Mechanizing the Metatheory of Standard ML

Daniel K. Lee
Carnegie Mellon University

Joint work with Karl Crary and Robert Harper

Workshop on Mechanizing Metatheory, 9/21/06

Language definitions

- Toy languages enjoy:
  - Fully rigorous definitions
  - Extensive metatheoretic analysis
  - Type safety proofs
- We settle for much less in the languages we actually use.
  - Definitions are informal, semi-formal, or non-existent
  - No type safety proofs

This project

- Goal: A fully rigorous definition and type safety proof for Standard ML.
- Employ elaborative semantics.
  - Define SML by translation into a type theory.
- Mechanize all definitions and proofs in Twelf.

Elaborative semantics

- Specify syntax for *external language* (EL).
- Define an *internal language* (IL).
  - Including static and operational semantics.
  - Type-theoretically well-behaved.
  - Provide the same expressive power as the language.
  - Not necessarily the same convenience.
- Give a formal translation (*elaboration*) of EL into IL.

Benefits

- Use standard techniques to prove safety of the IL.
  - Don’t have to wrestle with handy but ill-behaved constructs (*e.g.,* open).
- Result applies to the full language.

This work

- Defined an IL for Standard ML.
  - Static semantics
  - Structured operational semantics
- Proved type safety.
- Everything formalized in Twelf.

- Elaboration is future work.

Why mechanize?

- Want to be confident in results.
- Elaboration is no silver bullet.
  - IL is well-behaved, not small.
- Mechanization exposed numerous errors in an earlier elaborative definition of SML. [Ashley-Rollman 2004]
  - Most minor, but a few were serious.

IL

- Module-oriented language
  - Modules, signatures
  - Translucent sums (singleton kinds)
- Polymorphism, recursive types
- Exceptions
- Dynamic tagging (exn type)
- References
- Products, sums, etc.

Issues

- Mathematical challenges to proving safety.
- Formalization of IL in LF.
- Mechanizing the safety proof in Twelf.

Formalization in LF

- Mostly straightforward.
- Why?
  - LF is great.
  - We allow formalization process to advise the design.
  - Don’t try to formalize off-the-shelf.
- Some interesting issues arose.
  - Ended up improving the IL’s design.

Phase distinction

- Need to maintain *phase distinction* between static and dynamic components.
- Types ought not depend on dynamic computations.
  - Don't want:
    (if phase_of_moon () then int else int -> int)

Achieving the phase distinction

Two approaches:

- Allow apparent dependencies of types on terms (*e.g.,* M.t), and prove that terms do not affect types.
  - Can be complicated.
  - Has not been done with singleton kinds.
- Make phase separation manifest in syntax.

Manifest phase separation

- Types cannot refer to module expressions.
- A meta-operation Fst associates modules with their type components.
- When introducing a module *variable* introduce a type variable also.

- Issue: how to maintain the association between module and type variables?

Association via spelling

- Employ a spelling convention to associate module variables and type variables. [Harper *et al.* 1990, Dreyer 2005]
  - Module variable s provides type variable s^c.
- Breaks alpha conversion.
  - Thus, does not formalize well in LF.

Association via judgement

- Introduce two distinct variables.
- Associate them using a hypothetical judgement:

$$m : \mathrm{mod},\ t : \mathrm{tp},\ d : \mathrm{Fst}(m) = t,\ \ldots\ \vdash J$$

- Propagate this back into the IL design.

Mechanized type safety

- Proved progress and type preservation in Twelf.
  - 62k lines of code
    (including comments and whitespace)
- Quite a lot of it was straightforward.
- Some interesting issues arose.

Pair inversion

- For preservation, we need an inversion lemma for pairs of modules.
  - If <M, N> : S x T, then M : S and N : T.
- Non-trivial because “selfification” rules type modules in terms of larger modules.
- Induction hypothesis must be strengthened to accommodate these larger modules.

Proving pair inversion

- Larger modules in premises captured with evaluation contexts in the form

E ::= [ ] | fst E | snd E

- Pair inversion proved alongside beta-reduction properties.
  - If E[fst <M, N>] : S, then E[M] : S.
  - If E[snd <M, N>] : S, then E[N] : S.
  - If <M, N> : S x T, then M : S and N : T.

Evaluation Contexts in LF

- Contexts in LF encoded using functions of **`(module -> module)`**.
- Use a judgement to isolate the evaluation contexts.

```
ec : (module -> module) -> type.

ec/empty : ec ([m] m).
ec/fst   : ec E -> ec ([m] fst(E m)).
ec/snd   : ec E -> ec ([m] snd(E m)).
```

- Instantiation of contexts is just application in LF.

Type inversion

- For canonical forms, we need inequality lemmas.
  - Such as int $\neq$ bool
- Also need inversion lemmas.
  - Such as, if t1 $\times$ t2 = t3 $\times$ t4 then t1 = t3 and t3 = t4

Proving type inversion

- Need to impose structure on type equality derivations.
- Typically done using reduction-based strategies.
  - Don't work here, singleton kinds make equality context sensitive.
- Also done using logical relations. [Stone & Harper 2000]
  - Can't (in general) do logical relations in Twelf.

Proving type inversion

- New proof based on interpretation of IL’s types and kinds in a canonical formulation.
  - Equal types must be written the same way.
- Maintain canonicity using hereditary substitution.
  [Watkins 2003]
- Uses explicit context technique to establish substitution.
  [Crary 9:30am]

Related work

- VanInwegen [1996] attempted to prove type safety for SML using The Definition with HOL
- Did not fully succeed:
  - Wasn’t type safe.
  - Awkwardness of the Definition.
  - Treatment of alpha conversion problematic.
  - Immaturity of available tools.

Related work

- Ashley-Rollman [2004] attempted to prove type safety for SML using Harper-Stone with Twelf.
- Did not fully succeed:
  - Technical problems involving “selfification” and module call stacks.
  - Soundness of Harper-Stone is still open.
- Lesson: allow formalization process to advise language design.

Future work

- Formalize elaboration of SML to IL in LF.
  – Prove static correctness in Twelf.
- Use this as a framework to explore language extensions.
- Exploit this work in a validated compiler.
  – An elaborator is a formal front-end.