Programming Languages (COP 4020/6021) [Spring 2018]

Assignment III

Objectives

1. To become familiar with recursive data types in ML.
2. To understand basic definitions related to variables in programming languages: free variables, capture-avoiding substitution, and alpha equivalence.
3. To gain experience with deductive systems and inference rules.

Due Date: Monday, February 26, 2018 (at 5pm).
Late submission: You may submit any part (or both parts) of this assignment late (i.e., between 5pm on 2/26 and 5pm on 2/28) with a 15% penalty on the whole assignment.

Machine Details: Complete this assignment by yourself, the programming portion 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

(1) **Programming portion:** Let's consider a language called STERLING (Simply Typed, Equality Recognizing, Likeable, Integer-Numeric Goodness). STERLING has only two kinds of types, integers and functions. The expressions in STERLING are integer values, successor and predecessor operations, function values (in which the function and/or parameter name may be anonymous), function applications, variables, and branch-if-equal expressions of the form beq(e1,e2) to e3 else e4 (meaning evaluate integer-type expressions e1 and e2; if they evaluate to the same integer then evaluate e3, and if they evaluate to unequal integers then evaluate e4).

The datatypes defining STERLING are as follows.

```
(* STERLING types *)
datatype typ = Int | Arrow of typ * typ; (* i.e., Arrow(argType, returnType) *)

(* STERLING expressions *)
datatype expr =
  IntExpr of int
| SuccExpr of expr (* computes the successor of the int-type subexpression *)
| PredExpr of expr (* computes the predecessor of the int-type subexpression *)
| FunExpr of string option * string option * typ * typ * expr
  (* i.e.: FunExpr(funName, paramName, paramType, returnType, bodyExpr) *)
| ApplyExpr of expr * expr
  (* i.e.: ApplyExpr(funExpr, argumentExpr) *)
| VarExpr of string
| BeqExpr of expr * expr * expr * expr;
  (* i.e.: BeqExpr(testExpr1, testExpr2, toBranchExpr, elseBranchExpr) *)
```

Create a new file called *as3.sml*, and begin that file with the datatypes given above. Then implement the following values in *as3.sml*.

(a) `fv : expr -> string list`
This function returns a list of all the free variables in the given STERLING expression. The returned list should not contain duplicates.

(b) `sub : expr -> string -> expr -> expr`
This function takes an expression *e*, a string *x*, and another expression *e'*, and returns *[e/x]e'*, that is, the expression resulting from substituting *e* for free *x* in *e'*. Your implementation may assume that if *e'* is a function named *f* with a parameter named *y*, then neither *f* nor *y* are free in *e*.

(c) `alphaEquiv : expr -> expr -> bool` [This function is 5% extra credit for undergrads.]
This function takes two expressions *e* and *e'* and returns true iff *e* is alpha-equivalent to *e'*. Note that *e* and *e'* may contain free variables. You may assume that no function *f* in the inputs to `alphaEquiv` has a parameter also named *f*.

(d) `plus : expr` [This value is 5% extra credit for all students.]
This part requires programming in STERLING. `plus` is a STERLING function that takes two Curried integer parameters *i* and *j* and returns *i*+*j*.

The file at http://www.cse.usf.edu/~ligatti/pl-18/as3/exprs.sml defines a few STERLING expressions, which may help with testing.

Sample Executions

```
- use "as3.sml";
...
- use "exprs.sml";
...
- fv e;
val it = [] : string list
- fv eBad;
val it = ["x","y"] : string list
- val e3 = sub (IntExpr(5)) "x" (sub (IntExpr(7)) "y" eBad)  (* i.e., [5/x,7/y]eBad *);
val e3 =
  ApplyExpr
    (FunExpr (SOME "f",NONE,Int,Int,ApplyExpr (VarExpr "f",IntExpr 5)),
     ApplyExpr
       (FunExpr
          (NONE,SOME "y",Arrow (Int,Int),Int,
           BeqExpr
             (ApplyExpr (IntExpr 5,IntExpr 0),IntExpr 0,IntExpr 1,IntExpr 0)),
        FunExpr (NONE,SOME "x",Int,Int,SuccExpr (IntExpr 7)))) : expr
- fv e3;
val it = [] : string list
- e = (sub (IntExpr(987)) "x" e);
val it = true : bool
- alphaEquiv e e';
val it = true : bool
- alphaEquiv e eBad;
val it = false : bool
- alphaEquiv e2 e2';
val it = false : bool
```

Grading and Submission Notes
For full credit, your implementation must (i) compile and execute on the C4 machines with no errors/warnings, (ii) contain no side effects, (iii) not define any extra top-level values, (iv) not use any library functions other than map, foldl, foldr, and Int.toString, (v) be commented and formatted properly, including using spaces instead of tabs and limiting line widths to 100 characters, (vi) use ML (and STERLING) constructs like pattern matching and anonymous variables when appropriate, (vii) not be significantly more complicated than necessary, and (viii) be reasonably efficient. As always, we will test submissions on inputs not shown above.

The submission process is the same as for Assignment II, except here you'll submit *as3.sml* in Canvas. Please remember to include the pledge as an initial comment; not doing so will lower your grade 50%.

(2) **Theory portion:**
Consider the following language L:

types $\tau ::= \text{int} \mid \tau_1 \times \tau_2$
expressions e ::= i | e_1+e_2 | x | (e_1,e_2) | case e_1 of p=> e_2 else _=> e_3
patterns p ::= x | _ | i | (p_1,p_2)

This language contains integers (i), addition expressions, variables (x), pairs (i.e., tuples of two elements), and simple case-expressions. Case-expressions in L have the same meaning as case-expressions in ML; we just force them to always have 2 cases separated by the keyword "else", with the second case's pattern always being a wildcard (_).

Provide definitions for: (a) free variables in expressions in L, (b) capture-avoiding substitution in L, and (c) alpha-equivalence of expressions in L.

Theory-portion Submission Reminders

- Write the following pledge at the end of your submission: "I pledge my Honor that I have not cheated, and will not cheat, on this assignment." Sign your name after the pledge. Not including this pledge will lower your grade 50%.
- For full credit, turn in a hardcopy (handwritten or printed) version of your solutions.
- Late submissions may be emailed or submitted in hardcopy.
- All emailed submissions, even if sent before the deadline, will be graded as if they were submitted late, i.e., with a 15% penalty.
- If you think there's a chance you'll be absent or late for class on the date this assignment is due, you're welcome to submit solutions early by giving them to me or the TA before or after class, or during any of our office hours.