**Sample final exam questions**

1. Consider the following grammar.

S →S ( S )

S →x

* 1. Is the grammar ambiguous? (Can you obtain two parse trees from the same input? Module 4, slide 37) (no you can’t, so it is not ambiguous)
  2. Is it suitable for top-down parsing? (NO, it’s left recursive)
  3. Is it suitable for bottom-up parsing? (Yes, it’s not ambiguous)

1. What characteristics of a programming language is best specified using regular expressions? Using a context-free grammar? (We use RegEx for lex for recognition of tokens, Scanner is used for regex. Using CFG is just regular grammar actually if we use it for token recognition, and RegEx is more compact) (There’s no semantics though.)
2. What is some of the information that is found in a symbol table?
3. If Σ = {0,1}, write a regular expression whose language is all strings containing exactly three 1s. (should be able to do this, ad hoc) [0\*10\*10\*10\*]
4. Consider the grammar   
    S → *a*S*a* | *b*S*b* | *c*   
   Draw the parse tree for the string *ababcbaba*.

//(get img from Stian)



1. For the **lex** specification given below, what is the output if the string **abbbaabcaaaaaaebbbbbbaaabbbaa** is the input?

First get longest string possible

If there’s 2, get the one that comes first  
  
**a { /\* print out a ‘\*’ \*/ }**

**aa { /\* print out first character in matched string \*/ }**

**a+ ;**

**b { /\* print out a ‘\*’ \*/ }**

**bbb { /\* print out first character in matched string \*/ }**

**b+ ;  
 . ;**

**OUTPUT: \*ba\*ba**

1. Consider this recursive descent parser. Assume match() is defined as given in class. What does the associated grammar look like? What is the input alphabet? (Assume $ is the end-of-input marker.)

**S() {**

**if (lookahead == ‘a’) { match(‘a’); S(); match(‘b’); }**

**else if (lookahead == ‘x’ or lookahead == ‘y’) { A(); }**

**else error();**

**}**

**A() {**

**if (lookahead == ‘x’) { match(‘x’); A(); }**

**else if (lookahead == ‘y’) { match(‘y’); B(); }**

**else error();**

**}**

**B() {**

**if (lookahead == ‘z’) {match(‘z’); B(); match(‘c’); }**

**else if (lookahead == $ or lookahead == ‘b’) return;**

**else error();**

**}**

**S 🡪 aSb | A**

**A 🡪 xA|yB**

**B 🡪 zBc|∑**

1. Construct an NFA for the following regular expression using the algorithm presented in class  
    a(b|c)\*a (Module 3, Slide 53)
2. Remove left recursion from the following grammar: (module 4 slide 56-60)  
    S -> S a A | A  
    A -> A b B | B  
    B -> C c B | C  
    C -> d S e | f

S 🡪

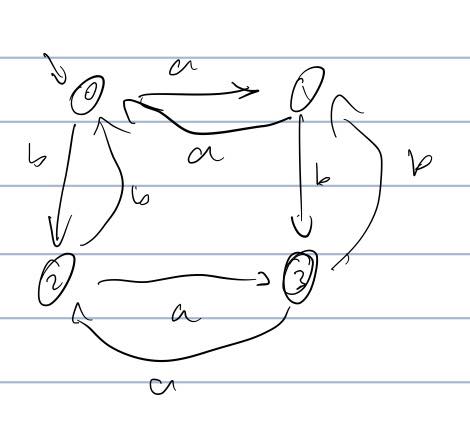
A 🡪

B 🡪

C 🡪

R 🡪

1. Construct a DFA which accepts all strings over the alphabet {a,b} with an odd number of a’s and an odd number of b’s.

(Should be able to do this, ad hoc)  


1. Write a regular expression for C-style comments, that is, comments starting with /\* and ending with \*/

(More challenging than it looks)

/\\*([^\\*]\*(\\*[^/])?)\*\\*/

-- ? == 0 or 1

NFA to DFA

From IMG

A = {0}

B = {1,2,3,5,8,9}

© = {10}

D = {4,7,2,3,5,8,9)

E = {6,7,2,3,5,8,9}

Not minimized:

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b | c |
| A | B | - | - |
| B | C | D | E |
| C | - | - | - |
| D | C | D | E |
| E | C | D | E |