Homework 4: Standard ML

Programming Languages

Due Friday, March 14th
(I will accept late submissions until Monday, March 16th)

Instructions:

1. Submit three files on E-Course: `coding.sml` with your answers to problems 1 and 2, `mlscheme.sml` with the modified interpreter, and `questions.txt` with your answers to problem 3, tests you wrote for the interpreter in problem 4, and (optionally) solutions to problem 5.

2. All code that you submit must compile with `MLton`. Solutions that do not compile will recieve a grade of 0.

3. I encourage you to try all problems, especially 3 and 4. There are a lot of instructions but you only need to write a few lines of code / text and you will get experience looking at an interpreter in ML!

4. This homework is due on Friday but you can submit until Monday without penalty if you need more time – I will be grading these starting on Tuesday, March 17th.

Resources: Several of you have asked for additional resources / materials on ML. Here are some often recommended resources:

1. Bob Harper's Programming in Standard ML

2. A guide to Learning Standard ML by Norman Ramsey

3. Mads Tofte's Tips for Computer Scientists on Standard ML

1 Coding in ML: You First ML Function [5 pts]

Define a function `mynull` (of type `'a list -> bool`), which when applied to a list tells whether the list is empty. Your function must not call any other functions; use pattern matching instead.

2 Using ML's higher-order functions [10 pts]

2.1 Reverse

Define a function `reverse` (of type `'a list -> 'a list`). Use ML's predefined functions `foldl` or `foldr`, which are like the μScheme versions except the ML versions are curried.

2.2 Minlist

Define a function `minlist` (of type `int list -> int`), which returns the smallest element of a nonempty list of integers. If given an empty list of integers, your solution must fail with the built-in `Match` exception (i.e., by `raise Match`). Use `foldl` or `foldr` with `Int.min`, which is part of ML's initial basis; do not use recursion.

3 Understanding μScheme interpreter [15 pts]

In this exercise, you will look at the μScheme interpreter, which is written in Standard ML. You can download the interpreter form this homework's page on E-Course (`mlscheme.sml`). Answer the following questions about this code:

1. μScheme's initial basis has functions `negated` and `-`. One of them is primitive and one is predefined. Which one is which? How do you know that from the code (reference specific lines in the interpreter)? Why do you think the author of the interpreter made the choice that they did when deciding which of these to make a primitive function?

2. What does the `expString` function do? Why does the interpreter need this functionality (reference specific lines in the interpreter in your response)?

3. In μScheme, calling a user-defined function with a wrong number of arguments causes a runtime error. Where is the error raised (give the line number)? Where is the error originally detected (give the line number)? You may want to look at the `eval` function, which is the interpreter proper, and at the helper functions that it calls.

4 Extending μScheme interpreter [20 pts]

In this exercise, you will extend the μScheme interpreter to support short-circuiting `||`, and `&&`. Follow the steps below and upload your modified version of `mlscheme.sml` on E-Course. You will only need to write about 10 lines of code.

1. First you need to add `||` and `&&` to the abstract syntax. To do that, search for the definition of `datatype exp` and add two new constructors. You can call them `OR` and `AND`.

2. Run `MLton` on the file you just modified. It should produce two warnings: `Case is not exhaustive` and `Function is not exhaustive`. This is because you extended the datatype `exp` but did not modify the functions that use it. The first of these functions is `expString`. Extend the corrsponding case expression to support the two new forms of abstract syntax. Your code will look very similar to the `IFX (e1, e2, e3)` case.

3. The second function that branches on the values of datatype `exp` is `eval`, which is the interpreter proper. Add two new cases to it (you will actually be adding the cases to the helper function `ev`). As before, look for the `(IFX (e1, e2, e3))` case for inspiration. Compile the code again and make sure that it produces no warnings.

4. You have now added short-circuiting `OR` and `AND` to the abstract syntax and the interpreter but the parser still does not know about this. Search for `(if e1 e2 e3)` to find where the parser behavior is defined. The weird `<$>` and `<*>` operators are parser combinators. You don't really need to know how they work (but if you want to learn, read this) – you should be able to add the two cases for `||` and `&&` by replicating the `if` case. The only difference is that there are only two subexpressions, not three.

5. The last step is two mark `||` and `&&` as reserved words to prevent the user from defining functions that are called this way. Search for the `reserved` list and modify it accordingly.

6. You should now be done. Compile `mlscheme.sml`, launch the resulting program (μScheme interpreter) and run a few tests to make sure that your code works as expected. **Put 5 such tests in `questions.txt` file that you will be submitting as part of this homework**.

5 Bonus Question [10 pts]

The `||` and `&&` operators really are just syntactic sugar for two different kinds of the `if` expression. How would you modify the `ev` function to reuse the `IFX` case when evaluating `OR` and `AND`?