SC-T-501-FMAL Programming languages, Assignment 4
Spring 2020
Due 5 April 2020 at 23:59

1. For each of the lambda-calculus terms

   (i) $\lambda x.\, x$

   (ii) $\lambda x.\, x\, x$

   (iii) $\lambda f.\, \lambda x.\, f\, x$

   say whether it can be given the following types in the polymorphic lambda calculus:

   (a) $\forall X.\, X \to X$

   (b) $(\forall X.\, X) \to (\forall X.\, X)$

   (c) $(\forall X.\, X \to X) \to (\forall X.\, X \to X)$

   (Give an answer for each of the nine (term, type) pairs.)

2. Write a statement in the abstract syntax of the naive imperative language Imp (which is implemented in Assignment4.fs) that computes the sum of the factors of a positive integer in the variable `n`, placing the result in the variable `r`. For example, if `n` contains 12 initially, after executing the statement, the variable `r` should contain $28 = 1 + 2 + 3 + 4 + 6 + 12$. Do this by defining an F# value `sumFactors` of type `stmt`.

3. The provided file extends the syntax of Imp with a conditional assignment operator `CAssign (x, e)` that evaluates `e` and assigns the result to `x` if `x` is zero, and does nothing otherwise. Implement the `CAssign` case of the `exec` function.

4. `CAssign` statements can be implemented using ordinary assignments and `if` statements. Complete the implementation of the `removeCAssign` function, so that it replaces all of the `CAssign` statements in this way.

5. Render the following MicroC statement, presented in here abstract syntax, in concrete syntax.

```
Expr (Assign (AccVar "x",
    Prim2 ("*", CstI 7,
        Assign (AccIndex (AccVar "a", Prim2 ("+", Access (AccVar "i"), CstI 5)),
           Access (AccDeref (Prim2 ("+", Addr (AccVar "y"), CstI 3)))))))
```

   (See the file MicroC.fs for the definition of the abstract syntax.)

6. Consider the following MicroC code:

```
void f(int x, int y) {
  int *p;
  int *q;
  int **r;
  p = &x;
  q = &y;
  r = &q;
  if (*p == 0) {
      q = p;
      x = 2;
  }
  print *q;
  *r = p;
  print *q;
}

void g(int x, int y)
```

```
    {
        int a[100];
        int *p;
        p = a;
        a[x] = 10;
        *(p + 1) = y;
        *p = *(p + x);
        print a[0];
        print a[1];
    }
```

What will be printed by the following function calls?

(i) `f(0, 10)`

(ii) `f(1, 10)`

(iii) `g(5, 6)`

(iv) `g(1, 2)`