CS235 Languages and Automata **Handout # 2**
Prof. Lyn Turbak **September 17, 2007**
Wellesley College

Standard ML of New Jersey (SML/NJ) and Forlan

The dialect of the ML programming language that we will be using in CS235 this semester is Standard ML (also known as SML). There are several implementations of SML; we will be using Standard ML of New Jersey (SML/NJ). This handout contains notes on how to run SML/NJ on the Linux workstations. It also has a few notes on the Alley Stoughton's Forlan system, which runs on top of SML/NJ.

In the SML/NJ programming environment, you will use Emacs to create program files and will load these files into an interactive SML/NJ top-level interpreter that executes in a special Emacs buffer. Type inference is performed on all SML programs and you cannot test your programs until they pass the type checker. Understanding error messages from the type checker and using them to pinpoint type errors in your program are important skills that you will need to hone.

There are two ways to run SML/NJ on the Linux workstations: within a shell window and within Emacs. These approaches are described below in Secs. 1 and 2. It is recommended that you use the Emacs interface, since it simplifies many interactions. Nevertheless, you should still read Section 1 as many of the notes still apply to the Emacs interface. Sec. 3 describes the few extra details you need to run Forlan.

Note that you can run SML/NJ remotely on the CS dept. Linux machines using `ssh`. So you needn't be in the Linux lab to do your SML programming.

This handout only scratches the surface of SML/NJ. For more detailed documentation, browse the official SML/NJ web page:

```
http://www.smlnj.org
```

By the way, SML/NJ also runs on PCs and Macs. See the above web page for details. In order to use install and use Forlan, you must use version 100.65 or above of SML/NJ.

1 Running SML/NJ in a Shell Window

1.1 Launching the SML/NJ interpreter

The simplest way to run SML/NJ on the Linux workstations is within a Linux shell window. Within such a window, execute `sml`. This will print a herald on the screen and eventually the '`-`' prompt of SML/NJ.

```
[fturbak@puma ~] sml
Standard ML of New Jersey v110.65 [built: Wed Sep 12 02:33:09 2007]
```

You can now evaluate expressions by typing them in followed by a semi-colon and a line return. The semi-colon tells the interpreter that you are done with the expression. This allows you to have expressions with multiple lines; SML/NJ will show a '`=`' at the beginning of every line of a multiple line expression. For example, Fig. 1 presents a transcript of a session in which two single line expressions and one multiple line expressions are evaluated.

If you forget the semi-colon at the end of an expression, SML will think you are continuing the expression onto the next line. In this case, you can just type a semi-colon followed by a line return to indicate that you are done. For example:

```
- 2 + 3
= ;
val it = 5 : int
```

```
- 1 + 2;
val it = 3 : int
- val lst = map (fn x => x * 2) [4,7,3];
val lst = [8,14,6] : int list
- let val a = 3+4
=     val b = 5*a
=  in (a,b)
= end;
val it = (7,35) : int * int
```

Figure 1: A transcript showing the evaluation of three SML expressions.

The SML interpreter expects that the unit of evaluation will be a declaration — either a value declaration of the form `val` *name* `=` *exp* or a function declaration of the form `fun` *name* var_1 ... var_n `=` *exp*. If you just enter an expression E, the SML interpreter treats it as a declaration of the form `val it =` E.

1.2 Loading Files

It is tedious to type all expressions directly at the SML interpreter. It is especially frustrating to type in a long definition only to notice that you made an error near the beginning and you have to type it in all over again. In order to reduce your frustration level, it is wise to use a text editor (e.g., Emacs) to type in all but the simplest SML definitions. This way, it is easy to correct bugs and to save your definitions between different sessions with the SML interpreter. (Note: the file extension that you should use for SML files is `.sml`. Using this extension will enable various SML features in Emacs. See Sec. 2.1 for details.)

If *filename* is the name of a file containing SML expressions and definitions, evaluating

```
use "filename";
```

will evaluate all of the expressions in the file, one by one, as if you had typed them in by hand. The `use` function can be used within a file to load other files.

The filename given to `use` may be either an absolute pathname (such as the absolute pathname `/students/gdome/cs235/sml/test.sml`) or a pathname relative to the current working directory. By default, the current working directory for the SML interpreter is the current working directory of the shell in which it was invoked, and by default this is your `puma` home directory (e.g. `/students/gdome`). So rather than evaluating

```
use "/students/gdome/cs235/sml/test.sml"
```

you could instead evaluate

```
use "cs235/sml/test.sml"
```

It is typical to load many files (or the same file many times) from the same directory. For this reason, it is useful to be able to change the current working directory within the SML interpreter. To do this, evaluate

```
Posix.FileSys.chdir("dirname")
```

where *dirname* is the name of the directory which you want to become the new current working directory. (This name can either be an absolute pathname, or relative to the current working directory.) For example:

```
Posix.FileSys.chdir("cs235/sml")
```

It's easy to lose track of what the current working directory is. The `Posix.FileSys.getcwd()`

command can be used to tell you what the current working directory is (as an absolute pathname). For example:

```
- Posix.FileSys.getcwd();
val it = "/students/gdome/cs235/sml" : string
```

1.3 Exiting SML

To terminate your session with the SML interpreter, type `C-d` (i.e., control-D). (It is common in Unix systems for `C-d` to represent the "end of input".)

2 Running SML within Emacs

You could do all of your SML programming in CS235 using just the techniques outlined in Sec. 1 above. However, you will find yourself constantly swapping attention between the Emacs editor (where you write your code) and the SML interpreter (where you evaluate your code). In particular, whenever you make a change to your Emacs file, you will have to save the file and reload it into the SML interpreter.

To reduce the overhead of swapping between Emacs and SML, you can run SML within Emacs. The following subsections outline the key aspects of running SML within Emacs.

2.1 Emacs Mode

An Emacs buffer can be in various modes. Each mode tells the buffer how keystrokes in that buffer should be interpreted. The most useful mode for editing SML code is SML mode. You can tell a buffer to enter SML mode by typing `M-x sml-mode`. (Notational convention: `M-x`, pronounced "Meta x", is entered by typing the `Meta` key and `x` key at the same time. On PC keyboards, the key labeled `Alt` is usually treated as the `Meta` key. On keyboards without a true `Meta` key, you can instead press the `ESC`ape key followed by pressing the `x` key.) You can tell that a buffer is in SML mode by the `(SML)` tag in the status line at the bottom of Emacs.

Whenever you edit a file that ends in `.sml`, Emacs will automatically enter SML mode for you. For this reason, it is wise to use the `.sml` extension for all of your SML files.

SML mode helps you write SML code because it understands the formatting conventions for SML code. Like many Emacs modes, it helps you match parentheses by flashing the matching open parenthesis whenever you type a close parenthesis. Addtionally, SML mode helps you put your code in pretty-printed format. Typing the `TAB` key will indent the code on that line according to the SML indentation conventions. You should get into the habit of hitting `TAB` after every return so that you start typing the next line at the appropriate indentation level. You can format an entire expression by typing `ESC C-q` when the cursor is at the first character of the expression. Keeping your SML expressions indented properly is important for reading and debugging code. Indenting the code will often highlight that the structure of the expression has a bug.

SML mode also understands what SML keywords are, and will color keywords, strings, and comments differently from the rest of the text to highlight them.

2.2 Launching SML in Emacs

To start an SML session within Emacs, execute the Emacs command `M-x sml`. This will create a buffer named `*sml*` that will be the location of the SML interpreter. In order to see this buffer, you will need to visit it by typing `C-x b *sml*`.

2.3 Interacting with the *sml* Buffer

The simplest way to interact with the *sml* buffer is to visit the buffer and enter expressions at the prompt. But there are a number of M-x commands that simplify interactions. For instance, M-x sml-cd will prompt for a directory name and change the current working directory to that directory; this is much more convenient than using Posix.FileSys.chdir. And a number of M-x commands will send text from another buffer in SML mode to the *sml* buffer.

M-x command	Keystroke Abbreviation	Description
sml-send-buffer	C-c C-b	Sends the contents of the entire buffer to the *sml* buffer as if you had typed it directly into the *sml* buffer.
sml-send-function		Sends the text of the current declaration to the *sml* buffer as if you had typed it directly into the *sml* buffer.
sml-send-function-and-go		Like sml-send-function except that it also displays the *sml* buffer and moves the cursor there.
sml-send-region	C-c C-r	Sends the text of the current region to the *sml* buffer as if you had typed it directly into the *sml* buffer.
sml-send-region-and-go		Like sml-send-region except that it also displays the *sml* buffer and moves the cursor there.

2.4 Exiting SML in Emacs

You can exit SML in Emacs by killing the *sml* buffer (using C-x C-k).

3 Forlan

In CS235 this semester, we will be using SML modules from Alley Stoughton's Forlan project. Invoking the forlan command in Linux launches a version of the SML/NJ interpreter in which the Forlan modules have been pre-loaded. For example:

```
[fturbak@puma ~] forlan
Standard ML of New Jersey Version 110.65 with Forlan Version 2.8 loaded
val it = () : unit
- val r = SymRel.fromString "(a,b), (b,c), (a,c)";
val r = - : sym_rel
- SymRel.symmetric r;
val it = false : bool
- SymRel.transitive r;
val it = true : bool
```

The Forlan modules are described in Stoughton's textbook. This and other Forlan documentation can be found at:

```
http://people.cis.ksu.edu/~stough/forlan
```

As with a regular SML session, it is possible to run a Forlan session in a separate Emacs buffer rather than in a shell. To do this, within Emacs type C-u M-x sml. You will be prompted for an ML command; type forlan. You will then be prompted for Any args; press the Enter key. This sequence of steps will create a new Emacs buffer named *forlan* for your Forlan session.