CS235 Languages and Automata **Handout # 4**
Prof. Lyn Turbak **September 17, 2007**
Wellesley College

Problem Set 2

Due: 11:59 pm on Monday, September 24

Required Reading:

- Handout #2: Standard ML of New Jersey (SML/NJ) and Forlan
- Handout #3: Concurrent Versions System (CVS)
- Stoughton, Chapter 2 and Section 3.1

Optional Reading: Paulson's *ML for the Working Programmer*, Chs. 1 – 3.

Note: Problems 6, 9, 10, and 11 from PS1 are due with PS2.

Submission:

You should turn in a hardcopy submission packet by slipping it under Lyn's office door by 11:59pm on the due date. This packet should include: (1) your written solutions for Problems 6, 9, 10, and 11 from PS1; (2) your written solutions for Problems 1a and 1b from this assignment; (3) your final version of `ps2.sml`; (4) your transcripts of the requested test cases.

You should also submit a softcopy (consisting of your final `ps2` directory) to the drop directory `~cs235/drop/ps2/`*username*, where *username* is your username. To do this, execute the following commands in Linux:

```
cd /students/username/cs235
cp -R ps2 ~cs235/drop/ps2/username/
```

Problem 0 [Getting Started]

a. [**Test Your Linux Account**] Begin this assignment by verifying that you can still log in to your CS Dept Linux account. Contact Lyn if you have any problems. Now is a good time to review Linux and Emacs commands: see the resources at the bottom of the CS235 home page if you need help.

b. [**Configure CVS**] In order to download code for your CS235 assignments, you will need to update your local copies of files that are in the CVS-controlled CS235 repository. Follow the directions in Handout #3 for how to install your local CVS filesystem. (This and all handouts are linked from the CS235 home page.) You only need to install your local CVS filesystem once.

Once you have installed your local CVS filesystem, you can access all CVS-controlled files by executing the following in a Linux shell:

```
cd ~/cs235
cvs update -d
```

Indeed, every time you log in to a Linux machine to work on a CS235 assignment, you should execute the above commands to ensure that you have the most up-to-date versions of the problem set materials.

On this assignment, executing the above commands will create the local directory `~/cs235/ps2` containing the file `ps2.sml`. This file contains some definitions for Problems 1 through 4, and you will add some new definitions to it.

c. [SML/NJ **and Forlan**] You will need to use SML/NJ and Forlan on this assignment. Following the instructions in Handout #2, launch an SML interpreter and evaluate a few expressions. Forlan is just a version of SML/NJ with some additional modules pre-loaded; it can be launched by executing the Linux command `forlan`.

Problem 1 [String Processing]

In this problem, you will study the four string-processing functions in Fig. 1. For your convenience, all four of these function can be found in `ps2.sml`. These function definitions use functions from the `List`, `String`, and `Char` modules in the SML/NJ library, which is described at

`http://www.standardml.org/Basis/overview.html#section:13`

Documentation pages for commonly-used modules like `List`, `String` and `Char` are also linked from the *Resources* section of the CS235 home page.

```
fun extend (pre, strings) =
  List.map (fn str => str ^ "s")
           (List.filter (String.isPrefix pre) strings)

fun caesar (n,str) =
  String.implode
    (List.map (fn c => Char.chr (((Char.ord c) + n) mod 256))
              (String.explode str))

fun twist (c,s) =
  let val (a,b) = List.partition (fn x => x <= c) (String.explode s)
  in String.implode (a @ b)
  end

fun transform s =
  let val len = String.size s
      fun process i =
            if i = len then
              ""
            else let val rest = process (i+1)
                     val c = String.sub (s,i)
                 in if Char.isAlpha c then
                       (String.str (Char.toLower c)) ^ rest
                    else
                      rest
                 end
  in process 0
  end
```

Figure 1: Four functions that manipulate strings.

a. Predict the values of each of the following expressions. You may use the SML interpreter to check your answers.

- `extend ("com", ["compare","contrast","commune","compute","think"])`
- `caesar (1, "HAL")`
- `twist (#"n", "computation")`
- `transform "Isn't your CS235 class at 9:50 in SCI 104?"`

b. Give an English specification for the behavior of each of the four functions.

c. It is possible to define a function `transform2` that behaves exactly like `transform` but does not use explicit recursion. In the file `ps2.sml`, define such a nonrecursive function `transform2` in terms of the following functions: `Char.isAlpha`, `Char.toLower`, `String.explode`, `String.implode`, `List.filter`, `List.map`. Turn in a transcript of an SML/NJ interpreter session showing that your `transform2` works as expected.

Problem 2 [Counting Characters]

This problem considers ways to count the lowercase characters `a` and `b` appearing in a string.

a. [A Recursive Counter] In `ps2.sml`, define a recursive function `countabRec` that takes a string and returns a pair of the the number of (lowercase) `a` and `b` characters in the string:

```
- countabRec "abracadabra";
val it = (5,2) : int * int
- countabRec "Abracadabra";
val it = (4,2) : int * int
- countabRec "Barbara";
val it = (3,1) : int * int
- countabRec "dog";
val it = (0,0) : int * int
```

Your definition should not use any recursive helper functions. But you may find it helpful to use the following nonrecursive helper functions, which have been defined for you in `ps2.sml`:

```
fun first s = String.str (String.sub (s,0))
fun butFirst s = String.substring (s, 1, (String.size s) - 1)
```

You should also turn in a transcript showing that your `countabRec` function works as desired.

b. [An Iterative Counter] In `ps2.sml`, define a function `countabIter` that has the same behavior as `countabRec` but is defined in terms of a tail-recursive helper function named `loop`. Your function should have the following form:

```
fun countabIter s =
  let val len = String.size s
      fun loop <tuple of variables encoding the loop state> =
        <body of the loop>
  in loop <tuple of initial values for the loop>
  end
```

You should also turn in a transcript showing that your `countabIter` function works as desired.

c. [Accepting a Language] Consider the language L_1 consisting of all strings with an even number of `a`s or an odd number of `b`s. In `ps2.sml`, define a function `isInL1 : string -> bool` that takes a string and returns `true` if the string is in L_1 and `false` otherwise. You may use `countabRec` or `countabIter` in your definition.

For example, suppose that `LTest` is the following list of strings:

```
val LTests = ["",
              "a", "b",
              "aa", "ab", "ba", "bb",
              "aaa", "aab", "aba", "abb", "baa", "bab", "bba", "bbb"]
```

Then `List.partition isInL1 LTests` denotes the following pair of string lists:

```
(["","b","aa","ab","ba","bb","aab","aba","baa","bbb"],["a","aaa","abb","bab","bba"])
```

For testing purposes, `LTests` is defined for you in `ps2.sml`.

You should also turn in a transcript showing that your `isInL1` function works as desired.

Problem 3 [String Generation]

In the previous problem, `LTest` is a list of all strings of up to size 3 that can be generated from the two characters `a` and `b`. It is often helpful to have a list of all the strings of up to a certain size that can be generated from a given set of characters. This is the purpose of the following `genStrings` function:

> `genStrings: int * char list -> string list`
> Suppose that `n` is a nonnegative integer and `cs` is a list of characters without duplicates. Then `genStrings (n,cs)` is a list of all strings with size $\leq$ `n` that can be generated using the characters in `cs`, where any character in `cs` may appear any number of times (including zero) in each of the strings.
>
> If c is the length of `cs`, then `genStrings (n,cs)` should generate a list with $\frac{c^{n+1}-1}{c-1}$ strings.
>
> Furthermore, if the characters in `cs` are ordered lexicographically (in dictionary order), then the strings in `genStrings (n,cs)` should be ordered first by size, and then lexicographically within each size.
>
> See Fig. 2 for a transcript showing examples of `genStrings`.

In this problem, your task is to define `genStrings` in the file `ps2.sml`. Turn in a transcript showing that it works as desired.

Notes/hints:

- Think recursively! For `n` > 0, assume that `genStrings(n-1, cs)` returns the correct answer. How can the result for `genStrings(n, cs)` be constructed from the result for `genStrings(n-1, cs)`?
- The functions `String.str`, `List.map`, and `List.concat` are helpful in this problem.
- The definition of `genStrings` can be remarkably short (my solution is 7 lines) and requires no helper functions.
- By default, SML/NJ will only display up to 12 elements of a list. If there are more than 12 elements, it will display the first 12 elements followed by ellipses (...). The number of list elements displayed is controlled by the variable `Control.Print.printLength`. See the middle of the transcript in Fig. 2 for how to set this variable to a larger number. Other settable control variables for printing are:
  - `printDepth`: the maximum depth of nested data structures to display.
  - `stringDepth`: the maximum number of characters in a long string to display.
  - `linewidth`: the number of characters printed on a single line.

```
- genStrings (0, [#"a"]);
val it = [""] : string list
- genStrings (1, [#"a"]);
val it = ["","a"] : string list
- genStrings (2, [#"a"]);
val it = ["","a","aa"] : string list
- genStrings (3, [#"a"]);
val it = ["","a","aa","aaa"] : string list
- genStrings (1, [#"a",#"b"]);
val it = ["","a","b"] : string list
- genStrings (2, [#"a",#"b"]);
val it = ["","a","b","aa","ab","ba","bb"] : string list
- genStrings (3, [#"a",#"b"]);
val it = ["","a","b","aa","ab","ba","bb","aaa","aab","aba","abb","baa",...]
  : string list
- Control.Print.printLength;
val it = ref 12 : int ref
  (* The printLength variable controls how many elements of a list will be printed.
     By default it is set to only 12 elements. *)
- Control.Print.printLength := 1000; (* Change printLength to 1000 *)
val it = () : unit
- genStrings (3, [#"a",#"b"]);
val it =
  ["","a","b","aa","ab","ba","bb","aaa","aab","aba","abb","baa","bab","bba",
   "bbb"] : string list
- genStrings (4, [#"a",#"b"]);
val it =
  ["","a","b","aa","ab","ba","bb","aaa","aab","aba","abb","baa","bab","bba",
   "bbb","aaaa","aaab","aaba","aabb","abaa","abab","abba","abbb","baaa",
   "baab","baba","babb","bbaa","bbab","bbba","bbbb"] : string list
- genStrings (1, [#"a",#"b",#"c"]);
val it = ["","a","b","c"] : string list
- genStrings (2, [#"a",#"b",#"c"]);
val it = ["","a","b","c","aa","ab","ac","ba","bb","bc","ca","cb","cc"]
  : string list
- genStrings (3, [#"a",#"b",#"c"]);
val it =
  ["","a","b","c","aa","ab","ac","ba","bb","bc","ca","cb","cc","aaa","aab",
   "aac","aba","abb","abc","aca","acb","acc","baa","bab","bac","bba","bbb",
   "bbc","bca","bcb","bcc","caa","cab","cac","cba","cbb","cbc","cca","ccb",
   "ccc"] : string list
- genStrings (4, [#"a",#"b",#"c"]);
val it =
  ["","a","b","c","aa","ab","ac","ba","bb","bc","ca","cb","cc","aaa","aab",
   "aac","aba","abb","abc","aca","acb","acc","baa","bab","bac","bba","bbb",
   "bbc","bca","bcb","bcc","caa","cab","cac","cba","cbb","cbc","cca","ccb",
   "ccc","aaaa","aaab","aaac","aaba","aabb","aabc","aaca","aacb","aacc",
   "abaa","abab","abac","abba","abbb","abbc","abca","abcb","abcc","acaa",
   "acab","acac","acba","acbb","acbc","acca","accb","accc","baaa","baab",
   "baac","baba","babb","babc","baca","bacb","bacc","bbaa","bbab","bbac",
   "bbba","bbbb","bbbc","bbca","bbcb","bbcc","bcaa","bcab","bcac","bcba",
   "bcbb","bcbc","bcca","bccb","bccc","caaa","caab","caac","caba","cabb",
   "cabc","caca","cacb","cacc","cbaa","cbab","cbac","cbba","cbbb","cbbc",
   "cbca","cbcb","cbcc","ccaa","ccab","ccac","ccba","ccbb","ccbc","ccca",
   "cccb","cccc"] : string list
- List.length(genStrings (4, [#"a",#"b",#"c"]));
val it = 121 : int (* (3^5 - 1)/(3 - 1) = 242/2 = 121 *)
```

Figure 2: Some sample invocations of `genStrings`.

Problem 4 [Regular Expressions]

Chapter 3.1 of Stoughton defines regular expressions and describes how they denote regular languages (sets of strings). It also shows how regular expressions can be written as SML strings in the FORLAN system. For example, consider the regular language L_2 consisting of all strings that are either (possibly empty) sequences of `1`s or (possibly empty) sequences of `10`s. (This is the same language defined in slide 1-4 from the first lecture.) L_2 is specified by the following regular expression, written as an SML string:

```
val L2RegExpString = "1* + (10)*"
```

The file `ps2.sml` provides a function

```
testRegExpString : string -> string list -> string list
```

for testing regular expressions written as SML strings.[1] Suppose that *`res`* is a regular expression string and *`strs`* is a list of strings. Then `testRegExpString` *`res strs`* returns a list of all of the strings in *strs* that are in the regular language described by *res*. For example,

```
- testRegExpString L2RegExpString (genStrings (6, [#"0", #"1"]));
val it = ["","1","10","11","111","1010","1111","11111","101010","111111"]
```

In this problem, your task is to define (in the file `ps2.sml`) a regular expression string named `L1RegExpString` that describes the language L_1 from part (c) of Problem 2. Use `testRegExpString` to test your definition, and turn in a transcript of your test(s). For example:

```
- testRegExpString L1RegExpString (genStrings (5, [#"a", #"b"]));
val it =
  ["","b","aa","ab","ba","bb","aab","aba","baa","bbb","aaaa","aaab","aaba",
   "aabb","abaa","abab","abba","abbb","baaa","baab","baba","babb","bbaa",
   "bbab","bbba","bbbb","aaaab","aaaba","aabaa","aabbb","abaaa","ababb",
   "abbab","abbba","baaaa","baabb","babab","babba","bbaab","bbaba","bbbaa",
   "bbbbb"] : string list
```

In this problem, you will need to use the FORLAN interpreter rather than the plain SML interpreter. See Sec. 3 of Handout #2 (*Standard ML of New Jersey (SML/NJ) and Forlan*) for details on how to launch the FORLAN interpreter.

[1] You do not have to understand the implementation of the `testRegExpString` function until later in the semester.