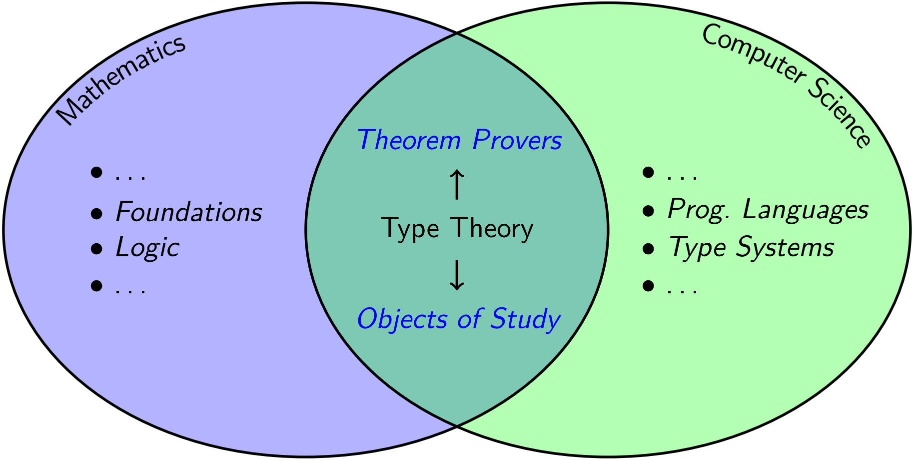
Formal Verification of the Equivalence of System F and the Pure Type System L2

Jonas Kaiser

July 11, 2019

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Where are we? – Some Context



Type Theories as Objects of Study

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. Syntax (things we can express) — *relate*
   - atomic/basic expressions
   - compound expressions
   - possibly grouped into multiple classes
   - may involve variable binding
2. Semantics (assigning meaning to expressions) — *preserve*
   - assertions / judgements
   - inference systems
   - may involve contextual assumptions
3. Theory — *transfer*
   - properties of the Semantics

Two Typed λ-Calculi

SAARLAND UNIVERSITY
COMPUTER SCIENCE

PLC

- proof theory [Girard '72]
  - 2^{nd} ord. intuitionistic logic
- polymorphism [Reynolds '74]
  - prototypical prog. language
- syntactic separation of terms and types $\Rightarrow$ two-sorted
  - two judgements: type formation & typing
- 2 variable scopes, 3 binders

$\lambda 2$

- a pure type system (PTS)
  - [Terlouw '89, Berardi '90]
- corner of λ-cube / study of CC
  - [Barendregt '91]
- only one syntactic class of expressions $\Rightarrow$ single-sorted
  - uniform typing judgement
  - type/term distinction implicit
- 1 variable scope, 2 binders

Goal: Bidirectional Reduction of Type Formation and Typing

J_{P} derivable in PLC $\iff$ J_{λ} derivable in $\lambda 2$

A (Trivial?) Problem

SAARLAND UNIVERSITY
COMPUTER SCIENCE

The system $\lambda 2$ is the polymorphic or second order typed lambda calculus.

[H. Barendregt, λ-cube JFP article, 1991]

To show that the two representations of these systems are in fact the same requires some technical but not difficult work.

[H. Geuvers, Proefschrift, 1993]

vs

We may think of the proof as an iceberg. In the top of it, we find what we usually consider the real proof; underwater, the most of the matter, consisting of all the mathematical preliminaries a reader must know in order to understand what is going on.

[S. Berardi, 1st LF Workshop 1990, Antibes]

Binders should have been a solved problem 10 years ago . . .

[A. Ahmed, paraphrased at FSCD 2016, Porto].

Syntactic Comparison

SAARLAND UNIVERSITY
COMPUTER SCIENCE

PLC a la [Harper '13]:

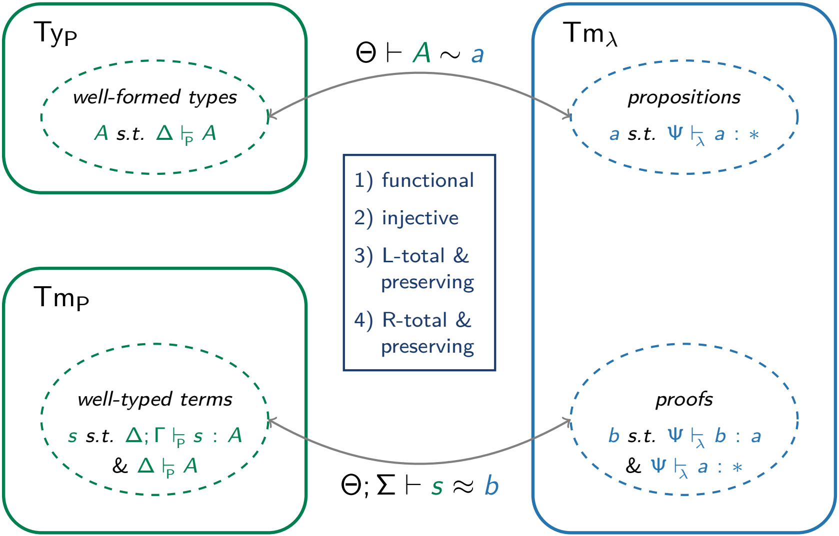
Ty_P	$A, B ::= X \mid A \to B \mid \forall X.A$	$X \in \mathcal{V}_{\mathsf{ty}}$
Tm_P	$s, t ::= x \mid s\ t \mid \lambda x{:}A.s \mid s\ A \mid \Lambda X.s$	$x \in \mathcal{V}_{\mathsf{tm}}$
Type Form.	$\Delta \vdash_{\mathsf{P}} A$	
Typing	$\Delta; \Gamma \vdash_{\mathsf{P}} s : A$	$\vdash_{\mathsf{P}} \Lambda X.\lambda x{:}X.x : \forall X.X \to X$

$$\vdash_{\lambda} \lambda y{:}*.\lambda x{:}y.x : \Pi y{:}*.\Pi x{:}y.y$$

$\lambda 2$:

Tm_λ	$a, b ::= x \mid \mathsf{s} \mid a\ b \mid \lambda x{:}a.b \mid \Pi x{:}a.b$	$x \in \mathcal{V}$
Typing	$\Psi \vdash_{\lambda} a : b$	$\mathsf{s} \in \{*, \Box\}$

Proof Structure



Establishing the Reductions

Let $\sim$ and $\approx$ both be:

1. functional
2. injective
3. left-total and judgement preserving on suitable fragment
4. right-total and judgement preserving on suitable fragment

Theorem (Reduction PLC to $\lambda 2$)

$$\models_{\mathsf{P}} A \iff \exists a.\ \vdash A \sim a \ \wedge\ \models_{\lambda} a : *$$

$$\models_{\mathsf{P}} s : A \iff \exists ba.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \models_{\lambda} b : a \ \wedge\ \models_{\lambda} a : *$$

Theorem (Reduction $\lambda 2$ to PLC)

$$\models_{\lambda} a : * \iff \exists A.\ \vdash A \sim a \ \wedge\ \models_{\mathsf{P}} A$$

$$\models_{\lambda} b : a \ \wedge\ \models_{\lambda} a : * \iff \exists sA.\ \vdash s \approx b \ \wedge\ \vdash A \sim a \ \wedge\ \models_{\mathsf{P}} s : A$$

– Coq –

1st-order de Bruijn Syntax, generalised CMLs

finding the right inductive invariants

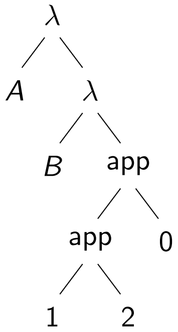
Coq – de Bruijn Syntax

- The α-Equivalence Problem

$$\lambda x : A.\lambda y : B.(x\ z)\ y \equiv_\alpha \lambda w : A.\lambda x : B.(w\ z)\ x$$

- Nameless Representation $\Rightarrow$ canonical

$$\lambda A.\lambda B.(1\ 2)\ 0$$



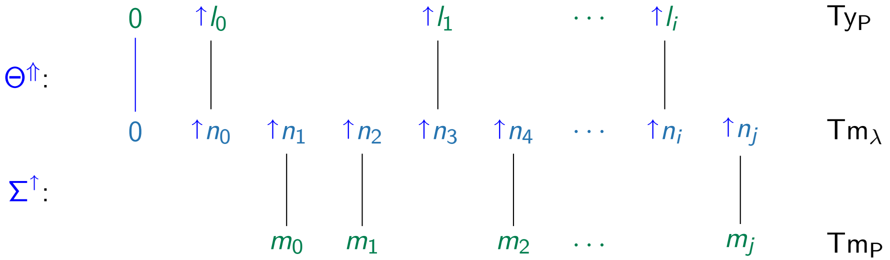
- Contexts: plain lists, positional indexing

$$\Gamma = \ldots, z : C \quad \rightsquigarrow \quad \Gamma = \ldots, C$$

- Instantiation: Parallel Substitutions

$$\sigma : \mathbb{N} \to \mathsf{Tm}_\mathsf{P}$$
$$(\lambda A.s)[\sigma] := \lambda A.s[\Uparrow\sigma]$$
$$\Uparrow\sigma := 0 \cdot (\sigma \circ \uparrow)$$

Coq – Relating Indices



$$\Theta^{\Uparrow} := (0,0) :: \text{map}\,(\uparrow \times \uparrow)\,\Theta$$

$$\Theta^{\uparrow} := \text{map}\,(\text{id} \times \uparrow)\,\Theta$$

$$\frac{\Theta^{\Uparrow}; \Sigma^{\uparrow} \vdash s \approx b}{\Theta; \Sigma \vdash \Lambda.s \approx \lambda *.b}$$

$$\frac{\Theta \vdash A \sim a \qquad \Theta^{\uparrow}; \Sigma^{\Uparrow} \vdash s \approx b}{\Theta; \Sigma \vdash \lambda A.s \approx \lambda a.b}$$

Coq – Context Morphism Lemmas, basic

1. Define invariant:

$$\sigma \Vdash_\lambda \Psi \to \Xi \;:=\; \forall na.\ \Psi \vdash_\mathsf{V} n : a \implies \Xi \vdash_\lambda \sigma n : a[\sigma]$$

2. Prove extension law:

$$\sigma \Vdash_\lambda \Psi \to \Xi \implies \Uparrow\sigma \Vdash_\lambda \Psi, a \to \Xi, a[\sigma] \qquad \textit{– new variable}$$

3. Prove by induction on $\Psi \vdash_\lambda a : b$:

$$\Psi \vdash_\lambda a : b \implies \sigma \Vdash_\lambda \Psi \to \Xi \implies \Xi \vdash_\lambda a[\sigma] : b[\sigma]$$

4. Special case for β-substitutivity:

$$\frac{\Psi, d \vdash_\lambda a : b \qquad \dfrac{\Psi \vdash_\lambda c : d}{c \cdot \mathrm{id} \Vdash_\lambda \Psi, d \to \Psi}}{\Psi \vdash_\lambda a[c \cdot \mathrm{id}] : b[c \cdot \mathrm{id}]}$$

Coq – Context Morphism Lemmas, generalised

1. Define invariant:

$$\Theta \Vdash N \mapsto \Psi \;:=\; \forall n.\; n < N \implies \exists m.\; \Theta \vdash n \simeq m \;\wedge\; \Psi \vDash_{\mathsf{V}} m : *$$

2. Prove extension laws:

$$\Theta \Vdash N \mapsto \Psi \implies \Theta^{\uparrow} \Vdash N \mapsto \Psi, a \qquad \text{– \textit{new term variable}}$$

$$\Theta \Vdash N \mapsto \Psi \implies \Theta^{\Uparrow} \Vdash N + 1 \mapsto \Psi, * \qquad \text{– \textit{new type variable}}$$

3. Prove by induction on $N \vDash_{\mathsf{P}} A$:

$$N \vDash_{\mathsf{P}} A \implies \Theta \Vdash N \mapsto \Psi \implies \exists a.\; \Theta \vdash A \sim a \;\wedge\; \Psi \vDash_{\lambda} a : *$$

4. Special case for $N = 0$:

$$\vDash_{\mathsf{P}} A \implies \exists a.\; \vdash A \sim a \;\wedge\; \vDash_{\lambda} a : *$$

Higher-Order Abstract Syntax [Pfenning/Elliott '88]

- *Key Observation:*
  - host theory has abstraction, application, instantiation $\Rightarrow$ *reuse!*
- Binders are higher-order expression constructors, e.g.

$$\lambda\_.\_ \; : \; \mathsf{Ty_P} \to (\mathsf{Tm_P} \to \mathsf{Tm_P}) \to \mathsf{Tm_P}$$
$$\Pi\_.\_ \; : \; \mathsf{Tm_\lambda} \to (\mathsf{Tm_\lambda} \to \mathsf{Tm_\lambda}) \to \mathsf{Tm_\lambda}$$

  in $\lambda A.s$ the $s : \mathsf{Tm_P} \to \mathsf{Tm_P}$ is a (host-level) abstraction
- β-contraction as (host-level) application: $(\lambda A.s)\; t \rightsquigarrow s\langle t \rangle$
- *Problems:*
  - expression types are not inductive
  - function spaces must be weak/definable/intensional
    $\Rightarrow$ no case analysis, no recursion
  - adequacy of representation

– Abella –

HOAS, nominals, ∇, context managment

finding the right context predicates

Abella – Two-Level Logic

- *Specification Level:* λProlog
  - HOAS definitions of Ty_P, Tm_P & Tm_λ
  - judgements (typing, correspondence) as logic predicates, e.g.

$$\_ : \_ \quad : \quad \mathsf{Tm}_\mathsf{P} \to \mathsf{Ty}_\mathsf{P} \to \mathbf{o}$$
$$\_ \approx \_ \quad : \quad \mathsf{Tm}_\mathsf{P} \to \mathsf{Tm}_\lambda \to \mathbf{o}$$

  - inference rules as extended Horn clauses
  - ambient context tracked implicitly
- *Reasoning Level:* $\mathcal{G}$
  - intuitionistic, predicative fragment of Church's Simple Type Theory
  - inductive predicates
  - nominal constants $\mathsf{n}_1, \mathsf{n}_2, \ldots$ in every type
  - generic quantification ensures freshness:

$$\nabla x.\ \nabla y.\ x \neq y$$

  - implicit specification contexts exposed as lists $L : [\mathbf{o}]$

Abella – Layer Connection

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- *Logical Embedding:*

$$\{\_ \vdash \_\} \ : \ [\mathbf{o}] \to \mathbf{o} \to \mathbf{Prop}$$

$$\{L \vdash J\} \text{ holds in } \mathcal{G} \iff J \text{ has a } \lambda\text{Prolog-derivation from } L$$

- *Mobility of Binders:*

$$\frac{\mathbf{\Pi} x\, y.\ x \sim y \ \Rightarrow\ s\langle x\rangle \approx b\langle y\rangle}{\Lambda.s \approx \lambda{*}.b}$$

$$\{L \vdash \mathbf{\Pi} x\, y.\ x \sim y \ \Rightarrow\ s\langle x\rangle \approx b\langle y\rangle\} \rightsquigarrow \nabla x, y.\{L, x \sim y \vdash s\langle x\rangle \approx b\langle y\rangle\}$$

$$\rightsquigarrow \{L, \mathsf{n}_1 \sim \mathsf{n}_2 \vdash s\langle \mathsf{n}_1\rangle \approx b\langle \mathsf{n}_2\rangle\}$$

Abella – Context Management

- A priori, $L : [\mathbf{o}]$ may contain arbitrary propositions
- Backchaining rule:

$$J \in L \;\Rightarrow\; \{L \vdash J\}$$

- Restrict L to facts about free variables (i.e. *nominal constants*)

$$\begin{aligned}
&\text{Define } C_{\approx} : [\mathbf{o}] \to \mathbf{Prop} \text{ by}\\
&\quad C_{\approx}(\bullet);\\
&\quad \nabla x\, y,\; C_{\approx}(L, x \sim y) \;:=\; C_{\approx}(L);\\
&\quad \nabla x\, y,\; C_{\approx}(L, x \approx y) \;:=\; C_{\approx}(L).
\end{aligned}$$

  - avoids spurious instances of backchaining
  - constrains L to exactly track related variables
  - forces L to be injective & functional

Abella – Connecting Contexts

1. Define a combined context predicate $C_R(- \mid - \mid -)$:

$$\frac{}{C_R(\bullet \mid \bullet \mid \bullet)} \qquad \frac{C_R(L_{\mathsf{P}} \mid L_{\approx} \mid L_{\lambda}) \qquad x, y \text{ fresh for } L_{\mathsf{P}}, L_{\approx}, L_{\lambda}}{C_R(L_{\mathsf{P}}, x\,\mathbf{ty} \mid L_{\approx}, x \sim y \mid L_{\lambda}, y : *)}$$

$$\frac{\begin{array}{ccc}\{L_{\mathsf{P}} \vdash A\,\mathbf{ty}\} & \{L_{\approx} \vdash A \sim a\} & \{L_{\lambda} \vdash a : *\} \\ C_R(L_{\mathsf{P}} \mid L_{\approx} \mid L_{\lambda}) & & x, y \text{ fresh for } L_{\mathsf{P}}, L_{\approx}, L_{\lambda}, A, a\end{array}}{C_R(L_{\mathsf{P}}, x : A \mid L_{\approx}, x \approx y \mid L_{\lambda}, y : a)}$$

2. Prove extraction laws that yield connected assumptions:

$$x\,\mathbf{ty} \in L_{\mathsf{P}} \;\Rightarrow\; C_R(L_{\mathsf{P}} \mid L_{\approx} \mid L_{\lambda}) \;\Rightarrow\; \ldots$$

3. Prove by induction on $\{L_{\mathsf{P}} \vdash A\,\mathbf{ty}\}$:

$$\{L_{\mathsf{P}} \vdash A\,\mathbf{ty}\} \;\Rightarrow\; \forall L_{\approx}\, L_{\lambda}.\; C_R(L_{\mathsf{P}} \mid L_{\approx} \mid L_{\lambda}) \;\Rightarrow\; \exists a.\; \{L_{\approx} \vdash A \sim a\} \;\wedge\; \{L_{\lambda} \vdash a : *\}$$

beluga

– Beluga –

HOAS, dependently typed programming with 1st-class contexts

finding the right context schemas

Beluga – Two-Level Logic

SAARLAND UNIVERSITY COMPUTER SCIENCE

- *Specification Level:* LF
  - HOAS LF types Ty_P, Tm_P & Tm_λ, similar to Abella
  - judgements as LF type families
- *Reasoning Level:* Contextual Modal Type Theory (CMTT)
  - 1^{st}-class contexts: γ : S
  - LF terms/types/derivations K as *contextual objects*: $[\gamma \vdash K]$
  - Proofs: total programs using pattern matching & HO unification

Beluga – Contextual Objects

SAARLAND UNIVERSITY COMPUTER SCIENCE

- Open LF entities K paired with a context γ that gives them meaning

$$[\gamma \vdash K]$$

- Object variables cannot escape into reasoning context
- In fact: no concept of *free object variable*
  - Coq $\quad 0 \models_{\mathsf{P}} 0 \to 0 \implies \bot \quad$ provable
  - Abella $\quad \{\bullet \vdash \mathsf{n}_0 \to \mathsf{n}_0\ \mathbf{ty}\} \implies \bot \quad$ provable
  - Beluga $\quad [\bullet \vdash x \to x] \quad$ ill-formed since $x \notin \bullet$
- No PLC type formation judgement $A\,\mathbf{ty}$
  - affects representation and proofs
- Supports inductive reasoning on contextual objects

Beluga – Context Schemas, γ : S

- Rich contexts: heterogeneous dependent lists of dependent records
- Schemas S constraint shape

$$\overline{S_\lambda} \;:=\; [x : \mathrm{Tm}_\lambda, h : x : *] + [x : \mathrm{Tm}_\lambda, h : x : a, j : a : *]$$

- Schema ascription checked as part of type checking
- Canonical vs non-canonical, derivability

$$S_\lambda \;:=\; [x : \mathrm{Tm}_\lambda, h : x : a, j : a : b, k : b \in \{*, \square\}]$$

Non-canonical schemas turn contexts into conduits which carry semantic information from binding sites to usage sites.

Beluga – Complex Schemas

1. Define schema $\overline{\mathsf{S}_{\sim,\mathsf{P}}}$ with specific typing information:

$$\overline{\mathsf{S}_{\sim,\mathsf{P}}} \;:=\; [x : \mathsf{Ty}_\mathsf{P}, y : \mathsf{Tm}_\lambda, x \sim y, y : *] + [y : \mathsf{Tm}_\lambda, y : a]$$

2. Implement a function $p_{\sim,\mathsf{P}}$ by recursion on $A : [\gamma \vdash \mathsf{Ty}_\mathsf{P}]$

$$p_{\sim,\mathsf{P}} \;:\; \forall\gamma : \overline{\mathsf{S}_{\sim,\mathsf{P}}}.\ \forall A : [\gamma \vdash \mathsf{Ty}_\mathsf{P}].\ [\gamma \vdash \exists a.A \sim a \wedge a : *]$$

Conclusion – Main Contribution

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. A formal equivalence proof for two System F variants
2. A continuation of benchmarking efforts
   - POPLMARK [Aydemir et al. '05]
     metatheory focused on single type theory
   - ORBI [Felty et al. '15]
     multiple systems, basic contextual reasoning
   - Here:
     cross-theory
     multiple systems
     complex contextual reasoning

Conclusion – Technical Remarks

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- Formalisation effort:

	mode	**approx. LOC**
Coq	tactics	2140
Abella	tactics	450
Beluga	proof terms	340

- Regarding syntax with binding: *there is no silver bullet!*
- However, certain techniques go well together:
  - de Bruijn: parallel substitutions, generalised CMLs
  - HOAS: careful context control (predicates / schemas)
  - syntactic inductive correspondence relations

Conclusion – Future Directions

SAARLAND UNIVERSITY
COMPUTER SCIENCE

1. *Co-reducibility*
   - likely requires $\sim$, $\approx$ as bisimulations
   - restriction to well-typed fragments essential
2. *Include other variants of System F*
   - PTS with weakening built in and/or well-scoped syntax, see [Adams '04]
   - PLC with different levels of type ascription (Church vs Curry)
3. *Consider other representations / frameworks / proof assistants*
   - LN [Aydemir et al. '08], nominal [Pitts '03]
   - Autosubst 2, HYBRID [Capretta/Felty '06] (both Isabelle and Coq)
   - Twelf, Lean, Agda, ...
4. *Consider other type theories*
   - F_ω, $\mathsf{F}_{<:}$, ...

Thank you for your attention.

http://www.ps.uni-saarland.de/static/kaiser-diss/index.php