HW3
50pts

Posted Friday, October 3

Due Thursday, October 16 at midnight

Submit written part in `HW3Solutions.pdf` and code in `predictive_rec_descent.py` and `predictive_rec_descent.pl`. Submission size limit is 2.5MB.

Problem 1 (20pts). Consider the pseudocode with nested subroutines:

```
procedure main
     g : integer

     procedure B(a : integer)
          x : integer

          procedure A(x : integer)
               g := x

          procedure R(m : integer)
               write_integer(x)
               x /:= 2 -- integer division
               if x > 1
                   R(m + 2)
               else
                   A(m + 1)

          -- body of B
          x := a * a
          R(1)

     -- body of main
     B(3)
     write_integer(g)
```

a) (5pts) What does the program print under static scoping?

b) (5pts) Show the frames on the stack when A has just been called assuming static scoping rules. Show the static and dynamic links of each frame, as well as the local variables and their values right after the assignment `g := x`. Explain how `A` finds `g`.

c) (5pts) Now, what does this program print under dynamic scoping?

d) (5pts) Explain how `R` finds `x` under dynamic scoping rules.

Problem 2 (30pts). The grammar below generates boolean expressions in prefix form:

$$\begin{array}{lll} \textit{start} & \rightarrow & \textit{expr}\ \texttt{\$\$} \\ \textit{expr} & \rightarrow & \texttt{or}\ \textit{expr}\ \textit{expr} \mid \texttt{and}\ \textit{expr}\ \textit{expr} \mid \texttt{not}\ \textit{expr} \mid \texttt{id} \end{array}$$

a) (5pts) Write an attribute grammar (in pseudocode) to translate expressions into fully parenthesized infix form. For example, expression `and and a or b c d` turns into the following fully parenthesized expression `((a and (b or c)) and d)`.

b) (5pts) Now write an attribute grammar (in pseudocode again) to translate the expressions into *parenthesized* expressions in infix form *without redundant parentheses* assuming the standard convention: unary `not` has highest precedence, followed by `and`, followed by `or`, and `and` and `or` are left-associative. For example, the above expression turns into `a and (b or c) and d`. *Hint:* Assign a precedence attribute *prec* to operators and expressions. In part c) and part d) you will code your solution respectively in Python and in Prolog.

c) (10pts) Code in Python a predictive recursive descent parser for the grammar that also does infix translation without redundant parentheses. (Note that we can code such a parser because the grammar is LL(1).) For simplicity, assume that identifiers are lowercase letters from `a` to `z` and that the input has already been tokenized. Submit file `predictive_rec_descent.py` with function `parse`. `parse` takes the input list of tokens and outputs the properly parenthesized infix string if the input is in the language. It produces the empty string otherwise. Follow the structure of recursive descent from lecture (e.g., Lecture4, slides 9 and 10). Here is how Submitty runs the function:

```
python -i predictive_rec_descent.py
>>> parse(['and','and','a','or','b','c','d'])
a and ( b or c ) and d
>>> parse(['and','and','a','or','b','c'])

>>> parse(['and','a','and','b','c'])
a and ( b and c )
```

d) (10pts) Now code the same parser in Prolog. Submit file `predictive_rec_descent.pl`

```
?- parse([and,and,a,or,b,c,d],R).
R = 'a and ( b or c ) and d' ;
false.

?- parse([and,and,a,or,b,c],R).
false.

?- parse([and,a,and,b,c],R).
R = 'a and ( b and c )' ;
false.
```