On Implementing the Forlan Formal Language Theory Toolset in Standard ML

Alley Stoughton
alley.stoughton@gmail.com

Forlan [Sto05, Sto08, Sto12] is a toolset supporting sophisticated experimentation with the objects of formal language (automata) theory: regular expressions, finite automata, grammars, a universal programming language, parse trees, etc. It is implemented in Standard ML (SML) [MTHM97], as a library on top of Standard ML of New Jersey (SML/NJ) [AM91]. Forlan is used interactively, but users are able to extend it by defining SML functions. Forlan currently consists of approximately 12,000 lines of SML code, distributed over nearly 40 modules. The Forlan Project also includes a draft textbook on formal language theory, and I have attempted to keep the notational distance between the book and toolset as small as possible.

I chose SML as the implementation language for Forlan because SML's concepts and notation are similar to those of mathematics, and yet it is easy to write efficient code in SML. My hope was that it would prove possible to naturally and simply express the definitions of formal language theory in SML, and to implement the algorithms of formal language theory in ways that were simultaneously natural and reasonably efficient. My presentation will assess how well SML—and SML/NJ—supported achieving this goal.

My conclusions are largely positive: it was typically easy to naturally and efficiently implement the book's mathematics in SML, with both the core and module languages serving well. For example, most of the objects of formal language theory—different kinds of finite automata, grammars, etc.—are naturally implemented as abstract types, and this was neatly done using structures, signatures and opaque ascription.

But limitations in SML sometimes made it difficult or impossible to perfectly capture the mathematics. For instance:

- The lack of a type `nat` of natural numbers, and the impossibility of using successor as a constructor, created a small, but irritating, gap between the mathematics and toolset.
- The lack of recursive modules necessitated careful, and sometimes non-optimal, placement of functions involving types from multiple modules.
- The lack of subtyping necessitated the definition and use of explicit injection functions (e.g., for giving a value of type `dfa` the type `nfa`, even though this involved no change of data).
- The lack of dependent types necessitated an insecure implementation of the type family `'a set` of finite, ordered sets.

- The lack of dependent types made it impossible to use the type system to express common constraints regarding alphabets (e.g., that a string be in $\{0,1\}^*$, or that a finite automaton's alphabet be a subset of $\{0,1\}$).

SML/NJ proved robust and provided sufficient speed. However, the impossibility of configuring SML/NJ so as to implement integers as arbitrary precision integers necessitated another minor but annoying gap between the mathematics and the toolset.

In documenting Forlan's implementation, I was frustrated by the lack of support in SML for expressing specifications, both of correctness and termination. Of course, SML even lacks syntax for specifying the types of local functions; putting such type specifications in (unchecked) comments is a poor substitute. And, making the programmer express correctness and termination specifications in an adhoc syntax within comments means that such specifications can't even be typechecked.

I will argue that a modern language should allow such specifications to be formally expressed, so that implementations may type-check them, and—ideally—will employ a theorem prover or proof assistant to assess their validity. Experience with Leino's Dafny language [Lei10] suggests that—in a version of SML with support for specifications—many specifications could be automatically verified.

References

[AM91] A. W. Appel and D. B. MacQueen. Standard ML of New Jersey. In *Programming Language Implementation and Logic Programming*, volume 528 of *Lecture Notes in Computer Science*, pages 1–26. Springer-Verlag, 1991.

[Lei10] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370, 2010.

[MTHM97] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML—Revised 1997*. The MIT Press, 1997.

[Sto05] A. Stoughton. Experimenting with formal languages. In *Thirty-sixth ACM SIGCSE Technical Symposium on Computer Science Education*, page 566. ACM Press, 2005.

[Sto08] A. Stoughton. Experimenting with formal languages using Forlan. In *FDPE '08: Proceedings of the 2008 International Workshop on Functional and Declarative Programming in Education*, pages 41–50, New York, NY, USA, 2008. ACM.

[Sto12] A. Stoughton. Forlan Project. `http://alleystoughton.us/forlan`, 2012.