Open Notebook for Standard ML (SML)

2013-01-20 22:37

Contents

1 References

1.1 Standard ML

- Standard ML of New Jersey (SML/NJ)
- Moscow ML - Lightweight implementation of Standard ML
- Standard ML Wikipedia entry
- SML, Some Basic Examples - a good cheatsheet of simple examples, very good
- Basics of Standard ML - Michael P. Fourman - A nice basic 21 page introduction.
- SML Tutorial - University of Chicago - 34 page PDF presentation
- Tips for Computer Scientists on Standard ML - All of SML in a 20 page PDF
- A Gentle Introduction to ML - A nice online overview (about 55 pages)
- Standard ML Basis Library Documentation - documents the standard functions
- Programming in Standard ML 97 - Tutorial Introduction - Stephen Gilmore, Stephen Gilmore - 94 page PDF Book
- Notes on Programming Standard ML of New Jersey - Riccardo Pucella, Cornell University - 249 page PDF book
- Programming in SML - Robert Harper, Carnegie Mellon - 297 page PDF book
  - Programming in SML Example Source Code
- SML Style Guide from a Cornell course
- Using the SML REPL

1.2 GNU Emacs

- Emacs SML Editing Mode Documentation
- GNU Emacs Main Web Page
- GNU Emacs on-line documentation

1.2.1 GNU Emacs Reference Cards

These reference cards (except for the SML Mode one) comes with Emacs in the etc/refcards directory)

- GNU Emacs Reference Card source - Nice PDF cheat sheet on one double-sided page
- GNU Emacs Survival Card source - A basic overview
- GNU Emacs SML Mode Card source - The beginnings of an SML Mode reference card
- GNU Emacs Dired Card source - Directory mode reference card

1.3 Sublime Text 2

You can use the standard Sublime Text 2 Package Control system to add SML support:

- First, if you have not done so already, install the Package Contol package by going to Sublime Package Control
- Next install the Sublime REPL package using Package Control. Note that Sub,ime REPL does *not* support SML. We will fix that later
- Next install the SML Language Definitions again using Package Control
- Now we will add SML support to the REPL.
  - Go to the Default Sublime Commands Gist and download the 2 files there. These files will add support for SML in the SublimeREPL
  - To use, create a SML subfolder in Packages/User and drop these files inside. (You can navigate to the Packages/User folder by selecting the `Preferences -> Browse Packages` command from within Sublime)
  - Assuming you installed the SML Language definitions using Sublime Package Control, its location in the `Main.sublime-menu` will be incorrect. The edit the location to be `Packages/User/SML (Standard ML)/SML.tmLanguage` (or whereever your SML Language definitions package got installed). You might also have to edit the location of your SML executable as well (but I did not)

The above instructions should work, and are the simplest way I know of to add SML support, including an SML REPL to Sublime Text 2. There are some excellent posts with more details on the subject locaated at:

- How I Configured Sublime text 2
- Sublime Text 2 for SML code errors highlighting
- Sublime Text 2

2 Basic Standard ML

```
(* This is a comment (* Comments can be nested *) *)

(* Bind a value to a variable *)
(* val <var> = <expression> *)
val x = 42;  (* binds a value to a variable *)

(* Bind a function value to a variable *)
(* fun <var> (<var>: <type>, <var>: <type>, ...) = <expression> *)
fun pow(x: int, exp: int) =
   if y = 0
     then 1
     else x * pow(x, y-1)
```

3 SML Basis Library

3.1 Lists

```
open List         (* documentation - will return a list of functions on the list type *)

foo: int list     (* declares foo to be a list of type int *)
val foo = []      (* make foo an empty list*)
val foo = [42, 0, 25, ~2]

19 :: foo         (* ”cons” an element onto the head of a list *)
[1, 2] @ foo      (* returns a list with [1, 2] appended to foo *)
revAppend (l1, l2 (* returns (rev l1) @ l2 *)
null foo          (* returns true if list is null*)
hd foo            (* returns the first element of the list. It raises Empty if foo is nil*)
tl foo            (* returns the rest of the list. It raises Empty if foo is nil *)
last foo          (* returns the last element of the list. It raises Empty if foo is nil *)
length foo        (* returns the number of element the list. *)
nth (foo, i)      (* returns the i(th) element of the list foo, counting from 0.
                   It raises Subscript if i < 0 or i >= length foo. nth(foo,0) = hd foo *)
take (foo, i)     (* returns the first i elements of the list foo.
                     It raises Subscript if i < 0 or i > length foo. take(foo, length foo) = foo *)
drop (foo, i)     (* returns what is left after dropping the first i elements of the list foo.
                     It raises Subscript if i < 0 or i > length foo. drop(foo, length foo) = [] *)
rev foo           (* returns a list consisting of foo’s elements in reverse order. *)
concat l          (* takes a list of lists, and returns the list that is the concatenation of
                     all the lists in l in order. *)
app f foo         (* applies function f to the elements of foo, from left to right *)
map f foo         (* applies f to each element of l from left to right, returning the list of results
    *)
mapPartial f foo  (* applies f to each element of l from left to right, returning a list of results,
                     with SOME stripped, where f was defined.
                     f is not defined for an element of l if f applied to the element returns NONE.
                     This is equivalent to: ((map valOf) o (filter isSome) o (map f)) foo  *)
find f foo        (* applies f to each element x of the list l, from left to right, until f x
    evaluates to true.
                     It returns SOME(x) if such an x exists; otherwise it returns NONE. *)
filter f foo      (* applies f to each element x of foo, from left to right, and returns the list of
    those x
                     for which f x evaluated to true, in the same order as they occurred in the
                         argument list. *)
partition f foo   (* applies f to each element x of l, from left to right, and returns a pair (pos,
    neg)
                     where pos is the list of those x for which f x evaluated to true, and neg is the
                         list of
                    those for which f x evaluated to false. The elements of pos and neg retain the
                        same relative
                    order they possessed in foo. *)
foldl f init foo  (* ”fold left” returns f(xn,...,f(x2, f(x1, init))...) or init if the list is empty.
    *)
foldr f init foo  (* ”fold right” returns f(x1, f(x2, ..., f(xn, init)...)) or init if the list is
    empty. *)
exists f foo      (* ”There Exists” applies f to each element x of the list l, from left to right,
                     until f x evaluates to true; it returns true if such an x exists and false
                         otherwise. *)
all f foo         (* ”For All” applies f to each element x of the list l, from left to right,
                      until f x evaluates to false; it returns false if such an x exists and true
                          otherwise.
                      It is equivalent to not(exists (not o f) l)). *)
tabulate (n, foo) (* returns a list of length n equal to [f(0), f(1), ..., f(n-1)], created from left
    to right.
                     It raises Size if n < 0  *)
collate f (l1, l2)(* performs lexicographic comparison of the two lists using the given ordering f on
    the list elements. *)
```

3.2 Strings

A String in SML is a Vector of Chars

```
open Strings          (* documentation - will return a list of functions on the String type *)

val foo = ”A String”  (* declares foo to be a string *)
var emp = ””          (* declare smp to be the empty string *)
str c                 (* is the string of size one containing the character c. *)
toString s            (* returns a string corresponding to s, with non-printable characters replaced by SML escape sequences.
                         This is equivalent to translate Char.toString s *)
Int.toString n        (* Converts the integer n to a string *)

size foo              (* returns |foo|, the number of characters in string foo. *)
sub (foo, i)          (* returns the i(th) character of foo, counting from zero. This raises Subscript if i < 0 or |foo| <= i. *)

extract (foo, i, NONE)
extract (foo, i, SOME j)
substring (foo, i, j)
                      (* These return substrings of foo.
                         The first returns the substring of foo from the i(th) character to the end of the string, i.e., the string foo[i..|foo|-1].
                         This raises Subscript if i < 0 or |foo| < i.
                         The second form returns the substring of size j starting at index i, i.e., the string foo[i..i+j-1].
                         It raises Subscript if i < 0 or j < 0 or |foo| < i + j.
                         Note that, if defined, extract returns the empty string when i = |foo|.
                         The third form returns the substring foo[i..i+j-1], i.e., the substring of size j starting at index i.
                         This is equivalent to extract(foo, i, SOME j). *)

foo ^ bar             (* is the concatenation of the strings foo and bar. This raises Size if |foo| + |bar| > maxSize. *)
concat l              (* takes a list of lists, and returns the list that is the concatenation of
                         all the lists in l in order. *)
concatWith s l        (* returns the concatenation of the strings in the list l using the string s as a separator.
                         This raises Size if the size of the resulting string would be greater than maxSize. *)
implode l             (* generates the string containing the characters in the list l. This is equivalent to concat (List.map str l).
                         This raises Size if the resulting string would have size greater than maxSize. *)
explode foo           (* is the list of characters in the string foo *)
```