The Future Of Standard ML

Robert Harper
Carnegie Mellon University

Whither SML?

- SML has been hugely influential in both theory and practice.
- The world is slowly converging on ML as the language of choice.
- There remain big opportunities to be exploited in research and education.

Convergence

- The world moves inexorably toward ML.
  - Eager, not lazy evaluation.
  - Static, not dynamic, typing.
  - Value-, not object-, oriented.
  - Modules, not classes.
- Every new language is more “ML-like”.

Convergence

- Lots of ML's and ML-like languages being developed.
  - O'Caml, F#, Scala, Rust
  - SML#, Manticore
- O'Caml is hugely successful in both research and industry.

Convergence

- Rich typing supports verification.
  - Polytyping >> Unityping
  - Not all types are pointed.
- Useful cost model, especially for parallelism and space usage.
- Modules are far better than objects.

Standard ML

- Standard ML remains important as a vehicle for teaching and research.
  - Intro CS @ CMU is in SML.
  - Lots of extensions proposed.
- We should consolidate advances and move forward.

Standard ML

- SML is a language, not a compiler!
  - It "exists" as a language.
  - Stable, definitive criterion for compatibility.
- Having a semantics is a huge asset, provided that it can evolve.

Standard ML

- At least five compatible compilers: SML/NJ, PolyML, MLKit, MosML, MLton, MLWorks (?).
- Several important extensions: CML, SML#, Manticore, SMLtoJS, ParallelSML (and probably more).
- Solid foundation on which to build and develop.

The Way Forward

- Correct obvious shortcomings.
  - eg, structure sharing is broken
- Consolidate advances
  - eg, separate compilation
- Encourage innovation.
  - eg, parallelism, concurrency

The Way Forward

- Requires a community effort for both design and implementation.
  - A compiler is not enough.
  - A semantics is not enough.
- Key: open-source The Definition.

Open-Sourcing The Definition

- MIT Press has released the copyright on The Definition.
- Plan to recreate the (lost) sources as a GitHub.
- Institute a HoTT-book style revision process.

Opening The Definition of SML

- Correct the obvious errors.
  - Structure sharing is broken.
  - Equality, overloading are a mess.
- Consider obvious extensions.
  - Local structure bindings.
  - Separate compilation.

Opening The Definition of SML

- Enrich dynamics semantics with costs.
  - exp ⇒ val / cost
  - cost specifies dependencies among subcomputations and their data
- Express parallel time and space requirements.

Mechanizing The Definition

- Mechanize the metatheory!
  - Sanity check on revisions facilitates evolution.
  - See D. Lee, K. Crary, and H (POPL 06 paper)
- Twelf (or Celf) is ideal for formalization.

Mechanizing The Definition

- But the existing Definition is not amenable to such analysis!
  - van Inwegen's experience
- Requires a re-structuring of The Definition using types, structural operational semantics.

Mechanizing The Definition

- Two broadly similar approaches are already available.
  - Russo, Dreyer, Rossberg
  - Stone and H.
- The latter (at least) has been fully mechanized and proved sound.

Mechanizing The Definition

- Define an Internal Language.
  - Well-defined binding and scope.
  - Well-understood type system.
  - Dynamics given by SOS, not ES.
- Prove the internal language sound.
  - Progress + Preservation

Mechanizing The Definition

- Define an elaboration of Standard ML into the Internal Language.
  - Type reconstruction.
  - Coercive subtyping.
- Prove the static correctness of the elaboration.

Some Obvious Extensions

- Local structure and functor bindings.
  - Polymorphic fcns are functors.
  - Functors within structures.
  - Let-bound structures and functors.
- Crucial for modular type classes.

Some Obvious Extensions

- More flexible treatment of records?
  - O'Caml row polymorphism (in MLKit)
  - SML# extensions (see which)
- Foreign-function interface?
  - SML/NJ, SML#, ...

Separate Compilation

- Separate compilation.
  - See Swasey, et al MLW 2006.
  - There are several incompatible versions extant.
- Resist the “mixin” temptation.
  - “open recursion” sucks.

Implicit Parallelism

- Language constructs for parallel programming:
  - Comprehensions, sequences.
  - Make “and” mean “parallel”?
- Deterministic: semantics is the same as sequential, only cost differs.

Implicit Parallelism

- Parallel interpretation of “and”.
  - `val x = e and x’ = e’`
- Parallel sequences.
  - `$[0,1,2,3,4,5]`
  - `map`, etc with parallel costs

Segregation of Effects?

- See Ph. Ajoux's Monadic MosML.
  - Change basis, not language.
  - Exceptions are not effects.
  - Syntax for imperative code.
  - Top-level changes.
- Bonus: `performIO` is safe!

Segregation of Effects

- Imperative code blocks:
```
begin
   do print “hello”
   val s = “good-bye”
   do print s
end
```
- Top-level: `eval exp, do cmd`

Some More Ambitious Extensions

- Concurrent composition (non-determinism).
  - Reppy’s CML.
  - Fluet’s transactional CML.
  - Rust? Manticore?
- Goal is expressiveness, not cost.

Modular Type Classes

- Dreyer, Chakravarty, and H POPL 07
  - Type classes are signatures.
  - Instances are structures.
  - Polymorphic fcns are functors.
- Generalizes the HS semantics of SML.

Modular Type Classes

```
signature EQ = sig
  type t
  val eq : t * t -> bool
end

signature ORD = sig
  include EQ
  val lt : Eq.t * Eq.t -> bool
end
```

Modular Type Classes

```
structure IntEq : EQ = ...
structure IntOrd : ORD = ...
functor LexOrd(X:ORD,Y:ORD):ORD = ...

fun (Ord:ORD)compare(x:ORD.t, y) =
 let using Ord in ... eq ... lt ...
```

"`using`" actives instances in a scope

Integrating Modules and Datatypes

- Datatypes spec's are signatures!

```
signature LIST = data
  type 'a t
  con nil : 'a t
  con cons : 'a * 'a t -> 'a t
 end
```

- (Or use re-use existing syntax.)

Integrating Modules and Datatypes

- Datatype decl's are structures!
  - `data structure List : LIST` (default implementation)
  - `data structure List : LIST = ...` (non-standard implementation)
- "`data`" makes available for pattern matching

Integrating Modules and Datatypes

- Extends pattern-matching to user-defined abstract types.
  - datatypes are just adt’s with default implementations
  - purity is required to ensure predictable behavior.
- Eliminates redundancy problem in SML.

Signature-Specific Syntax

- Signature-specific syntax extension?
  - infix in signatures is a start
  - F# has done a lot with this
- Attach "comprehension" to COLLECTION.
  - eg, $[x|f(x)] always means map f
  - which map det'd by declaration

Algebraic Effects?

- Eff (Bauer, Pretnar) declares effects.

```
type a ref = effect
  opn ! : unit -> a
  opn := : a -> unit
end
let ref x = new ref @ x with
  opn ! () @ s = (s, s)
  opn := s' @ _ = ((), s')
end
```

Type Refinements?

- Type refinements capture useful invariants.
  - Inductive data types.
  - Array bounds, sizes.
- A practical pathway to dependent types and stronger specifications?

Dependencies?

- Dependent types are the future.
  - GADT’s are hacky DT’s
  - Purity is essential, but equivalence is also a problem.
- ML is ideal for exploring dependency (cf Idris, F*)

Libraries?

- The biggest problem is to develop a rich set of libraries.
  - FFI's, interoperability a must.
  - Standard Basis is far too minimal.
- Smackage is a good start, but only a beginning.

Compilers?

- With a half dozen viable compilers for SML, evolution seems possible.
  - Definition consolidates
  - Compilers incorporate
- Some changes require more fundamental re-thinks than others.

Grand Unification?

- It would be great to consolidate the advances made in O'Caml and Standard ML into the next great ML.
- There are no fundamental impediments, but lots of social and practical issues to manage.

Conclusion

- Standard ML remains the ideal basis for teaching and language research.
- We should consolidate disparate efforts and form a community process for evolution.
- There are many good opportunities!