Programming Languages (COP 4020/6021) [Spring 2016]

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: Sunday, January 24, 2016 (at 11:59pm).

Machine Details

Complete this assignment by yourself on the following CSEE network computers: c4lab01, c4lab02, ..., c4lab20. These machines are physically located in ENB 220. Do not use any server machines like grad, babbage, sunblast, etc. 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 4.1-4.2 of the *Elements of ML Programming* textbook.

1) Let's represent (finite) sets of integers as pairs in ML. The first element of the pair is an `int list` indicating the universe the set is drawn from; the second element of the pair is a `bool list` indicating which elements of the universe are in the set. Hence, our sets have type `int list * bool list`. For example, the value `([1,3,5,7,9],[true,false,true,false,true])` represents the set {1, 5, 9} drawn from the universe of {1, 3, 5, 7, 9}. The order of integers in the `int list` is irrelevant, so `([3,7,1,9,5],[false,false,true,true,true])` also represents the set {1, 5, 9}. The empty set can be written as `([],[])`.

   Two invariants must be maintained on all sets represented in this way, as a pair of lists (*L*, *B*):
   - No element may appear more than once in *L*.
   - The two lists *L* and *B* must have the same length.

   For example, `([1,1],[true,true])` and `([1,2],[true])` are invalid sets. During grading, we will only pass valid sets to your code. Conversely, you must ensure that all sets returned by your code obey these invariants.

   Using this representation of sets, implement the following functions in a file named *sets.sml*:

   (a) `cardinality`
   This function returns the size of a set of integers. For example, `cardinality(([2,7,4],[false,true,false]))` returns `1`.

   (b) `elementOf`
   This function tests whether an integer is an element of a given set of integers. For example, `elementOf(7, ([2,7,4],[false,true,false]))` returns `true`, while `elementOf(2, ([2,7,4],[false,true,false]))` returns `false`.

(c) `subset`
This function tests whether one set is a subset of another. For example, `subset(([2,7,5],[false,true,false]),([7,4],[true,true]))` returns `true` because {7} is a subset of {7, 4}.

(d) `equals`
This function tests whether two sets are equal. For example, `equals(([9,7,5],[false,true,true]), ([5,7,2],[true,true,false]))` returns `true` because {7,5} equals {5,7}.

(e) `union`
This function returns the union of two sets. For example, `union(([2,3],[true,false]),([3,4],[false,true]))` may return `([2,3,4],[true,false,true])` because {2} ∪ {4} = {2, 4}. Other correct return values include `([2,4],[true,true])` and `([4,3,2],[true,false,true])`.

(f) `funion`
This function returns the union of every set in a list (family) of sets. When passed an empty list, `funion` returns an empty set. For example, `funion([([2,3],[true,false]), ([3,4],[false,true]), ([],[])])` may return `([2,3,4],[true,false,true])` because {2} ∪ {4} ∪ ∅ = {2, 4}.

(g) `intersection`
This function returns the intersection of two sets. For example, `intersection((([2,3],[true,true]),([3,4],[true,true])))` may return `([3],[true])` because {2,3} ∩ { 3,4} = {3}.

(h) `fintersection`
This function returns the intersection of every set in a list (family) of sets. When passed an empty list, `fintersection` returns the empty set. For example, `fintersection([([2,3],[true,true]),([3,4],[true,true]),([],[])])` may return `([],[])` because {2, 3} ∩ {3, 4} ∩ ∅ = ∅.

(i) `complement`
This function returns the complement of a set with respect to its universe. For example, `complement(([1,2,3],[false,true,false]))` may return `([1,2,3],[true,false,true])` because {1,3} is the complement of {2} in the universe containing 1, 2, and 3.

(j) `cartesian`
This function takes a pair of sets *S*, *T* and returns the set of all the pairs containing an element of *S* in their first position and an element of *T* in their second position. If either *S* or *T* is the empty set, then `cartesian` simply returns the empty set. For example, `cartesian(([2,7,4],[true,false,true]), ([7,3],[true,true]))` may return `([(2,7),(2,3),(7,7),(7,3),(4,7),(4,3)], [true,true,false,false,true,true])` because {2, 4} x {7, 3} = {(2,7), (2,3), (4,7), (4,3)}. The order of pairs returned by `cartesian` is irrelevant.

(k) `powerset` [Note: For undergraduates this function is optional, extra credit (+5%).] This function takes a set of integers *S* and returns a set of the subsets of *S*. For example, `powerset(([2,7,4],[false,true,true]))` may return `([[],[2],[7],[4],[2,7],[7,4],[2,4],[2,7,4]], [true,false,true,true,false,true,false,false])` because $2^{\{7,4\}} = \{\varnothing, \{7\}, \{4\}, \{7,4\}\}$. The order of elements in lists returned by `powerset` is irrelevant.

(l) `invpowerset` [Note: For undergraduates this function is optional, extra credit (+5%).] This function takes a set of sets *T* and tests whether there exists a set *S* such that `powerset`(*S*)=*T*. If such an *S* exists then `invpowerset` returns `SOME` *S*; otherwise `invpowerset` returns `NONE`. For example, `invpowerset(([[],[2],[7],[4],[2,7],[7,4],[2,4],[2,7,4]], [true,false,true,true,false,true,false,false]))` may return `SOME ([7,4],[true,true])` because $2^{\{7,4\}} = \{\{4\}, \{7\}, \varnothing, \{7,4\}\}$, while `invpowerset(([[],[2],[7],[4],[2,7],[7,4],[2,4],[2,7,4]], [true,false,true,true,false,false,false,false]))` returns `NONE`.

To summarize, your functions should have the types shown below (except your complement function may alternatively have type `int list * bool list -> int list * bool list`).

```
- use "sets.sml";
[opening sets.sml]
val cardinality = fn : int list * bool list -> int
val elementOf = fn : int * (int list * bool list) -> bool
val subset = fn : (int list * bool list) * (int list * bool list) -> bool
val equals = fn : (int list * bool list) * (int list * bool list) -> bool
val union = fn
  : (int list * bool list) * (int list * bool list) -> int list * bool list
val funion = fn : (int list * bool list) list -> int list * bool list
val intersection = fn
  : (int list * bool list) * (int list * bool list) -> int list * bool list
val fintersection = fn : (int list * bool list) list -> int list * bool list
val complement = fn : 'a list * bool list -> 'a list * bool list
val cartesian = fn
  : (int list * bool list) * (int list * bool list)
    -> (int * int) list * bool list
val powerset = fn : int list * bool list -> int list list * bool list
val invpowerset = fn
  : int list list * bool list -> (int list * bool list) option
val it = () : unit
```

Grading

For full credit, your implementation must:

- be commented and formatted appropriately (please use spaces instead of tabs for indentation).
- use anonymous variables, pattern matching, and let-environments when appropriate (e.g., define all helper functions in let-environments).
- compile 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).
- not use any built-in (i.e., predefined or library) functions.
- not be significantly more complicated than is necessary.
- assume that incoming set arguments satisfy the data-structure invariants described on Page 1 (i.e., your functions should not try to enforce the validity of their set arguments).
- never return a set that violates the data-structure invariants described on Page 1.

Please note that we will test submissions on inputs not shown in the explanations above.

Hints

It took me 2 hours to implement and test my *sets.sml*, which is 80 lines of code (not counting whitespace/comments). If, after completely reading Sections 1-3.6.3 and 4.1-4.2 of the textbook, you find yourself spending a significant amount of time (e.g., more than 12 hours) on this assignment, please visit or email the teaching assistant to ask for help with whatever problems you are having.

Submission Notes

- Type the following pledge as an initial comment in your *sets.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 *sets.sml* file in the Canvas folder for this assignment.
- You may submit your assignment in Canvas as many times as you like; we will grade your latest submission.
- For every day that your assignment is late (up to 3 days), your grade reduces 10%.