HW3

50pts

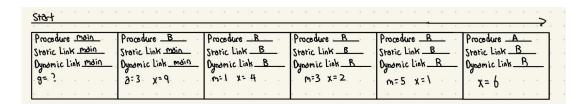
Posted Friday, October 3 Due Thursday, October 16

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?
       Output:
                 2
```

6

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?

Output: 9 4 2 1

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

Under dynamic scoping, when R executes write_integer(x), it looks for x in this order up the call chain:

- 1. It's own frame R's local vars if not found, then move onto
- 2. The caller's frame B's local vars if not found, then onto
- 3. The caller's caller's frame main's local vars, and so on

Here, R does not have its own x. So it searches up the call chain and finds x in the B frame, which it then uses continuously to reference, modify, and print out.

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

$$start \rightarrow expr \$\$$$
 $expr \rightarrow or expr expr | and expr expr | not expr | id$

- 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).
 - 1. $\exp r \rightarrow \operatorname{or} \exp r_1 \exp r_2$ $\exp r.val := "(" + \exp r_1.val + " \text{ or "} + \exp r_2.val + ")"$
 - 2. $\exp r \rightarrow \text{ and } \exp r_1 \exp r_2$ $\exp r.val := "(" + \exp r_1.val + " and " + \exp r_2.val + ")"$
 - 3. $\exp r \rightarrow \text{not } \exp r_1$ $\exp r.\text{val} := \text{"(not "} + \exp r_1.\text{val} + \text{")"}$
 - 4. $\exp r \rightarrow id$ $\exp r.val := id.s$
- 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 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.

First, let's assign two synthesized attributes to the expressions/terminals:

- 1. val the resulting infix string
- 2. prec it's precendence (higher number means higher precedence)

Then, synthesize precedences to the operators/terminals(prec(operator))(Higher number means higher precedence):

$$\operatorname{prec}(\operatorname{id}) = 4 \quad \operatorname{prec}(\operatorname{not}) = 3 \quad \operatorname{prec}(\operatorname{and}) = 2 \quad \operatorname{prec}(\operatorname{or}) = 1$$

Now, let's write the attribute grammar with helper functions $\operatorname{paren}_L(E, p)$, $\operatorname{paren}_R(E, p)$, and $\operatorname{paren}_{un}(E, p)$ that adds parentheses to the left, right, or unary expression E if its precedence is lower than p.:

Helper functions:

$$\operatorname{paren}_L(E,p) = \begin{cases} (E.\operatorname{val}) & \text{if } E.\operatorname{prec} < p, \\ E.\operatorname{val} & \text{otherwise.} \end{cases}$$
$$\operatorname{paren}_R(E,p) = \begin{cases} (E.\operatorname{val}) & \text{if } E.\operatorname{prec} \le p, \\ E.\operatorname{val} & \text{otherwise.} \end{cases}$$
$$\operatorname{paren}_{\operatorname{un}}(E,p) = \begin{cases} (E.\operatorname{val}) & \text{if } E.\operatorname{prec} < p, \\ E.\operatorname{val} & \text{otherwise.} \end{cases}$$

CONTINUED ON NEXT PAGE

Attribute Grammer:

 $\begin{array}{lll} 1. & \exp r \rightarrow \text{or } \exp r_1 \, \exp r_2 & & \exp r. \operatorname{prec} := 1 \\ & \exp r. \operatorname{val} := \operatorname{paren}_L(\exp r_1, 1) \,\, + \,\, " \,\, \operatorname{or} \,\, " \,\, + \,\, \operatorname{paren}_R(\exp r_2, 1) \end{array}$

 $\begin{array}{lll} \text{2.} & \exp r \rightarrow \text{and } \exp r_1 \, \exp r_2 & & \exp r. \operatorname{prec} := 2 \\ & & \exp r. \operatorname{val} := \operatorname{paren}_L(\exp r_1, 2) \, \, + \, \, " \, \, \operatorname{and} \, " \, \, + \, \operatorname{paren}_R(\exp r_2, 2) \\ \end{array}$

 $\begin{array}{ll} 3. & & \exp r \to \operatorname{not} \ \exp r_1 & & \exp r.\operatorname{prec} := 3 \\ & & \exp r.\operatorname{val} := \operatorname{``not''} + \ \operatorname{paren}_{un}(\exp r_1, 3) \end{array}$

4. $\exp r \rightarrow id$ $\exp r.prec := 4$ $\exp r.val := id.s$