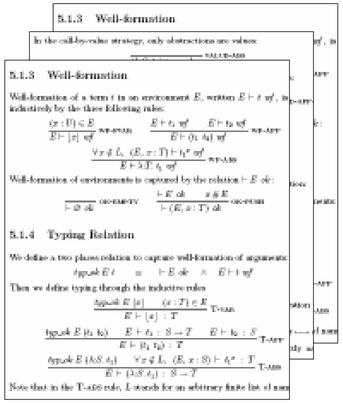
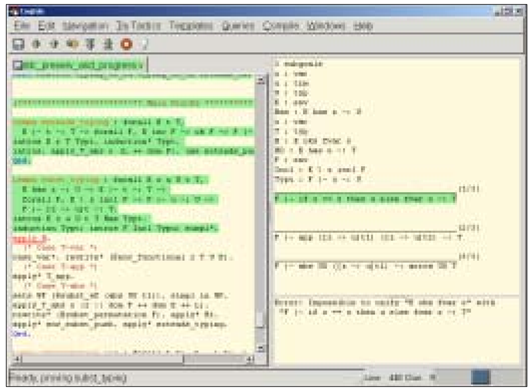
Engineering Formal Metatheory

Arthur Charguéraud

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

Motivation

A metatheory paper proof		**A metatheory mechanized proof**
	Mechanize	

– many tedious cases	→	use automation
– never 100% confident	→	machine-checked
– hard to reuse	→	re-run proof script

The POPLMark Challenge

How to formalize metatheory:

– with a generally applicable method,

– faithful to informal practice style,

– with 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) Locally nameless representation of syntax

2) Cofinite quantification of free variable names

1– Locally Nameless

Representation of Binders

Two basic approaches:

– first-order: represents variables "concretely"

– higher-order: encode object language binders into the function space of another language

Lot of work have been completed with both approaches.

The general perception is that first-order approaches require a lot more low-level work.

$\rightarrow$ Our goal: make this as light as possible.

First-Order Representations

- Names, α-quotiented
  - $\rightarrow$ quotient, α-conversion, capture
  - names without quotient $\rightarrow$ severe restrictions
  - nominal techniques $\rightarrow$ significant tool support
- De Bruijn indices $\rightarrow$ shifting of indices
- Distinguishing bound and free variables
  - locally named $\rightarrow$ α-conversion
  - locally nameless $\rightarrow$ our choice...

Locally Nameless Syntax

Representation:

– bound variables represented by de Bruijn indices

– free variables represented by names

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

Benefits:

– each λ-term has a unique representation

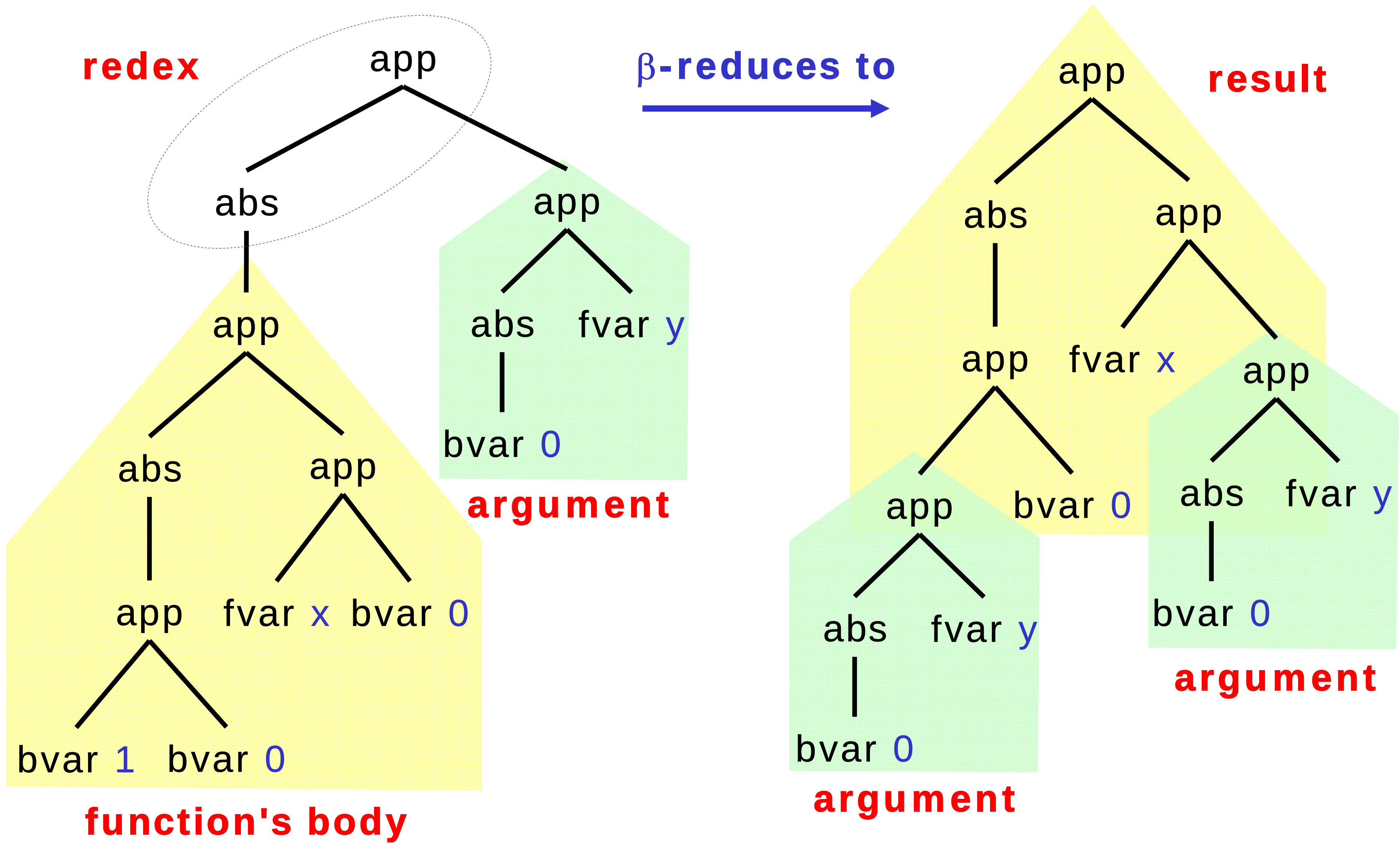
→ no quotient structure, no α-conversion

– straight-forward implementation of substitution

→ no shifting necessary, no variable capture

app (abs t) u --> t^u

β-reduction in Locally Nameless



This is a textual replacement: no renaming, no shifting.

Operations on Syntax

Operations on locally nameless terms:

– substitution for bound variables $t^u, \; t^x$

$\rightarrow$ to open up abstractions

– substitution for free variables $[x \rightarrow u]\, t$

$\rightarrow$ to reason about reductions

– computation of the set of free variables $\mathsf{FV}(t)$

$\rightarrow$ to state freshness properties

The definitions of these operations are simple, and it follows that their properties have simple proofs.

Restriction to Terms

<u>Problem:</u>

The locally nameless syntax contains objects that do not correspond to a lambda term, e.g. **`(bvar 3)`**.

<u>Solution:</u>

We define the predicate **`"term"`** to characterize objects in which all bound variables resolve to a binder.

$$\frac{}{\text{term (fvar } x)} \qquad \frac{\text{term } t_1 \qquad \text{term } t_2}{\text{term (app } t_1 \ t_2)} \qquad \frac{\text{term } (t^x)}{\text{term (abs } t)}$$

– Definitions $\rightarrow$ relations restricted to terms
– Infrastructure $\rightarrow$ operations compatible with term
– Core proofs $\rightarrow$ obligations handled by automation

2– Cofinite Quantification

How to Introduce Free Names?

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

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 strengthen if not	strong enough, 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

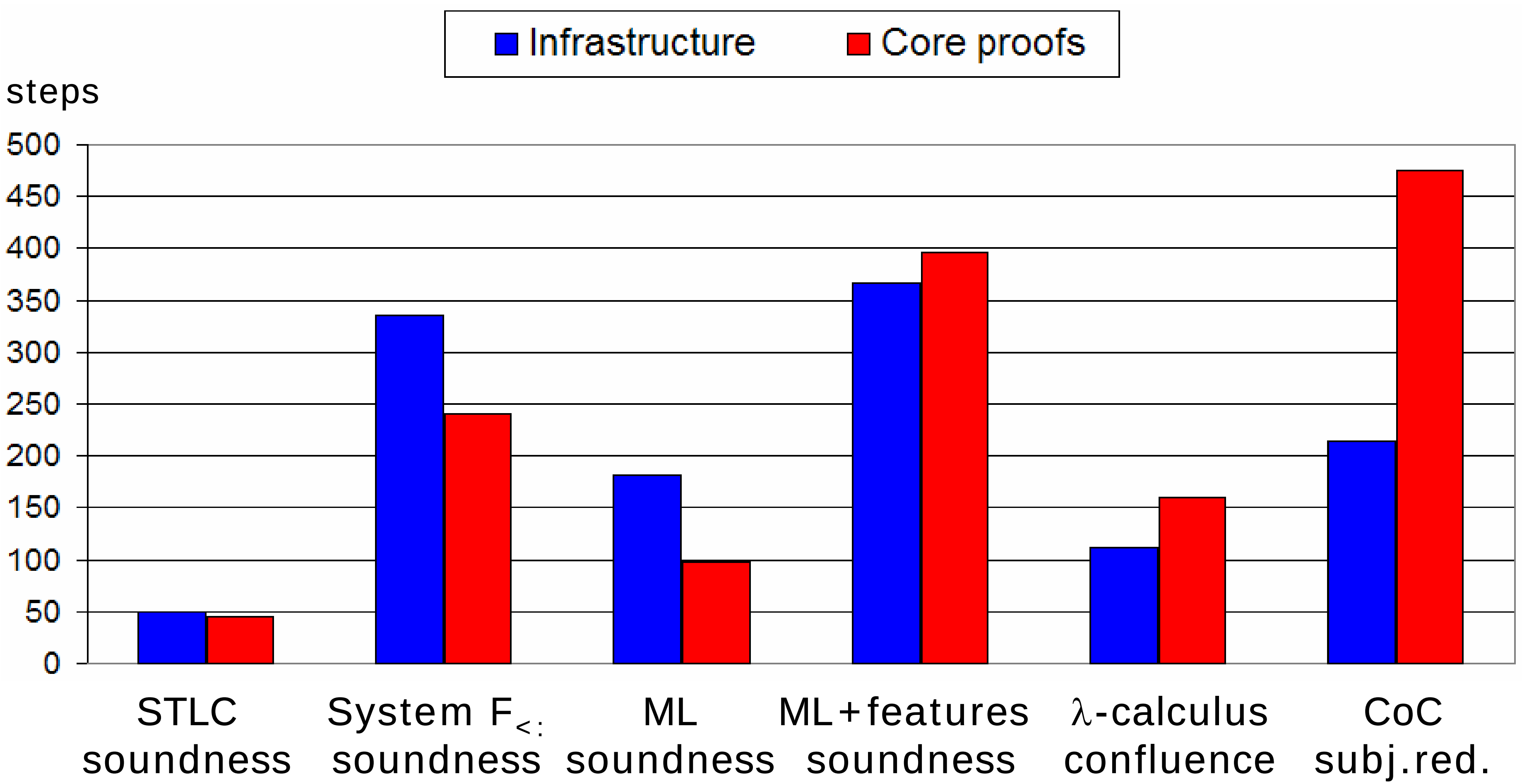
$$\frac{\forall x \notin L.\ \mathsf{term}\ (t^x)}{\mathsf{term}\ (\mathsf{abs}\ t)}$$

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 (in Coq)

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

Developments Completed



A step is defined as the application of a non-trivial tactic
(i.e. not "intro" or "auto" or a simple variations of these two).

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,
– amount of infrastructure required is reasonable,
– support by the OTT tool is work in progress.

Give it a try!

Developments scripts: http://arthur.chargueraud.org
Tutorial material: http://plclub.org/popl08-tutorial