Programming Languages (COP 4020/CIS 6930) [Fall 2014]

Assignment IV

Objectives

1. To become familiar with recursive data types in ML.
2. To understand basic definitions related to variables in programming languages: free variables, alpha conversion, and substitution of expressions for variables.

Due Date: Sunday, October 12, 2014 (at 11:59pm).

Machine Details: Complete this assignment by yourself on the following CSEE network computers: c4lab01, c4lab02, ..., c4lab20. Do not use any server machines like grad, babbage, sunblast, etc. You can connect to the C4 machines from home using SSH. You are responsible for ensuring that your programs compile and execute properly on these machines.

Assignment Description

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 *as4.sml*, and begin that file with the diML+P datatypes given above. Then implement the following values in *as4.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 "as4.sml";
...
- use "exprs.sml"; (* using http://www.cse.usf.edu/~ligatti/pl-14/as4/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 *as4.sml* file in Canvas. Please remember to include the pledge as an initial comment in your *as4.sml* file; not doing so will lower your grade 50%.