Working with SML/NJ

Anya Tafliovich

1 SML/NJ (on mathlab)

1. How do I start SML/NJ?

   On mathlab type `sml`:

   ```
   mathlab:$ sml
   Standard ML of New Jersey v110.65 [built: Wed Oct 31 17:24:44 2007]
   -
   ```

   Typing `rlfe sml` instead of `sml` will save you lots of pain. Read `rlfe`'s man pages (type `man rlfe`) to learn about `rlfe`.

   The '-' is the SML/NJ's prompt. Now you can type in ML code and the interpreter immediately evaluates it and produces an output.

2. How do I quit SML/NJ?

   Press `Ctrl-D`.

3. How do I load a file?

   The function `use` is defined at top level and will load a file containing SML source code.

   For example, loading a file named `myfile.sml`:

   ```
   - use "myfile.sml";
   [opening myfile.sml]
   val it = () : unit
   -
   ```

   What is `val it = () : unit` ? The function `use` returns `()` of type `unit`.

   If you are are loding a file that is in a directory different from the one you started smlnj in, then you need to provide a full path to the file.

   Note the semicolon at the end: to SML/NJ, it means "the user is done tying, now I should start working".

4. What do I use for comments?

   Use `(* comments are here *)`. The comment can be multi-line.

5. Note: SML is case sensitive.

6. SML interaction:

   Input to the top level interpreter (i.e., declarations and expressions) must be terminated by a semicolon (and carriage return). Expressions are treated as implicit declarations of a standard variable `it`. For example,

```
- 3;                       (* user input after prompt *)
val it = 3 : int           (* system response *)
```

This means that the value of the last expression evaluated can be referred to using the variable `it`. For example,

```
- 2.5;
val it = 2.5 : real
- it;
val it = 2.5 : real
-
```

7. Multi-line expressions:

   If you type a long expression, you can enter it in several lines. In this case the prompt for subsequent lines changes to `=`. You end the expression with a semicolon.

   ```
     - if 10 > 5
     = then 1
     = else 2;
     val it = 1 : int
   ```

8. Interrupting:

   Typing `Ctrl-C` should interrupt the interpreter and return you to top level (useful to interrupt infinite recursion).

9. Error messages:

   The error messages include line numbers and character positions within the line. For example:

   ```
   - if true
   = then 5 true
   = else 6;
   stdIn:6.6-6.12 Error: operator is not a function [literal]
     operator: int
     in expression:
       5 true
   -
   ```

   This means: there is an error between line 6, character 6 and line 6 character 12. The error is: I am trying to apply `5` to `true`, and I can't because `5` is not a function.

10. More error messages:

    There are a number of different forms of type error message, and it may require some practice before you become adept at interpreting them. The most common form indicates a mismatch between the type of a function (or operator) and its argument (or operand). A representation of the offending expression is usually included, but this is an image of the internal abstract syntax for the expression and may differ significantly from the original source code.

    A very useful reference for a list of all error messages produced by SML/NJ can be found here:

    `http://www.smlnj.org/doc/errors.html`

11. Debugging in SML/NJ:

    There is no debugger installed with SML/NJ. SML is a strongly typed language and its type inference system is powerful enough to capture a wide variety of errors and ambiguities. Furthermore, there is run-time checking so there's no way you can crash the system.

12. What editor should I use to develop sml code?

    Use your favourite editor. I use XEmacs, which has a nice sml mode, which does things like syntax highlighting, auto-indentation, etc. and allows you to run sml in the emacs buffer. If you've never used Emacs before, it is quite a bit of a learning curve. I think it's worth it, but it's just my opinion.

    `http://www.xemacs.org/` `http://www.xemacs.org/Documentation/packages/html/sml-mode_toc.html#SEC_Contents`

2 More or less useful things...

Suppose we have written the following ML program in file `sum.sml` to compute the sum of elements of a list.

Note that the line numbers are here for your convenience, they are not actually in the file.

```
1. (* sumlist L: returns the sum of the elements of L.
2. *
3. fun sumlist([])   = 0;
4.   | sumlist(h::t) = h + sumlist t;
```

Now, let's load the file in SML and test it:

```
- use sum.sml;
stdIn:1.5-1.12 Error: unbound structure: sum in path sum.sml
```

The file name should be a string:

```
[opening sum.sml]
sum.sml:1.1-5.2 Error: unclosed comment
val it = () : unit
```

There is an error in the file. I forgot to close the comment:

```
1. (* sumlist L: returns the sum of the elements of L.
2.  *)
3. fun sumlist([])   = 0;
4.   | sumlist(h::t) = h + sumlist t;
```

```
[opening sum.sml]
sum.sml:3.5-3.22 Warning: match nonexhaustive
          nil => ...

val sumlist = fn : 'a list -> int
sum.sml:4.3 Error: syntax error: replacing  BAR with  EQUALOP

uncaught exception Compile [Compile: "syntax error"]
  raised at: ../compiler/Parse/main/smlfile.sml:15.24-15.46
             ../compiler/TopLevel/interact/evalloop.sml:44.55
             ../compiler/TopLevel/interact/evalloop.sml:296.17-296.20
```

I have an extra ";" at line 3. So I've defined a function `sum` that returns `0` on input `[]`. I'm warned that not all lists look like `[]`. I then have line 4, which by itself is invalid syntax.

```
1. (* sumlist L: returns the sum of the elements of L.
2.  *)
3. fun sumlist([])   = 0
4.   | sumlist(h::t) = h + sumlist t;
```

```
- use "sum.sml";
[opening sum.sml]
val sumlist = fn : int list -> int
val it = () : unit
```

Great! Finally! Now I need to test it.

```
- sumlist([]);
val it = 0 : int
- sumlist([42]);
val it = 42 : int
- sumlist([1, 2, 3]);
val it = 6 : int
- sumlist([-1, 2, 3]);
stdIn:5.10 Error: expression or pattern begins with infix identifier "-"
stdIn:5.9-5.19 Error: operator and operand don't agree [literal]
  operator domain: 'Z * 'Z
  operand:         int
  in expression:
    - 1
-
```

What's wrong with the last expression? `-` is a binary infix operator. I need to use the unary `~`.

```
- sumlist([~1, 2, 3]);
val it = 4 : int
- sumlist([~1.0, 2.0, 3.1]);
stdIn:1.1-6.5 Error: operator and operand don't agree [tycon mismatch]
  operator domain: int list
  operand:         real list
  in expression:
    sumlist (~1.0 :: 2.0 :: 3.1 :: nil)
-
```

The argument to `sum` must be an `int list` and I'm giving it a `real list`. But you knew that.