A Comparison Between Concrete Representation for Bindings

Arthur Charguéraud

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

Workshop on Mechanizing Metatheory   Portland, 2006-09-21

Our Goal and Our Approach

Formalize

Coq

Take paper

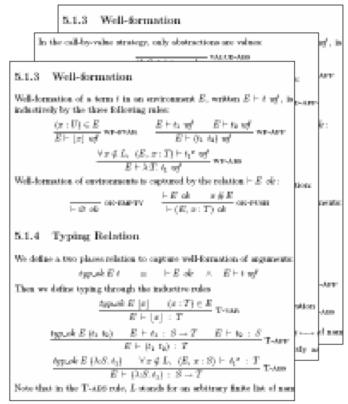
Simple

Intuitive

Transparent

Translate

Arrange


5.1.3 Well-formation

5.1.3 Well-formation

5.1.4 Typing Relation

First-Order and Higher-Order

We focus on first-order representations.

First-Order Abstract Syntax

e.g. representation with names:

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

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

Higher-Order Abstract Syntax

e.g. as done in Twelf:

```
term : type

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

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

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

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) Representation of Bindings

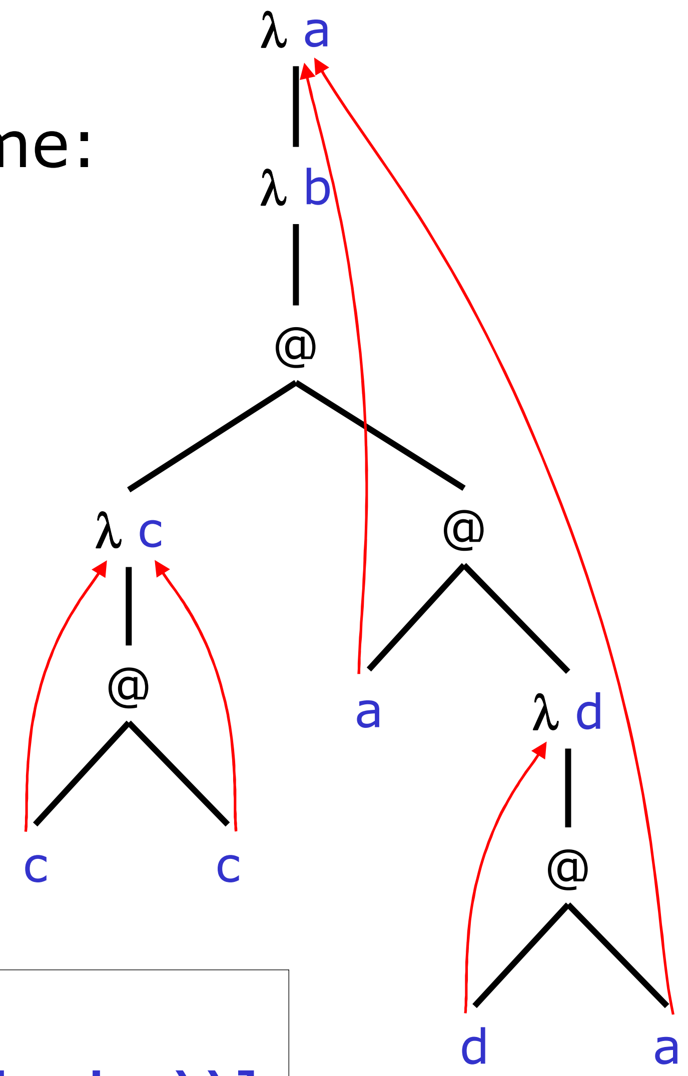
2) Bindings in Environments

3) Formalization in Coq

1) Represention of Bindings

λ-term with names

Each abstraction introduces a name:



Pros:

– as on paper

Cons:

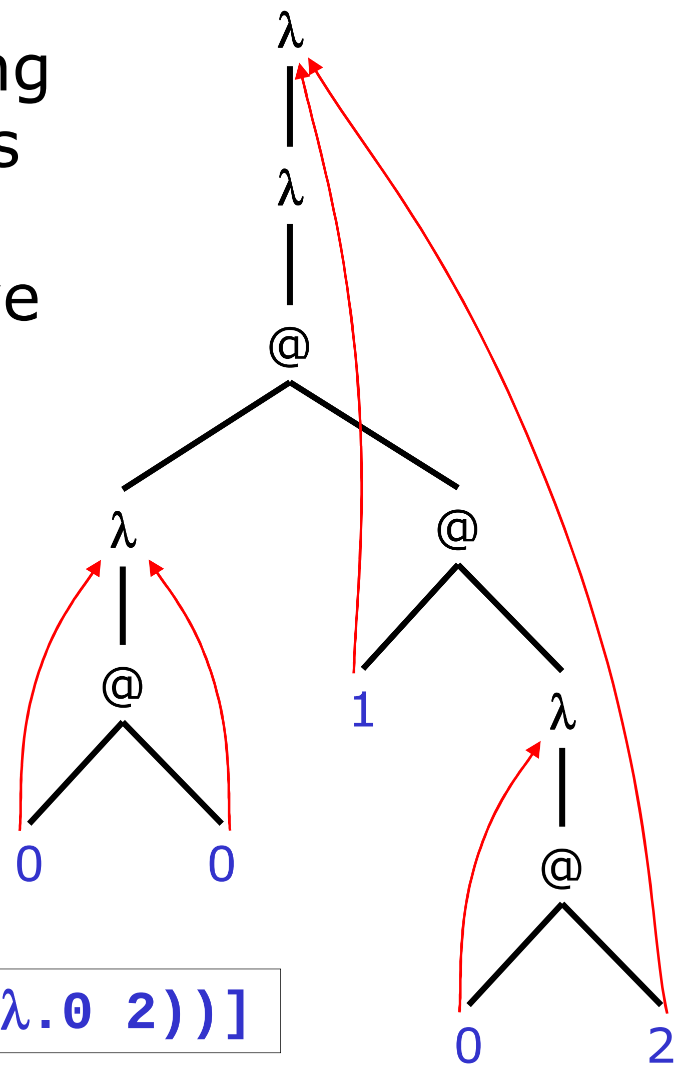
– α-conversion

– quotient by α

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

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

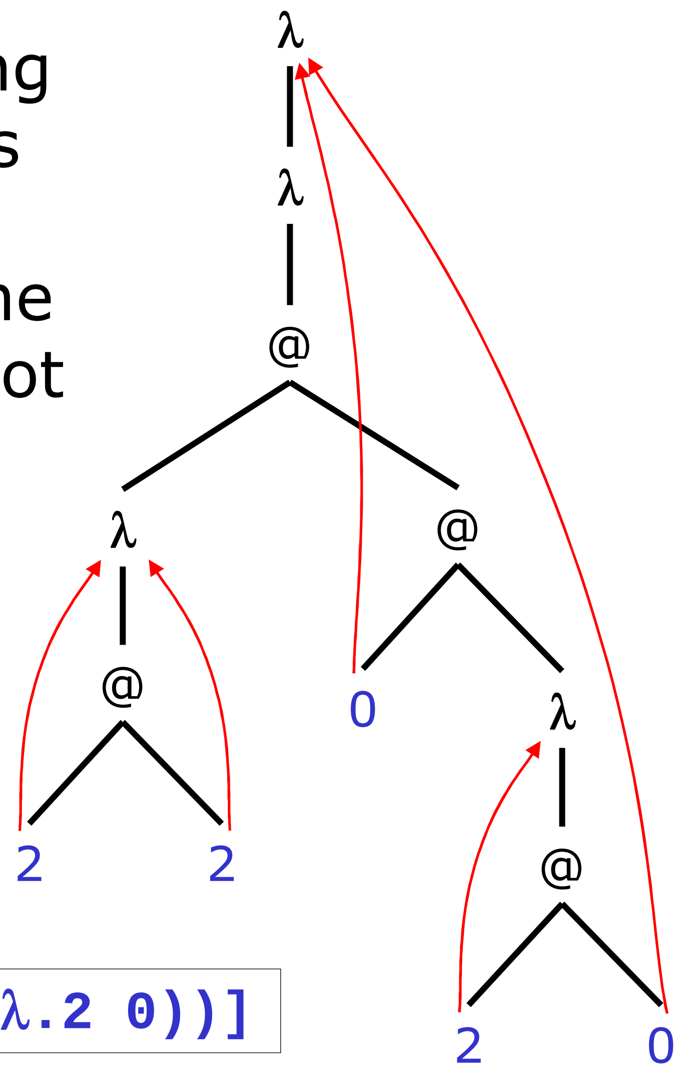
Cons:

– 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.

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

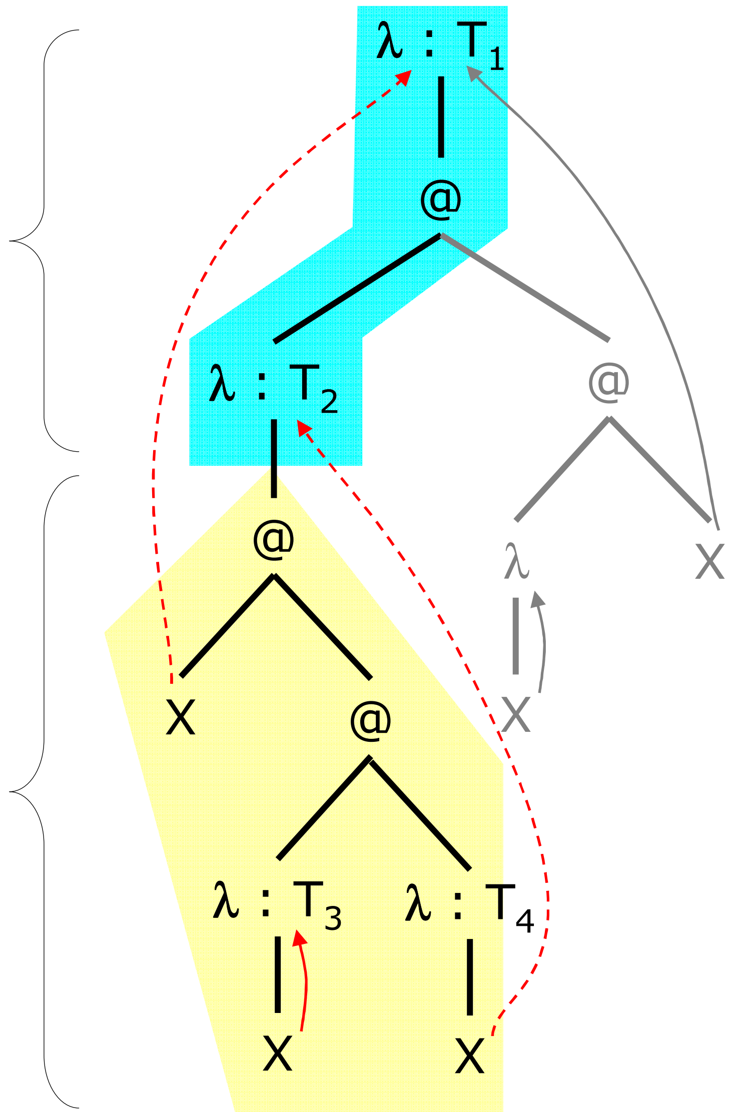
Bound and Free Variables

environment *E*

typing judgment

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

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

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

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

[k –> u]t

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

Full β-reduction in Locally Nameless

$$\frac{}{((\lambda{:}S.\ t_1)\ \ t_2) \longmapsto (\{0 \to t_2\}\, t_1)}\ \textsc{Red-Beta}$$

$$\frac{t_1 \longmapsto t_1'}{(t_1\ \ t_2) \longmapsto (t_1'\ \ t_2)}\ \textsc{Red-App-1} \qquad \frac{t_2 \longmapsto t_2'}{(t_1\ \ t_2) \longmapsto (t_1\ \ t_2')}\ \textsc{Red-App-2}$$

$$\frac{(\{0 \to x\}\, t_1) \longmapsto (\{0 \to x\}\, t_1')}{(\lambda{:}S.\ t_1) \longmapsto (\lambda{:}S.\ t_1')}\ \textsc{Red-Abs} \qquad x \# t_1, x \# t_1'$$

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) Bindings in Environments

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

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

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

Sets are Better than Lists

Motivation for representing environment as lists:
bindings enter the environment one by one.

But environments only require a set interface.
So lists are just overspecifying our needs.

Reasonning about the high-level interface is nicer
than dealing with the low-level implementation.

Names pushed in the Environment

$$\frac{Quantify(x) \qquad (E,\, x:S) \vdash (\{0 \to x\}\, t_1) \;:\; T}{E \vdash (\lambda{:}S.\, t_1) \;:\; S \to T}\ \textrm{T-ABS}$$

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

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

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

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

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

Conclusions

Related Work

Annotated Bibliography

30+ references available on the POPLMark website.

Closely Related Work

– Gordon (1993): locally nameless for formal proofs.

– McKinna and Pollak (1993 to 1997): distinguishing bound and free variables, but with names for both.

– Nominal (Pitts, Gabbay, Urban... 2000 to 2006): different approach with similar interface in the end.

– POPLMark locally nameless (Leroy, Chlipala, 2006).

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 (Huet 89) (McKinna, McBride 04).

– 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 language constructions.

Thanks !