Programming Languages (COP 4020/6021) [Fall 2016]

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 equivalence, and substitution of expressions for variables.

Due Date: Sunday, February 28, 2016 (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 language called diML+ops, which is diML with a few extra operations. The datatypes defining diML+ops are:

```
(* diML+ops types *)
datatype typ = Bool | Int | Arrow of typ*typ; (* i.e., Arrow(argType, returnType) *)

(* diML+ops operations *)
datatype binop = Plus | Minus | Times | Less | Conjunction | Disjunction | Equal;

(* diML+ops expressions *)
datatype expr = TrueExpr | FalseExpr | IntExpr of int
  | IfExpr of expr*expr*expr (* i.e.: IfExpr(condition, thenBranch, elseBranch) *)
  | OpExpr of expr*binop*expr | VarExpr of string | ApplyExpr of expr*expr
  | FunExpr of string*typ*typ*string*expr;
  (* i.e., FunExpr(functionName, parameterType, returnType, parameterName, body) *)
```

Create a new file called *as4.sml*, and begin that file with the diML+ops 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+ops expression. The returned list should not contain any duplicates.

(b) `exception Capture`

This exception is raised whenever a substitution is attempted that could result in variable capture. Exceptions are described in Section 5.2 of the textbook.

(c) `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'. Exception *Capture* must be raised whenever the substitution isn't defined because a variable could be captured.

(d) `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.

Hints: My *fv* is 15 lines, *sub* is 15 lines, and *alphaEquiv* is 18 lines, written in about 2 hrs total.

Sample Executions:

```
- use "as4.sml";
...
- val identity = FunExpr("i",Int,Int,"x",VarExpr("x"));
val identity = FunExpr ("i",Int,Int,"x",VarExpr "x") : expr
- val f = FunExpr("f",Int,Int,"y",OpExpr(VarExpr("y"),Plus,VarExpr("z")));
val f = FunExpr ("f",Int,Int,"y",OpExpr (VarExpr #,Plus,VarExpr #)) : expr
- Control.Print.printDepth:=20;
...
val it = () : unit
- fv identity;
val it = [] : string list
- fv f;
val it = ["z"] : string list
- val subs = [(IntExpr 3,"x"),(IntExpr 4,"y"),(IntExpr 5,"z")];
val subs = [(IntExpr 3,"x"),(IntExpr 4,"y"),(IntExpr 5,"z")]
  : (expr * string) list
- sub subs identity;
val it = FunExpr ("i",Int,Int,"x",VarExpr "x") : expr
- sub subs f;
val it = FunExpr ("f",Int,Int,"y",OpExpr (VarExpr "y",Plus,IntExpr 5)) : expr
- sub [(VarExpr "y","z")] identity;
val it = FunExpr ("i",Int,Int,"x",VarExpr "x") : expr
- sub [(VarExpr "y","z")] f;

uncaught exception Capture
  raised at: as4.sml:45.62-45.69
- alphaEquiv identity (FunExpr("g",Int,Int,"h",VarExpr "h"));
val it = true : bool
- alphaEquiv f (FunExpr("f",Int,Int,"y",OpExpr(VarExpr("y"),Plus,VarExpr("x"))));
val it = false : bool
- alphaEquiv f f andalso alphaEquiv identity identity;
val it = true : bool
- (* as always, we'll 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 (though the *sub* function may raise *Capture* exceptions), (2) contain no side effects, (3) not define any top-level values beyond the functions and exception described in this handout, and (4) not use any library (i.e., built-in) functions other than `map`, `foldl`, `foldr`, and `Int.toString`. *Unlike Assignment II, your implementations for this assignment may be recursive and may define local environments (with the let keyword) and "helper functions".*

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%.