Pure Type Systems revisited

Herman Geuvers

Radboud University Nijmegen
and
Eindhoven University of Technology
The Netherlands

LIX Colloquium
Theory and Application of Formal Proofs
Paris, 5-7 November 2013

It was 20 years ago ...

that I defended my PhD. thesis ("Logics and Type Systems") on Pure Type Systems

It was 20 years ago ...

that I defended my PhD. thesis ("Logics and Type Systems") on Pure Type Systems

It was 45 years ago today ...

that this much more interesting album appeared

Pure Type Systems

- Unified presentation of systems of dependently typed λ calculus
- Barendregt: Fine structure of the Calculus of Constructions: λ-cube
- Berardi: Study the interpretation of logics in systems of the λ-cube $\mapsto$ the logic cube
- Terlouw, Geuvers & Nederhof: study the normalization of Calculus of Constructions and type systems in general.
- First definition of PTSs: Generalized Type Systems 1991

Pure Type Systems revisited

Content

- Rules of Pure Type Systems and examples
- Meta-theory: Subject Reduction, Church-Rosser, Normalization
- Combining with η
- Looping and fixed-point combinators and an open problem
- A conjecture about WN and SN
- Revisiting Contexts (PTSs without contexts)
- Revisiting Conversion (Making conversion explicit)

Pure Type Systems revisited

Content

- Rules of Pure Type Systems and examples
- Meta-theory: Subject Reduction, Church-Rosser, Normalization
- Combining with η
- Looping and fixed-point combinators and an open problem
- A conjecture about WN and SN
- Revisiting Contexts (PTSs without contexts)
- Revisiting Conversion (Making conversion explicit)

Not treated:

- Classical PTSs and Domain-free PTSs
- PTS with explicit substitution
- Syntax-directed PTS & Type checking
- Sequent calculus PTS
- ...

Pure Type Systems revisited

Content

- Rules of Pure Type Systems and examples
- Meta-theory: Subject Reduction, Church-Rosser, Normalization
- Combining with η
- Looping and fixed-point combinators
- A conjecture about WN and SN
- PTSs without contexts
- Making conversion explicit

Based on work of and joint work with (non-exhaustive):
H. Barendregt, B. van Benthem Jutting, S. Berardi, G. Barthe, Th. Coquand, F. van Doorn, G. Gonthier, H. Herbelin, D. Howe, T. Hurkens, R. Krebbers, J. McKinna, M.-J. Nederhof, R. Nederpelt, R. Pollack, V. Siles, M.H. Sørensen, J. Terlouw, J. Verkoelen, B. Werner, F. Wiedijk

Rules of Pure Type Systems

- Application and λ-abstraction.
- Π-types: $\Pi x{:}A.B$ think of $\{f | \forall a : A(f\,a : B[a/x]\}$.
- Rules:

$$(\lambda) \quad \frac{\Gamma, x{:}A \vdash M : B \qquad \Gamma \vdash \Pi x{:}A.B : \mathsf{s}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$$

$$(\mathsf{app}) \quad \frac{\Gamma \vdash M : \Pi x{:}A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

Notation: $A \to B$ for $\Pi x{:}A.B$ when $x \notin \mathsf{FV}(()B)$.
Examples: Given $A : \mathsf{Type}, P : A {\to} \mathtt{Prop}$,

- $\lambda x : A.\lambda h : P\,x.x : \Pi x : A.P\,x \to P\,x$
- $(\lambda x : A.\lambda h : P\,x.x)\,a : P\,a \to P\,a$

Rules of Pure Type Systems

- Structural (context) rules.
- Parameter: $\mathcal{S}$ is the set of "sorts" of the PTS (or the "universes")
- Rules:

$$(\mathsf{var})\quad \frac{\Gamma \vdash A : \mathsf{s}}{\Gamma, x{:}A \vdash x : A} \qquad (\mathsf{weak})\quad \frac{\Gamma \vdash A : \mathsf{s} \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$$

Rules of Pure Type Systems

- Structural (context) rules.
- Parameter: $\mathcal{S}$ is the set of "sorts" of the PTS (or the "universes")
- Rules:

$$\text{(var)}\ \frac{\Gamma \vdash A : \mathsf{s}}{\Gamma, x{:}A \vdash x : A} \quad \text{(weak)}\ \frac{\Gamma \vdash A : \mathsf{s} \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$$

- Relations between sorts and Π-type formation.
- Parameters: $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$.

$$\text{(axiom)}\ \vdash \mathsf{s}_1 : \mathsf{s}_2 \qquad \text{if } (\mathsf{s}_1, \mathsf{s}_2) \in \mathcal{A}$$

$$(\Pi)\ \frac{\Gamma \vdash A : \mathsf{s}_1 \quad \Gamma, x{:}A \vdash B : \mathsf{s}_2}{\Gamma \vdash \Pi x{:}A.B : \mathsf{s}_3} \quad \text{if } (\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3) \in \mathcal{R}$$

- The triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ determines the PTS.

Rules of Pure Type Systems

- Special rule: β-conversion

$$(\textcolor{red}{\text{conv}}) \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \mathsf{s}}{\Gamma \vdash M : B} \text{ if } A =_\beta B$$

- β-equal types have the same inhabitants
- $x : \mathrm{Vec}(3+2) \vdash x : \mathrm{Vec}(5)$.

Pure Type Systems: all rules

Parameters: $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ with $\mathcal{A} \subset \mathcal{S} \times \mathcal{S}$, $\mathcal{R} \subset \mathcal{S} \times \mathcal{S} \times \mathcal{S}$.

(axiom) $\vdash \mathsf{s}_1 : \mathsf{s}_2$ if $(\mathsf{s}_1, \mathsf{s}_2) \in \mathcal{A}$

(weak) $$\frac{\Gamma \vdash A : \mathsf{s} \quad \Gamma \vdash M : C}{\Gamma, x{:}A \vdash M : C}$$

(var) $$\frac{\Gamma \vdash A : \mathsf{s}}{\Gamma, x{:}A \vdash x : A}$$

(app) $$\frac{\Gamma \vdash M : \Pi x{:}A.B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}$$

(λ) $$\frac{\Gamma, x{:}A \vdash M : B \quad \Gamma \vdash \Pi x{:}A.B : \mathsf{s}}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}$$

(Π) $$\frac{\Gamma \vdash A : \mathsf{s}_1 \quad \Gamma, x{:}A \vdash B : \mathsf{s}_2}{\Gamma \vdash \Pi x{:}A.B : \mathsf{s}_3}$$ if $(\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3) \in \mathcal{R}$

(conv) $$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \mathsf{s}}{\Gamma \vdash M : B}$$ if $A =_\beta B$

PTSs: the λ-cube

The cube of typed λ-calculi: $\mathcal{S} = \{\texttt{Prop}, \mathsf{Type}\}$

- In the Π-rule:

$$(\Pi) \quad \frac{\Gamma \vdash A : \mathsf{s}_1 \quad \Gamma, x{:}A \vdash B : \mathsf{s}_2}{\Gamma \vdash \Pi x{:}A.B : \mathsf{s}_2}$$

  we always take $\mathsf{s}_2 = \mathsf{s}_3$.
- We take $(\texttt{Prop}, \texttt{Prop})$ in every $\mathcal{R}$
- For the rest we vary on all possible combinations for

  $\mathcal{R} \subseteq \{ (\texttt{Prop}, \texttt{Prop}), (\mathsf{Type}, \texttt{Prop}), (\mathsf{Type}, \mathsf{Type}), (\texttt{Prop}, \mathsf{Type}) \}$

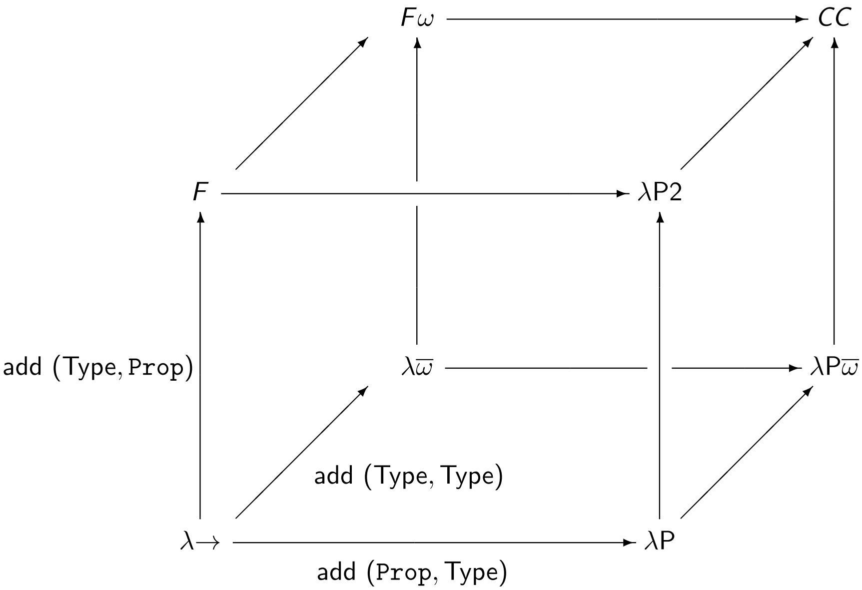
PTSs: The λ-cube

$$(\Pi) \quad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_2} \quad \text{if } (s_1, s_2) \in \mathcal{R}$$

System	$\mathcal{R}$			
$\lambda\!\to$	$(\mathtt{Prop}, \mathtt{Prop})$			
$\lambda 2$ (system F)	$(\mathtt{Prop}, \mathtt{Prop})$	$(\mathsf{Type}, \mathtt{Prop})$		
λP (LF)	$(\mathtt{Prop}, \mathtt{Prop})$		$(\mathtt{Prop}, \mathsf{Type})$	
$\lambda\overline{\omega}$	$(\mathtt{Prop}, \mathtt{Prop})$			$(\mathsf{Type}, \mathsf{Type})$
λP2	$(\mathtt{Prop}, \mathtt{Prop})$	$(\mathsf{Type}, \mathtt{Prop})$	$(\mathtt{Prop}, \mathsf{Type})$	
$\lambda\omega$ (system Fω)	$(\mathtt{Prop}, \mathtt{Prop})$	$(\mathsf{Type}, \mathtt{Prop})$		$(\mathsf{Type}, \mathsf{Type})$
$\lambda P\overline{\omega}$	$(\mathtt{Prop}, \mathtt{Prop})$		$(\mathtt{Prop}, \mathsf{Type})$	$(\mathsf{Type}, \mathsf{Type})$
λPω (CC)	$(\mathtt{Prop}, \mathtt{Prop})$	$(\mathsf{Type}, \mathtt{Prop})$	$(\mathtt{Prop}, \mathsf{Type})$	$(\mathsf{Type}, \mathsf{Type})$

N.B. $\lambda\!\to$, λP, $\lambda 2$ ... in this presentation are equivalent to the well-known ones.

PTSs: The λ-cube



Some other Pure Type Systems

λHOL	
$\mathcal{S}$	$\mathtt{Prop}$, Type, Type′
$\mathcal{A}$	$\mathtt{Prop}$: Type, Type : Type′
$\mathcal{R}$	($\mathtt{Prop}$, $\mathtt{Prop}$), (Type, Type), (Type, $\mathtt{Prop}$)

λU^-	
$\mathcal{S}$	$\mathtt{Prop}$, Type, Type′
$\mathcal{A}$	$\mathtt{Prop}$: Type, Type : Type′
$\mathcal{R}$	($\mathtt{Prop}$, $\mathtt{Prop}$), (Type, Type), (Type′, Type), (Type, $\mathtt{Prop}$)

$\lambda*$	
$\mathcal{S}$	$\star$
$\mathcal{A}$	$\star : \star$
$\mathcal{R}$	$(\star, \star)$

- λHOL corresponds to constructive Higher Order Logic under the Curry-Howard isomorphism
- λU^- is Higher Order Logic over impredicative domains and is inconsistent (Girard's paradox)
- $\lambda*$ is the system with 'Type : Type', which is also inconsistent.

Some meta-theory

- β-reduction is Church-Rosser on the pseudo-terms

$$\mathsf{T} ::= \mathcal{S} \mid \mathsf{Var} \mid (\Pi \mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid (\lambda \mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid \mathsf{T}\mathsf{T}.$$

  Therefore we have

$$\Pi x : A.B =_\beta \Pi x : C.D \Longrightarrow A =_\beta C \wedge B =_\beta D \qquad (\dagger)$$

- From that we conclude Subject Reduction:

$$\Gamma \vdash M : A \wedge M \twoheadrightarrow_\beta P \Longrightarrow \Gamma \vdash P : A$$

  Interesting case: M itself is a β-redex. It follows from $(\dagger)$

Some meta-theory

- β-reduction is Church-Rosser on the pseudo-terms

$$\mathsf{T} ::= \mathcal{S} \mid \mathsf{Var} \mid (\Pi\mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid (\lambda\mathsf{Var}{:}\mathsf{T}.\mathsf{T}) \mid \mathsf{T}\mathsf{T}.$$

  Therefore we have

$$\Pi x : A.B =_\beta \Pi x : C.D \Longrightarrow A =_\beta C \wedge B =_\beta D \qquad (\dagger)$$

- From that we conclude Subject Reduction:

$$\Gamma \vdash M : A \wedge M \twoheadrightarrow_\beta P \Longrightarrow \Gamma \vdash P : A$$

  Interesting case: M itself is a β-redex. It follows from ($\dagger$)
- Uniqueness of Types holds for functional PTSs:

$$\Gamma \vdash M : A \wedge \Gamma \vdash M : B \twoheadrightarrow_\beta P \Longrightarrow A =_\beta B$$

- Strong normalization holds for CC, but not for λU^- and $\lambda *$.

Adding η to the conversion rule

Problem: $\beta\eta$-reduction is not Church-Rosser on the pseudo-terms:

$$\begin{tikzcd}
 & \lambda y : B.(\lambda x : A.x)\, y \arrow[dl, "\eta"'] \arrow[dr, "\beta"] & \\
\lambda x : A.x & & \lambda y : B.y
\end{tikzcd}$$

If $A \neq_{\beta\eta} B$, these terms are not convertible.

Adding η to the conversion rule

Problem: $\beta\eta$-reduction is not Church-Rosser on the pseudo-terms:

$$\begin{tikzcd}
 & \lambda y : B.(\lambda x : A.x)\, y \arrow[dl, "\eta"'] \arrow[dr, "\beta"] & \\
\lambda x : A.x & & \lambda y : B.y
\end{tikzcd}$$

If $A \neq_{\beta\eta} B$, these terms are not convertible.

Question: How do we prove Subject Reduction? We need

$$\Pi x : A.B =_{\beta\eta} \Pi x : C.D \Longrightarrow A =_{\beta\eta} C \wedge B =_{\beta\eta} D$$

Adding η to the conversion rule

Problem: $\beta\eta$-reduction is not Church-Rosser on the pseudo-terms:

$$\begin{tikzcd} & \lambda y : B.(\lambda x : A.x)\, y \arrow[dl, "\eta"'] \arrow[dr, "\beta"] & \\ \lambda x : A.x & & \lambda y : B.y \end{tikzcd}$$

If $A \neq_{\beta\eta} B$, these terms are not convertible.
Question: How do we prove Subject Reduction? We need

$$\Pi x : A.B =_{\beta\eta} \Pi x : C.D \Longrightarrow A =_{\beta\eta} C \wedge B =_{\beta\eta} D$$

Domain Lemma!: For pseudo-terms we have $(\forall C[-], A, B, M)$

$$C[\lambda x : A.M] =_{\beta\eta} C[\lambda x : B.M]$$

Adding η to the conversion rule

Problem: $\beta\eta$-reduction is not Church-Rosser on the pseudo-terms:

$$\begin{tikzcd}
 & \lambda y : B.(\lambda x : A.x)\, y \arrow[dl, "\eta"'] \arrow[dr, "\beta"] & \\
\lambda x : A.x & & \lambda y : B.y
\end{tikzcd}$$

If $A \neq_{\beta\eta} B$, these terms are not convertible.

Question: How do we prove Subject Reduction? We need

$$\Pi x : A.B =_{\beta\eta} \Pi x : C.D \Longrightarrow A =_{\beta\eta} C \wedge B =_{\beta\eta} D$$

Domain Lemma!: For pseudo-terms we have $(\forall C[-], A, B, M)$

$$C[\lambda x : A.M] =_{\beta\eta} C[\lambda x : B.M]$$

From that we conclude

$$\Pi x : A.B =_{\beta\eta} \Pi x : C.D \Longrightarrow A =_{\beta\eta} C \wedge B =_{\beta\eta} D \qquad (\dagger)$$

and thereby Subject Reduction for β.

Adding η to the conversion rule

Summarizing we have:

- Subject Reduction for β
- If β-reduction is weakly normalizing, then
  - Subject Reduction for η holds.
  - Church-Rosser for $\beta\eta$ on well-typed terms holds:

$$\begin{array}{rl} \text{If} & \Gamma \vdash M : A \wedge \Gamma \vdash P : A \wedge M =_{\beta\eta} P \\ \text{then} & \exists Q (M \twoheadrightarrow_{\beta\eta} Q \wedge P \twoheadrightarrow_{\beta\eta} Q). \end{array}$$

Adding η to the conversion rule

Summarizing we have:

- Subject Reduction for β
- If β-reduction is weakly normalizing, then
  - Subject Reduction for η holds.
  - Church-Rosser for $\beta\eta$ on well-typed terms holds:

$$\begin{array}{rl} \text{If} & \Gamma \vdash M : A \wedge \Gamma \vdash P : A \wedge M =_{\beta\eta} P \\ \text{then} & \exists Q(M \twoheadrightarrow_{\beta\eta} Q \wedge P \twoheadrightarrow_{\beta\eta} Q). \end{array}$$

This is strange ... usually normalization is hard, while confluence is combinatorial

Can't we prove $\mathrm{CR}_{\beta\eta}$ for arbitrary PTSs?

The situation for $\mathrm{PTS}_{\beta\eta}$

If $\lambda *_{\beta\eta}$ has a fixed point combinator, then $\lambda *_{\beta\eta} \not\models CR_{\beta\eta}$.
(G. and Werner)

The situation for $\mathrm{PTS}_{\beta\eta}$

If $\lambda *_{\beta\eta}$ has a fixed point combinator, then $\lambda *_{\beta\eta} \not\models CR_{\beta\eta}$. (G. and Werner)

Proof. Let $Y : \Pi\alpha : \star.(\alpha \to \alpha) \to \alpha$ be the fixed-point comb. Let C, D, E be distinct types.

$$\begin{array}{rcl} A_c & := & Y\,(\lambda\beta : \star.\beta{\to}(C{\to}C){\to}E) \\ A_d & := & Y\,(\lambda\beta : \star.\beta{\to}(D{\to}D){\to}E) \end{array}$$

Then $A_c =_{\beta\eta} A_c{\to}(C{\to}C){\to}E$ (and idem for A_d).

$$\begin{array}{rcccl} M_c & := & \lambda x : A_c.x\,x & : & A_c \\ M_d & := & \lambda x : A_d.x\,x & : & A_d \end{array}$$

For $M_c\,M_c\,(\lambda z : C.z)$ (and similarly for $M_d\,M_d\,(\lambda z : D.z)$) the only reduction is

$$M_c\,M_c\,(\lambda z : C.z) \ \to_{\beta\eta}\ M_c\,M_c\,(\lambda z : C.z)$$

But $M_c\,M_c\,(\lambda z : C.z) =_{\beta\eta} M_d\,M_d\,(\lambda z : D.z)$, so we don't have $\mathrm{CR}_{\beta\eta}$.

Is there a fixed point combinator?

For the PTS_β case:

- Howe: in $\lambda *$, from Girard's paradox, we can derive a looping combinator:

$$\text{family of terms } (Y_i)_{i\in\mathbb{N}} : \Pi\alpha : \star.(\alpha \to \alpha) \to \alpha$$

  with $Y_i\, A\, f =_\beta f\,(Y_{i+1}\, A\, f)$.
  - This enables the definability of all partial recursive functions.
  - Coquand-Herbelin: use A-translation to extend to paradoxes in arbitrary "logical" PTSs.

Is there a fixed point combinator?

For the PTS_β case:

- Howe: in $\lambda *$, from Girard's paradox, we can derive a looping combinator:

$$\text{family of terms } (Y_i)_{i \in \mathbb{N}} : \Pi\alpha : \star.(\alpha \to \alpha) \to \alpha$$

  with $Y_i\, A\, f =_\beta f\, (Y_{i+1}\, A\, f)$.
  - This enables the definability of all partial recursive functions.
  - Coquand-Herbelin: use A-translation to extend to paradoxes in arbitrary "logical" PTSs.
- Hurkens' paradox: "simple" proof of inconsistency of λU^-; we can actually study the derived term $Y : \Pi\alpha : \star.(\alpha \to \alpha) \to \alpha$.
  - G., Pollack: it is a looping combinator
  - Barthe, Coquand: it is not a fixed-point combinator, but if we erase all domains in λ-abstractions, it is a fixed point combinator.
  - If we erase all type information, we get the untyped fixed-point combinator

$$Y := \omega\, (\lambda p\, q.f\, (q\, p\, q))\, \omega$$

    where $\omega = \lambda x.x\, x$. So $Y\, f \twoheadrightarrow_\beta f\, (Y\, f)$.

So, is there a fixed-point combinator?

- Yes ... (Barthe, Coquand) The domain-erased term from Hurkens' inconsistency proof is a fixed-point combinator, so the term is also a fixed-point combinator in $\lambda*_{\beta\eta}$:

  $$Y_i\, A\, f \twoheadrightarrow f(Y_{i+1}\, A\, f) =_{\beta\eta} f(Y_i\, A\, f)$$

  (By the Domain Lemma: $C[\lambda x : A.M] =_{\beta\eta} C[\lambda x : B.M]$.)
- No ... (G., Verkoelen) In λU^-, we cannot type an untyped λ-term of the shape

  $$(\lambda x. \dots (x\, x) \dots)\, (\lambda y. \dots (y\, y) \dots).$$

  So: Curry's fixed-point combinator $Y := \lambda f.(\lambda x.f\,(x\,x))(\lambda x.f\,(x\,x))$ and Turing's fixed-point combinator $\Theta := (\lambda x\, y.y\,(x\,x\,y))(\lambda x\, y.y\,(x\,x\,y))$ are not typable in λU^-.

Normalization

Is there a generic proof of normalization?

- Known proofs proceed by defining a saturated sets or candidat de réducibilité interpretation.
- These are proofs for strong normalization (SN)

Normalization

Is there a generic proof of normalization?

- Known proofs proceed by defining a saturated sets or candidat de réducibilité interpretation.
- These are proofs for strong normalization (SN)
- Terlouw:
  Given a PTS, define $\mathtt{TYPE} := \{A \mid \exists \mathsf{s} \in cS(\vdash A : \mathsf{s})\}$.
  Define the relation $\prec$ on well-typed terms as follows:

$$\text{If } P\,B\,\vec{C} \in \mathtt{TYPE}, \text{ then } B \prec P \text{ and } P\,B \prec P.$$

Theorem (Terlouw): If $\prec$ is well-founded, then the PTS is SN.

Weak and Strong Normalization

Observation

- All type theories we know are either SN or not WN ...
- The set of well-typed terms of a PTS is not just some subset of T: it is
  - closed under sub-terms
  - closed under reduction
  - closed under freezing of redexes

Weak and Strong Normalization

Observation

- All type theories we know are either SN or not WN ...
- The set of well-typed terms of a PTS is not just some subset of T: it is
  - closed under sub-terms
  - closed under reduction
  - closed under freezing of redexes

  Closure under freezing:

  If $C[(\lambda x : A.M)P]$ is well-typed, then $C[d\,P]$ is well-typed

  for some "neutral term" d. (So $d\,P$ cannot reduce.)

Weak and Strong Normalization

Observation

- All type theories we know are either SN or not WN ...
- The set of well-typed terms of a PTS is not just some subset of T: it is
  - closed under sub-terms
  - closed under reduction
  - closed under freezing of redexes

  Closure under freezing:

  $$\text{If } C[(\lambda x : A.M)P] \text{ is well-typed, then } C[d\,P] \text{ is well-typed}$$

  for some "neutral term" d. (So $d\,P$ cannot reduce.)

Conjecture: Every PTS that is WN is also SN.

Idea: if there is a well-typed term M that exhibits an infinite reduction path (M is not SN), then we can create out of M a well-typed term P that has only infinite reduction paths (P is not WN).

WN $\Longrightarrow$ SN?

Consider the following conjecture.
Given a set $X \subseteq \Lambda$ that is

1. closed under sub-terms
2. closed under reduction
3. closed under freezing of redexes

Then $X \models \mathsf{WN}_\beta \implies X \models \mathsf{SN}_\beta$.
NB. X is closed under freezing if

$$C[(\lambda x : A.M)P] \in X \Longrightarrow C[d\, P]$$

WN $\Longrightarrow$ SN?

Consider the following conjecture.
Given a set $X \subseteq \Lambda$ that is

1. closed under sub-terms
2. closed under reduction
3. closed under freezing of redexes

Then $X \models \mathsf{WN}_\beta \quad \Longrightarrow \quad X \models \mathsf{SN}_\beta$.
NB. X is closed under freezing if

$$C[(\lambda x : A.M)P] \in X \Longrightarrow C[d\, P]$$

Gonthier: this conjecture is false!
Counterexample: consider X to be the closure under 1, 2, 3 of

$$\{\omega\,(\lambda z.F(z\,u\,z\,z))\}$$

where $F = \lambda x\,y.y$ and u is a variable.

Revisiting Contexts

Traditional presentation of dependent type theory

- Terms considered with respect to an explicit context Γ

$$\Gamma \vdash M : A$$

- A **bound** variable is bound **locally** by a λ or Π
- A **free** variable is bound **globally** by Γ

Can we present dependent type theory without contexts?

Motivation

First-order logic and contexts

Predicate logic

$$\dfrac{\dfrac{A \vdash P(x)}{A \vdash \forall x. P(x)}}{\vdash A \to \forall x. P(x)}$$

'sea' of free variables

Type theory

$$\dfrac{\dfrac{H : A, x : D \vdash M_3 : P(x)}{H : A \vdash M_2 : \Pi x : D. P(x)}}{\vdash M_1 : A \to \Pi x : D. P(x)}$$

context of 'free' variables

What about?

$$(\forall x.\, P(x)) \to (\exists x.\, P(x))$$

Motivation

Theorem provers

- Correctness of a theorem prover based on the *LCF-architecture* relies on the **kernel**
- Kernels always have a **state**

  definitions from the formalization that already have been processed

- Corresponds to a context in the formal treatment

$$\Gamma \vdash M : A$$

Dependent Type Theory without Contexts

H.G., R. Krebbers, J. McKinna, F. Wiedijk, LFMTP 2010

- We simulate the sea of free variables
- Infinitely many variables x^A for each type A
- This gives an "infinite context" called Γ_∞
- For example

$$s^{N^* \to N^*}$$

Dependent Type Theory without Contexts

H.G., R. Krebbers, J. McKinna, F. Wiedijk, LFMTP 2010

- We simulate the sea of free variables
- Infinitely many variables x^A for each type A
- This gives an “infinite context” called Γ_∞
- For example

$$s^{N^* \to N^*}$$

- Variable carries history of how it comes to be well-typed
- Judgments of the shape $A : B$
- Should be imagined as $\Gamma_\infty \vdash A : B$

Labelled PTS terms

- Type labels should be considered as **strings**
- Labels are insensitive to α and β-conversion
- That is to say

$$x^A[A := B] \not\equiv x^B$$

and

$$\begin{array}{ccc} (\lambda \dot{A} : *.\dot{A})\, B^* & =_\beta & B^* \\ x^{(\lambda \dot{A}:*.\dot{A}) B^*} & \neq_\beta & x^{B^*} \end{array}$$

Labelled PTS terms

- Type labels should be considered as **strings**
- Labels are insensitive to α and β-conversion
- That is to say

$$x^A[A := B] \not\equiv x^B$$

  and

$$\begin{array}{ccc} (\lambda \dot{A} : *.\dot{A})\, B^* & =_\beta & B^* \\ x^{(\lambda \dot{A}:*.\dot{A})B^*} & \neq_\beta & x^{B^*} \end{array}$$

- But we **do** have (by type conversion)

$$x^{(\lambda \dot{A}:*.\dot{A})B^*} : B^*$$

- We avoid the need to consider substitution in labels of bound variables, e.g. in

$$(\lambda x^A \lambda P^{A\to *} \lambda y^{P^{A\to *} x^A} \ldots) a^A \to_\beta \lambda P^{A\to *} \lambda y^{P^{A\to *} a^A} \ldots$$

Typing rules

Two of the six rules

PTS rules | **Γ_∞ rules**

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\; x \notin \Gamma \qquad\qquad \frac{A : s}{x^A : A}$$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A.B : s_3} \qquad \frac{A : s_1 \qquad B : s_2}{\Pi \dot{x} : A.B[y^A := \dot{x}] : s_3}$$

Remark:

- Binding a variable in Γ_∞

  replace a free variable by a bound variable

- No weakening rule

But this does not correspond to PTSs!

Now we would have

$$\frac{x^{A^*} \ : \ A^*}{\lambda \dot{A} : *. x^{A^*} \ : \ \Pi \dot{A} : *. \dot{A}}$$

but, in ordinary PTS-style

$$\frac{A : *, x : A \ \vdash \ x \ : \ A}{x : A \ \vdash \ \lambda A : *. x \ : \ \Pi A : *. A}$$

which is nonsense because A^* occurs free in the type label of x.

Taking the type annotations seriously

It is not enough to consider the free variables in a type label, but the *hereditary* free variables of a type label.

$$\frac{A : s_1 \quad B : s_2}{\Pi \dot{x} : A.B[y^A := \dot{x}] : s_3}\ \textit{Incorrect}$$

Taking the type annotations seriously

It is not enough to consider the free variables in a type label, but the *hereditary* free variables of a type label.

$$\frac{A : s_1 \quad B : s_2}{\Pi \dot{x} : A . B[y^A := \dot{x}] : s_3} \qquad y^A \notin \mathrm{hfvT}(B)$$

Taking the type annotations seriously

It is not enough to consider the free variables in a type label, but the *hereditary* free variables of a type label.

$$\frac{A : s_1 \quad B : s_2}{\Pi\dot{x} : A.B[y^A := \dot{x}] : s_3} \qquad y^A \notin \mathrm{hfvT}(B)$$

$$\frac{M : B \quad \Pi\dot{x} : A.B[y^A := \dot{x}] : s}{\lambda\dot{x} : A.M[y^A := \dot{x}] : \Pi\dot{x} : A.B[y^A := \dot{x}]} \; y^A \notin \mathrm{hfvT}(M) \cup \mathrm{hfvT}(B)$$

Taking the type annotations seriously

Hereditary free type-variables are defined as

$$
\begin{array}{rcl}
\mathrm{hfvT}(s) = \mathrm{hfvT}(\dot{x}) & = & \emptyset \\
\mathrm{hfvT}(F\,N) & = & \mathrm{hfvT}(F) \cup \mathrm{hfvT}(N) \\
\mathrm{hfvT}(\lambda\dot{x} : A.N) = \mathrm{hfvT}(\Pi\dot{x} : A.N) & = & \mathrm{hfvT}(A) \cup \mathrm{hfvT}(N) \\
\mathrm{hfvT}(x^{A}) & = & \mathrm{hfv}(A)
\end{array}
$$

Where the **hereditary free variables** are defined as

$$
\begin{array}{rcl}
\mathrm{hfv}(s) = \mathrm{hfv}(\dot{x}) & = & \emptyset \\
\mathrm{hfv}(F\,N) & = & \mathrm{hfv}(F) \cup \mathrm{hfv}(N) \\
\mathrm{hfv}(\lambda\dot{x} : A.N) = \mathrm{hfv}(\Pi\dot{x} : A.N) & = & \mathrm{hfv}(A) \cup \mathrm{hfv}(N) \\
\mathrm{hfv}(x^{A}) & = & \{x^{A}\} \cup \mathrm{hfv}(A)
\end{array}
$$

Back to the example

$$\frac{x^{A^*} \ : \ A^*}{\lambda \dot{A} : *. x^{A^*} \ : \ \Pi \dot{A} : *. \dot{A}}$$

Not correct, because

$$A^* \in \mathrm{hfvT}(x^{A^*}) \cup \mathrm{hfvT}(A^*) = \{A^*\} \cup \emptyset$$

The correspondence theorems

derivable PTS judgment $\longleftrightarrow$ **derivable Γ_∞ judgment**

(α-)rename $\Gamma \vdash M : A$ to $\Gamma' \vdash M' : A'$ such that $\Gamma' \subset \Gamma_\infty$ and

$$\Gamma \vdash M : A \qquad \Longrightarrow \qquad M' : A'$$

for $M : A$ generate a context $\Gamma(M, A)$ such that

$$\Gamma(M, A) \vdash M : A \qquad \Longleftarrow \qquad M : A$$

Remarks

Advantages of the context-free approach:

- Strengthening is implicit
- Some theorems might be easier to prove
- Closer to LCF-style provers

Formalization in Coq

- One direction completely finished
- Locally nameless approach: bound variables are De Bruijn indices
- Suits distinction between variables well

Future work

- Γ_∞ presentation for other type theories, e.g. theories with definitions
- LCF-style kernel based on Γ_∞. Efficiency?

Revisiting the conversion rule

Three uses of β-reduction and the conversion rule in Logical Frameworks

1. To deal with substitution (and the proper renaming of bound vars etc).
2. For comprehension
3. To define functions as (executable) programs

- The first 2 are typically used in HOL and LF and involve β-reduction in simple type theory or first order dependent type theory, which is relatively easy.
- The third is available in CC and Coq, and used heavily for proof automation.

The conversion rule: examples in Church' HOL

Church' HOL:

$\forall x : \mathbb{N}. x + 0 = x$ is defined as $\forall(\lambda x : \mathbb{N}. x + 0 = x)$.

A derivation involving substitution:

$$\dfrac{\dfrac{\forall(\lambda x : \mathbb{N}. x + 0 = x)}{(\lambda x : \mathbb{N}. x + 0 = x)\,5}\ \forall\text{-elim}}{5 + 0 = 5}\ \beta\text{-conv}$$

The conversion rule: examples in Church' HOL

Church' HOL:

$\forall x : \mathbb{N}.x + 0 = x$ is defined as $\forall(\lambda x : \mathbb{N}.x + 0 = x)$.

A derivation involving substitution:

$$\dfrac{\dfrac{\forall(\lambda x : \mathbb{N}.x + 0 = x)}{(\lambda x : \mathbb{N}.x + 0 = x)\,5}\ \forall\text{-elim}}{5 + 0 = 5}\ \beta\text{-conv}$$

Comprehension: for all formulas φ:

$$\exists X \forall \vec{x}.X\,\vec{x} \leftrightarrow \varphi$$

In type theory this is easy, because we have $X := \lambda\vec{x}.\varphi$ available in the language.

$$\dfrac{\dfrac{\varphi \leftrightarrow \varphi}{(\lambda\vec{x}.\varphi)\,\vec{x} \leftrightarrow \varphi}\ \beta\text{-conv}}{\exists X \forall \vec{x}.X\,\vec{x} \leftrightarrow \varphi}\ \exists\text{-in}$$

More computation in the system

- Inductive types and (well-founded) recursive functions turn a PA (Coq, Matita, Agda, Nuprl, ...) into a programming language.
- This allows programming automated theorem proving techniques inside the system. (Via Reflection)
  To prove A, type-check

$$\texttt{reflexivity} \quad : \quad \texttt{solve}[\![A]\!] = \texttt{true}$$

More computation in the system

- Inductive types and (well-founded) recursive functions turn a PA (Coq, Matita, Agda, Nuprl, ...) into a programming language.
- This allows programming automated theorem proving techniques inside the system. (Via Reflection)
  To prove A, type-check

  $$\texttt{reflexivity} \quad : \quad \texttt{solve}[\![A]\!] = \texttt{true}$$

- When the power of this was first shown to Per Martin-Löf (Kloster Irsee 1998), his first reaction was ...
  "But these aren't proofs!"

How can we believe a proof assistant?

- Check the checker. Verify the correctness of the PA inside the system itself, or in another system.

How can we believe a proof assistant?

- Check the checker. Verify the correctness of the PA inside the system itself, or in another system.
- The De Bruijn criterion

De Bruijn, July 9, 1918 - February 17, 2012

Some PAs generate proof objects that can be checked independently from the system by a simple program that a skeptical user could write him/herself.

Back to simple (linear time?) type-checking?

Storing a trace of the conversion in the proof-term

- A PTS_f is a PTS with conversion replaced by the following rule

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash B : \mathsf{s} \quad \Gamma \vdash H : A = B}{\Gamma \vdash t^H : B}\ (\text{conv})$$

- In addition we have rules to construct expressions H to record the *conversion trace* between A and B, $H : A = B$. This H is meant to encode the β-conversion-path between A and B.
- The terms $H : A = B$ are also well-typed in a context Γ.
- In λH, type-checking is linear

F. van Doorn, H.G. and F. Wiedijk – Explicit Convertibility Proofs in Pure Type Systems, LFMTP 2013

Rules for constructing equality proof terms

We have the term-construction rules for reflexivity, symmetry and transitivity

$$\frac{\Gamma \vdash_f A : B}{\Gamma \vdash_f \overline{A} : A = A}$$

$$\frac{\Gamma \vdash_f H : A = A'}{\Gamma \vdash_f H^\dagger : A' = A}$$

$$\frac{\Gamma \vdash_f H : A = A' \qquad \Gamma \vdash_f H' : A' = A''}{\Gamma \vdash_f H \cdot H' : A = A''}$$

Some of the rules for constructing equality proof terms

We have the term-construction rules for Π-types, the β-rule and a rule to erase equality proof-annotations

$$\frac{\begin{array}{ll} \Gamma \vdash_f A : s_1 & \Gamma, x : A \vdash_f B : s_2 \\ \Gamma \vdash_f A' : s_1' & \Gamma, x' : A' \vdash_f B' : s_2' \\ \Gamma \vdash_f H : A = A' & \Gamma, x : A \vdash_f H' : B = B'[x' := x^H] \end{array}}{\Gamma \vdash_f \{H, [x : A]H'\} : \Pi x{:}A.B = \Pi x'{:}A'.B'} \begin{array}{l} (s_1, s_2, s_3) \in \mathcal{R} \\ (s_1', s_2', s_3') \in \mathcal{R} \end{array}$$

$$\frac{\Gamma \vdash_f a : A : s_1 \qquad \Gamma, x : A \vdash_f b : B : s_2}{\Gamma \vdash_f \beta((\lambda x{:}A.b)a) : (\lambda x{:}A.b)a = b[x := a]} \; (s_1, s_2, s_3) \in \mathcal{R}$$

$$\frac{\Gamma \vdash_f a : A \qquad \Gamma \vdash_f A' : s \qquad \Gamma \vdash_f H : A = A'}{\Gamma \vdash_f \iota(a^H) : a = a^H}$$

Soundness and Completeness of PTS_f with respect to PTS

Let $| - |$ be the map that erases all equality-proof annotations.
If $|A'| = A$, we call A' a lift of A.

- (Soundness, easy) If $\Gamma \vdash_f M : A$, then $|\Gamma| \vdash |M| : |A|$.

Soundness and Completeness of PTS_f with respect to PTS

Let $|-|$ be the map that erases all equality-proof annotations.
If $|A'| = A$, we call A' a lift of A.

- (Soundness, easy) If $\Gamma \vdash_f M : A$, then $|\Gamma| \vdash |M| : |A|$.
- (Completeness for typing, hard!) If $\Gamma \vdash A : B$, then there are lifts Γ' , A' and B' of Γ , A and B, such that

$$\Gamma' \vdash_f A' : B'$$

- (Completeness for equality, hard!) If $\Gamma \vdash A : C$, $\Gamma \vdash A : D$ and $A =_\beta B$, then there are lifts Γ' , A' and B' of Γ , A and B and a term H such that

$$\Gamma' \vdash_f H : A' = B'$$

All proofs have been completely formalized in Coq by F. van Doorn.

Questions?

Advertisement: our book
"Type Theory and Formal Proof" will appear with CUP in 2014.
(Authors: Rob Nederpelt, Herman Geuvers)