Engineering Formal Metatheory

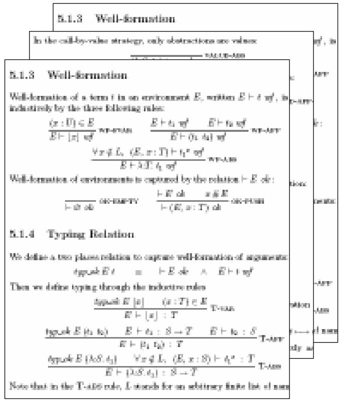
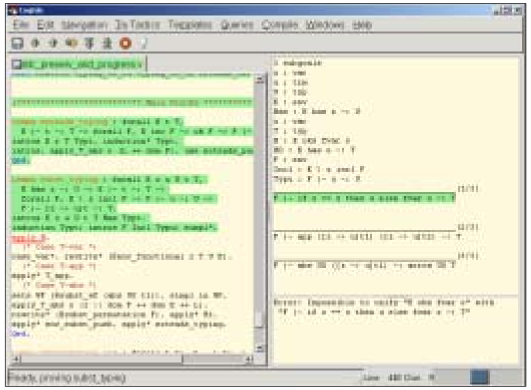
Arthur Charguéraud

Joint work with Brian Aydemir, Benjamin C. Pierce,
Randy Pollack and Stephanie Weirich

Talk at PPS

Paris, 2007-12-06

Motivation

A metatheory paper proof		A metatheory mechanized proof
	Mechanize	

– many tedious cases → use automation

– never 100% confident → machine-checked

– hard to reuse → reusable

– certified type-checkers? → a first step...

The POPLMark Challenge [1]

How to formalize metatheory with:

– a generally applicable method,

– faithful to informal practice style,

– reasonable infrastructure overhead,

– and using a technology with low cost of entry ?

Our contribution is the proposal of a novel style for formalizing metatheory that achieve these goals.

[1] Brian Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic; TPHOLs 2005.

Representations of Bindings

– With names: α-equivalence, variable capture

$$(x_1{:}T_1)..(x_2{:}T_2) \vdash ...\lambda x_3 ...\lambda x_4 ... x_3 ... x_2$$

– With de Bruijn indices: shifting of indices

$$T_1 \;..\; T_2 \vdash ...\lambda.....\lambda.....1....2$$

– With distinguished bound and free variables: ok?

$$(x_1{:}T_1)..(x_2{:}T_2) \vdash ...\lambda.....\lambda.....1....x_2$$

```
t := bvar i | fvar x | app t1 t2 | abs t
```

Locally Nameless

```
t := bvar i | fvar x | app t1 t2 | abs t
```

– Bound variables are represented using indices
    $\Rightarrow$ no α-equivalence
– Free variables are represented using names
    $\Rightarrow$ no shifting
– Bound and free variables are distinguished
    $\Rightarrow$ no capture

Only catch: the syntax allows ill-formed terms.
We need all bound variables to resolve to a binder.
For instance, **`"abs (bvar 2)"`** is not a valid term.

Opening Binders

Opening of the body of a term with some term u

λ(...λ...**1**...λ...**0**...**y**...**2**...)

gives

(...λ...**u**...λ...**0**...**y**...**u**...)

If t is the body of an abstraction, t^u is the result of opening it with u. Notation: we write t^x for $t^{(\text{fvar } x)}$.

$$\text{RED-BETA}\quad \frac{}{\mathsf{app}\ (\mathsf{abs}\ t)\ u \longmapsto (t^u)}$$

$$\text{TYPING-ABS}\quad \frac{(E, x{:}S) \vdash (t^x) : T}{E \vdash (\mathsf{abs}\ t) : S \to T}$$

Opening Binders – Test!

Reduction rule for reducing below abstractions?

Representation with names:

$$\frac{t_1 \longmapsto t_2}{\lambda x.t_1 \longmapsto \lambda x.t_2}$$

Locally nameless:

$$\frac{?}{\mathsf{abs}\ t_1 \longmapsto \mathsf{abs}\ t_2}$$

$$\frac{t_1^x \longmapsto t_2^x}{\mathsf{abs}\ t_1 \longmapsto \mathsf{abs}\ t_2}$$

Implementation of Opening

Opening t with u: $t^u \equiv \{0 \to u\}\, t$ where:

λ(. . . λ . . . **1** . . . λ . . . **0** . . . **y** . . . **2** . . .)

(. . . λ . . . **u** . . . λ . . . **0** . . . **y** . . . **u** . . .)

$$
\begin{array}{lcl}
\{k \to u\}\,(\mathsf{bvar}\ i) & = & \text{if } i = k \text{ then } u \text{ else } \mathsf{bvar}\ i \\
\{k \to u\}\,(\mathsf{fvar}\ x) & = & \mathsf{fvar}\ x \\
\{k \to u\}\,(\mathsf{app}\ t_1\ \ t_2) & = & \mathsf{app}\,(\{k \to u\}\, t_1)\ (\{k \to u\}\, t_2) \\
\{k \to u\}\,(\mathsf{abs}\ t) & = & \mathsf{abs}\ (\{(k+1) \to u\}\, t)
\end{array}
$$

Implementation of subst and FV

Substitution from variable z to term u:

$$
\begin{aligned}
[z \to u]\,(\mathsf{bvar}\ i) &= \mathsf{bvar}\ i \\
[z \to u]\,(\mathsf{fvar}\ x) &= \text{if } x = z \text{ then } u \text{ else } \mathsf{fvar}\ x \\
[z \to u]\,(\mathsf{app}\ t_1\ \ t_2) &= \mathsf{app}\ ([z \to u]\,t_1)\ \ ([z \to u]\,t_2) \\
[z \to u]\,(\mathsf{abs}\ t) &= \mathsf{abs}\ ([z \to u]\,t)
\end{aligned}
$$

Set of free variables in a term:

$$
\begin{aligned}
\mathsf{FV}(\mathsf{bvar}\ i) &= \varnothing \\
\mathsf{FV}(\mathsf{fvar}\ x) &= \{x\} \\
\mathsf{FV}(\mathsf{app}\ t_1\ \ t_2) &= \mathsf{FV}(t_1) \cup \mathsf{FV}(t_2) \\
\mathsf{FV}(\mathsf{abs}\ t) &= \mathsf{FV}(t)
\end{aligned}
$$

Properties of Operations

Substitution for a fresh name is the identity:

$$x \notin \mathsf{FV}(t) \quad \Rightarrow \quad [x \to u]\, t = t$$

Substitution distributes over open:

$$[x \to u]\,(t^w) = ([x \to u]\, t)^{([x \to u]\, w)}$$

Substitution commutes with open (on ≠ names):

$$[x \to u]\,(t^y) = ([x \to u]\, t)^y \qquad \text{when } x \neq y$$

Substitution is used to decompose opening:

$$[x \to u]\,(t^x) = t^u \qquad \text{when } x \notin \mathsf{FV}(t)$$

Proof of Preservation

Case beta-reduction:

$$\textsc{Typing-App}\ \dfrac{\textsc{Typing-Abs}\ \dfrac{(E, x{:}S) \vdash (t^x) : T}{E \vdash (\mathsf{abs}\ t) : S \rightarrow T} \qquad E \vdash u : S}{E \vdash (\mathsf{app}\ (\mathsf{abs}\ t)\ u) : T}$$

$$\textsc{Rewrite}\ \dfrac{\textsc{Substitute}\ \dfrac{(E, x{:}S) \vdash (t^x) : T \qquad E \vdash u : S}{E \vdash [x \rightarrow u]\,(t^x) : T}}{E \vdash (t^u) : T}$$

Well-formed Terms

Predicate "term t" caracterizes well-formed terms.

$$\textsc{Term-Var}\quad \frac{}{\mathsf{term}\ (\mathsf{fvar}\ x)} \qquad \textsc{Term-App}\quad \frac{\mathsf{term}\ t_1 \qquad \mathsf{term}\ t_2}{\mathsf{term}\ (\mathsf{app}\ t_1\ \ t_2)} \qquad \textsc{Term-Abs}\quad \frac{\mathsf{term}\ (t^x)}{\mathsf{term}\ (\mathsf{abs}\ t)}$$

(This gives us a natural induction principle for terms.)

Relations are restricted to well-formed terms:

$$\begin{array}{lcl} E \vdash t : T & \Rightarrow & \mathsf{term}\ t \\ t \longmapsto t' & \Rightarrow & \mathsf{term}\ t \ \wedge\ \mathsf{term}\ t' \end{array}$$

Operations preserve well-formedness:

$$\begin{array}{lcl} \mathsf{term}\ (\mathsf{abs}\ t) \ \wedge\ \mathsf{term}\ u & \Rightarrow & \mathsf{term}\ (t^u) \\ \mathsf{term}\ u \ \wedge\ \mathsf{term}\ t & \Rightarrow & \mathsf{term}\ ([x \to u]\, t) \end{array}$$

Restriction to Well-formed Terms

Call-by-value reduction for λ-calculus:

$$\textsc{value-abs}\ \frac{\mathsf{term}\ (\mathsf{abs}\ t)}{\mathsf{value}\ (\mathsf{abs}\ t)} \qquad \textsc{red-beta}\ \frac{\mathsf{term}\ (\mathsf{abs}\ t) \qquad \mathsf{value}\ u}{\mathsf{app}\ (\mathsf{abs}\ t)\ u \longmapsto t^u}$$

$$\textsc{red-app-1}\ \frac{t_1 \longmapsto t_1' \qquad \mathsf{term}\ t_2}{\mathsf{app}\ t_1\ t_2 \longmapsto \mathsf{app}\ t_1'\ t_2} \qquad \textsc{red-app-2}\ \frac{\mathsf{value}\ t_1 \qquad t_2 \longmapsto t_2'}{\mathsf{app}\ t_1\ t_2 \longmapsto \mathsf{app}\ t_1\ t_2'}$$

Typing Relation in STLC

An environment **E** has type **list (var × type)**, and **ok E** tells that variables are bound at most once.

TYPING-VAR

$$\frac{\mathsf{ok}\ E \qquad (x{:}T) \in E}{E \vdash (\mathsf{fvar}\ x) : T}$$

TYPING-APP

$$\frac{E \vdash t_1 : S \to T \qquad E \vdash t_2 : S}{E \vdash \mathsf{app}\ t_1\ t_2 : T}$$

TYPING-ABS

$$\frac{(E, x{:}S) \vdash (t^x) : T}{E \vdash (\mathsf{abs}\ t) : S \to T} \quad (\text{for } x \text{ fresh})$$

Results in STLC

Preservation theorem:

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

```
Theorem preservation : forall E t t' T,
  E |= t ~: T   ->   t --> t'   ->   E |= t' ~: T.
```

Substitution lemma:

$$E, z{:}S, F \vdash t : T \quad \Rightarrow \quad E \vdash u : S \quad \Rightarrow$$
$$E, F \vdash [z \rightarrow u]\, t : T$$

```
Lemma typing_subst : forall F U E t T z u,
  (E & z ~ S & F) |= t ~: T   ->   E |= u ~: S  ->
  (E & F) |= [z ~> u]t ~: T.
```

Which Quantification?

$$\frac{\text{Quantify}(x) \qquad (E, x{:}T_1) \vdash (t^x) : T_2}{E \vdash (\mathsf{abs}\ t) : T_1 \to T_2}$$

Quantification	Introduction	Elimination
Existential $x \notin \mathsf{FV}(t)$	maximally strong	very weak
Universal $\forall x \notin \mathsf{dom}(E)$	very weak	maximally strong
Cofinite $\forall x \notin L$	nearly always sufficient; easy to strenghten if not	maximally strong, provided cofinite used everywhere

Cofinite Quantification in Practice

TYPING-ABS
$$\frac{\forall x \notin L.\ (E, x{:}T_1) \vdash (t^x) : T_2}{E \vdash (\mathsf{abs}\ t) : T_1 \to T_2}$$

TERM-ABS
$$\frac{\forall x \notin L.\ \mathsf{term}\ (t^x)}{\mathsf{term}\ (\mathsf{abs}\ t)}$$

RED-ABS
$$\frac{\forall x \notin L.\ t_1^x \longmapsto t_2^x}{\mathsf{abs}\ t_1 \longmapsto \mathsf{abs}\ t_2}$$

```
| term_abs : forall L t,
    (forall x, x \notin L -> term (t^x)) -> term (abs t)
```

1) state all rules using cofinite quantification
$\Rightarrow$ no need to worry about freshness details

2) induction and inversion principles are available
$\Rightarrow$ automatically generated

3) to apply: instantiate L so as to avoid name clashes
$\Rightarrow$ a generic tactic automates this

4) [if necessary] derive the strong introduction form.
$\Rightarrow$ this proof is only two lines

DEMO

DEMO:

Simply typed λ-calculus

Proof of Preservation

Case beta-reduction:

$$\textsc{Typing-App}\ \dfrac{\textsc{Typing-Abs}\ \dfrac{(E, x{:}S) \vdash (t^{x}) : T}{E \vdash (\mathsf{abs}\ t) : S \to T} \qquad E \vdash u : S}{E \vdash (\mathsf{app}\ (\mathsf{abs}\ t)\ u) : T}$$

$$\textsc{Rewrite}\ \dfrac{\textsc{Substitute}\ \dfrac{(E, x{:}S) \vdash (t^{x}) : T \qquad E \vdash u : S}{E \vdash [x \to u]\,(t^{x}) : T}}{E \vdash (t^{u}) : T}$$

Main Developments

System $F_{<:}$

- binder-intensive: a good stress test,
- some proofs turn out shorter than on paper!
- others have shown that using specialized tactics, and further automation can shorten scripts a lot.

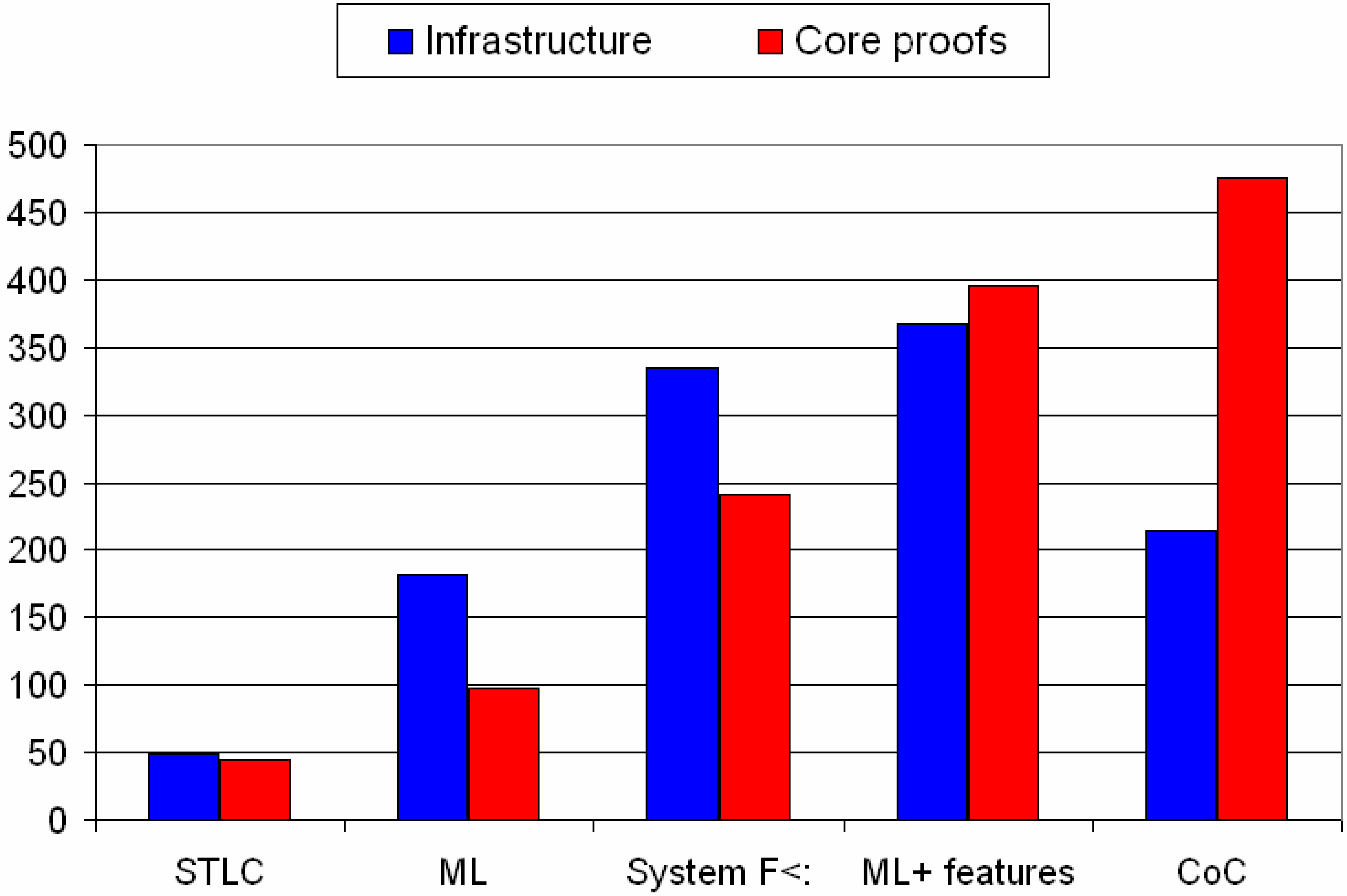
DEMO:

Calculus of Construction

DEMO:

ML + reference, exceptions, datatypes, patterns

Complexity of Developments



Measured in number of steps: a step is defined as the application of one tactic which is not "intro" or "auto" or a simple variations of these two.

Related to Locally Nameless

– De Bruijn (1972): representation with indices and suggestion of locally nameless.

– Huet (1989): *The Constructive Engine*. Source of inspiration for the implementations of Coq, LEGO, HOL4, Isabelle, EPIGRAM.

– Gordon (1993): locally nameless as an underlying representation for named terms.

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

– Leroy (2006): POPLMark, locally nameless, in Coq; later variations on his solution by others.

Related to Cofinite Quantification

– Universal quantification appears in McKinna and Pollak (1993-1997) and Leroy (2006).

– Gordon (1993): strengthened induction principle.
case-abs: $\exists L$, finite $L \wedge \forall x, x \notin L \wedge P(t) \Rightarrow P(\lambda x.t)$

– Krivine (1990), Ford and Mason (2001): cofinite quantification in definitions of alpha-equivalence.

– Gabbay and Pitts' *Nominal Logic* (1999-2003): idea of reasoning about the freshness of names by considering all but those in some finite set.

– Urban et al (2000-2007): Nominal Package for Isabelle/HOL: quotiented named terms, with a tool that generates induction principles.

Conclusion

Formalize programming language metatheory with:

locally nameless + cofinite quantification

– this leads to a generally applicable method,
– directly usable in general-purpose theorem provers,
– proofs closely follow their informal equivalents,
– the amount of infrastructure required is reasonable,
– and several templates provided as starting points.

Try it yourself!

The full paper and the documented Coq proof scripts are available from "http://arthur.chargueraud.org".

Freshness Side Conditions

Named representation:

$$\frac{(E, x{:}T_1) \vdash t \;:\; T_2}{E \vdash (\lambda x.t) \;:\; T_1 \rightarrow T_2}$$

Locally nameless:

$$\frac{(E, x{:}T_1) \vdash (t^x) \;:\; T_2}{E \vdash (\mathsf{abs}\ t) \;:\; T_1 \rightarrow T_2}$$

As an introduction form

premise(x)
$\Rightarrow$ conclusion(x)

premise(x) $\wedge$ x$\notin$FV(t)
$\Rightarrow$ conclusion

As an elimination form

conclusion(x) $\wedge$ x$\notin$dom E
$\Rightarrow$ premise(x)

conclusion $\wedge$ x$\notin$dom E
$\Rightarrow$ premise(x)