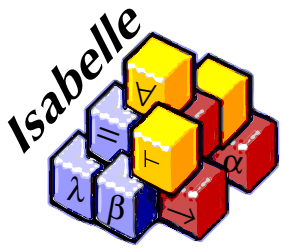
Program extraction from a proof of weak normalization

Stefan Berghofer
Institut für Informatik, TU München



Background

Strong normalization

- Every reduction sequence starting from a well-typed term is finite
- Formalized using accessible part
- Extra work needed to obtain a normalization algorithm

$$e \vdash t : T \Longrightarrow t \in acc\ (\rightarrow_\beta{}^{-1})$$

Weak normalization

- Every well-typed term has a normal form
- Proof corresponds to a specific evaluation strategy
- Can directly extract normalization algorithm from proof

$$e \vdash t : T \Longrightarrow \exists\, t'.\ t \rightarrow_\beta{}^* t' \wedge t' \in NF$$

Problems

- Normalization proofs usually based on strong computability predicates (W. Tait, JSL 1967)

$$e \vdash s : T \Rightarrow U \wedge (\forall t.\ e \vdash t : T \wedge SC\ e\ t\ T \longrightarrow SC\ e\ U\ (s \circ t)) \longrightarrow SC\ e\ (T \Rightarrow U)\ s$$

- SC not admissible as an inductive definition, because SC occurs negatively
- SC must be defined by recursion on type expression
  - Requires strong elimination (i.e. definition of predicates by recursion)
  - May give rise to extracted programs using dependent types

Alternative approach (R. Matthes and F. Joachimski, AML 2003)

- Inductive characterization of normal forms
- Normalization proof is by induction on normal forms and type expressions
- Particularly well-suited for program extraction

Related work

- T. Altenkirch (TLCA 1993): A Formalization of the Strong Normalization Proof for System F in LEGO
- U. Berger (TLCA 1993): Extraction of a normalization by evaluation algorithm from a strong normalization proof in MINLOG
- R. Pollack (1994): The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions
- B. Barras, B. Werner: Coq in Coq
- C. Urban (2004): Formalization of a Tait-style strong normalization proof in Isabelle/HOL using nominal techniques
- T. Coquand (2004)
- H. Schwichtenberg (2004)

λ-terms with de Bruijn indices

datatype $dB = Var\ nat \mid dB \circ dB \mid Abs\ dB$

Notation

$t \circ\circ [t_1, \ldots, t_n]$	$=$	$t \circ t_1 \circ \cdots \circ t_n$
$t \uparrow i$	:	increment all indices $\geq i$
$t[u/i]$	:	substitute u for variable i in t
$t \rightarrow_\beta t'$	:	β reduction

Simple types

datatype $type = Atom\ nat \mid type \Rightarrow type$

Notation: $[T_1, \ldots, T_n] \Rrightarrow T \quad = \quad T_1 \Rightarrow \cdots \Rightarrow T_n \Rightarrow T$

Type environments

- Modelled as functions from nat to $type$
- Inserting type T in environment e at position i:
  $e\langle i{:}T\rangle \equiv \lambda j.\ if\ j < i\ then\ e\ j\ else\ if\ j = i\ then\ T\ else\ e\ (j - 1)$

Typing judgement

$VarT{:}\ e\ x = T \Longrightarrow e \vdash Var\ x : T$
$AbsT{:}\ e\langle 0{:}T\rangle \vdash t : U \Longrightarrow e \vdash Abs\ t : (T \Rightarrow U)$
$AppT{:}\ e \vdash s : T \Rightarrow U \Longrightarrow e \vdash t : T \Longrightarrow e \vdash (s \circ t) : U$

Normal forms

$AppN\colon listall\ (\lambda t.\ t \in NF)\ ts \Longrightarrow Var\ x \circ\circ ts \in NF$
$AbsN\colon t \in NF \Longrightarrow Abs\ t \in NF$

where

$listall\ P\ xs \equiv (\forall i.\ i < length\ xs \longrightarrow P\ (xs\ !\ i))$

Computational content

datatype $NFT = Dummy \mid AppN\ (dB\ list)\ nat\ (nat \Rightarrow NFT) \mid AbsN\ dB\ NFT$

Type NFT used by extracted program to represent normal forms.

Main theorems

Well-typed substitution preserves existence of normal forms

lemma $subst\text{-}type\text{-}NF$: $t \in NF \Longrightarrow e\langle i{:}U\rangle \vdash t : T \Longrightarrow u \in NF \Longrightarrow e \vdash u : U \Longrightarrow \exists\, t'.\ t[u/i] \rightarrow_\beta{}^* t' \wedge t' \in NF$

Proof: Main induction on T, side induction on $t \in NF$

Type of program:

$$dB \Rightarrow (nat \Rightarrow type) \Rightarrow nat \Rightarrow type \Rightarrow type \Rightarrow dB \Rightarrow NFT \Rightarrow NFT \Rightarrow dB \times NFT$$

Well-typed terms have a normal form

theorem $type\text{-}NF$: $e \vdash_R t : T \Longrightarrow \exists\, t'.\ t \rightarrow_\beta{}^* t' \wedge t' \in NF$

Proof: Induction on $e \vdash_R t : T$

Type of program: $rtypingT \Rightarrow dB \times NFT$

Correctness theorem:

$$(x,\ e,\ t,\ T) \in rtypingR \Longrightarrow t \rightarrow_\beta{}^* fst\ (type\text{-}NF\ x) \wedge fst\ (type\text{-}NF\ x) \in NF$$

Note: Typing derivation computationally irrelevant / relevant in proof of $subst\text{-}type\text{-}NF$ / $type\text{-}NF$

Proof of the application case

How to normalize $(\underbrace{Var\ i}_{T' \Rightarrow\ Ts\ \Rrightarrow\ T} \circ a \circ\circ as)[u/i] \;=\; u \circ a[u/i] \circ\circ as[u/i]$

1. Substitute and normalize tail of argument list (by side induction) $\quad as[u/i] \to_\beta^* as'$
2. Substitute and normalize first argument (by side induction) $\quad a[u/i] \to_\beta^* a'$
3. Apply head term to result term, normalize again (by main induction) $\quad u \circ \underbrace{a'}_{T'} \to_\beta^* ua$
4. Apply result term to normalized argument list, normalize again (by main induction) $\quad \underbrace{ua}_{Ts\ \Rrightarrow\ T} \circ\circ as' \to_\beta^* r$

Computational content of lemma

$$
\begin{array}{l}
\mathit{subst\text{-}type\text{-}NF} \equiv \lambda x\ xa\ xb\ xc\ xd\ xe\ H\ Ha. \\
\quad \mathit{type\text{-}induct\text{-}P}\ xc\ (\lambda x\ H2\ H2a\ xa\ xb\ xc\ xd\ xe\ H. \\
\qquad \mathit{NFT\text{-}rec}\ \mathit{arbitrary} \qquad\qquad \text{dummy case} \\
\qquad (\lambda ts\ xa\ xaa\ r\ xb\ xc\ xd\ xe\ H.\ \mathit{var\text{-}app\text{-}typesE\text{-}P}\ (xb\langle xe{:}x\rangle)\ xa\ ts \qquad \text{application case} \\
\qquad\quad (\lambda Us.\ \mathit{case}\ \mathit{nat\text{-}eq\text{-}dec}\ xa\ xe\ \mathit{of} \qquad \text{head variable affected} \\
\qquad\qquad \mathit{Left} \Rightarrow \mathit{case}\ ts\ \mathit{of}\ [] \Rightarrow (xd,\ H) \qquad \text{empty argument list} \\
\qquad\qquad\ \mid a\ \#\ \mathit{list} \Rightarrow \qquad \text{non-empty argument list} \\
\qquad\qquad\quad \mathit{case}\ Us\ \mathit{of}\ [] \Rightarrow \mathit{arbitrary} \qquad \text{ill-typed term} \\
\qquad\qquad\quad \mid T''\ \#\ Ts \Rightarrow \\
\qquad\qquad\qquad \mathit{let}\ (x,\ y) = \mathit{norm\text{-}list}\ (\lambda t.\ t \uparrow 0)\ xd\ xb\ xe\ \mathit{list}\ Ts\ (\lambda t.\ \mathit{lift\text{-}NF}\ 0)\ H \\
\qquad\qquad\qquad\quad (\mathit{listall\text{-}conj2\text{-}P\text{-}Q}\ \mathit{list}\ (\lambda i.\ (xaa\ (\mathit{Suc}\ i),\ r\ (\mathit{Suc}\ i)))); \\
\qquad\qquad\qquad (xa,\ ya) = \mathit{snd}\ (xaa\ 0,\ r\ 0)\ xb\ T''\ xd\ xe\ H; \\
\qquad\qquad\qquad (xd,\ yb) = \mathit{app\text{-}Var\text{-}NF}\ 0\ (\mathit{lift\text{-}NF}\ 0\ H); \\
\qquad\qquad\qquad (xa,\ ya) = H2\ T''\ (Ts \Rrightarrow xc)\ xd\ xb\ (Ts \Rrightarrow xc)\ xa\ 0\ yb\ ya; \\
\qquad\qquad\qquad (x,\ y) = H2a\ T''\ (Ts \Rrightarrow xc)\ (\mathit{Var}\ 0 \circ\circ \mathit{map}\ (\lambda t.\ t \uparrow 0)\ x)\ xb\ xc \\
\qquad\qquad\qquad\quad xa\ 0\ (y\ 0)\ ya \\
\qquad\qquad\quad \mathit{in}\ (x,\ y) \\
\qquad\qquad \mid \mathit{Right} \Rightarrow \ldots\ )) \qquad \text{head variable not affected} \\
\qquad (\lambda t\ x\ r\ xa\ xb\ xc\ xd\ H.\ \ldots\ ) \qquad \text{abstraction case} \\
\quad x\ xa\ xd\ xe\ xb\ H\ Ha
\end{array}
$$

Proof of the main theorem

theorem *type-NF*: **assumes** T: $e \vdash_R t : T$
  shows $\exists\, t'.\ t \to_\beta^*\ t' \wedge t' \in NF$ **using** T
proof *induct*
  case $(AppRT\ T\ U\ e\ s\ t)$
  from $AppRT$ **obtain** $s'\ t'$ **where** *sred*: $s \to_\beta^*\ s'$ **and** *sNF*: $s' \in NF$
    and *tred*: $t \to_\beta^*\ t'$ **and** *tNF*: $t' \in NF$ **by** *rules*
  have $\exists\, u.\ (Var\ 0 \circ t' \uparrow 0)[s'/0] \to_\beta^*\ u \wedge u \in NF$
  proof $(rule\ subst\text{-}type\text{-}NF)$
    ⋮
  qed
  then obtain u **where** *ured*: $s' \circ t' \to_\beta^*\ u$ **and** *unf*: $u \in NF$ **by** *simp rules*
  from *sred tred* **have** $s \circ t \to_\beta^*\ s' \circ t'$ **by** $(rule\ rtrancl\text{-}beta\text{-}App)$
  hence $s \circ t \to_\beta^*\ u$ **using** *ured* **by** $(rule\ rtrancl\text{-}trans)$
  with *unf* **show** *?case* **by** *rules*
next
  ...
qed

Computational content of main theorem

$$
\begin{array}{ll}
\mathit{type\text{-}NF} \equiv & \\
\lambda H.\ \mathit{rtypingT\text{-}rec}\ (\lambda e\ x\ T.\ (\mathit{Var}\ x,\ \mathit{Var\text{-}NF}\ x)) & \text{variable case} \\
\quad (\lambda e\ T\ t\ U\ x\ r.\ \mathit{let}\ (x,\ y) = r\ \mathit{in}\ (\mathit{Abs}\ x,\ \mathit{AbsN}\ x\ y)) & \text{abstraction case} \\
\quad (\lambda e\ s\ T\ U\ t\ x\ xa\ r\ ra. & \text{application case} \\
\qquad \mathit{let}\ (x,\ y) = r;\ (xa,\ ya) = ra; & \\
\qquad \mathit{in}\ \mathit{subst\text{-}type\text{-}NF}\ (\mathit{Var}\ 0 \circ xa \uparrow 0)\ e\ 0\ (T \Rightarrow U)\ U\ x & \\
\qquad\quad (\mathit{AppN}\ [xa \uparrow 0]\ 0\ (\mathit{listall\text{-}cons\text{-}P}\ (\mathit{lift\text{-}NF}\ 0\ ya)\ \mathit{listall\text{-}nil\text{-}P}))\ y) & \\
\quad H &
\end{array}
$$

Benchmark: Multiplication of Church numerals

$2 =_{def} (\lambda f.\ \lambda x.\ f\ (f\ x))$
$\star =_{def} (\lambda m.\ \lambda n.\ \lambda f.\ m\ (n\ f))$
$x \star 2 \rightarrow_\beta{}^* x'$

Runtime of normalization function

x	extracted	hand-written
2	0.002	0.000
4	0.016	0.000
8	0.012	0.000
16	0.036	0.000
32	0.096	0.000
64	0.430	0.000

x	extracted	hand-written
128	2.615	0.000
256	27.786	0.001
512	364.193	0.002
1024	?	0.007
2048	?	0.010
4096	?	0.030

Done on a Pentium III with 1 GHz

Conclusion

- Realistic example for program extraction
- Relatively compact proof (800 lines of Isabelle code)
- Extracted program reasonably efficient for medium-size input

Future work

- Increase efficiency of normalization function
  - Eliminate redundant arguments (e.g. typing information)
    May require computationally irrelevant universal quantifiers
  - Avoid re-examining terms
  - Avoid generating garbage
- Extract other normalization algorithms, e.g. NBE
- Other encodings, e.g. nominal techniques