Compilers for Standard ML'97

Stephen Gilmore

June 18th 1998

Caveats

I compiled the PEPA Workbench Version 0.7 on a number of SML'97 compilers and measured the compilation time and the run time of the compiled code, averaging over a number of runs. All timings, however, are approximate. Except where noted, all compilations and runs were on polly.dcs.ed.ac.uk, a 32Mb Linux PC.

This shows only the differences in the compilers *for this program*, as they might be perceived by the author of the program and one of its users. A real benchmarking study would try the compilers on a number of SML programs of different kinds (FFT, Game of Life, Sorting, Mandelbrot, Boyer-Moore string searching, Knuth-Bendix completion). Plus, the PEPA Workbench is perhaps *not a typical ML program*.

The PEPA Workbench

- ✓ All new and improved!
- ✓ Certified 100% SML'97 compliant!
- ✓ Totally Basis Library enhanced!
- ✓ No foreign functions! No calling into C!
- ✓ Free from graphical user interfaces!
- ✓ Fully modular with opaque signature matching!
- ✓ Portable! Lightweight! It rocks!

The purpose of the PEPA Workbench is to convey a good impression of the PEPA language. It should be easy to install and use. It is distributed via the PEPA Web page at http://www.dcs.ed.ac.uk/pepa in the form of a standalone executable.

Workbench signatures

Lines	Name
22	Error.sig
19	Options.sig
34	HashTable.sig
14	Files.sig
26	Pepa.sig
17	Lexer.sig
11	Sort.sig
10	Parser.sig
12	Semantic.sig
12	Prettyprinter.sig
23	Table.sig
27	Statespace.sig
11	Report.sig
13	Derivatives.sig
9	Graph.sig
11	PWB.sig

Total code size
(including structure bodies):
about 1500 lines

Typical use of the Workbench

A typical user of the PEPA Workbench would make a PEPA model of an application, compile it with the Workbench and process the results with a package such as Maple or Matlab.

- typical example: 100 states*
- large example: 17,000 states†

The PEPA Workbench builds the graph of the model and writes this out to two files. For a large example (Holton's) the files are quite large (6Mb and 2Mb). For some of its processing the Workbench manipulates some parts of the graph as strings.

I compiled the PEPA Workbench with the various SML '97 compilers and ran it on Holton's example.

* *Analysis of a multimedia stream using stochastic process algebra*, Bowman, Bryans and Derrick.

† *A PEPA specification of an industrial production cell*, Holton.

Compiler 1: Moscow ML

I used Moscow ML version 1.43 from April 1998. I separately compiled the Workbench signatures and structures, which were then linked together. (I used a Makefile.)

Moscow ML supports only unnested structures and no functors. Moscow ML enforces a naming convention on files with one compilation unit stored in each file. It warned me about having a signature file which was not used.

Moscow ML compile time: 10 seconds

PEPA Workbench run time: 7 minutes

Peter Sestoft writes: *Moscow ML generates bytecode and spends little time trying to optimize it – hence the compiled programs are not so fast.*

Compiler 2: SML of New Jersey

I used Standard ML of New Jersey version 110 of January 1998 with the Compilation Manager loaded. I compiled the Workbench with CM.make.

SML/NJ compile time: 10 seconds

PEPA Workbench run time: 7 minutes

Compiler 3: SMLC

I used SMLC of June 10th, 1998. SMLC is a 'whole program optimising compiler' for Standard ML '97. It is being developed by Stephen Weeks at NEC. It conforms very closely to the revised Definition (there are no higher-order functors, no weak or imperative type variables, no local functors or local signatures, no **where** structure specifications)

Unlike many SML compilers, SMLC has no separate compilation and no interactive top-level loop. SMLC compiles to C which is then compiled with gcc. It is a batch compiler which runs on a whole program, like an old-fashioned Pascal compiler. SMLC is only available for Linux.

SMLC compile time: 32 minutes

SMLC compilation to C: 50 seconds

PEPA Workbench run time: 11 minutes

SMLC: optimisation, error messages

One of the optimisations which is performed involves specialisation of each functor use. Because of this SMLC accepts a superset of valid SML programs, because it expands away all functors before type inference.

Stephen Weeks writes: *At present SMLC should only be run on valid SML programs. There is quite a bit of error-checking that is not done. If you feed SMLC an invalid program, it may do any of the following: terminate and produce an executable, fail to terminate, or produce a terse error message.*

```
Error: unify Con
```

Compiler 4: MLJ

I used Persimmon MLJ 0.1 of May 1998. MLJ compiles to Java bytecodes, writing a zipped class file as its output. It compiles "approximately everything except for functors" and has "automatic recompilation management" and "whole-program optimisation to produce compact compiled code."

Persimmon writes: *MLJ brings the benefits of Java's Write Once, Run Anywhere™ cross-platform bytecode to Standard ML. Applications written using MLJ can be delivered as applications or as web applets and run unchanged on any computer with a compliant Java Virtual Machine.*

Once compiled with MLJ, an ML application can be run with a Java bytecode interpreter such as SUN's **java** or Microsoft's JIT compiler **jview**. I used the command

```
java -classpath $MLJPATH:MLApp.zip pwb
```

MLJ: language and compilation

The MLJ language allows the programmer to mix ML and Java.

```
let
  type colour = "java.awt.Color"
  val red = valOf(_getfield colour "red")
in
  Java.toInt (_invoke "getCyan" (red))
end
```

MLJ uses typed internal languages including MIL (Monadic Intermediate Language) to which monadic transformations are applied. “The initial translation from typed SML to MIL is essentially Moggi’s call-by-value translation.[‡]” Compilation units are compiled into MIL terms which are then linked together and optimised upon the basis of effect inference before being turned into basic blocks which are compiled down to Java class files.

MLJ specialises polymorphic functions at each of the (Java equivalent) types where it is used.

[‡] *Compiling Standard ML to Java Bytecodes*, Benton, Kennedy and Russell.

MLJ: Performance

Benton, Kennedy and Russell write: *MLJ is still a work-in-progress . . . , but it is already quite usable on source programs of several thousand lines and produces code which, with a good JVM, often outperforms Moscow ML, a popular SML bytecode interpreter which is certainly regarded as fast enough for many real applications.*

MLJ compile time: 1 minute

PEPA Workbench run time: 12 hours 12 minutes!!

I re-ran the Workbench on hope.dcs.ed.ac.uk (64Mb Linux PC) and the run time reduced to 52 minutes.

Compiler 5: ML Works 1.0r2

I used Harlequin's MLWorks 1.0r2 of 1997 on scar.dcs.ed.ac.uk, where the license resides.

ML Works compile time: 10 seconds

PEPA Workbench run time: 6 minutes

Table of results
Moscow ML (10 secs; 7 mins)
SML/NJ (10 secs; 7 mins)
SMLC (32 mins; 11 mins)
MLJ (1 min; 52 mins[§])
ML Works (10 secs; 6 mins[¶])

[§]on hope.dcs.ed.ac.uk, because of the need for more real memory

[¶]on scar.dcs.ed.ac.uk, because of the license manager

The ones that got away . . .

- ML Kit Version 3
- TIL/ML or TILT/ML
- FLINT/ML

I may make some improvements to the PEPA Workbench's data structures and try it again on these compilers and others.