Changes in PL Research

- Increasing complexity of language features being considered
  - E.g., module systems, dependently typed programming, types ensuring resource bounds or deadlock freedom, …

- Increasing concern for scale and integration

- Decreasing cycle time for tech transfer
  - Extensions to production languages
    - e.g., generics in Java/C#
  - Adoption of research languages
    - e.g. (OCaml, F#, Haskell, Scheme)

Big changes in the way proofs are communicated…
We have proved the soundness of LIL$_C$, in the style of [34], and the decidability of type checking. Full proofs are in the technical report.

**Theorem 1 (Preservation).** If $\Sigma \vdash P : \tau$ and $P \rightarrow P'$, then $\exists \Sigma'$ such that $\Sigma' \vdash P' : \tau$.

**Theorem 2 (Progress).** If $\Sigma \vdash P : \tau$, then either the main expression in $P$ is a value, or $\exists P'$ such that $P \rightarrow P'$.

Proof sketch: by standard induction over the typing rules.
We’ve Got a Problem Here

• As a practical matter, a large fraction of the proofs being done about programming languages today are not possible for humans to check.
Are Proof Checkers the Answer?

• Proof checking (or “proof assistant”) technology has made amazing strides in recent years
  – Several mature systems
    • E.g., Coq, HOL, Isabelle, Twelf, MetaPRL, etc., etc.
  – Some very impressive achievements in several domains
A Marriage Made in Heaven

So we’ve got…

A. A community with a burning need
B. A community with a great technology

Can we put them together?
Lots of Work Already Underway

- Leroy’s verified C compiler
- Nipkow et al’s formalization of a large part of Java
- Appel et al’s Foundational Proof-Carrying Code project
- Crary et al’s machine-checked development of a typed assembly language
- Harper et al’s formalization of Standard ML
- Sewell et al’s formalization of TCP/IP
- Etc., etc.
But We’re Not There Yet!

Challenges:

– Current achievements are mostly heroic efforts by heroic individuals and small teams
– Many different proof assistants and diverse technical machinery
– Lots of black magic; high cost of entry; little sharing of knowledge across projects
– Some significant unresolved technical issues
  • Lightweight methods for reasoning about variable binding
  • Scalability and modularity of proofs

What’s needed is some synergy…
Vision

A world where every PL paper is accompanied by a machine-checked appendix

• Plan:
  – Identify current best practices
  – Gather community consensus around them
  – Build additional tools and other infrastructure as needed
  – Address technical challenges specific to programming language research

First step…
The PopIIMark Challenge

• A set of benchmark problems to help evaluate progress in the area
  – Based around metatheory of F<:, a typed lambda-calculus with polymorphism and subtyping
  – Presented at TPHOLs 2005
• Has generated tremendous interest in both PL and theorem proving communities
  – many solutions submitted
  – 6 different proof assistants
  – 7 different treatments of binding
• Much has been learned
  – JAR special issue CFP now open
Community Development

- Wiki & Mailing list for POPLmark challenge
  - Gathering place for news/solutions/discussion/advice

- Workshop on Mechanizing Metatheory
  - 4th Wksp, 4 Sep 2009, Edinburgh

- Using Proof Assistants for Programming Language Research or, How to write your next POPL paper in Coq
  - Tutorial for novices
  - POPL tutorial, January, 2008
  - Oregon Summer School, June 2008
Engineering Formal Metatheory
Resolving technical issues

• *Engineering Formal Metatheory*
  – Brian Aydemir, Arthur Charguéraud, Benjamin Pierce, Randy Pollack, and Stephanie Weirich
  – POPL 2008

• Describes a lightweight first-order methodology for representing binding and specifying induction principles

• Two essential components:
  – Locally nameless representation
  – Cofinite name quantification
What is so hard about binding?

- **Alpha-equivalence**
  - Identify "\( \lambda x.x \)" and "\( \lambda y.y \)"

- **Barendregt convention**
  - Assumption that bound variables are "sufficiently fresh"

```agda
Inductive exp : Set :=
  | var : atom -> exp
  | abs : atom -> exp -> exp
  | app : exp -> exp -> exp.
```
Alpha-equivalence

- Important when we need to compare terms with binding structure:
  - Type system of polymorphic language
  - Confluence for pure lambda calculus

- Formalism simpler if alpha-equals is "="

  Lemma preservation: forall G e t,
  typing G e t -> red e e' ->
  exists t', alpha_eq t t' /\
  typing G e' t'.

  Lemma preservation: forall G e t,
  typing G e t -> red e e' -> typing G e' t.
Barendregt Convention

Fixpoint subst (x:atom) (u:exp) (e:exp) {struct e} :=
match e with
| var y     => if x == y then u else e
| app t1 t2 => app (subst x u t1) (subst x u t2)
| abs z t   => abs z (subst x u t)
end.

• What if z == x?
• What if z in free variables of u?
Existing approaches

• No completely satisfactory mechanism for binding
  – **Name representation**: must explicitly rename terms, define alpha-equivalence
  – **de Bruijn indices**: difficult to work with as must shift indices
  – **Nominal logic**: only available in Isabelle/HOL
  – **HOAS**: exotic terms, specialized logic
**Locally nameless representation**

Names for free variables

de Bruijn indices for bound variables

Example: Untyped lambda calculus terms

\[ t ::= \text{bvar } i \mid \text{fvar } x \mid \text{app } t_1 t_2 \mid \text{abs } t \]

\[ \lambda x. \lambda y. (x \ y) \ z \quad \text{and} \quad \lambda w. \lambda v. (w \ v) \ z \]

represented as

\[ \text{abs } (\text{abs } (\text{app } (\text{app } (\text{bvar } 1) (\text{bvar } 0)) (\text{fvar } "z")) \]

Inductive \( \text{exp} : \text{Set} :\) :=

| \text{bvar} : \text{nat} \to \text{exp} |
| \text{fvar} : \text{atom} \to \text{exp} |
| \text{abs} : \text{exp} \to \text{exp} |
| \text{app} : \text{exp} \to \text{exp} \to \text{exp}. |
Basic operations

• Variable opening
  $t^u$ replace bound index 0 with exp $u$

• Free variable calculation
  $FV(t)$ finite set of free atoms in $t$

• Substitution
  $[x \mapsto u]t$ replace free variable $x$ with $u$

• All operations have simple definitions.
Variable opening

Fixpoint open_rec (k : nat) (f : exp) (e : exp) {struct e} : exp :=
  match e with
  | bvar i       => if k == i then f else e
  | fvar x       => fvar x
  | abs e1       =>
    abs (open_rec (1 + k) f e1)
  | app e1 e2    => app (open_rec k f e1) (open_rec k f e2)
  end.

Notation "e ^ u" := (open_rec 0 u e).
Free variable substitution

Fixpoint subst (z : atom) (u : exp) (e : exp) {struct e} : exp :=
  match e with
  | bvar i    => bvar i
  | fvar x    => if x == z then u else e
  | abs T e1  => abs T (subst z u e1)
  | app e1 e2 => app
      (subst z u e1) (subst z u e2)
  end.
Free variable calculation

Fixpoint \( \text{fv} \) \((e : \text{exp})\)
\{\text{struct } e\} : \text{atoms} :=

match \( e \) with
  | \text{bvar } i \Rightarrow \{\}
  | \text{fvar } x \Rightarrow \text{singleton } x
  | \text{abs } e_1 \Rightarrow (\text{fv } e_1)
  | e_1 \text{ e}_2 \Rightarrow (\text{fv } e_1)
    \text{ `union` (fv } e_2)
Local closure

- Not all members of type term are lambda calculus terms
  - abs (bvar 3)?
- Predicate lc picks out members datatype that are *locally-closed*
- Definitions respect local closure
  - If lc u and lc t, then lc ([x ↦ u]t)
  - If t ↦ t' then lc t and lc t'
Managing local closure

- Many theorems need not refer to local closure
  - If $t \rightarrow t'$ and $t \rightarrow t''$, then $t' = t''$

- Some theorems require it, tactics discharge assumptions
  - If $x \neq y$ and $lc\ u$, then
    $[x \mapsto u](t^y) = ([x \mapsto u]t)^y$
Induction principles

Definition of typing rules generates an induction principle for typing derivations

\[
\begin{align*}
\text{ok } E & \quad (x:T) \in E \\
E \vdash \text{fvar } x : T \\
E \vdash t_1 : S \rightarrow T & \quad E \vdash t_2 : S \\
E \vdash \text{app } t_1 \ t_2 : T \\
x \notin \text{FV } t \cup \text{dom}(E) & \quad E,x:S \vdash t^x : T \\
E \vdash \text{abs } t : S \rightarrow T
\end{align*}
\]
Exists vs. Cofinite Quantification

Induction hypothesis holds for some particular, unknown $x$

$$x \notin FV \ t \cup dom(E) \quad E,x:S \vdash t^x : T$$

$$E \vdash \text{abs } t : S \rightarrow T$$

Induction hypothesis holds for all but some finite set of variables.

$$\forall x \notin L \quad E,x:S \vdash t^x : T$$

$$E \vdash \text{abs } t : S \rightarrow T$$
Weakening Lemma

If \( E, G \vdash t : T \) and \( \text{ok}(E, F, G) \)
then \( E, F, G \vdash t : T \)

Proof (?):

by induction on \( E, G \vdash t : T \)

Abstraction case:

\[
\begin{array}{c}
\text{if} \quad x \notin \text{FV}(t \cup \text{dom}(E,G)) \quad \text{then} \quad E,G,x:S \vdash t^x : T \\
\end{array}
\]

\[
E,G \vdash \text{abs } t : S \rightarrow T
\]

WTP \( E,F,G \vdash \text{abs } t : S \rightarrow T \)

By typing rule, holds if \( E,F,G,x:S \vdash t^x : T \)
and \( x \notin \text{FV}(t \cup \text{dom}(E,F,G)) \)
Weakening Lemma

If \( E, G \vdash t : T \) and \( \text{ok} (E, F, G) \)
then \( E, F, G \vdash t : T \)

Proof: by induction on \( E, G \vdash t : T \)

abstraction case: \[
\forall x \not\in L \quad E, G, x: S \vdash t^x : T \\
\overline{E, G \vdash \text{abs} \ t : S \rightarrow T}
\]

WTP: \( E, F, G \vdash \text{abs} \ t : S \rightarrow T \)

By rule, holds if exists a set \( L' \), such that
\[
\forall x \not\in L', \ E, F, G, x: S \vdash t^x : T
\]

IH: \( \forall x \not\in L, \ \text{ok} (E, F, G, x: S) \Rightarrow E, F, G, x: S \vdash t^x : T \)

Choose \( L' = \text{dom} (E, F, G) \cup L \)

By definition, \( \forall x \not\in L', \ \text{ok} (E, F, G, x: S) \)
Renaming lemma

• Similar reasoning proves substitution lemma
  If $E, x:S, F \vdash t : T$ and $E \vdash u : S$
  then $E, F \vdash [z \mapsto u]t : T$

• Important corollary of substitution and weakening

Renaming:
If $x \not\in \text{FV } t \cup \text{dom } E$ and $E, x:S \vdash t^x : T$
then for all $y \not\in \text{FV } t \cup \text{dom } E$, $E, y:S \vdash t^y:T$
Corollaries of renaming

• Renaming lemma gives strongest intro form:

If exists $x$, s.t. $x \notin \text{FV } t \cup \text{dom } E$ and $E, x : S \vdash t^x : T$ then $E \vdash \text{abs } t : S \rightarrow T$

• And strongest inversion principle:

If $E \vdash \text{abs } t : S \rightarrow T$ then for all $x \notin \text{FV } t \cup \text{dom } E$, $E, x : S \vdash t^x : T$
Equivalence of systems

• Renaming lemma also shows equivalence of two systems:

\[ E \vdash t : T \text{ with exists-fresh rule for abs} \]
\[ E \vdash t : T \text{ if and only if} \]
\[ E \vdash t : T \text{ with cofinite rule for abs} \]

• We are proving properties about the language we actually care about!
General Form of Development

• Def. of language syntax
• Def. of variable opening and local closure
• Def. of free variable substitution
• Def. of free variable function
• Interaction lemmas
• Def. of semantic relations using cofinite quantification for binders
• Show semantic relations respect local closure
• Substitution and weakening lemmas for each relation w/ binding
• Preservation and Progress
• Derive renaming lemmas
Code Distribution

- Reference *examples* & supporting experience
  - multiple calculi (STLC, F<:, CoC)
  - multiple theorems (type soundness, confluence)
- A *library* that supports this methodology
  - atoms, finite sets
  - reasoning about freshness
  - representing environments

Let's look closer at POPLmark…
syntax of types and terms, defn. of typing relation…

"The TCB"

facts about syntax, substitution…

"The overhead"

facts about well-formedness, environments…

"Things we should have proved on paper but didn't"

Substitution lemma, preservation, progress…

"The meat"
Basic operations (fv and subst)

Tactic to choose fresh atoms

Properties of open, fv and subst

Properties of subst and local closure
Fsub_Lemmas.v

type well-formedness

environment well-formedness

substitution in environments

regularity of relations

hints about local closure
Proofs of lemmas from the Appendix of POPLmark paper
<table>
<thead>
<tr>
<th>Author</th>
<th>Binding</th>
<th>Lemmas</th>
<th>Proof steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vouillon</td>
<td>de Bruijn</td>
<td>30</td>
<td>402</td>
</tr>
<tr>
<td>Leroy</td>
<td>Locally nameless</td>
<td>49</td>
<td>495</td>
</tr>
<tr>
<td>Stump</td>
<td>Levels/names</td>
<td>56</td>
<td>938</td>
</tr>
<tr>
<td>Hirschowitz &amp; Maggesi</td>
<td>de Bruijn (nested datatype)</td>
<td>49</td>
<td>1574</td>
</tr>
<tr>
<td>Chlipala</td>
<td>Locally nameless</td>
<td>23</td>
<td>75</td>
</tr>
<tr>
<td>This work</td>
<td>Locally nameless</td>
<td>22</td>
<td>101</td>
</tr>
</tbody>
</table>
McKinna & Pollack Rule

Exists-fresh: IH holds for some fresh \( x \)

\[
\begin{align*}
  x \notin \text{FV} & \quad t \cup \text{dom}(E) \\
  E,x : S \vdash t^x : T \\ 
  E \vdash \text{abs } t : S \to T
\end{align*}
\]

Cofinite: IH holds for all but some unknown set

\[
\begin{align*}
  \forall x \notin L & \\
  E,x : S \vdash t^x : T \\ 
  E \vdash \text{abs } t : S \to T
\end{align*}
\]

Forall-fresh: IH holds for all fresh variables

\[
\begin{align*}
  \forall x \notin \text{FV} & \quad t \cup \text{dom}(E) \\
  E,x : S \vdash t^x : T \\ 
  E \vdash \text{abs } t : S \to T
\end{align*}
\]
<table>
<thead>
<tr>
<th>Forall   vs. Cofinite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define system with forall-fresh rules</td>
</tr>
<tr>
<td>Define swapping and show relations are stable under swapping</td>
</tr>
<tr>
<td>Show exists-fresh intro</td>
</tr>
<tr>
<td>Prove weakening and substitution</td>
</tr>
<tr>
<td>Prove type soundness</td>
</tr>
</tbody>
</table>
Conclusions

• Can use Coq's standard mechanisms for reasoning (inductive defs, tactics, etc.)
• Swapping does not appear to be essential.
• Seldom need to rename during proofs. IH applies to an infinite # of suitably fresh variables.
• Specialized tactics help
  – local closure obligations
  – fresh variable introduction
Thanks to

Brian Aydemir
Arthur Charguéraud (INRIA)
Randy Pollack (Edinburgh)
Peter Sewell (Cambridge)
Aaron Bohannon
Matthew Fairbairn (Cambridge)
J. Nathan Foster
Benjamin Pierce
Jeffrey Vaughan
Dimitrios Vytiniotis
Geoffrey Washburn
Steve Zdancewic
Fixpoint subst (t : term) (x : nat) (t' : term) {struct t} : term :=
  match t with
  | var y =>
    match lt_eq_lt_dec y x with
    | inleft (left _) => var y
    | inleft (right _) => t'
    | inright _ => var (y - 1)
    end
  | abs T1 t2 =>
    abs T1 (subst t2 (1 + x) (shift 0 t'))
  | app t1 t2 => app (subst t1 x t')
    (subst t2 x t')
  | tabs T1 t2 => tabs T1
    (subst t2 x (shift_typ 0 t'))
  | tapp t1 T2 => tapp (subst t1 x t') T2
  end.