A POPLmark retrospective

Using proof assistants in programming language research

Stephanie Weirich
University of Pennsylvania

The POPLmark Challenge

- A set of challenge problems meant to demonstrate the effectiveness of proof assistants in programming language research
- Issued at TPHOLs 2005
- Brian Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich and Steve Zdancewic

Why?

- A little PL research history…
- Since early 90s, trend in programming language research towards syntactic methods

A SYNTACTIC APPROACH TO TYPE SOUNDNESS

Andrew K. Wright* Matthias Felleisen*

Department of Computer Science
Rice University
Houston, TX 77251-1892

June 18, 1992

Rice Technical Report TR91-160
To appear in: *Information and Computation*

Abstract

We present a new approach to proving type soundness for Hindley/Milner-style polymorphic type systems. The keys to our approach are (1) an adaptation of subject reduction theorems from combinatory logic to programming languages, and (2) the use of rewriting techniques for the specification of the language semantics. The approach easily extends from polymorphic functional languages to imperative languages that provide references, exceptions, continuations, and similar features. We illustrate the technique with a type soundness theorem for the core of STANDARD ML, which includes the first type soundness proof for polymorphic exceptions and continuations.

Define the syntax and type system of a simple language (STLC + unit)

$$
\begin{array}{lcl}
\tau & ::= & \\
 & | & \mathbf{Unit} \\
 & | & \tau_1 \rightarrow \tau_2 \\
\\
e & ::= & \\
 & | & \mathbf{unit} \\
 & | & x \\
 & | & \lambda x{:}\tau.e \\
 & | & e_1\ e_2 \\
\Gamma & ::= & \\
 & | & \cdot \\
 & | & \Gamma, x{:}\tau
\end{array}
$$

$$\frac{}{\Gamma \vdash \mathbf{unit} : \mathbf{Unit}}\ \textsc{Unit}$$

$$\frac{(x{:}\tau) \in \Gamma}{\Gamma \vdash x : \tau}\ \text{VAR}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x{:}\tau_1.e : \tau_1 \rightarrow \tau_2}\ \text{ABS}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \text{APP}$$

Now what is the semantics?

- Denotational semantics: maps programs to mathematical objects, such as functions
- Operational semantics: describes how programs rewrite to values

$$v ::= \mid \textbf{unit} \mid \lambda x{:}\tau.e$$

$$\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\ \text{APP\_1}$$

$$\frac{e_2 \mapsto e_2'}{v_1\ e_2 \mapsto v_1\ e_2'}\ \text{APP\_2}$$

$$\frac{}{(\lambda x{:}\tau.e_1)\ v_2 \mapsto \{v_2/x\}\ e_1}\ \text{BETA}$$

Type soundness

'Well-typed programs don't go wrong':

Theorem 1 (Type Soundness) *If* $\cdot \vdash e : \tau$ *and* e *terminates, then there is some* v *such that* $e \mapsto^* v$.

i.e. all evaluation either diverges or produces a valid final configuration.

Syntactic Soundness Proof

Lemma 2 (Preservation) *If* $\cdot \vdash e : \tau$ *and* $e \mapsto e'$ *then* $\cdot \vdash e' : \tau$.

Lemma 3 (Substitution) *If* $\Gamma, x{:}\tau' \vdash e : \tau$ *and* $\Gamma \vdash e' : \tau'$ *then* $\Gamma \vdash \{e'/x\}e : \tau$

Lemma 4 (Progress) *If* $\cdot \vdash e : \tau$ *and* e *is not a value, then there exists an* e' *such that* $e \mapsto e'$.

Lemma 5 (Canonical Forms)

1. *If* $\cdot \vdash v : \mathbf{Unit}$ *then* v *must be* **unit.**
2. *If* $\cdot \vdash v : \tau_1 \to \tau_2$ *then* v *must be* $\lambda x{:}\tau_1.e$.

All of these lemmas proved by simple techniques (induction or inversion).

Why this technique?

- Low demands on the semanticists
  - Requires an 'operational' view of program execution
  - Easy to define because it resembles how the machine actually executes
  - Requires little mathematical machinery
    - "Just" inductive datatypes, alpha-conversion
- Proofs (of easy results) are easy
  - Series of straightforward inductions
  - Same form of lemmas each time
  - Cleverness is in setting up the type system the right way so that the usual properties work out

Covers many language features

references

transactional memory

concurrency

nontermination

I/O

exceptions

polymorphism

continuations

higher-order functions

dependent types

aspects

objects

A Syntactic Approach to Type Soundness

Andrew K. Wright* Matthias Felleisen*

Department of Computer Science
Rice University
Houston, TX 77251-1892

June 18, 1992

Rice Technical Report TR91-160
To appear in: *Information and Computation*

Abstract

We present a new approach to proving type soundness for Hindley/Milner-style polymorphic type systems. The keys to our approach are (1) an adaptation of subject reduction theorems from combinatory logic to programming languages, and (2) the use of rewriting techniques for the specification of the language semantics. The approach easily extends from polymorphic functional languages to imperative languages that provide references, exceptions, continuations, and similar features. We illustrate the technique with a type soundness theorem for the core of Standard ML, which includes the first type soundness proof for polymorphic exceptions and continuations.

What is wrong this method?

- Although the math is simple, there can be many cases
- Syntactic methods mean intuition can fail
- Easy to get something wrong in the details, especially in the combination of features

To: sml-list@cs.cmu.edu
From: Harper and Lillibridge
Sent: 08 Jul 91
Subject: Subject: ML with callcc is unsound

The Standard ML of New Jersey implementation of callcc is not type safe, as the following counterexample illustrates:…

The counterexample does contradict a claim by Felleisen and Wright to the effect that the type system is sound; it is my understanding that they have repaired the proof by restricting the language.

In good company

```
To: Types List
From: Alan Jeffrey
Sent: 17 Dec 2001
Subject: Generic Java type inference is
  unsound

The core of the type checking system was
shown to be safe… but the type inference
system for generic method calls was not
subjected to formal proof. In fact, it is
unsound … This problem has been verified
by the JSR14 committee, who are working
on a revised language specification…
```

Again and again

From: Xavier Leroy
Sent: 30 Jul 2002
To: John Prevost
Cc: Caml-list
Subject: Re: [Caml-list] Serious typechecking error involving new polymorphism (crash)
…
Yes, this is a serious bug with polymorphic methods and fields. Expect a 3.06 release as soon as it is fixed.
…

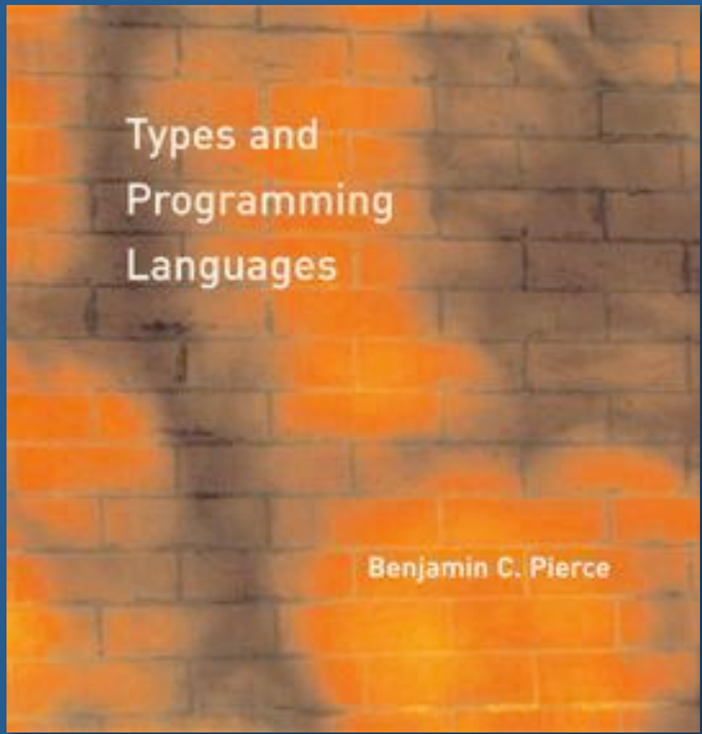
It happens to the best of us…

```
From: Dimitrios Vytiniotis
Subject: very serious bug in one lemma for
completeness ...
Date: 21 April 2005
To: Stephanie Weirich
Cc: Simon Peyton Jones
As I was typing up the proofs I discovered
that the strengthening lemma I have is not
correct ... this might affect the whole
paper ...Stephanie can we meet if you are
around? (otherwise tomorrow ...) :-( ...

-d
```

Syntactic methods continue to be popular

- Foundation for programming language study
- But it *can* be too much of a good thing...



The State of the Art

Chen and Tarditi,
A Simple Typed Intermediate Language for Object-Oriented Languages,
Principles of Programming Languages (POPL), 2005

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* $P \mapsto P'$, *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* $P \mapsto P'$.

Proof sketch: by standard induction over the typing rules.

3. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : \forall tvs(\tau_1,\ldots,\tau_n) \to \tau$ $(tvs = \alpha_1 \ll u_1,\ldots,\alpha_m \ll u_m)$ *and* $\Theta \vdash H : \Sigma$, *then* v *is a label and* $H(v) = \text{fix } g\langle tvs'\rangle(x_1 : \tau_1',\ldots,x_n : \tau_n') : \tau' = e_m$ *and* $tvs' = \alpha_1 \ll u_1',\ldots,\alpha_m \ll u_m'$.

4. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : \exists\alpha \ll \tau_u.\ \tau$, *then* $v = \text{pack } \tau_0 \text{ as } \alpha \ll \tau_u' \text{ in } (v' : \tau')$.

5. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : Tag(\tau)$, *then* $v = tag(C)$ *for some* C.

6. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : Tag(C)$, *then* $v = tag(C)$.

7. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : C$, *then* $v = C(v')$ *for some value* v'.

8. *If* $\Theta;\bullet;\Sigma;\Gamma \vdash v : int$, *then* v *is an integer.*

Proof: by inspection on expression typing rules, heap value rules and subtyping inversion Lemma 41. □

heorem 57 *If* $freetvs(\Sigma) = \emptyset$, *and* $\Theta \vdash H : \Sigma$, *and* $\Theta;\Sigma \vdash V : \Gamma$ *and* $\Theta;\bullet;\Sigma;\Gamma \vdash E : T$, *then either* E *is value, or* E *can evaluate one step, that is,* $\exists H', V'$ *and* E' *such that* $(H;V;E) \mapsto (H';V';E')$.

Proof: by induction on expression typing rules.

Case int, **Case label**, **Case tag**: all the expressions are values already.

Case var: $E = x$

By $\Theta;\Sigma \vdash V : \Gamma$, domain($\Gamma$) = domain($V$). From $x \in$ domain(Γ), we know $x \in$ domain(V). Let $H' = H$, $= V$ and $E' = V(x)$. By **ev_var** $(H;V;E) \mapsto (H';V';E')$.

Case error: $E = \text{error}[\tau]$: by **ev_error**, E steps to itself.

Case object: $E = C(e)$ with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : R(C)$.

By induction hypothesis, either e is a value or $\exists H'$, V' and e' such that $(H;V;e) \mapsto (H';V';e')$. If e is value, then E is a value. Otherwise, let $E' = C(e')$. By the congruence rule, $(H;V;E) \mapsto (H';V';E')$.

Case c2r_c: $E = \text{c2r}(e)$ with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : C$.

By induction hypothesis, either e is a value or $\exists H'$, V' and e' such that $(H;V;e) \mapsto (H';V';e')$. If e a value, then by Lemma 56 $e = C(v)$. Let $E' = v$. By **ev_c2r** $(H;V;E) \mapsto (H';V';E')$. Otherwise let $= \text{c2r}(e')$. By the congruence rule $(H;V;E) \mapsto (H';V';E')$.

Case c2r_tv: not applicable because by Lemma 17, the subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : \alpha$ is invalid.

Case record: $E = \text{new}[\tau]\{l_1 = e_1,\ldots,l_n = e_n\}$ with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_i : \tau_i\ \forall 1 \le i \le n$.

By induction hypothesis, each subexpression e_i either is a value or can evaluate one more step.

If all e_i are values, then let $H' = H, \ell \rightsquigarrow \{l_1 = e_1,\ldots,l_n = e_n\}$ (ℓ is a fresh label), $V' = V$ and $E' = \ell$. **ev_record** $(H;V;E) \mapsto (H';V';E')$.

If $\exists e_i$ such that $e_1,\ldots,e_{i-1}$ are values and $\exists H',V',e_i'$ such that $(H;V;e_i) \mapsto (H';V';e_i')$ $w[\tau]\{l_1 = e_1,\ldots,l_i = e_i',\ldots,l_n = e_n\}$. By the congruence rule, $(H;V;E) \mapsto (H';V';E')$.

Case field: $E = e.l_i$ with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : \{l_1^{\phi_1} : \tau_1,\ldots,l_n^{\phi_n} : \tau_n\}$ and $1 \le i \le n$.

By induction hypothesis, either e is a value or e can evaluate one step.

If e is a value, by canonical form Lemma 56, e is a label and $H(e) = \{l_1 = v_1,\ldots,l_n =$ $= H$, $V' = V$ and $E' = v_i$. By **ev_field** $(H;V;E) \mapsto (H';V';E')$.

If $\exists H',V',e'$ such that $(H;V;e) \mapsto (H';V';e')$, then let $E' = e'.l_i$ and by the congruence rule $';V';E')$.

In the rest cases, we state only which evaluation rule to apply, but omit the new H', V' or

Case assignR $E = e_1.l_i := e_2$ in e_3 with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \{l_1^{\phi_1} : \tau_1,\ldots,l_i^M :$ $\}$ and $\Theta;\bullet;\Sigma;\Gamma \vdash e_2 : \tau_i$.

By induction hypothesis, either e_1 is a value or e_1 can evaluate one step. Similarly, either e_2 can evaluate one step.

If both e_1 and e_2 are values, then by canonical form Lemma 56 e_1 is a label and $H(e_1) = \{l_1 = v_1,\ldots,l_n =$ $\ldots\}$, and by **ev_assignR** E can evaluate one step.

If either e_1 or e_2 can evaluate one step, by the congruence rule, E can evaluate one step.

Case array $E = \text{new}[e_0,\ldots,e_{n-1}]^\tau$ with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_i : \tau\ \forall 0 \le i \le n-1$

Lemmas 13 and 14 $\Theta;\bullet;\Sigma';\Gamma' \vdash e_j : \tau_j\ \forall 1 \le j \le n$. By

values, $H' = H, \ell \rightsquigarrow \{l_1 = e_1,\ldots,l_n = e_n\}$ where ℓ is a fresh

$\Gamma' = \Gamma$. Each e_i ($\forall 1 \le i \le n$) is a value with no free variables.

If rule **ev_call** applies, then (1) e is a label and $H(e) = \text{fix } g < tvs' > (x_1 : s_1,\ldots,x_n : s_n) : s = e_0$ where $tvs' = \alpha_1 \ll u_1',\ldots,\alpha_m \ll u_m'$, (2) all $e_1,\ldots,e_n$ are values, (3) $H' = H$, (4) $V' = V, g = e, x_1 = e_1,\ldots,x_n = e_n$ and (5) $E' = e_0[\sigma]$. By inversion of value Lemma 52, (1) let $\tau_f' = \forall\alpha_1 \ll u_1',\ldots,\alpha_m \ll u_m'.(s_1,\ldots,s_n) \to s$, then $\Sigma(e) = \tau_f'$ and $\Theta;\bullet \vdash \tau_f' \le \tau_f$ and (2) $\Theta;tvs';\Sigma;g : \tau_f', x_1 : s_1,\ldots,x_n : s_n \vdash e_0 : s$. By subtyping inversion Lemma 41, (1) $\forall 1 \le i \le n$, $\Theta;\alpha_1 \ll \text{Topc},\ldots,\alpha_m \ll \text{Topc} \vdash \tau_i \le s_i$, (2) $\Theta;\alpha_1 \ll \text{Topc},\ldots,\alpha_m \ll \text{Topc} \vdash s \le \tau$, and (3) $\forall 1 \le i \le m$, $\Theta;\bullet \vdash t_i \ll u_i'[\sigma]$. By type substitution Lemma 51, $\Theta;\bullet;\Sigma;g : ty_f'[\sigma], x_1 : s_1[\sigma],\ldots,s_n[\sigma] \vdash E' : s[\sigma]$. Σ has no free type variables, therefore, τ_f' has no free type variables and $\tau_f'[\sigma] = \tau_f'$. By the property that type substitution preserves subtyping

$\Delta;\Sigma;\Gamma \vdash e_i : \tau_{mi}$, $\forall 0 \le i \le n-1$. By **sub** and $\Theta;\Delta \vdash \tau_{mi} \le \tau$, $\Theta;\Delta;\Sigma;\Gamma \vdash E : T$.

$T = \tau$ with subderivations $\Theta;\Delta;\Sigma;\Gamma \vDash e_1 : \text{array}(\tau)$ and $\Theta;$

By induction hypothesis, each subexpression e_i either is a value or can evaluate one step. If are values, by **ev_array** E can evaluate one step. Otherwise, $\exists e_i$ such that e_i can evaluat $e_0,\ldots,e_{i-1}$ are all values. By the congruence rule, E can evaluate one step.

Case subscript $E = e_1[e_2]$ with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \text{array}(\tau)$ and $\Theta;\bullet;\Sigma;\Gamma \vdash$

By induction hypothesis, either e_1 is a value or e_1 can evaluate one step. So does e_2.

If e_1 and e_2 are both values, by canonical form Lemma 56, e_1 is a label and $H(v) = [v_0,$ e_2 is an integer. The runtime array bounds check guarantees that the index e_2 is within $0 \le e_2 \le n-1$. By **ev_sub** E can evaluate one step.

If e_1 or e_2 can evaluate one step, by the congruence rule E can evaluate one step.

Case assignA $E = e_1[e_2] := e_3$ in e_4 with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \text{array}(\tau)$, $\Theta;\bullet$ and $\Theta;\bullet;\Sigma;\Gamma \vdash e_3 : \tau$.

By induction hypothesis, either e_1 is a value or e_1 can evaluate one step. So do e_2 and e_3

If all e_1, e_2 and e_3 are values, by canonical form Lemma 56, e_1 is a label and $H(v) =$ guarantees that the index e_2 is within ep.

ruence rule E can evaluate one step.

If e_{t1} or e_{t2} can evaluate one step, then by the congruence rule E can evaluate one step.

Case ifTag_tv: $E = \text{ifEqTag}^\tau(e_{t1}, e_{t2})$ then e_1 else e_2 with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e_{t1} : \text{Tag}(\gamma)$. Not applicable because the subderivation is invalid by Lemma 17.

Case sub: $E = e$ with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : \tau_1$.

By induction hypothesis, either e is a value or e can evaluate one step. That is, either E is a value or E can evaluate one step. □

Corollary 58 (progress) *If* $\Sigma \vdash P : \tau$, *then either the main expression in* P *is a value, or* $\exists P'$ *such that* $P \mapsto P'$.

Theorem 59 *LIL$_C$ is Sound. Well-typed LIL$_C$ programs do not get stuck.*

Proof: by progress and preservation. □

$\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \tau$.

an evaluate one step. If e_1 is a value,

rule, E can evaluate one step.

$\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \Gamma(x)$.

an evaluate one step. By $\Theta;\Sigma \vdash V : \Gamma$

is a value, by **ev_assign** E can eva step.

lerivations $\Theta;\bullet;\Sigma;\Gamma \vdash e : \forall tvs(\tau_1,\ldots$

E.3 Pro

Lemma 56

$\Theta;\bullet;\Sigma;\Gamma \vdash e_i : \tau_i[\sigma]\ \forall 1 \le i \le n$, where $\sigma = t_1,\ldots,t_m/tvs$.

by induction hypothesis, either e is a value or e can evaluate one step. So does each $\forall 1 \le i \le n$.

If all e and $e_1,\ldots,e_n$ are values, then by canonical form Lemma 56, e is a label and $H(e) =$ $\tau_1',\ldots,x_n : \tau_n') : \tau' = e_m$. By **ev_call**, E can evaluate one step.

If e or any of e_i can evaluate one step, by the congruence rule E can evaluate one step.

Case pack: $E = \text{pack } \tau \text{ as } \alpha \ll \tau_u$ in $(e : \tau')$ with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e : \tau'[\tau/\alpha]$.

By induction hypothesis, either e is a value or e can evaluate one step. If e is a value, the

If e can evaluate one step, then by the congruence rule E can evaluate one step.

Case open: $E = (\alpha, x) = \text{open}(e_1)$ in e_2 with subderivation $\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \exists\beta \ll \tau_u.\ \tau$.

By induction hypothesis, either e_1 is a value or e_1 can evaluate one step. If e_1 is a value, the form Lemma 56, $e_1 = \text{pack } \tau_0 \text{ as } \beta \ll \tau_u'$ in $(v : \tau')$. By **ev_open** E can evaluate one step. If one step, then by the congruence rule E can evaluate one step.

Case ifParent: $E = \text{ifParent}(e)$ then bind (α, x) in e_1 else e_2 with subderivation $\Theta;\bullet;\Sigma$

$\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \tau$, we have $\Theta;\Sigma \vdash V' : \Gamma'$.

Case assign $E = x := e_1$ in e_2, $T = \tau$ with subderivations $\Theta;\bullet;\Sigma;\Gamma \vdash e_2 : \tau$.

If **ev_assign** applies, then (1) $V = V_1, x = v, V_2$, (2) $H' = H$

Let $\Sigma' = \Sigma$ and $\Gamma' = \Gamma$. By $\Theta;\bullet;\Sigma;\Gamma \vdash e_1 : \Gamma(x)$, we have $\Theta;\Sigma$ $\Theta;\bullet;\Sigma';\Gamma' \vdash E' : T$.

Case call: $E = e[t_1,\ldots,t_m](e_1,\ldots,e_n)$, $T = \tau[\sigma]$ with sub $\forall tvs(\tau_1,\ldots,\tau_n) \to \tau$ and $tvs = \alpha_1 \ll u_1,\ldots,\alpha_m \ll u_m$, $\Theta;\bullet$ $t_1,\ldots,t_m/\alpha_1,\ldots,\alpha_m$ and $\forall 1 \le i \le m$, $\Theta;\bullet \vdash t_i \ll u_i[\sigma]$.

Lemma 27 premises. Also t

Theorem 48

E Sound

Personal Experience

Publication	TR length	Heroic grad student
JFP 07	83 pages	Dimitrios
ICFP 06	59 pages	Dimitrios
ICFP 06	58 pages	Dimitrios
ICFP 05	60 pages	Geoff, Dan
LICS 05	60 pages	Geoff
TLDI 04	51 pages	Geoff, Dimitrios
WOOD 04	49 pages	Liang
ICFP 03	61 pages	Geoff

Why write-only TRs?

Proofs optimized for conveying understanding

vs.

Proofs optimized for conveying certainty

i.e. we believe this is true because we actually worked out the details. And you can check our details if you have the patience

Who has more patience than a machine?

Existing research community of logics for expressing such proofs and tools for checking them

Some were already doing this…

- 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 no common knowledge

- What proof assistant to use?
- How to get started? Manuals? Tutorials?
- Libraries?
- Existing developments?

The POPLmark challenge was a community and infrastructure building project

THE CHALLENGE, SPECIFICALLY

Metatheory of System F-sub

Challenge 1: Transitivity of subtyping

$$\textit{If}\ \Gamma \vdash S \leq Q\ \textit{and}\ \Gamma \vdash Q \leq T,\ \textit{then}\ \Gamma \vdash S \leq T.$$

- Transitivity must be proven simultaneously with narrowing, which states:

$$\textit{If}\ \Gamma, X \leq Q, \Gamma' \vdash S \leq T\ \textit{and}\ \Gamma \vdash P \leq Q,$$
$$\textit{then}\ \Gamma, X \leq P, \Gamma' \vdash S \leq T.$$

- What's tested here: Non-trivial inductive proofs, isolating elements of the context

Challenge 2: Type safety

1. *If* $\Gamma \vdash e : \mathrm{T}$ *and* $\mathrm{e} \longrightarrow e'$, *then* $\Gamma \vdash e' : \mathrm{T}$.

2. *If* $\Gamma \vdash e : \mathrm{T}$, *then either* e *is a value or else* $e \longrightarrow e'$ *for some* e'.

- Extended language with records and pattern matching
- What's tested here: Reasoning about syntax with variable numbers of components
  - Record patterns may bind arbitrarily many variables
  - Record values may contain an arbitrary number of fields

Challenge 3: "Animation"

1. *Given e and e', decide if* $e \to e'$.
2. *Given e and e', decide if* $e \to^* e' \not\to$.
3. *Given e, find e' such that* $e \to e'$.

- What's tested here: the ability to explore a language's properties on particular examples
- Solutions for (1) and (2) can check an interpreter
- Solution for (3) is an interpreter

Evaluation criteria

- Readers:
  - Adequacy of the encoding: Is it correct?
  - Obviousness of the encoding: How difficult is it to understand adequacy?
- Writers:
  - Clutter, inconvenience introduced by the technology
  - Effort required beyond a paper proof, even for experts
- Cost of entry:
  - Quality of documentation
  - Maturity of technology

What happened next?

POPLmark results

- Lots of interest!
- 15 submitted solutions recorded on wiki
  - 7 tools used (Coq, Isabelle/HOL, Twelf, ATS, Matita, Abella, Alpha-Prolog)
- Other solutions discussed elsewhere (ACL2, MetaPRL, Nominal-Isabelle)

"POPLmark tarpit"

- Techniques for representing variable binding caused the most heated discussion
  - 7 different techniques used in 15 solutions
  - Hit a pre-existing, active research area
- Our own efforts to understand this issue resulted in new research results
  - *Engineering Formal Metatheory,* POPL 08
    Aydemir, Chargueraud, Pierce, Pollack, Weirich
- Other parts of the challenge relatively ignored
  - Many did not complete full challenge with records or animation

Community development

- We worked hard to promote the use of proof assistants among PL researchers…
  - Organized workshops (4 instances of WMM so far)
  - Developed tutorial material
  - Developed a library for PL reasoning
  - Distributed all of our own developments
  - Integrated proof assistant use into our graduate PL course

Had to pick something…

- Devoted our efforts to Coq Proof Assistant
  - Wanted a general purpose logic
  - Wanted a mature platform
  - Constructive logic, dependent types were attractive

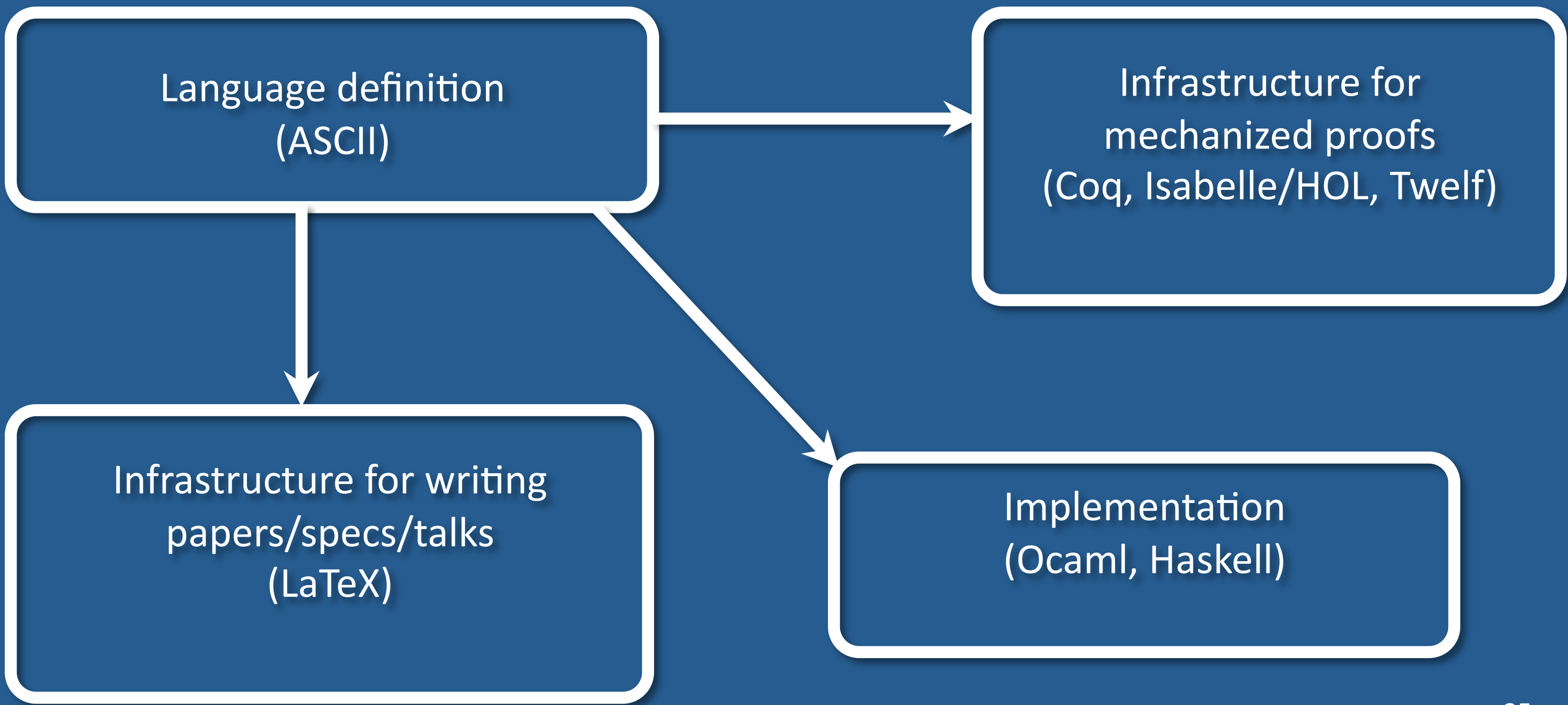
- Could have chosen others with equal success
  - Exciting new developments in the meantime: Nominal-Isabelle, Abella, etc.

It started to work…

- More papers with machine checked appendices start appearing
  - Some bootstrapped from our own work
- AURA – Zdancewic et al. ICFP 2007
  - Language for reasoning about authorization
  - Security-orientation motivates more certainty
  - Sophisticated dependent type system
  - Metatheory completely developed in Coq
  - 12.4k LOC

New tool - Ott: Sewell et al.

A tool should generate many outputs given
a single "naturally written" definition of a language



Example: lambda terms

```
metavar atom, x, y, z ::= {{ coq nat }}{{ coq-equality }}

exp, e, f, g   :: '' ::=
| x            ::    :: var
| \ x . e      ::    :: abs (+ bind x in e +)
| e1 e2        ::    :: app
| { e / x } e' :: M :: subst {{ coq subst [[e]] [[x]] [[e']] }}

substitutions
  single e x :: subst
```

```
defn
 e1 --> e2 :: :: reduce ::''  by

    --------------------------  :: ax_app
    (\x.e1) e2 -->  {e2/x}e1

    e1 --> e1'
    -------------- :: ctx_app_fun
    e1 e --> e1’ e
```

Ott code

Example: Typed lambda terms

```
Definition atom := nat.

Inductive exp : Set :=
| var : atom -> exp
| abs : atom -> typ -> exp -> exp
| app : exp -> exp -> exp.
```

Coq code
output by OTT

```
Inductive reduce : exp -> exp -> Prop :=    (* defn reduce *)

 | ax_app : forall (x:atom) (e12 e2:exp),
     reduce (app (abs x e12)  e2) (subst e2 x e12 )

 | ctx_app_fun : forall (e1 e_5 e1’:exp),
     reduce e1 e1' ->
     reduce (app e1 e_5) (app e1’ e_5).
```

Substitution output

```
Lemma eq_atom: forall (x y : atom), {x = y} + {x <> y}.
…
Fixpoint list_mem (A:Set) (eq:forall a b:A,{a=b}+{a<>b})
     (x:A) (l:list A) {struct l} : bool :=
…
```

```
Fixpoint subst (e_6:exp) (x5:atom) (e__7:exp) {struct e__7}
: exp :=
  match e__7 with
  | (var x) => (if eq_atom x x5 then e_6 else (var x))
  | (abs x e5) => abs x (if list_mem eq_atom x5 (cons x nil)
                            then e5 else (subst e_6 x5 e5))
  | (app e5 t') => app (subst e_6 x5 e5) (subst e_6 x5 e')
end.
```

Coq code
output by OTT

How did the POPLmark challenge impact my research?

My research methods have changed

- I use OTT for all of my type setting
  - including parts of this talk
  - especially exploratory, development work
- I find formalizing the definitions in a paper often helps my understanding
- I sometimes pop open a Coq window to try out some thoughts
- Collaboration is easier this way
  - Version control
  - Definitions, proof status always up-to-date
- New research on variable binding

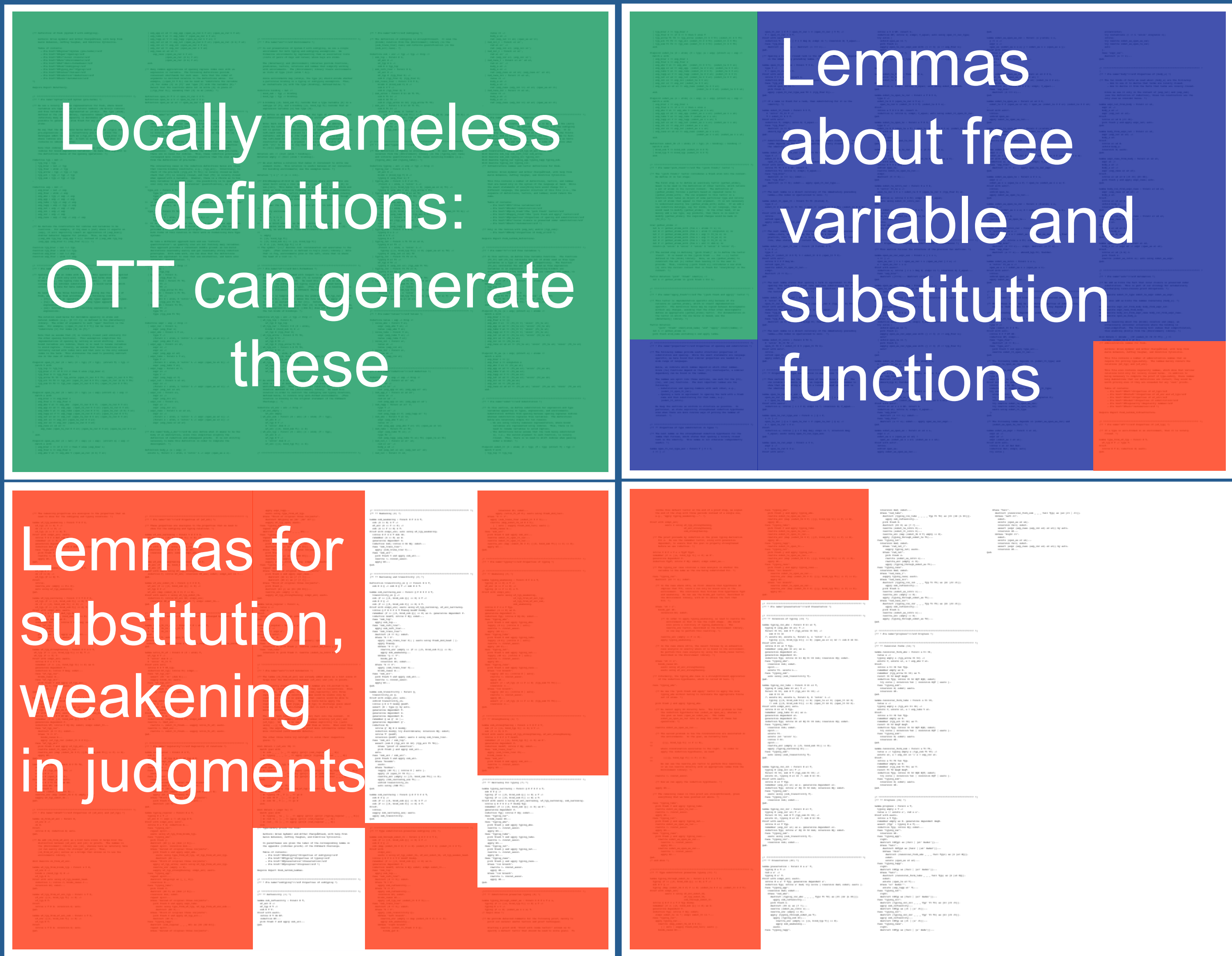
The issue with variable-binding

- Bound variables must alpha-vary
  - Identify \x.x and \y.y
- Free variables must be 'sufficiently fresh'
  - Capture-avoiding substitution e { e' / x } --- bound variables in e must not be the same as the free variables in e'
  - "Barandregt Variable Convention"

Locally nameless rep

- POPL 08 paper advocated two ideas for variable binding
- Locally nameless representation (old idea)
  - Separate bound and free variables
  - Use numbers for bound variables (unique representation of alpha-equivalent terms) and strings for free variables
- Cofinite quantification (new idea)
  - Premise of judgments quantifies over all variables except for some finite set
  - Strong induction principle

POPLmark challenge in Coq



Other experiences

- Rossberg, Russo, Dreyer. *F-ing modules*. TLDI 2010
- 13k line Coq development
- Used locally nameless approach
- 400 out of 550 lemmas were tedious "infrastructure" results

LNgen – Work in Progress

- Brian Aydemir and Stephanie Weirich. *LNgen: Tool Support for Locally Nameless representations.*
- Works with OTT tool
- Generates and proves 'infrastructure' lemmas based on locally nameless representation
- Example lemma: if fv(t) = 0 then [x |-> u]t = t

Example: STLC development

- Ott (locally nameless backend) – 134 lines
  - 5 inductive definitions (typ, exp, lc, typing, step)
  - 3 functions (open, fv, subst)
  - 1 tactic (to collect all free vars in a proof)
- Lngen – 1533 lines
  - 3 functions (close, size_typ, size_exp)
  - 2 inductive definitions (degree, lc_set)
  - 47 lemmas
  - 2 tactics, 90 Hints
- Hand proofs – 108 lines
  - 8 lemmas (4 adeqacy, weakening, subst, preservation, progress)

What are those 47 lemmas

Why proof generation is ok

- Code generators (rightly so) have a bad name
- Why is this a reasonable way to do things?
- Proof-irrelevance: don't care how a lemma was proved, only that it was proved
- lots of regular structure
  - F-omega: substitute types in terms, terms in terms, types in types
- Clear scope: Reasoning restricted to 5 operations
  - open, close, subst, fv, lc
  - lemmas concern only these operations and their interactions with eachother

Case studies

- LNgen provided infrastructure for two POPL 2010 papers
- Greenberg, Weirich, Pierce. *Contracts Made Manifest*
  - Most proofs by hand (60 page TR)
  - Tricky reasoning about parallel reduction done in Coq. Replaced 8 dense pages of TR appendix
- Jia, Zhao, Sjöberg, Weirich. *Dependent Types and Program Equivalence*
  - Varied language for 9 months, doing proofs by hand
  - Used LNgen to check results in about 2 weeks

Contracts

438	terms.v	Generated by OTT
3965	infrastructure.v	Generated by LNgen
764	prelim.v	
3090	thy.v	
8257	total	

Dependent types

991	lang.v	Generated by OTT
267	langExtra.v	
7638	infrastructure.v	Generated by LNgen
169	isEq.v	
6116	thy.v	
2126	thyPP.v	
290	progress.v	
862	reductions.v	
61	isEqSpecification.v	
691	isEqBeta.v	
2284	isEqC.v	
97	inclusions.v	
21592	total	

Proofs instead of TRs (mostly)

Venue	Mech.	How	Tech report	Heroic students
POPL 10	some	Coq	60 pages	Michael
POPL 10	yes	Coq		Limin (post-doc), Jianzhou, Vilhelm
PLPV 10	yes	Agda		Chris
CCS 09	yes	Coq		Aaron, Vilhelm
ICFP 08	no		Dissertation	Dimitrios
POPL 08	yes	Coq		Arthur, Brian
MFPS 07	yes	Isabelle/HOL		Dimitrios

Where to from here? What next for PL community?

Active research into variable binding

- Just in Cambridge:
  - Pitts – Nominal System T [POPL 2010]
  - Urban – Nominal Isabelle
  - Kennedy, Benton – Strongly typed Coq
- I don't think we have the complete story yet

Proof engineering

- Proof engineering
  - How to make sure that proofs are maintainable?
  - Haven't tactical theorem provers failed before?

- I don't know the answer to this problem

Role in Education

- Pierce: new textbook using Coq for grad students at Penn
- Excellent tool for teaching about proofs by induction, syntactic approach to programming language definitions, etc.
- What about discrete math?

Language definition

- What do we need to do to make sure that it is standard practice to have a machine-checked language specification?
- Again, heroic efforts exist…
  - SML, OCAML (light), Java (light)
- … but consensus is necessary
  - Language designers want accessible specs

Goes for logics too…

```
From: Hugo Herbelin
Sent: November 2, 2009
To: Coq club

> Hi, I have been looking on the web without
> success. Is there any paper/tech report
> that gives the precise rules of the pCIC as
> it is currently implemented in Coq 8.2.
> (something like a latex version of Chapter
> 4 from the reference manual)

There is a latex version of the reference manual in the
Coq source archive and a pdf version at
http://coq.inria.fr/distrib/V8.2pl1/files/.

AFAIK there is no other description on paper of the entire
set of features of pCIC in its 8.2 implementation. Note
however that there is a work in progress by Gyesik Lee and
Benjamin Werner on the set-theoretical model of a
formulation of pCIC that is very close to Coq.
```

CONCLUSION

Conclusions

- I plan to keep on using proof assistants in my day to day research

The Success of Typed Languages

- It is difficult for programmers to prove properties about individual programs
- Instead, language designers prove properties about languages that imply properties of all programs in that language

- Example: A scheme programmer must prove that his program never executes (1 + true)
- An ML programmer knows this already.

Fundamental idea: Type safety

- Milner – Well typed programs don't go wrong
- i.e. programs maintain certain invariants during their execution
- those invariants are described by the type system
  - Functions called with particular forms of arguments

How to prove type safety?

- Since the early 90s, type safety proved 'syntactically'
- Two key lemmas:
  - Preservation: If a program type checks, and it takes a step, it will still type check
  - Progress: If a program type checks and it is not in an (approved) terminal configuration then it can take a step

Current state of the art: Ott

- Input: Language definitions in ASCII
  - Syntax (BNF grammar)
  - Binding specifications
  - Relations (Typing judgments, operational semantics)

- Output: multiple tool definitions
  - LaTeX: Typesetting macros
  - Proof assistants: Inductive datatypes; functions for free variables and substitution

- http://www.cl.cam.ac.uk/~pes20/ott/

What did we do?

- Compared submitted solutions with our own explorations:
  - FJ in Coq / Twelf / Isabelle/HOL
  - Parametricity theorem in Isabelle/HOL
  - Damas-Milner in Nominal-Isabelle
  - Created our own solutions to POPLmark challenge in Coq