $\overline{f_{\min}}(j + 1)$, thereby contradicting $f_{\min}(i) \leq^* f_{\min}(j)$.

CLAIM 3. $\forall i \neg f_{\min}(i) = []$.

Assume i such that $f_{\min}(i) = []$. Then $\bot$ follows from $P(\overline{f_{\min}}(i{+}1))$.

CLAIM 4. Every boolean valued sequence has a constant subsequence.

$$\forall h^{\mathsf{nat}\to\mathsf{boole}} \exists^{\mathsf{cl}} e^{\mathsf{nat}\to\mathsf{nat}} \forall n, k.\ \neg\neg(e(n) < e(n{+}k{+}1) \wedge h(e(n)) = h(e(n{+}k))).$$

Similar to the example in section 3.3 we first prove

$$\forall h \exists^{\mathsf{cl}} e \forall n.\ \neg\neg(e(n) < e(n{+}1) \wedge h(e(n)) = h(e(n{+}1))),$$

using (DC), and then proceed by induction on k.

From CLAIM 4, instantiated to $\lambda i.\,\mathsf{Last} f_{\min}(i)$ we obtain an index function e such that $\lambda i.\mathsf{Last} f_{\min}(e(i))$ is constant. Now, we show that there is an infinite sequence which is still bad but lexicographically smaller than $f_{\min}$, contradicting the choice of the minimal bad sequence $f_{\min}$. This sequence is computed using the constant G which takes a function $f_{\min}$ and an index function e as input and produces the desired sequence:

$$G(f_{\min}, e) := \lambda n.\ [\mathsf{if}\ (n < e(0))\ \ f_{\min}(n)\ \ \mathsf{Lead}(f_{\min}(e(n - e(0))))].$$

CLAIM 5. $G(f_{\min}, e)$ is bad.

Assume i, j such that $i < j$ and $G(f_{\min}, e)(i) \leq^* G(f_{\min}, e)(j)$. Then the goal $\bot$ follows from the claims 3 and 4 by case distinction on whether or not $i < e(0)$ and $j < e(0)$ from the badness of $f_{\min}$.

CLAIM 6. $G(f_{\min}, e)$ is contradicting the choice of the minimal bad sequence.

Formally, by applying claim 1 to $e(0)+1$, we have $B(\overline{f_{\min}}(e(0)+1))$. From its right hand side we conclude

$$\neg P((\overline{f_{\min}}e(0))*\mathsf{Lead}(f_{\min}(e(0)))).$$

On the other hand, we know $\mathsf{Isinit}(Gf_{\min}e, (\overline{f_{\min}}e(0))*\mathsf{Lead}(f_{\min}(e(0))))$ and hence, by claim 5,

$$P((\overline{f_{\min}}e(0))*\mathsf{Lead}(f_{\min}(e(0)))).$$

□

6.3 Discussion of the extracted program

We briefly want to analyze the program that has been obtained by the MINLOG system via the A-translation method.

Size. The size of the normalized extracted program is about two screen pages. Thus, the program is substantially smaller than the program extracted by Murthy which was about 12MB. Nevertheless, we are not entirely satisfied with the result when it comes to running the program, and some work still needs to be done to understand the program.

Behavior. The normalized program is of the form $\lambda f^{\mathsf{nat}\to\mathsf{word}}.\Psi\,\mathtt{term1}\,\mathtt{term2}[\,]$. So, its behavior essentially depends on Ψ, the realizer of DC-seq, and its animation. Unfortunately, the definition of Ψ, as given in section 6.1, involves multiple computations, since it is of the form $\tilde{Y}(t\#\lambda n.\mathtt{const})$ where the term `const` is computed over and over again. If, in MINLOG, we add the definition of Ψ as a term rewriting rule, indeed, the program only runs for some trivial inputs. In our first attempts the extracted program was much larger and normalization did not even terminate when Ψ was selfevaluating; this, however, was overcome by assigning a realizer directly to DC-seq.

Animation of Minlog constants using Scheme programs. Another attempt to solve these difficulties was to replace the term defining Ψ by an algorithm avoiding the multiple computations via a `set!` construction (higher order memoization). This is only possible on a SCHEME level and not in MINLOG itself, but fortunately MINLOG's built-in normalization mechanism is based on SCHEME evaluation (normalization by evaluation). We provide an implementation of this idea in an appendix to the example. It might be used for other applications as well. The speed up is enormous, when, for instance applied to the example of computing a list of indices such that a given boolean function on this list is constant (see last paragraph in section 3.3).

Further improvements. For further experiments, it would be better to transform the extracted program into a SCHEME program, however such a translation mechansim has not been written up to now. We also have to admit that we have not entirely investigated the defintion of Ψ with respect to the difference between a call-by-value and a call-by-name evalution and its effect on termination.

7 An inductive proof of Kruskal's Theorem

Kruskal's Theorem is a famous theorem in infinitary combinatorics with important applications in term rewriting theory. In this section we give a constructive proof using inductive definitions only. In particular, we use an inductive characterization of a well quasiorder which is formulated via an accessibility definition and has been introduced in section 4.2 and two Lemmas which we have already been proven there. So, Kruskal's Theorem reads as

$$\mathsf{acc}_{\ll_A}[] \to \mathsf{acc}_{\ll_{T(A)}}[].$$

There are already several constructive proofs of Kruskal's Theorem. The proof of Rathjen and Weiermann [RW93] uses a reification into the ordinal $\theta\Omega^\omega$, while the proof of Hasegawa [Has94] is built on a more general concept of algebras, similar to ordinal notations. The combinatorial idea behind both proofs was already proposed by Schmidt [Sch79] and also our proof uses this construction. Thus the connection between the three proofs in [Sch79], [RW93] and our proof is the same as the one outlined for Higman's Lemma in section 4.3. However, whereas the proofs of Schmidt and Rathjen/Weiermann require a thorough knowledge of ordinal notation systems, our proof only uses inductive definitions which are commonly well understood and available in most theorem provers. Of course, also the proof due to Rathjen/Weiermann can be carried out in a theory of inductive definitions: first, use the inductive definitions to prove that $\theta\Omega^\omega$ is well-founded and then Peano arithmetic plus the wellfoundedness of this ordinal to derive Kruskal's Theorem. Particularly the second part is quite involved and depends on the right assignment of the ordinals. Giving a direct proof of Kruskal's Theorem avoids these problems, and could be preferred when the quantitative aspect is not needed: for instance when dealing with problems where the main focus lies on the algorithmic aspect. The proof we give in this chapter has been published in [Sei01b].

The combinatorial idea

It is essential for this constructive proof to introduce an additional structure on trees: we look at trees with one or more label sets where the number of immediate successors of a node is bound by an ordinal assigned to the set where the label of the node comes from.

The combinatorial idea is similar to that of Higman's Lemma described in chapter 4. Again, we determine the space for badly continuing a singleton sequence $[t]$ by means of sums and products of sets which by induction hypothesis can be proven to be well quasiordered. Roughly the idea is as follows: if $t \not\leq u$ and u is a tree with i subtrees, then u can identified with a tree u' with less than i subtrees and a root containing the rest of the subtrees. u' then lies in a set, of which, as said above, we already know that it is well quasiordered. Formally, the identification can be found in Lemma 7.1 (5).

Definition. The set $T(A)$ of finite trees with labels in A is inductively defined by

$$\frac{a \in A \qquad t_1, \ldots, t_n \in T(A)}{a\, t_1 \ldots t_n \in T(A),}$$

where $a\, t_1 \ldots t_n$, $0 \leq n$, is the tree consisting of a root with label a and the subtrees $t_1, \ldots, t_n$. Moreover, we define the embeddability relation $\leq_{T(A)}$ inductively by the following rules:

$$\frac{t \leq_{T(A)} u_j, \text{ for some } j \leq m}{t \leq_{T(A)} a\, u_1 \ldots u_m} \qquad \frac{a \leq_A b \quad [t_1, \ldots, t_n] \leq_{T(A)^*} [u_1, \ldots, u_m]}{a\, t_1 \ldots t_n \leq_{T(A)} b\, u_1 \ldots u_m}.$$

Here $\leq_{T(A)^*}$ refers to the Higman embedding. $\leq_{A^*}$ on A^* is inductively defined by

$$\frac{}{[] \leq_{A*} []} \qquad \frac{as \leq_{A^*} bs}{as \leq_{A^*} bs*b} \qquad \frac{as \leq_{A^*} bs \quad a \leq_A b}{as*a \leq_{A^*} bs*b}.$$

Definition. Given quasi orders $(A_1, \leq_{A_1}), \ldots, (A_n, \leq_{A_n})$ we define disjoint union $\dot\cup\{A_i : i \leq n\}$ and cartesian product $\times\{A_i : i \leq n\}$ via

$$\begin{aligned} a \leq_{\dot\cup A_i} a' &\leftrightarrow a, a' \in A_i \wedge a \leq_{A_i} a' \text{ for some } i \leq n, \\ (a_1, \ldots, a_n) \leq_{\times A_i} (a_1', \ldots, a_n') &\leftrightarrow a_i \leq_{A_i} a_i' \text{ for all } i \leq n \end{aligned}$$

Definition. Let $(A_0, \leq_{A_0}), \ldots, (A_n, \leq_{A_n})$ and $0 < \alpha_0 < \cdots < \alpha_n \leq \omega$ be given. By $T\binom{A_n \ldots A_0}{\alpha_n \ldots \alpha_0}$ we denote the set of all trees with labels in $\dot\cup\{A_i : i \leq n\}$ such that every node with a label in A_i has less than α_i immediate successors. The embeddability relation on $T\binom{A_n \ldots A_0}{\alpha_n \ldots \alpha_0}$ is the restriction of $\leq_{T(\dot\cup\{A_i : i \leq n\})}$ on $T\binom{A_n \ldots A_0}{\alpha_n \ldots \alpha_0}$.

The structure of the proof

Next, we give a rough idea of the structure of the proof, in order to motivate further definitions. We want to show $\mathsf{acc}_{\ll_{T\binom{A}{\omega}}}[]$. Because of Lemma 4.5, it suffices to show $\mathsf{acc}_{\ll_{T\binom{A}{\omega}_{[t]}}}[]$ for an arbitrary tree t. If there is a quasi embedding of $T\binom{A}{\omega}_{[t]}$ into an appropriate tree set $T\binom{B\, A_{[a]}}{n\ \omega}$ and if $\mathsf{acc}_{\ll_{T\binom{B\, A_{[a]}}{n\ \omega}}}[]$ holds, we are done by Lemma 4.6. Now, $\binom{B\, A_{[a]}}{n\ \omega}$ is in a certain sense lexicographically smaller than $\binom{A}{\omega}$. So we will be finished by induction on this order once we have defined it. However, such a definition would require quantification over sets. Since in this definition we only need sets B of a certain structure, we can restrict ourselves to elements of the set of cartesian compositions of A, a set of names denoted $\mathsf{Cart}(A)$, thus avoiding second order quantification. We define a relation $<$ on $\mathsf{Cart}(A)$ such that for all cartesian compositions $\mathcal{X}$ $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\mathcal{X}$ implies

$\mathsf{acc}_{\ll_X}[]$. In a second step we define the relation $<_{\mathsf{Lex}}$ and show that all cartesian compositions $\mathcal{X}$ are in the accessible part of $<_{\mathsf{Cart}(A)}$, i.e. $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\mathcal{X}$ holds. The definition of $\mathsf{Cart}(A)$ as well as the notion $\mathcal{X}^x$ and Lemma 7.1 are due to [RW93]. In [Has94] the set $\mathsf{Cart}(A)$ corresponds, with a small restriction, to the class of algebras.

Cartesian compositions

Definition. We inductively define the set $\mathsf{Cart}(A)$ by the following rules:

1. If as is a bad sequence in A^*, then $\mathcal{A}_{as} \in \mathsf{Cart}(A)$ is a name for A_{as}.
2. If $\mathcal{X}_1, \mathcal{X}_2 \in \mathsf{Cart}(A)$ are names for X_1, X_2,
   then $\mathcal{X}_1 \dot\cup \mathcal{X}_2 \in \mathsf{Cart}(A)$ is a name for $X_1 \dot\cup X_2$.
3. If $\mathcal{X}_1, \mathcal{X}_2 \in \mathsf{Cart}(A)$ are names for X_1, X_2,
   then $\mathcal{X}_1 \times \mathcal{X}_2 \in \mathsf{Cart}(A)$ is a name for $X_1 \times X_2$.
4. If $\mathcal{X} \in \mathsf{Cart}(A)$ is a name for X,
   then $\mathcal{X}^* \in \mathsf{Cart}(A)$ is a name for X^*.
5. Let $\mathcal{X}_0, \ldots, \mathcal{X}_m \in \mathsf{Cart}(A)$ be names for quasi orders $X_0, \ldots, X_m$ and let $0 < \gamma_0 < \cdots < \gamma_m \leq \omega$. Then $\mathcal{T}\binom{\mathcal{X}_m \ldots \mathcal{X}_0}{\gamma_m \ldots \gamma_0}$ is a name for $T\binom{X_m \ldots X_0}{\gamma_m \ldots \gamma_0}$.

Remark 7.1. Every name $\mathcal{X} \in \mathsf{Cart}(A)$ obviously denotes a set X. Observe that $\mathcal{A}_{[]}$ is a name for A. Occasionally we will use the binary operations $\dot\cup$ and $\times$ for finitely many arguments, including the one and zero case. For these cases the definition is to be extended in the obvious way.

Definition (Recursive Definition of $\mathcal{X}^x$ for given $\mathcal{X} \in \mathsf{Cart}(A)$ and $x \in X$).

1. $(\mathcal{A}_{as})^a := \mathcal{A}_{as*a}$.
2. $(\mathcal{X}_1 \dot\cup \mathcal{X}_2)^x := \begin{cases} \mathcal{X}_1{}^x \dot\cup \mathcal{X}_2, & \text{if } x \in X_1, \\ \mathcal{X}_1 \dot\cup \mathcal{X}_2{}^x, & \text{if } x \in X_2. \end{cases}$
3. $(\mathcal{X}_1 \times \mathcal{X}_2)^{(x_1,x_2)} := (\mathcal{X}_1{}^{x_1} \times \mathcal{X}_2) \dot\cup (\mathcal{X}_1 \times \mathcal{X}_2{}^{x_2})$.
4. $(\mathcal{X}^*)^{[x_1,\ldots,x_n]} := \dot\cup\{(\mathcal{X}^{x_1})^* \times \mathcal{X} \times (\mathcal{X}^{x_2})^* \times \cdots \times \mathcal{X} \times (\mathcal{X}^{x_j})^* : j \leq n\}$.
5. Let $t \in T\binom{X_m \ldots X_0}{\gamma_m \ldots \gamma_0}$ be given. Let t consist of the root $x \in \mathcal{X}_i$ and the subtrees $t_1, \ldots, t_n$. We may assume that $\mathcal{T}\binom{\mathcal{X}_m \ldots \mathcal{X}_0}{\gamma_m \ldots \gamma_0}^{t_j}$ is already defined for all $0 \leq j \leq$

n. Set

$$Ł := \mathcal{X}_i\times\ \dot\cup\{(\mathcal{T}(\ldots)^{t_1})^*\times\cdots\times(\mathcal{T}(\ldots)^{t_j})^* : j\leq n\}.$$

If $n=0$, i.e. if t only consists of a root, then

$$\mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots\mathcal{X}_0\\ \gamma_m\ \ldots\gamma_0\end{pmatrix}^t := \mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots{\mathcal{X}_i}^x\ldots\mathcal{X}_0\\ \gamma_m\ldots\ \gamma_i\ \ \ldots\gamma_0\end{pmatrix}.$$

If $n=\gamma_k$ for $k<i$, we define

$$\mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots\mathcal{X}_0\\ \gamma_m\ \ldots\gamma_0\end{pmatrix}^t := \mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots{\mathcal{X}_i}^x\ldots\mathcal{X}_k\dot\cup\mathcal{X}_i\dot\cup Ł\ldots\mathcal{X}_0\\ \gamma_m\ldots\ \gamma_i\ \ \ldots\quad\ \gamma_k\quad\ \ \ldots\gamma_0\end{pmatrix},$$

If $n\neq\gamma_k$ for all $k<i$, then we define

$$\mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots\mathcal{X}_0\\ \gamma_m\ \ldots\gamma_0\end{pmatrix}^t := \mathcal{T}\begin{pmatrix}\mathcal{X}_m\ldots{\mathcal{X}_i}^x\ldots\mathcal{X}_i\dot\cup Ł\ldots\mathcal{X}_0\\ \gamma_m\ldots\ \gamma_i\ \ \ldots\quad n\quad\ \ldots\gamma_0\end{pmatrix},$$

where the column with $\mathcal{X}_i\dot\cup Ł$ and n has to be filled in at the appropriate place.

Definition. On $\mathsf{Cart}(A)$ we define a relation $<_{\mathsf{Cart}(A)}$ via

$$\mathcal{Y}<_{\mathsf{Cart}(A)}\mathcal{X}\leftrightarrow\exists x\in X\,\mathcal{Y}=\mathcal{X}^x.$$

Lemma 7.1. *Let $\mathcal{X}\in\mathsf{Cart}(A), x\in X$. Then there is a quasi embedding $e\colon X_{[x]}\to X^x$.*

Proof. $\mathsf{Ind}(\mathsf{Cart}(A))$.

1. Let $a\in A_{as}$. Then, because of $(A_{as})_{[a]}\subseteq(A_{as})^a$, the identity is a quasi embedding from $(A_{as})_{[a]}$ to $(A_{as})^a$.

2. Let $x\in X_1\dot\cup X_2$. W.l.o.g. $x\in X_1$. By ih there is a quasi embedding $e_{X_1,x}\colon X_{1[x]}\to {X_1}^x$. We define

$$\begin{aligned}e\colon\quad & (X_1\dot\cup X_2)_{[x]}\to {X_1}^x\dot\cup X_2\\ & y\mapsto\begin{cases}e_{X_1,x}(y), & \text{if } y\in X_{1[x]},\\ y, & \text{if } y\in X_2.\end{cases}\end{aligned}$$

3. Let $(x_1,x_2)\in X_1\times X_2$. By ih, we already have quasi embeddings $e_{X_i,x_i}\colon X_{i[x_i]}\to {X_i}^{x_i}$ for $x_i\in X_i,\ i\in\{1,2\}$. Set

$$\begin{aligned}e\colon\quad & (X_1\times X_2)_{[(x_1,x_2)]}\to ({X_1}^{x_1}\times X_2)\,\dot\cup\,(X_1\times {X_2}^{x_2})\\ & (y_1,y_2)\mapsto\begin{cases}(e_{X_1,x_1}(y_1),y_2), & \text{if } y_1\in X_{1[x_1]},\\ (y_1,e_{X_2,x_2}(y_2)), & \text{otherwise.}\end{cases}\end{aligned}$$

4. Let $[x_1, \ldots, x_n] \in X^*$. By ih we already have quasi embeddings $e_{X,x_i} \colon X_{[x_i]} \to X^{x_i}$, for all $1 \leq i \leq n$. We look for a quasi embedding

$$e \colon X^*{}_{[[x_1,\ldots,x_n]]} \to \dot\cup\{(X^{x_1})^* \times X \times (X^{x_2})^* \times X \times \cdots \times (X^{x_j})^* \colon j \leq n\}.$$

If $n = 0$ then the quasi embedding is the empty quasi embedding. Otherwise, let $[y_1, \ldots, y_m] \in X^*{}_{[[x_1,\ldots,x_n]]}$. Since $[x_1, \ldots, x_n] \not\leq_{X^*} [y_1, \ldots, y_m]$ there exists $0 \leq l < n$ such that $[x_1, \ldots, x_l] \leq_{X^*} [y_1, \ldots, y_m]$ holds, but $[x_1, \ldots, x_{l+1}] \leq_{X^*} [y_1, \ldots, y_m]$ does not. We choose $j_1 < \cdots < j_l$ minimal such that $x_i \leq y_{j_i}$, for all $i \leq l$. Then we have $[y_1, \ldots, y_{j_1-1}] \in (X_{[x_1]})^*, \ldots, [y_{j_l+1}, \ldots, y_m] \in (X_{[x_{l+1}]})^*$, and by induction hypothesis we may define

$$\begin{aligned} ws_1 &:= [e_{X,x_1}(y_1), \ldots, e_{X,x_1}(y_{j_1-1})] \in (X^{x_1})^* \\ &\ldots \\ ws_{l+1} &:= [e_{X,x_{l+1}}(y_{j_l+1}), \ldots, e_{X,x_{l+1}}(y_m)] \in (X^{x_{l+1}})^* \end{aligned}$$

and set $e([y_1, \ldots, y_m]) := (ws_1, y_{j_1}, ws_2, y_{j_2}, \ldots, y_{j_l}, ws_{l+1})$.

5. Let $t \in T(\begin{smallmatrix} X_m & \ldots & X_0 \\ \gamma_m & \ldots & \gamma_0 \end{smallmatrix})$ be given. Ind(structure of t). Let us first of all consider the case of t consisting only of a root with a label $x \in X_i$. Let $u \in T(\begin{smallmatrix} X_m & \ldots & X_0 \\ \gamma_m & \ldots & \gamma_0 \end{smallmatrix})_{[t]}$. Then for every node in u with a label $y \in X_i$ it holds $x \not\leq_{X_i} y$, i.e. u lies in $T\left(\begin{smallmatrix} X_m \ldots X_{i[x]} \ldots X_0 \\ \gamma_m \ldots \quad \gamma_i \quad \ldots \gamma_0 \end{smallmatrix}\right)$. A quasi embedding into $T\left(\begin{smallmatrix} X_m \ldots X_i{}^x \ldots X_0 \\ \gamma_m \ldots \quad \gamma_i \quad \ldots \gamma_0 \end{smallmatrix}\right)$ can easily be constructed by ih, applied to to X_i and x.
Now, assume t consists of a root $x \in X_i$ and the immediate subtrees $t_1, \ldots, t_n$ with $n = \gamma_k < \gamma_i$ (The case $n \neq \gamma_k$, for all $k < i$ is analogous). By induction hypothesis, for all $j \leq n$ there exist quasi embeddings

$$e_{t_j} \colon T\left(\begin{smallmatrix} X_m \ldots X_0 \\ \gamma_m \ \ldots \ \gamma_0 \end{smallmatrix}\right)_{[t_j]} \to T\left(\begin{smallmatrix} X_m \ldots X_0 \\ \gamma_m \ \ldots \ \gamma_0 \end{smallmatrix}\right)^{t_j}.$$

We look for a quasi embedding

$$e \colon T\left(\begin{smallmatrix} X_m \ldots X_0 \\ \gamma_m \ \ldots \ \gamma_0 \end{smallmatrix}\right)_{[t]} \to T\left(\begin{smallmatrix} X_m \ldots X_i{}^x \ldots X_k \dot\cup X_i \dot\cup L \ldots X_0 \\ \gamma_m \ \ldots \ \gamma_i \ \ldots \quad \gamma_k \quad \ldots \gamma_0 \end{smallmatrix}\right).$$

Let $u \in T\left(\begin{smallmatrix} X_m \ldots X_0 \\ \gamma_m \ \ldots \ \gamma_0 \end{smallmatrix}\right)_{[t]}$ be a tree with root y and immediate subtrees $u_1, \ldots, u_r$. We may assume that the quasi embeddings on the subtrees are already defined, and have to map the root label y suitably to a label in $X_0, \ldots, X_k \dot\cup X_i \dot\cup L, \ldots, X_i{}^x, \ldots,$ or X_m such that the condition concerning the number of immediate subtrees is fulfilled.

$t \not\leq_{T(\ldots)} u$ only can hold for one of the following reasons:

i. $y \notin X_i$. Then map y either to itself or, in the case $y \in X_k$, to y in the first component of $X_k \dot\cup X_i \dot\cup L$. We set $e(y\,u_1 \dots u_r) := y\,e(u_1) \dots e(u_r)$.

ii. $y \in X_i$, but $x \not\leq y$. Hence, we have $y \in X_{i[x]}$ and by induction hypothesis $e_{X_i,x}(y) \in {X_i}^x$. We set

$$e(y\,u_1 \dots u_r) := e_{X_i,x}(y)\,e(u_1) \dots e(u_r).$$

iii. $y \in X_i$ and $x \leq y$, but y has less than γ_k immediate successors. Then anyhow we may map y to itself, if we regard y as a label in the second component of $X_k \dot\cup X_i \dot\cup L$. We set $e(y\,u_1 \dots u_r) := y\,e(u_1) \dots e(u_r)$.

iv. $y \in X_i$, $x \leq y$ and y has more than γ_k successors, but $[t_1, \dots, t_{\gamma_k}] \not\leq_{T(\dots)^*} [u_1, \dots, u_r]$. Then there exists an l, $0 \leq l < \gamma_k$ such that $[t_1, \dots, t_l] \leq_{T(\dots)^*} [u_1, \dots, u_r]$ and $[t_1, \dots, t_{l+1}] \not\leq_{T(\dots)^*} [u_1, \dots, u_r]$. We map u to a tree with $l < \gamma_k$ subtrees and a root label in L, where this label contains the images of the remaining subtrees. More formally: Choose $j_1 < \dots < j_l$ minimal such that $t_1 \leq_{T(\dots)} u_{j_1}, \dots, t_l \leq_{T(\dots)} u_{j_l}$. Then it holds

$$\begin{aligned}
&[u_1, \dots, u_{j_1-1}] \in T\binom{X_m \dots X_0}{\gamma_m \dots \gamma_0}_{[t_1]}^{\,*}\\
&\dots\\
&[u_{j_l+1}, \dots, u_r] \in T\binom{X_m \dots X_0}{\gamma_m \dots \gamma_0}_{[t_{l+1}]}^{\,*}
\end{aligned}$$

and by ih it follows

$$\begin{aligned}
&ts_1 := [e_{t_1}(u_1), \dots, e_{t_1}(u_{j_1-1})] \in T\binom{X_m \dots X_0}{\gamma_m \dots \gamma_0}^{t_1\,*}\\
&\dots\\
&ts_{l+1} := [e_{t_{l+1}}(u_{j_l+1}), \dots, e_{t_{l+1}}(u_r)] \in T\binom{X_m \dots X_0}{\gamma_m \dots \gamma_0}^{t_{l+1}\,*}
\end{aligned}$$

Finally we put

$$e(y\,u_1 \dots u_r) := (\,y,\,(ts_1, \dots, ts_{l+1}))\,e(u_{j_1}) \dots e(u_{j_l}).$$

It is left to the reader to check that e is actually a quasi embedding. □

Lemma 7.2. $\forall \mathcal{X} \in \mathsf{Cart}(A).\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X} \to \mathsf{acc}_{\ll_X} []$.

Proof. $\mathsf{Ind}(\mathsf{acc}_{<_{\mathsf{Cart}(A)}})$. Let $\mathcal{X} \in \mathsf{Cart}(A)$. Assume $\mathsf{ih}\colon \forall \mathcal{Y} <_{\mathsf{Cart}(A)} \mathcal{X}\, \mathsf{acc}_{\ll_Y} []$. Because of Lemma 4.5 it suffices to show $\forall x \in X\, \mathsf{acc}_{\ll_{X_{[x]}}} []$. Let $x \in X$. By Lemma 7.1 there exists a quasi embedding $e\colon X_{[x]} \to X^x$. Therefore, using Lemma 4.6 we only have to show $\mathsf{acc}_{\ll_{X^x}} []$. But this follows by ih. □

Definition. Given the set $\mathsf{Lex}:=$

$$\left\{ \binom{\mathcal{X}_n \ldots \mathcal{X}_1}{\gamma_n \ldots \gamma_1},\ \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_i,\ i \leq n,\ 0 \leq n < \omega,\ 0 < \gamma_1 < \cdots < \gamma_n \leq \omega \right\},$$

we define a relation $<_{\mathsf{Lex}}$: $\binom{\mathcal{Y}_m \ldots \mathcal{Y}_1}{\delta_m \ldots \delta_1} <_{\mathsf{Lex}} \binom{\mathcal{X}_n \ldots \mathcal{X}_1}{\gamma_n \ldots \gamma_1} \iff$

$m, n > 0$ and

$\delta_m = \gamma_n \ \wedge\ \mathcal{Y}_m <_{\mathsf{Cart}(A)} \mathcal{X}_n$ or

$\delta_m = \gamma_n \ \wedge\ \mathcal{Y}_m = \mathcal{X}_n \ \wedge\ \binom{\mathcal{Y}_{m-1} \ldots \mathcal{Y}_1}{\delta_{m-1} \ldots \delta_1} <_{\mathsf{Lex}} \binom{\mathcal{X}_{n-1} \ldots \mathcal{X}_1}{\gamma_{n-1} \ldots \gamma_1}$.

Lemma 7.3. *Assume* $\mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X}_0, \ldots, \mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X}_n$ *and let* $0 < \gamma_0 < \cdots < \gamma_n \leq \omega$. *Then it follows that* $\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{X}_n \ldots \mathcal{X}_0}{\gamma_n \ldots \gamma_0}$.

Proof. We show

$$\forall \gamma \leq \omega.\, \forall \mathcal{X}.\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X} \to \forall \mathbb{X}.\, \mathsf{acc}_{<_{\mathsf{Lex}}} \mathbb{X} \to$$
$$\forall n, \mathcal{X}_1, \ldots, \mathcal{X}_n, \gamma_1, \ldots, \gamma_n.\, \gamma_1 < \cdots < \gamma_n < \gamma \wedge \mathbb{X} = \binom{\mathcal{X}_n \ldots \mathcal{X}_1}{\gamma_n \ldots \gamma_1} \to$$
$$\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{X}\, \mathcal{X}_n \ldots \mathcal{X}_1}{\gamma\ \gamma_n \ldots \gamma_1}$$

by $\mathsf{Ind}_1(\gamma)$, $\mathsf{Ind}_2(\mathsf{acc}_{<_{\mathsf{Cart}(A)}})$ and $\mathsf{Ind}_3(\mathsf{acc}_{<_{\mathsf{Lex}}})$. We are done once we have shown $\mathsf{acc}_{<_{\mathsf{Lex}}} \mathbb{Y}$ for all $\mathbb{Y} <_{\mathsf{Lex}} \binom{\mathcal{X}\, \mathcal{X}_n \ldots \mathcal{X}_1}{\gamma\ \gamma_n \ldots \gamma_1}$.

1. Let $\mathbb{Y} = \binom{\mathcal{X}^x\, \mathcal{Y}_m \ldots \mathcal{Y}_1}{\gamma\ \ \delta_m \ldots \delta_1}$ such that $x \in X$, $\delta_1 < \cdots < \delta_m < \gamma$ and $\forall i \leq m\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{Y}_i$. By an m-fold application of ih_1 we obtain $\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{Y}_m \ldots \mathcal{Y}_1}{\delta_k \ \ldots \delta_1}$ and by ih_2 $\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{X}^x\, \mathcal{Y}_m \ldots \mathcal{Y}_1}{\gamma\ \ \delta_m \ldots \delta_1}$.

2. Let $\mathbb{Y} = \binom{\mathcal{X}\, \mathcal{Y}_m \ldots \mathcal{Y}_1}{\gamma\ \delta_m \ldots \delta_1}$ such that $\binom{\mathcal{Y}_m \ldots \mathcal{Y}_1}{\delta_m \ \ldots \delta_1} <_{\mathsf{Lex}} \binom{\mathcal{X}_n \ldots \mathcal{X}_1}{\gamma_n \ldots \gamma_1}$. Then, by ih_3, we obtain $\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{X}\, \mathcal{Y}_m \ldots \mathcal{Y}_1}{\gamma\ \delta_m \ldots \delta_1}$.

The Lemma follows from $\mathsf{acc}_{<_{\mathsf{Lex}}}()$ by $n+1$ applications of this assertion. □

Lemma 7.4. *Assume* $\mathsf{acc}_{\ll_A}[]$. *Then* $\forall \mathcal{X}.\mathcal{X} \in \mathsf{Cart}(A) \to \mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X}$.

Proof. $\mathsf{Ind}(\mathsf{Cart}(A))$.

1. We show $\forall as \in \mathrm{Bad}(A).\, \mathsf{acc}_{\ll_A}\, as \to \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{A}_{as}.\ \mathsf{Ind}(\mathsf{acc}_{\ll_A})$. Let $as \in \mathrm{Bad}(A)$ and assume ih: $\forall bs \ll_A as\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{A}_{bs}$. We need to prove $\forall a \in A_{as}\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, (\mathcal{A}_{as})^a$. But this follows by ih because of $(\mathcal{A}_{as})^a = \mathcal{A}_{as*a}$ and $as*a \ll_A as$.

2. Let $\mathcal{X}_1$ and $\mathcal{X}_2$ s.t. $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1$ and $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_2$. In order to show $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1 \dot\cup \mathcal{X}_2$ use induction on $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1$ and $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_2$. Let $x \in \mathcal{X}_1 \dot\cup \mathcal{X}_2$, w.l.o.g. $x \in X_1$, then the induction hypothesis implies $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1{}^{x_1} \dot\cup \mathcal{X}_2$.

3. Let $\mathcal{X}_1$ and $\mathcal{X}_2$ s.t. $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1$ and $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_2$. Now we show $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1 \times \mathcal{X}_2$ by induction on $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1$ and $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_2$. We have

$$\begin{aligned} &\mathsf{ih}_1\colon \forall x_1 \in X_1, \forall \mathcal{X}_2.\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_2 \to \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1^{x_1} \times \mathcal{X}_2 \\ &\mathsf{ih}_2\colon \forall x_2 \in X_2\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_1 \times \mathcal{X}_2^{x_2} \end{aligned}$$

   Now, let $(x_1, x_2) \in \mathcal{X}_1 \times \mathcal{X}_2$. $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, (\mathcal{X}_1^{x_1} \times \mathcal{X}_2) \dot\cup (\mathcal{X}_1 \times \mathcal{X}_2^{x_2})$ follows by ih_1, applied to x_1 and $\mathcal{X}_2$, ih_2 and 2.

4. Let $\mathcal{X}$ such that $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}$ be given. We prove $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}^*$ by induction on $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}$. Assume ih: $\forall x \in \mathcal{X}\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, (\mathcal{X}^x)^*$. We have to show $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}^*$, i.e. $\forall w \in X^*\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, (\mathcal{X}^*)^w$. Let $w = [x_1, \ldots, x_n]$. Now by ih, 2., and 3., we end up with $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \dot\cup \{(\mathcal{X}^{x_1})^* \times \mathcal{X} \times \cdots \times \mathcal{X} \times (\mathcal{X}^{x_j})^* \colon j \le n\}$.

5. Let $\mathcal{X}_0, \ldots, \mathcal{X}_m$ such that $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_i$ for all $i \le m$ and $0 < \gamma_0 < \cdots < \gamma_m \le \omega$. Since, by Lemma 7.3, $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_0, \ldots, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{X}_m$ implies $\mathsf{acc}_{<_{\mathsf{Lex}}} \binom{\mathcal{X}_m \ldots \mathcal{X}_0}{\gamma_m \ldots \gamma_0}$, we get $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{T}\binom{\mathcal{X}_m \ldots \mathcal{X}_0}{\gamma_m \ldots \gamma_0}$ once we have shown more generally

$$\forall \mathbb{X} \neq ().\, \mathsf{acc}_{<_{\mathsf{Lex}}}\, \mathbb{X} \to \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{T}(\mathbb{X}).$$

   $\mathsf{Ind}_1(\mathsf{acc}_{<_{\mathsf{Lex}}})$. Fix $\mathbb{X}$ and assume $\mathsf{ih}_1 : \forall \mathbb{Y} <_{\mathsf{Lex}} \mathbb{X}.\, \mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{T}(\mathbb{Y})$. Suppose $\mathbb{X}$ to be of the form $\binom{\mathcal{X}_r \ldots \mathcal{X}_0}{\gamma_r \ldots \gamma_0}$ show $\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{T}(\mathbb{X})^t$ for an arbitrary $t \in T(\mathbb{X})$ by

   Ind_2 (structure of t). Assume first that t is a branch with a label $x \in X_i$. Since $\binom{\mathcal{X}_r \ldots \mathcal{X}_i{}^x \ldots \mathcal{X}_0}{\gamma_r \ldots\ \gamma_i\ \ldots \gamma_0} <_{\mathsf{Lex}} \mathbb{X}$, by ih_1, we may conclude

$$\mathsf{acc}_{<_{\mathsf{Cart}(A)}}\, \mathcal{T}\binom{\mathcal{X}_r \ldots \mathcal{X}_i{}^x \ldots \mathcal{X}_0}{\gamma_r \ldots\ \gamma_i\ \ldots \gamma_0}.$$

Now, assume that t consists of the root $x \in \mathcal{X}_i$ and the subtrees $t_1, \ldots, t_n$, $n = \gamma_k < \gamma_i$. (The case $n \neq \gamma_k$ for all $k < i$ is analogous.) By ih_2 it follows $\mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{T}\binom{\mathcal{X}_r \ldots \mathcal{X}_0}{\gamma_r \ldots \gamma_0}^{t_j}$, for all $j \leq n$. By 2., 3., and 4, we are able to conclude

$$\mathsf{acc}_{<_{\mathsf{Cart}(A)}} (\mathcal{X}_i \times \dot{\cup}\{(\mathcal{T}(\mathbb{X})^{t_1})^* \times \cdots \times (\mathcal{T}(\mathbb{X})^{t_j})^* : j \leq n\})$$

Using the abbreviation

$$Ł = \mathcal{X}_i \times \dot{\cup}\{(\mathcal{T}(\mathbb{X})^{t_1})^* \times \cdots \times (\mathcal{T}(\mathbb{X})^{t_j})^* : j \leq n\}$$

we get $\mathsf{acc}_{<_{\mathsf{Cart}(A)}} Ł$ and $\mathsf{acc}_{<_{\mathsf{Cart}(A)}} (\mathcal{X}_k \dot{\cup} \mathcal{X}_i \dot{\cup} Ł)$ by a further application of 2. Therefore we have

$$\mathcal{T}\binom{\mathcal{X}_r \ldots \mathcal{X}_i^x \ldots \mathcal{X}_k \dot{\cup} \mathcal{X}_i \dot{\cup} Ł \ldots \mathcal{X}_0}{\gamma_r \ldots \gamma_i \ldots \gamma_k \ldots \gamma_0} <_{\mathsf{Lex}} \mathcal{T}\binom{\mathcal{X}_r \ldots \mathcal{X}_i \ldots \mathcal{X}_k \ldots \mathcal{X}_0}{\gamma_r \ldots \gamma_i \ldots \gamma_k \ldots \gamma_0}$$

and by ih_1 we obtain

$$\mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{T}\binom{\mathcal{X}_r \ldots \mathcal{X}_i^x \ldots \mathcal{X}_k \dot{\cup} \mathcal{X}_i \dot{\cup} Ł \ldots \mathcal{X}_0}{\gamma_r \ldots \gamma_i \ldots \gamma_k \ldots \gamma_0}.$$

□

Proposition 7.2 (Kruskal's tree Theorem).

$$\mathsf{acc}_{\ll_A} [] \to \mathsf{acc}_{\ll_{T(\mathcal{A}_\omega)}} [].$$

Proof. We want to show $\mathsf{acc}_{\ll_A} [] \to \mathsf{acc}_{\ll_{T(\mathcal{A}_\omega)}} []$. Assume $\mathsf{acc}_{\ll_A} []$. For all $\mathcal{X} \in \mathsf{Cart}(A)$ we have $\mathsf{acc}_{<_{\mathsf{Cart}(A)}} \mathcal{X}$ by Lemma 7.4 and therefore $\mathsf{acc}_{\ll_X} []$ by Lemma 7.2. This holds especially for $\mathcal{X} = \mathcal{T}(\mathcal{A}_{\omega[]})$ and since $A_{[]} = A$ we may conclude $\mathsf{acc}_{\ll_{T(\mathcal{A}_\omega)}} []$. □

Remark 7.3. 1. A constructive proof of Kruskal's Theorem in the style of chapter 5 would be highly desirable, firstly, with respect to the extracted program (compare Remark 5.3), and secondly, with respect to further generalizations: Because of its analogy to the classical proof, which easily can be extended to Kruskal's Theorem with gap condition, this approach could also lead to a constructive proof of the latter.

2. Another extension would be a proof of Kruskal's Theorem not requiring decidability of the given relation on the set of labels. Veldman [Vel00] has given such a proof for Higman's Lemma and Kruskal's Theorem using intuitionistic methods. Fridlender [Fri97] has shown how to transform Veldman's proof of Higman's Lemma into proof in a theory using inductive definitions. Similarly could be proceeded with Kruskal's Theorem.

8 Conclusion and further work

The aims achieved in this thesis are threefold:

- We have extended program extraction from constructive and classical proofs - with respect to inductive definitions and choice principles.
- We have presented new constructive proofs of Higman's Lemma and Kruskal's theorem. Moreover, in the case of Higman's Lemma, we have given two approaches to answer the question of what is the constructive content behind the Nash-Williams proof.
- As a practical part, we have shown that these methods as well as the proofs can be carried out in a theorem prover.

While developing our techniques, we saw that there are still problems left for future work and we recognized several related topics to which our techniques are applicable as well. The following addresses some of them.

A-translation using inductive definitions

Inductive definitions and A-translation have played an important role in this thesis. A first question that arises when we think of combining these two methods is: (1) Are inductive definitions allowed in proofs to be A-translated? With respect to inductive definitions without computational content the answer is 'yes', as we have shown in section 3.1. Regarding inductive definitions with computational content, problems occur, e.g., when we have to double negate the predicate in order to obtain definite and goal formulas; since the predicate also has to be double negated in its closure axiom, this would lead to a non-strictly-positive inductive definition.

Computational content of the minimal bad sequence argument

By means of Higman's Lemma, we have shown two ways of making proofs using a minimal-bad-sequence argument constructive. The general question is: (2) Is there a constructive counterpart to the minimal-bad-sequence argument? An approach in this direction is the so-called open induction principle with its classical formulation due to Raoult [Rao88] on the one hand, and its inductive formulation [Coq92] on the other. The classical formulation can be justified by a minimal-bad-sequence argument, the constructive one amounts to a simple inductive proof. Examples of reformulations using the open induction principle have been given in [CP98, CP99, Per99].) The resulting proofs of the reformulations are inductive proofs similar to those presented in this thesis. However, the ideal would be a mechanical transformation of proofs using the minimal bad sequence argument into constructive proofs.

The second approach, given in chapter 6, via the A- translation method is based on adding a realizer for the axiom of dependent choice. An even more direct approach might exist if we, here presented as a third problem, (3), could assign a realizer directly to the minimal-bad-sequence argument.

Better quasiorderings

Higman's Lemma and Kruskal's Theorem may be generalized to infinite structures, that is, infinite words, infinite trees respectively [NW68, NW65]. The formulation of the theorems for infinite structures needs the more general concept of a better quasiordering instead of a well quasiordering. In the proofs the minimal-bad-sequence argument is replaced by the so-called minimal-bad-array argument which has the same proof theoretic strength as the minimal-bad-sequence argument; namely $|\Pi^1_1 - \mathsf{CA}_0|$ [Mar96]. Related questions and problems are: (4) Is it possible to give an inductive characterization of a better quasiordering? (5) Is it possible to give a realizer for the minimal-bad-array argument? (6) The strength of the generalization of Higman's Lemma to infinite words, also called Generalized Higman's theorem, is still unknown [Mar94]. A lower bound is $|\mathsf{ATR}_0|$, an upper bound $|\Pi^1_1 - \mathsf{CA}_0|$ (since its classical proof involves the minimal-bad-array argument). As in the case of Higman's Lemma, a constructive proof might give more insight into this problem.

Applications

The 'classical' field for applications of the theorems studied in this thesis is term rewriting theory; here we refer to [Wei94, Tou97, Lep01]. Among the manyfold algorithmic applications we want to mention one where in particular our inductive proof of Higman's Lemma might be useful. In [Oga01], Ogawa investigated the problem of deciding whether a disjunctive monadic query is satisfiable in a monadic database. Whereas a naive solution leads to an exponential algorithm, a linear algorithm can be found with the help of a constructive proof of Higman's Lemma as was shown by means of Murthy/Russell's proof [MR90]. (7) It would be interesting to see whether our proof gives rise to a different or even better algorithm.

Kruskal's Theorem

We have put the chapter on Kruskal's Theorem at the end since we believe that the main challenges are there. The first question is: (8) what would a proof of Kruskal's Theorem in the spirit of chapter 5 look like? Unfortunately there is no straightforward generalization of the proof for Higman's Lemma to Kruskal's Theorem. However, in analogy to chapter 6, it is possible, (9), to apply the A-translation method to Kruskal's Theorem. A first step towards this will be a generalization of chapter 6 to an arbitrary

alphabet.

Kruskal's Theorem has a famous extension due to Friedman: Kruskal's Theorem with gap condition, also called the Extended Kruskal Theorem, which classically is proven using a minimal bad sequence argument [Sim85, Kri89]. (10) No constructive proof is known up to now. We claim that our methods of translating and directly extracting a program are applicable here as well. The result would be a constructive proof of the Extended Kruskal Theorem which, according to the theory given in chapter 3, depends on relativized quantifier free bar induction.

Towards the Graph Minor Theorem

The extended Kruskal Theorem may be seen as the main step towards the Graph Minor Theorem [RS99]. Friedman, Robertson and Seymour [FRS87] have shown that the Extended Kruskal Theorem implies the Bounded Graph Minor Theorem and vice versa. A constructive proof of the Graph Minor Theorem would be highly desirable and also bring more insight into its algorithmic aspects (see, for instance, Fellows and Langston, [FL88], Robertson and Seymour, Graph Minor II, XIII [RS86, RS99]).

Last but not least, also the Graph Minor Theorem has an infinite counterpart. The proof of a (restricted) infinite version of the Graph Minor Theorem [Tho89] involves the finite Graph Minor Theorem on the one hand, and the Extended Kruskal Theorem for infinite trees on the other. So the circle Higman's Lemma, Extended Kruskal Theorem, Graph Minor Theorem and Higman's Lemma, Higman's Lemma on infinite sequences, Extended Kruskal Theorem on infinite trees, as well as this thesis closes here.

References

[ABHS03] Klaus Aehlig, Ulrich Berger, Martin Hofmann, and Helmut Schwichtenberg. An arithmetic for non-size-increasing polynomial time computation, 2003. Accepted for publication in *Theoretical Computer Science*.

[Acz77] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90, pages 739–782, Amsterdam, 1977. North-Holland.

[AH02] Klaus Aehlig and Schwichtenberg Helmut. A syntactical analysis of non-size-increasing polynomial time computation. *ACM Transactions on Computational Logic*, 3(3):383–401, 2002.

[BBC+97] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicael Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, Catherine Parent, Christine Paulin-Mohring, Amokrane Saibi, and Benjamin Werner. The Coq proof assistant reference manual : Version 6.1. Technical Report RT-0203, Inria, France, 1997.

[BBC98] Stefano Berardi, Marc Bezem, and Thierry Coquand. On the computational content of the axiom of choice. *The Journal of Symbolic Logic*, 63(2):600–622, 1998.

[BBS+98] Holger Benl, Ulrich Berger, Helmut Schwichtenberg, Monika Seisenberger, and Wolfgang Zuber. Proof theory at work: Program development in the Minlog system. In W. Bibel and P.H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume II: Systems and Implementation Techniques of *Applied Logic Series*, pages 41–71. Kluwer Academic Publishers, Dordrecht, 1998.

[BBS02] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

[BC85] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, 1985.

[Bee85] Michael J. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*, volume 6 of *Ergebnisse der Mathematik und ihrer Grenzgebiete, 3. Folge*. Springer-Verlag, Berlin, 1985.

[Ben98] Holger Benl. Konstruktive Interpretation induktiver Definitionen. Master's thesis, Mathematisches Institut der Universität München, 1998.

[Ber93] Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.

[Ber95] Ulrich Berger. A constructive interpretation of positive inductive definitions. Unpublished draft, 1995.

[Ber03] Stefan Berghofer. Program Extraction in simply-typed Higher Order Logic. In *Types for Proofs and Programs, TYPES 2002*, volume 2624 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, 2003.

[BFPS81] Wilfried Buchholz, Solomon Feferman, Wolfram Pohlers, and Wilfried Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof–Theoretical Studies*, volume 897 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1981.

[BO03] Ulrich Berger and Paulo Oliva. Modified Barrecursion and Classical Dependent Choice. 2003. To appear in Lecture Notes in Logic, 19 pages.

[BS95a] Ulrich Berger and Helmut Schwichtenberg. Program development by proof transformation. In H. Schwichtenberg, editor, *Proof and Computation*, volume 139 of *Series F: Computer and Systems Sciences*, pages 1–45. Springer Verlag, Berlin, Heidelberg, New York, 1995.

[BS95b] Ulrich Berger and Helmut Schwichtenberg. Program Extraction from Classical Proofs. In D. Leivant, editor, *Logic and Computational Complexity, LCC '94*, volume 960 of *Lecture Notes in Computer Science*, pages 77–97. Springer Verlag, Berlin, Heidelberg, New York, 1995.

[BS96] Ulrich Berger and Helmut Schwichtenberg. The greatest common divisor: a case study for program extraction from classical proofs. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs, TYPES '95*, volume 1158 of *Lecture Notes in Computer Science*, pages 36–46. Springer Verlag, Berlin, Heidelberg, New York, 1996.

[BSS01] Ulrich Berger, Helmut Schwichtenberg, and Monika Seisenberger. The Warshall Algorithm and Dickson's Lemma: Two Examples of Realistic Program Extraction. *Journal of Automated Reasoning*, 26:205–221, 2001.

[Buc95] Wilfried Buchholz. Proof–theoretic analysis of termination proofs. *Annals of Pure and Applied Logic*, 75:57–65, 1995.

[Bus98] Samuel R. Buss. *Handbook of Proof Theory*. North–Holland, Amsterdam, 1998.

[CAB+86] Robert L. Constable, Stuart F. Allen, H. M. Bromley, Walter R. Cleaveland, J. F. Cremer, Robert W. Harper, Douglas J. Howe, Ted B. Knoblock, Nax P. Mendler, Prakash Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice–Hall, New Jersey, 1986.

[CF94] Thierry Coquand and Daniel Fridlender. A proof of Higman's lemma by structural induction, 1994. Unpublished Manuscript, ftp://ftp.cs.chalmers.se/pub/users/coquand/open1.ps.Z.

[CM91] Robert Constable and Chetan Murthy. Finding computational content in classical proofs. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 341–362. Cambridge University Press, 1991.

[Coq92] Thierry Coquand. Constructive topology and combinatorics. In *Constructivity in Computer Science*, volume 613 of *Lecture Notes in Computer Science*, pages 28–32, 1992.

[CP98] Thierry Coquand and Hendrik Persson. A Proof-Theoretical Investigation of Zantema's Problem. In M. Nielson and W. Thomas, editors, *Selected papers from CSL'97*, volume 1414 of *Lecture Notes in Computer Science*, pages 177–188. Springer Verlag, Berlin, Heidelberg, New York, 1998.

[CP99] Thierry Coquand and Hendrik Persson. Gröbner Bases in Type Theory. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Types for Proofs and Programs*, volume 1657 of *Lecture Notes in Computer Science*, pages 33–46. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[Cro00] Laura Crosilla. *Realizability Models for Constructive Set Theories with Restricted Induction Principles.* PhD thesis, University of Leeds, 2000.

[Cro02] Laura Crosilla. A tutorial for Minlog, Version 4.0, 2002. University of Munich, `http://www.minlog-system.de`.

[CTB94] Eugeniusz A. Cichon and Elias Tahhan Bittar. Ordinal recursive bounds for Higman's theorem. *Theoretical Computer Science*, 201:63–84, 1994.

[dJP77] Dick de Jongh and Rohit Parikh. Well partial orderings and their order types. *Indagationes Mathematicae*, 39:195–207, 1977.

[Dra79] Albert Dragalin. New kinds of realizability. In *Abstracts of the 6th International Congress of Logic, Methodology and Philosophy of Sciences*, pages 20–24, Hannover, Germany, 1979.

[FFKD87] Matthias Felleisen, Daniel P. Friedman, E. Kohlbecker, and B.F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52:205–237, 1987.

[FL88] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *Journal of the ACM, JACM*, 35(3):727–739, 1988.

[Fri93] Daniel Fridlender. Ramsey's Theorem in Type Theory, 1993. Licentiate Thesis, Chalmers University of Technology and University of Göteburg.

[Fri97] Daniel Fridlender. *Higman's Lemma in Type Theory*. PhD thesis, Chalmers University of Technology and University of Göteburg, 1997.

[FRS87] Harvey Friedman, Neil Robertson, and Paul Seymour. The metamathematics of the graph minor theorem. In *Logic and Combinatorics*, volume 65 of *AMS-series Contemporary Mathematics*, pages 229–261, 1987.

[Gir87] Jean-Yves Girard. *Proof theory and complexity*. Bibliopolis, Naples, 1987.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts. *Dialectica*, 12:280–287, 1958.

[Has94] Ryu Hasegawa. Well–Ordering of Algebra and Kruskal's Theorem. In N. D. Jones, M. Hagiya, and M. Sato, editors, *Logic, Language and Computation. Festschrift in Honor of Satoru Takasu*, volume 792 of *Lecture Notes in Computer Science*, pages 133–172. Springer, 1994.

[Hay90] Susumu Hayashi. An introduction to PX. In G. Huet, editor, *Logical Foundations of Functional Programming*, pages 432–486. Addison–Wesley, Reading, 1990.

[Her94] Hugo Herbelin. A program from an A-translated impredicative proof of Higman's Lemma. http://coq.inria.fr/contribs/higman.html, 1994.

[Hig52] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7):326–336, 1952.

[Hof99] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proceedings 14'th Symposium on Logic in Computer Science (LICS'99)*, pages 464–473, 1999.

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

[Kle60] Stephen Kleene. Realizability: A retrospective survey. In *Proceedings of the Cambridge Summer School in Mathematics*, volume 37 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1960.

[KO03] Ulrich Kohlenbach and Paulo Oliva. Proof mining: A systematic way of analyzing proofs in mathematics. *Proceedings of the Steklov Institute of Mathematics*, 242:136–164, 2003.

[Koh90] Ulrich Kohlenbach. *Theorie der majorisierbaren und stetigen Funktionale und ihre Auswirkungen bei der Extraktion von Schranken aus inkonstruktiven Beweisen: Effektive Eindeutigkeitsmodule bei besten Approximationen aus ineffektiven Eindeutigkeitsbeweisen.* PhD thesis, Frankfurt, 1990.

[KP90] Jean-Louis Krivine and Michel Parigot. Programming with proofs. *Journal of Information Processing and Cybernetics*, 26(3):149–167, 1990.

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

[Kri89] Igor Kriz. Well-quasiordering finite trees with gap condition. *Annals of Mathematics*, 130:215–226, 1989.

[Kru60] Joseph B. Kruskal. Well–quasi–orderings, the tree theorem and Vaszonyi's conjecture. *Transactions American Mathematical Society*, 95:210–255, 1960.

[Lei85] Daniel Leivant. Syntactic translations and provably recursive functions. *The Journal of Symbolic Logic*, 50(3):682–688, 1985.

[Lep01] Ingo Lepper. *Simplification Orders in Term Rewriting.* PhD thesis, Münster, 2001.

[Mar94] Alberto Marcone. Foundations of bqo theory. *Transactions American Mathematical Society*, 345:641–660, 1994.

[Mar96] Alberto Marcone. On the logical strength of Nash-Williams' theorem on transfinite sequences. In W. Hodges, M. Hyland, C. Steinhorn, and J. Truss, editors, *Logic: from Foundations to Applications; European logic colloquium*, pages 327–351. 1996.

[Mat98] Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types.* PhD thesis, Fachbereich Mathematik, Ludwig-Maximilians-Universität München, 1998.

[McC84] David C. McCarty. *Realizability and Recursive Mathematics.* PhD thesis, Oxford, 1984.

[MP02] Favio E. Miranda-Perea. A Curry-Style Realizability Interpretation for Monotone Inductive Definitions. In M. Nissim, editor, *Proc. of 7th ESSLLI Student Session, ESSLLI 2002*, pages 155–166. 2002.

[MR90] Chetan R. Murthy and James R. Russell. A Constructive proof of Higman's Lemma. In *Proceedings of the Fifth Symposium on Logic in Computer Science*, pages 257–267, 1990.

[Mur90] Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs.* Technical report, Ithaca, New York, 1990. PhD thesis.

[Mur91] Chetan R. Murthy. An evaluation semantics for classical proofs. In *Proc. of 6th Annual IEEE Symp. on Logic in Computer Science, LICS'91*, pages 96–107. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[Mur93] Chetan R. Murthy. Finding the answers in classical proofs: a unifying framework. In G. Huet and G. Plotkin, editors, *Logical Environments*, pages 247–272. Cambridge Universtiy Press, Cambridge, 1993.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[NW63] Crispin St. J. A. Nash-Williams. On well–quasi–ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59:833–835, 1963.

[NW65] Crispin St. J. A. Nash-Williams. On well–quasi–ordering infinite trees. *Proceedings of the Cambridge Philosophical Society*, 61:697–720, 1965.

[NW68] Crispin St. J. A. Nash-Williams. On better–quasi–ordering transfinite sequences. *Proceedings of the Cambridge Philosophical Society*, 64:273–290, 1968.

[Oga01] Mizuhito Ogawa. Generation of a linear time query processing algorithm based on well-quasi-orders. *Lecture Notes in Computer Science*, 2215:283–297, 2001.

[Par92] Michel Parigot. Recursive programming with proofs. *Theoretical Computer Science*, 94:335–356, 1992.

[Per99] Henrik Persson. *Type Theory and the Integrated Logic of Programs.* PhD thesis, Chalmers University of Technology and University of Göteburg, Sweden, 1999.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[PM89] Christine Paulin-Mohring. *Extraction de programmes dans le Calcul des Constructions.* PhD thesis, Université Paris, 1989.

[PMW93] Christine Paulin-Mohring and Benjamin Werner. Synthesis of ML programs in the system Coq. *Journal of Symbolic Computation*, 15(5–6):607–640, 1993.

[Ram30] Frank Plumpton Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 30:264–286, 1930.

[Rao88] Jean-Claude Raoult. Proving open properties by induction. *Information Processing Letters*, 29:19–23, 1988.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors II. *Journal of Algorithms*, 7(3):309–322, 1986.

[RS93] Fred Richman and Gabriel Stolzenberg. Well Quasi–Ordered sets. *Advances in Mathematics*, 97:145–153, 1993.

[RS99] Neil Robertson and Paul D. Seymour. Graph minors I, III-XVII. *Journal of Combinatorial Theory, Series B*, 1983-1999.

[RW93] Michael Rathjen and Andreas Weiermann. Proof–theoretic investigations on Kruskal's theorem. *Annals of Pure and Applied Logic*, 60:49–88, 1993.

[Sch79] Diana Schmidt. Well–orderings and their maximal order types, 1979. Habilitationsschrift, Mathematisches Institut der Universität Heidelberg.

[Sch03] Helmut Schwichtenberg. Minimal logic for computatable functionals. Unpublished manuscript. `http://www.mathematik.uni-muenchen.de/~schwicht`, 2003.

[Sei98] Monika Seisenberger. Konstruktive Aspekte von Higman's Lemma. Master's thesis, Mathematisches Institut der Ludwig–Maximilians–Universität München, 1998.

[Sei01a] Monika Seisenberger. An Inductive Version of Nash-Williams' Minimal-Bad-Sequence Argument for Higman's Lemma. In *Types for Proofs and Programs*, volume 2277 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, 2001.

[Sei01b] Monika Seisenberger. Kruskal's tree theorem in a constructive theory of inductive definitions. In *Reuniting the Antipodes – Constructive and Nonstandard Views of the Continuum*, volume 306 of *Synthese Library*. Kluwer Academic Publishers, Dordrecht, 2001.

[Sim85] Stephen G. Simpson. Nonprovability of certain combinatorial properties of finite trees. In L.A. Harrington, M.D. Morley, A. Scedrov, and S.G. Simpson, editors, *Harvey Friedman's Research on the Foundations of Mathematics*, pages 87–117. North–Holland, Amsterdam, 1985.

[Spe62] Clifford Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathmatics. In F. D. E. Dekker, editor, *Recursive function theory*, pages 1–27. North–Holland, Amsterdam, 1962.

[SS85] Kurt Schütte and Stephen G. Simpson. Ein in der reinen Zahlentheorie unbeweisbarer Satz über endliche Folgen von natürlichen Zahlen. *Archiv für Mathematische Logik und Grundlagenforschung*, 25:75–89, 1985.

[Tho89] Robin Thomas. Well-quasiordering infinite graphs with forbidden finite planar minor. *Transactions American Mathematical Society*, 312(1), 1989.

[Tou97] Hélène Touzet. *Propriétés combinatoires pour la terminaison de systèmes de réécriture*. PhD thesis, Université Henri Poincaré, Nancy, 1997.

[Tou02] Hélène Touzet. A characterisation of multiply recursive functions with Higman's lemma. *Information and Computation*, 178:534–544, 2002.

[Tro73] Anne S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1973.

[Tro98] Anne S. Troelstra. *Realizability*, pages 407–473. In [Bus98], 1998.

[TvD88] Anne S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics. An Introduction*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North–Holland, Amsterdam, 1988.

[Vel00] Wim Veldman. An intuitionistic proof of Kruskal's theorem, 2000. Report no. 0017, Department of Mathematics, University of Nijmegen.

[Wei94] Andreas Weiermann. Complexity bounds for some finite forms of Kruskal's theorem. *Journal of Symbolic Computation*, 18(5):463–488, 1994.

A Implementations in the MINLOG system

In the appendix we present the implementations of some of the examples discussed in the thesis. The MINLOG system [BBS+98] is available via the page `www.minlog-system.de`. There also some documentation and articles on the system can be found. For an explanation of the commands used in the demo files we refer to the MINLOG tutorial [Cro02].

In all examples we use the datatypes of natural numbers `nat` and reverse lists `tsil` which can be loaded to the system via

```
(mload "../lib/nat2.scm")
(mload "../lib/tsil.scm")
```

The latter contains the type of reverse lists with the constructors: Lin and Snoc. (Snoc as a) is displayed as as::a. In the first example we, in addition, need to load the A-translation module

```
(mload "../modules/atr.scm")
```

Comment lines starting with ';' as well as *emphasised* expressions are added to provide a better understanding and more structure.

A.1 A-translation using an external realizer for dependent choice

```
; dc-first.scm, demofile for section 3.3
; This is an example for program extraction from classical proofs
; using the axiom of dependent choice.

 1. Declarations

(av "a" "b" "c" (py "boole"))
(av "i" "j" (py "nat"))
(av "e" (py "nat=>nat"))
(av "h" (py "nat=>boole"))
(add-global-assumption "<-lemma" (pf "all n,m. n < m + n + 1"))
(add-global-assumption "=-lemma" (pf "all a,b,c. a=b -> c=b -> a=c"))

(define only-two
        (pf "all a. (a = F -> bot) -> (a = T -> bot) -> bot"))
(define dc-inst
        (pf " all h. (all n excl m. (n<m & (h m) = F -> bot) -> bot) ->
                      excl e. e 0 = 0 &
                              all k. (e k< e(k+1) &
                                     h(e(k+1)) = F -> bot) -> bot"))
```

2. Interactive proofs

```
; Using DC we prove the lemma saying that every function h:nat->boole
; has an infinite constant subsequence.

(set-goal (mk-imp dc-inst only-two
          (pf " all h. excl a,e. all k. (e k < e (k+1) &
                                         h (e (k+1)) = a->bot)->bot")))
(assume "dc-inst" "only-two" "h" 3)
(use-with "dc-inst" (pt "h") "?" "?")

; all n excl m.(n<m & h m=False -> bot) -> bot from
(assume "n" 4)
(use-with 3 (pt "True") (pt "lambda m . m+n") "?")
(assume "k")
(ng)
(strip)
(drop "dc-inst" 3)
(use-with "only-two" (pt "h(Succ(k+n))") "?" "?")
(assume 6)
(use-with 4 (pt "Succ(k+n)") "?")
(strip)
(use 7)
(split)
(use "<-lemma")
(use 6)
(strip)
(use 5)
(split)
(use "Truth-Axiom")
(use 6)

; all e.e 0=0 &
        (all k.(e k<e(k+1) & h(e(k+1))=False -> bot) -> bot) -> bot
(assume "e" 4)
(use-with 3 (pt "F") (pt "e") "?")
(use 4)
(save "Constantsubsequence")

; We now prove the corollary: all h excl i,j. i<j ! h i = h j

(set-goal (mk-all (pv "h")
                  (mk-imp dc-inst only-two
                          (pf "excl i,j. i < j ! h(i) = h(j)"))))
(assume  "h" "dc-inst" "only-two" 3)
```

```
(use-with "Constantsubsequence" "dc-inst" "only-two" (pt "h") "?")
(assume "a" "e" 4)
(use-with 4 (pt "0") "?")
(strip)
(use-with 4 (pt "1") "?")
(strip)
(use 3 (pt "e 1")(pt "e 2"))
; e 1<e 2
(use 6)
; h (e 1)=h (e 2)
(use "=-lemma" (pt "a"))
(use 5)
(use 6)
(define min-excl-proof (np (expand-theorems (current-proof))))


 3. Realizing classical dependent choice

; The following definitions and notations refer to section 3.2
(define rho (py "nat"))
(define nu (py "nat@@nat"))
(define sigma (mk-arrow nu nu))
(define type-of-G (mk-arrow rho (mk-arrow rho sigma nu) nu))
(define type-of-Y (mk-arrow (mk-arrow (py "nat") rho)
                            (mk-arrow (py "nat") sigma)
                            nu))
(av "x" rho)
(av "z" sigma)
(av "G" type-of-G)
(av "Y" type-of-Y)
(define type-of-tsil (py "tsil (nat@@(nat@@nat=>nat@@nat))"))
(av "t" type-of-tsil)

; [G,Y]Psi G Y Lin is a realizer for DC^X
(add-program-constant "Psi"
                      (mk-arrow type-of-G type-of-Y type-of-tsil nu))


 4. A-translation

(define program (atr-min-excl-proof-to-structured-extracted-term
                 min-excl-proof
                 (pt "[h,G,Y]Psi G Y (Lin (nat@@(nat@@nat=>nat@@nat)))")))

; For a better display we add some additional variable names:
```

```
(av "f" (py "nat=>(nat@@nat=>nat@@nat)=>nat@@nat"))
(av "g" (py "nat=>nat@@nat=>nat@@nat"))
(av "p"(py "nat@@nat"))

(define nprogram (nt program))
(term-to-expr nprogram)

(lambda (h0)
 ((((|Psi|
     (lambda (n1)
       ("if" ((h0 (|Succ| n1)) "=" |True|)
             ("if" ((h0 (|Succ| (|Succ| n1))) "=" |True|)
                   (lambda (f2) ((|Succ| n1) "@" (|Succ| (|Succ| n1))))
                   ("if" ((h0 (|Succ| (|Succ| n1))) "=" |False|)
                         (lambda (f2)
                           ((f2 (|Succ| (|Succ| n1))) (lambda (p3) p3)))
                         (lambda (f2) ("0" "@" "0"))))
             ("if" ((h0 (|Succ| n1)) "=" |False|)
                   (lambda (f2) ((f2 (|Succ| n1)) (lambda (p3) p3)))
                   (lambda (f2) ("0" "@" "0"))))))
   (lambda (e1)
     (lambda (g2)
       ((g2 "0") ((g2 "1") ((e1 "1") "@" (e1 "2")))))))
  |(Lin nat@@(nat@@nat=>nat@@nat))|))
```

5. Animation of Psi

```
; in tsil.scm: Lh: tsil alpha -> nat and
; Proj: tsil alpha -> nat ->alpha, displayed, infix, as __.
; projection: if t=[t_0,...,t_n-1] (t__i) yields t_i

(define type-of-beta (mk-arrow (py " nat") (make-star rho sigma)))
(av "beta" type-of-beta)

(define type-of-Tilde (mk-arrow type-of-Y type-of-beta nu))
(add-program-constant "Tilde" type-of-Tilde)

(add-computation-rule (pt "Tilde Y beta")
                      (pt "Y
                           ([n][if (n=0) 0 (left (beta (Pred n)))])
                           ([n] right (beta n))"))

; H is a realizer for an instance of the efq-axiom,
; here the instance is bot -> (bot -> bot)
```

```
(add-program-constant "H" (mk-arrow nu nu nu))
(add-computation-rule (pt "H p1 p2")(pt "p1"))

; Finally, we add the computation-rule for Psi

(add-computation-rule (pt "Psi G Y t")
                      (pt "Tilde
                           Y
                           ([n][if (n < (Lh t))
                                   (t__n)
                                   (0@
                                     H (G
                                         [if (Lh t = 0)
                                             0
                                             (left (t__(Pred (Lh t))))]
                                         ([x,z] (Psi G Y (t::(x@z))))))])"))
```

6. Running the extracted program

```
(add-program-constant "Constsequence" (py "nat=>boole"))
(add-rewrite-rule (pt "Constsequence n")(pt "F"))
(term-to-string
   (nt (mk-term-in-app-form nprogram (pt "Constsequence"))))
; ==> "1@2"

; Note that the 0th element is not found
; the reason being h circ e is only constant for inputs n>0.

(add-program-constant "Interesting" (py "nat=>boole"))
(add-computation-rule (pt "Interesting 0")(pt "F"))
(add-computation-rule (pt "Interesting 1")(pt "T"))
(add-computation-rule (pt "Interesting 2")(pt "F"))
(add-computation-rule (pt "Interesting 3")(pt "T"))
(add-computation-rule (pt "Interesting 4")(pt "T"))
(term-to-string
   (nt (mk-term-in-app-form nprogram (pt "Interesting"))))
; ==> "3@4"
```

A.2 Inductive Definitions, an example

We implement the inductive predicate `Bar` and demonstrate the use of the introduction and elimination axioms by means of two lemmas which will be both needed for the proof of Higman's Lemma.

a) Bar [] implies that every infinite sequence has a good initial segment.

b) Bar $ws * []$.

1. Definitions

```
(define nat (py "nat"))
(define word (py "tsil nat"))
(define seq (py "tsil (tsil nat)"))

(av "a" "b" "c" "i" "j" (py "nat"))
(av "w" "u" "v" "x" "y" "z" "as" "bs" word)
(av "ws" "vs" "xs" "ys" "zs" seq)
(av "f" (make-arrow nat word))

(add-program-constant "Init" (mk-arrow (mk-arrow nat word) nat seq) 1)
(add-computation-rule (pt "Init f 0") (pt "(Lin (tsil nat))"))
(add-computation-rule (pt "Init f (Succ n)") (pt "(Init f n)::(f(n))"))

; Emb, L, Good are inductive definitions without computational content
; L vs v corresponds to Good(vs,v) in the main text (section 5.1).

(add-ids (list (list "Emb" (make-arity word word)))
         '("Emb (Lin nat) (Lin nat)" )
         '("allnc v,w,a. Emb v w -> Emb v (w::a)")
         '("allnc v,w,a. Emb v w -> Emb (v::a) (w::a)"))

(add-ids (list (list "L" (make-arity seq word)))
         '("allnc vs,v,w. Emb v w -> L (vs::v) w")
         '("allnc vs,v,w. L vs w  -> L (vs::v) w"))

(add-ids (list (list "Good" (make-arity seq)))
         '("allnc ws,w. L ws w -> Good (ws::w)")
         '("allnc ws,w. Good ws -> Good (ws::w)"))

; Bar is an inductive predicate with computatinal content.
; The 'type' of Bar is the tree with the constructors Leaf and Branch.
(add-ids (list (list "Bar" (make-arity seq) "tree"))
         '("allnc ws. Good ws -> Bar ws" "Leaf")
         '("allnc ws. (all w  Bar (ws::w)) -> Bar ws" "Branch"))
```

2. The interactive proof (a)

```
(set-goal (pf "allnc ws. Bar ws ->
                         all f,n. Init f n = ws ->
                                  ex m. Good (Init f m)"))
```

```
(assume "vs")

;Ind(Bar).
(elim)

; 1. Good ws
(assume "ws" "Good ws"  "f" "n" "Init f n=ws")
(ex-intro (pt "n"))
(simp "Init f n=ws")
(use "Good ws")

; 2. all w Bar(ws::w)
(assume "ws" "all w Bar(ws::w)" "ih" "f" "n" "Init f n=ws")
(use-with "ih" (pt "f  n ")(pt "f") (pt "n+1") "?")

;Init f(n+1)=(ws::f n)
(ng)
(use "Init f n=ws")
(save "Bar-thm")
```

3. The extracted program

```
(av "ga" (py "tsil nat=>tree"))
(av "gb" (py "tsil nat=>(nat=>tsil nat)=>nat=>nat"))
(term-to-expr (nt (proof-to-extracted-term (current-proof))))

((|(Rec tree=>(nat=>tsil nat)=>nat=>nat)|
    (lambda (f3) (lambda (n4) n4)))
  (lambda (ga3)
    (lambda (gb4)
      (lambda (f5)
        (lambda (n6) (((gb4 (f5 n6)) f5) (|Succ| n6)))))))
```

4. We give a second example to demonstrate the use of the introduction axioms

```
; (b) Bar (ws*[])
(set-goal (pf "all w. Emb (Lin nat) w"))
(ind)
(intro 0)
(assume "w" "a" 1)
(intro 1)
(use 1)
(save "Emb-lemma")
```

```
(set-goal (pf "allnc ws Bar (ws::(Lin nat))"))
(assume "ws")
(intro 1)
(assume "w")
(intro 0)
(intro 0)
(intro 0)
(use "Emb-lemma")
(save "Prop1")

(term-to-expr (proof-to-extracted-term (current-proof)))
; ==> (|Branch| (lambda (w) |Leaf|))
```

A final comment

```
; We could have reformulated our first statement:
;    allnc ws. Bar ws -> all f. Isinit f ws -> ex m. Good (Init f m)"))

; where  Isinit f [] = True
;        Isinit f (ws*w) = [if (f (Lh ws)=w)(Isinit f ws) False]

; However, this would lead to problems concerning the CV-variables
; since, in case 'Good ws',  we have to set m = Lh ws, but
; ws is a CV variable which must not be used in the interactive proof.

; In general, having a proof of Bar ws does not imply that we have given
; the object ws.
```

A.3 Higman's Lemma for a 0/1 alphabet

In this section, we present a MINLOG-formalization of Coquand and Fridlender's proof of Higman's Lemma [CF94]. We prove that every infinite sequence in a 0/1 alphabet has a good initial segment.

1. Definitions

```
; loading the definitions of the preceeding example
(mload "../examples/bar/bar.scm")

(aga "only-two-letters" (pf "all a,b,c.(a=b -> F) -> (c=a -> F) -> c=b"))

(add-ids (list (list "R" (make-arity nat seq seq)))
         '("R a (Lin (tsil nat)) (Lin (tsil nat))")
         '("allnc vs,ws,w,a. R a vs ws -> R a (vs::w) (ws::(w::a))"))
```

```
(add-ids (list (list "TT" (make-arity nat seq seq)))
         '("allnc ws,zs,w,a,b. (a=b -> F) -> R b ws zs ->
                              TT a (zs::w) (zs::(w::a))")
         '("allnc ws,zs,w,a. TT a ws zs ->
                             TT a (ws::w) (zs::(w::a))")
         '("allnc ws,zs,w,a. (a=b -> F) -> TT a ws zs ->
                             TT a ws (zs::(w::b))"))

(aga "lemma2nc" (pf "allnc ws,zs,a. R a ws zs -> Good ws -> Good zs"))
(aga "lemma3nc" (pf "allnc ws,zs,a. TT a ws zs -> Good ws -> Good zs"))
(aga "lemma4nc" (pf "allnc ws,zs,a.(ws=(Lin (tsil nat)) -> F) ->
                       R a ws zs -> TT a ws zs "))
```

2. Interactive proofs

```
; Prop1 has been proven in bar.scm

; Prop2

(set-goal  (pf "allnc xs. Bar xs ->
                allnc ys. Bar ys ->
                all zs,a,b. (a=b -> F) -> TT a xs zs  -> TT b ys zs ->
                Bar zs"))
(assume "xs1")
(elim)

; 1. Good xs
(assume "xs" "Good xs" "ys" "Bar ys" "zs" "a" "b" "a=b -> F"
        "TT a xs zs" "TT b ys zs")
(intro 0)
(use-with "lemma3nc" (pt "xs") (pt "zs") (pt "a") "TT a xs zs" "Good xs")

; 2. all w Bar(xs::w)
(assume "xs" "all w Bar(xs::w)" "ih1" "ys1")
(elim)

; 2.1
(assume "ys" "Good ys"  "zs" "a" "b" "a=b -> F" "TT a xs zs" "TT b ws zs")
(intro 0)
(use-with "lemma3nc" (pt "ys") (pt "zs") (pt "b") "TT b ws zs" "Good ys")

; 2.2
(assume "ys" "all w Bar(ys::w)" "ih2" "zs" "a" "b"
        "a=b -> F" "TT a xs zs" "TT b ws zs")
(intro 1)
```

```
; structural induction on w
(ind)

; 2.2.1
(use "Prop1")

; 2.2.2
(assume "z" "c" "Bar(zs::z)")

(cases (pt "c=a"))
(assume "c=a")
(simp "c=a")
(use "ih1" (pt "z") (pt "ys") (pt "a") (pt"b"))
; Bar ys
(intro 1)
(use "all w Bar(ys::w)")
; a=b -> F
(use "a=b -> F")
; TT a(xs::z) (zs::z::a) from
(intro 1)
(use "TT a xs zs")
(intro 2)
(assume "b=a")
(use "a=b -> F")
(simp "b=a")
(prop)
(use "TT b ws zs")

; false
(assume "c=a -> F")
(cut (pf "c=b"))
(assume "c=b")
(use-with "ih2" (pt "z") (pt "zs::z::c") (pt "a") (pt "c") "?" "?" "?")
(assume "a=c")
(use "c=a -> F")
(simp "a=c")
(ng)
(use "Truth-Axiom")
(simp "c=b")
(intro 2)
(use "a=b -> F")
(use "TT a xs zs")
(simp "c=b")
(intro 1)
(use "TT b ws zs")
```

```
(use "only-two-letters" (pt "a"))
(use "a=b -> F")
(use "c=a -> F")
(save  "Prop2")

; The extracted program from Prop2

(av "gc" (py "tsil nat=>tsil(tsil nat)=>tree"))
(av "gd" (py "tsil nat=>tree=>tsil(tsil nat)=>nat=>nat=>tree"))
(av "ge" (py "tsil nat=>tsil(tsil nat)=>nat=>nat=>tree"))

(term-to-expr (nt (proof-to-extracted-term (current-proof))))

((|(Rec tree=>tree=>tsil(tsil nat)=>nat=>nat=>tree)|
   (lambda (tree5)
     (lambda (ws6) (lambda (a7) (lambda (a8) |Leaf|)))))
 (lambda (ga5)
   (lambda (gd6)
     ((|(Rec tree=>tsil(tsil nat)=>nat=>nat=>tree)|
        (lambda (ws11) (lambda (a12) (lambda (a13) |Leaf|))))
      (lambda (ga11)
        (lambda (ge12)
          (lambda (ws13)
            (lambda (a14)
              (lambda (a15)
                (|Branch|
                  ((|(Rec tsil nat=>tree)| |cPropOne|)
                   (lambda (w17)
                     (lambda (a18)
                       (lambda (tree19)
                         ("if" (a18 "=" a14)
                               (((((gd6 w17) (|Branch| ga11))
                                  (ws13 "::" (w17 "::" a14)))
                                 a14)
                                a15)
                               ((((ge12 w17) (ws13 "::" (w17 "::" a18)))
                                 a14)
                                a18)))))))))))))))

; Prop3

(set-goal  (pf "all a. allnc xs. Bar xs -> (xs = (Lin (tsil nat)) -> F) ->
                           all zs. R a xs zs -> Bar zs"))
(assume "a" "xs1")
(elim)
```

```
; base: all ws.good ws -> formula[a,ws]
(assume "xs" "Good xs" "xs ne Lin" "zs" "R a xs zs")
(intro 0)
(use-with "lemma2nc" (pt "xs") (pt "zs") (pt "a") "R a xs zs" "Good xs")

; step
(assume "xs"  "all w Bar xs::w" "ih"  "xs ne Lin" "zs" "R a xs zs")
(intro 1)
(ind)
(use "Prop1")
(assume "z" "c" "Bar zs::z")
(cases (pt "c=a"))
(assume "c=a")
(use-with "ih" (pt "z") "?" (pt "zs::z::c") "?")
(ng)
(prop)

; R a(xs::z)(zs::z::c) from
(simp "c=a")
(intro 1)
(use "R a xs zs")

; (c=a -> F) -> Bar(zs::z::c) from
(assume "c=a -> F")
(cut (pf "a=c -> F"))
(assume "a=c -> F")
(use-with "Prop2"  (pt "xs") "?"
                   (pt "zs::z") "Bar zs::z"
                   (pt "zs::z::c")(pt "a") (pt "c") "?" "?" "?")
; Bar xs
(intro 1)
(use "all w Bar xs::w")

; a=c -> F
(use "a=c -> F")

; TT a xs(zs::z::c)
(intro 2)
(use "a=c -> F")

; TT a xs zs from
(use "lemma4nc" )
(use "xs ne Lin")
(use "R a xs zs")
```

```
; TT c(zs::z)(zs::z::c) from
(intro 0 (pt "xs")(pt "a"))
(use "c=a -> F")
(use "R a xs zs")

; a=c -> F from
(assume "a=c")
(use "c=a -> F")
(simp "a=c")
(ng)
(use "Truth-Axiom")
(save "Prop3")

; The extracted program from Prop3

(term-to-expr (nt (proof-to-extracted-term (current-proof))))

(lambda (a0)
  ((|(Rec tree=>tsil(tsil nat)=>tree)| (lambda (ws3) |Leaf|))
   (lambda (ga3)
     (lambda (gc4)
       (lambda (ws5)
         (|Branch|
           ((|(Rec tsil nat=>tree)| |cPropOne|)
            (lambda (w7)
              (lambda (a8)
                (lambda (tree9)
                  ("if" (a8 "=" a0)
                        ((gc4 w7) (ws5 "::" (w7 "::" a8)))
                        (((((|cPropTwo| (|Branch| ga3)) tree9)
                           (ws5 "::" (w7 "::" a8)))
                          a0)
                         a8)))))))))))

; The proof of the Theorem

(set-goal (pf "Bar (Lin (tsil nat))"))
(intro 1)

(ind)
;1.
(use "Prop1")
;2.
(assume "w"  "a" 1)
(use-with "Prop3" (pt "a") (pt ":w") 1 "?" (pt ":(w::a)") "?")
```

```
(ng)
(prop)

; R a(:w)(:(w::a))
(intro 1)
(intro 0)
(save "Thm")
(term-to-expr (nt (proof-to-extracted-term (current-proof))))

(|Branch|
  ((|(Rec tsil nat=>tree)| |cPropOne|)
   (lambda (w1)
     (lambda (a2)
       (lambda (tree3)
         ((((|cPropThree| a2) tree3) (":" (w1 "::" a2)))))))))

(set-goal (pf " all f  ex m. Good (Init f m)"))
(assume "f")
(use-with "Bar-thm" (pt "(Lin (tsil nat))") "Thm" (pt "f")(pt "0") "?")
; Init f 0=(Lin tsil nat)
(ng)
(use "Truth-Axiom")

(define program (proof-to-extracted-term (current-proof)))
(animate "Bar-thm")
(animate "Thm")
(animate "Prop1")
(animate "Prop2")
(animate "Prop3")
(term-to-expr program)

(define nprogram (nt program))
(term-to-string nprogram)
```

3. Test of the extracted program

```
(define (run-higman infinite-sequence)
  (dt (nt (mk-term-in-app-form nprogram infinite-sequence))))

; a. [0 0], [1 1 0], [0 1 0 1], [0], ...

(apc "Seq" (mk-arrow (py "nat")(py "(tsil nat)")) 1)
(add-rewrite-rule (pt "Seq 0")(pt ":0::0"))
(add-rewrite-rule (pt "Seq 1")(pt "(:1::1)::0"))
```

```
(add-rewrite-rule (pt "Seq 2")(pt "((:0::1)::0)::1"))
(add-rewrite-rule (pt "Seq (++(++(++ n)))")(pt ":0"))
(run-higman (pt "Seq"))
; ==> 3
; i.e., the subsequence consisting of the first three words is good

; b. [0 0], [1], [1 0], [], ...

(apc "Interesting" (mk-arrow (py "nat")(py "(tsil nat)")) 1)
(add-rewrite-rule (pt "Interesting 0")(pt ":0::0"))
(add-rewrite-rule (pt "Interesting 1")(pt ":1"))
(add-rewrite-rule (pt "Interesting 2")(pt ":1::0"))
(add-rewrite-rule (pt "Interesting (++(++(++ n)))")(pt "(Lin nat)"))
(run-higman (pt "Interesting"))
; ==> 5
; This is an example in which not the shortest good initial segment
; is found.
```

A.4 Higman's Lemma for a finite alphabet

We prove Lemma 5.8 i), ii) and Propostition 5.9.

```
(mload "../examples/bar/bar.scm")
(animate "Prop1")
(animate "Bar-thm")

 1. Inductive definition bar on letters (use lower case letters)

(apc "ll" (py "(tsil nat)=>nat=>boole")  1)
; note that we have introduced ll (corresponds to Good(.,.) in the main
; text, see section 5.1) as program constant (and not as inductive
; predicate) since it should be decidable.

(add-computation-rule (pt "ll(Lin nat) b")(pt "F"))
(add-computation-rule (pt "ll (as::a) b")
                      (pt "[if (a=b) T (ll as b)]"))

(add-ids (list (list "good" (make-arity word)))
         '("allnc as,a. ll as a -> good (as::a)")
         '("allnc as,a. good as -> good (as::a)"))

(add-ids (list (list "bar" (make-arity word) "lltree"))
         '("allnc as. good as -> bar as" "leaf")
         '("allnc as. (all a  bar (as::a)) -> bar as" "branch"))
```

2. Inductive definition of Bars

```
(define seqtsil (py "(tsil(tsil(tsil nat)))"))
(av "vss" "wss" seqtsil)

(apc "Insertfolder"(mk-arrow seqtsil word nat seqtsil) 1)
(add-computation-rule (pt "Insertfolder (vss::ws) w i")
                      (pt "[if (i=Lh vss)
                               (vss::(ws::w))
                               ((Insertfolder vss w i)::ws)]"))

(add-ids (list (list "Bars" (make-arity seqtsil) "trees"))
         '("allnc vss,i. i< Lh vss -> Good ((vss)__i) -> Bars vss" "Leafs")
         '("allnc vss. (all w,i,n. n=Lh vss -> i< n ->
                                Bars(Insertfolder vss w i))
                       -> Bars vss" "Branchs"))
```

3. Definitions of lasts, bseq and folder

```
(apc "lasts" (mk-arrow seq word)1)
(add-computation-rule (pt "lasts (Lin (tsil nat))")(pt "(Lin  nat)"))
(add-computation-rule (pt "lasts (ws::(w::a))")(pt "(lasts ws)::a"))

(apc "bseq" (mk-arrow word word) 1)
(add-computation-rule (pt "bseq (Lin nat)")(pt "(Lin  nat)"))
(add-computation-rule (pt "bseq (as::a)")
                      (pt "[if (ll (bseq as) a)
                               (bseq as)
                               ((bseq as)::a)] "))

(apc "memb" (mk-arrow nat word nat) 1)
(add-computation-rule (pt "memb a (w::b)")
                      (pt "[if (a=b) (Lh w) (memb a w)]"))

(apc "folder" (mk-arrow seq seqtsil) 1)
(add-computation-rule (pt "folder (Lin (tsil nat))")
                      (pt "(Lin (tsil(tsil nat)))"))
(add-computation-rule (pt "folder (ws::(w::a))")
                      (pt "[if (ll (bseq (lasts ws)) a)
                               (Insertfolder (folder ws) w
                                  (memb a (bseq (lasts ws))))
                               ((folder ws)::(ws::w))]"))
```

4. Interactive proofs and Program extraction

```
; Lemma 1 (Lemma 5.7, i)

(aga "Lemma1" (pf "allnc vss,ws,i. i< Lh vss -> Good (vss__i) ->
                                    folder ws= vss -> Good ws"))
; Lemma 2i (Lemma 5.8 i)

(set-goal (pf "Bars (Lin (tsil(tsil nat)))"))
(intro 1)
(ng)
(assume "w" "i" "n" 1)
(simp 1)
(ng)
(strip)
(use "Efq")
(use 2)
(save "Lemma2i")

; Lemma 2ii (Lemma 5.8ii)

(set-goal (pf " allnc ws. Bar ws -> allnc wss.Bars wss -> Bars (wss::ws)"))
(assume "ws0")

; 1. Ind(Bar).
(elim)
; 1.1
(strip)
(intro 0 (pt "Lh wss"))
(ng)
(use "Truth-Axiom")
(ng)
(use 1)

; 1.2
(assume "ws" "ih1a" "ih1b" "wss0")
(drop "ih1a")

; 2. Ind(Bars).
(elim)
; 2.1.
(strip)
(intro 0 (pt "i"))
(aga "Aux1" (pf "allnc i,j.i<j -> i<j+1"))
(use-with "Aux1" (pt "i")(pt "Lh vss") 3)
```

```
(ng)
(aga "Aux2" (pf "allnc i,j.i<j -> i=j->F"))
(inst-with  "Aux2" (pt "i")(pt "Lh vss") 3)
(simp 5)
(ng)
(use 4)

; 2.2
(assume "wss" "ih2a" "ih2b")
(intro 1)
(assume "w" "i" "n" 5)
(simp 5)
(strip)
(ng)

; 6:i<Succ Lh wss, hence either i=Lh vss  or  i<Lh vss
; instead of cases on i=Lh wss, which is not allowed since wss is a cv-var,
; we do cases on i+1=n (Note: 5:n=Succ Lh wss).
(cases (pt "i+1=n"))
; case1: i=Lh vss
(simp 5)
(ng)
(strip)
(simp 7)
(ng)
(use "ih1b")
(intro 1)
(use "ih2a")

;case 2: i<Lh vss (= n-1)
(simp 5)
(ng)
(strip)
(simp 7)
(ng)
(use "ih2b" (pt "n-1"))
(simp 5)
(ng)
(use "Truth-Axiom")
(simp 5)
(ng)
(aga "Aux3" (pf "allnc i,k.i<k+1 -> (i=k -> F) -> i<k"))
(use "Aux3")
(use 6)
(use 7)
```

```
(save "Lemma2ii")

(av "gc" (py "tsil nat=>trees=>trees"))
(av "gd" (py "tsil nat=>nat=>nat=>trees"))
(term-to-expr (nt (proof-to-extracted-term (current-proof))))
((|(Rec tree=>trees=>trees)| (lambda (trees2) |Leafs|))
 (lambda (ga2)
   (lambda (gc3)
     ((|(Rec trees=>trees)| |Leafs|)
      (lambda (gd5)
        (lambda (gd6)
          (|Branchs|
            (lambda (w7)
              (lambda (a8)
                (lambda (a9)
                  ("if" ((|Succ| a8) "=" a9)
                        ((gc3 w7) (|Branchs| gd5))
                        (((gd6 w7) a8) (|Pred| a9))))))))))))


; Higman's Lemma (Proposition 5.9)

(set-goal (pf "allnc as. bar as ->
               allnc vss. Bars vss ->
               all ws. bseq (lasts ws) = as ->
                       folder ws = vss ->
                       Bar ws"))
(assume "as0")

; Ind(bar)
(elim)

; 1.1
(strip)
(use "Efq")
(aga "Aux4" (pf "allnc as,ws . good as -> bseq (lasts ws)=as -> F"))
(use-with "Aux4" (pt "as")(pt "ws") 1 3)

; 1.2
(assume "as" "ih1a" "ih1b" "vss0")
(drop "ih1a")

;Ind(Bars)
(elim)
```

```
; 2.1.
(strip)
(intro 0)
(use-with "Lemma1" (pt "vss")(pt "ws")(pt "i") 3 4 6)

; 2.2.
(assume "vss" "ih2a" "ih2b" "ws" 5 6)
(intro 1)

; Ind(w)
(ind)

; 3.1
(use "Prop1")

; 3.2
(assume "w" "a" "ih3")

; To show: Bar(ws::w::a)
(cases (pt "ll (bseq(lasts ws)) a"))

; case1: ll (bseq(lasts ws)) a
(strip)
(use "ih2b" (pt "w") (pt "memb a (bseq (lasts (ws::w::a)))")
            (pt "Lh (folder ws)"))
(simp 6)
(ng)
(use "Truth-Axiom")

(aga "Aux5" (pf " allnc ws,a. ll(bseq(lasts ws))a ->
                  memb a(bseq(lasts ws::a))<Lh(folder ws)"))
(use "Aux5")
(use 8)

; bseq(lasts(ws::w::a))=as
(ng)
(simp 8)
(ng)
(use 5)

; folder(ws::w::a)=Insertfolder vss w(memb a(bseq(lasts(ws::w::a))))
(ng)
(simp 8)
(ng)
(simp 6)
```

```
(ng)
(use "Truth-Axiom")

; case2: (ll(bseq(lasts ws))a -> F)
(strip)
(use "ih1b" (pt "a")(pt "(folder ws)::(ws::w)"))

; Bars (folder ws::ws::w)
(use "Lemma2ii")

; Bar(ws::w)
(use "ih3")

; Bars vss
(simp 6)
(intro 1)
(use "ih2a")

; bseq(lasts(ws::w::a))=(as::a)
(ng)
(simp 8)
(ng)
(use 5)

; folder(ws::w::a)=((folder ws)::(ws::w))
(ng)
(simp 8)
(ng)
(use "Truth-Axiom")
(save "Theorem")

(av "ge" (py "nat=>lltree"))
(av "gf" (py "nat=>trees=>tsil(tsil nat)=>tree"))
(av "gg" (py "tsil nat=>nat=>nat=>tsil(tsil nat)=>tree"))
(define program (proof-to-extracted-term (current-proof)))
(term-to-expr (nt program))

((|(Rec lltree=>trees=>tsil(tsil nat)=>tree)|
   (lambda (trees3) (lambda (ws4) |Leaf|)))
 (lambda (ge3)
   (lambda (gf4)
     ((|(Rec trees=>tsil(tsil nat)=>tree)| (lambda (ws7) |Leaf|))
      (lambda (gd7)
        (lambda (gg8)
          (lambda (ws9)
```

```
            (|Branch|
              (((|(Rec tsil nat=>tree)| (|Branch| (lambda (w11) |Leaf|)))
               (lambda (w11)
                 (lambda (a12)
                   (lambda (tree13)
                     ("if" ((ll (bseq (lasts ws9))) a12)
                           ((((gg8 w11)
                              ((memb a12)
                               ("if" ((ll (bseq (lasts ws9))) a12)
                                     (bseq (lasts ws9))
                                     ((bseq (lasts ws9)) "::" a12))))
                             ("Lh" (|folder| ws9)))
                            (ws9 "::" (w11 "::" a12)))
                           (((gf4 a12)
                             ((|cLemmaTwoii| tree13) (|Branchs| gd7)))
                            (ws9 "::" (w11 "::" a12))))))))))))))))

(animate "Theorem")
(animate "Lemma2i")
(animate "Lemma2ii")

; For simplicity, we assume that our finite alphabet consists of
; a given number of letters, e.g. 5.
(aga "finitealphabet" (pf " all as. 5<Lh as -> good as"))

(set-goal (pf "bar (Lin nat)"))
(intro 1)(assume "a0")
(intro 1)(assume "a1")
(intro 1)(assume "a2")
(intro 1)(assume "a3")
(intro 1)(assume "a4")
(intro 1)(assume "a5")
(intro 0)(use "finitealphabet")
(ng)
(use "Truth-Axiom")
(save "barNil")
(animate "barNil")

; Higman's Lemma, Bar[]

(set-goal (pf "Bar (Lin (tsil nat))"))
(use "Theorem" (pt "(Lin nat)")(pt "(Lin (tsil (tsil nat)))"))
(use "barNil")
(use "Lemma2i")
(ng)
```

```
(use "Truth-Axiom")
(ng)
(use "Truth-Axiom")
(save "Higman-finite")
(animate "Higman-finite")

; Every infinite sequence has a good initial segment

(set-goal (pf "all f. ex m. Good (Init f m)"))
(assume "f")
(use "Bar-thm" (pt "(Lin (tsil nat))")(pt "0"))
(use "Higman-finite")
(ng)
(use "Truth-Axiom")

(define program (proof-to-extracted-term (current-proof)))
(define nprogram (nt program))


 5. Test of the program

; We define sequences: nat->word via adding term rewriting rules.
; The extracted program yields a number n
; such the initial segment of length n is good.

(define (run-higman infinite-sequence)
  (dt (nt (mk-term-in-app-form nprogram infinite-sequence))))

; a. [4 1], [3 3 0], [0 4 0 1], [2], ...

(apc "Seq" (mk-arrow (py "nat")(py "(tsil nat)")) 1)
(add-rewrite-rule (pt "Seq 0")(pt ":4::1"))
(add-rewrite-rule (pt "Seq 1")(pt "(:3::3)::0"))
(add-rewrite-rule (pt "Seq 2")(pt "((:0::4)::0)::1"))
(add-rewrite-rule (pt "Seq (++(++(++ n)))")(pt ":2"))
(run-higman (pt "Seq"))
; ==> 3

; b. [0 0], [1], [1 0], [], [], ...

(apc "Interesting" (mk-arrow nat word))
(add-rewrite-rule (pt "Interesting 0")(pt ":0::0"))
(add-rewrite-rule (pt "Interesting 1")(pt ":1"))
(add-rewrite-rule (pt "Interesting 2")(pt ":1::0"))
(add-rewrite-rule (pt "Interesting (++(++(++ n)))")(pt "(Lin nat)"))
```

```
(run-higman (pt "Interesting"))
; ==> 5
; Example that not the shortest good initial segment is found!

; c. [1], [3], [5], [7], [9], [0], ...

(apc "Sixelts" (mk-arrow nat word))
(add-rewrite-rule (pt "Sixelts 0")(pt ":1"))
(add-rewrite-rule (pt "Sixelts 1")(pt ":3"))
(add-rewrite-rule (pt "Sixelts 2")(pt ":5"))
(add-rewrite-rule (pt "Sixelts 3")(pt ":7"))
(add-rewrite-rule (pt "Sixelts 4")(pt ":9"))
(add-rewrite-rule (pt "Sixelts 5")(pt ":0"))
(run-higman (pt "Sixelts"))
;==> 6
; So the proof yields that the sequence [[1], [3], [5], [7], [9], [0]]
; is good; i.e., two of the used numbers must be equal.
; This is because we have assumed that our finite alphabet consists of
; only five letters. More generally, we could have also proven:
; for any n, there holds Higman's Lemma for n letters.

; Note that in order to keep the formalization of this theorem as
; short as possible we allowed unproven lemmas. For proofs see
; the Minlog repository.
```

Lebenslauf

Persönliche Daten

Name	Monika Seisenberger
Geburtsdatum	30.04.1968
Geburtsort	Obertaufkirchen
Nationalität	deutsch
Familienstand	ledig

Schulischer Werdegang

01.09.78 – 26.06.87	Gymnasium Gars am Inn
26.06.87	Abitur
01.11.87 – 31.12.90	Studium Lehramt Gymnasium Mathematik/ katholische Theologie an der Universität München
01.01.91 – 20.12.94	Studium Lehramt Gymnasium Mathematik/Physik an der Universität München
20.12.94	1. Staatexamen Lehramt Mathematik/Physik
01.10.95 – 31.07.96	Grundstudium Informatik an der TU München
30.03.98	Diplomprüfung in Mathematik
01.05.98 – 30.04.01	Stidendiatin im Graduiertenkolleg 'Logik in der Informatik'

Beruflicher Werdegang

01.01.95 – 30.09.97	Wissenschaftliche Mitarbeiterin
01.02.98 – 31.05.98	an der Universität München im Rahmen des DFG-Projekts 'Deduktion'
01.04.02 – 30.09.02	Wissenschaftliche Mitarbeiterin an der Universität München
01.10.01 – 31.03.02	Wissenschaftliche Assistentin (Research Assistant)
Seit 01.11.02	an der University of Wales Swansea