Formal Semantics

This assignment focuses on formal semantics, with a tiny application to the construction of interpreters.

Deadline Extended!
New deadline: Sunday April 17 at 11:59PM.

Denotational Semantics (to do with a partner)

Problems 1-3 involve modifying the denotational semantics given in the lecture notes. For each problem, give only those parts of the semantics that have changed.

1. Modify the "Locations and Environments" semantics in the lecture notes so that variables are automatically initialized to zero at the time they are declared.
   Difficulty: *
2. Modify the "Gotos and Labels" semantics in the lecture notes so that *A* (answers) are the same as *V* (integers), and the answer given by a program is the value of the variable `Answer` at the end of termination. *Hint: it is sufficient to give a definition of the* `done` *function.*
   Difficulty: *
3. Again, start with the "Gotos and Labels" semantics, but modify the language to add a print statement. Add a grammar rule

   ```
   stm ::= PRINT exp
   ```

   Let *A*, the type of answers, be the same as *V list* (list of integer), and assume you are given the constructors `nil` and `cons` on lists. (Simply add `nil : V list` and `cons : V * V list -> V list` to the list of predefined functions.) Finally change the semantics so that a program like this one:

   ```
   A := -5;
   LOOP: ;
   PRINT A+5;
   A := A+1 ;
   IF A GOTO LOOP
   ```

   yields the answer *cons(0, cons(1, cons(2, cons(3, cons(4, nil)))))*. That is, the answer yielded by a program should be a list, in order, of everything printed.
   Hint: just fiddling with the "done" function is not enough. Think about the semantics of the `PRINT` *statement.*
   Warning: There is no variable called `Answer`*, so don't try to use one.*
   Difficulty: *

Each of these problems can be solved with a one- or two-line change to the denotational definitions given in class. I have listed the difficulty of each problem as low, but you will need to spend significant effort on understanding the original semantics. I would give that part of the task three stars for difficulty.

For those of you who may wish to use TeX for your answers, the source for the lecture notes is available online.

Fixed points (to do on your own)

4. Define, in Standard ML or uML, a fixed-point operator `fix` that works **on functions only**. It will have type

   ```
   forall 'a, 'b . (('a -> 'b) -> ('a -> 'b)) -> ('a -> 'b)
   ```

   Difficulty: **

   Hints:

- ♦ You will need to use `val-rec` or `define` or `fun`.
- ♦ `fix` satisfies the same algebraic law as the Y combinator in the lecture notes.
- ♦ A simple definition based on that algebraic law will loop forever.
- ♦ You can prevent the looping by eta-expanding the right-hand side.

Note: You can easily look this one up, but try it on your own first.

5. Use your `fix` function to define new versions of `map` and `length`. You may use `val` or `lambda` or `fn` but not `val-rec` or `define` or `fun`.
   Hint: If you use Standard ML instead of uML, you will need to avoid the value restriction on polymorphic functions. You can do this by eta-expanding any application of `fix`, as follows:

```
val reverse = fix (fn r => ...)              (* breaks value restriction *)
val reverse = fn xs => fix (fn r => ...) xs   (* obeys value restriction *)
```

   uML has no value restriction.
   Difficulty: *
6. Using the method of successive approximations, solve the following recursion equation:

```
  xs = (cons 0 (map ((curry +) 1) xs)))
```

   Carry out these steps:

   a. In the notation of your choice, write the function whose fixed point is a solution to the equation.
   b. Approximation zero is the bottom list _|_.
   c. Compute approximation 1 by hand by applying the function from step a to approximation 0. Continue using the method of successive approximations and write approximations 2 and 3.
      Keep in mind that `map` is a *strict* function: when applied to bottom, it returns bottom.
   d. In informal English, say what you think the denotation of `xs` is.
      If you wish, for **extra credit**, prove your claim.

   Put your answers in file `approx.txt`.
   Difficulty: *

My solution problems 4 and 5 totals 4 lines of Standard ML.

A definitional interpreter (to do on your own)

In the lecture notes and on the web you will find a small definitional interpreter for a semantics with expression continuations. The interpreter is stored in file `expk.sml`.

7. Your job is to **extend that interpreter with Boolean expressions**.

   You will add the following syntax:

```
datatype bexp = NOT of bexp
              | AND of bexp * bexp
              | OR  of bexp * bexp
              | EQ  of exp * exp
```

   In addition, you will add a *value constructor for expressions* and a *value contructor for statements*

```
   CAST : bexp -> exp
   IF   : bexp * stm * stm -> stm
```

   The semantics of `CAST` is that a true expression is cast to the value 1 and a false expresion is cast to the value 0.

   You will then add semantic equations for the new syntax. The semantics of a Boolean expression is that **a Boolean expression has no value; it is executed purely to determine control flow**. A Boolean expression takes two continuations: a success continuation and a failure continuation. This is a exactly how Booleans are handled in real compilers. (The `CAST` is there in case a Boolean appears in a value context.)

The parts of this problem are

a. Add the new `bexp` syntax to the interpreter. It is mutually recursive with `exp`; you will need to use `and` *instead* of `datatype`.
b. Write the semantic function `B` for Booleans, with type

```
B : bexp -> C -> C -> C
```

   The first argument to `B` is a Boolean expression. The second argument says how the program should continue if the expression is true. The third argument says how the program should continue if the expression is false. The result what happens if the program is started right before the evaluation of the Boolean expression.

   The semantic function `B` should implement **short-circuit semantics** for `AND` and `OR`.
c. Add the `IF` statement to `stm` and a corresponding clause to the `S` function.
d. Add the `CAST` expression to `exp` and a corresponding clause to the `E` function.

Difficulty: **

To solve this problem, I had to add 13 nonblank lines to the existing interpreter: 6 lines of datatype definitions and 7 lines of function definitions.

What to submit

For the denotational semantics part, submit your joint solutions (with your partner) in the PDF file `sem.pdf`, using `submit105-sem-pair`.

For the remaining parts, please submit the work that **should be entirely your own**

- A file `fix.sml` or a file `fix.uml` that solves the two fixed-point problems.
- A file `expk.sml` that contains your modified definitional interpreter.
- A README file that tells us, among other things,
  e. With whom you collaborated
  f. Whether you developed the `fix` function entirely on your own, and if not, where you got help and of what kind
  g. How much time you spent

The submit script for the solo parts is `submit105-sem-solo`.