How SML started – Where it might go from here

David MacQueen

University of Chicago

WG 2.8 Utrecht, April 2024

Rant on Type Explicitness – somewhat polemic

The association between a type (constructor) and its name is *fragile* and *tricky*.

The problem is complicated by:

- *shadowing* (especially in a REPL with linear, cumulative environment)
- *localized type declarations* in `let` and `local`
- *aliasing* declarations (multiple names, with different scopes)
- SML's `open` declaration

Apology: SML/NJ could work harder at finding appropriate names for type constructors (in the REPL and in error messages) before resorting to "?".

Part I: A Personal summary of where Standard ML came from

I entered the PL research world upon arriving Edinburgh in May, 1975 (49 years ago!).

The Standard ML design started in Edinburgh in April 1983.

What was known when Standard ML was designed?

Low hanging fruit! (PL theory and design)

- The fun comes at the beginning!
- We get to *design* new languages *de novo*.
- Dito for the corresponding theory.
- There is lots of *low-hanging fruit*.

What was going on when I joined the game?

Consider the *state of the art* in programming languages in the mid- to late-1970s.

- Not too many (significant) languages.
- Not much theory (Hoare logic, denotational semantics, beginnings of operational semantics)
- About 1 decade of cumulative research and literature on "principles" of programming languages

Hence lots of (what appear from our perspecitive to be) *low hanging fruit*.

What was new 60 years ago, in the 1960s?

- McCarthy's papers, around 1963
- Formal Language Description Languages (1963) - first sematics of PL conference
- Strachey and Landin
  - *Varieties of Programming Languages* (1967), parametric polymorphism
  - Landin's papers from 1964 to 1966, relevance of lambda calculus to PL (translational semantics of Algol, SECD machine, ISWIM)
  - denotational semantics, Strachey and Scott, Milne (typed lambda calculus as metalanguage, domain theory)
- Reynolds: Gedanken, semantics, types
- Pascal, Algol 68, Algol W, (leading later to: Modula, Mesa)
- Floyd (reasoning about simple imperative programs)
- Simula 67, first *object-oriented* language, based on Algol 60

What was new 50 years ago, in the 1970s?

- abstract types (72-75): Hoare, Alphard, CLU
- Milner arrives in Edinburgh in 1973: LCF project begins, including LCF/ML
- Burstall (at Edinburgh):
  - Functional program transformation and synthesis with Darlington (1st order functional)
  - *NPL* & *Hope* (1980) languages: algebraic data types, pattern matching
  - *Clean* algebraic specification language (with Goguen)
- *Bob Boyer Memorial Society*
- Floyd and Hoare: formalisms for reasoning about programs in simple imperative languages
- Edinburgh-style *stuctured operational semantics* (Milner, Plotkin); static & dynamic semantics

More of what was new 50 years ago, in the 1970s?

- INRIA group (Kahn, Huet, Levy, Lang, Berry, Mentor project)
- Martin-Löf et al, type theory
- Algebraic stuff: the ADJ group at IBM, algebraic specification, *Clean* specification language
- Lazy evaluation (Friedman & Wise, Morris & Henderson, Ashcroft & Wadge (Lucid), Kahn & MacQueen)
- Russell: Donahue and Demers at Cornell - 1st class types (Mathews's *Poly*)
- Object-oriented languages (Smalltalk)

Parallel developments in Logic

- Russel and Whitehead Principia Mathematica (1910s) - types
- Gödel (1930) - general recursive functions, the first functional language?
- Haskell Curry (1930, 1934, 1956) combinatory logic, *functionalities* (*i.e.*, types)
- Alonzo Church (lambda calculus, 1932, 1936; simply typed lambda calculus, 1940)
- Max Newman (1943) type inference
- Roger Hindley (1959) type inference
- Per Martin Löf type theory
- J.-Y. Girard: System F

1980s - emergence of serious functional languages

- Cardelli ML (1981)
- Standard ML design (1983 - 1986) Not quite *de novo*, preceded by LCF/ML, Cardelli ML, and Hope
- Standard ML formal definition (1986 - 1989)
- Early SML implementations: Edinburgh SML (85), SML/NJ (86), PolyML (85)
- INRIA: CAML fork of ML
- Compilation and runtime technologies for ML family (strict, functional, impure) languages
- Haskell
- continuing development of type theory and semantics of types (and modules)
  - types and logic: Martin-Löf type theory, Huet's Calculus of Constructions (COQ)
  - types as ideals (MacQueen, Plotkin, Sethi)

1990s - emergence of serious functional languages

- SML '97 revision (with difficulty) Definition as a *book* is a blessing and a curse (language = the book?)
- (SML/NJ) FLINT project at Yale
- (SML at CMU) TIL, TILT, and Foxnet, type-theoretic SML semantics
- (SML) new implementations: ML Kit, Moscow ML, MLton
- (SML/NJ; BL+CMU) SourceGroups leading to CM (Compilation Manager) in SML/NJ Static dependency analysis, Cutoff recompilation, CM 2
- SML Basis Library (and smlnj-lib libraries)
- (SML/NJ) 6 RISC architecture ports, MLRISC framework, x86 (finally)
- SML text books
- ML2K discussions (1993 - 2000)
- OO type theory (Cardelli, Mitchell, Bruce)
- Appel moves on to verification research (1995)

2000s - emergence of serious functional languages

- DBM, JHR, Blume move from Bell Labs to Chicago
- Continuing work on libraries (SML Basis, smlnj-lib)
- correctness of ranked variable techniques for type generalization
- Kuan's thesis: refined semantics of higher-order functors

2010s

- DBM becomes emeritus, moves to Los Gatos
- F-ing modules (2010) , 1ML, *etc.* (Rossberg, Russo, Dryer)

2020s - technology-driven reaction

- (SML/NJ) *Survival Crisis 1*: 32 bit to 64 bit transition
- (SML/NJ) move to new home at GitHub (not quite finished)
- (SML/NJ) LLVM-based code generation
- (SML/NJ) *Survival Crisis 2*: ARM port?
- (SML/NJ) *Survival Crisis 3*: internal complexity and orphan code
- (SML/NJ) restoration/rejuvination projects:
  - match compiler, prettyprinting, typechecking, module system;
  - LLVM code generation, runtime system
  - Problem of FLINT (technological obsolescence)
  - Problem of CM and bootstrapping

Emergence of something new, followed by ...

- design stabilization (stasis?), after a few "successful" precursor designs
- documentation
- gradual/incremental refinement of implementation
- gradual/incremental refinement of theory
- gradual/incremental development of programming technique (methodology, pragmatics)
- possible design *drift* involving incremental addition of features
- maybe "standards", following sufficient *impact* (*industrialization*)

Successful "Real World" Programming Langauges

- Fortran (IBM)
- C (Unix, Bell Labs, Center 1127)
- C++ (Unix, Bell Labs, OO hype)
- Java (Sun Microsystems)
- Javascript (Netscape, web programming)
- C# (Microsoft)
- Swift (Apple)
- Python (libraries?)
- Perl, Ruby, ... (web programming)
- *maybe* Rust (Mosaic)

Programming Language Research ...

... produces successful *ideas*, but rarely (never?) a "popular" programming language

Success for a programming language usually happens because

- it comes with something else (*e.g.*, Unix, libraries, web browsers), or
- it is backed and promoted by a sufficiently powerful institution (e.g. a company)

Part II. Is a little more ML language design is possible?

Early on, Standard ML was put into statis by the existence of "the book" (= the language?).

But perhaps we have moved beyond that by now.

What defines a programming language?

Typical situation

language = implementation (unique or dominant) Implementor(s) own the language.

SML

language = formal definition =¿ implementations Who owns the language (the book authors)?

- In either case, there will typically be multiple designers.
- For each implementation, there may be multiple implementors.
- For a formal definition (i.e., a book), there may be multiple authors.
- Multiple independent implementations is not necessarily an advantage.

Levels of commitment to an implementation

1. **DEMO**: demonstrate that a language design can be implemented
2. **PERSONAL USE**: platform for personal use (& student research)
3. **PUBLIC USE**: support for general use

Each step to the next level involves 10X more time and effort.

Standard ML did not conquer the world!

- long dominance of "Object-Oriented" ideology
- lack of "strong enough" user and implementor communities
- failure to reach "commercialization" (Milner's dream)
- slow osmosis of concepts, compilation technology, even syntax elements, into "mainstream languages"
- possible useful byproduct: exposing and propagating the *lessons learned* and *technology*

What to fix in Standard ML

The Standard ML design was not perfect and was put into stasis prematurely by publication of the Definition 90 book. Partial revision was achieved with Definition (Revised) in 1997. Is it worth thinking about Standard ML even though it has not conquered the world?

A few believe that it is still worth tinkering with the language (or seeking a "successor"). But one must fight entropy!

Perfection (or even improvement!) not usually achieved by growing the language!

What to fix in Standard ML: Core syntax

- Fix shift-reduce conflicts in yacc-style grammar (syntactic ambiguities).
- Remove lexically scoped fixity declarations.
  Question: How and whether to accomodate user-defined infix operators?
- Enforce a data constructor name policy (capitalized alphanumeric?)
  Violated by `::` (symbolic) and `nil`, `cons`, `true`, `false`.

What to fix in Standard ML: Core types

- Change type variable lexical convention:
  Distinguishing type variables and constructors by case (as in Haskell), with traditional special cases of infix type constructors "-¿" and "*".
- Drop inference of binding sites of inferred polymorphic type variables
  Infer instantiations, not abstractions of (explicit) type variables.
  Slogan: "Any variable explicitly introduced by the programmer should be bound by the programmer."
- Drop equality polymorphism and equality type variables
  Can be seen as a unique, ad hoc, "type class" in SML, though not necessarily implemented as such.
- (*New*): record update of some form (successor ML has one suggestion).

What to fix in Standard ML: Modules

- Add "signature functors" signatures parameterized by structures
- Add "type-only" signatures to deal with duplication issue
- Deprecate/eliminate "sharing equations" - use definitional specs and where clauses instead.
  does small loss of expressiveness matter?

What to fix in Standard ML: Successor ML

The Successor ML project has further proposals – see the web page at Github.com.

Legacy issues in a 40-year old system

- New implementation of
  - explicit type variables
  - explicit type constructor variables
  - related abstract syntax
- New implementation of HOF following Kuan-MacQueen (possibly influenced by Rossberg)
- Improved type error messages and type error management.
- FLINT (particularly FLINT types)
  - needs thorough rewrite or replacement (largely orphan code)
  - FLINT contamination of Front End should be removed
  - overly complex FLINT types (hash consing, Nadathur closures)
- Compilation Manager (CM)
  - needs to be redesigned and re-implemented, CM 2
  - proper MML (meta-ML) language features
  - (partial?) compatability with MLB
- Simplify and document system bootstrapping process (CM dependent) currently based on an excessively complex web of CM specification files

Questions or Discussion?