CS251 Programming Languages **Handout # 31**
Prof. Lyn Turbak **April 10, 2001**
Wellesley College

SML Exercises

This handout contains some simple exercises to familiarize you with using and programming in SML.

1 Preliminaries

Here are the steps you need to follow to use SML:

1. In a shell in the `~/cs251` directory, perform a `cvs update -d` to grab all relevant files.
2. Configure your `~/.emacs` file to interface properly with SML by adding the code in Figure 1. You need not type in the code; you can find it in `~/cs251/sml/emacs.txt`. You only need to update your `~/.emacs` file once, not every time you want to run SML.

   Once you have added the above lines to your `~/.emacs` file, exit Emacs and relaunch it so that your changes will take effect. You will need to relaunch Emacs before going on to step 3.
3. Start SMLNJ within Emacs by typing `M-x sml` ENTER.
4. Go to the SMLNJ interpreter buffer via `C-x b *sml*` ENTER.
5. In Emacs, change the default directory used by `sml` by typing `M-x sml-cd` ENTER *dir*, where *dir* is the name of the directory you wish to be the default directory for finding SML files. For this exercise, you *dir* to be `~/cs251/sml/test`.

2 The HOFL evaluator

Follow the steps below to use the SML implementation of the HOFL evaluator:

1. To install the HOFL evaluator, compile and load the SML implementation of the HOFL evaluator by evaluating the following `use` command in the SML interpreter:

   ```
   use("loadHoflEval.sml");
   ```

   Executing the above expressions will cause many lines of text to appear on the screen. Although some of the lines seem to indicate some sort of error, you can ignore these. Here's an example of something you can safely ignore:

   ```
   [checking ../sml/hoflemt/CM/x86-unix/Pretty.cm.stable ... not usable]
   ```

   You know that everything has compiled OK if the lines of text generated after `use` ends with the following:

```
(setq load-path
 (append '("/usr/share/emacs/20.3/lisp/sml-mode-3.3"
           "/usr/share/emacs/site-lisp/sml-mode-3.3")
          load-path))

(require 'sml-site)

(add-hook 'sml-load-hook '(lambda () (require 'sml-font)))

(setq auto-mode-alist
      (append auto-mode-alist
              '(
                ("\.itx$"  . scheme-mode) ;; INTEX
                ("\.bdx$"  . scheme-mode) ;; BINDEX
                ("\.ibx$"  . scheme-mode) ;; IBEX
                ("\.ffl$"  . scheme-mode) ;; FOFL
                ("\.fbs$"  . scheme-mode) ;; FOBS
                ("\.hfl$"  . scheme-mode) ;; HOFL
                ("\.hem$"  . scheme-mode) ;; HOFLEMT
                ("\.him$"  . scheme-mode) ;; HOFLIMT
                ("\.hep$"  . scheme-mode) ;; HOFLEPT
                ("\.hip$"  . scheme-mode) ;; HOFLIPT
                )))

(put 'program 'scheme-indent-hook 1)
(put 'abs 'scheme-indent-hook 1)
(put 'bind 'scheme-indent-hook 2)
(put 'bindpar 'scheme-indent-hook 1)
(put 'bindseq 'scheme-indent-hook 1)
(put 'bindrec 'scheme-indent-hook 1)
(put 'funrec 'scheme-indent-hook 1)
(put 'prepend 'scheme-indent-hook 1)
```

Figure 1: Code to add to your `.emacs` file for SML.

```
val it = () : unit
```

2. Try evaluating a simple HOFL program using `Eval.runString`, as shown below:

   ```
   Eval.runString "(program (a b) (div (+ a b) 2))" [3,5];
   ```

3. For larger programs, it is more convenient to write them in a file, and use `Eval.runFile` to evaluate them. Write a simple HOFL program named `hofl-test.hfl` in the directory `~/cs251/sml/test`, and evaluate it as follows:

   ```
   Eval.runFile "hofl-test.hfl" args;
   ```

   where *args* is an appropriate argument list for your program.

3 The HOFLEMT evaluator

Follow the steps below to use the SML implementation of the HOFLEMT type checker:

1. To install the HOFL type checker, evaluate the following in the SML interpreter:

   ```
   use("loadHoflemtTypeCheck.sml");
   ```

2. Try type checking a simple HOFL program using `TypeCheck.checkString`, as shown below:

   ```
   TypeCheck.checkString "(program (x) (+ x 1))";

   TypeCheck.checkString "(program (x) (if x 1 2))";
   ```

   The second example should give a type checking error because `x` (assumed to be an integer since it's a program parameter) is used as the boolean test expression in an `if`.

3. For larger programs, it is more convenient to write them in a file, and use `Eval.checkFile` to evaluate them. Write a simple HOFLEMT program named `hoflemt-test.hem` in the directory `~/cs251/sml/test`, and type check it as follows:

   ```
   TypeCheck.checkFile "hoflemt-test.hem";
   ```

4 Writing and Compiling SML code

1. Create a file named `~/cs251/sml/test/insert.sml` that contains the definition of SML function `insert` that takes two arguments (curried): (1) an integer and (2) a list of integers. Assume the second argument is sorted from low to high. The `insert` function should insert the first argument into the correct position within the sorted list and return the new list.

2. Create a configuration file named `~/cs251/sml/test/insert.cm` that contains the following lines:

```
Group is

  insert.sml
```

3. Compile your `insert` function by executing the following:

```
CM.make'("insert.cm");
```

   Fix any errors that are reported by the SML type checker.

4. Test your insert function in the SML interpreter on appropriate arguments.