Strong Normalization for Guarded Types

Andreas Abel    Andrea Vezzosi

Department of Computer Science and Engineering
Chalmers and Gothenburg University, Sweden

PLS Seminar
ITU, Copenhagen, Denmark
20 August 2014
Introduction

- Guarded recursive types (Nakano, LICS 2000)
- (Negative) recursive types in type theory
- Applications in semantics (abstracting step-indexing)
- Applications in FRP (causality)
- This talk: strong normalization
Guarded types

- Types and terms.

\[ A, B ::= A \to B \mid \triangleright A \mid X \mid \mu X A \]
\[ t, u ::= x \mid \lambda x t \mid t u \mid \text{next } t \mid t * u \]

- Type equality: congruence closure of \( \vdash \mu X A = A[\mu X A/X] \).

- Typing \( \Gamma \vdash t : A \).

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash \text{next } t : \triangleright A} \quad \frac{\Gamma \vdash t : \triangleright (A \to B) \quad \Gamma \vdash u : \triangleright A}{\Gamma \vdash t * u : \triangleright B} \quad \frac{\Gamma \vdash t : A \quad \vdash A = B}{\Gamma \vdash t : B}
\]
Reduction

- Redex contraction $t \mapsto t'$.

  $$(\lambda x t) u \mapsto t[u/x]$$

  $\text{next } t \ast \text{next } u \mapsto \text{next } (t u)$$

- Full one-step reduction $t \longrightarrow t'$: Compatible closure of $\mapsto$.
Recursion from recursive types

Guarded recursion combinator can be encoded.

\[ f : \text{\textmu}X. \upnu X \rightarrow A = \upnu (\upnu B \rightarrow A) \]
\[ B : \text{\textmu}X. \upnu X \rightarrow A = \upnu (\upnu B \rightarrow A) \]
\[ x : \upnu B = \upnu (\upnu B \rightarrow A) \]
\[ x \ast \text{next} \, x : \upnu A \]
\[ f (x \ast \text{next} \, x) : A \]
\[ \omega := \lambda x. f (x \ast \text{next} \, x) : \upnu B \rightarrow A = \upnu B \rightarrow A \]
\[ Y := \omega (\text{next} \, \omega) : A \]

\[ Y \rightarrow f (\text{next} \, \omega \ast \text{next} \, (\text{next} \, \omega)) \rightarrow f (\text{next} \, (\omega (\text{next} \, \omega))) = f (\text{next} \, Y) \]

Full reduction \(\longrightarrow\) diverges.
Restricted reduction

- Restore normalization: do not reduce under next.
- Relaxed: reduce only under next up to a certain depth.
- Family $\rightarrow_n$ of reduction relations.

\[
\frac{t \mapsto t'}{t \rightarrow_n t'} \quad \quad \frac{t \rightarrow_n t'}{\text{next } t \rightarrow_{n+1} \text{next } t'}
\]

- Plus compatibility rules for all other term constructors.
- $\rightarrow_n$ is monotone in $n$ (more fuel gets you further).
- Goal: each $\rightarrow_n$ is strongly normalizing.
Strong normalization as well-foundedness

- $t \in \text{sn}_n$ if $\rightarrow_n$ reduction starting with $t$ terminates.

\[
\forall t'. t \rightarrow_n t' \implies t' \in \text{sn}_n \\
\implies t \in \text{sn}_n
\]

- $\text{sn}_n$ is antitone in $n$, since $\rightarrow_n$ occurs negatively.

- More reductions $\implies$ less termination.
Inductive SN

- Lambda-calculus:
  - $\vec{u} \in \text{SN}$
  - $t \in \text{SN}$
  - $\lambda x t \in \text{SN}$
  - $t[u/x] \in \text{SN}$
  - $(\lambda x t) u \vec{u} \in \text{SN}$

- With evaluation contexts $E ::= _- | E u$:
  - $E \in \text{SN}$
  - $u \in \text{SN}$
  - $E[u] \in \text{SN}$
  - $\lambda x t \in \text{SN}$
  - $E[\lambda x t] \in \text{SN}$
  - $E[(\lambda x t) u] \in \text{SN}$
Inductive SN (ctd.)

- Strong contraction \( t \mapsto^{SN} t' \).

\[
\begin{align*}
  & u \in SN \\
  \therefore & (\lambda x t) u \mapsto^{SN} t[u/x]
\end{align*}
\]

- "Strong head reduction" \( t \longrightarrow^{SN} t' \).

\[
\begin{align*}
  & t \mapsto^{SN} t' \\
  \therefore & E[t] \longrightarrow^{SN} E[t']
\end{align*}
\]

- SN with strong head reduction.

\[
\begin{align*}
  & t \longrightarrow^{SN} t' \\
  & t' \in SN \\
  \therefore & t \in SN
\end{align*}
\]
SN with guarded types

- Extending evaluation contexts: \( E ::= \cdots | E * u | \text{next } t * E \)

- Extending strong contraction:

\[
\begin{align*}
  u & \in \text{SN}_n \\
  (\lambda x t) u & \mapsto_{\text{SN}_n} t[u/x] \\
  \text{next } t * \text{next } u & \mapsto_{\text{SN}_n} \text{next } (t u)
\end{align*}
\]

- Adding index to strong head reduction:

\[
\begin{align*}
  t & \mapsto_{\text{SN}_n} t' \\
  E[t] & \mapsto_{\text{SN}_n} E[t'] \\
  t & \mapsto_{\text{SN}_n} t' \quad t' \in \text{SN}_n \\
  t & \in \text{SN}_n
\end{align*}
\]

- Adding rule for introduction:

\[
\begin{align*}
  \text{next } t & \in \text{SN}_0 \\
  \text{next } t & \in \text{SN}_{n+1}
\end{align*}
\]

- \( \text{SN}_n \) is antitone in \( n \).
Notions of s.n. coincide?

- Rules for $\text{SN}_n$ are closure properties of $\text{sn}_n$.
- $\text{SN}_n \subseteq \text{sn}_n$ follows by induction on $\text{SN}_n$.
- Converse $\text{sn}_n \subseteq \text{SN}_n$ does not hold!
- Counterexamples are ill-typed s.n. terms, e.g.,

$$ (\lambda x. x) \ast y \quad \text{or} \quad (\text{next } x) \ast y. $$

- Solution: consider only well-typed terms.
- Proof of $t \in \text{sn}_n \implies t \in \text{SN}_n$ by case distinction on $t$: neutral $(E[x])$, introduction $(\lambda x t, \text{next } t)$, or weak head redex.
Saturated sets (semantic types)

- Types are modeled by sets $\mathcal{A}$ of s.n. terms.
- Semantic function space should contain $\lambda$s and terms that weak head reduce to $\lambda$s.
- $n$-closure $\overline{\mathcal{A}}_n$ of $\mathcal{A}$ inductively:

$$
\frac{t \in \mathcal{A}}{t \in \overline{\mathcal{A}}_n} \quad \frac{E \in \text{SN}_n}{E[x] \in \overline{\mathcal{A}}_n} \quad \frac{t \stackrel{\text{SN}}{\rightarrow}_n t'}{t' \in \overline{\mathcal{A}}_n}
$$

- $\mathcal{A}$ is $n$-saturated ($\mathcal{A} \in \text{SAT}_n$) if $\overline{\mathcal{A}}_n \subseteq \mathcal{A}$.
- Saturated sets are non-empty (contain e.g. the variables).
Constructions on semantic types

- Function space and “later”:

\[ A \rightarrow B = \{ t \mid t u \in B \text{ for all } u \in A \} \]
\[ \triangleright_n A = \{ \text{next } t \mid t \in A \text{ if } n > 0 \} \]

- If \( A, B \in SAT_n \) then \( A \rightarrow B \in SAT_n \).
- \( \triangleright_0 A \in SAT_0 \).
- If \( A \in SAT_n \) then \( \triangleright_{n+1} A \in SAT_{n+1} \).
Type interpretation

- Type interpretation $\llbracket A \rrbracket_n \in \text{SAT}_n$

$$\llbracket A \rightarrow B \rrbracket_n = \bigcap_{n' \leq n} (\llbracket A \rrbracket_{n'} \rightarrow \llbracket B \rrbracket_{n'})$$

$$\llbracket ▶ A \rrbracket_0 = ▶_0 \text{SN}_0 = \{\text{next } t\}_0$$

$$\llbracket ▶ A \rrbracket_{n+1} = ▶_{n+1} \llbracket A \rrbracket_n$$

$$\llbracket \mu X A \rrbracket_n = \llbracket A[\mu X A/X] \rrbracket_n$$

By lex. induction on $(n, \text{size}(A))$ where $\text{size}(▶ A) = 0$.

Requires recursive occurrences of $X$ to be guarded by a $▶$. 

Abel Vezzosi (Chalmers/GU)
Type soundness

- Context interpretation:

\[ \rho \in [\Gamma]_n \iff \rho(x) \in [A]_n \text{ for all } (x:A) \in \Gamma \]

- Identity substitution \(\text{id} \in [\Gamma]_n\) since \(x \in [A]_n\).

- Type soundness: if \(\Gamma \vdash t : A\) then \(t\rho \in [A]_n\) for all \(n\) and \(\rho \in [\Gamma]_n\).

- Corollary: \(t \in \text{SN}_n\) for all \(n\).
Formalization in Agda

- Intensional type theory does not support quotients well: in our case, types modulo type equality.
- Use infinite type expressions instead (coinduction).
- Only guarded types admit an interpretation.
- Typing judgement needs to be restricted to guarded types.
- Use mixed inductive-coinductive representation of types to express guard condition.

\[
A , B \ ::= \ A \to B \ | \ \triangleright A' \\
A' , B' \ ::=^{\text{co}} A
\]

- Intensional (propositional) equality too weak for coinductive types.
- Add an extensionality axiom for our coinductive type.
Introduction

Well-typed terms

- We used intrinsically well-typed terms (data structure indexed by typing context and type expression).
- Second Kripke dimension (context) required “everywhere”, e.g., in SN and [A].
Conclusions

- **Strong** normalization is a new result, albeit expected.
- Main focus: Agda formalization.
- Needed dedication (mostly Andrea’s).
- Forthcoming APLAS 2014 paper (literate Agda, fully machine-checked).
- Fuzzy hope that HoTT will improve equality situation for coinductive types.
Acknowledgments

- Rasmus Møgelberg, for discussions and the present invitation.
- Lars Birkedal, for a previous invitation that initiated this research.
- Neel Krishnaswami, for email input.
- The (other) Agda developers, especially Stevan Andjelkovic for the \LaTeX backend.