Mechanizing Language
Definitions

Robert Harper
Carnegie Mellon University
June, 2005
Acknowledgements

This talk represents joint work with

- Michael Ashley-Rolman
- Karl Crary
- Frank Pfenning

Thanks to TTI-C/UC for the invitation!
Language Definitions

What does it mean for a programming language to exist?

The “standard” answer is exemplified by C.

- Informal description (à la K&R, say).
- A “reference” implementation (gcc, say).
- Social processes such as standardization committees.
Language Definitions

- The PL research community has developed better definitional methods.

  - Classically, various grammatical formalisms, denotational and axiomatic semantics.

  - Most successfully, type systems and operational semantics.

- Nearly all theoretical studies use these methods! (e.g., every other POPL paper)
Language Definitions

- What good is a language definition?
- Precise specification for programmers.
- Ensures compatibility among compilers.
- Admits rigorous analysis of properties.
- The Definition of Standard ML has proved hugely successful in these respects!
Language Definitions

- But a language definition is also a burden!
  - Someone has to maintain it.
  - Not easy to make changes.
- Definitions can be mistaken too!
  - Internally incoherent.
- Difficult or impossible to implement.
Language Definitions

- A definition alone is not enough! Must maintain a body of meta-theory as well.

- Type safety: coherence of static and dynamic semantics.

- Decidability of type checking, determinacy of execution, ....

- Developing and maintaining the meta-theory is onerous.
Mechanized Definitions

- Some of the burden can be alleviated through mechanization.
- Formulate the definition in a logical framework.
- Automatically or semi-automatically verify key meta-theoretic properties.
- But can this be done at scale?
Mechanized Definitions

Yes, using LF/Twelf!

- Formalize definition in LF.
- State meta-theorems relationally in LF.
- Use Twelf to prove “totality”.

Remarkably, this approach works well both “in the small” and “in the large”!
LF Methodology

- Establish a compositional bijection between
  - objects of each syntactic category of object language
  - canonical forms of associated types of the LF lambda calculus
- "Compositional" means "commutes with substitution" (aka "natural").
LF Methodology

Here the syntactic categories include:

- abstract syntax, usually including binding and scoping conventions
- typing derivations
- evaluation derivations

The latter two cases give rise to the slogan "judgements as types".
Example: STLC

% abstract syntax

tp : type.
b : tp.
arrow : tp -> tp -> tp.
tm : type.
lam : tp -> (tm -> tm) -> tm.
app : tm -> tm -> tm.
Example: STLC

% typing (excerpt)

of : tm -> tp -> type.

of_lam :
  ({x : tm}{dx : of x T} of (F x) U) ->
  of (lam T F) (arr T U)

of_app :
  of E1 (arr T U) -> of E2 T ->
  of (app E1 E2) U.
Example: STLC

% evaluation (excerpt)

step : tm -> tm -> type.

beta :
   step (app (lam T F) E) (F E).

fun :
   step E1 E1' -> step (app E1 E2) (app E1' E2).
## Adequacy Theorem

<table>
<thead>
<tr>
<th>Cat’y</th>
<th>Rep’n</th>
<th>Contexts/World</th>
</tr>
</thead>
<tbody>
<tr>
<td>T type</td>
<td>T : tp</td>
<td></td>
</tr>
<tr>
<td>E term</td>
<td>E : tm</td>
<td>x : tm</td>
</tr>
<tr>
<td>E : T</td>
<td>D : of E T</td>
<td>x : tm, dx : of x U</td>
</tr>
</tbody>
</table>
Meta-Reasoning

- Adequacy ensures that we can reason about the object language by analyzing canonical forms of appropriate LF type.
- Canonical forms are long $\beta\eta$ normal forms.
- Structural induction, parallel and lexicographic extension to tuples.
- Applies to informal and formal reasoning!
Meta-Reasoning in Twelf

Twelf supports checking of proofs of \( \Pi_2 \) (\( \forall \exists \)) propositions over canonical forms in a specified class of contexts (world).

- Enough for preservation, progress, ...

- These are totality assertions for a relation between inputs (\( \forall/+ \)) and outputs (\( \exists/- \))!

- Polarity notation is an unfortunate relic.
Relational Meta-Theory

Preservation Theorem as a relation:

\[ \text{pres} : \text{of } E \ T \rightarrow \text{step } E \ E' \rightarrow \text{of } E' \ T \rightarrow \text{type.} \]

Axiomatize this relation:

\[ \text{pres}\_\beta : \]
\[ \text{pres } (\text{of}\_\text{app } (\text{of}\_\text{lam } D) \ D') \]
\[ \text{beta } (D \_ D'). \]

etc.
Relational Meta-Theory

- Ask Twelf to verify the totality of the relation representing the theorem.
- Specify the worlds to consider.
- Specify mode of the relation.
- Specify induction principle to use.
- Checks that all cases are covered, and induction is used appropriately.
Relational Meta-Theory

- For preservation this consists of decl's
  
  \%mode pres +D1 +D2 -D3
  \%worlds () (pres _ _ _)
  \%total D (pres _ D _)

- Twelf performs a mode check, coverage check, and termination check.

- Errors are similar to ML match errors.
Relational Meta-Theory

- The worlds for preservation are empty.
- Consider only closed terms in this case.
- The mode specifies $\forall$ typing derivs $\forall$ steps $\exists$ typing deriv
- Totality specifies proof by induction on transition step.
Scaling Up

Well and good, but does it scale?

Yes, surprisingly well, but ...

Some language features are hard to handle in LF.

Some meta-theory is trickier than this.

But we use Twelf daily in our work at CMU!
Some Examples

- TALT, a full-scale certified object code format with a generic safety policy.
- Compilation through closure conversion, type safety for Classical S5 for dist’d prog’ing.
- First, and only, solution to the POPLmark Challenge to verify meta-theory of F<:.
- Type safety (almost), regularity for HS semantics of Standard ML.
Adding A Store

- Ideally, locations would be treated like variables.
  - Location typing consists of assumptions about types of locations.
  - Store contents consists of assumptions about the values of locations.

- But this requires linearity, which we do not currently have at our disposal.
Adding A Store

Manage stores explicitly as mappings from locations to types or values.

- Explicit lookup, update, extension.
- Unpleasant, technically, but unavoidable.

How to represent the typing judgment?

- Where does the location typing go?
Adding A Store

The “obvious” approach is to add a location typing to the typing judgement:

\[ \text{of : lt \to tm \to tp \to type.} \]

We suppress here the details of how the location typing is managed.

Trust me, they’re ugly.
Adding A Store

For what contexts is the encoding adequate? The “obvious” choice would seem to be

\[ x : \text{tm}, \, dx : \text{of L x T} \]

Typing rules change accordingly:

\[ \text{of}_\text{lam} : \]
\[ (\{ x : \text{tm} \}\{ dx : \text{of L x T} \} \text{ of L (F x) U} \rightarrow \text{of L (lam T F) (arrow T U)}. \]
Meta-Theory for Stores

Unfortunately, we cannot push through proofs of the required meta-theory!

Example: weakening of the location typing.

Extending the store with new locations preserves typing.

Required for type safety.
Meta-Theory for Stores

- Relational formulation of weakening:

  weaken :
  of L E T → ext L L' → of L' E T → type.

- Formalize a proof by induction on the first typing derivation.
%mode weaken +D1 +D2 -D3
%total D (weaken D _ _ )
Consider the case of a lambda:

\[
\text{weaken\_lambda :}
\]

\[
\text{weaken (of\_lam T D) X (of\_lam T D')} <- \\
\{ \ x : \text{tm} \}\{ \ dx : \text{of } L' \times T \}
\]

\[
(\text{weaken } (D \times dx) \times (D' \times dx)).
\]

But this clause is not type-correct!

\[
\text{D x : of } L \times T \rightarrow ..., \text{ but } dx : \text{of } L' \times T!
\]

There is no fcn of \(L' \times T \rightarrow \text{of } L \times T\).
Meta-Theory for Stores

- The “trick” is to remove the location typing from assumptions!
- Side-steps the mismatch just observed.
- But is substitution still valid?
- Illustrates a recurring technique of isolating variables for special treatment.
Adding A Store, Revisited

Retain location typing on main judgement:
\[ \text{of} : \text{lt} \to \text{tm} \to \text{tp} \to \text{type}. \]

Add a typing judgement for assumptions:
\[ \text{assm} : \text{tm} \to \text{tp} \to \text{type}. \]

Consider worlds of the form
\[ \text{x : tm, dx : assm x T} \]
Adding A Store, Revisited

Add an explicit "hypothesis" rule:
\[ \text{of}_\text{var} : \text{assm } E \ T \rightarrow \text{of } L \ E \ T. \]

Revise typing rules accordingly:
\[ \text{of}_\text{lam} : \\
( \{ x : \text{tm} \} \{ dx : \text{assm } x \ T \} \text{of } L \ (F \ x) \ U ) \\
\rightarrow \text{of } L \ (\text{lam } T \ F) \ (\text{arrow } T \ U). \]
Penalty: we now must prove that substitution preserves typing.

\[ \text{subst\_pres:} \]
\[ ({\{x : \text{tm}\}{dx : \text{assm \ times T}}} \text{ of } L (F \ x) \ U) \to \]
\[ \text{of } L \ E \ T \to \text{ of } L (F \ E) \ U. \]

Why does this work?
Meta-Theory For Stores

Proof is by structural induction on $F$.

- If it is constant, $[x] M$, substitution of $E$ has no effect, so result follows from typing of $M$ independently of $x$.

- If it is the identity, $[x]x$, the typing derivation for $E$ suffices.

- Otherwise proceed by induction.
Reasoning About Variables

Quite often one wishes to prove a meta-theorem about the behavior of variables.

- eg, substitution preserves typing
- eg, narrowing a variable to a subtype

Since the context is typically represented only implicitly in LF, these can be tricky.
Reasoning About Variables

For example, why does this type ...

\[
\{x : \text{tm}\}\{dx : \text{assm } x \times T\} \text{ of } (F x) \ U \rightarrow \\
\text{of } E \ T \rightarrow \text{of } (F E) \ U \rightarrow \text{type}.
\]

... codify this substitution principle?

if \( G, x : T, G' \vdash F : U \) and \( G \vdash E : T \),
then \( G, G' \vdash [E/x]F : U \)
Reasoning About Variables

The key is permutation, which permits us to regard $G,x:T,G'$ as $G,G',x:T$ in STLC.

When permutation is available, we can readily use relational methods to prove properties of variables.

Any given variable is implicitly “last”.

But what if we don’t have permutation?
Reasoning About Variables

From the POPLmark challenge for F::<
if G, X::Q, G' |- A :: B, and G |- P :: Q,
then G, X::P, G' |- A :: B.

Stated relationally, as for substitution,
narrow : 
( {X::tp} {dX : assm X Q} sub A B ) ->
sub P Q ->
( {X::tp} {dX : assm X P} sub A B ) ->
type.
Reasoning About Variables

- But this statement cannot be proved!
- Descending into a binder introduces an additional assumption, say $Y <: X$. 
- Cannot permute $Y <: X$ before $X <: Q$!
- So we must consider a general $G'$, which cannot be done uniformly in LF.
- The context $G'$ is not a “single thing”.
Reasoning About Variables

Adequacy for $F<:is$ for worlds built from declaration pairs of the form

$X : tp, dX : assm X T$

For example,

$t\text{lam}_\text{of} :$

$({X : tp}{dX : assm X T} \ of \ (F X) \ (U X)) \rightarrow$

of (t\text{lam} T F) (all T U).
Reasoning About Variables

We cannot, in general, permute such pairs past one another due to dependencies.

But, a limited form of permutation is OK:

\[
\begin{align*}
\{ X : \text{tp} \} & \{ Y : \text{tp} \} \\
\{ dY : \text{assm} \ Y \ X \} & \{ dX : \text{assm} \ X \ P \}
\end{align*}
\]

The strategy is to permit “mixed” permutations so that an \text{assm} can be last!
Reasoning About Variables

Revised relational statement of narrowing permits $X$ to be separated from $dX$:

$$\{X:\text{tm}\} \ ((dX : \text{assm} \ X \ Q) \ \text{sub} \ A \ B) \rightarrow \  
\text{sub} \ P \ Q \rightarrow \ 
(dX : \text{assm} \ X \ P) \ \text{sub} \ A \ B \rightarrow \ 
\text{type}.$$ 

But now $\text{assm} \ X \ Q$ no longer ensures that $X$ is a variable!
Reasoning About Variables

The sol’n is to “tag” each variable as such:
\[ \text{var} : \text{tm} \rightarrow \text{type}. \]

Then “link” each variable to an \textit{assm}:
\[ \text{var\_assm} : \text{var} \, \text{X} \rightarrow \text{assm} \, \text{X} \, \text{T} \rightarrow \text{type}. \]

Consider context blocks of these forms:

\[ \text{X} : \text{tp}, \, \text{vX} : \text{var} \, \text{X} \]
\[ \text{dX} : \text{assm} \, \text{X} \, \text{T}, \, \text{dvX} : \text{var\_assm} \, \text{vX} \, \text{dX} \]
Solving POPLmark

This was the hardest problem in the POPLmark challenge!

The rest was handled easily using standard methods with no serious complications.

This solution is a simplification of another that was much harder.

We finished the challenge in one week!
Scaling Up

- A full-scale language such as SML presents many other complications.
- Complex scoping rules.
- Type inference, overloading.
- Pattern matching.
- Coercive signature matching.
Scaling Up

One solution is to formalize elaboration of the external to an internal language.

Handle scope resolution, type inference, overloading, etc.

Target is chose to be amenable to formalization.

Examples: Russo, Harper-Stone, Epigram, ...
Scaling Up

Properties such as type safety are proved for the internal language.

Using methods sketched earlier.

These are transferred to external language by proving that a successful elaboration is well-typed.

Actually, has a principal type.
Formalizing Standard ML

- We are in the process of doing this for the HS semantics of ML.
- Progress, regularity for the IL done.
- Preservation for the IL mostly done.
- Elaboration is still “to do”.
- One significant complication arose ...
A Complication

- The HS IL has non-trivial type equality.
  - eg, to handle sharing spec’s, type definitions
- Typical meta-theorems need inversion properties of typing and type equality.
  - eg, if $A \rightarrow B = A' \rightarrow B'$, then $A = A'$ and $B = B'$
A Complication

- These are non-obvious for a “declarative” presentation of equality.
  - Transitivity obstructs a direct proof.
- We rely on an “algorithmic” presentation.
  - Inversion is easy.
- Completeness wrt declarative left open.
Conclusions

- Mechanized meta-theory for language definitions is feasible today.

- Requires some facility with LF and Twelf, but in the main it is smooth sailing.

- For this to work well we must formulate a definition with mechanization in mind.
Questions?