COSC 461/561

Midterm 1 – Preparation Questions

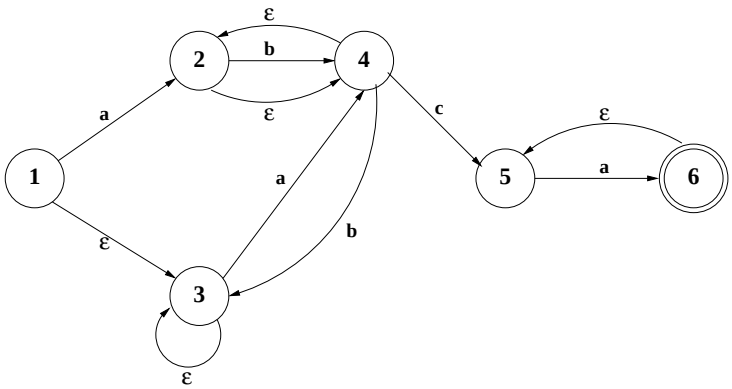
1. Show that the following grammar is ambiguous using the string: **( )**. Explain why.
   S $\rightarrow$ S S | (S) | ϵ

2. Use the construction algorithm in the book and illustrated in class to build a nondeterministic FSA for the following regular expression. Your answer should show the steps for constructing the transition diagram. Also draw the parse tree.

$$((ab|\epsilon)^*c)^*$$

(extra page for constructing nondeterministc FSA ...)

3. Give a DFA equivalent to the NFA below using the algorithm in the book and illustrated in class. Express your answer as a transition table. Also, identify the start state and final state(s). The initial state is state 1.



4. Minimize the following DFA using the algorithm in the book and illustrated in class. Set of final states, F = {4,5,6,7}, Initial state S = 0, Input alphabet, Σ = {a,b}. Express your answer as a transition diagram.

State	Input	
	a	b
0	1	2
1	1	3
2	2	3
3	5	4
4	6	-
5	7	-
6	-	-
7	-	-

5. Given the following grammar:

1. $A \rightarrow AB \mid c$
2. $B \rightarrow AC \mid a$
3. $C \rightarrow B \mid b$

use the algorithm presented in class to eliminate left recursion. Indicate the final set of productions.

6. Consider grammar 4.28 and corresponding parsing table in Fig. 4.17 from the text. Show the moves made by a LL(1) predictive parser for the input string: **(id * id)**.

7. Build a recursive descent parser for the following grammar.

   A $\rightarrow$ aBB | cAcB
   B $\rightarrow$ bca | c

8. Construct the sets of FIRST and FOLLOW for each non-terminal in the grammar below, using the algorithms we saw in class. Then, construct a predictive parsing table for this grammar.

Production	FIRST		FOLLOW	
S $\rightarrow$ aAF	{	}	{	}
A $\rightarrow$ B | C	{	}	{	}
B $\rightarrow$ D+	{	}	{	}
C $\rightarrow$ E*	{	}	{	}
D $\rightarrow$ x | y | z | ϵ	{	}	{	}
E $\rightarrow$ a | b | c	{	}	{	}
F $\rightarrow$ A | ϵ	{	}	{	}

9. Is the following grammar LL(1)? Prove your answer.

   stmt → label unlabeled_stmt
   label → ID : | ϵ
   unlabeled_stmt → ID := expr ;
   expr → ID | NUM

10. Write a (C or C++) routine called `print_identifiers` that prints the line number and column number of each identifier that occurs in an input program provided on standard input. You should make the following assumptions:

    (a) Comments are not allowed.

    (b) Whitespace is allowed, but ignored.

    (c) Delimiters and all other punctuation are allowed, but ignored.

    (d) Numeric literals are allowed, but ignored.

    (e) String literals are allowed, but identifiers that appear within a string literal are ignored.

    (f) Each identifier is at most 64 characters long.

    (g) There are no keywords (or keywords are simply matched as identifiers).

    (h) You have access to the `get_char` routine from the cscan assignment.

    (i) `get_char` updates the global integer variables `cur_line` and `cur_column` so that they correspond to the current position of the input cursor.

    (j) You can use the standard C or C++ libraries to complete your implementation.

    Example input:

```
int i = 0;
int sum = 0;

fprintf(stdout, "computing sum\n");
for (i < 10) {
    sum += i;
    i++;
}
```

    Example output:

```
line: 1    column: 1    lexeme: int
line: 1    column: 5    lexeme: i
line: 2    column: 1    lexeme: int
line: 2    column: 5    lexeme: sum
line: 4    column: 1    lexeme: fprintf
line: 4    column: 9    lexeme: stdout
line: 5    column: 1    lexeme: for
line: 5    column: 6    lexeme: i
line: 6    column: 3    lexeme: sum
line: 6    column: 10   lexeme: i
line: 7    column: 3    lexeme: i
```

11. Short answers:

    (a) Define a compiler.

    (b) Explain the difference between a compiler and an interpreter. Give one example of interpreted language and compiled language.

    (c) Name and provide a one-line description of the six compiler phases.

    (d) What is importance of intermediate code in compiler construction?

    (e) Explain assemblers, linkers, loaders.

    (f) Explain recursive-decent parsing.

    (g) Why use grammars for language specification?

    (h) Give an example of syntactic checking and type checking.

    (i) Describe the basic approach for implementing recursive-decent parsers.

    (j) What is precedence climbing? How does it work in a recursive descent parser?

    (k) How is bottom-up parsing different from top-down parsing?

    (l) Why is bottom-up parsing (with one symbol of lookahead) more powerful than top-down parsing (with one lookahead symbol)?