

**CS480/580: Introduction to Artificial Intelligence**  
**Fall 2009**  
**Course Project**  
**100 Points**  
**Due: December 3, 2009**

- (1) Given the 8-piece puzzle with the initial state and the final state, using the A\* algorithm write an algorithm to transform the given puzzle from the initial to the final state. (Use any language of your choice.) Your submission includes: (i) Readme file (ii) Source code (iii) Executable (iv) Sample inputs and outputs (v) Report indicating lessons learned. The output should include the sequence of steps to be taken to go from initial to final state. Make sure to summarize the outputs which indicates the number of nodes evaluated for each input.
- (2) Given the following grammar, build a Prolog parser to parse the input expressions. If the input does not follow the rules of the grammar, then say so. Otherwise, print the parser output. For example, if  $a+b*c$  is given as input, the output should look something like this:
- $\langle \text{Exp} \rangle \Rightarrow \langle \text{Exp} \rangle + \langle \text{Term} \rangle \Rightarrow \langle \text{Term} \rangle + \langle \text{Term} \rangle \Rightarrow \langle \text{Factor} \rangle + \langle \text{Term} \rangle \Rightarrow x + \langle \text{Term} \rangle \Rightarrow a + \langle \text{Term} \rangle * \langle \text{Factor} \rangle \Rightarrow a + \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \Rightarrow a + b * \langle \text{Factor} \rangle \Rightarrow a + b * c$

**A grammar for simple arithmetic expressions**

**Production rules:**

$\langle \text{Exp} \rangle ::= \langle \text{Exp} \rangle + \langle \text{Term} \rangle \mid \langle \text{Exp} \rangle - \langle \text{Term} \rangle \mid \langle \text{Term} \rangle$   
 $\langle \text{Term} \rangle ::= \langle \text{Term} \rangle * \langle \text{Factor} \rangle \mid \langle \text{Term} \rangle / \langle \text{Factor} \rangle \mid \langle \text{Factor} \rangle$   
 $\langle \text{Factor} \rangle ::= a \mid b \mid c \mid d \mid e \mid ( \langle \text{Exp} \rangle ) \mid - \langle \text{Factor} \rangle$

Non-terminal symbols:  $\langle \text{Exp} \rangle, \langle \text{Term} \rangle, \langle \text{Factor} \rangle$

Terminal symbols:  $+, -, *, /, (, ), a, b, c, d, e$

Start symbol:  $\langle \text{Exp} \rangle$

Note the following are meta symbols:

$::=$  (read it as "can be replaced by")

$\rightarrow$  is alternative notation for  $::=$ ,

$\mid$  (read it as "or by")

e.g. An  $\langle \text{Exp} \rangle$  can be replaced by  $\langle \text{Exp} \rangle + \langle \text{Term} \rangle$  or by  $\langle \text{Exp} \rangle - \langle \text{Term} \rangle$  or by  $\langle \text{Term} \rangle$ .

Note that the production (replacement) rules for  $\langle \text{Exp} \rangle$ ,  $\langle \text{Term} \rangle$  and  $\langle \text{Factor} \rangle$  are recursive. The rule for  $\langle \text{Exp} \rangle$  states that an Expression is a list of Terms separated by '+' or '-' symbols. A Term is a list of Factors separated by '\*' or '/' symbols. A Factor is a variable (x, y, ...), or a parenthesized sub-expression, or a unary '-' followed by a Factor.

The operators '\*' and '/' bind more tightly to their operands than '+' and binary '-' because '\*' and '/' appear in the rule for  $\langle \text{Term} \rangle$ , closer to the operands ( $\langle \text{Factor} \rangle$ ). Unary '-' binds most tightly of all operators.

The rules for  $\langle \text{Exp} \rangle$  and  $\langle \text{Term} \rangle$  are *left recursive*. This is because '+', '-', '\*', and '/' are *left associative*: 'a-b-c' gives the same parse tree as '(a-b)-c', not 'a-(b-c)'.

A grammar like this can be turned into a "recursive descent parser" by writing a routine for each non-terminal in the grammar. The routines call each other as specified by the replacement rules. Left recursion, that is a replacement that begins with a recursive non-terminal, must be removed so that the parser does not loop.

What to submit:

- (i) Readme file (ii) Source code (in prolog) (iii) Sample inputs and outputs (iv) Summary report indicating lessons learned.