Luca Cardelli and the Early Evolution of ML

David MacQueen

Abstract

Luca Cardelli has made an enormous range of contributions, but the focus of this paper is the beginning of his career and, in particular, his role in the early development of ML. He saw the potential of ML as a general purpose language and was the first to implement a free-standing compiler for ML. He also made some important innovations in the ML language design which had a major influence on the design of Standard ML, in which he was an active participant. My goal is to examine this early work in some detail and explain its impact on Standard ML.

1 Introduction

My goal here is to tell the story of the early days of ML as it emerged from the LCF system via Luca Cardelli's efforts to create a general purpose version of ML, called VAX ML. Starting in 1983, the new ideas developed in VAX ML and the HOPE functional language inspired Robin Milner to begin a new language design project, Standard ML, and for the next two years Luca was an essential contributor to that effort.

2 VAX ML

We will start by giving a brief history of Luca's ML compiler, before considering the language innovations it introduced and how the language evolved in Section 2.1 below.[1]

Luca began working on his own ML compiler sometime in 1980. The compiler was developed on the Edinburgh Department of Computer Science VAX/VMS system, so Luca called it "VAX ML" to distinguish from "DEC-10 ML", the

[1]Note that Luca was building his VAX ML at the same time as he was doing research for his PhD thesis (Cardelli 1982b). He also developed his own text formatting software, inspired by Scribe, that he used to produce his thesis and the compiler documentation, and a simulation of the solar system!

LCF version of ML[2] which ran under Stanford Lisp on the DEC-10/TOPS-10 mainframe.[3] Luca's preferred working language at the time was Pascal, so both the compiler and the runtime system were written in Pascal, using Pascal's unsafe union type to do coercions for low-level programming (e.g. for the garbage collector).[4] The compiler generated VAX machine code, and was much faster than the DEC-10 (LCF) version, which used the host Lisp interpreter to execute translated ML code. The VAX ML compiler was working and had begun to be distributed to users by the summer of 1981 (version 12-6-81, i.e. 12 June 81), although a working garbage collector was not added until version 13-10-81.

The earliest surviving documents relating to the compiler date to late 1980: "The ML Abstract Machine", a description of the abstract machine AM (Cardelli 1980a), (which would develop into the FAM (Cardelli 1983a)), and "A Module Exchange Format", a description of an external string format for exporting ML runtime data structures (Cardelli 1980b). There is a README file titled "Edinburgh ML" from March, 1982 that describes how to install and run the system (Cardelli 1982a), and a partial manual titled "ML under VMS" providing a tutorial introduction to the language (Cardelli 1982d), corresponding to Section 2.1 of "Edinburgh ML" (Gordon et al. 1979).

In early 1982, Nobuo Saito, then a postdoc at CMU, ported VAX ML to Unix, using Berkeley Pascal (Saito 1982). In April, 1982, Luca completed his PhD at Edinburgh (Cardelli 1982b) and moved to the Computing Science Research Center (the home of Unix) at Bell Labs, and immediately began his own Unix port, which was available for distribution in August, 1982. The runtime system for the Unix port was rewritten in C, but most of the compiler itself remained in Pascal. The first edition of the Polymorphism newsletter (Volume I, Number 0) contained a list of known distribution sites (Cardelli 1982c) in November 1982; at that time, there were at least 23 sites spread around the world, several using the new Unix port. The Unix port had three releases during 1982 (13-8-82, 24-8-82, and 5-11-82), accompanied with some shifts in language design and system features, notably a new type checker for ref types and an early version of file I/O primitives.

The next major milestone was the first Standard ML meeting in Edinburgh in April, 1983 (See Section 3). Luca agreed to a request from Robin Milner to suspend work on his VAX ML manual pending developments around Robin's initial proposal for Standard ML (Milner 1983a). Following the meeting Luca began to change his compiler to include new features of the emerging Standard

[2]Commonly called LCF/ML

[3]Luca referred to these implementations as two varieties of "Edinburgh ML".

[4]Since the entire system was written in Pascal, there was no sharp distinction between the compiler and the runtime, which was simply the part of the system responsible for executing the abstract machine instructions (FAM code).

ML design, resulting in Pose[5] 2 (August 1983), Pose 3 (November 1983), and finally Pose 4 (April 1984). This last version is described in the paper "Compiling a Functional Language" in the 1984 Lisp and Functional Programming conference (Cardelli 1984a).

2.1 Language Innovations

The first description of the language of VAX ML was a file mlchanges.doc (Cardelli 1981) that was part of the system distribution. This file describes the language by listing the changes made relative to DEC-10 ML (i.e. LCF/ML). The changes include a number of minor notational shifts. For instance, LCF/ML used "." for list cons, while VAX ML initially used "_", later shifting to the "::" used in POP-2 (Burstall and Popplestone 1968). The trivial type (called "unit" in Standard ML) was denoted by "." in LCF/ML and by "triv" in VAX ML. A number of features of LCF/ML were omitted from VAX ML, e.g. the "do" operator, "sections", and the "!" and "!!" looping failure traps (Gordon et al. 1979, Chapter 2).

But the really interesting changes in VAX ML involved (1) new labelled record and union types, (2) the ref type for mutable values, (3) declaration combinators for building compound declarations, and (4) modules. We will describe these in the following subsections.

2.1.1 Labelled records and unions

Luca was of course familiar with the conventional record construct provided in languages like Pascal (Jensen and Wirth 1978, Chapter 7). But inspired by Gordon Plotkin's lectures on domain theory Luca looked for a purer and more abstract notion, where records and discriminated union types were an expression of pure structure, representing themselves without the need of being declared and named. The notation for records used *decorated parentheses*:

```
(|a1 = e1; ... ; an = en|)  :  (|a1: t1; ... ; an: tn|)
```

where `ei` is an expression of type `ti`. The order of the labelled fields in a record type did not matter – any permutation represented the same type.

Accessing the value of a field of a record was done using the conventional dot notation: `r.a`, where `r` is a record and `a` is a label. Records could also be deconstructed in declarations and function arguments by pattern-matching with a record *varstruct* (pattern), as in the declaration:

```
let (|a=x; b=y|) = r
```

[5] Luca called his compiler versions "Poses" adopting a terminology from dance.

From the beginning, Luca included an abbreviation feature for record patterns where a field name could double as a default field variable, so

```
let (|a; b|) = r
```

would introduce and bind variables named `a` and `b` to respective field values in `r`. All these features of the VAX ML record construct eventually carried over to Standard ML, with just a change in the bracket notation to use `{...}`.

Labelled unions were expressed as follows, using decorated square brackets:

```
[|a1 = e1|] : [|a1: t1; ... ; an: tn|]
```

The union type to which a given variant expression belonged had to be determined by the context or given explicitly by a type ascription. Variant varstructs could be used in varstructs for declaration and argument bindings, with their own defaulting abbreviation where a `[|a|]` stood for `[|a = ()|]`, both in varstructs and expressions, which supported an enumeration type style.[6] A case expression based on variant varstructs was used to discriminate on and deconstruct variant values, with the syntax

```
case e
  of [| a1 = v1 . e1;
        ...
        an = vn . en
     |]
```

where `e` is of type `[|a1: t1; ... ; an: tn|]`.

2.1.2 The ref type

In LCF/ML, mutable variables could be declared using the `letref` declaration keyword. In VAX ML, Luca replaced this with the `ref` type operator with its interface of operations `ref`, `:=`, and `!`. This approach was carried over unchanged into Standard ML, though the issue of how `ref` behaved relative to polymorphism took many years to resolve. In 1979, Mike Gordon wrote a brief note "Locations as first class objects in ML" (Gordon 1980) proposing the ref type for ML, with a restricted type discipline using *weak polymorphism* and *weak type variables*. Gordon suggested this as a research topic to Luis Damas, who eventually addressed the problem in his PhD thesis (Damas 1985, Chapter 3) using a rather complex method where typing judgements were decorated by sets of types involved in refs. Luca got the idea to use the ref type either from Gordon's note or via discussions with Damas, with whom he shared an office for a time. At the end of Chapter

[6] Initially, the record and variant abbreviation conventions were also applied to types, but this was not found useful and was quickly dropped.

3 of his thesis, Damas returns to a simpler approach to the problem of refs and polymorphism similar to the weak polymorphism suggested by Gordon, and he mentions that both he and Luca implemented this approach. However, the issue is not mentioned in the various versions of Luca's ML manuals (Cardelli 1982d, 1983c, 1984b), and the `ref` operator is described as having a normal polymorphic type.[7]

2.1.3 Declaration combinators

Another innovation in VAX ML was a set of (more or less) independent and orthogonal declaration combinators for building compound declarations. These are

- `enc`, for sequential composition, equivalent to nesting `lets`: "`d1 enc d2`" yields the bindings of `d1` augmented by or overridden by the bindings of `d2`.
- `and`, for simultaneous or parallel composition, usually used with recursion.
- `ins`, for localized declarations: "`d1 ins d2`" yields the bindings of `d2`, which are evaluated in an environment containing the bindings of `d1`.
- `with`, a kind of hybrid providing the effect of `enc` for type bindings and `ins` for value bindings; usually used with the special "`<=>`" type declaration to implement abstract types.
- `rec`, for recursion

There were also reverse forms of `enc` and `ins` called `ext` and `own` for use in `where` expressions, thus "`let d1 enc d2 in e`" is equivalent to "`e where d2 ext d1`".

This "algebra" of declarations (possibly inspired by ideas in Robert Milne's PhD thesis (Milne 1974)) was very interesting, but in programming the combinators would normally be used in a few limited patterns that would not take advantage of the generality of the idea. Indeed, certain combinations seemed redundant or problematical, such as "`rec rec d`", or "`rec d1 ins d2`" (`rec` syntactically binds weaker than the infix combinators).

Luca factored the `abstype` and `absrectype` of LCF/ML using `with` (and possibly `rec`) in combination of a special type declaration `tname <=> texp` that produced an opaque type binding[8] of `tname` to the type expression `texp` together with value bindings of two isomorphism functions:

[7] The notes at the end of (Cardelli 1982c), mention that the 5-11-82 Unix version has a "New typechecker for ref types." It is not known whether this typechecker used weak polymorphism.

[8] Meaning that `tname` was not equivalent to `texp`.

```
abstname : texp -> tname
reptname : tname -> texp
```

A compound declaration `tname <=> texp with decl` would compose the type binding of `tname` with `decl` while localizing the bindings of `abstname` and `reptname` to `decl`. Thus `with` acted like `enc` at the type level and `ins` at the value level. This in principle was more general than `abstype`/`absrectype` in LCF/ML, in that the declaration `d1` in `d1 with d2` was arbitrary and not restricted to a possibly recursive simultaneous set of isomorphism (`<=>`) type bindings, but it was not clear whether this greater generality would be exploited.

In the end, the `and` combinator was used in Standard ML, but at the level of value and type bindings, not declarations (just as it was used in LCF/ML), the `ins` combinator became the "`local d in e end`" declaration form, the `rec` combinator was adopted, but at the level of bindings (and still with too general a syntax!), and the `<=>`, `with` combination were replaced by a variant of the old `abstype` declaration, but using the datatype form for the type part and restricting the scope of the associated data constructors.

In later versions, starting with ML under Unix, Pose 2 (Cardelli 1983c), another declaration form using the `export` keyword was added. A declaration of the form

```
export exportlist from decl end
```

produced the bindings of *`decl`*, but restricted to the type and value names listed in the *`exportlist`*. Exported type names could be specified as abstract (in ML under Unix, Pose 4) meaning that constructors associated with the type were not exported. Thus both `local` and `abstype` declarations could be translated into export declarations.

2.1.4 Modules

LCF/ML had no module system; the closest approximation was the `section` directive that could delimit scope in the interactive top-level. Since VAX ML aimed to support general purpose programming, Luca provided a basic module system. A module declaration was a named collection of declarations. Modules were independent of the environment in the interactive system, and required explicit import declarations to access the contents of other modules (other than the standard library of primitive types and operations, which was always accessible); Luca called this *module hierarchy*. Compiling a module definition produced an external file that could be loaded into the interactive system or accessed by other modules using the import declaration. Importing (loading) a module multiple times would only create one copy, so two modules `B` and `C` that both imported a module `A` would share a single copy of `A`. The export declaration was com-

monly used to restrict the interface provided by a module. There was no way to separately specify interfaces (*signatures* in Standard ML).

3 The Standard ML Design

LCF/ML had great potential as a language, but its scope was severely limited by its context as a tool embedded in the LCF proof assistant. Luca realized its wider potential, which motivated him to create VAX ML. By 1982, both LCF and Luca's VAX ML had created a lot of interest, as had HOPE (Burstall et al. 1980), another functional language developed at Edinburgh by Rod Burstall, Don Sannella and myself. In November, 1982 a meeting was convened at the Rutherford Appleton Laboratory (RAL) to discuss the future of these three systems (Witty and Wadsworth 1982). The meeting was attended by 20 people, including Robin Milner, Rod Burstall, and several others from Edinburgh, John Darlington from Imperial College, and Bernard Sufrin from Oxford. The topics discussed included how the ML and HOPE languages could be supported and further developed (e.g. by creating new ports, etc.). Sometime during or after this meeting, Bernard Sufrin urged Robin to think about creating a new ML design that would consolidate what appeared successful about LCF/ML, VAX ML, and HOPE.

3.1 Meetings

By early April, 1983, Robin had created a hand-written first draft of "A Proposal for Standard ML" (Milner 1983a). By coincidence, both Luca and I were in Edinburgh at that time, as were Robin and Rod Burstall and their research groups (in particular, Kevin Mitchell, Alan Mycroft, and John Scott), and some other visitors (including, I believe, Guy Cousineau from INRIA)[9]. So this was a serendipitous opportunity for many of those most directly interested to immediately get together to discuss Robin's proposal. Many of the discussions took place in Robin's living room at Garscube Terrace. In the period immediately following these meetings, Robin summarized some suggested changes (Milner 1983b) and by June produced a second draft of the proposal (Milner 1983c). We collectively decided on several parallel efforts: Robin would continue to work on the Core language proposal, I would work on a module system,[10] and Luca would develop a proposal for stream Input/Output, based on his stream I/O interface in VAX ML. Luca also undertook to modify his VAX ML compiler by adding Standard ML design features and removing some features (records, labelled unions, type

[9]Unfortunately, the records of this first gathering in Edinburgh are spotty, and don't include a list of all the people involved

[10]I had already been working on a module system for HOPE (MacQueen 1981).

identity abbreviations) that did not get included in the Standard ML proposal. Discussion by correspondence and exchange of design proposals continued for the rest of 1983 and into 1984, including another draft of the Core language proposal (Milner 1983d), and Luca's proposal for stream I/O (Cardelli 1983b).

The next year, in early June, a second, more formally organized meeting was held to continue work on the design (MacQueen and Milner 1985). Luca was not able to attend this meeting, but I acted as his representative and he was active in providing comments. This meeting lead to the publication of the fourth draft of the Core proposal and my first Modules proposal in the Lisp and Functional Programming Conference that August (Milner 1984; MacQueen 1984). Luca's paper "Compiling a Functional Language", describing the ML under Unix (Pose 4) compiler, also appeared in that conference (Cardelli 1984a). The meeting report by Robin and myself appeared in December (MacQueen and Milner 1985).

The third meeting, which was called the ML Workshop, took place in May, 1985, and Luca was again able to participate in person. There are extensive reports and design proposals that were presented at the workshop or followed in its aftermath, including a meeting report by Bob Harper (Harper 1985a). There were also two further proposals for the stream I/O library, one by Robin and Kevin Mitchell (Mitchell and Milner 1985), and another by Bob Harper (Harper 1985b). Another revision of the Modules design was published in Polymorphism in October (MacQueen 1985).

From 1985 on, Luca's involvement in Standard ML tapered off because of other interests (see Section 4 below). Meanwhile, between 1985 and 1989, the refinement of the Standard ML design proceeded and work on its formal definition was completed. Counting design documents and versions of the formal definition, there were a total of (at least) 21 design or definition versions produced between April 1983 and the end of 1989, not counting modules and I/O.

3.2 Edinburgh ML

In 1982, while Luca was working on the Unix port of VAX ML at Bell Labs, back in Edinburgh Kevin Mitchell began a project to rewrite the VAX ML compiler in ML (the VAX ML dialect), replacing the Pascal code and thus allowing the compiler to bootstrap itself. John Scott and Alan Mycroft soon joined Kevin on the project. The runtime system of Edinburgh ML was rewritten in VAX assembly language, and then later rewritten in C on Unix based on a FAM bytecode interpreter, which saved significant code space. John Scott rewrote the parser, using an elaborate version of Vaughan Pratt's top-down precedence parser technique

(Pratt 1973)[11], and Alan Mycroft rewrote the type checker. One of the technical challenges of the project was that it was hoped the compiler could be ported to run on a locally developed workstation that had limited memory and no virtual memory.

The resulting new compiler was called Edinburgh ML, and initially it did not modify the language from VAX ML. But after the first Standard ML meeting in April, 1983 Edinburgh ML began to serve as a prototyping testbed for the Standard ML design along side VAX ML. Alan Mycroft left for Cambridge in the fall of 1984, and later Nick Rothwell and Bob Harper (in 1985) joined the effort, and K. V. S. Prasad was also involved at some point. Bob Harper rewrote the type checker again and implemented the module system as it stood at that point. This made Edinburgh a quite close approximation to Standard ML, although internally much of the compiler code remained in the VAX ML dialect, so the compiler had a general abstract syntax into which both VAX ML and Standard ML could be translated. Unfortunately this meant that the module system could not be exploited to organize the structure of the compiler itself, so the code was rather difficult to work with. In March, 1986, when Andrew Appel arrived in Princeton, he and I used the Edinburgh ML compiler to begin developing a new compiler, Standard ML of New Jersey, from scratch in Standard ML. We bootstrapped our compiler in the spring of 1987 just before exhausting certain size/space limitations in the Edinburgh compiler.

4 Beside and Beyond ML

During 1984 and 1985, Luca's effort on Standard ML was diluted as he began to work on other projects, such as the design and implementation of his own language, Amber (Cardelli 1986a,b). He also found time to write one of the most influential papers on type theory of the time, "A semantics of multiple inheritance" (Cardelli 1984c), which introduced the concept of record (width) subtyping.

I should also mention his very valuable expository efforts, which helped to propagate and popularize important ideas in type systems research. These include particularly the papers "Basic Polymorphic Typechecking" (Cardelli 1985), "On Understanding Types, Data Abstraction, and Polymorphism" (Cardelli and

[11] Vaughan Pratt had spent the summer of 1975 in Edinburgh, and had taught everyone his clever, quick hack for writing parsers. Both the LCF/ML parser (Malcolm Newey) and the HOPE parser (MacQueen) were written using his technique, partly because none of us knew about parser generators at that time, and it seemed much easier than writing a recursive descent parser. The drawback was that it was hard to use symbols for more than one purpose, hence the ".", ";", ";;" separators!

Wegner 1985), and "Typeful Programming" (Cardelli 1989).

In the fall of 1985 Luca moved from Bell Labs to DEC SRC in California, where his language research did not slow down at all! During the next decade he developed Quest, Fsub, and Obliq, and collaborated on the design of Modula 3. He also became a major player in research on type systems for object-oriented languages, and participated in the ML2000 discussions. But I will leave the story of those accomplishments to other contributors to this celebration.

5 Summary

Luca had the vision to see ML as a successful general purpose language, and he had the energy and talent to pursue this vision by designing and implementing the first free-standing version of ML. He followed that accomplishment up by becoming a key participant in the collective process of designing and prototyping Standard ML.

6 Acknowledgements

I want to thank Luca Cardelli, Bob Harper, Kevin Mitchell, Alan Mycroft, Mike Gordon, and Larry Paulson for useful recollections, helpful answers to my many questions and some bits of useful documentation.

References

R. M. Burstall and R. J. Popplestone. Pop-2 reference manual. In E. Dale and D. Mitchie, editors, *Machine Intelligence 2*, pages 207–46. University of Edinburgh Press, 1968.

R. M. Burstall, D. B. MacQueen, and D. Sannella. HOPE: an experimental applicative language. In *Conference Record of the 1980 Lisp Conference*, pages 136–143, August 1980.

L. Cardelli. The ML abstract machine. Early description of the ML abstract machine (AM), precursor to the FAM., November 1980a.

L. Cardelli. A module exchange format. Description of a textual format for exporting internal ML data structures., December 1980b.

L. Cardelli. Differences between VAX and DEC-10 ML. file mlchanges.pdf, 1981.

L. Cardelli. Edinburgh ML. README file for VAX ML (ML under VMS) distribution., March 1982a.

L. Cardelli. *An Algebraic Approach to Hardware Description and Verification*. PhD thesis, University of Edinburgh, April 1982b.

L. Cardelli. Known vax-ml system locations. *Polymorphism: The ML/LCF/Hope Newsletter*, 1(0), November 1982c.

L. Cardelli. *ML under VMS*. Department of Computer Science, Univ of Edinburgh, 1982d.

L. Cardelli. The Functional Abstract Machine. *Polymorphism: The ML/LCF/Hope Newsletter*, 1(1), January 1983a.

L. Cardelli. Stream input/output. *Polymorphism: The ML/LCF/Hope Newsletter*, 1(3), December 1983b.

L. Cardelli. *ML under Unix*. Bell Labs, August 1983c. Pose 2.

L. Cardelli. Compiling a functional language. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, pages 208–217, New York, NY, USA, 1984a. ACM.

L. Cardelli. *ML under Unix*. Bell Labs, April 1984b. Manual for ML under Unix, Pose 4.

L. Cardelli. A semantics of multiple inheritance. In G. Kahn, D. MacQueen, and G. Plotkin, editors, *International Symposium on Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 479–504, June 1984c.

L. Cardelli. Basic polymorphic typechecking. *Polymorphism: The ML/LCF/Hope Newsletter*, 2(1), January 1985.

L. Cardelli. Amber. In *Proceedings of the Thirteenth Spring School of the LITP on Combinators and Functional Programming Languages*, volume 242 of *Lecture Notes in Computer Science*, pages 21–47, 1986a.

L. Cardelli. The amber machine. In *Proceedings of the Thirteenth Spring School of the LITP on Combinators and Functional Programming Languages*, volume 242 of *Lecture Notes in Computer Science*, pages 48–70, 1986b.

L. Cardelli. Typeful programming. Technical Report 45, Digital Systems Research Center, May 1989.

L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, December 1985.

L. Damas. *Type Assignment in Programming Languages*. PhD thesis, Department of Computer Science, University of Edinburgh, 1985.

M. J. Gordon, A. J. Milner, and C. P. Wadsworth. *Edinburgh LCF*, volume 78 of *LNCS*. Springer-Verlag, New York, 1979.

M. J. C. Gordon. Locations as first class objects in ML. Note to Luis Damas proposing a research topic., 1980.

R. Harper. Report on the Standard ML Meeting, Edinburgh, May 23-25, 1985 (DRAFT). Draft minutes of the May, 1985 Standard ML meeting., 1985a.

R. W. Harper. Standard ML input/output. *Polymorphism: The ML/LCF/Hope Newsletter*, 2(1), January 1985b.

K. Jensen and N. Wirth. *Pascal User Manual and Report*. Springer-Verlag, 2nd edition, 1978.

D. MacQueen. Modules for Standard ML. *Polymorphism: The ML/LCF/Hope Newsletter*, 2(2), October 1985.

D. MacQueen and R. Milner. Report on the Standard ML Meeting, Edinburgh, 6-8 June 1984. *Polymorphism: The ML/LCF/Hope Newsletter*, 2(1), January 1985.

D. B. MacQueen. Structure and parameterization in a typed functional language. In *Proc. 1981 Symposium on Functional Languages and Computer Architecture*, pages 524–538, June 1981. Gothenburg, Sweden.

D. B. MacQueen. Modules for Standard ML. In *Proceedings 1984 ACM Symposium on LISP and Functional Programming*, pages 198–207, August 1984.

R. Milne. *The Formal Semantics of Computer Languages and their Implementations*. PhD thesis, Oxford University, 1974.

R. Milner. A Proposal for Standard ML (TENTATIVE). first manuscript draft of the Standard ML Proposal, April 1983a.

R. Milner. Changes to Proposal for Standard ML. first manuscript draft of the Standard ML proposal, May 1983b.

R. Milner. A Proposal for Standard ML (second draft). second manuscript draft of the Standard ML proposal, June 1983c.

R. Milner. A Proposal for Standard ML. Technical Report CSR-157-83, Dept of Computer Science, Univ of Edinburgh, December 1983d.

R. Milner. A Proposal for Standard ML. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, pages 184–197, New York, NY, USA, 1984. ACM.

K. Mitchell and R. Milner. Proposal for I/O in Standard ML. draft of an I/O proposal, February 1985.

V. R. Pratt. Top down operator precedence. In *POPL '73: Proceedings of the 1st annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Langauges*, pages 41–51, 1973.

N. Saito. ML System on Vax Unix. README for Saito's Unix port of VAX ML, March 1982.

R. W. Witty and C. P. Wadsworth. ML, LCF, and HOPE. Record of a meeting about the future of ML, LCF, and HOPE at Rutherford Appleton Laboratory, November 1982.