Functional Programming I, 5.0 c

Course code: 1DL330, Report code: 11006, 33%, DAG, NML, week: 35 - 43 Semester: Autumn 2017

LAB 0: INTRODUCTION TO SML

This information is not available in English. Now showing the Swedish version.

This lab introduces you to the programming environment to develop Standard ML (SML) programs, and proposes several warm-up exercises to familiarize you with SML. **Please work on these exercises in pairs.**

The Programming Environment

In this course, we are using the Poly/ML compiler, version 5.5.2 (Uppsala modified 1). This is one of various implementations of the SML programming language. You can find this compiler on the Unix computers at the department. You can also install it on your own computer if you like.

INSTALLING POLY/ML ON YOUR OWN COMPUTER

You can find the web page of Poly/ML here. The official version has a small issue in that it does not accept certain non-English characters. If you want to be able to use, e.g., the Swedish vowels, please use the patched version available on the department computers. (That's what the "Uppsala modification" is about.)

TWO WAYS TO EDIT AND RUN SML PROGRAMS

You can either a) run Poly/ML under Emacs, or b) run Poly/ML directly, from the Unix command line. You should at least explore the first option!

a) Running under Emacs

A popular development environment for SML is Emacs. There is an Emacs mode to run SML compilers from within Emacs (download here for Emacs 23.x, or install it with `M-x list-packages` for Emacs 24.x and on). Please understand that we can only provide limited help for installation problems. (You can ask, but don't expect us to spend a long time solving your problem.)

The SML mode for Emacs greatly facilitates the development of SML programs. Among other things it provides syntax highlighting and several useful shortcuts. You can find the documentation of this mode here.

- Type **emacs myprog.sml &**
- Analyse the emacs toolbar: do you notice anything SML-related?
- Investigate the SML menu and learn how to start the ML compiler.
- When Emacs asks you for the "ML command", **replace sml with poly**.
- If you want Emacs to suggest "poly" by default, add the following lines to your ".emacs" configuration file (in your home directory, typically):

  (defun my-sml-mode-hook () "Local defaults for SML mode" (setq sml-program-name "poly"))
  (add-hook 'sml-mode-hook 'my-sml-mode-hook)

- Inside the Poly/ML window, you can use Ctrl-UPARROW or Alt-P to re-enter the last typed expression.

- Learn about the options provided in the SML menu and use them as much as you can.
- Make sure that you know
  - how to open and save files,
  - the basic editing commands,
  - how to load an SML file into the Poly/ML compiler, and
  - how to split the window in half so that you can edit a file in one half and interact with the SML system in the other.
- The Tab key indents the current line according to SML conventions.
- Ctrl-h m provides you with documentation of the SML mode.
- Ctrl-h i provides you with documentation of Emacs (and other Unix programs).

Try to play a bit with Emacs and the SML mode until you are comfortable with it.

b) Running from the command line

- Start Poly/ML in interactive mode: enter **poly** at the Unix prompt and see what happens.
- Use Ctrl-C to break an ongoing computation, and Ctrl-D to leave Poly/ML.
- Try to evaluate several simple SML expressions (one- and multi-line).
- To load an existing SML file, write **use "foo.sml";** at the Poly/ML prompt.
- Alternatively, enter **poly < foo.sml** at the Unix prompt.
- To get line-editing and history capabilities (where, e.g., UPARROW recalls the last command), enter **rlwrap poly** to start Poly/ML.

Other editors with SML syntax highlighting

A few other text editors have support for SML syntax highlighting. These include

- jEdit
- kate

The gedit editor offers syntax highlighting for Objective Caml, which has similar syntax to SML. (To activate it go to "View->Highlight mode->Sources->Objective Caml".)

Student pick: For Windows, previous students recommended the "Sublime Text 2" editor. Instructions can be found here: http://developerinmotion.wordpress.com/2013/02/07/sublime-text-2-and-sml/ Note that you should install Poly/ML instead of SML/NJ, and change "/usr/local/smlnj-110.75/bin/sml" to "C:/Program Files (x86)/Poly ML/polyml" in "Main.sublime-menu". Also note that we don't have any experience with installing or using this editor – you're wholly on your own if you want to try.

Warm-up Exercises

Once you are familiar with editing and executing SML code in your favourite environment, practice basic SML programming by solving the warm-up exercises below. You might also try examples presented in lectures or in the textbook.

You don't need to submit these exercises for correction, but if you want, you can show them to the teaching assistant during the lab for rapid feedback.

Do not despair if you cannot yet do everything. Do as much as you can. We understand that you may have different backgrounds and preliminary knowledge. Sometimes students have difficulties to get started. We

have seen many students that made a tremendous breakthrough from zero knowledge to mastering and loving FP in a short period of time. Do not hesitate to ask the teaching assistant for help or hints during the lab. **Not all exercises have been covered in lectures yet.**

If, on the other hand, you do not feel that these excercises are difficult enough for you, try to solve your own problem/puzzle using SML, or ask us to give you a problem to try. You can find a lot of fun programming problems to exercise on at Project Euler.

EXERCISES:

1. What are the types of the following SML expressions?
   - `(1.5,"3",7)`
   - `fn x => x*x`
   - `[1,2,3]`
2. What is wrong with each of the following SML expressions? If possible suggest appropriate corrections.
   - `1+2.0`
   - `10/3`
   - `-1`
   - `[1,"hello"]`
   - `[1,2.0]`
3. Give example values of the following SML types:
   - `string * int`
   - `real * (string * int)`
   - `int * int -> int`
   - `int list list`
4. Try to run a program from the textbook or the lecture notes.
5. Define a function ... (Think about how to handle corner cases where necessary.)
   1. ... that computes the cube of an integer.
   2. ... that takes three integers and returns the smallest.
   3. ... called `IF` such that `IF a b c` returns `b` if `a` is true and `c` otherwise.
   4. ... called `between` such that `between a b` returns an integer `x` such that `a < x < b` or `b < x < a`, if there is such a value, and `42` otherwise (e.g., `between 1 3 = 2`).
   5. ... called `pow` such that `pow x n` computes `x` to the power of `n` (e.g., `pow 2 3 = 8`)
   6. ... that returns the third element of a list.
   7. ... that computes the length of a list.
   8. ... that returns the largest element of a list of integers.