Connecting Loose Ends

- freshness function (recap)
- POPLmark challenge
- future work

Freshness Function

- A criterion when a construction that "picks a fresh atom" is independent of which fresh atom is chosen:

Given $pt_{\alpha,\iota}$, at_α and a function $h : \alpha \to \iota$:

- if $\mathsf{finite}(\mathsf{supp}_\alpha\ h)$ and

  $\exists a.\ a \# h \wedge a \# (h\,a)$

  then

$$\exists! fr.\ \forall a.\ a \# h \Rightarrow (h\,a) = fr$$

Freshness Function

- A criterion when a construction that "picks a fresh atom" is independent of which fresh atom is chosen:

Given $pt_{\alpha,\iota}$, at_α and a function $h : \alpha \longrightarrow \iota$:

- if $\mathsf{finite}(\mathsf{supp}_\alpha\ h)$ and

  $\exists a.\ a \# h \wedge a \# (h\,a)$

  then

$$\exists! fr(h).\ \forall a.\ a \# h \Rightarrow (h\,a) = fr(h)$$

Proof (Existence)

From the assumption there exists an a_0 with

1. $a_0 \# h$ and 2. $a_0 \# (h\ a_0)$

Proof (Existence)

From the assumption there exists an a_0 with

1. $a_0 \# h$ and 2. $a_0 \# (h\ a_0)$

We take $fr(h)$ to be $h\ a_0$ and have

3. $a \# h$ and claim that $(h\ a) = (h\ a_0)$

Proof (Existence)

From the assumption there exists an a_0 with

1. $a_0 \# h$ and 2. $a_0 \# (h\ a_0)$

We take $fr(h)$ to be $h\ a_0$ and have

3. $a \# h$ and claim that $(h\ a) = (h\ a_0)$

Trivially true for $a = a_0$. For $a \neq a_0$:

4. $a \# (h\ a_0)$

since $a \# h$ (3), $a \# a_0$, $\mathsf{finite}(\mathsf{supp}\ h)$, $\mathsf{finite}(\mathsf{supp}\ a_0)$ and thus

$$\mathsf{supp}(h\ a_0) \subseteq \mathsf{supp}(h) \cup \mathsf{supp}(a_0)$$

Proof (Existence)

From the assumption there exists an a_0 with

1. $a_0 \;\#\; h$ and 2. $a_0 \;\#\; (h\;a_0)$

We take $fr(h)$ to be $h\;a_0$ and have

3. $a \;\#\; h$ and claim that $(h\;a) = (h\;a_0)$

Trivially true for $a = a_0$. For $a \neq a_0$:

4. $a \;\#\; (h\;a_0)$

We get from (2,4) and (1,3):

5. $(a\,a_0){\bullet}(h\;a_0) = (h\;a_0)$
6. $(a\,a_0){\bullet}(h\;a_0) = ((a\,a_0){\bullet}h)((a\,a_0){\bullet}a_0) = (h\;a)$

Proof (Uniqueness)

From the assumption there exists an a_0 with $a_0 \# h$ and we have

1. $\forall a.\ a \# h \;\Rightarrow\; (h\ a) = fr_1$
2. $\forall a.\ a \# h \;\Rightarrow\; (h\ a) = fr_2$

Clearly, $fr_1 = fr_2$.

The freshness function can be used to define the lambda-case for recursive functions:

$$g([a].t) = fr(\lambda a'.\ f_{LAM}\, a'\, (a\, a')\bullet t)$$

and so for all "fresh" atoms a'' we have

$$g([a].t) = f_{LAM}\, a''\, (a\, a'')\bullet t$$

POPLmark Challenge

"Many proofs about programming languages are long, straightforward, and tedious, with just a few interesting cases. Their complexity arises from the management of many details rather than from deep conceptual difficulties; yet small mistakes or overlooked cases can invalidate large amounts of work. These effects are amplified as languages scale: it becomes hard to keep definitions and proofs consistent, to reuse work, and to ensure tight relationships between theory and implementations. Automated proof assistants offer the hope of significantly easing these problems."

POPLmark Challenge

However, despite much encouraging progress in recent years and the availability of several mature tools, their use is still not commonplace. . . . We would like to answer two questions:

- What is the current state of the art in formalizing language metatheory and semantics? What can be recommended as best practices for groups (typically not proof-assistant experts) embarking on formalizing language definitions, either small- or large-scale?
- What improvements are needed to make the use of tool support commonplace?

problems."

Some Tasks

- type-language of $F_{<:}$ (transitivity of subtyping)
- considering terms (evaluation, type-soundness — preservation and progress)

Start small, be able to reuse code:

- add records
- add pattern-matching (interesting binding issues)

Evaluation Criteria

- Overhead of formalisation should not be prohibitive (they say de Bruijn indices do not meet this criterion).
- The representation and proof-assistant syntax should not depart too radically from the usual conventions.
- The infrastructure should be usable (after, say, one semester of training) by someone who is knowledgeable about programming language theory but not an expert in theorem prover technology.

My Existing Implementation

- abstract theory about permutations
- automated construction of a new type corresponding to the alpha-equivalence classes (almost completed, summer)
- automation for induction principles and recursion combinators (non-trivial, autumn?)
- automated rule inductions with build-in variable convention (???)

My Existing Implementation

- abstract theory about permutations

$$(\forall x\, a)\ P\,(\mathsf{Var}(a))\,x$$

$$(\forall x\, t_1\, t_2)\ ((\forall z)\ P\, t_1\, z) \wedge ((\forall z)\ P\, t_2\, z) \Rightarrow P\,(\mathsf{App}(t_1, t_2))\,x$$

$$\frac{(\forall x\, a\, t)\ a \# x \wedge\ ((\forall z)\ P\, t\, z) \Rightarrow P\,([a].t)\,x}{(\forall x\, t)\ P\, t\, x}$$

variable convention (???)

My Existing Implementation

If a finite set A supports (f_1, f_2, f_3) and

$$(\exists a \notin A)(\forall t\, x)(a \,\#\, f_3\, a\, t\, x)$$

then there exists a function $\hat{f}$ satisfying:

$$\begin{array}{lcl}\hat{f}(Var\ a) & = & f_1\, a \\ \hat{f}(App\ t_1\, t_2) & = & f_2\, t_1\, t_2\, \hat{f}(t_1)\, \hat{f}(t_2) \\ \hat{f}(Lam\ [a].t) & = & f_3\, a\, t\, \hat{f}(t) \\ & & \text{for all } a \notin A\end{array}$$

My Existing Implementation

- abstract theory about permutations
- automated construction of a new type corresponding to the alpha-equivalence classes (almost completed, summer)
- automation for induction principles and recursion combinators (non-trivial, autumn?)
- automated rule inductions with build-in variable convention (???)

My Existing Implementation

$$
\begin{array}{l}
(\forall x\, \Gamma\, a\, \tau)\ \mathsf{valid}(\Gamma) \wedge a : \tau \in \Gamma \Rightarrow P\ \Gamma\ \mathsf{Var}(a)\ \tau\ x \\[1em]
(\forall x\, \Gamma\, M\, \tau\, \sigma) \\
\qquad\qquad \Gamma \vdash M : \tau \to \sigma \wedge P\ \Gamma\ M\ (\tau \to \sigma)\ x \wedge \\
\qquad\qquad \Gamma \vdash N : \tau \wedge P\ \Gamma\ N\ \tau\ x \Rightarrow \\
\qquad\qquad P\ \Gamma\ (App\, M\, N)\ \sigma\ x \\[1em]
(\forall x\, \Gamma\, a\, M\, \tau\, \sigma) \\
\qquad\qquad a \# x \wedge a \# \Gamma \wedge (a : \tau, \Gamma) \vdash M : \sigma \wedge \\
\qquad\qquad ((\forall z)\ P\ (a : \tau, \Gamma)\ M\ \sigma\ z) \Rightarrow \\
\qquad\qquad P\ \Gamma\ (Lam\,[a].M)\ (\tau \to \sigma)\ x \\
\hline
\qquad\qquad\qquad \Gamma \vdash M : \tau \Rightarrow P\ \Gamma\ M\ \tau\ x
\end{array}
$$

My Existing

$$\forall x\, M.\ P\ M\ M\ x$$

$$\begin{array}{l}
\forall x\, a\, M\, M'. \\
\quad a \# x \wedge M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \Rightarrow \\
\quad P\ (Lam\,[a].M))\ (Lam\,[a].M'))\ x
\end{array}$$

$$\begin{array}{l}
\forall x\, M\, M'\, N\, N'. \\
\quad M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \wedge \\
\quad N \longrightarrow_1 N' \wedge (\forall z.\ P\ N\ N'\ z) \Rightarrow \\
\quad P\ (App\, M\ N)\ (App\, M'\, N')\ x
\end{array}$$

$$\begin{array}{l}
\forall x\, a\, M\, M'\, N\, N'. \\
\quad a \# (x, N, N') \wedge \\
\quad M \longrightarrow_1 M' \wedge (\forall z.\ P\ M\ M'\ z) \wedge \\
\quad N \longrightarrow_1 N' \wedge (\forall z.\ P\ N\ N'\ z) \Rightarrow \\
\quad P\ (App\,(Lam\,[a].M)\ N))\ (M'[a := N'])\ x
\end{array}$$

$$\overline{\qquad M \longrightarrow_1 N \Rightarrow P\ M\ N\ x \qquad}$$

Future (I)

- recursion is definitely in need of simplification
  - Andy Pitts needs 11 densely written pages to justify that the NBE-function with names is justified.
- so far everything is “optimised” for HOL-based provers — obvious question, what would the nominal approach look for non-HOL-based provers? It might turn out that one cannot transparently have nominal types in Coq, for example.

Future (II)

- so far only single binders are considered (this is probably not enough, the theory on paper extends to more interesting cases)
- we already pushed the limits of Isabelle quite a bit (surprisingly):
  - the existing datatype-package needed to be extended
  - the definition/overloading mechanism needed to be fixed
- the type-class system proved very valuable, but a limitation is just a hassle (can this be fixed, or is this an in-principle limitation?)

Future (III)

- it seems difficult to characterise the set of finitely supported functions; this means they need to be defined explicitly, but then the Abs-Rep-mechanism becomes burdensome

  one needs to be able to overload λ, which goes to the fundamentals of HOL (can this be done; semantics)

- it would have been extremely nice if the nominal project could have already contributed to the "SN-proof/extraction of programs"-project; but the nominal work is just not yet in a state where it is really usable for this

Future (IV)

- of course one has to provide worked examples:
  - lambda-calculus (in various forms), my term-calculus from the PhD
  - definition of SML/FreshML — showing some form of correctness (VanInwegen)

    "Proving theorems about substitutions (and related operations such as alpha-conversion) required far more time and HOL code than any other variety of theorem."

What the POPLmark-People Say:

"Our own preliminary experiments with Urban's methods have been encouraging"

Well, it seems still some time and research before I can make good on this.

The End

Thank you for coming. Any comments and criticism about the lecture and work is most welcome!