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

- Simple
- Intuitive
- Transparent

Formalize

Take paper

Coq

Translate

Arrange
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 \rightarrow ^\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 \rightarrow ^\beta t u
```
Preservation and Progress for System-F<:

POPLMark Part 1:
Properties of subtyping

POPLMark Part 2:
Rest of the formalization

[extended]

Our Challenge B
Properties of subtyping, including preservation by type substitution

[extended]

Our Challenge A
Preservation and progress for simply typed λ-calculus

[simplified]
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) Representation of Bindings
Each abstraction introduces a name:

Pros:
- as on paper

Cons:
- \( \alpha \)-conversion
- quotient by \( \alpha \)

\[ \lambda a. \lambda b. \ ((\lambda c. c c) \ (a \ (\lambda d. d a))) \]
A variable bearing an index \( k \) points towards the \( k^{\text{th}} \) abstraction above that variable:

\[
\lambda.\lambda. \left[ (\lambda.0\ 0) \ (1 \ (\lambda.0\ 2)) \right]
\]

Pros:
- \( \alpha \)-equivalence is identity

Cons:
- shifting free variables in the argument
- unshifting free variables in the body
A variable bearing an index $k$ points towards the $k^{th}$ abstraction on the path from the root to that variable:

Pros:
– $\alpha$-equivalence is identity

Cons:
– shifting bound variables in the argument
– unshifting bound variables in the body

$$\lambda \cdot \lambda \cdot [ (\lambda \cdot 2 \ 2) \ (0 \ (\lambda \cdot 2 \ 0 \)) ]$$
shift and subst

Properties of shifting and substitution.
Not very difficult, but fiddly.

\[
\begin{align*}
i \leq j & \implies j \leq i + m \implies \uparrow T^n j (\uparrow T^m i T') = \uparrow T^m (m + n) i T \\
i + m \leq j & \implies \uparrow T^n j \uparrow T^m i T' = \uparrow T^m m i (\uparrow T^n (j - m) T') \\
k \leq k' & \implies k' < k + n \implies \uparrow T^n k T[k' \mapsto T]_{\tau} = \uparrow T^n (n - 1) k T \\
k \leq k' & \implies \uparrow T^n k T[k' + n \mapsto T]_{\tau} \\
k' < k & \implies \uparrow T^n k T[k' \mapsto T]_{\tau} = \uparrow T^n (k + 1) T[k' \mapsto T]_{\tau} \\
k \leq k' & \implies \uparrow T^n k T[k \mapsto Top]_{\tau} = \uparrow T^n (Suc k') T[k \mapsto Top]_{\tau} \\
k \leq k' & \implies k' \leq k + n \implies \uparrow T^n k' (\uparrow T^n k T) = \uparrow T^n (n + n') k T \\
i \leq j & \implies T[Suc j \mapsto T]_{\tau} i \mapsto T[j - i \mapsto T V]_{\tau} = T[i \mapsto T]_{\tau} j \mapsto T V]_{\tau}
\end{align*}
\]

source: Berghofer 2005

Weakening in System-F<:
Statements are polluted by shifting.

\[
\Gamma \vdash t : T \implies \Delta @ \Gamma \vdash_{wf} \Delta @ \Gamma \vdash \uparrow ||\Delta|| \ 0 t : \uparrow T \ ||\Delta|| \ 0 T
\]
Bound and Free Variables

Environment $E$

Typing judgment $E |- t : T$

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 \( t \):

\[
\{ k \rightarrow u \} t
\]

\[
\begin{align*}
\{ k \rightarrow u \} [i] & \equiv \text{if } i = k \text{ then } u \text{ else } [i] \\
\{ k \rightarrow u \} [x] & \equiv [x] \\
\{ k \rightarrow u \} (t_1 \ t_2) & \equiv ((\{ k \rightarrow u \} \ t_1) \ (\{ k \rightarrow u \} \ t_2)) \\
\{ k \rightarrow u \} (\lambda : T. \ t_1) & \equiv \lambda : T. (\{(k + 1) \rightarrow u\} \ t_1)
\end{align*}
\]

Substitution a term \( u \) for a free variable \( z \) in a term \( t \):

\[
[k \rightarrow u] t
\]

\[
\begin{align*}
[k \rightarrow u] [i] & \equiv [i] \\
[k \rightarrow u] [x] & \equiv \text{if } x = z \text{ then } u \text{ else } [x] \\
[k \rightarrow u] (t_1 \ t_2) & \equiv (([k \rightarrow u] \ t_1) \ ([k \rightarrow u] \ t_2)) \\
[k \rightarrow u] (\lambda : T. \ t_1) & \equiv \lambda : T. ([k \rightarrow u] \ t_1)
\end{align*}
\]
Full $\beta$-reduction in Locally Nameless

\[
\frac{((\lambda: S. \ t_1) \ t_2) \rightsquigarrow (\{0 \to t_2\} \ t_1)}{} \quad \text{RED-BETA}
\]

\[
\frac{t_1 \rightsquigarrow t'_1}{(t_1 \ t_2) \rightsquigarrow (t'_1 \ t_2)} \quad \text{RED-APP-1}
\]

\[
\frac{t_2 \rightsquigarrow t'_2}{(t_1 \ t_2) \rightsquigarrow (t_1 \ t'_2)} \quad \text{RED-APP-2}
\]

\[
\frac{(\{0 \to x\} \ t_1) \to (\{0 \to x\} \ t'_1)}{(\lambda: S. \ t_1) \to (\lambda: S. \ t'_1)} \quad \text{RED-ABS} \quad x \# t_1, x \# t'_1
\]
Summary of Representations

<table>
<thead>
<tr>
<th></th>
<th>bound variables</th>
<th>free variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>names</strong></td>
<td>requires reasoning on $\alpha$-equivalence</td>
<td>ok</td>
</tr>
<tr>
<td><strong>de Bruijn</strong></td>
<td>ok</td>
<td>shifting is necessary</td>
</tr>
<tr>
<td><strong>indices</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>de Bruijn</strong></td>
<td>shifting is necessary</td>
<td>shifting is necessary</td>
</tr>
<tr>
<td><strong>levels</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Winner is: Locally Nameless**
2) Bindings in Environments
Environments as Lists or Sets?

Weakening Preserves Typing

**Paper:** \[ E \vdash S <: T \Rightarrow E, F \vdash S <: T \]

**Formal:** \[ E \vdash S <: T \land E \subseteq F \Rightarrow F \vdash S <: T \]

**where:** \[ E \subseteq F \equiv \forall x T, (x : T) \in E \Rightarrow (x : T) \in F \]

Substitution Preserves Typing

**Paper:** \[ E, z : U, F \vdash t : T \land E \vdash u : U \Rightarrow E, F \vdash [z \rightarrow u] t : T \]

**Formal:** \[ E \vdash t : T \land F \vdash u : U \land (z : U) \in E \land E \setminus z \subseteq F \Rightarrow F \vdash [z \rightarrow u] t : T \]

**where:** \[ E \setminus z \subseteq F \equiv \forall x T, (x : T) \in E \land x \neq z \Rightarrow (x : T) \in F \]
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

\[
\frac{\text{Quantify}(x)}{(E, x : S) \vdash (\{0 \rightarrow x\} t_1) : T \quad \text{T-ABS}}
\]

\[
E \vdash (\lambda S. t_1) : S \rightarrow T
\]

<table>
<thead>
<tr>
<th>Quantify(x) =</th>
<th>(\exists x \not\in E)</th>
<th>(\forall x \not\in E)</th>
<th>(\forall x \not\in L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weakening</td>
<td>swapping required</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>Substitution</td>
<td>ok</td>
<td>swapping required</td>
<td>ok</td>
</tr>
<tr>
<td>Transitivity</td>
<td>swapping required</td>
<td>ok</td>
<td>ok</td>
</tr>
</tbody>
</table>

**Weakening**

\[E \vdash t : T \Rightarrow E, F \vdash t : T\]

**Substitution**

\[E, z : U, F \vdash t : T \land E \vdash u : U \Rightarrow E, F \vdash [z \rightarrow u] t : T\]

**Transitivity**

\[E \vdash S <: Q \land E \vdash Q <: T \Rightarrow E \vdash S <: T\]
3) Formalization in Coq
Example: Weakening on Subtyping

**Informal:**

\[ E \vdash S <: T \Rightarrow E, F \vdash S <: T \]

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

**Formalizable:**

\[ E \vdash S <: T \land E \subseteq F \land \vdash F \ ok \Rightarrow F \vdash S <: T \]

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

**Formal:**

\textbf{Lemma sub_extension} : \(\forall E \ S \ T, E \vdash S <: T \Rightarrow \forall F, E \text{ inc } F \Rightarrow \text{ ok } F \Rightarrow F \vdash S <: T\).

\texttt{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.
<table>
<thead>
<tr>
<th></th>
<th>Simply typed λ-calculus</th>
<th>Properties of subtyping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitions</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Axioms</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lemmas</td>
<td>26</td>
<td>34</td>
</tr>
<tr>
<td>Theorems</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Lines of proofs</td>
<td>63</td>
<td>104</td>
</tr>
<tr>
<td>Number of tactics</td>
<td>202</td>
<td>279</td>
</tr>
<tr>
<td>Non-dummy tactics in the main proofs:</td>
<td>36</td>
<td>67</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>Author</th>
<th>Tactics</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jérome Vouillon</td>
<td>431</td>
<td>de-Bruijn indices</td>
</tr>
<tr>
<td>Aaron Stump</td>
<td>1147</td>
<td>names / levels</td>
</tr>
<tr>
<td>Xavier Leroy</td>
<td>630</td>
<td>locally nameless</td>
</tr>
<tr>
<td>Hirschowitz, Maggesi</td>
<td>1615</td>
<td>de-Bruijn (nested)</td>
</tr>
<tr>
<td>Adam Chlipala</td>
<td>342</td>
<td>locally nameless</td>
</tr>
<tr>
<td>Arthur Charguéraud</td>
<td>233</td>
<td>locally nameless</td>
</tr>
</tbody>
</table>
Conclusions
Related Work

Annotated Bibliography
30+ references available on the POPLMark website.

Closely Related Work
– 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 $\lambda$-calculus (e.g. confluence).
- Address more complex type systems (CoC).
- Support more advanced language constructions.
Thanks!