Formal Proofs with Binders

Arthur Charguéraud

Work completed at the University of Pennsylvania
with Benjamin Pierce and Stephanie Weirich

MSR-INRIA Orsay, 2006-10-18

Our Goal and Our Approach

Formalize

Coq

Simple

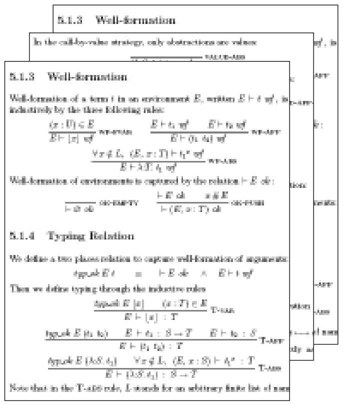
Intuitive

Transparent

Take paper

Translate

Arrange



Formalizations Involving Binders

- Certification of a Theorem Prover

  Coq in Coq

  Bruno Barras and Benjamin Werner, 1996

- Soundness for a Type System

  SML in Twelf

  Karl Crary, Daniel Lee et Robert Harper, 2006

- Certification of a Compiler

  C-light in Coq

  Xavier Leroy, Sandrine Blazy, Zaynah Dargaye, 2006

The POPLMark Challenge

Mechanized Metatheory for the Masses:
The POPLMark Challenge

By B. Aydemier, A. Bohannon, M. Fairbairn, N. Foster, B. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, S. Zdancewic – March 2005.

- To formalize results from their POPL papers
- A set of benchmarks: Metatheory of System-$F_{<:}$
- Basis for comparing technologies and techniques

Previous Work

First-Order and Higher-Order

First-Order Abstract Syntax	*Higher-Order Abstract Syntax*
e.g. representation with names:	*e.g. as done in Twelf:*

```
term : Set :=
| Var : name -> term
| App : term -> term -> term
| Abs : name -> term -> term
```

App (Abs x t) u $\xrightarrow{\beta}$ [x->u]t

```
term : type

app : term -> term -> term
abs : (term -> term) -> term
```

app (abs t) u $\xrightarrow{\beta}$ t u

We focus on first-order representations.

Previous Work in First-Order

Yr	Author	Formalization	Prover	Encoding
85	Natarajan Shankar	Church-Rosser in λ	Boyer-Moore	de-Bruijn
93	Thorsten Altenkirch	SN of System-F	LEGO	de-Bruijn
93	J.McKinna, R.Pollack	Pure Type Systems	LEGO	names
94	Gérard Huet	Residual Theory in λ	Coq	de-Bruijn
95	Ole Rasmussen	Church-Rosser in λ	Isabelle/ZF	de-Bruijn
96	Tobias Nipkow	Church-Rosser in λ	Isabelle/HOL	de-Bruijn
96	B.Barras, B.Werner	Kernel of Coq	Coq	de-Bruijn
97	J.McKinna, R.Pollack	λ-calculus & types	LEGO	names
01	Vestergaard, Brotherston	Church-Rosser in λ	Isabelle/HOL	names
01	J.Ford, I.Mason	Church-Rosser in λ	PVS	names
01	Peter Homeier	Church-Rosser in λ	HOL	names

Submissions to POPLMark

Author	Part 1	Part 2	Prover	Encoding
Stephan Berghofer	Y	Y	Isabelle	de-Bruijn indices
Ashley, Crary, Harper	Y	Y	Twelf	higher-order
Jérome Vouillon	Y	Y	Coq	de-Bruijn indices
Hongwei Xi		Y	ATS/LF	higher-order
Jevgenijs Sallinens	Y		Coq	de-Bruijn indices
Xavier Leroy	Y		Coq	locally nameless
Aaron Stump	Y		Coq	names / levels
Christian Urban	Y		Isabelle	nominal
Hirschowitz, Maggesi	Y		Coq	de-Bruijn (nested)
Adam Chlipala	Y		Coq	locally nameless
Arthur Charguéraud	Y		Coq	locally nameless

Contribution

The POPLMark Challenge

Preservation and Progress for System-$F_{<:}$

POPLMark Part 1:

Properties of subtyping

POPLMark Part 2:

Rest of the formalization

[extended]

[simplified]

Our Challenge B

Properties of subtyping, including preservation by type substitution

Our Challenge A

Preservation and progress for simply typed λ-calculus

A) Simply Typed λ-calculus

$$T \quad := \quad A \quad | \quad T_1 \rightarrow T_2$$

$$t \quad := \quad x \quad | \quad (t_1 \ t_2) \quad | \quad \lambda x{:}T.\ t_1$$

$$\frac{(x : T) \in E}{E \vdash x : T} \ \textsc{T-Var} \qquad \frac{E \vdash t_1 : S \rightarrow T \qquad E \vdash t_2 : S}{E \vdash (t_1 \ t_2) : T} \ \textsc{T-App}$$

$$\frac{x \# E \qquad E, x : S \vdash t_1 : T}{E \vdash (\lambda x{:}S.\ t_1) : S \rightarrow T} \ \textsc{T-Abs}$$

Preservation: $t \longmapsto t' \quad \wedge \quad E \vdash t : T \quad \Rightarrow \quad E \vdash t' : T$

Progress: $\varnothing \vdash t : T \quad \Rightarrow \quad [\, t \text{ is a value} \quad \vee \quad \exists t',\ t \longmapsto t' \,]$

B) Subtyping in System-$F_{<:}$

$$T \quad := \quad \mathrm{Top} \quad | \quad X \quad | \quad T_1 \to T_2 \quad | \quad \forall X{<:}T_1.\,T_2$$

$$\frac{}{E \vdash S <: \mathrm{Top}}\ \textsc{SA-Top} \qquad \frac{E \vdash T_1 <: S_1 \qquad E \vdash S_2 <: T_2}{E \vdash (S_1 \to S_2) <: (T_1 \to T_2)}\ \textsc{SA-Arrow}$$

$$\frac{}{E \vdash X <: X}\ \textsc{SA-Refl-TVar} \qquad \frac{(X{<:}U) \in E \qquad E \vdash U <: T}{E \vdash X <: T}\ \textsc{SA-Trans-TVar}$$

$$\frac{E \vdash T_1 <: S_1 \qquad X \# E \qquad (E,\, X{<:}T_1) \vdash S_2 <: T_2}{E \vdash (\forall X{<:}S_1.\,S_2) <: (\forall X{<:}T_1.\,T_2)}\ \textsc{SA-All}$$

Reflexivity: $E \vdash T <: T$

Transtitivity: $E \vdash S <: Q \quad \wedge \quad E \vdash Q <: T \quad \Rightarrow \quad E \vdash S <: T$

Preservation by type substitution:

$$E,\, Z{<:}Q, F \vdash S <: T \quad \wedge \quad E \vdash P <: Q$$
$$\Rightarrow \quad E, [Z \to P]\, F \vdash [Z \to P]\, S <: [Z \to P]\, T$$

Contribution

We answer the following questions:

– What are the big design issues?

– What are the possible solutions?

– What is the best solution in each case?

Selecting a set of good design choices, we formalized in Coq the two subchallenges.

The result is short, simple and intuitive.

Plan

1) Represention of Bindings

2) Other Design Choices

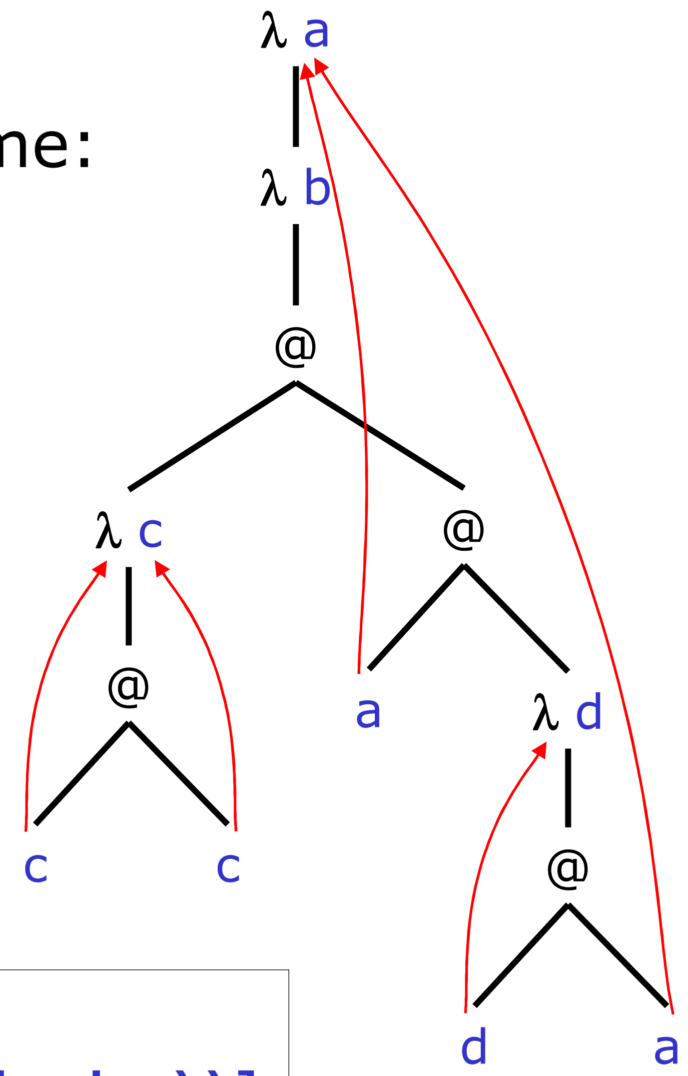
3) Formalization in Coq

4) Comments on Using Coq

1) Represention of Bindings

λ-term with names

Each abstraction introduces a name:



Pros:

– as on paper

Cons:

– quotient by α

– α-conversion

```
λa.λb.
[(λc. c c) (a (λd. d a))]
```

Handling α-equivalence (1)

– With quotient by alpha-equivalence

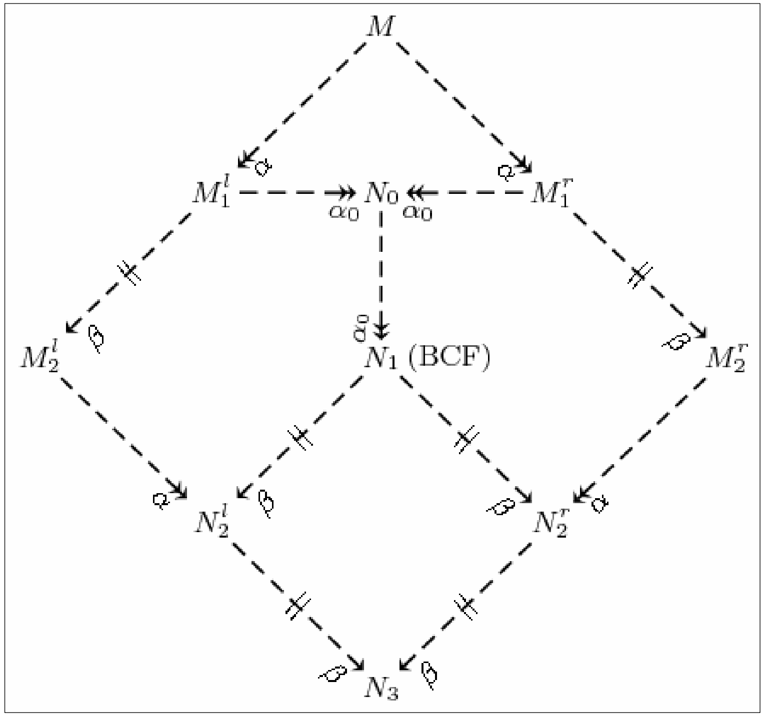
[Homeier, confluence in 359 lemmas, HOL, 2001]

[Ford & Mason, confluence in 236 lemmas, PVS, 2001]

$$u_1 \equiv_\alpha u_2 \quad \wedge \quad t_1 \equiv_\alpha t_2 \quad \Rightarrow \quad [x \to u_1]\, t_1 \equiv_\alpha [x \to u_2]\, t_2$$

– Without the quotient

[Verstergaard & Brotherston,
confluence in 200+ lemmas over
4000 lines of Isabelle/HOL, 2001]



Handling α-equivalence (2)

– Without alpha-conversion, nor the quotient

[McKinna & Pollack, lambda-calculus + type theory, LEGO, 1993-97]

$$A \downarrow M \;\Rightarrow\; (A \downarrow N \;\Leftrightarrow\; M \stackrel{\alpha}{\sim} N).$$

– The nominal approach

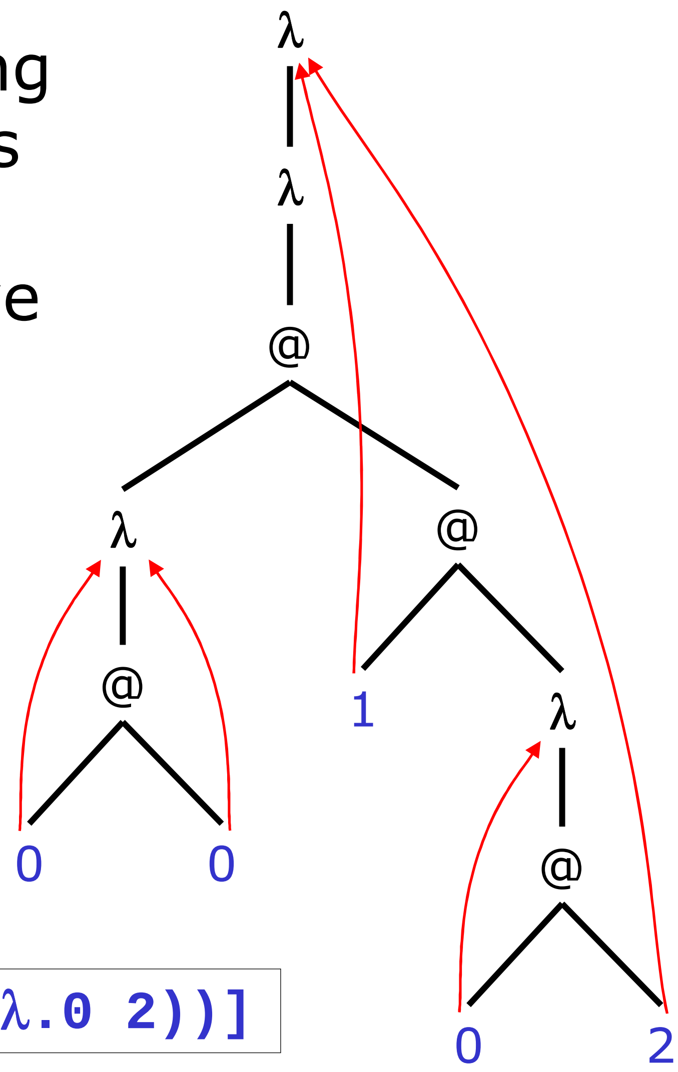
[Urban's "nominal package", Isabelle/HOL, still under development, and also work by Norrish, in HOL, 2004.]

$$
\begin{array}{l}
\forall con\ var\ app\ lam. \\
\quad \exists hom. \\
\qquad (\forall\, k.\ \ hom(\mathrm{CON}\, k) = con(k)) \wedge \\
\qquad (\forall\, s.\ \ hom(\mathrm{VAR}\, s) = var(s)) \wedge \\
\qquad (\forall\, t\ u.\ \ hom(\mathrm{APP}\ t\ u) = app\ (hom\ t)\ (hom\ u)\ t\ u) \wedge \\
\qquad (\forall\, v\ t.\ \ hom(\mathrm{LAM}\ v\ t) = \\
\qquad\qquad\qquad lam\ (\lambda y.\, hom(t[v \mapsto \mathrm{VAR}(y)]))\ (\lambda y.\, t[v \mapsto \mathrm{VAR}(y)]))
\end{array}
$$

$$v \notin \mathrm{FV}(t) \quad \Rightarrow \quad \mathrm{LAM}\ u\ t = \mathrm{LAM}\ v\ (\mathrm{swap}\ u\ v\ t)$$

λ-term with de-Bruijn indices

A variable bearing an index k points towards the k^{ith} abstraction above that variable:



`λ.λ.[(λ.0 0) (1 (λ.0 2))]`

Pros:

– α-equivalence is identity

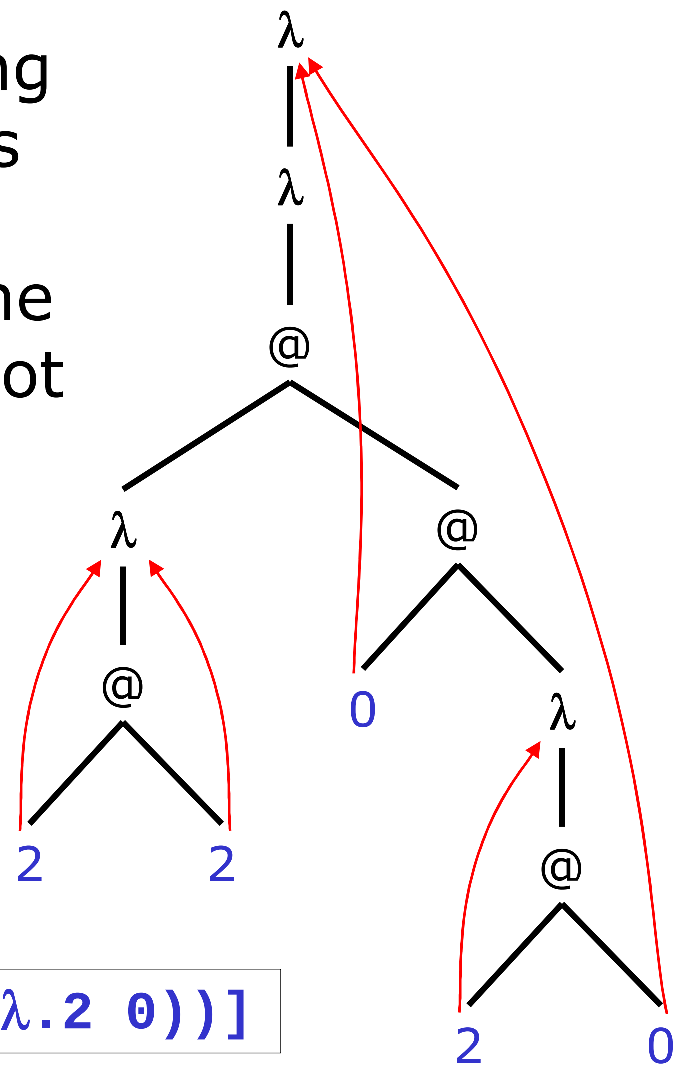
Cons:

– shifting free variables in the argument

– unshifting free variables in the body

λ-term with de-Bruijn levels

A variable bearing an index *k* points towards the k^{ith} abstraction on the path from the root to that variable:



`λ.λ.[(λ.2 2) (0 (λ.2 0))]`

Pros:

– α-equivalence is identity

Cons:

– shifting bound variables in the argument

– unshifting bound variables in the body

shift and *subst*

Properties of shifting and substitution.

Not very difficult, but fiddly.

$$
\begin{array}{l}
i \le j \Longrightarrow j \le i + m \Longrightarrow \uparrow_\tau n\ j\ (\uparrow_\tau m\ i\ T) = \uparrow_\tau (m + n)\ i\ T \\
i + m \le j \Longrightarrow \uparrow_\tau n\ j\ (\uparrow_\tau m\ i\ T) = \uparrow_\tau m\ i\ (\uparrow_\tau n\ (j - m)\ T) \\
k \le k' \Longrightarrow k' < k + n \Longrightarrow \uparrow_\tau n\ k\ T[k' \mapsto_\tau U]_\tau = \uparrow_\tau (n - 1)\ k\ T \\
k \le k' \Longrightarrow \uparrow_\tau n\ k\ (T[k' \mapsto_\tau U]_\tau) = \uparrow_\tau n\ k\ T[k' + n \mapsto_\tau U]_\tau \\
k' < k \Longrightarrow \uparrow_\tau n\ k\ (T[k' \mapsto_\tau U]_\tau) = \uparrow_\tau n\ (k + 1)\ T[k' \mapsto_\tau \uparrow_\tau n\ (k - k')\ U]_\tau \\
k \le k' \Longrightarrow \uparrow_\tau n\ k'\ (T[k \mapsto_\tau Top]_\tau) = \uparrow_\tau n\ (Suc\ k')\ T[k \mapsto_\tau Top]_\tau \\
k \le k' \Longrightarrow k' \le k + n \Longrightarrow \uparrow n'\ k'\ (\uparrow n\ k\ t) = \uparrow (n + n')\ k\ t \\
i \le j \Longrightarrow T[Suc\ j \mapsto_\tau V]_\tau[i \mapsto_\tau U[j - i \mapsto_\tau V]_\tau]_\tau = T[i \mapsto_\tau U]_\tau[j \mapsto_\tau V]_\tau
\end{array}
$$

source: Berghofer 2005

Weakening in System-$F_{<:}$

Statements are polluted by shifting.

$$
\Gamma \vdash t : T \Longrightarrow \Delta\ @\ \Gamma \vdash_{wf} \Longrightarrow \Delta\ @\ \Gamma \vdash \uparrow \|\Delta\|\ 0\ t : \uparrow_\tau \|\Delta\|\ 0\ T
$$

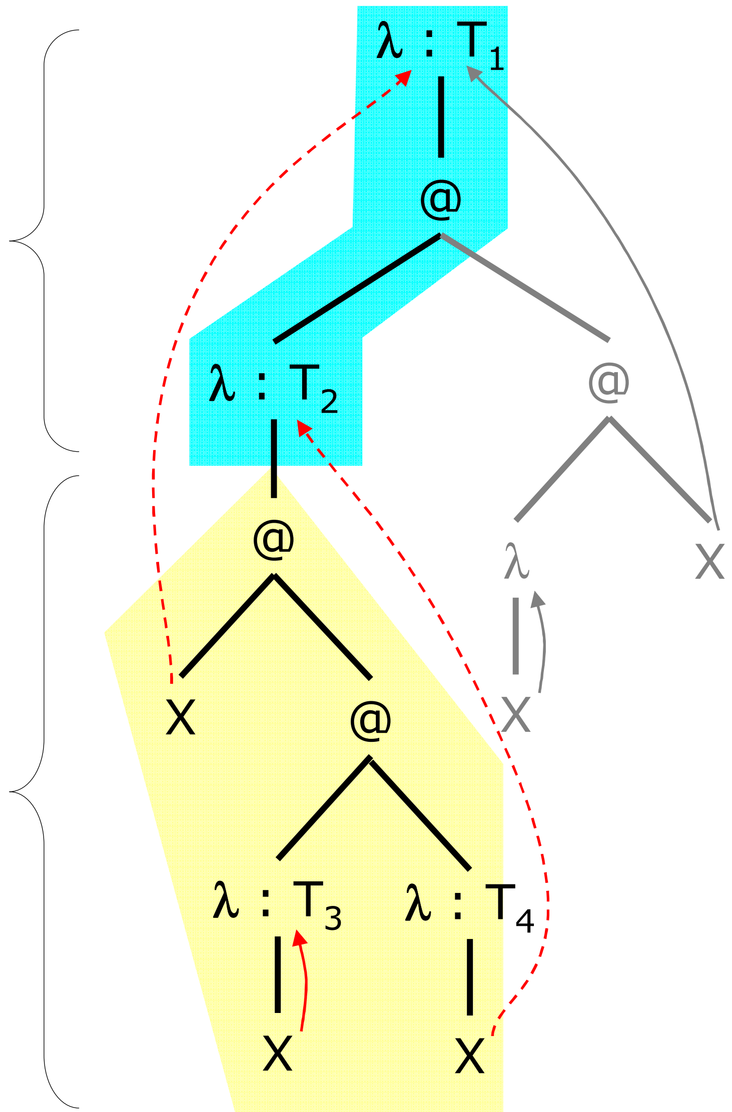
Bound and Free Variables

environment *E*

typing judgment

$$\boxed{E \mid\!- \; t : T}$$

term *t*



Distinguishing Bound and Free

For example the locally nameless representation, where
– bound variables represented as de-Bruijn indices,
– free variables represented using names.

Substitution a term *u* for a bound variable *k* in a term *t*:

{k −> u}t

$\{k \to u\}\,\lceil i \rceil$	$\equiv$	if $i = k$ then u else $\lceil i \rceil$
$\{k \to u\}\,\lfloor x \rfloor$	$\equiv$	$\lfloor x \rfloor$
$\{k \to u\}\,(t_1\ t_2)$	$\equiv$	$((\{k \to u\}\,t_1)\ (\{k \to u\}\,t_2))$
$\{k \to u\}\,(\lambda{:}T.\,t_1)$	$\equiv$	$\lambda{:}T.\,(\{(k+1) \to u\}\,t_1)$

Substitution a term *u* for a free variable *z* in a term *t*:

[k −> u]t

$[z \to u]\,\lceil i \rceil$	$\equiv$	$\lceil i \rceil$
$[z \to u]\,\lfloor x \rfloor$	$\equiv$	if $x = z$ then u else $\lfloor x \rfloor$
$[z \to u]\,(t_1\ t_2)$	$\equiv$	$(([z \to u]\,t_1)\ ([z \to u]\,t_2))$
$[z \to u]\,(\lambda{:}T.\,t_1)$	$\equiv$	$\lambda{:}T.\,([z \to u]\,t_1)$

Full β-reduction in Locally Nameless

$$\frac{}{((\lambda{:}S.\ t_1)\ \ t_2) \longmapsto {t_1}^{t_2}}\ \text{RED-BETA}$$

$$\frac{t_1 \longmapsto t_1'}{(t_1\ \ t_2) \longmapsto (t_1'\ \ t_2)}\ \text{RED-APP-1} \qquad \frac{t_2 \longmapsto t_2'}{(t_1\ \ t_2) \longmapsto (t_1\ \ t_2')}\ \text{RED-APP-2}$$

$$\frac{{t_1}^{x} \longmapsto {t_1'}^{x}}{(\lambda{:}S.\ t_1) \longmapsto (\lambda{:}S.\ t_1')}\ \text{RED-ABS} \qquad x \# t_1,\ x \# t_1'$$

Where: $\boxed{{t_1}^{t_2} \ \equiv\ \{0 \to t_2\}\, t_1}$ *and* $\boxed{{t_1}^{x} \ \equiv\ \{0 \to \lfloor x \rfloor\}\, t_1}$

Properties of Substitution

– Introduction of a name to decompose a beta-reduction step:

$$\boxed{{t_1}^{t_2} = [x \to t_2]\,({t_1}^{x}) \quad \text{when} \quad x \# t_1}$$

– Propagation of a substitution on name through a reduction:

$$\boxed{[z \to u]\,({t_1}^{t_2}) = ([z \to u]\,t_1)^{([z \to u]\,t_2)} \quad \text{when} \quad z \# t_2 \;\wedge\; u \text{ well-formed}}$$

and its weaker form:

$$\boxed{[z \to u]\,(t^{x}) = ([z \to u]\,t)^{x} \quad \text{when} \quad x \neq z \;\wedge\; u \text{ well-formed}}$$

Summary of Representations

	bound variables	free variables
names	~~requires reasoning on α-equivalence~~	ok
de Bruijn indices	ok	~~shifting is necessary~~
de Bruijn levels	~~shifting is necessary~~	~~shifting is necessary~~

Winner is: Locally Nameless

2) Other Design Choices

Environments as Lists or Sets?

Weakening Preserves Typing

Paper: $E \vdash S <: T \quad \Rightarrow \quad E, F \vdash S <: T$

Formal: $E \vdash S <: T \quad \wedge \quad E \subset F \quad \Rightarrow \quad F \vdash S <: T$

where: $E \subset F \quad \equiv \quad \forall\, x\, T, \quad (x : T) \in E \quad \Rightarrow \quad (x : T) \in F$

Substitution Preserves Typing

Paper: $E, z : U, F \vdash t \ : \ T \quad \wedge \quad E \vdash u \ : \ U \quad \Rightarrow \quad E, F \vdash [z \to u]\, t \ : \ T$

Formal: $E \vdash t \ : \ T \quad \wedge \quad F \vdash u \ : \ U \quad \wedge$
$(z : U) \in E \quad \wedge \quad E \setminus z \ \subset F \quad \Rightarrow \quad F \vdash [z \to u]\, t \ : \ T$

where: $E \setminus z \ \subset F \quad \equiv \quad \forall\, x\, T, \quad (x : T) \in E \quad \wedge \quad x \neq z \quad \Rightarrow \quad (x : T) \in F$

Names pushed in the Environment

$$\frac{Quantify(x) \qquad (E, x : S) \vdash (t_1{}^x) \;:\; T}{E \vdash (\lambda{:}S.\, t_1) \;:\; S \rightarrow T}\ \text{T-ABS}$$

Quantify(x) =	∃ x # E	∀ x # E	∀ x ∉ L
Weakening	**swapping required**	**ok**	**ok**
Substitution	**ok**	**swapping required**	**ok**
Transitivity	**swapping required**	**ok**	**ok**

Weakening $E \vdash t \;:\; T \quad \Rightarrow \quad E, F \vdash t \;:\; T$

Substitution $E, z : U, F \vdash t \;:\; T \quad \wedge \quad E \vdash u \;:\; U \quad \Rightarrow \quad E, F \vdash [z \rightarrow u]t \;:\; T$

Transitivity $E \vdash S <: Q \quad \wedge \quad E \vdash Q <: T \quad \Rightarrow \quad E \vdash S <: T$

Well-formedness of Terms

– With recursive functions

- all the free variables of *t* belong to the domain *D*
- indices are smaller than the number of lambda above them

FV(t) ⊂ D ∧ wf_indices(0,t)

– With an inductive relation

t : term ∧ D |- t wf

$$\frac{x \in D}{D \vdash \lfloor x \rfloor \; wf} \text{ WF-FVAR} \qquad \frac{D \vdash t_1 \; wf \qquad D \vdash t_2 \; wf}{D \vdash (t_1 \;\; t_2) \; wf} \text{ WF-APP}$$

$$\frac{\forall\, x \notin L, \;\; (D \cup \{x\}) \vdash ({t_1}^{x}) \; wf}{D \vdash (\lambda{:}T.\, t_1) \; wf} \text{ WF-ABS}$$

– With dependent types

t : term D

3) Formalization in Coq

Example: Weakening on Subtyping

Informal:

$$E \vdash S <: T \quad \Rightarrow \quad E, F \vdash S <: T$$

Proof by induction on the subtyping derivation,
using the reordering lemma for case SA-all.
α-equivalence, Barendregt's convention, well-formedness.

Formalizable:

$$E \vdash S <: T \quad \wedge \quad E \subset F \quad \wedge \quad \vdash F\ ok \quad \Rightarrow \quad F \vdash S <: T$$

Proof by induction on the subtyping derivation, easy
but in case SA-all: pick a variable X outside of dom(F)
and then use lemma "extends_push".

Formal:

```
Lemma sub_extension : forall E S T, E |- S <: T
    -> forall F, E inc F -> ok F -> F |- S <: T.
intros E S T H. induction H; intros; auto**.
apply_SA_all X (L ++ dom F). use extends_push.
```

Example: Transitivity of Subtyping

```
Theorem subtyping_transitivity : forall E S Q T,
  E |- S <: Q  ->  E |- Q <: T  ->  E |- S <: T.
intros. apply* (@sub_transitivity E Q). Qed.

Lemma sub_transitivity :
 forall E Q (WQ : E wf Q), sub_trans_prop WQ.
intros. unfold sub_trans_prop. generalize_equality Q Q'.
induction WQ; intros Q' EQ F S T EincF SsubQ QsubT;
induction SsubQ; try discriminate; try injection EQ; intros;
inversion QsubT; subst; intuition eauto.
  (* Case SA-arrow *)
apply SA_arrow. auto. apply* IHWQ1. apply* IHWQ2.
  (* Case SA-all *)
apply_SA_all X ((dom E0) ++ L ++ L0 ++ L1). apply* H0.
asserts* WQ1 (E0 wf T1). apply* (sub_narrowing (WQ := WQ1)).
Qed.
```

Statistics on our Coq Scripts

	Simply typed λ-calculus	Properties of subtyping
Definitions	8	9
Axioms	0	0
Lemmas	26	34
Theorems	2	5
Lines of proofs	63	104
Number of tactics	202	279
Non-dummy tactics in the main proofs:	36	67

Complexity of Solutions in Coq

Number of tactics invoked, not counting calls to proof-search, on part 1A of the POPLMark Challenge (properties of subtyping).

Author	Tactics	Representation
Jérome Vouillon	431	de-Bruijn indices
Aaron Stump	1147	names / levels
Xavier Leroy	630	locally nameless
Hirschowitz, Maggesi	1615	de-Bruijn (nested)
Adam Chlipala	342	locally nameless
Arthur Charguéraud	233	locally nameless

4) Comments on Using Coq

1) Automation really is Cool

Automation...

– shortens proof script,

– saves a lot of time,

– let focus on difficulties,

– makes proofs more resistant to changes,

– makes people believe they are talking to something clever.

deserves...

– a complete tutorial so that more people can truely benefit from it,

– further development so as to make *auto* solve more goals,

– to be made intuitive even to those who don't know how it works.

Clever Automation

$$E \subset F \quad \equiv \quad \forall\, x\, T, \quad (x : T) \in E \;\Rightarrow\; (x : T) \in F$$

Lemma extends_push: $E \subset F \quad \Rightarrow \quad (E, x : T) \subset (F, x : T)$

```
Definition extends E F := forall x U,
  (E has x ~: U)  ->  (F has x ~: U).

Notation "E 'inc' F" := (extends E F).

Lemma extends_push : forall E F x T,
  E inc F -> (E & x ~: T) inc (F & x ~: T).
```

~~unfold extends. intros. inversion* H0. Qed.~~

```
be_clever. Qed.
```

Intuitive Automation

~~Lemma my_lemma := hypB -> hypA -> conclusion.~~

```
Lemma my_lemma := hypA -> hypB -> conclusion.
```

```
Theorem my_result : ...
...
eapply my_lemma; eauto.
...
Qed.
```

breaks!

Memory for Automation

Interest of saving extra information:

– save waiting time during development,

– help recovering from broken proofs.

How? After a proof search is called:

– if successful, store the main steps for next time,

– if failed, remember not to try it again each time.

2) Structuring Proofs

Problem:

– Non mathematical proofs need to get updated.

– Current layout of scripts is not suited for that.

Solution?

– Have a tree presentation, relating branches to constructors and subgoals to hypotheses.

– Give IDs to all variables introduced and relate them to the hypothesis they come from.

Conclusions

Locally Nameless is not New!

1972: N.G. de-Bruijn

A Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem.

1989: G. Huet

The Constructive Engine

1994: A. Gordon

A Mechanisation of Name-carrying Syntax up to Alpha-conversion

2005-2006: X. Leroy, A. Chlipala, A. Charguéraud,

Solutions to the POPLMark Challenge

Locally Nameless is Good!

– All the work from McKinna and Pollack could be rewritten and simplified using locally nameless.

– Locally nameless has been used to implement type checkers (Coq, LEGO, HOL4, Epigram).

– Locally nameless enables us to make short and simple proofs, faithful to informal practice.

Future Work

- Complete the solution to POPLMark Challenge.
- Formalize some λ-calculus (e.g. confluence).
- Address more complex type systems (CoC).
- Support more advanced binding constructions.

Thanks !