Higman's Lemma and its computational content

Helmut Schwichtenberg
(j.w.w. Monika Seisenberger and Franziskus Wiesnet)

Mathematisches Institut, LMU, München

Schloß Dagstuhl, Januar 2016

Higman's lemma

$(A, \preceq)$ is a well-quasiorder (wqo) if

- $\preceq$ is transitive, and
- every infinite sequence in A is good, i.e., $\forall_{(a_i)_{i<\omega}} \exists_{i,j} (i < j \wedge a_i \preceq a_j)$.

Call $[a_1, \ldots, a_n]$ embeddable ($\preceq^*$) in $[b_1, \ldots, b_m]$ if there exists a strictly increasing map $f\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that $a_i \preceq b_{f(i)}$ for all $i \in \{1, \ldots n\}$.

Lemma (Higman)

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

Nash-Williams' proof

(1) Assume that there is a bad sequence of words in A^*.

(2) Choose a minimal bad one, i.e. $(w_i)_{i<\omega}$ s. t. $w_0, \dots, w_n$ starts an infinite bad sequence, but $w_0, \dots, w_{n-1}, v$ does not, where v is a proper end segment of w_n.

(3) $w_i = a_i * v_i$. There is 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 $w_0, \dots, w_{\kappa_0 - 1}, v_{\kappa_0}, v_{\kappa_1}, \dots$.

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

Computational content of Nash-Williams' proof

- Murthy (1990) applied Friedman's A-translation to the classical proof (in NuPRL). Huge resulting program.
- Seisenberger (2003) applied a refined version of the A-translation (in Minlog), not eliminating the axiom of classical dependent choice. Much smaller extracted program.
- Powell (2012) applied Gödel's Dialectica Interpretation.
- Sternagel (2014): formalization of Higman and Kruskal (in Isabelle). No extraction.

Constructive reformulation of Nash-Williams' proof

- Coquand & Fridlender (1993): for a $\{0, 1\}$-alphabet. Formalizations: Fridlender (in Agda), Seisenberger (in Minlog), Berghofer (in Isabelle and Coq).
- General case: Seisenberger (2001). Much more elaborate.
- Formalization by Delobel (in Coq). No extraction. Problem (pointed out by Fridlender): $\mathrm{acc}_{\preceq}$-based definition of well-quasiorders results in a brute force search. To let the proof determine the case, one needs a "positive" formulation of well-quasiorders, with 2 rules.
- Here: formalization and extraction for the general case.

Constructive reformulation of Nash-Williams' proof (ctd.)

Inductively define $\mathrm{BarA} \subseteq A^*$, by

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

$\mathrm{BarW}\, ws$ is defined similarly, using the corresponding $\mathrm{GoodW}\, ws$.

(1) Prove inductively "$\mathrm{BarA}([]) \to \mathrm{BarW}([])$".

(2) Replace minimality argument by structural induction on ws.

(3)

Given $ws = [w_n, \ldots, w_0]$ s.t. $w_i{=}a_i{*}v_i$. Consider subsequences $a_{\kappa_l} \succeq \cdots \succeq a_{\kappa_0}$ of maximal length & 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". Define $\mathrm{Forest}(ws)$ with nodes labelled in $A^{**} \times A^*$.

- In the produced forest the right-hand components of each node form such a descending subsequence $[a_{\kappa_i}, \ldots, a_{\kappa_0}]$. Corresponding left component: $[v_{\kappa_i}, \ldots, v_{\kappa_0}, w_{\kappa_0 - 1}, \ldots, w_0]$.
- If we extend ws 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.
- Idea: if in $\mathrm{Forest}(ws)$ 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.

Why formalize?

- Correctness.
- Get hold of the computational content of a (non-trivial) proof by means of its extracted term.

Underlying theory: TCF, with Kleene-Kreisel modified realizability.

- Computational content only arises from inductive predicates.
- Need "non computational" (n.c.) universal quantification (Berger 1993) written $\forall^{\mathrm{nc}}$ to correctly express the type of a computational problem of the form

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

Relation of TCF to type theory

- Main difference: partial functionals are first class citizens.
- "Logic enriched": Formulas and types kept separate.
- Minimal logic: $\to, \forall$ only. $x = y$ (Leibniz equality), $\exists$, $\vee$, $\wedge$ inductively defined (Russell, Martin-Löf).
- $\bot := (\mathrm{False} = \mathrm{True})$. Ex-falso-quodlibet: $\bot \to A$ provable.
- "Decorations" $\to^{\mathrm{nc}}$, $\forall^{\mathrm{nc}}$ (i) allow abstract theory (ii) remove unused data.

BarA

Inductively define $\mathrm{BarA} \subseteq A^*$, by the clauses

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

GoodA n.c. inductive predicate.

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

Constructors of $\mathbf{T}_A$:

$$\texttt{CInitBarA}\colon \mathbf{T}_A,$$
$$\texttt{CGenBarA}\colon \quad (\mathbf{N} \to \mathbf{T}_A) \to \mathbf{T}_A.$$

BarF

Inductively define BarF, by the clauses

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

$$\begin{aligned}\texttt{GenBarF}\colon \forall^{\mathrm{nc}}_{\preceq,ts}(\forall_{tas,a,v}(tas = \mathrm{ProjF}(ts) \to \mathrm{LA}_{\preceq}(a, \mathrm{Roots}(tas)) \to \\ \mathrm{BarF}_{\preceq}(\mathrm{InsertF}_{\preceq}(ts, v, a)) \to \\ \mathrm{BarF}_{\preceq}(ts)).\end{aligned}$$

- Need "A-projection" of a tree $t \in T(A^{**} \times A^{*})$, where each head of the rhs of a label in t is projected out.
- Only the A-projection of ts (not ts) is used computationally.
- The predecessors of $\mathrm{BarF}_{\preceq}(ts)$ are all $\mathrm{InsertF}_{\preceq}(ts, v, a)$ for v, a with $\mathrm{LA}_{\preceq}(a, \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(ts))))$.
- To decide the latter, we need (computationally) $\mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(ts)))$, i.e., the A-projection of ts.

Witnesses for BarF

Recall

$$\begin{aligned}
\texttt{InitBarF}\colon & \forall^{\mathrm{nc}}_{\preceq, ts, i}(i < \mathrm{Lh}(ts) \to \mathrm{GLT}_{\preceq}(ts)_i \to \mathrm{BarF}_{\preceq}(ts)), \\
\texttt{GenBarF}\colon & \forall^{\mathrm{nc}}_{\preceq, ts}(\forall_{tas, a, v}(tas = \mathrm{ProjF}(ts) \to \mathrm{LA}_{\preceq}(a, \mathrm{Roots}(tas)) \to \\
& \qquad \mathrm{BarF}_{\preceq}(\mathrm{InsertF}_{\preceq}(ts, v, a)) \to \\
& \quad \mathrm{BarF}_{\preceq}(ts)).
\end{aligned}$$

The (free) algebra of witnesses for the inductive predicate BarF is called $\mathbf{T}_F$ (or `treeF`). Constructors:

$$\begin{aligned}
\texttt{CInitBarF}\colon & \mathbf{T}_F, \\
\texttt{CGenBarF}\colon & (\mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F) \to \mathbf{T}_F.
\end{aligned}$$

BarFAppdAux

```
all wqo allnc ts(BarF wqo ts ->
  allnc ss(BarF wqo ss ->
  all m(m=Lh ss -> BarF wqo(ts++ss))))
```

BarFNew

```
all wqo(BarA wqo(Nil nat) ->
  allnc ws(BarW wqo ws ->
  all as BarF wqo((NewTree(ws pair as)):)))
```

HigmanAux

```
all wqo(all a,b,c(wqo a b -> wqo b c -> wqo a c) ->
  BarA wqo(Nil nat) ->
  allnc as(BarA wqo as ->
  allnc ts(BarF wqo ts ->
  all ws(Adm ws -> BSeq wqo(Heads ws)=as ->
  all tas(tas=ProjF ts ->
  Forest wqo ws=ts -> BarW wqo ws)))))
```

BarFNew

Inserting new nodes in a singleton forest finally makes it good:

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

Proof. $\mathsf{Ind}_1(\mathrm{BarW})$. 1.1. $\mathrm{GoodW}(ws_0)$. Easy. 1.2. Assume

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

Let $as_0 \in A$. Goal: $\mathrm{BarF}[\mathrm{Newtree}\,\langle ws, as_0\rangle]$. Instead we show more generally that this assertion holds for all t with $\mathrm{Root}\,(t) = \langle ws, as_0\rangle$ 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

$$\forall^{\mathrm{nc}}_{as}(\mathrm{BarA}(as) \to \neg\mathrm{GoodA}(as) \to$$
$$\forall^{\mathrm{nc}}_{ts}(\mathrm{BarF}(ts) \to as = \mathrm{Heads}(\mathrm{Rights}(\mathrm{Roots}(ts))) \to \mathrm{BarF}[\langle ws, as_0\rangle ts])).$$

...

```
[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 (LargerAR 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)))
```

with gw: $\mathbf{L}(\mathbf{N}) \to \mathbf{T}_W$, hw: $\mathbf{L}(\mathbf{N}) \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F$, ga: $\mathbf{N} \to \mathbf{T}_A$, hatt: $\mathbf{N} \to \mathbf{T}_F \to \mathbf{T}_F$, g: $\mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F$.
cBarFAppd abbreviates the extracted term of BarFAppd.

Structure of the extracted term

Three nested recursions:

- an outer one on $\mathbf{T}_W$ with value type $\mathbf{L}(\mathbf{N}) \to \mathbf{T}_F$,
- then on $\mathbf{T}_A$ with value type $\mathbf{T}_F \to \mathbf{T}_F$,
- and innermost on $\mathbf{T}_F$ with value type $\mathbf{T}_F$.

This corresponds to the three elimination axioms used in the proof.

Recall the constructors of $\mathbf{T}_F$ (or `treeF`):

$$\begin{aligned} &\texttt{CInitBarF}\colon \mathbf{T}_F, \\ &\texttt{CGenBarF}\colon \quad (\mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F) \to \mathbf{T}_F. \end{aligned}$$

$\Phi :=$ `(Rec treeF=>alpha)`, the recursion operator on $\mathbf{T}_F$ with value type α, has type

$$\begin{aligned} \mathbf{T}_F \to \alpha \to (\mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F) \to \\ (\mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \alpha) \to \alpha. \end{aligned}$$

It is given by recursion equations

$$\begin{aligned} \Phi(\texttt{CInitBarF}) \quad &:= G, \\ \Phi(\texttt{CGenBarF}(g)) &:= H(g, \lambda_{\vec{x}} \Phi(g(\vec{x}\,))). \end{aligned}$$

with $g\colon \mathbf{L}(\mathbf{T}_N) \to \mathbf{N} \to \mathbf{L}(\mathbf{N}) \to \mathbf{T}_F$.

An experiment

- To run the extracted terms we need to "animate" the lemmas involved.
- Final proposition: $\forall_f \exists_n \mathrm{GoodW}_{\leq}(\mathrm{Rev}(\bar{f}(n)))$.
- Let `neterm` be the result of normalizing the term extracted from this proof.

```
(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:")

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

Result: 4.

Conclusion, further work

- Formalized a constructive proof of Higman's Lemma that contains the same combinatorial idea as Nash-Williams' indirect proof. Extracted and discussed its inherent algorithm.
- Many other constructive proofs of Higman's Lemma are based on a different combinatorial idea: De Jongh & Parikh (1977), Schmidt (1979), Schütte & Simpson (1985), Richman & Stolzenberg (1993), Hasegawa (1994), Veldman (2004). Open: comparison with the algorithm here.
- Explore applications to termination proofs for string- and term rewriting systems. Cf. Ogawa (2001), Vytiniotis & Coquand & Wahlstedt (2012), Goubault & Larrecq (2013), Sternagel (2014).