Toward a standard Standard ML

We would like to begin seriously and systematically working on the problem of incompatible environments and libraries that is threatening to cause us all continuing grief and to retard the future development of Standard ML.

History

Unfortunately, the consideration of what should be in the "initial static and dynamic basis" (i.e. pervasive environment) occurred at the tail end of the Standard ML design discussions. Bob Harper did a fairly careful job on a proposal for basic I/O facilities. But, perhaps because we were tired or out of time, the rest of the initial basis was chosen on a rather casual basis. For instance, I have no recollection of any discussion leading to a rational decision that "map" belonged in the initial basis but "app" or "fold" did not. Types and functions that were available and commonly used in the Edinburgh compiler were left out. Even in the minimal set of functions provided there were naive mistakes such as the choice of exceptions for the arithmetic operators and designation of append (`@`) as left-associative.

It was clear to me that any realistic implementation would have to provide more than this minimal basis, and in retrospect it would have been a good idea to devote another year to working on the design of the basic environment. However, this did not happen; the Definition was published with this initial basis and it became very hard make any corrections.

Any serious implementation of Standard ML must go beyond the specified initial basis, and since there is no guidance on how this should be done and little coordination, the basis has been extended incompatibly by the three major implementations (Poly/ML, SML of NJ, and POPLOG ML). The result is the current mess, manifested recently by the argument about the type and semantics of substring. It is our task to do what can be done to reduce the confusion and resulting porting difficulties.

Issues

There are at least three related problems:

1. What environment should be used for writing portable code, and how should that environment be made available?

2. What should be bound in the default top-level environment (what we call the "pervasive" environment)?

3. What library modules should be available in all implementations?

The basic goal is to be able to configure each implementation so that it presents a standard environment for compiling source code conforming to certain simple guidelines.

We want to achieve this without fragmenting and unnecessarily complicating the environments. For instance, we want to avoid having to look for basic string functions in several different places, e.g. the pervasive environment, a basis structure, and one or more string library modules. We also want to avoid having to sacrifice efficiency for the sake of portability.

Andrew has made a proposal to define a "Standard" structure ("Base" or "Basis" might be a better name). This structure would contain (at least) critical primitives needed to define efficient library modules in an "implementation independent" way. Components would only be added to the Standard module if there was a consensus on their name, type, and semantics. If an adequate Standard structure could be established, it should be possible to implement a library such as the Edinburgh Library solely in terms of that structure, allowing it to compile on all SML systems without change. This would eliminate the need for implementation-specific versions of the library source code.

One thing that must be decided is the precise role of the `Standard` module. Should it be restricted to only isolated operations, such as substring, that require compiler support to be implemented or implemented efficiently? Or should it be as a minimum portability base supplied by each compiler, something of a alternative to the pervasives? A closely related question is: what should be the relation between the `Standard` structure and the pervasive environment? There are at least three possibilities:

- The pervasive environment can be defined in terms of the `Standard` structure. This might mean that the elements of the pervasive environment are all contained in the `Standard` structure (or its substructures), or they may just be definable (in ML) in terms of the `Standard` structure.
- The `Standard` structure and the pervasive environment are complementary and have no elements in common.
- The `Standard` structure and the pervasive environment overlap, containing elements in common without either being definable in terms of the other.

It may be possible to give a "reference implementation" of large parts of the `Standard` structure in terms of the Definition's initial basis, but this implementation would not be acceptably efficient (e.g. the definition of substring in terms of explode).

One thing the `Standard` structure cannot provide is overloaded identifiers. Nor can it provide infix operators (assuming that `Standard` has a signature and infix specifications are remain excluded from signatures). So providing for overloadings and infixes is the minimum necessary role of the pervasive environment.

Another and more controversial issue is the relation between the pervasive environment and the initial basis described in Appendices C and D of the Definition. We could choose to minimize the pervasive environment, making it a subset of the Definition's initial basis by moving nonoverloaded, noninfix value bindings into structures. We could make the pervasive environment be a superset of the initial basis, adding at least the `Standard` structure binding. Or we might insist that the pervasive environment be identical with the Definition's initial basis. In the later case, some special mechanism would have to be provided to get access to necessary extensions.

Among the libraries based on `Standard` would be ones providing most of the current implementation-dependent extended environments (e.g. NJ, Poly, POPLOG). These would support existing application code, and would also make it possible for each compiler to mimic the others to some extent. Of course, it is likely that the different compilers will continue to provide unique and nonportable libraries (e.g. signals, continuations, and concurrency in SML of NJ), but implementations should insure that these do not get in the way when one wants to write portable applications or libraries. Future versions of SML of NJ will provide new facilities that will give programmers more explicit control of the compilation environment for this purpose.

We will also have to address certain technological issues about how environments are managed and how sophisticated the loading facilites are. For instance, large library modules are more acceptable if a loader can selectively load only the functions needed.

Strategy

There are various ways we might approach the problem, but I suggest that the most efficient method is to start with a concrete, if incomplete, proposal for the `Standard` structure and try to come to agreement one component at a time. In order to insure reasonable progress we might agree on a time table for the negotiations. For instance, it would be useful to have an adequate `Standard` structure specified by the time of next June's ML workshop in San Fransisco.

As a basis for discussion, we have the current extended environments provided by SML of NJ, Poly/ML, and POPLOG ML, as well as the Edinburgh library. Unfortunately, we do not have access to the Poly/ML or POPLOG ML systems, so it would be very helpful if you could supply us with complete descriptions of your environments. It would also be worth considering libraries for other languages such as CAML, Common Lisp, Scheme, and Modula-3.

One of our first tasks should be to set out some general guidelines or principles for designing the standard environment (e.g. to decide between curried vs uncurried versions of functions). It is clear that one such guideline is to provide algorithmically efficient primitives (e.g. a constant time rather than linear time substring operation). Another issue requiring guidelines is where to introduce exceptions and how to name them.

John Reppy has drafted a very preliminary proposal for a `STANDARD` signature, with a reference implementation structure `Standard : STANDARD` to specify the semantics. We'll send this out for your information in a separate message.

To keep the process manageable, I feel that it would be best to conduct discussions among the implementors via email. At appropriate points we can circulate proposals to a wider audience via the sml-impl list or the sml-list. It would of course be most productive to conduct the discussion with as little flaming as possible. Here is the proposed list of participants:

Standard ML of New Jersey:

Andrew Appel `appel@princeton.edu`
Emden Gansner `erg@ulysses.att.com`
Lal George `george@research.att.com`
Dave MacQueen `macqueen@research.att.com`
John Reppy `jhr@research.att.com`

Poly/ML:

Mike Crawley `mjc@abstract-hardware-ltd.co.uk`
Simon Finn `simon@abstract-hardware-ltd.co.uk`
Mike Fourman `mikef@lfcs.edinburgh.ac.uk`
Dave Matthews `dcjm@computer-lab.cambridge.ac.uk`

POPLOG/ML:
R. J. Duncan `robd@cogs.sussex.ac.uk`
Simon Nichols `simonn@cogs.sussex.ac.uk`

Harlequin:

Nick Haines `nickh@harlequin.co.uk`

Edinburgh:

Dave Berry `db@lfcs.ed.ac.uk`

Could each implementation group please respond with your comments on the issues and the strategy I have described above? I hope you will decide to join us in this effort, since we will all benefit if we are successful.