Higman's Lemma and Its Computational Content

Helmut Schwichtenberg, Monika Seisenberger and Franziskus Wiesnet

Dedicated to Gerhard Jäger on the occasion of his 60th birthday

Abstract Higman's Lemma is a fascinating result in infinite combinatorics, with manyfold applications in logic and computer science. It has been proven several times using different formulations and methods. The aim of this paper is to look at Higman's Lemma from a computational and comparative point of view. We give a proof of Higman's Lemma that uses the same combinatorial idea as Nash-Williams' indirect proof using the so-called minimal bad sequence argument, but which is constructive. For the case of a two letter alphabet such a proof was given by Coquand. Using more flexible structures, we present a proof that works for an arbitrary well-quasiordered alphabet. We report on a formalization of this proof in the proof assistant Minlog, and discuss machine extracted terms (in an extension of Gödel's system T) expressing its computational content.

Keywords Higman's Lemma · Inductive definitions · Minimal bad sequence · Program extraction

H. Schwichtenberg (✉) · F. Wiesnet
Mathematisches Institut, LMU, Theresienstr. 39, D-80333 Munich, Germany
e-mail: schwicht@math.lmu.de

F. Wiesnet
e-mail: franziskus.wiesnet@gmx.de

M. Seisenberger
Department of Computer Science, Swansea University, Swansea SA28PP, UK
e-mail: m.seisenberger@swansea.ac.uk

R. Kahle et al. (eds.), *Advances in Proof Theory*, Progress in Computer Science and Applied Logic 28, DOI 10.1007/978-3-319-29198-7_11

1 Introduction

Without exaggeration it can be said that Higman's Lemma [15] is one of the most often proven theorems in Mathematical Logic and Theoretical Computer Science. The fascination of this theorem is due to the fact that it has various formulations and is of interest in different areas such Proof theory, Constructive Mathematics, Reverse Mathematics, and Term rewriting, as we will briefly discuss further below.

Nash-Williams [19] gave a very concise classical proof using the so-called minimal bad sequence argument. In the following we briefly recall well-quasiorderings and sketch Nash-Williams' proof.

Definition A binary relation $\preceq$ on a set A is a well-quasiorder (wqo) if (i) it is transitive and (ii) every infinite sequence in A is "good", i.e., $\forall_{(a_i)_{i<\omega}} \exists_{i,j} (i < j \wedge a_i \preceq a_j)$.

Let A^* denote the set of finite sequences ("words") with elements in A. We call a word $[a_1, \ldots, a_n]$ embeddable ($\preceq^*$) in $[b_1, \ldots, b_m]$ if there exists a strictly increasing map $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_i \preceq b_{f(i)}$ for all $i \in \{1, \ldots n\}$.

Now Higman's Lemma says

If $(A, \preceq)$ is a well-quasiorder, then so is $(A^*, \preceq^*)$.

Nash-Williams' proof proceeds as follows. That a bad sequence of words, i.e., a sequence that is not good, is impossible is basically a consequence of two facts: (a) for each bad sequence exists a bad sequence with is smaller in a lexicographical sense, and (b), if there exists a bad sequence, then exists also a minimal bad sequence with respect to this lexicographical order. We give the proof in more detail:

(1) In order to show "wqo $(A, \preceq)$ implies wqo $(A^*, \preceq^*)$" assume for contradiction that there is a bad sequence of words in A^*.
(2) Among all infinite bad sequences of words 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 a proper initial segment of w_n, does not.
(3) Since for all i $w_i \neq [\,]$, let $w_i = a_i * v_i$. By Ramsey's theorem and the fact that our alphabet A is a well-quasiorder, there exists an infinite subsequence $a_{\kappa_0} \preceq a_{\kappa_1} \preceq \cdots$ 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).

The computational content of Nash-Williams' proof was first investigated by Murthy [17], by applying Friedman's A-translation in the interactive theorem prover NuPRL to the classical proof. Murthy represented functions as relations to eliminate choice, and used second order classical logic. However, due to the size of the translated proof and program, the resulting program could only be run on trivial input. In [29], the second author formalized Nash-Williams' proof in the proof assistant Minlog, by applying a refined version of the A-translation and, contrary to Murthy,

not eliminating the axiom of classical dependent choice, but rather adding computational content to it using bar recursion. This resulted in a considerable smaller extracted program, but still with infeasible run-times due to the eager evaluation strategy of Minlog's term language. Reasonable results have been only obtained recently thanks to a Minlog extension which translates extracted terms to Haskell. Other formalizations of the classical proof include Herbelin's formalization of Murthy's A-translated proof in Coq [14] and Sternagel's formalization of Nash-Williams' proof in Isabelle [30] which also provides a proof of Kruskal's theorem. However, [30] does not include the extraction of a program. Recently, Powell [21] applied Gödel's Dialectica Interpretation to this proof. The interpretation yields a program, but no formalization has been provided so far.

In this paper, we aim at a constructive proof (without choice) which has the same underlying construction as Nash-Williams' proof but allows us to directly read off the program. For a {0, 1}-alphabet such a proof was given by Coquand and Fridlender [6, 7]. Here we provide a proof and a formalization for full Higman's Lemma, and also discuss how this proof is related to other constructive proofs. The paper is organized as follows: We give a constructive reformulation of Nash-Williams' proof in Sect. 2 and comment on its formalization in Sect. 3. In Sect. 4 we spell out the computational content of some of the proofs. Each time it comes in the form of a term (in an extension T^+ of Gödels T) machine extracted from a formalization of the respective proof. We give an overview on existing formalizations of the Coquand/Fridlender proof at the end of Sect. 2 and add a comparison with other constructive proofs in Sect. 5.

2 A Constructive Reformulation of Nash-Williams' Proof

The objective of this section is to present a constructive proof of Higman's Lemma that uses the same combinatorial idea as Nash-Williams' classical proof and generalizes the proof by Coquand and Fridlender. Such a proof (without formalization) has been given in [28]. However, if one is interested in the computational content one has to reformulate this proof and to change it at various places to make the computational content visible (see also the remark at the end of this section). We use an inductive characterization of a binary relation satisfying condition (ii) in the definition of a well-quasiorder; such relations have been called "almost full" in [33]. Our characterization is via a "bar" predicate which comes in two variants, one for the alphabet and one for words, see below for a definition. Thus, the statement we are going to prove is

$$\mathrm{BarA}_{\preceq}[\,] \to \mathrm{BarW}_{\preceq}[\,].$$

Throughout the whole paper we assume $\preceq$ to be a binary relation on a set A which is decidable in the sense that it is given by a binary total function into the booleans; transitivity will not be needed. It suffices to let A be the set of natural numbers. Most of our notions will depend on the $\preceq$-relation. However, we usually suppress this dependence, since $\preceq$ will be kept fixed most of the time.

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^*,
$vs, ws, \ldots$ for finite sequences of words, i.e., elements of A^{**}.

Definition (*Higman embedding, inductive*) The embedding relation $\preceq^*$ on A^* is defined inductively by the following axioms (written as rules):

$$\frac{}{[] \preceq^* []} \qquad \frac{v \preceq^* w}{v \preceq^* a*w} \qquad \frac{a \preceq b \quad v \preceq^* w}{a*v \preceq^* b*w}$$

where $*$ denotes the cons operation on lists.

Definition GoodA as expresses that a finite sequence as of letters is good; note that finite sequences grow to the left, i.e., a finite sequence is *good* if there are two elements such that the one to the left is larger than or equal to w.r.t. $\preceq$ to the one on the right. A sequence is called *bad* if it is not good. Furthermore, we use

$$\begin{aligned}
\mathrm{Ge}_\exists(a, as) &:= \exists_{i<|as|} a \succeq (as)_i,\\
\mathrm{Ge}_\forall(a, as) &:= \forall_{i<|as|} a \succeq (as)_i,\\
\mathrm{Ge}_{\exists\forall}(a, ws) &:= \exists_{i<|ws|} \forall_{j<|(ws)_i|} a \succeq (ws)_{i,j}.
\end{aligned}$$

A finite sequence $as = [a_{n-1}, .., a_0]$ is *decreasing* if $a_j \succeq a_i$ whenever $j \geq i$. Further, BSeq as determines the "first" bad subsequence occurring in as:

$$\begin{aligned}
\mathrm{BSeq}\,[] &:= [],\\
\mathrm{BSeq}\,(a*as) &:= \begin{cases} a*\mathrm{BSeq}\,as & \text{if } \neg\mathrm{Ge}_\exists(a, as),\\ \mathrm{BSeq}\,as & \text{otherwise.}\end{cases}
\end{aligned}$$

Definition We inductively define a set BarA $\subseteq A^*$ by the following rules:

$$\frac{\mathrm{GoodA}\ as}{\mathrm{BarA}\ as} \qquad \frac{\forall_a\,\mathrm{BarA}\ a*as}{\mathrm{BarA}\ as}.$$

BarW ws is defined similarly, using the corresponding GoodW ws. However, since GoodW is a predicate on words, it refers to the embedding relation $\preceq^*$ on A^* rather than $\preceq$ directly.

As in the end we are interested in getting a program that for any sequence of words yields witnesses that this sequence is good we also prove the following.

Proposition (BarWToGoodInit) BarW[] *implies that every infinite sequence of words has a good initial segment.*

Proof Let f be a variable of type `nat=>list nat`. We show, more generally,

$$\forall_{ws,f,n}(\mathrm{BarW}\ ws \to \mathrm{Rev}(\bar{f}n) = ws \to \exists_m \mathrm{GoodW}(\mathrm{Rev}(\bar{f}m)))$$

by induction on BarW. The proposition then follows with $ws = [\,]$.

1. GoodW ws. Assume that there are an infinite sequence f and a number n such that $\mathrm{Rev}(\bar{f}n) = ws$ (i.e., $[f(n-1), \ldots, f0] = ws$). Since ws is good, we can take m to be n.
2. Using the induction hypothesis

$$\forall_{w,f,n}(\mathrm{Rev}(\bar{f}n) = w{*}ws \to \exists_m \mathrm{GoodW}(\mathrm{Rev}(\bar{f}m)))$$

with fn, f and $n+1$, we only have to prove $\mathrm{Rev}(\bar{f}(n+1)) = fn{*}ws$, which follows from $\mathrm{Rev}(\bar{f}n) = ws$. □

Note that the reverse direction expresses a form of bar induction. However, for the proof below the present direction suffices.

In the following we want to first highlight the idea behind the constructive proof. This is best done by showing how the steps (1)–(4) in the proof of Nash-Williams given in the introduction are dealt with in the inductive proof.

(1) Prove inductively "BarA $[\,] \to$ BarW$[\,]$".
(2) The minimality argument will be replaced by structural induction on words.
(3) Given a sequence $ws = [w_n, \ldots, w_0]$ s.t. $w_i = a_i{*}v_i$, we are interested in all decreasing subsequences $[a_{\kappa_l}, \ldots, a_{\kappa_0}]$ of maximal length and their corresponding sequences $v_{\kappa_l}, \ldots, v_{\kappa_0}, w_{\kappa_0-1}, \ldots, w_0$. The sequences $[a_{\kappa_l}, \ldots, a_{\kappa_0}]$ 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 node form such a descending subsequence $[a_{\kappa_i}, \ldots, a_{\kappa_0}]$ and the corresponding left-hand component consists of the sequence $[v_{\kappa_i}, \ldots, v_{\kappa_0}, w_{\kappa_0-1}, \ldots, w_0]$. If we extend the sequence ws to the left by a word $a{*}v$, then in the existing forest either new nodes, possibly at several places, are inserted, or a new singleton tree with root node $\langle v{*}ws, [a]\rangle$ is added. Now the informal idea of the inductive proof is: if in Forest ws new nodes cannot be inserted infinitely often (without ending up with a good left-hand component in a node) and if also new trees cannot be added infinitely often, then ws can not be extended badly infinitely often. Formally, this will be captured by the statement:

$$\forall_{ws}(\mathrm{BarW}(\mathrm{BSeq}(\mathrm{Heads}\ ws)) \to \mathrm{BarF}(\mathrm{Forest}\ ws) \to \mathrm{BarW}\ ws).$$

(4) The first part of item (4) corresponds to GoodWForestToGoodW.

Definition For a finite sequence ws of words let Heads ws denote the finite sequence consisting of the starting letters of the non-empty words. We call a finite sequence ws of words admissible (Adm ws) if each word in ws is non-empty.

Notation We use t for elements in $T(A^{**} \times A^*)$, i.e., trees labeled by pairs in $A^{**} \times A^*$, and ts, ss for elements in $(T(A^{**} \times A^*))^*$, i.e., forests. The tree with root $\langle ws, as\rangle$ and list of subtrees ts is written $\langle ws, as\rangle ts$. We use the destructors Left and Right for pairs and the destructors Root and Subtrees for trees. For better readability we set:

$$\begin{array}{ll}
\text{Newtree}\ \langle ws, as\rangle & := \langle ws, as\rangle[\,], \\
\text{Roots}\ [t_{n-1}, \ldots, t_0] & := [\text{Root}\ t_{n-1}, \ldots, \text{Root}\ t_0], \\
\text{Lefts}\ [\langle vs_{n-1}, as_{n-1}\rangle, \ldots, \langle vs_0, as_0\rangle] & := [vs_{n-1}, \ldots, vs_0], \\
\text{Rights}\ [\langle vs_{n-1}, as_{n-1}\rangle, \ldots, \langle vs_0, as_0\rangle] & := [as_{n-1}, \ldots, as_0].
\end{array}$$

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

$$\begin{array}{l}
\text{Forest}\,[\,] := [\,], \\
\text{Forest}\,[\,]{*}ws := \text{Forest}\ ws, \\
\text{Forest}\,(a{*}v){*}ws := \\
\quad \begin{cases}
\text{InsertF}(\text{Forest}\ ws, v, a) & \text{if } \text{Ge}_{\exists}(a, \text{BSeq}(\text{Heads}\ ws)), \\
\text{Newtree}\ \langle v{*}ws, [a]\rangle{*}(\text{Forest}\ ws) & \text{otherwise}
\end{cases}
\end{array}$$

where

$$\text{InsertF}(ts, v, a) := \text{map}\left(\lambda_t \left[\begin{array}{l} \text{if } \text{Ge}_{\forall}(a, \text{Right}\ (\text{Root}\ t)) \\ \quad \text{InsertT}(t, v, a) \\ \quad t \end{array}\right]\right) ts$$

and

$$\begin{array}{l}
\text{InsertT}(\langle vs, as'\rangle ts, v, a) := \\
\quad \begin{cases}
\langle vs, as'\rangle \text{InsertF}(ts, v, a) & \text{if } \text{Ge}_{\exists\forall}(a, \text{Rights}(\text{Roots}\ ts)), \\
\langle vs, as'\rangle(\text{Newtree}\ \langle v{*}vs, a{*}as'\rangle{*}ts) & \text{otherwise.}
\end{cases}
\end{array}$$

Example Take as (almost full) relation the natural numbers with $\leq$. For better readability we use underlining rather than parentheses to indicate a list. We will only use one-digit numbers, hence every digit stands for a natural number. Then Forest $[\underline{28}, \underline{421}, \underline{69}, \underline{35}]$ is

$$(\underline{[8}, \underline{421}, \underline{69}, \underline{35}],\underline{2}) \qquad \frac{(\underline{[21}, \underline{5}],\underline{43}) \qquad (\underline{[9}, \underline{5}],\underline{63})}{(\underline{[5}],\underline{3})}$$

and Forest $[\underline{52}, \underline{28}, \underline{421}, \underline{69}, \underline{35}]$ is

$$\dfrac{\dfrac{([\underline{2},\,\underline{8},\,\underline{421},\,\underline{69},\,\underline{35}],\underline{52})}{([\underline{8},\,\underline{421},\,\underline{69},\,\underline{35}],\underline{2})} \qquad \dfrac{\dfrac{([\underline{2},\,\underline{21},\,\underline{5}],\underline{543})}{([\underline{21},\,\underline{5}],\underline{43})} \qquad ([\underline{9},\,\underline{5}],\underline{63})}{([\underline{5}],\underline{3})}}$$

If we "project" each node to its right-hand-side we obtain

$$\underline{2} \qquad \dfrac{\underline{43} \qquad \underline{63}}{\underline{3}} \qquad \text{and} \qquad \dfrac{\underline{52}}{\underline{2}} \qquad \dfrac{\dfrac{\underline{543}}{\underline{43}} \qquad \underline{63}}{\underline{3}}$$

The leaves of e.g. the final tree are the maximal decreasing subsequences of the heads [52463] of [$\underline{52}$, $\underline{28}$, $\underline{421}$, $\underline{69}$, $\underline{35}$]. Recall that the left-hand-side of each leaf consists of the sequence $[v_{\kappa_i}, \ldots, v_{\kappa_0}, w_{\kappa_0-1}, \ldots, w_0]$, and its right-hand-side is the maximal descending subsequence $[a_{\kappa_i}, \ldots, a_{\kappa_0}]$ of $[a_n, \ldots, a_0]$. In the example the leaf ([$\underline{2}$, $\underline{8}$, $\underline{421}$, $\underline{69}$, $\underline{35}$,], $\underline{52}$,) has exactly this form: $\underline{52}$ is a maximal descending subsequence of $\underline{52463}$, and we have $[\underline{52}, \underline{28}, \underline{421}, \underline{69}, \underline{35}] = [(5 * \underline{2}), (2 * \underline{8}), \underline{421}, \underline{69}, \underline{35}]$.

Definition Let $t \in T(A^{**} \times A^*)$. Then t is a *tree with a good leaf* (GLT t) if there is a leaf with a good left side. We inductively define the predicate BarF $\subseteq (T(A^{**} \times A^*))^*$ by the rules

$$\dfrac{\mathrm{GLT}(ts)_i}{\mathrm{BarF}\,ts} \qquad \dfrac{\forall_{a,v}(\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\,ts)) \to \mathrm{BarF}(\mathrm{InsertF}(ts, v, a)))}{\mathrm{BarF}\,ts}$$

Lemma (GoodWProjForestToGoodW, BSeqHeadsEqRhtsRootsForest)

(a) $\forall_{ws,i}(i < \mathrm{Lh}\,ws \to \mathrm{GLT}(\mathrm{Forest}\ ws)_i \to \mathrm{GoodW}\ ws)$.
(b) $\forall_{ws}(\mathrm{Adm}\,ws \to \mathrm{BSeq}(\mathrm{Heads}\,ws) = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,(\mathrm{Forest}\ ws))))$.

Proof Both parts follow from the construction of Forest; the proof of (a) is rather laborious and involves a number of auxiliary notions. However, since we are mainly interested in computational content and this lemma has none, we do not give details. □

Lemma (BarFNil, BarFAppd)

(a) BarF[ ].
(b) $\forall_{t,ts}(\mathrm{BarF}[t] \to \mathrm{BarF}ts \to \mathrm{BarF}t{*}ts)$.

Proof (a) BarF[ ] follows from the second rule of the definition of BarF, using ex-falso-quodlibet.

(b) This assertion holds since InsertF is defined by a map operation. In more detail, using # for the concatenation of two lists, we prove

$$\forall_{ts}(\mathrm{BarF}ts \to \forall_{ss}(\mathrm{BarF}ss \to \mathrm{BarF}ts\#ss))$$

by induction on BarFts. The base case is straightforward as $\mathrm{GLT}(ts)_i$ implies $\mathrm{GLT}(ts\#ss)_i$. In the step case we have

$$\mathrm{ih}_1 : \forall_{v,a}(\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ts)) \to \forall_{ss}(\mathrm{BarF}\, ss \to \mathrm{BarF}(\mathrm{InsertF}(ts, v, a)\#ss)))$$

and need to prove $\forall_{ss}(\mathrm{BarF}\, ss \to \mathrm{BarF}\, ts\#ss)$. Fix $ss \in A^*$ and use induction on $\mathrm{BarF}\, ss$. The base case again is easy since $\mathrm{GLT}(ss)_i$ implies that there is j such that $\mathrm{GLT}(ts\#ss)_j$. In the step case we have

$$\mathrm{ih}_2 : \forall_{v,a}(\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ss)) \to \mathrm{BarF}\, ts\#\mathrm{InsertF}(ss, v, a))$$

as well as its "strengthening"

$$\mathrm{ih}_{2a} : \forall_{v,a}(\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ss)) \to \mathrm{BarF}(\mathrm{InsertF}(ss, v, a))).$$

To show $\mathrm{BarF}\, ts\#ss$, assume v, a with $\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ts\#ss))$ and show $\mathrm{BarF}(\mathrm{InsertF}(ts\#ss, v, a))$.

Case 1. $\neg\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ts))$, i.e., new nodes are only added to ss; ts remains unchanged. Then $\mathrm{BarF}\, ts\#ss'$ follows by ih_2.

Case 2. $\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ts)))$. First assume that new nodes are added to both ts and ss, i.e., $\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ss))$. In this case we use with v, a and $\mathrm{InsertF}(ss, v, i)$. We still need to show $\mathrm{BarF}(\mathrm{InsertF}(ss, v, a))$, which holds because of ih_{2a}.

Now assume $\neg\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ss))$, i.e., new nodes are only added to ts. In this case we apply ih_1 with v, a and ss where we use ih_{2a} and the definition of BarF to obtain $\mathrm{BarF}\, ss$. □

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

Lemma (BarFNew) *Assume* BarA [ ]. *Then*

$$\forall_{ws_0}(\mathrm{BarW}\ ws_0 \to \forall_{as_0}\mathrm{BarF}[\mathrm{Newtree}\ \langle ws_0, as_0\rangle]).$$

Proof $\mathrm{Ind}_1(\mathrm{BarW})$. 1.1. $\mathrm{GoodW}\ ws_0$. Then $\mathrm{GLT}(\mathrm{Newtree}\ \langle ws_0, as_0\rangle)$, i.e., BarF $[\mathrm{Newtree}\ \langle ws_0, as_0\rangle]$. 1.2. Assume

$$\mathrm{ih}_1 : \forall_{w,as}\mathrm{BarF}[\mathrm{Newtree}\ \langle w{*}ws_0, as\rangle].$$

Let $as_0 \in A$. Instead of proving $\mathrm{BarF}[\mathrm{Newtree}\ \langle ws_0, as_0\rangle]$ we show more generally that this assertion holds for all t with $\mathrm{Root}\, t = \langle ws_0, as_0\rangle$ and (a) $\mathrm{Subtrees}\, t$ in BarF, and (b) $\mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(\mathrm{Subtrees}\, t)))$ in BarA. We do this by main induction on (b) and side induction on (a), i.e., we prove

$$\forall_{as}(\mathrm{BarA}\ as \to \neg\mathrm{GoodA}\ as \to \forall_{ts}(\mathrm{BarF}\, ts \to as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\, ts)) \to \mathrm{BarF}[\langle ws_0, as_0\rangle ts])).$$

$\mathrm{Ind}_2(\mathrm{BarA})$. 2.1. GoodA as. Then the conclusion follows immediately by ex-falso-quodlibet with the premise $\neg\mathrm{GoodA}\,as$. 2.2. BarA as is obtained by the second rule. We assume as and

$$\mathrm{ih}_2 : \forall_{a,ts}(\mathrm{BarF}\,ts \to a{*}as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts)) \to \mathrm{BarF}[\langle ws_0, as_0\rangle ts]),$$

and have to show

$$\forall_{ts}(\mathrm{BarF}\,ts \to as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts)) \to \mathrm{BarF}[\langle ws_0, as_0\rangle ts]).$$

$\mathrm{Ind}_3(\mathrm{BarF})$. 3.1. Fix $(ts)_i$ such that $\mathrm{GLT}(ts)_i$. By the first clause of BarF, for any t such that $\mathrm{Subtrees}\,t = ts$, $\mathrm{GLT}(ts)_i$ implies $\mathrm{BarF}[t]$. 3.2. Fix ts with $as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts))$ and assume the induction hypothesis

$$\begin{aligned}\mathrm{ih}_3 : \forall_{v,a}(&\mathrm{Ge}_\exists(a, \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts))) \to\\ &as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(\mathrm{InsertF}(ts, v, a)))) \to\\ &\mathrm{BarF}[\langle ws_0, as_0\rangle \mathrm{InsertF}(ts, v, a)])\end{aligned}$$

together with its strengthening

$$\mathrm{ih}_{3a} : \forall_{v,a}(\mathrm{Ge}_\exists(a, \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts))) \to \mathrm{BarF}(\mathrm{InsertF}(ts, v, a))).$$

To show $\mathrm{BarF}[\langle ws_0, as_0\rangle ts]$ we use the second clause, i.e., prove

$$\forall_{v,a}(\mathrm{Ge}_\exists(a, \mathrm{Head}[as_0]) \to \mathrm{BarF}(\mathrm{InsertF}([\langle ws_0, as_0\rangle ts], v, a))).$$

We fix v and a with $\mathrm{Ge}_\exists(a, as_0)$ and prove the statement by case distinction on how a relates to as, i.e., whether nodes in the existing subtrees ts need to be inserted, or whether a new subtree has to be added.

Case 1. $\mathrm{Ge}_\exists(a, as)$. In this case we have

$$as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts)) = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(\mathrm{InsertF}(ts, v, a))))$$

and by applying ih_3 we obtain $\mathrm{BarF}[\langle ws_0, as_0\rangle \mathrm{InsertF}(ts, v, a)]$.

Case 2. $\neg\mathrm{Ge}_\exists(a, as)$. In this case we need to show

$$\mathrm{BarF}[\langle ws_0, as_0\rangle \mathrm{Newtree}\,\langle w{*}ws_0, a{*}as_0\rangle{*}ts]$$

which can be obtained by applying ih_2 to a and $\mathrm{Newtree}\,\langle w{*}ws_0, a{*}as_0\rangle{*}ts$ provided we can show

$$\mathrm{BarF}(\mathrm{Newtree}\,\langle w{*}ws_0, a{*}as_0\rangle{*}ts).$$

This follows from $\mathrm{BarF}\,ts$ and $\mathrm{BarF}[\mathrm{Newtree}\,\langle w{*}ws_0, a{*}as_0\rangle]$ via BarFAppd. The former holds by ih_{3a}, the latter follows by ih_1.

Now, the proof of the general assertion is completed. Since BarA [ ] by assumption and BarF[ ] by BarFNil, we may in the assertion put $as = [\,]$ and $ts = [\,]$ and end up with BarW $ws \to$ BarF[Newtree $\langle ws_0, as_0\rangle$]. □

Theorem (Higman) BarA [ ] → BarW[ ].

Proof Assume $\mathrm{BarA}_{[\,]}$. We show more generally

$$\begin{aligned}&\forall_{as}(\mathrm{BarA}\ as \to\\ &\forall_{ts}(\mathrm{BarF}\,ts \to\\ &\forall_{ws}(\mathrm{Adm}\ ws \to \mathrm{BSeq}(\mathrm{Heads}\ ws) = as \to \mathrm{Forest}\ ws = ts \to \mathrm{BarW}\ ws))).\end{aligned}$$

$\mathrm{Ind}_1(\mathrm{BarA})$. 1.1. GoodA as. Then, the result follows by ex-falso-quodlibet since for any ws, BSeq(Heads ws) is bad.

1.2. Let $as \in A^*$ and assume

$$\begin{aligned}&\mathrm{ih}_1 : \forall_{a,ts}(\mathrm{BarF}\,ts \to\\ &\quad\forall_{ws}(\mathrm{Adm}\ ws \to \mathrm{BSeq}(\mathrm{Heads}\ ws) = a{*}as \to \mathrm{Forest}\ ws{=}ts \to \mathrm{BarW}\ ws)).\end{aligned}$$

$\mathrm{Ind}_2(\mathrm{BarF})$. 2.1. $\mathrm{GLT}(ts)_i$. Then, by GoodWProjForestToGoodW, for any ws such that Forest $ws = ts$ we obtain GoodW ws and hence BarW ws. 2.2. Fix ts and assume

$$\begin{aligned}&\mathrm{ih}_2 : \forall_{v,a}(\mathrm{Ge}_\exists(a, \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts))) \to\\ &\quad\forall_{ws}(\mathrm{Adm}\ ws \to \mathrm{BSeq}(\mathrm{Heads}\ ws) = as \to \mathrm{Forest}\ ws{=}\mathrm{InsertF}(ts, v, a) \to\\ &\qquad\mathrm{BarW}\ ws))\end{aligned}$$

as well as the strengthening of the induction hypothesis

$$\mathrm{ih}_{2a} : \forall_{v,a}(\mathrm{Ge}_\exists(a, \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}\,ts))) \to \mathrm{BarF}(\mathrm{InsertF}(ts, v, a))).$$

Assume that we have ws such that $\mathrm{BSeq}(\mathrm{Heads}\,ws) = as$ and Forest $ws = ts$. In order to prove BarW ws, we fix a word w and show BarW $w{*}ws$ by induction on the structure of w:

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

Case 1. $\mathrm{Ge}_\exists(a, as)$.

In this case, we have

$$\begin{aligned}&\mathrm{BSeq}(\mathrm{Heads}((a{*}w){*}ws)) = as,\\ &\mathrm{Forest}((a{*}w){*}ws) \qquad = \mathrm{InsertF}(ts, w, a).\end{aligned}$$

By BSeqHeadsEqRhtsRootsForest and the definition of Forest $(a*w)*ws$, we know that at least one node has been inserted into Forest ws. In this situation, we may apply ih_2 (to InsertF(ts, w, a) and $(a*w)*ws$) and conclude BarW$(a*w)*ws$.

Case 2. $\neg\mathrm{Ge}_\exists(a, as)$. Then we have

$$\begin{aligned}\mathrm{BSeq}(\mathrm{Heads}((a*w)*ws)) &= a*as,\\ \mathrm{Forest}(a*w)*ws) &= \mathrm{Newtree}\,\langle w*ws, [a]\rangle*ts.\end{aligned}$$

By ih_{2a} and ih_3, we have BarFts and BarW$w*ws$. Hence, by BarFNew applied to $w*ws$ and $[a]$, we obtain BarF[Newtree $\langle w*ws, [a]\rangle$]. By BarFAppd we may conclude

$$\mathrm{BarF}[\mathrm{Newtree}\,\langle w*ws, [a]\rangle*ts].$$

Now we are able to apply ih_1 (to a, Newtree $\langle w*ws, [a]\rangle*ts$ and $(a*w)*ws$) and end up with BarW$(a*w)*ws$. This completes the proof of the general assertion.

Now, by putting $as = [\,]$, $ts = [\,]$ and $ws = [\,]$ and the fact that BarF[ ] always holds (by BarFNil) we obtain BarA $[\,] \to$ BarW[ ]. □

Remark 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 [10]. Having a proof of BarW ws implies that the proof yields the information whether BarW 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 BarW $w * ws$. If we used a definition consisting of only one rule, i.e., an $\mathrm{acc}_\preceq$-notion as in [28], BarW ws would correspond to

$$\neg\exists_i(i < \mathrm{Lh}\ ws \to (ws)_i \text{ embeds into } ws) \to \mathrm{BarW}\,w*ws$$

where the test whether or not the premise holds results in a brute-force search; it is not given by the proof itself.

In the next section we discuss a formalization of this proof. For the special case of {0,1} there are formalizations in Agda (Fridlender), Minlog (Seisenberger), Isabelle (Berghofer, [3]) and Coq (Berghofer). The formalization of the general case is much more elaborate. Such a formalization has been given in Coq, by Delobel.[1] However, its computational content has not been extracted and investigated. It would suffer from the point made in the previous remark, and its usage of the Set/Prop distinction (see footnote 2) in Coq. Here we want to demonstrate how to get hold of the computational content of a (non-trivial) proof by means of an extracted term, and that this term clearly represents the computationally relevant aspects of the underlying proof.

[1] http://coq.inria.fr/V8.2pl1/contribs/HigmanS.html.

3 Formalization

Why should we formalize the rather clear proof given in the previous section? There is of course the obvious reason that we want to be sure that it is correct. However, in addition we might want to get hold of its computational content. We will present this content in the form of an "extracted term", in (an extension T^+ of) Gödel's T. This term can be applied to another term representing an infinite sequence of words, and then evaluated (i.e., normalized). The normal form is a numeral determining a good finite initial segment of the input sequence.

When formalizing we of course need a theory (or formal system) where this is done. Now what features of such a theory are essential for our task? First of all, we have to get clear about (i) what "computational content" is, and (ii) where it arises. We use the Kleene-Kreisel concept of modified realizability for the former. In fact, we will have a formula expressing "the term t is a realizer for the formula A" inside our formal system. For the latter, we take it that computational content *only* arises from inductive predicates; prime examples are the Bar predicates introduced in the previous section. But then a particular aspect becomes prominent: we need "non computational" (n.c.) universal quantification [1] written $\forall^{\mathrm{nc}}$ to correctly express the type of a computational problem of the form

$$\forall^{\mathrm{nc}}_{as}(\mathrm{BarA}\ as \to A).$$

Its intended computational content is a function f mapping a witness that as is in BarA into a realizer of A. It is important that f does *not* get as as an argument.[2]

On the more technical side, we use TCF [26], a form of HA^ω extended by inductively defined predicates and n.c. logical connectives. TCF has the (Scott-Ershov) partial continuous functions as its intended model.

It is also mandatory to use a proof assistant to help with the task of formalization. We use Minlog[3] [2], which is designed to support these features.

Space does not permit to present the full formalization[4] of the constructive proof above of Higman's Lemma. We restrict ourselves to comment on some essential aspects.

Most important are of course the basic definitions of the data structures (free algebras) and predicates involved. Their formal definitions are very close to the informal ones above and do not need to be spelled out. However, already at this level computational content crops up: an inductive predicate may or may not have computational content. Examples for the former are the Bar predicates, and for the latter the GoodA predicate. It is convenient to define the GoodA predicate inductively,

[2] A similar phenomenon is addressed in Coq [5] by the so-called Set/Prop distinction. However, enriching the logic by n.c. universal quantification (and similarly n.c. implication) seems to be more flexible.

[3] See http://www.minlog-system.de.

[4] See http://www.git/minlog/examples/bar/higman.scm.

but—since it is decidable—we can also view it as a (primitive) recursive boolean valued function.

The first point in the proof above where we have to be careful with n.c. quantification is the inductive definition of BarA, with clauses

$$\begin{aligned}&\texttt{InitBarA}\colon \forall^{\mathrm{nc}}_{\preceq,as}(\mathrm{GoodA}_{\preceq}as \to \mathrm{BarA}_{\preceq}as),\\ &\texttt{GenBarA}\colon \quad \forall^{\mathrm{nc}}_{\preceq,as}(\forall_a \mathrm{BarA}_{\preceq}a * as \to \mathrm{BarA}_{\preceq}as).\end{aligned}$$

The (free) algebra of witnesses for this inductive predicate is called `treeA`. In the clause `GenBarA` the generation tree of $\mathrm{BarA}_{\preceq}as$ should have infinitely many predecessors indexed by a, hence we need $\forall_a$. However, the outside quantifier is $\forall^{\mathrm{nc}}_{\preceq,as}$, since we do not want to let the argument as be involved in the computational content of $\mathrm{BarA}_{\preceq}as$. Hence `treeA` has constructors

$$\begin{aligned}&\texttt{CInitBarA}\colon \texttt{treeA},\\ &\texttt{CGenBarA}\colon \quad (\texttt{nat} \Rightarrow \texttt{treeA}) \Rightarrow \texttt{treeA}.\end{aligned}$$

A similar (but slightly more involved) comment applies to the inductive definition of BarF. For readability we omit the dependency on $\preceq$ here. The clauses are

$$\texttt{InitBarF}\colon \forall^{\mathrm{nc}}_{ts,i}(i < \mathrm{Lh}\, ts \to \mathrm{GLT}\, (ts)_i \to \mathrm{BarF}ts),$$

and `GenBarF`:

$$\begin{aligned}\forall^{\mathrm{nc}}_{ts}(\forall_{tas,a,v}(tas = \mathrm{ProjF}\, ts \to \mathrm{Ge}_{\exists\forall}(a, \mathrm{Roots}\, tas) \to\\ \mathrm{BarF}(\mathrm{InsertF}(ts, v, a)) \to\\ \mathrm{BarF}ts).\end{aligned}$$

We need the concept of the "A-projection" of a tree t, where each rhs of a label in t is projected out. Here only the A-projection of ts (but *not* ts) is used computationally. More precisely, the predecessors of $\mathrm{BarF}ts$ are all $\mathrm{InsertF}(ts, v, a)$ for v, a with $\mathrm{Ge}_{\exists\forall}(a, \mathrm{Rights}(\mathrm{Roots}\, ts))$. To decide the latter, we need (computationally) $\mathrm{Rights}(\mathrm{Roots}\, ts)$, i.e., the A-projection of ts.

The (free) algebra of witnesses for the inductive predicate BarF is called `treeF`; its constructors are

$$\begin{aligned}&\texttt{CInitBarF}\colon \texttt{treeF},\\ &\texttt{CGenBarF}\colon \quad (\texttt{list lntree nat} \Rightarrow \texttt{nat} \Rightarrow \texttt{list nat} \Rightarrow \texttt{treeF}) \Rightarrow \texttt{treeF}.\end{aligned}$$

4 Extraction

We now spell out the computational content of some of the proofs above. Each time it comes in the form of a term (in T^+) machine extracted from a formalization of the respective proof.

When reading the extracted terms please note that lambda abstraction is displayed via square brackets; so `[n]n+m` means $\lambda_n n + m$. Our notation $\langle ws, as\rangle ts$ for the tree with root $\langle ws, as\rangle$ and list of subtrees ts is displayed as `(ws pair as)%ts`. Also types are implicit in variable names; for example, `n`, `a` both range over natural numbers. One can also use the display string for a type as a variable name of this type; for example, `treeW` is a name for a variable of type `treeW`.

4.1 BarWToGoodInit

We use typed variable names

```
f:    nat=>list nat
gw:   nat=>list nat
hwfa: list nat=>(nat=>list nat)=>nat=>nat
```

The term extracted from the proof of the proposition BarWToGoodInit is

```
[treeW]
 Rec treeW=>(nat=>list nat)=>nat=>nat)treeW([f,a]a)
 ([gw,hwfa,f,a]hwfa(f a)f(Succ a))
```

It takes some effort to understand such an extracted term. The recursion operator on `treeW` with value type `alpha` has type

```
treeW=>alpha=>((list nat=>treeW)=>(list nat=>alpha)=>alpha)
     =>alpha
```

Let `Leaf: treeW` and `Branch: (list nat=>treeW)=>treeW` be the constructors of `treeW`. Then $\Phi :=$ `(Rec treeW=>alpha)` is given by the recursion equations

$$\begin{aligned}\Phi(\texttt{Leaf}) &:= G,\\ \Phi(\texttt{Branch}(g)) &:= H(g, \lambda_v \Phi(g(v))).\end{aligned}$$

Here the value type `alpha` is `(nat=>list nat)=>nat=>nat`, and

$$\begin{aligned}G &:= \lambda_{f,a} a,\\ H(\texttt{gw}, \texttt{hwfa}) &:= \lambda_{f,a}\texttt{hwfa}(f(a), f, a+1).\end{aligned}$$

4.2 BarFNil, BarFAppd

For BarFNil we have the simple extracted term

```
CGenBarF([tas,a,v]CInitBarF)
```

For BarFAppd we use the variable names

```
g:    list lntree nat=>nat=>list nat=>treeF
htat: list lntree nat=>nat=>list nat=>treeF=>nat=>treeF
hat:  list lntree nat=>nat=>list nat=>nat=>treeF
```

Then the extracted term is

```
[wqo,treeF]
(Rec treeF=>treeF=>nat=>treeF)treeF([treeF0,a]CInitBarF)
([g,htat,treeF0]
 (Rec treeF=>nat=>treeF)treeF0([a]CInitBarF)
 ([g0,hat,a]
   CGenBarF
   ([tas,a0,v]
     [if (LargerARExAll wqo a0 Roots((Lh tas--a)init tas))
       (htat((Lh tas--a)init tas)a0 v
       [if (LargerARExAll wqo a0 Roots((Lh tas--a)rest tas))
         (g0((Lh tas--a)rest tas)a0 v)
         (CGenBarF g0)]
       a)
       (hat((Lh tas--a)rest tas)a0 v a)]))))
```

The recursion operator on `treeF` with value type `alpha` has type

```
treeF=>alpha=>
     ((list lntree nat=>nat=>list nat=>treeF)=>
      (list lntree nat=>nat=>list nat=>alpha)=>alpha)=>alpha
```

`LargerARExAl wqo a ws` means $\exists_{i<|ws|}\forall_{j<|(ws)_i|} a \succeq (ws)_{i,j}$. `treeF` has constructors `CInitBarF: treeF` and `CGenBarF: (list nat=>treeF)=>` textttttreeF. Then $\Phi := $ `(Rec treeF=>alpha)` is given by the recursion equations

$$\begin{aligned}\Phi(\texttt{CInitBarF}) &:= G,\\ \Phi(\texttt{CGenBarF}(g)) &:= H(g, \lambda_v \Phi(g(v))).\end{aligned}$$

The value type of the first recursion is `treeF=>nat=>treeF`, and

$$G \quad := \lambda_{\texttt{treeF},a}\texttt{CInitBarF},$$
$$H(g,\texttt{htat}) := \lambda_{\texttt{treeF0}}K(g,\texttt{htat},\texttt{treeF0})$$

with $K(g,\texttt{htat},\texttt{treeF0})$ given by

```
(Rec treeF=>nat=>treeF)treeF0([a]CInitBarF)
([g0,hat,a]
  CGenBarF
  ([tas,a0,v]
    [if (LargerARExAll wqo a0 Roots((Lh tas--a)init tas))
      (htat((Lh tas--a)init tas)a0 v
      [if (LargerARExAll wqo a0 Roots((Lh tas--a)rest tas))
        (g0((Lh tas--a)rest tas)a0 v)
        (CGenBarF g0)]
      a)
      (hat((Lh tas--a)rest tas)a0 v a)]))
```

The inner recursion is on `treeF` again, with value type `nat=>treeF`, and

$$G_1 \quad := \lambda_a\texttt{CInitBarF},$$
$$H_1(\texttt{g0},\texttt{hat}) := \lambda_a\texttt{CGenBarF}\ldots$$

4.3 BarFNew

With the variable names

```
gw: list nat=>treeW  hw:   list nat=>list nat=>treeF
ga: nat=>treeA       hatt: nat=>treeF=>treeF
```

we extract

```
[wqo,treeA,treeW]
 (Rec treeW=>list nat=>treeF)treeW([v]CInitBarF)
 ([gw,hw,v]
   (Rec treeA=>treeF=>treeF)treeA([treeF]CInitBarF)
   ([ga,hatt,treeF]
     (Rec treeF=>treeF)treeF CInitBarF
     ([g,g0]
       CGenBarF
       ([tas,a,v0]
         [if (LargerARExAll wqo a Roots Subtrees Head tas)
           (g0 Subtrees Head tas a v0)
           (hatt a
```

```
            (cBarFAppd wqo(hw v0(a::v))
                       (CGenBarF g)Lh Subtrees Head tas))])))
    (CGenBarF([tas,a,v0]CInitBarF)))
```

This time we have three nested recursions: an outer one on `treeW` with value type `list nat=>treeF`, then on `treeA` with value type `treeF=>treeF`, and innermost on `treeF` with value type `treeF`. This corresponds to the three elimination axioms used in the proof. Notice that the computational content `cBarFAppd` of the theorem `BarFAppd` appears as a constant inside the term.

4.4 Higman

```
[wqo,treeA]
 (Rec treeA=>treeF=>list list nat=>list lntree list nat=>treeW)
 treeA
 ([treeF,ws,tas]CInitBarW)
 ([ga,ha,treeF]
   (Rec treeF=>list list nat=>list lntree list nat=>treeW)treeF
   ([ws,tas]CInitBarW)
   ([g,h,ws,tas]
    CGenBarW
    ([v](Rec list nat=>treeW)v(CGenBarW([v0]CInitBarW))
     ([a,v0,treeW]
      [if (LargerAR wqo a(BSeq wqo Heads ws))
  (h tas a v0((a::v0)::ws)(InsertAF wqo tas a))
  (ha a (cBarFAppd wqo(cBarFNew wqo treeA treeW a:)
   (CGenBarF g)Lh tas)
      ((a::v0)::ws)
      ((a: %(Nil lntree list nat))::tas))])))))
 (CGenBarF([tas,a,v]CInitBarF))
 (Nil list nat)
 (Nil lntree list nat)
```

4.5 Experiments

To run the extracted terms we need to "animate" the theorems involved. This means that the constant denoting their computational content (e.g., `cBarFAppd` for the theorem `BarFAppd`) unfolds into the term extracted from the proof of the theorem. Then for an arbitrary infinite sequence extending e.g. the example in Sect. 2 we obtain the expected good initial segment.

In more detail, we first have to animate the computationally relevant propositions in the proof given above of Higman's Lemma. Then we need to prove and animate lemmas relating to the particular relation `NatLe`:

$$\begin{array}{ll}\texttt{BarNatLeAppdOne}: & \forall_{i,m,as}(i + \mathrm{Lh}(as) = m + 1 \to \mathrm{BarA}_{\leq}(as\#[m])),\\ \texttt{BarANilNatLe}: & \mathrm{BarA}_{\leq}[],\\ \texttt{HigmanNatLe}: & \mathrm{BarW}_{\leq}[].\end{array}$$

Using these we can prove the final proposition

$$\texttt{GoodWInitNatLe}: \forall_f \exists_n \mathrm{GoodW}_{\leq}(\mathrm{Rev}(\bar{f}n)).$$

Let `neterm` be the result of normalizing the term extracted from this proof. Next we provide an infinite sequence (extending the example in Sect. 2), e.g. in the form of a program constant:

```
(add-program-constant "Seq" (py "nat=>list nat"))
(add-computation-rules
"Seq 0" "5::2:"
"Seq 1" "2::8:"
"Seq 2" "4::2::1:"
"Seq 3" "6::9:"
"Seq 4" "3::5:"
"Seq(Succ(Succ(Succ(Succ(Succ n)))))" "0:")
```

Finally we run the our normalized extracted term by evaluating

```
(pp (nt (mk-term-in-app-form neterm (pt "Seq"))))
```

(Here `nt` means "normalize term" and `pt` means "parse term"). The result is 4, the length of a good initial segment of our infinite sequence.

5 Related Work: Other Proofs of Higman's Lemma

As mentioned at the beginning of Sect. 2, our constructive proof of Higman's Lemma does not need transitivity; it works for arbitrary almost full relations. However, in the following discussion we disregard this fine point and assume that the underlying relation is a well-quasiorder. This will make it easier to compare different proofs in the literature.

There are quite a number of constructive proofs of Higman's Lemma, thus the natural question arises: are they all different? The number of proofs is due to the fact that researchers from different areas, algebra, proof theory, constructive mathematics, term rewriting, to name a few, became interested in Higman's Lemma. In addition, there are various formulations of a well-quasiorder which include different proof principles. These are for instance proofs using ordinal notation systems and transfinite induction as used in [24, 25] or inductively defined predicates and structural induction as used in [9, 18, 23]. Below we argue that these proofs are the same from a computational point of view.

The proof theoretic strength of Higman's Lemma is that of Peano Arithmetic, i.e. ϵ_0, as was shown in [12] using the constructive proof of [25]. Speaking in terms of Reverse Mathematics, Higman's Lemma can be proven in the theory ACA_0. In term rewriting theory, Higman's Lemma and its generalization to trees, 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 [4], 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_α, 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 $H_{\omega^{\omega^{|A|}}}$ [4, 31]. This bound is optimal since there are term rewriting systems which "reach" these bounds [32].

5.1 Equivalent Formulations of a Well-Quasiorder

We define the maximal ordertype of a well-quasiorder $(A, \preceq)$ as the supremum of the ordertypes of all extensions of $(A, \preceq)$ to a linear order. Equivalently, in a more constructive manner, the maximal ordertype can be defined by the height of the tree of all bad sequences $(\mathrm{Bad}_\preceq)$ with elements in A. A reification of a quasi order $(A, \preceq)$ into a wellordering $(\sigma, <)$ is a map

$$r : \mathrm{Bad}_\preceq \to \sigma,$$

such that for all $a{*}as \in \mathrm{Bad}_\preceq$ we have $r(a{*}as) < r(as)$. On the set $\mathrm{Bad}_\preceq$ of bad sequences in A we define a relation $\ll_A$ by $as' \ll_A as$ iff $as' = a{*}as$ for some $a \in A$. The accessible part of the relation $\ll_A \subseteq \mathrm{Bad}_\preceq \times \mathrm{Bad}_\preceq$ is inductively given by the rule

$$\frac{\forall_{as'}(as' \ll_A as \to \mathrm{acc}_{\ll_A} as')}{\mathrm{acc}_{\ll_A} as}$$

It is obvious that the following are equivalent for a quasiorder $(A, \preceq)$:

(i) $(A, \preceq)$ is a well-quasiorder (i.e., $\mathrm{Wqo}(A, \preceq)$).
(ii) $(A, \preceq)$ has a maximal ordertype.
(iii) There is a reification of $(A, \preceq)$ into a wellorder.
(iv) $(\mathrm{Bad}_{\preceq}, \ll_A)$ is wellfounded, i.e., $\mathrm{acc}_{\ll_A}[\,]$.
(v) $\mathrm{BarA}_{\preceq}[\,]$.

5.2 A Generic Proof of Higman's Lemma

In the following we sketch a generic proof of

$$\mathrm{Wqo}(A, \preceq) \to \mathrm{Wqo}(A^*, \preceq^*).$$

which differs from the proof presented in the earlier sections. We start by choosing a characterization of a well-quasiorder, either using ordinal notations ((ii) or (iii)) or inductive definitions ((iv) or (v)). (Note that in the latter case we need to generalize the statement; for instance, in (iv) we prove more generally $\mathrm{acc}_{\ll_A} as \to \mathrm{acc}_{\ll_{(A_{as})^*}}[\,]$ and use this proof with $as = [\,]$, and in the proof below instead of $A_{[a]}$ we use everywhere A_{a*as}, etc.). Here we define A_{as} as the set of all elements that extend as badly, i.e. $\forall_i as_i \not\preceq a$. Similarly, we define $A_{[a]}$ to be the set of all elements b such that $a \not\preceq_A b$. Assume that for our choice of characterization we are able to prove (with the obivous extension of $\preceq$ to $\cup$ and $\times$):

$$\begin{aligned}
&(a)\ \ \forall a\ \mathrm{Wqo}(A_{[a]}, \preceq) \to \mathrm{Wqo}(A, \preceq),\\
&(b)\ \ A \subseteq B \to \mathrm{Wqo}(A, \preceq) \to \mathrm{Wqo}(B, \preceq),\\
&(c)\ \ \mathrm{Wqo}(A) \wedge \mathrm{Wqo}(B) \to \mathrm{Wqo}(A \cup B),\\
&(d)\ \ \mathrm{Wqo}(A) \wedge \mathrm{Wqo}(B) \to \mathrm{Wqo}(A \times B).
\end{aligned}$$

Assume $\mathrm{Wqo}(\preceq)$. We proceed to prove Higman's Lemma by using either structural induction or transfinite induction, depending on our choice. From the induction hypothesis we get

$$\mathrm{Wqo}(A_{[a]}) \to \mathrm{Wqo}({A_{[a]}}^*). \tag{1}$$

By (a) it suffices to prove $\forall_v \mathrm{Wqo}(A^*_{[v]})$. Let $v = [a_1, \ldots, a_n]$. The main combinatorial idea is now contained in the following statement

$$(A)^*_{[[a_1,\ldots,a_n]]} \subseteq \bigcup\{(A_{[a_1]})^* \times A \times (A_{[a_2]})^* \times \cdots \times A \times (A_{[a_l]})^* \mid l < n\} \tag{2}$$

which holds by a simple combinatorial argument. Using (b) we are done once we have shown

$$\mathrm{Wqo}(\bigcup\{(A_{[a_1]})^* \times A \times (A_{[a_2]})^* \times \cdots \times A \times (A_{[a_l]})^* \mid l < n\}).$$

But this follows immediately from (c), (d) and (1).

Remark Instantiated versions of this proof, using characterizations (ii), (iii), (iv) or (v) of a well-quasiorder, can be found in the following articles: (ii) is used by de Jongh and Parikh [8] and Schmidt [24]. (iii) is used in the proof by Schütte and Simpson [25] (and Hasegawa [13]) (and is the characterization which is most promising in terms of generalizations beyond Kruskal's Theorem). (iv) has been used by Fridlender [9], using an acc notation. His proof is a reformulation of the proof by Richman and Stolzenberg [23]. To a less formal extent this characterization is also used in [18], where also structural induction and a similar construction describing the space to which a sequence can be extended badly are used. Characterization (v): the proof in [9] can be easily reformulated using (v). Fridlender [10] gives a variant where he does not need the decidability of $\preceq_A$. His proof is a type theoretic version of an intuitionistic proof by Veldman, later published in [34].

Finally, the proof of [18] forms the basis of the formalization and proof of Higman's Lemma in [16], in ACL2. Their work however starts with a program solving the problem, and then proving its properties rather than extracting the program from the proof.

Remark Higman's Lemma extends naturally to Kruskal's Theorem, the corresponding statement for trees. Constructive proofs of Kruskal's Theorem have been given by Schmidt [24] using characterization (ii), by Rathjen and Weiermann [22] and Hasegawa [13] using (iii), and in [27] using (iv). Finally, also Goubault-Larrecq's proof [11] which generalizes the proof in [18] falls under this category.

It remains to compare how the computational content behind this generic proof of Higman Lemma is related to the constructive proof given in this paper. Although we have not yet formalized the proof above, it is quite obvious that the construction, in particular Eq. (2) differs from the construction in our proof, and therefore would result in a different algorithm.

6 Conclusion and Further Work

We presented and formalized a constructive proof of Higman's Lemma that contains the same combinatorial idea as Nash-Williams' indirect proof, and extracted and discussed its inherent program in detail. We also argued that a number of constructive proofs of Higman's Lemma are based on a combinatorial idea different from ours. It is still open to make that claim formal, i.e. to formalize the proof presented in the previous section, and compare the resulting program with our extracted program.

Similarly, there are a number of formalizations of Nash-Williams' classical proof as mentioned in the introduction. It would be worthwhile to confirm that they, in principle, lead to the same algorithm, which also corresponds to the algorithm in our extracted program.

Equally interesting is the question which of the discussed proofs are most suitable for applications such as termination of string- and term rewriting systems, see e.g. [11, 30, 35] for recent discussions on applications to termination proofs. A particularly promising application has been given in [20]. It will be worth checking how our alternative proof of Higman's Lemma and its extracted program can be utilized with regard to these applications or further generalizations.

Acknowledgments We would like to thank Daniel Fridlender and Iosif Petrakis for helpful contributions and comments.

References

1. U. Berger, Program extraction from normalization proofs, in *Typed Lambda Calculi and Applications*, ed. by M. Bezem, J. Groote, LNCS, vol. 664, (Springer, Berlin, 1993), pp. 91–106
2. U. Berger, K. Miyamoto, H. Schwichtenberg, M. Seisenberger, Minlog–A Tool for Program Extraction Supporting Algebras and Coalgebras, in *Algebra and Coalgebra in Computer Science*, ed. by A. Corradini, B. Klin, C. Cîrstea, CALCO'11, LNCS, vol. 6859, (Springer, Berlin, 2011), pp. 393–399
3. S. Berghofer, A constructive proof of Higman's lemma in Isabelle, *Types for Proofs and Programs*. Lecture Notes in Computer Science, vol. 3085 (Springer, Berlin, 2004), pp. 66–82
4. E.A. Cichon, E.T. Bittar, Ordinal recursive bounds for Higman's theorem. Theor. Comput. Sci. **201**(1–2), 63–84 (1998)
5. Coq Development Team. The Coq Proof Assistant Reference Manual—Version 8.2. Inria (2009)
6. T. Coquand. A proof of Higman's lemma by structural induction. Unpublished Manuscript, April 1993
7. T. Coquand, D. Fridlender, A proof of Higman's lemma by structural induction. Unpublished Manuscript (1993), http://www.cse.chalmers.se/~coquand/open1.ps, Nov 1993
8. D. de Jongh, R. Parikh, Well partial orderings and their order types. Indagationes Mathematicae **14**, 195–207 (1977)
9. D. Fridlender, Ramsey's Theorem in Type Theory. Licentiate Thesis, Chalmers University of Technology and University of Göteburg, 1993
10. D. Fridlender. Higman's Lemma in Type Theory. Ph.D. thesis, Chalmers University of Technology, Göteborg, 1997
11. J. Goubault-Larrecq, A constructive proof of the topological Kruskal theorem, in *Mathematical foundations of computer science 2013*, ed. by K. Chatterjee, J. Sgall, 38th international symposium, MFCS 2013, LNCS, vol. 8087, (Springer, Berlin, 2013), pp. 22–41
12. J.-Y. Girard, *Proof Theory and Logical Complexity* (Bibliopolis, Napoli, 1987)
13. R. Hasegawa, Well-ordering of algebra and Kruskal's theorem, in *Logic, Language and Computation*, ed. by N.D. Jones, M. Hagiya, M. Sato, Festschrift in Honor of Satoru Takasu. Lecture Notes in Computer Science, vol. 792, (Springer, Berlin, 1994), pp. 133–172
14. H. Herbelin, A program from an A-translated impredicative proof of Higman's lemma (1994), http://www.lix.polytechnique.fr/coq/pylons/contribs/files/HigmanNW/trunk/HigmanNW.Higman.html

15. G. Higman, Ordering by divisibility in abstract algebras. Proc. London Math. Soc. **2**, 326–336 (1952)
16. F.-J. Martín-Mateos, J.-L. Ruiz-Reina, J.-A. Alonso, M.-J. Hidalgo, Proof pearl: a formal proof of Higman's lemma in ACL2. J. Autom. Reasoning **47**(3), 229–250 (2011)
17. C. Murthy, Extracting constructive content from classical proofs. Technical Report 90–1151, Department of Computer Science, Ph.D. thesis. Cornell University, Ithaca, New York, 1990
18. C.R. Murthy, J.R. Russell, A constructive proof of Higman's lemma, in *Proceedings of the Sixth Symposium on Logic in Computer Science*, pp. 257–267 (1990)
19. C. Nash-Williams, On well-quasi-ordering finite trees. Proc. Cambridge Phil. Soc. **59**, 833–835 (1963)
20. M. Ogawa, Generation of a linear time query processing algorithm based on well-quasi-orders, in *TACS '01 Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software*, LNCS, vol. 2215 (Springer, Berlin, 2001), pp. 283–297
21. T. Powell, Applying Gödel's Dialectica interpretation to obtain a constructive proof of Higman's lemma, in *Proceedings of Classical Logic and Computation (CLAC'12)*, EPTCS, vol. 97 pp. 49–62 (2012)
22. M. Rathjen, A. Weiermann, Proof-theoretic investigations on Kruskal's theorem. Ann. Pure a. Appl. Logic **60**, 49–88 (1993)
23. F. Richman, G. Stolzenberg, Well quasi-ordered sets. Adv. Math. **97**, 145–153 (1993)
24. D. Schmidt, *Well-Orderings and Their Maximal Order Types* (Mathematisches Institut der Universität, Habilitationsschrift, Heidelberg, 1979)
25. K. Schütte, S.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)
26. H. Schwichtenberg, S.S. Wainer, *Proofs and Computations* (Association for Symbolic Logic (Cambridge University Press, Cambridge, Perspectives in Logic, 2012)
27. M. Seisenberger, Kruskal's tree theorem in a constructive theory of inductive definitions, in *Reuniting the Antipodes—Constructive and Nonstandard Views of the Continuum*, ed. by P. Schuster, U. Berger, H. Osswald, Synthese Library, vol. 306 (Kluwer Academic Publisher, Dordrecht, 2001), pp. 241–255
28. M. Seisenberger, *An inductive version of Nash-Williams Minimal-Bad-Sequence argument for Higman's lemma, in Types for Proofs and Programs, LNCS*, vol. 2277 (Springer, Berlin, 2001)
29. M. Seisenberger, On the Constructive Content of Proofs. Ph.D. thesis, Mathematisches Institut der Universität München, 2003
30. C. Sternagel, Certified Kruskal's tree theorem. J. Formalized Reasoning **7**(1), 45–62 (2014)
31. H. Touzet, Propriétés combinatoires pour la terminaison de systèmes de réécriture. Ph.D. thesis, Université Henri Poincaré, Nancy, 1997
32. H. Touzet, A characterisation of multiply recursive functions with Higman's lemma. Inf. Comput. **178**, 534–544 (2002)
33. V. Veldman, M. Bezem, Ramsey's theorem and the pigeonhole principle in intuitionistic mathematics. J. London Math. Soc. **47**, 193–211 (1993)
34. W. Veldman, An intuitionistic proof of Kruskal's theorem. Arch. for Math. Logic **43**(2), 215–264 (2004)
35. D. Vytiniotis, T. Coquand, D. Wahlstedt, Stop when you are almost-full–Adventures, in *constructive termination, in Proceedings, ITP 2012*, ed. by L. Beringer, A. Felty, LNCS, vol. 7406, (Springer Verlag, Berlin, Heidelberg, New York, 2012), pp. 250–265