Embedding logics in the $\lambda\Pi$-calculus modulo rewriting

Ali Assaf
Inria Paris-Rocquencourt (Deducteam)
Ecole polytechnique

Stockholm logic seminar
October 21, 2014

Motivation

Many different proof assistants:

- HOL Light
- Coq
- Mizar
- ...

Motivation

Many different proof assistants:

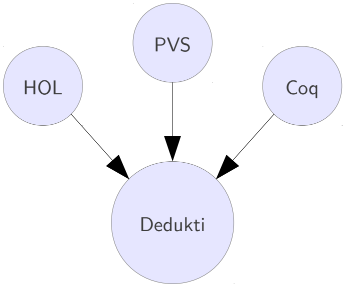
- HOL Light
- Coq
- Mizar
- ...

Many different formalisms:

- Simple type theory
- Calculus of inductive constructions
- Set theory
- ...

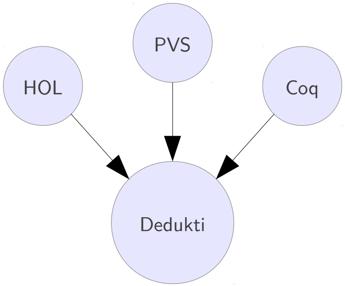
Motivation

A universal proof checker: Dedukti



Motivation

A universal proof checker: Dedukti



A universal framework: the $\lambda\Pi$-calculus modulo rewriting

Universal proof checker

Source: HOL, Coq, ...

- Pure type systems, inductive types, universes...
- Proof reconstruction, proof search, ...

Target: Dedukti

- $\lambda\Pi$-calculus modulo rewriting
- Proof checking (no proof reconstruction, no proof search, ...)

Deduction modulo

First-order logic modulo congruence

$$\frac{\Gamma \vdash A \qquad A \equiv B}{\Gamma \vdash B}$$

A theory is expressed by axioms + rewrite rules

Deduction modulo

First-order logic modulo congruence

$$\frac{\Gamma \vdash A \qquad A \equiv B}{\Gamma \vdash B}$$

A theory is expressed by axioms + rewrite rules

Example

The property

$$\forall x \forall y, \mathsf{s}\, x = \mathsf{s}\, y \iff x = y$$

can be expressed by the rewrite rule

$$\mathsf{s}\, x = \mathsf{s}\, y \longrightarrow x = y$$

Deduction modulo

First-order logic modulo congruence

$$\frac{\Gamma \vdash A \qquad A \equiv B}{\Gamma \vdash B}$$

A theory is expressed by axioms + rewrite rules

Example

The property

$$\forall x \forall y, \mathsf{s}\, x = \mathsf{s}\, y \iff x = y$$

can be expressed by the rewrite rule

$$\mathsf{s}\, x = \mathsf{s}\, y \longrightarrow x = y$$

Idea: replace axioms by rewrite rules

- Give computational meaning
- Preserve constructivism (disjunction & witness property)

From deduction modulo to $\lambda\Pi$-modulo

Curry-Howard correspondence

- Propositions $\longleftrightarrow$ types
- Proofs $\longleftrightarrow$ terms

From deduction modulo to $\lambda\Pi$-modulo

Curry-Howard correspondence

- Propositions $\longleftrightarrow$ types
- Proofs $\longleftrightarrow$ terms

Minimal first-order logic: $\lambda\Pi$-calculus

- Congruence modulo β
- Implemented in Twelf

From deduction modulo to $\lambda\Pi$-modulo

Curry-Howard correspondence

- Propositions $\longleftrightarrow$ types
- Proofs $\longleftrightarrow$ terms

Minimal first-order logic: $\lambda\Pi$-calculus

- Congruence modulo β
- Implemented in Twelf

Minimal deduction modulo: $\lambda\Pi$-calculus modulo rewriting

- Congruence modulo βR
- Implemented in Dedukti

The $\lambda\Pi$-calculus modulo rewriting

An extension of the $\lambda\Pi$-calculus with rewrite rules

- Typed λ-calculus (Curry-Howard correspondence)
- Dependent types
- Rewriting to express equivalence

The $\lambda\Pi$-calculus modulo rewriting

An extension of the $\lambda\Pi$-calculus with rewrite rules

- Typed λ-calculus (Curry-Howard correspondence)
- Dependent types
- Rewriting to express equivalence

A variation of the logical framework of Martin-Löf

- Equalities oriented into rewrite rules
- Confluence + normalization $\implies$ decidable checking
- Efficient checking algorithm (Boespflug 2012, Saillard 2013)

Martin-Löf's logical framework

Type formation: A **Type**

$$\frac{A \ \textbf{Type} \qquad B \ \textbf{Type}}{A \times B \ \textbf{Type}}$$

Term formation (intro/elim): $M : A$

$$\frac{M : A \qquad N : A}{(M, N) : A \times B}$$

$$\frac{M : A \times B}{\mathsf{fst}\, M : A} \qquad \frac{M : A \times B}{\mathsf{snd}\, M : B}$$

Martin-Löf's logical framework

Type equality: $A \equiv B$

$$\mathsf{T}_{i+1}\,\mathsf{u}_i \equiv \mathsf{U}_i$$

Term equality: $M \equiv N : A$

$$\mathsf{fst}\,(M, N) \equiv M : A \qquad \mathsf{snd}\,(M, N) \equiv N : B$$

Towards formalism

Variables:

- explicit context Γ

Arities:

- currying $A_1 \to \ldots \to A_n \to B$
- kinds $A_1 \to \ldots \to A_n \to \mathbf{Type}$
- unifying terms and types

Syntax

sorts	s	$::=$	$\textbf{Type} \mid \textbf{Kind}$
terms	A, B, M, N	$::=$	$x \mid s \mid \Pi x : A.\, B \mid \lambda x : A.\, M \mid M\, N$
contexts	Γ	$::=$	$\cdot \mid \Gamma, x : A$

Typing rules

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{}{\Gamma \vdash \textbf{Type} : \textbf{Kind}}$$

$$\frac{\Gamma \vdash A : \textbf{Type} \qquad \Gamma, x : A \vdash B : s}{\Gamma \vdash \Pi x : A.\, B : s}$$

$$\frac{\Gamma \vdash A : \textbf{Type} \qquad \Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : \{N/x\}\, B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s \qquad A \equiv B}{\Gamma \vdash M : B}$$

Restrictions on rewrite rules

$$(\Gamma)\ \ M \longrightarrow N$$

Restrictions on rewrite rules

$$(\Gamma) \;\; M \longrightarrow N$$

Restrictions:

- Subject reduction for $\longrightarrow_{\beta R}$: $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$

Restrictions on rewrite rules

$$(\Gamma)\ \ M \longrightarrow N$$

Restrictions:

- Subject reduction for $\longrightarrow_{\beta R}$: $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$
- Confluence for $\longrightarrow_{\beta R}$: $FV(N) \subseteq FV(M)$ + no divergent critical pair

Restrictions on rewrite rules

$$(\Gamma)\ \ M \longrightarrow N$$

Restrictions:

- Subject reduction for $\longrightarrow_{\beta R}$: $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$
- Confluence for $\longrightarrow_{\beta R}$: $FV(N) \subseteq FV(M)$ + no divergent critical pair
- Normalization for $\longrightarrow_{\beta R}$: ???

Summary

- $\lambda\Pi$-calculus modulo = dependent types + rewrite rules
- Decidable type-checking under certain conditions

Using $\lambda\Pi$ as a logical framework

Logical framework in Wikipedia:

> *In logic, a logical framework provides a means to define (or present) a logic as a signature in a higher-order type theory in such a way that provability of a formula in the original logic reduces to a type inhabitation problem in the framework type theory.*

Using $\lambda\Pi$ as a logical framework

Logical framework in Wikipedia:

> *In logic, a logical framework provides a means to define (or present) a logic as a signature in a higher-order type theory in such a way that provability of a formula in the original logic reduces to a type inhabitation problem in the framework type theory.*

To embed a given theory X in $\lambda\Pi$, one must:

1. define a *signature* context Σ in $\lambda\Pi$ describing the theory X
2. define a *translation* from the terms of X to the terms of $\lambda\Pi$ in the context Σ.

System F in $\lambda\Pi$

Define the signature context Σ as:

$$
\begin{array}{lcl}
\mathsf{type} & : & \mathbf{Type} \\
\mathsf{arrow} & : & \mathsf{type} \to \mathsf{type} \to \mathsf{type} \\
\mathsf{forall} & : & (\mathsf{type} \to \mathsf{type}) \to \mathsf{type} \\
\\
\mathsf{term} & : & \mathsf{type} \to \mathbf{Type} \\
\mathsf{lam} & : & (\mathsf{term}\, A \to \mathsf{term}\, B) \to \mathsf{term}\,(\mathsf{arrow}\, A\, B) \\
\mathsf{app} & : & \mathsf{term}\,(\mathsf{arrow}\, A\, B) \to \mathsf{term}\, A \to \mathsf{term}\, B \\
\mathsf{Lam} & : & (\Pi A : \mathsf{type}.\, \mathsf{term}\,(F\, A)) \to \mathsf{term}\,(\mathsf{forall}\, F) \\
\mathsf{App} & : & \mathsf{term}\,(\mathsf{forall}\, F) \to \Pi A : \mathsf{type}.\, \mathsf{term}\,(F\, A)
\end{array}
$$

System F in $\lambda\Pi$

Translate the types and the terms as:

$$\begin{array}{rcl}
[\alpha] & = & \alpha \\
[A \to B] & = & \mathsf{arrow}\,[A]\,[B] \\
[\forall\alpha : \mathbf{Type}.\, B] & = & \mathsf{forall}\,(\lambda\alpha : \mathsf{type}.\,[B])
\end{array}$$

$$\begin{array}{rcl}
[x] & = & x \\
[\lambda x : A.\, M] & = & \mathsf{lam}\,(\lambda x : \mathsf{term}\,[A].\,[M]) \\
[M\, N] & = & \mathsf{app}\,[M]\,[N] \\
[\Lambda\alpha : \mathbf{Type}.\, M] & = & \mathsf{Lam}\,(\lambda\alpha : \mathsf{type}.\,[M]) \\
[M\,\langle A\rangle] & = & \mathsf{App}\,[M]\,[A]
\end{array}$$

System F in $\lambda\Pi$

Example

The identity function $\mathsf{id} = \Lambda\alpha : \mathbf{Type}.\, \lambda x : \alpha.\, x$ is translated as:

$$[\mathsf{id}] \quad = \quad \mathsf{Lam}\,(\lambda\alpha : \mathsf{type}.\, \mathsf{lam}\,(\lambda x : \mathsf{term}\,\alpha.\, x))$$

The type $A = \forall\alpha : \mathbf{Type}.\, \alpha \to \alpha$ is translated as:

$$[A] \quad = \quad \mathsf{forall}\,(\lambda\alpha : \mathsf{type}.\, \mathsf{arrow}\,\alpha\,\alpha)$$

Completeness

If M is well-typed then $[M]$ is well-typed in the context Σ:

$$\vdash M : A \quad \Longrightarrow \quad \Sigma \vdash [M] : \mathsf{term}\,[A]$$

Completeness

If M is well-typed then $[M]$ is well-typed in the context Σ:

$$\vdash M : A \quad \Longrightarrow \quad \Sigma \vdash [M] : \mathsf{term}\,[A]$$

Define $[\![A]\!] = \mathsf{term}\,[A]$:

$$\vdash M : A \quad \Longrightarrow \quad \Sigma \vdash [M] : [\![A]\!]$$

Completeness

If M is well-typed then $[M]$ is well-typed in the context Σ:

$$\vdash M : A \quad \Longrightarrow \quad \Sigma \vdash [M] : \mathsf{term}\,[A]$$

Define $[\![A]\!] = \mathsf{term}\,[A]$:

$$\vdash M : A \quad \Longrightarrow \quad \Sigma \vdash [M] : [\![A]\!]$$

If M is well-typed in Γ then $[M]$ is well-typed in the context $\Sigma, [\![\Gamma]\!]$:

$$\Gamma \vdash M : A \quad \Longrightarrow \quad \Sigma, [\![\Gamma]\!] \vdash [M] : [\![A]\!]$$

System F in $\lambda\Pi$

Example

The self-application of id is well-typed in the empty context:

$$\vdash \mathsf{id}\,\langle A \rangle\,\mathsf{id} : A$$

Its translation is well-typed in Σ:

$$\Sigma \vdash \mathsf{app}\,(\mathsf{App}\,[\mathsf{id}]\,[\![A]\!])\,[\mathsf{id}] : [\![A]\!]$$

Completeness

1. If M is a proof of (has type) A in X then $[M]$ is a proof of (has type) $[\![A]\!]$ in $\lambda\Pi$.

Completeness

1. If M is a proof of (has type) A in X then $[M]$ is a proof of (has type) $[\![A]\!]$ in $\lambda\Pi$.
2. If A is provable (is inhabited) in X then $[\![A]\!]$ is provable (is inhabited) in $\lambda\Pi$.

Completeness

1. If M is a proof of (has type) A in X then $[M]$ is a proof of (has type) $[\![A]\!]$ in $\lambda\Pi$.
2. If A is provable (is inhabited) in X then $[\![A]\!]$ is provable (is inhabited) in $\lambda\Pi$.
3. If X is inconsistent (every A is inhabited) then $\lambda\Pi$ is inconsistent (every $[\![A]\!]$ is inhabited).

Completeness

1. If M is a proof of (has type) A in X then $[M]$ is a proof of (has type) $[\![A]\!]$ in $\lambda\Pi$.
2. If A is provable (is inhabited) in X then $[\![A]\!]$ is provable (is inhabited) in $\lambda\Pi$.
3. If X is inconsistent (every A is inhabited) then $\lambda\Pi$ is inconsistent (every $[\![A]\!]$ is inhabited).

What about the converse?

Soundness

1. **Consistency:** if X is consistent then $\lambda\Pi$ is consistent.

Soundness

1. **Consistency:** if X is consistent then $\lambda\Pi$ is consistent.
2. **Conservativity:** if $[\![A]\!]$ is provable in $\lambda\Pi$ then A is provable in X.

Soundness

1. **Consistency:** if X is consistent then $\lambda\Pi$ is consistent.
2. **Conservativity:** if $[\![A]\!]$ is provable in $\lambda\Pi$ then A is provable in X.
3. **Adequacy:** every (normal) proof in $\lambda\Pi$ corresponds to a proof in X.

Soundness

1. **Consistency:** if X is consistent then $\lambda\Pi$ is consistent.
2. **Conservativity:** if $[\![A]\!]$ is provable in $\lambda\Pi$ then A is provable in X.
3. **Adequacy:** every (normal) proof in $\lambda\Pi$ corresponds to a proof in X.

These are important properties for a logical framework!

Summary

- **Source** $= X$, **Target** $= \lambda\Pi$
- **Embedding** $=$ signature Σ $+$ translation $[\cdot]$
- **Completeness** $=$ typing in $X \implies$ typing in $\lambda\Pi$
- **Soundness** $=$ typing in $\lambda\Pi \implies$ typing in X

Limitations of $\lambda\Pi$

The embedding does not preserve term (proof) reduction :

$$M \longrightarrow^* M' \quad \not\Longrightarrow \quad [M] \longrightarrow^* \left[M'\right]$$

Limitations of $\lambda\Pi$

The embedding does not preserve term (proof) reduction :

$$M \longrightarrow^* M' \quad \not\Longrightarrow \quad [M] \longrightarrow^* \left[M'\right]$$

The embedding does not preserve term (proof) equivalence:

$$M \equiv M' \quad \not\Longrightarrow \quad [M] \equiv \left[M'\right]$$

Limitations of $\lambda\Pi$

Systems with dependent types (e.g. the calculus of constructions) have a conversion rule:

$$\frac{\Gamma \vdash M : A \qquad A \equiv B}{\Gamma \vdash M : B}$$

In $\lambda\Pi$, $[\![\Gamma]\!] \vdash [M] : [\![A]\!]$ but $[\![\Gamma]\!] \not\vdash [M] : [\![B]\!]$ (no completeness).

Conversion in $\lambda\Pi$

Approach 1: Introduce explicit equivalence judgements and a conversion term:

$$\begin{array}{rcl} \mathsf{equiv} & : & \mathsf{type} \to \mathsf{type} \to \mathbf{Type} \\ \mathsf{refl} & : & \mathsf{equiv}\, M\, M \\ \mathsf{beta} & : & \mathsf{equiv}\, (\mathsf{app}\, (\mathsf{lam}\, F)\, N)\, (F\, N) \\ \dots & & \\ \mathsf{conv} & : & \mathsf{term}\, A \to \mathsf{equiv}\, \mathsf{A}\, \mathsf{B} \to \mathsf{term}\, B \end{array}$$

Cons:

- Need to explicitely give the equivalence derivations.
- Adding conv pollutes the structure of the terms and needs to be taken care of in the equivalence relation.

Conversion in $\lambda\Pi$

Approach 2: Translate typing derivations instead of λ-terms

$$
\begin{array}{rcl}
\text{term} & : & \mathbf{Type} \\
\text{lam} & : & (\text{term} \to \text{term}) \to \text{term} \\
\ldots & & \\
\text{hastype} & : & \text{term} \to \text{type} \to \mathbf{Type} \\
\text{typelam} & : & (\Pi x : \text{term}.\, \text{hastype}\, x\, A \to \text{hastype}\, (F\, x)\, B) \to \\
 & & \text{hastype}\, (\text{lam}\, F)\, (\text{arrow}\, A\, B) \\
\ldots & &
\end{array}
$$

Pros:

- conv does not interfere with the structure of the λ-terms.

Cons:

- Lose Curry-Howard correspondence?
- Still need to explicitely give the equivalence derivations.

The $\lambda\Pi$-calculus modulo rewriting

Idea: extend the conversion rule of the $\lambda\Pi$-calculus with a rewrite system R:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \mathbf{Type} \qquad A \equiv_{\beta R} B}{\Gamma \vdash M : B}$$

The $\lambda\Pi$-calculus modulo rewriting

Idea: extend the conversion rule of the $\lambda\Pi$-calculus with a rewrite system R:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \textbf{Type} \qquad A \equiv_{\beta R} B}{\Gamma \vdash M : B}$$

Add rewrite rules so that the translation preserves reduction.

The $\lambda\Pi$-calculus modulo rewriting

Idea: extend the conversion rule of the $\lambda\Pi$-calculus with a rewrite system R:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : \mathbf{Type} \qquad A \equiv_{\beta R} B}{\Gamma \vdash M : B}$$

Add rewrite rules so that the translation preserves reduction (in addition to binding and typing).

Preserving reduction

Signature context Σ:

$$\begin{array}{rcl} \mathsf{type} & : & \mathbf{Type} \\ \mathsf{arrow} & : & \mathsf{type} \to \mathsf{type} \to \mathsf{type} \\ \\ \mathsf{term} & : & \mathsf{type} \to \mathbf{Type} \\ \mathsf{lam} & : & (\mathsf{term}\, A \to \mathsf{term}\, B) \to \mathsf{term}\,(\mathsf{arrow}\, A\, B) \\ \mathsf{app} & : & \mathsf{term}\,(\mathsf{arrow}\, A\, B) \to \mathsf{term}\, A \to \mathsf{term}\, B \end{array}$$

Rewrite rules R:

$$\mathsf{app}\,(\mathsf{lam}\, F)\, N \quad \longrightarrow \quad F\, N$$

Preserving reduction

Signature context Σ:

$$\begin{array}{rcl} \mathsf{type} & : & \mathbf{Type} \\ \mathsf{arrow} & : & \mathsf{type} \to \mathsf{type} \to \mathsf{type} \\ \\ \mathsf{term} & : & \mathsf{type} \to \mathbf{Type} \\ \mathsf{lam} & : & (\mathsf{term}\, A \to \mathsf{term}\, B) \to \mathsf{term}\,(\mathsf{arrow}\, A\, B) \\ \mathsf{app} & : & \mathsf{term}\,(\mathsf{arrow}\, A\, B) \to \mathsf{term}\, A \to \mathsf{term}\, B \end{array}$$

Rewrite rules R:

$$\begin{array}{rcl} \mathsf{term}\,(\mathsf{arrow}\, A\, B) & \longrightarrow & \mathsf{term}\, A \to \mathsf{term}\, B \\ \mathsf{lam}\, F & \longrightarrow & F \\ \mathsf{app}\, M\, N & \longrightarrow & M\, N \end{array}$$

Preserving reduction

Signature context Σ:

$$\begin{array}{rcl} \mathsf{type} & : & \mathbf{Type} \\ \mathsf{arrow} & : & \mathsf{type} \to \mathsf{type} \to \mathsf{type} \\ \\ \mathsf{term} & : & \mathsf{type} \to \mathbf{Type} \end{array}$$

Rewrite rules R:

$$\mathsf{term}\,(\mathsf{arrow}\,A\,B) \quad \longrightarrow \quad \mathsf{term}\,A \to \mathsf{term}\,B$$

Translation:

$$\begin{array}{rcl} [\lambda x : A.\, M] & = & \lambda x : [\![A]\!].\, [M] \\ [M\,N] & = & [M]\,[N] \end{array}$$

Preserving reduction

Theorem
If $M \longrightarrow M'$ *then* $[M] \longrightarrow^{+} [M']$.

Preserving reduction

Theorem

If $M \longrightarrow M'$ *then* $[M] \longrightarrow^{+} [M']$.

Corollary

If $M \longrightarrow^{*} M'$ *then* $[M] \longrightarrow^{*} [M']$.

Preserving reduction

Theorem
If $M \longrightarrow M'$ *then* $[M] \longrightarrow^{+} [M']$.

Corollary
If $M \longrightarrow^{*} M'$ *then* $[M] \longrightarrow^{*} [M']$.

Corollary
If $M \equiv M'$ *then* $[M] \equiv [M']$.

Completeness

Recovered typing preservation.

Theorem (Cousineau & Dowek 2007)

If $\Gamma \vdash M : A$ *then* $\Sigma, [\![\Gamma]\!] \vdash [M] : [\![A]\!]$.

Works for any functional pure type system:

- System F
- Calculus of constructions
- Simple type theory

Completeness

Recovered typing preservation.

Theorem (Cousineau & Dowek 2007)

If $\Gamma \vdash M : A$ *then* $\Sigma, \llbracket \Gamma \rrbracket \vdash [M] : \llbracket A \rrbracket$.

Works for any functional pure type system:

- System F
- Calculus of constructions
- Simple type theory

What about soundness?

On termination and soundness

Link between termination and soundness:

- $\lambda\Pi$ is strongly normalizing

On termination and soundness

Link between termination and soundness:

- $\lambda\Pi$ is strongly normalizing
- Adding axioms (Σ) does not influence termination

On termination and soundness

Link between termination and soundness:

- $\lambda\Pi$ is strongly normalizing
- Adding axioms (Σ) does not influence termination
- Can be used to show soundness:
  - **Consistency:** there is no normal term of type $[\![\bot]\!]$

On termination and soundness

Link between termination and soundness:

- $\lambda\Pi$ is strongly normalizing
- Adding axioms (Σ) does not influence termination
- Can be used to show soundness:
  - **Consistency:** there is no normal term of type $\llbracket \bot \rrbracket$
  - **Adequacy:** if $\llbracket \Gamma \rrbracket \vdash M : \llbracket A \rrbracket$ and M is a normal form, then $M = [N]$ for some N such that $\Gamma \vdash N : A$

On termination and soundness

Link between termination and soundness:

- $\lambda\Pi$ is strongly normalizing
- Adding axioms (Σ) does not influence termination
- Can be used to show soundness:
  - **Consistency:** there is no normal term of type $[\![\bot]\!]$
  - **Adequacy:** if $[\![\Gamma]\!] \vdash M : [\![A]\!]$ and M is a normal form, then $M = [N]$ for some N such that $\Gamma \vdash N : A$
  - **Conservativity:** if $[\![\Gamma]\!] \vdash M : [\![A]\!]$ then M reduces to a normal form $[N]$ for some N such that $\Gamma \vdash N : A$.

On termination and soundness

Adding rewrite rules (R) can break strong normalization:

- because $\longrightarrow_R$ does not terminate

On termination and soundness

Adding rewrite rules (R) can break strong normalization:

- because $\longrightarrow_R$ does not terminate
- or because $\longrightarrow_R \cup \longrightarrow_\beta$ does not terminate

On termination and soundness

Adding rewrite rules (R) can break strong normalization:

- because $\longrightarrow_R$ does not terminate
- or because $\longrightarrow_R \cup \longrightarrow_\beta$ does not terminate
- or even because $\longrightarrow_\beta$ does not terminate for well-typed terms

On termination and soundness

Adding rewrite rules (R) can break strong normalization:

- because $\longrightarrow_R$ does not terminate
- or because $\longrightarrow_R \cup \longrightarrow_\beta$ does not terminate
- or even because $\longrightarrow_\beta$ does not terminate for well-typed terms

Need to find other solutions.

Summary

- $\lambda\Pi$ embeddings do not preserve reduction.
- Obstacle for embedding theories with dependent types.
- Adding rewrite rules to $\lambda\Pi$ helps recover completeness...
- ... but can break soundness.

Approach 1: models

Idea: build a model for $\lambda\Pi/X$

- in the algebra of reducibility candidates
- or in a general notion of Π-algebra.

Approach 1: models

Idea: build a model for $\lambda\Pi/X$

- in the algebra of reducibility candidates
- or in a general notion of Π-algebra.

The model implies strong normalization of $\lambda\Pi/X$. Use this to prove soundness (consistency, adequacy, conservativity).

Approach 1: models

Idea: build a model for $\lambda\Pi/X$

- in the algebra of reducibility candidates
- or in a general notion of Π-algebra.

The model implies strong normalization of $\lambda\Pi/X$. Use this to prove soundness (consistency, adequacy, conservativity).

Theorem (Dowek 2014)

There is a model for $\lambda\Pi/\mathbb{STT}$ and for $\lambda\Pi/\mathbb{COC}$.

Approach 1: models

Idea: build a model for $\lambda\Pi/X$

- in the algebra of reducibility candidates
- or in a general notion of Π-algebra.

The model implies strong normalization of $\lambda\Pi/X$. Use this to prove soundness (consistency, adequacy, conservativity).

Theorem (Dowek 2014)

There is a model for $\lambda\Pi/\mathbb{STT}$ *and for* $\lambda\Pi/\mathbb{COC}$.

Problem: implies strong normalization in X, so at least as hard to prove as strong normalization in X.

Approach 2: relative normalization

If $[\![\Gamma]\!] \vdash M : [\![A]\!]$, what can we say about M?

Approach 2: relative normalization

If $[\![\Gamma]\!] \vdash M : [\![A]\!]$, what can we say about M?

Example

If X is the simply-typed λ-calculus, the polymorphic identity function is not well-typed:

$$\beta : \mathbf{Type} \nvdash (\lambda\alpha : \mathbf{Type}.\, \lambda x : \alpha.\, x)\, \beta : \beta \to \beta$$

It is well-typed in $\lambda\Pi/X$:

$$\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\, \alpha.\, x)\, \beta : \mathsf{term}\, \beta \to \mathsf{term}\, \beta$$

Approach 2: relative normalization

If $[\![\Gamma]\!] \vdash M : [\![A]\!]$, what can we say about M?

Example

If X is the simply-typed λ-calculus, the polymorphic identity function is not well-typed:

$$\beta : \mathbf{Type} \nvdash (\lambda\alpha : \mathbf{Type}.\, \lambda x : \alpha.\, x)\, \beta : \beta \to \beta$$

It is well-typed in $\lambda\Pi/X$:

$$\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\, \alpha.\, x)\, \beta : \mathsf{term}\, \beta \to \mathsf{term}\, \beta$$

But it reduces to $\lambda x : \mathsf{term}\, \beta.\, x = [\lambda x : \beta.\, x]$, a term that is well-typed in X.

Approach 2: relative normalization

If $[\![\Gamma]\!] \vdash M : [\![A]\!]$, what can we say about M?

Example

If X is the simply-typed λ-calculus, the polymorphic identity function is not well-typed:

$$\beta : \mathbf{Type} \nvdash (\lambda\alpha : \mathbf{Type}.\, \lambda x : \alpha.\, x)\, \beta : \beta \to \beta$$

It is well-typed in $\lambda\Pi/X$:

$$\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\, \alpha.\, x)\, \beta : \mathsf{term}\, \beta \to \mathsf{term}\, \beta$$

But it reduces to $\lambda x : \mathsf{term}\, \beta.\, x = [\lambda x : \beta.\, x]$, a term that is well-typed in X.
Idea: reduce only what is necessary.

Erasure

Define an erasure from $\lambda\Pi/X$ to X:

$$\begin{array}{rcl} |x| & = & x \\ |\lambda x : A.\, M| & = & \lambda x : \|A\|.\, |M| \\ |M\,N| & = & |M|\,|N| \end{array}$$

$$\begin{array}{rcl} \|\mathsf{type}\| & = & \mathbf{Type} \\ \|\mathsf{term}\, A\| & = & |A| \\ \|A \to B\| & = & \|A\| \to \|B\| \end{array}$$

Erasure is the inverse of the translation:

$$\begin{array}{rcl} |[M]| & = & M \\ \|[\![A]\!]\| & = & A \end{array}$$

Proving soundness

What statement should we prove?

- If $[\![\Gamma]\!] \vdash M : [\![A]\!]$ in $\lambda\Pi/X$ then $\Gamma \vdash |M| : A$ in X?

Proving soundness

What statement should we prove?

- ~~If $[\![\Gamma]\!] \vdash M : [\![A]\!]$ in $\lambda\Pi/X$ then $\Gamma \vdash |M| : A$ in X?~~

  $\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\, \alpha.\, x)\, \beta : \mathsf{term}\, \beta \to \mathsf{term}\, \beta$

Proving soundness

What statement should we prove?

- ~~If $[\![\Gamma]\!] \vdash M : [\![A]\!]$ in $\lambda\Pi/X$ then $\Gamma \vdash |M| : A$ in X?~~

  $\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\, \alpha.\, x)\, \beta : \mathsf{term}\, \beta \to \mathsf{term}\, \beta$

- If $[\![\Gamma]\!] \vdash M : [\![A]\!]$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ such that $\Gamma \vdash |M'| : A$ in X?

Proving soundness

What statement should we prove?

- ~~If $\llbracket \Gamma \rrbracket \vdash M : \llbracket A \rrbracket$ in $\lambda\Pi/X$ then $\Gamma \vdash |M| : A$ in X?~~

  $\beta : \mathsf{type} \vdash (\lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\,\alpha.\, x)\,\beta : \mathsf{term}\,\beta \to \mathsf{term}\,\beta$

- ~~If $\llbracket \Gamma \rrbracket \vdash M : \llbracket A \rrbracket$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ such that $\Gamma \vdash |M'| : A$ in X?~~

$$\frac{\llbracket \Gamma \rrbracket \vdash M : \Pi x : {\color{red}A}.\, \llbracket B \rrbracket \qquad \llbracket \Gamma \rrbracket \vdash N : {\color{red}A}}{\llbracket \Gamma \rrbracket \vdash M\, N : \llbracket B \rrbracket}$$

Proving soundness

What statement should we prove?

- If $[\![\Gamma]\!] \vdash M : A$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ and $A \longrightarrow^* A'$ such that $\Gamma \vdash |M'| : \|A'\|$ in X?

Proving soundness

What statement should we prove?

- ~~If $\llbracket \Gamma \rrbracket \vdash M : A$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ and $A \longrightarrow^* A'$ such that $\Gamma \vdash |M'| : \|A'\|$ in X?~~

$$\frac{\llbracket \Gamma \rrbracket , x : {\color{red}A} \vdash M : B}{\llbracket \Gamma \rrbracket \vdash \lambda x : A.\, M : \Pi x : {\color{red}A}.\, B}$$

Proving soundness

What statement should we prove?

- ~~If $[\![\Gamma]\!] \vdash M : A$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ and $A \longrightarrow^* A'$ such that $\Gamma \vdash |M'| : \|A'\|$ in X?~~

$$\frac{[\![\Gamma]\!], x : \color{red}{A} \vdash M : B}{[\![\Gamma]\!] \vdash \lambda x : A.\, M : \Pi x : \color{red}{A}.\, B}$$

- If $\Gamma \vdash M : A$ in $\lambda\Pi/X$ then $\Gamma \longrightarrow^* \Gamma'$, $M \longrightarrow^* M'$, and $A \longrightarrow^* A'$ such that $\|\Gamma'\| \vdash |M'| : \|A'\|$ in X?

Proving soundness

What statement should we prove?

- ~~If $\llbracket\Gamma\rrbracket \vdash M : A$ in $\lambda\Pi/X$ then $M \longrightarrow^* M'$ and $A \longrightarrow^* A'$ such that $\Gamma \vdash |M'| : \|A'\|$ in X?~~

$$\frac{\llbracket\Gamma\rrbracket, x : A \vdash M : B}{\llbracket\Gamma\rrbracket \vdash \lambda x : A.\, M : \Pi x : A.\, B}$$

- ~~If $\Gamma \vdash M : A$ in $\lambda\Pi/X$ then $\Gamma \longrightarrow^* \Gamma'$, $M \longrightarrow^* M'$, and $A \longrightarrow^* A'$ such that $\|\Gamma'\| \vdash |M'| : \|A'\|$ in X?~~

$$\vdash \lambda\alpha : \mathsf{type}.\, \lambda x : \mathsf{term}\,\alpha.\, x : \Pi\alpha : \mathsf{type}.\, \mathsf{term}\,\beta \rightarrow \mathsf{term}\,\beta$$

Proving soundness

What have we learned?

1. $\lambda\Pi/X$ can type more terms than X.
2. These terms can be used to construct proofs for the translation of X types.
3. The $\lambda\Pi/X$ terms that inhabit the translation of X types can be reduced to the translation of X terms.

Proving soundness

What have we learned?

1. $\lambda\Pi/X$ can type more terms than X.
2. These terms can be used to construct proofs for the translation of X types.
3. The $\lambda\Pi/X$ terms that inhabit the translation of X types can be reduced to the translation of X terms.

Need higher-order reasonning.

Reducibility method

Let Γ' be a context in X. Define the predicate $\Gamma' \vDash M : A$ by induction on A:

Reducibility method

Let Γ' be a context in X. Define the predicate $\Gamma' \vDash M : A$ by induction on A:

- If $A = \text{type}$ then $\Gamma' \vDash M : A$ when $M \longrightarrow^* M'$ such that $\Gamma' \vdash |M'| : \textbf{Type}$.

Reducibility method

Let Γ' be a context in X. Define the predicate $\Gamma' \vDash M : A$ by induction on A:

- If $A = \text{type}$ then $\Gamma' \vDash M : A$ when $M \longrightarrow^* M'$ such that $\Gamma' \vdash |M'| : \textbf{Type}$.

- If $A = \text{term}\, B$ then $\Gamma' \vDash M : A$ when $M \longrightarrow^* M'$ and $B \longrightarrow^* B'$ such that $\Gamma' \vdash |M'| : |B'|$.

Reducibility method

Let Γ' be a context in X. Define the predicate $\Gamma' \models M : A$ by induction on A:

- If $A =$ type then $\Gamma' \models M : A$ when $M \longrightarrow^* M'$ such that $\Gamma' \vdash |M'| : \mathbf{Type}$.

- If $A =$ term B then $\Gamma' \models M : A$ when $M \longrightarrow^* M'$ and $B \longrightarrow^* B'$ such that $\Gamma' \vdash |M'| : |B'|$.

- If $A = \Pi x : B.\, C$ then $\Gamma' \models M : A$ when for for all N such that $\Gamma' \models N : B$, $\Gamma' \models M\, N : \{N/x\}\, C$.

Reducibility method

Let Γ' be a context in X. Define the predicate $\Gamma' \vDash M : A$ by induction on A:

- If $A = \text{type}$ then $\Gamma' \vDash M : A$ when $M \longrightarrow^* M'$ such that $\Gamma' \vdash |M'| : \textbf{Type}$.
- If $A = \text{term}\, B$ then $\Gamma' \vDash M : A$ when $M \longrightarrow^* M'$ and $B \longrightarrow^* B'$ such that $\Gamma' \vdash |M'| : |B'|$.
- If $A = \Pi x : B.\, C$ then $\Gamma' \vDash M : A$ when for for all N such that $\Gamma' \vDash N : B$, $\Gamma' \vDash M\, N : \{N/x\}\, C$.

If σ is a substitution mapping variables to terms:

- $\Gamma' \vDash \sigma : \Gamma$ when $\Gamma' \vDash \sigma\,(x) : \sigma\,(A)$ for all $(x : A) \in \Gamma$

Soundness

Theorem (Assaf 2013)

If $\Gamma \vdash M : A$ *in* $\lambda\Pi/X$ *then for any* X *context* Γ' *and substitution* σ *such that* $\Gamma' \vDash \sigma : \Gamma$, $\Gamma' \vDash \sigma(M) : \sigma(A)$.

Proof.

By induction on the derivation of $\Gamma \vdash M : A$. $\square$

Soundness

Theorem (Assaf 2013)

If $\Gamma \vdash M : A$ *in* $\lambda\Pi/X$ *then for any* X *context* Γ' *and substitution* σ *such that* $\Gamma' \vDash \sigma : \Gamma$, $\Gamma' \vDash \sigma(M) : \sigma(A)$.

Proof.

By induction on the derivation of $\Gamma \vdash M : A$. $\square$

Corollary (Conservativity)

If $[\![\Gamma]\!] \vdash M : [\![A]\!]$ *then* $M \longrightarrow^* M'$ *such that* $\Gamma \vdash |M'| : A$.

Proof.

By taking the identity substitution, $\|\sigma([\![A]\!])\| = \|[\![A]\!]\| = A$. $\square$

Relative normalization

- Avoid complex techniques such as reducibility candidates.
- Works for non-terminating theories!
- For pure type systems, $\lambda\Pi/X$ corresponds to a conservative completion of X.

Summary

- **Strong normalization** $=$ all terms in $\lambda\Pi/X$ are strongly normalizing
  - Proved using termination models
- **Relative normalization** $=$ terms in $\lambda\Pi/X$ can be reduced to terms in X.
  - Proved by using reducibility on a more general statement
- Both approaches show conservativity of $\lambda\Pi/X$

Pure type systems

Specification $S = (\mathcal{S}, \mathcal{A}, \mathcal{R})$

- $\mathcal{S}$ a set of *sorts*
- $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ a set of *axioms*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ a set of *rules*

Syntax

$$
\begin{array}{llcl}
\text{sorts} & s & \in & \mathcal{S} \\
\text{terms} & M, N, A, B & ::= & x \mid s \mid \Pi x : A.\, B \mid \lambda x : A.\, M \mid M\, N \\
\text{contexts} & \Gamma & ::= & \cdot \mid \Gamma, x : A
\end{array}
$$

Typing rules

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{(s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}$$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2 \qquad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x : A.\, B : s_3}$$

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

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : \{N/x\}\, B}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : s \qquad A \equiv B}{\Gamma \vdash M : B}$$

Example

Example

The calculus of constructions (COC) is the PTS defined by the signature:

$$\begin{array}{rcl} \mathcal{S} & = & \textbf{Type}, \textbf{Kind} \\ \mathcal{A} & = & (\textbf{Type}, \textbf{Kind}) \\ \mathcal{R} & = & (\textbf{Type}, \textbf{Type}, \textbf{Type}), (\textbf{Kind}, \textbf{Type}, \textbf{Type}), \\ & & (\textbf{Type}, \textbf{Kind}, \textbf{Kind}), (\textbf{Kind}, \textbf{Kind}, \textbf{Kind}) \end{array}$$

The polymorphic identity function $\mathsf{id} = (\lambda\alpha : \textbf{Type}.\, \lambda x : \alpha.\, x)$ is well-typed in COC:

$$\vdash \mathsf{id} : (\Pi\alpha : \textbf{Type}.\, \alpha \to \alpha)$$

Example

Example

Simple type theory is (STT) is the PTS defined by the signature:

$$
\begin{array}{rcl}
\mathcal{S} & = & \textbf{Prop}, \textbf{Type}, \textbf{Kind} \\
\mathcal{A} & = & (\textbf{Prop}, \textbf{Type})\,(\textbf{Type}, \textbf{Kind}) \\
\mathcal{R} & = & (\textbf{Prop}, \textbf{Prop}, \textbf{Prop}), (\textbf{Type}, \textbf{Prop}, \textbf{Prop}), (\textbf{Type}, \textbf{Type}, \textbf{Type})
\end{array}
$$

Translations

- $\lambda\Pi$-calculus: types may only depend on terms
- PTS: terms and types may depend on types (polymorphism, type operators)

Translations

- $\lambda\Pi$-calculus: types may only depend on terms
- PTS: terms and types may depend on types (polymorphism, type operators)

Tarski style universes (Palmgren 1988): two representations

- $[A]$ as a **term**
- $[\![A]\!]$ as a **type**

Two Translations

As a **type**:

- $\llbracket s \rrbracket = \mathsf{U}_s$, the symbol for the universe of types s
- $\llbracket \Pi x : A.\, B \rrbracket = \Pi x : \llbracket A \rrbracket.\, \llbracket B \rrbracket$

Two Translations

As a **type**:

- $[\![s]\!] = \mathsf{U}_s$, the symbol for the universe of types s
- $[\![\Pi x : A.\, B]\!] = \Pi x : [\![A]\!].\, [\![B]\!]$

As a **term**:

- $[s] = \mathsf{u}_s$, a constant
- $[\Pi x : A.\, B] = \pi\, [A]\, (\lambda x : [\![A]\!].\, [B])$

Decoding function

- Decoding function T

$$[\![A]\!] = \mathsf{T}\,[A]$$

Decoding function

- Decoding function T

$$[\![A]\!] = \mathsf{T}\,[A]$$

- Constraints:

$$\begin{array}{rcl} [\![s]\!] & = & \mathsf{U}_s \\ [\![\Pi x : A.\, B]\!] & = & \Pi x : [\![A]\!].\, [\![B]\!] \end{array}$$

Decoding function

- Decoding function T

$$[\![A]\!] = \mathsf{T}\,[A]$$

- Constraints:

$$\begin{array}{rcl} \mathsf{T}\,[s] & = & \mathsf{U}_s \\ \mathsf{T}\,[(\Pi x : A.\, B)] & = & \Pi x : \mathsf{T}\,A.\, \mathsf{T}\,B \end{array}$$

Decoding function

- Decoding function T

$$[\![A]\!] = \mathsf{T}\,[A]$$

- Constraints:

$$\begin{array}{rcl}
\mathsf{T}\,\mathsf{u}_s & \longrightarrow & \mathsf{U}_s \\
\mathsf{T}\,(\pi\,A\,B) & \longrightarrow & \Pi x : \mathsf{T}\,A.\,\mathsf{T}\,(B\,x)
\end{array}$$

The embedding

Constants

$$
\begin{array}{lcll}
\mathsf{U}_s & : & \textbf{Type} & \forall s \in \mathcal{S} \\
\mathsf{T}_s & : & \mathsf{U}_s \to \textbf{Type} & \forall s \in \mathcal{S} \\
\mathsf{u}_{s_1} & : & \mathsf{U}_{s_2} & \forall (s_1, s_2) \in \mathcal{A} \\
\pi_{s_1,s_2,s_3} & : & \Pi\alpha : \mathsf{U}_{s_1} . \, (\mathsf{T}_{s_1}\, \alpha \to \mathsf{U}_{s_2}) \to \mathsf{U}_{s_3} & \forall (s_1, s_2, s_3) \in \mathcal{R}
\end{array}
$$

Rewrite rules

$$
\begin{array}{lcll}
\mathsf{T}_{s_2}\, \dot{s}_1 & \longrightarrow & s_1 & \forall (s_1, s_2) \in \mathcal{A} \\
\mathsf{T}_{s_3}\, (\pi_{s_1,s_2,s_3}\, \alpha\, \beta) & \longrightarrow & \Pi x : \mathsf{T}_{s_1}\, \alpha . \, \mathsf{T}_{s_2}\, (\beta\, x) & \forall (s_1, s_2, s_3) \in \mathcal{R}
\end{array}
$$

The embedding

$$
\begin{array}{rcl}
[x] & = & x \\
[s] & = & \mathsf{u}_s \\
[\Pi x : A.\, B] & = & \pi_{s_1,s_2,s_3}\, [A]\, [B] \\
[\lambda x : A.\, M] & = & \lambda x : [\![A]\!].\, [M] \\
[M\, N] & = & [M]\, [N] \\
\\
[\![s]\!] & = & \mathsf{U}_s \\
[\![\Pi x : A.\, B]\!] & = & \Pi x : [\![A]\!].\, [\![B]\!] \\
[\![A]\!] & = & \mathsf{T}_{\mathsf{s}}\, [A]
\end{array}
$$

Infinite universe hierarchy

In MLTT, Coq, Agda:

$$\mathsf{U}_0 : \mathsf{U}_1 : \mathsf{U}_2 : \ldots$$

Infinite universe hierarchy

In MLTT, Coq, Agda:

$$\mathsf{U}_0 : \mathsf{U}_1 : \mathsf{U}_2 : \ldots$$

In $\lambda\Pi$-calculus modulo:

$$\begin{array}{lcll}
\mathsf{U}_i & : & \mathbf{Type} & \forall i \in \mathbb{N} \\
\mathsf{T}_i & : & \mathsf{U}_i \to \mathbf{Type} & \forall i \in \mathbb{N} \\
\mathsf{u}_i & : & \mathsf{U}_{i+1} & \forall i \in \mathbb{N} \\
\pi_i & : & \Pi\alpha : \mathsf{U}_i . \left((\mathsf{T}_i\, \alpha \to \mathsf{U}_i)\right) \to \mathsf{U}_i & \forall i \in \mathbb{N}
\end{array}$$

Infinite universe hierarchy

In MLTT, Coq, Agda:

$$\mathsf{U}_0 : \mathsf{U}_1 : \mathsf{U}_2 : \ldots$$

In $\lambda\Pi$-calculus modulo:

$$\begin{array}{lcl}
\mathsf{U} & : & \mathsf{nat} \to \mathbf{Type} \\
\mathsf{T} & : & \Pi i : \mathsf{nat}.\, \mathsf{U}\, i \to \mathbf{Type} \\
\mathsf{u} & : & \Pi i : \mathsf{nat}.\, \mathsf{U}\,(i+1) \\
\pi & : & \Pi i : \mathsf{nat}.\, \Pi \alpha : \mathsf{U}\, i.\, ((\mathsf{T}\, i\, \alpha \to \mathsf{U}\, i)) \to \mathsf{U}\, i
\end{array}$$

Cumulativity

In MLTT, Coq:

$$\frac{\Gamma \vdash A : \mathrm{U}_i}{\Gamma \vdash A : \mathrm{U}_{\mathrm{i+1}}}$$

Cumulativity

In MLTT, Coq:

$$\frac{\Gamma \vdash A : \mathsf{U}_i}{\Gamma \vdash A : \mathsf{U}_{\mathsf{i+1}}}$$

In $\lambda\Pi$-calculus modulo:

$$\uparrow_i \quad : \quad \mathsf{U}_i \to \mathsf{U}_{i+1}$$

$$\mathsf{T}_{\mathsf{i+1}}\,(\uparrow_i A) \quad \longrightarrow \quad \mathsf{T}_i\, A$$

Cumulativity

In MLTT, Coq:

$$\frac{\Gamma \vdash A : \mathsf{U}_i}{\Gamma \vdash A : \mathsf{U}_{\mathsf{i+1}}}$$

In $\lambda\Pi$-calculus modulo:

$$\uparrow_i \quad : \quad \mathsf{U}_i \to \mathsf{U}_{i+1}$$

$$\mathsf{T}_{\mathsf{i+1}}\,(\uparrow_i A) \quad \longrightarrow \quad \mathsf{T}_i\, A$$

Warning: need to reflect equality for completeness

Summary

In $\lambda\Pi$-calculus modulo, can embed:

- functional pure type systems
- infinite type hierarchies
- cumulativity

and much more!

Conclusion

- Universal proof framework based on $\lambda\Pi$-calculus modulo
- Sound and complete embeddings that preserve the reduction semantics
- Automated translation and verification
  - **Coqine:** Coq proofs in Dedukti
  - **Holide:** HOL Light proofs in Dedukti
  - **Zenonide:** Zenon traces in Dedukti
  - **Focalide:** Focalize specifications in Dedukti

Future work

- Universal proof framework?
  - Classical logic without double negation?
  - Linear logic?
  - Intersection types?

Tack!

Bibliography

- Denis Cousineau and Gilles Dowek, *Embedding pure type systems in the lambda-Pi calculus modulo*, TLCA 2007.
- Mathieu Boespflug, Quentin Carbonneaux and Olivier Hermant, *The lambda-Pi calculus modulo as a universal proof language*, PXTP 2012.
- Ronan Saillard, *Towards explicit rewrite rules in the lambda-Pi calculus modulo*, IWIL 2013.
- Ali Assaf, *Conservativity of embeddings in the lambda-Pi calculus modulo*, draft.
- Ali Assaf, *A calculus of constructions with explicit subtyping*, draft.
- Gilles Dowek, *Models and termination of proof-reduction in the lambda-Pi calculus modulo theory*, draft.