Homework 2 . Context free grammar

Submit a PDF file of your solution to blackboard by 11:59pm Monday Feb 22

1. a) (5) The name(s) of people who have contributed to your solution of this homework, and b) their contribution (briefly). If you worked by yourself, the answer of this question would be "N.A."

   b) (5)Please make your writing for the homework easy to read. Acknowledge this.

2. (15) Show the NFA that results from applying the NFA construction method, discussed in the class, to the regular expression `( a | b | 0 )^ b`.

3. (40) Context Free Grammar.

   (a) Write a context free grammar for arithmetic expressions which can use numbers, variables and operation + only.

   (b) Write a context free grammar for the arithmetic expressions above that captures right associativity.

   (c) Write a context free grammar for arithmetics expressions which can use numbers, variables and binary operations − and + only. Your grammar has to capture the precedence that − must be computed before +. Your grammar should also capture the left associativity of + and −. For example $5+5-5-5+6-5$ should be computed as $5 + (5 - 5 - 5) + (6 - 5)$.

   (d) Refine your grammar above to allow parenthesis (and).

   Hint. Recall the techniques on how precedence and associativity were dealt with in the class.

4. (15) Given a grammar

   $< P > \rightarrow < S >$
   $< S > \rightarrow lrp\ < S >\ rrp\ < S > \mid lsp\ < S >\ rsp\ < S > \mid \epsilon$

   The terminals are defined as follows

   $lrp \rightarrow$ (
   $rrp \rightarrow$)
   $lsp \rightarrow$ [
   $rsp \rightarrow$]

a) Draw a parse tree for each of the sentences:

`()`

`(()[]).`

b) Write a rightmost derivation for `[]`.

5. (20) Parsing.

   Consider the CFG for `<program>` in the slides (of L6) for parsing (Page 16), and an input program

   `sum := A + B write sum.`

   (a) Trace the table driven parser as done in the class discussion. You can use a table for tracing as we did in L6 slides.

   (b) Trace the recursive descent parser (starting from the main function in P29 of L6 slides). You may write the function call sequences (and changes of global variable input_token) in executing the parser on the input program as shown below. If any function is missing, you can create your own following the problem decomposition method we used during class. **Note.** To design the recursive descent parser, the best way is still the use of problem decomposition method following each production. This tracing just help you to see that it does work.

```
program()
|--stmt_list() // input_token: sum
|----stmt() // input_token: ?
|------ ...
|----stmt_list() // input_token: ?
|------ ...
|--match($$)  outputs true/false which is returned
```