Programming Languages (COP 4020/6021)

Assignment I

Objectives

1. To become acquainted with the SML/NJ compiler.
2. To understand basic ML constructs such as lists, functions, pattern matching, anonymous variables, and let-environments.
3. To gain experience defining recursive functions in a functional programming language.

Due Date: Monday, January 21, 2019 (at 5pm)

Machine Details

Complete this assignment by yourself on the following CSEE network computers: c4lab01, c4lab02, ..., c4lab19. These machines are physically located in ENB 216. You can connect to the C4 machines from home using SSH. (Example: Host name: *c4lab01.csee.usf.edu* Login ID and Password: <your NetID username and password>) You are responsible for ensuring that your programs compile and execute properly on these machines.

Assignment Description

0) Read Sections 1-3.6.3 and 4.1 of the *Elements of ML Programming* textbook.

1) Let's represent integer-coefficient polynomials as `int list list`s. Each item in the high-level list represents a term within a polynomial, where each term is a list containing a coefficient, a nonnegative exponent for the x_1 variable, a nonnegative exponent for the x_2 variable, etc.

   For example, the following `int list list`:

   ```
   [[1],[3,2,0,0,3],[~5,1,0,7]]
   ```

   represents the polynomial:

   $$P(x_1,x_2,x_3,x_4) = 1 + 3(x_1)^2(x_2)^0(x_3)^0(x_4)^3 - 5(x_1)^1(x_2)^0(x_3)^7 = 1+3x_1^2x_4^3-5x_1x_3^7.$$

   Notice that for every polynomial `P`, there are an infinite number of equivalent representations of `P`. For example, representations of the polynomial 0 include:

   - `[[0]]`
   - `[[0,0,0]]`
   - `[[0,2]]`
   - `[[0,1,2,3]]`
   - `[[]]`
   - `[]`
   - `[[0,1,0,1],[0,0,0,0,0],[],[0,1,1]]`
   - `[[4,0,2,0,0],[~4,0,2]]`

   Let the *canonical form* of a polynomial `P`, written `c(P)`, be the shortest list-based representation of `P`, such that all terms having smaller x_1 exponents appear before all terms having larger x_1 exponents, then within sequences of terms having equal x_1 exponents, all terms having smaller x_2 exponents appear before all terms having larger x_2 exponents, etc.

Every polynomial has exactly one canonical form. For example:

- `c([]) = []`
- `c([[5,1],[~5,1],[],[0,6,8]]) = []`
- `c([[1],[3,2,0,0,3],[~5,1,0,7]]) = [[1],[~5,1,0,7],[3,2,0,0,3]]`
- `c([[4,2,0,0],[6,1,7],[~8,1,7,2],[~5,1,7,0,0],[1,1,2],[7,0],[2],[3,0,9]]) = [[9],[3,0,9],[1,1,2],[1,1,7],[~8,1,7,2],[4,2]]`

2) Create a new file called *as1.sml*. This file will contain implementations of several functions for manipulating polynomials (as described below).

For full credit, *graduate students* must implement functions that only output polynomials in canonical forms. For example, to receive full credit for a function for adding two polynomials, a graduate student's implementation must return the sum in canonical form. Similarly, to receive full credit for a function for printing polynomials, a graduate student's implementation must output the polynomials in canonical form.

In contrast, *undergraduate students* can receive full credit for this assignment without implementing anything related to canonical forms. Whereas a grad student's implementation would have to return `[]` to indicate the polynomial 0, an undergrad implementation could return any representation of 0. *Optionally, undergraduate students may complete this assignment at the graduate level, for +10% extra credit.*

Here are the functions you need to implement in *as1.sml* for full credit:

(a) **`constplus`**
This function returns the polynomial obtained by adding a constant to a polynomial. For example, `constplus(4,[[~5,0,1],[1,2],[~1]])` may (and at the grad level must) return `[[3],[~5,0,1],[1,2]]`.

(b) **`polyplus`**
This function returns the polynomial obtained by adding two polynomials. For example,
```
polyplus([[~5,0],[2,2,0],[],[~3,0,1],[6]], [[~5,0,1,0],[1,2],[~1]])
```
may (and at the grad level must) return `[[~8,0,1],[3,2]]`.

(c) **`constmult`**
This function returns the polynomial obtained by multiplying a polynomial by an integer scalar. For example, `constmult(4,[[~5,0,1],[1,2],[~1]])` may (and at the grad level must) return `[[~4],[~20,0,1],[4,2]]`.

(d) **`polymult`**
This function returns the polynomial obtained by multiplying two polynomials. For example,
```
polymult([[1,1],[~1,0,1],[~1]],[[1,0,1],[1],[1,1,0]])
```
may (and at the grad level must) return `[[~1],[~2,0,1],[~1,0,2],[1,2]]`.

(e) **`partialderivative`**
This function returns the polynomial obtained by partially differentiating another polynomial with respect to a variable (whose number is provided as an argument). For example,
```
partialderivative(2, [[1,2,3,4],[5,6,7],[8,0]])
```
may (and at the grad level must) return `[[3,2,2,4],[35,6,6]]`.

(f) `polyeval`

This function evaluates a polynomial for a given list of variable values. For example,

`polyeval([[~5,0,1],[1,2],[~1]], [4,1])`

evaluates $-5x_2+x_1^2-1$ with $x_1=4$ and $x_2=1$, returning the value 10. The polyeval function should assume that any variable whose value is not defined in the second argument has a value of 1, so e.g., `polyeval([[~5,0,1],[1,2],[~1]],[4])` also returns 10.

(g) `polyprint`

[This is a grad-level function, part of the +10% extra credit for undergraduates.]

This function prints a polynomial in non-list notation. For example,

`polyprint([[1],[3,2,0,0,3],[~5,1,0,7]])`

prints `1 - 5(x1)^1(x2)^0(x3)^7 + 3(x1)^2(x2)^0(x3)^0(x4)^3` and returns `()`. As another example, `polyprint([[~2],[~1,2]])` prints `-2 - 1(x1)^2` and returns `()`.

Notes:

- `polyprint` must print polynomials in the format shown above. For example, variable names are always printed within parentheses, exponents are preceded by the "caret" symbol `^`, leading plus signs (at the very beginning of the printed polynomial) are not printed, whitespace exists before and after all plus signs and non-leading minus signs, the minus symbol is `-` rather than `~`, etc.
- `polyprint` always prints at least one integer, even if it is just a `0`. That is, polynomials equivalent to zero cannot be printed as nothing.
- `polyprint` always prints a newline (`\n`) after a polynomial.

Sample Execution

```
> sml
Standard ML of New Jersey v110.74 [built: Thu Aug 16 11:25:45 2012]
- (* This is a sample grad-level execution.  An undergrad-level execution    *)
- (* would be the same, except (1) the c and polyprint functions would not   *)
- (* be declared or used, and (2) all polynomials output would only have to  *)
- (* be equivalent to the outputs shown below.                               *)
- use "as1.sml";
[opening as1.sml]
[autoloading]
[library $SMLNJ-BASIS/basis.cm is stable]
[autoloading done]
val c = fn : int list list -> int list list
val constplus = fn : int * int list list -> int list list
val polyplus = fn : int list list * int list list -> int list list
val constmult = fn : int * int list list -> int list list
val polymult = fn : int list list * int list list -> int list list
val partialderivative = fn : int * int list list -> int list list
val polyeval = fn : int list list * int list -> int
val polyprint = fn : int list list -> unit
val it = () : unit
- constplus(4,[[~5,0,1],[1,2],[~1]]);
val it = [[3],[~5,0,1],[1,2]] : int list list
- polyplus([[~5,0],[2,2,0],[],[~3,0,1],[6]], [[~5,0,1,0],[1,2],[~1]]);
val it = [[~8,0,1],[3,2]] : int list list
- constmult(4,[[~5,0,1],[1,2],[~1]]);
val it = [[~4],[~20,0,1],[4,2]] : int list list
- polymult([[1,1],[~1,0,1],[~1]],[[1,0,1],[1],[1,1,0]]);
val it = [[~1],[~2,0,1],[~1,0,2],[1,2]] : int list list
- partialderivative(2, [[1,2,3,4],[5,6,7],[8,0]]);
val it = [[3,2,2,4],[35,6,6]] : int list list
- partialderivative(5, [[1,2,3,4],[5,6,7],[8,0]]);
val it = [] : int list list
```

```
- partialderivative(1,[[1],[~2,1,1],[3,4,1],[~5,0,6],[7,2,6]]);
val it = [[~2,0,1],[14,1,6],[12,3,1]] : int list list
- polyeval([[~5,0,1],[0,1],[1,2],[~1]], [4,1]);
val it = 10 : int
- polyeval([[~5,0,1],[1,2],[~1]],[4]);
val it = 10 : int
- polyprint([[1],[3,2,0,0,3],[~5,1,0,7]]);
1 - 5(x1)^1(x2)^0(x3)^7 + 3(x1)^2(x2)^0(x3)^0(x4)^3
val it = () : unit
- polyprint([[~2],[~1,2]]);
-2 - 1(x1)^2
val it = () : unit
```

Grading notes
For full credit, your implementation must:

- be commented and formatted appropriately. During grading, the TA will read and check your source code; full-credit assignments must be reasonably easy to understand.
- use spaces instead of tabs for indentation. *Reason why*: The TA has to read many submissions, written with many editors, each of which may have its own conventions for displaying tabs. By using spaces instead of tabs, the TA can view your code as you intend it to be viewed, without having to tailor his editor's tab settings to your individual submission.
- use ML features, such as anonymous variables, pattern matching, and let-environments, when appropriate.
- define no global (top-level) values, except for the functions specified in this document.
- Compile and run on the C4 machines with no errors or warnings.
- not use any advanced ML features that cause *side effects* to occur (e.g., I/O or pointer use). The one exception is that `polyprint` may invoke the built-in `print` function, which has the I/O side effect of printing a string.
- not use any built-in (i.e., predefined/library) functions besides `print` and `Int.toString`.
- not be significantly more complicated than is necessary.
- be reasonably efficient (e.g., not have exponential running time)

The TA will test submissions on inputs not shown in the examples above. However, you can assume that (1) all polynomials passed as arguments to *as1.sml* functions lack negative exponents, and (2) all variable numbers passed as arguments to `partialderivative` are positive.

Hints
Just to give an idea of the effort involved: It took me about 6 hours to implement and test my grad-level *as1.sml*, which is 97 lines of code, (not counting whitespace/comments). My undergrad-level implementation is 40 lines of code. If, after completely reading Sections 1-3.6.3 and 4.1 of the textbook, you find yourself spending a significant amount of time (e.g., more than a couple hours) stuck on any one of this assignment's functions, please visit or email the teaching assistant to ask for help.

Submission Notes

- Type the following pledge as an initial comment in your *as1.sml* file: "I pledge my Honor that I have not cheated, and will not cheat, on this assignment." Type your name after the pledge. Not including this pledge will lower your grade 50%.
- Upload and submit your *as1.sml* file in Canvas.
- You may submit *as1.sml* files as many times as you like; we'll grade your latest submission.
- You may submit late (between 5pm on 1/21 and 5pm on 1/23) with a 15% penalty.