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 (a 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.
  - Formalize 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

Cat'y	Rep'n	Contexts/World
T type	T : tp	
E term	E : tm	x : tm
E : T	D : of E T	x : tm, dx : of x U

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:

```
pres : of E T -> step E E' -> of E' T -> type.
```

- Axiomatize this relation:

```
pres_beta :
  pres (of_app (of_lam D) D')
        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
  ∀ typing derivs ∀ steps ∃ 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:

```
of : lt -> tm -> tp -> 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 : tm, dx : of L x T
```

- Typing rules change accordingly:

```
of_lam :
  ({ x : tm }{ dx : of L x T } of L (F x) U) ->
    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 _ _)
  ```

Meta-Theory for Stores

- Consider the case of a lambda:

```
weaken_lambda :
  weaken (of_lam T D) X (of_lam T D') <-
   { x : tm }{ dx : of L' x T}
    (weaken (D x dx) X (D' x dx)).
```

- But this clause is not type-correct!
  - D x : of L x T -> ..., but dx : of L' x T!
  - There is no fcn of L' x T -> of L x 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:
  `of : lt -> tm -> tp -> type.`
- Add a typing judgement for assumptions:
  `assm : tm -> tp -> type.`
- Consider worlds of the form
  `x : tm, dx : assm x T`

Adding A Store, Revisited

- Add an explicit "hypothesis" rule:
  ```
  of_var : assm E T -> of L E T.
  ```
- Revise typing rules accordingly:
  ```
  of_lam :
   ( { x : tm } { dx : assm x T } of L (F x) U )
     -> of L (lam T F) (arrow T U).
  ```

Meta-Theory For Stores

- Penalty: we now must prove that substitution preserves typing.

```
subst_pres:
  ({x : tm}{dx : assm x T} of L (F x) U) ->
   of L E T -> 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 : tm}{dx : assm x T} of (F x) U) ->
  of E T -> of (F E) U -> type.
```

- ... codify this substitution principle?

  if G,x:T,G' |- F : U and G |- E : T,
  then G,G' |- [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,

```
tlam_of :
  ({X : tp}{dX : assm X T} of (F X) (U X)) ->
    of (tlam 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:

```
{ X : tp } { Y : tp }
{ dY : assm Y X } { dX : assm X P }
```

- The strategy is to permit "mixed" permutations so that an assm can be last!

Reasoning About Variables

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

```
{X:tm} ({dX : assm X Q} sub A B) ->
       sub P Q ->
       ({dX : assm X P} sub A B) ->
       type.
```

- But now assm X Q no longer ensures that X is a variable!

Reasoning About Variables

- The sol'n is to "tag" each variable as such: var : tm -> type.
- Then "link" each variable to an assm: var_assm : var X -> assm X T -> type.
- Consider context blocks of these forms:
  - X : tp, vX : var X
  - dX : assm X T, dvX : var_assm vX 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->B = A'->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?