Programming Languages (COP 4020/6021) [Spring 2019]

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 25, 2019 (at 5pm).
Late submission: You may submit any part (or both parts) of this assignment late (i.e., between 5pm on 2/25 and 5pm on 2/27) with a 15% penalty on the whole assignment.

Machine Details: Complete this assignment by yourself, the programming portion on the following CSEE computers: c4lab01, c4lab02, ..., c4lab19. These machines are located in ENB 216. 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 new language called diML+P, which is diML with pattern matching. The datatypes defining diML+P are:

```
(* diML+P types *)
datatype typ = Bool | Int | Arrow of typ*typ; (* i.e., Arrow(argType, returnType) *)
(* diML+P patterns *)
datatype pattern = IntPattern of int | TruePattern | FalsePattern
                 | WildcardPattern | VarPattern of string;
(* diML+P expressions *)
datatype expr = VarExpr of string | TrueExpr | FalseExpr | IntExpr of int
  | PlusExpr of expr*expr | LessExpr of expr*expr | ApplyExpr of expr*expr
  | IfExpr of expr*expr*expr (* i.e.: IfExpr(condition, thenBranch, elseBranch) *)
  | FunExpr of string*typ*typ*((pattern*expr) list);
  (* i.e. FunExpr(functionName, parameterType, returnType, body) *)
  (* A function body is a list of (pattern,expr) pairs. *)
  (* Each such pair encodes one case of the function. *)
```

Create a new file called *as3.sml*, and begin that file with the diML+P 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 diML+P expression. The returned list should not contain any duplicates.

(b) `sub : (expr * string) list -> expr -> expr`
This function takes a list of expression-string pairs (e_1, x_1), (e_2, x_2), …, (e_n, x_n) and then an expression e, and returns $[e_1/x_1][e_2/x_2]\ldots[e_n/x_n]e$, that is, $[e_1/x_1]$ ($[e_2/x_2]$ (… ($[e_n/x_n]e$)...)), where $[e/x]e'$ refers to the capture-avoiding substitution of e for free x in e'. Your implementation may assume that expressions passed to this *sub* function have already been alpha-converted to ensure that no variables will be captured.

(c) `uniquifyVars : expr -> expr` [Note: This function is +10% extra credit for undergrads.]
This function takes an expression e and returns an alpha-equivalent expression e' that never reuses a variable name (i.e., all variables in e' must be uniquely named). Your implementation may assume that no function named *f* in *e* has a parameter named *f*.

Hints: My *fv* is 23 lines, *sub* is 20 lines, and *uniquifyVars* is 35 lines, written in about 3 hrs total.

Sample Executions:

```
- use "as3.sml";
...
- use "exprs.sml"; (* using http://www.cse.usf.edu/~ligatti/pl-19/as3/exprs.sml *)
...
- (fv e1, fv e2, fv e2bad, fv mult, fv e3);
val it = ([],[],["z"],[],[])
  : string list * string list * string list * string list * string list
- sub [(e3,"z")] e2bad;
val it =
  FunExpr
    ("f",Int,Arrow (Int,Arrow (Int,Int)),
     [(VarPattern "x",
       FunExpr
         ("f",Int,Arrow (Int,Int),
          [(VarPattern "y",
            PlusExpr
              (PlusExpr (VarExpr "x",VarExpr "y"),
               ApplyExpr
                 (FunExpr
                    ("factorial",Int,Int,
                     [(IntPattern 0,IntExpr 1),
                      (VarPattern "x",
                       ApplyExpr
                         (ApplyExpr
                            (FunExpr
                               ("mult",Int,Arrow (Int,Int),
                                [(VarPattern "n",
                                  FunExpr
                                    ("multN",Int,Int,
                                     [(IntPattern 0,IntExpr 0),
                                      (VarPattern "m",
                                       PlusExpr
                                         (VarExpr "n",
                                          ApplyExpr
                                            (VarExpr "multN",
                                             PlusExpr (VarExpr "m",IntExpr ~1)))))])]),
                             VarExpr "x"),
                          ApplyExpr
                            (VarExpr "factorial",
                             PlusExpr (VarExpr "x",IntExpr ~1))))])],IntExpr 5))])))])
  : expr
- sub [(IntExpr 5,"x"),(IntExpr 6,"y"),(IntExpr 7,"z"),(IntExpr 8,"f")] e2bad;
val it =
  FunExpr
    ("f",Int,Arrow (Int,Arrow (Int,Int)),
     [(VarPattern "x",
       FunExpr
         ("f",Int,Arrow (Int,Int),
          [(VarPattern "y",
            PlusExpr (PlusExpr (VarExpr "x",VarExpr "y"),IntExpr 7))])])])
  : expr
- (* no tests shown for uniquifyVars, to avoid leaking ideas for solutions ☺ *)
- (* as always, we will test your submissions on inputs not shown above *)
```

Grading and Submission Notes

For full credit, your implementation must obey the formatting, documentation, performance, and complexity requirements of Assignment II. Your implementation must also (1) compile and execute on the C4 machines with no errors/warnings, (2) contain no side effects, (3) not define any top-level values beyond the functions described in this handout, and (4) not use any library (i.e., built-in) functions other than `foldr`, `foldl`, and `Int.toString`. *Unlike Assignment II, your implementations for this assignment may (and should) be recursive.*

The submission process and lateness penalties are the same as for Assignment II, except here you will be submitting your *as3.sml* file in Canvas. Please remember to include the pledge as an initial comment in your *as3.sml* file; not doing so will lower your grade 50%.

(2) **Theory portion:**
Consider the following language L:
types τ ::= bool | $\tau_1 \times \tau_2$
exprs e ::= x | true | false | e_1 NOR e_2 | (e_1,e_2) | let val $(x_1,x_2) = e_1$ in e_2 end

This language contains variables (x), true and false literals, logical-NOR expressions, binary tuples, and let-expressions. Let-expressions in L have the same meaning as in ML, except that L's let-expressions always declare a pair of variables. For example,

let val (x,y)=(true,false) in let val (x,z)=(y, x NOR y) in x NOR z end end

evaluates to true.

Provide definitions for: (1) free variables, (2) alpha-equivalence, and (3) capture-avoiding substitution 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.