Towards formalizing ⊤⊤-lifting in HOL-Nominal

Christian Doczkal

Advisor: Dr. Jan Schwinghammer
Supervisor: Prof. Gert Smolka

February 6, 2009

What's the story?

How to prove strong normalization

and

how to implement the proof in Isabelle/HOL-Nominal

Strong Normalization

Usually: "A term t is strongly normalizing if there is no infinite sequence $t \mapsto t_1 \mapsto t_2 \mapsto \ldots$ of reduction steps beginning at t."

Implicit: reduction is *finitely branching* so there exists an upper bound on the length of all possible reduction sequences.

For my formalization I use an inductive variant

Definition (strong normalization)

$$SN\ t \equiv \forall t'.t \mapsto t' \Rightarrow SN\ t'$$

Moggi's computational metalanguage

- terms and types:

$$\tau ::= b \mid \tau \to \tau \mid T\ \ \tau$$
$$t ::= x \mid \lambda x.t \mid t\ \ t \mid [t] \mid t\ \texttt{to}\ x\ \texttt{in}\ t$$

- typing rules:

$$\frac{\Gamma \vdash t : \tau}{\Gamma \vdash [t] : T\,\tau} \qquad \frac{\Gamma \vdash s : T\,\sigma \qquad \Gamma, x : \sigma \vdash t : T\,\tau}{\Gamma \vdash s\ \texttt{to}\ x\ \texttt{in}\ t : T\,\tau}$$

- reductions:

$$\begin{array}{lr} T.\beta & [s]\ \texttt{to}\ x\ \texttt{in}\ t \mapsto t[x ::= s] \\ T.\eta & s\ \texttt{to}\ x\ \texttt{in}\ [x] \mapsto s \\ T.assoc & (s\ \texttt{to}\ x\ \texttt{in}\ t)\ \texttt{to}\ y\ \texttt{in}\ u \mapsto s\ \texttt{to}\ x\ \texttt{in}\ (t\ \texttt{to}\ y\ \texttt{in}\ u) \end{array}$$

What's difficult about strong normalization

- β-reduction may increase the size (and depth) of a term
  - $\Rightarrow$ no naive inductive proof
- untyped λ-calculus is *not* strongly normalizing
  - $\Rightarrow$ need to exploit type structure

$\Rightarrow$ Use logical relations proof technique

What's difficult about strong normalization

- β-reduction may increase the size (and depth) of a term
  - $\Rightarrow$ no naive inductive proof
- untyped λ-calculus is *not* strongly normalizing
  - $\Rightarrow$ need to exploit type structure

$\Rightarrow$ Use logical relations proof technique

The logical relations approach

1. Define a type indexed family of relations red_τ
2. Show by induction on the type structure

$$t \in red_\tau \Rightarrow SN(t)$$
$$t \in red_\tau \wedge t \mapsto t' \Rightarrow t' \in red_\tau$$
$$neutral(t) \wedge (\forall t'.t \mapsto t' \Rightarrow t' \in red_\tau) \Rightarrow t \in red_\tau$$

3. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on the typing derivation

First attempt at defining reducibility:

$$t \in red_b \equiv SN\ t$$
$$t \in red_{\sigma \to \tau} \equiv \forall u \in red_\sigma . t\ u \in red_\tau$$
$$t \in red_{T\sigma} \equiv \forall u \in X . t \texttt{ to } x \texttt{ in } u \in X$$

The logical relations approach

1. Define a type indexed family of relations red_τ
2. Show by induction on the type structure

$$t \in red_\tau \Rightarrow SN(t)$$
$$t \in red_\tau \wedge t \mapsto t' \Rightarrow t' \in red_\tau$$
$$neutral(t) \wedge (\forall t'.t \mapsto t' \Rightarrow t' \in red_\tau) \Rightarrow t \in red_\tau$$

3. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on the typing derivation

First attempt at defining reducibility:

$$t \in red_b \equiv SN\ t$$
$$t \in red_{\sigma \to \tau} \equiv \forall u \in red_\sigma . t\ u \in red_\tau$$
$$t \in red_{T\,\sigma} \equiv \forall u \in X . t\ \texttt{to}\ x\ \texttt{in}\ u \in X$$

The logical relations approach

1. Define a type indexed family of relations red_τ
2. Show by induction on the type structure

$$t \in red_\tau \Rightarrow SN(t)$$

$$t \in red_\tau \wedge t \mapsto t' \Rightarrow t' \in red_\tau$$

$$neutral(t) \wedge (\forall t'.\, t \mapsto t' \Rightarrow t' \in red_\tau) \Rightarrow t \in red_\tau$$

$$\frac{\Gamma \vdash s : T\,\sigma \qquad \Gamma, x : \sigma \vdash t : T\,\tau}{\Gamma \vdash s\ \texttt{to}\ x\ \texttt{in}\ t : T\,\tau}$$

$$t \in red_b \equiv SN\ t$$

$$t \in red_{\sigma \to \tau} \equiv \forall u \in red_\sigma . t\ u \in red_\tau$$

$$t \in red_{T\,\sigma} \equiv \forall u \in X . t\ \texttt{to}\ x\ \texttt{in}\ u \in X$$

The logical relations approach

Definition (stack)

$$K ::= Id \mid (y)n :: L$$

$$t \star Id = t$$
$$t \star ((y)n :: L) = (t \; \texttt{to} \; y \; \texttt{in} \; n) \star L$$

Definition (reducibility)

$$t \in red_b \equiv SN \; t$$
$$t \in red_{\sigma \to \tau} \equiv \forall u \in red_\sigma . t \; u \in red_\tau$$
$$t \in red_{T\,\sigma} \equiv \forall K \in red_\sigma^\top . SN(t \star K)$$

$$K \in red_\sigma^\top \equiv \forall s \in red_\sigma . SN([s] \star K)$$

The logical relations approach

Definition (stack)

$$K ::= Id \mid (y)n :: L$$

$$t \star Id = t$$
$$t \star ((y)n :: L) = (t \;\texttt{to}\; y \;\texttt{in}\; n) \star L$$

Definition (reducibility)

$$t \in red_b \equiv SN\ t$$
$$t \in red_{\sigma \to \tau} \equiv \forall u \in red_\sigma . t\ u \in red_\tau$$
$$t \in red_{T\,\sigma} \equiv \forall K \in red_\sigma^\top . SN(t \star K)$$

$$K \in red_\sigma^\top \equiv \forall s \in red_\sigma . SN([s] \star K)$$

Stacks

```
nominal_datatype stack = Id | St "«name»lam" "stack"

consts
  length :: "stack ⇒ nat"
nominal_primrec
  "length Id = 0"
  "x ♯ K ⟹ length (St x t K) = 1 + length K"
  by (finite_guess+,auto simp add: fresh_nat )
     (fresh_guess)
```

Stacks and Dismantling

HOL-Nominal only provides infrastructure for defining *primitive recursive* functions

Stack dismantling ($\star$) is not primitive recursive

$$\begin{aligned} t \star Id &= t \\ t \star ((y)n :: L) &= (t\ \texttt{to}\ y\ \texttt{in}\ n) \star L \end{aligned}$$

```
function
  dismantle :: "lam ⇒ stack ⇒ lam" ("_ ⋆ _" [80,80] 80)
where
  "t ⋆ Id = t" |
  "x ♯ (K,t) ⟹ t ⋆ (St x s K) = (t to x in s) ⋆ K"
proof -          — pattern completeness
  { fix P :: bool and arg::"lam × stack"
    assume id: "⋀t. arg = (t, Id) ⟹ P"
      and st: "⋀x K t s. ⟦x ♯ (K, t); arg = (t, St x s K)⟧ ⟹ P"
    { assume "snd arg = Id"
      hence P by (metis id[of "fst arg"] surjective_pairing) }
    moreover
    { fix y n L assume "snd arg = St y n L" "y ♯ (L, fst arg)"
      hence P by (metis st[where t="fst arg"] surjective_pairing) }
    ultimately show P using stack_exhaust'[of "snd arg" "fst arg"]
      by(auto)
  }
```

```
function
  dismantle :: "lam ⇒ stack ⇒ lam" ("_ ⋆ _" [80,80] 80)
where
  "t ⋆ Id = t" |
  "x ♯ (K,t) ⟹ t ⋆ (St x s K) = (t to x in s) ⋆ K"
proof -        — pattern completeness
  { fix P :: bool and arg::"lam × stack"
    assume id: "⋀t. arg = (t, Id) ⟹ P"
      and st: "⋀x K t s. ⟦x ♯ (K, t); arg = (t, St x s K)⟧ ⟹ P"
    { assume "snd arg = Id"
      hence P by (metis id[of "fst arg"] surjective_pairing) }
    moreover
    { fix y n L assume "snd arg = St y n L" "y ♯ (L, fst arg)"
      hence P by (metis st[where t="fst arg"] surjective_pairing) }
    ultimately show P using stack_exhaust'[of "snd arg" "fst arg"]
      by(auto)
  }
```

function

lemma stack_exhaust' :
fixes c :: "'a::fs_name"
shows "b = Id ∨ (∃ x t K . x ♯ K ∧ x ♯ c ∧ b = St x t K)"
by(nominal_induct b avoiding: c rule: stack.strong_induct)
(auto)

hence P **by** (metis id[of "fst arg"] surjective_pairing) }
moreover
{ **fix** y n L **assume** "snd arg = St y n L" "y ♯ (L, fst arg)"
hence P **by** (metis st[**where** t="fst arg"] surjective_pairing) }
ultimately show P **using** stack_exhaust'[of "snd arg" "fst arg"]
by(auto)
}

```
— right uniqueness
 {
  fix t t' :: lam and x x' :: name and s s' :: lam and K K' :: stack
  assume "x ♯ (K, t)" "x' ♯ (K', t')"
    and "(t, St x s K) = (t', St x' s' K')"
  hence eq: "(t to x in s,K) = (t' to x' in s',K')"
    by (auto simp add: lam.inject stack.inject)
  let ?g = dismantle_sumC — graph of dismantle
  from eq show "?g (t to x in s, K) = ?g (t' to x' in s', K')"
    by (rule arg_cong)
 }
qed (simp_all add: stack.inject)


termination dismantle
by(relation "measure (λ(t,K). length K)")(auto)
```

Dismantling and Induction

Induction on a stack K has the cases $K = Id$ and $K = (y)n :: L$

Fact$_0$: $t \star ((y)n :: L)$ is of the form s `to` x `in` u

What is the connection between t, y, n, L and s, x, u? - None
Impossible to do case analysis like $t \star ((y)n :: L) \mapsto ?$

Fact$_1$: $t \star (L +\!\!+ (y)n :: Id) = (t \star L)$ `to` y `in` n

Want a reverse induction principle for stacks.

Dismantling and Induction

Induction on a stack K has the cases $K = Id$ and $K = (y)n :: L$

Fact$_0$: $t \star ((y)n :: L)$ is of the form s `to` x `in` u

What is the connection between t, y, n, L and s, x, u? - None
Impossible to do case analysis like $t \star ((y)n :: L) \mapsto ?$

Fact$_1$: $t \star (L +\!\!+ (y)n :: Id) = (t \star L)$ `to` y `in` n

Want a reverse induction principle for stacks.

Dismantling and Induction

Induction on a stack K has the cases $K = Id$ and $K = (y)n :: L$

Fact$_0$: $t \star ((y)n :: L)$ is of the form s `to` x `in` u

What is the connection between t, y, n, L and s, x, u? - None
Impossible to do case analysis like $t \star ((y)n :: L) \mapsto ?$

Fact$_1$: $t \star (L +\!\!+ (y)n :: Id) = (t \star L)$ `to` y `in` n

Want a reverse induction principle for stacks.

Rule

The standard rule:

$$\frac{\bigwedge z.\ P\ z\ Id \qquad \bigwedge y\ n\ L\ z.\ [\![y\ \sharp\ z;\ y\ \sharp\ L;\ \bigwedge z.\ P\ z\ L]\!] \Longrightarrow P\ z\ (St\ y\ n\ L)}{P\ z\ K}$$

The reverse rule:

$$\frac{\bigwedge z.\ P\ z\ Id \qquad \bigwedge y\ n\ L\ z.\ [\![y\ \sharp\ z;\ y\ \sharp\ L;\ \bigwedge z.\ P\ z\ L]\!] \Longrightarrow P\ z\ (L\ {+}{+}\ St\ y\ n\ Id)}{P\ z\ K}$$

lemma stack_reverse_strong_induct[case_names Id St]:
  fixes z :: "'a::fs_name"
  assumes id: "$\bigwedge$ z . P z Id"
  and   st: "$\bigwedge$ y n L z . $[\![$ y $\sharp$ z ; y $\sharp$ L ; $\bigwedge$ z . P z L $]\!]$
          $\Longrightarrow$ P z (L ++ St y n Id)"
  shows "P z K"
proof (subst srev_srev[THEN sym],
      rule stack.strong_induct[**where** P="λ z k . P z (srev k)"])
  { **fix** z **show** "P z (srev Id)" **using** id **by** simp }
  { **fix** y::name **and** n::lam **and** z::"('a::fs_name)" **and** L
    assume f: "y $\sharp$ z" "y $\sharp$ L"
      and ih: "$\bigwedge$ (z::'a::fs_name) . P z (srev L)"
    show "P z (srev (St y n L))"
      using f ih st[of y z "srev L" n]
      by (auto simp add: fresh_srev) }
qed

```
lemma stack_reverse_strong_induct[case_names Id St]:
  fixes z :: "'a::fs_name"
  assumes id: "⋀ z . P z Id"
  and     st: "⋀ y n L z . ⟦ y ♯ z ; y ♯ L ; ⋀ z . P z L ⟧
                 ⟹ P z (L ++ St y n Id)"
  shows "P z K"
proof ( subst srev_srev[THEN sym],
      rule stack.strong_induct[where P="λ z k . P z (srev k)"])
  { fix z show "P z (srev Id)" using id by simp }
  { fix y::name and n::lam and z::"('a::fs_name)" and L
    assume f: "y ♯ z" "y ♯ L"
      and ih: "⋀ (z::'a::fs_name) . P z (srev L)"
    show "P z (srev (St y n L))"
      using f ih st[of y z "srev L" n]
      by (auto simp add: fresh_srev) }
qed
```

```
lemma stack_reverse_strong_induct[case_names Id St]:
  fixes z :: "'a::fs_name"
  assumes id: "⋀ z . P z Id"
  and     st: "⋀ y n L z . ⟦ y ♯ z ; y ♯ L ; ⋀ z . P z L ⟧
                 ⟹ P z (L ++ St y n Id)"
  shows "P z K"
proof ( subst srev_srev[THEN sym],
        rule stack.strong_induct[where P="λ z k . P z (srev k)"])
  { fix z show "P z (srev Id)" using id by simp }
  { fix y::name and n::lam and z::"('a::fs_name)" and L
    assume f: "y ♯ z" "y ♯ L"
      and ih: "⋀ (z::'a::fs_name) . P z (srev L)"
    show "P z (srev (St y n L))"
      using f ih st[of y z "srev L" n]
      by (auto simp add: fresh_srev) }
qed
```

Application

Lemma

$$K \mapsto_k K' \Rightarrow \textit{length } K \geq \textit{length } K'$$

where $K \mapsto_k K' \equiv \forall t.\ t \star K \mapsto t \star K'$

Proof on Paper: "Suppose $x \star K \mapsto x \star K'$
and $K = (y_1)n_1 :: (y_2)n_2 :: \ldots :: Id$"
"There are only two reductions that might change the length of K"

Isabelle/HOL-Nominal: Roughly 90 lines inductive proof

Application

$$
\begin{array}{ll}
T.\beta & [s]\ \texttt{to}\ x\ \texttt{in}\ t \mapsto t[x ::= s] \\
T.\eta & s\ \texttt{to}\ x\ \texttt{in}\ [x] \mapsto s \\
T.assoc & (s\ \texttt{to}\ x\ \texttt{in}\ t)\ \texttt{to}\ y\ \texttt{in}\ u \mapsto s\ \texttt{to}\ x\ \texttt{in}\ (t\ \texttt{to}\ y\ \texttt{in}\ u)
\end{array}
$$

Proof on Paper: "Suppose $x \star K \mapsto x \star K'$
and $K = (y_1)n_1 :: (y_2)n_2 :: \ldots :: Id$"
"There are only two reductions that might change the length of K"

Isabelle/HOL-Nominal: Roughly 90 lines inductive proof

Application

Lemma

$$K \mapsto_k K' \Rightarrow \mathit{length}\ K \geq \mathit{length}\ K'$$

where $K \mapsto_k K' \equiv \forall t.\ t \star K \mapsto t \star K'$

Proof on Paper: "Suppose $x \star K \mapsto x \star K'$
and $K = (y_1)n_1 :: (y_2)n_2 :: \ldots :: Id$"
"There are only two reductions that might change the length of K"

Isabelle/HOL-Nominal: Roughly 90 lines inductive proof

Current State

1. Define a type indexed family of relations red_τ
2. Show by induction on the type structure

$$
\begin{aligned}
t \in red_\tau &\Rightarrow SN(t) \\
t \in red_\tau \wedge t \mapsto t' &\Rightarrow t' \in red_\tau \\
neutral(t) \wedge (\forall t'. t \mapsto t' \Rightarrow t' \in red_\tau) &\Rightarrow t \in red_\tau
\end{aligned}
$$

3. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on typing derivations

Current State

1. Formalize λ_{ml} incl. substitution, reduction, and types
2. Define a type indexed family of relations red_τ
3. Show by induction on the type structure

$$
\begin{aligned}
t \in red_\tau &\Rightarrow SN(t) \\
t \in red_\tau \wedge t \mapsto t' &\Rightarrow t' \in red_\tau \\
neutral(t) \wedge (\forall t'. t \mapsto t' \Rightarrow t' \in red_\tau) &\Rightarrow t \in red_\tau
\end{aligned}
$$

4. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on typing derivations

Current State

1. Formalize λ_{ml} incl. substitution, reduction, and types
2. Formalize stacks and their properties
3. Define a type indexed family of relations red_τ
4. Show by induction on the type structure

$$t \in red_\tau \Rightarrow SN(t)$$
$$t \in red_\tau \wedge t \mapsto t' \Rightarrow t' \in red_\tau$$
$$neutral(t) \wedge (\forall t'. t \mapsto t' \Rightarrow t' \in red_\tau) \Rightarrow t \in red_\tau$$

5. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on typing derivations

Current State

1. Formalize λ_{ml} incl. substitution, reduction, and types (280)
2. Formalize stacks and their properties (700)
3. Define a type indexed family of relations red_τ (30)
4. Show by induction on the type structure (400 incl. λ-cases))

$$t \in red_\tau \Rightarrow SN(t)$$

$$t \in red_\tau \wedge t \mapsto t' \Rightarrow t' \in red_\tau$$

$$neutral(t) \wedge (\forall t'.t \mapsto t' \Rightarrow t' \in red_\tau) \Rightarrow t \in red_\tau$$

5. Prove $\Gamma \vdash t : \tau \Rightarrow t \in red_\tau$ by induction on typing derivations (central part of the paper)

References

- T.Nipkow, L.C. Paulson, M. Wenzel A Proof Assistant for Higher-Order Logic, http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf
- T.Nipkow, A Tutorial Introduction to Structured Isar Proofs http://isabelle.in.tum.de/dist/Isabelle/doc/isar-overview.pdf
- C. Urban, Nominal Techniques in Isabelle/HOL, Journal of Automatic Reasoning, Vol. 40(4), pp: 327-356, 2008
- S. Lindley and I. Stark, Reducibility and TT-lifting for Computation Types, Typed Lambda Calculi and Applications, LNCS Vol. 3461, pp: 262-277, Springer-Verlag, 2005.

Thank You!