CS 3101 Practice Quiz 1

Pages: 4 printed sides

Name: _____ Time allotted: **45 minutes + 5 min scan/upload**

Maximum score: **36 points**

University rules dictate strict penalties for any form of academic dishonesty. Looking sideways will be penalized. Look at only your own exam at all times.

There are 7 questions, some with subparts. **Read them *carefully* to avoid throwing away points!!** Write your answer in the space provided. *Closed book, closed notes. Calculators are allowed.*

Partial credit rule: *Must show your intermediate steps clearly for partial credit!*

---

1. Fill in the blanks with at most a few words:

(1 point * 6 = 6 points)

(a) The _____Loader_____ is a part of the Operating System, which when a program is about to run, allocates the memory segments needed for it, and performs dynamic linking of libraries.

(b) In a compiler infrastructure, the part of the compiler that is rewritten to support each new instruction set is called the _____backend_____.

(c) In lexical analysis, according to the _____longest match_____ rule, the longest initial substring of the input that can match any regular expression is taken as the next token.

(d) Often control and data flow analysis is done on ___Intermediate Representation_____ instead of assembly (low level) code.

(e) Statements that define non-terminals in terms of terminals and non-terminals are called _Production Rules_____ .

(f) In Context Free Grammars, the act of finding out if a string is a member of presented grammar by the following a sequence of expansion rules is called a ___Derivation_____.

2. For each subpart (i) to (ii) below, circle ***all*** correct answers from among the four given - note that ***more than one, one, or none*** of the answers may be correct! (*Grading note*: each part will be graded as four independent questions, each worth ½ point.)

(2 points * 2 = 4 points)

(i) Regarding parsing,

(a) an ambiguous grammar is one in which there is more than one derivation for a given input being accepted by the grammar.

(b) CFGs are in general implemented by DFAs.

(c) unambiguous grammars are often more complex than ambiguous grammars for the same input language.

(d) CFGs can implement a greater set of languages than regular expressions can.

(ii) In lexical analysis,

(a) regular expressions are used to specify the set of strings accepted for each token.

(b) compiler writers in modern compilers usually write the code for the finite automaton(s) required for the language.

(c) There is only one unique regular expression possible for each language that can be accepted by regular expressions.

(d) rule priority ensures that "if8" is recognized as a single identifier, instead of a keyword followed by a number.

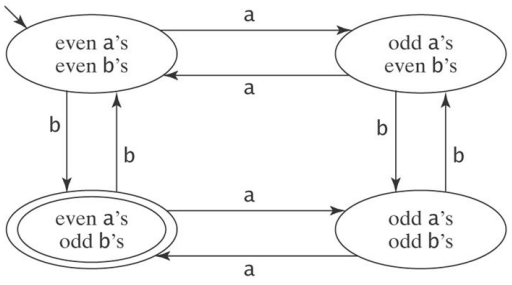
3. Write a regular expression for the language of strings containing only a's and b's that contain an odd number of a's.

(4 points)

b* a ((ab*a) | b)* OR
b*ab*(ab*ab*)*
Both of the above answers are correct. It is possible there may be other correct answers too.

4. Draw a DFA for the language of strings containing only a's and b's that contain an even number of a's and odd number of b's.

(4 points)



5. Below is a small **subset** of the CFG rules for statements and the if-then construct in a programming language:

STMTLIST → STMTLIST STMT SEMICOLON
STMTLIST → ε

IFTHEN → IF LPAREN COND RPAREN LCURLY STMTLIST RCURLY

Write a suitable abstract CFG for the above grammar subset.

(4 points)

STMTLIST → STMTLIST STMT
STMTLIST → ε

IFTHEN → COND STMTLIST

6. Consider the following grammar. **(3+3 = 6 points)**

S → S + S
S → S * S
S → id

(a) Write an unambiguous grammar for the ambiguous grammar above such that multiplication has higher precedence compared to addition.

S → S + M
S → M
M → M * id
M → id

(b) Write the leftmost derivation for the string id + id * id for the original ambiguous grammar above.

Leftmost derivation:

S
S + S
id + S
id + S * S
id + id * S
id + id * id

7. The C programming language includes switch statements. Here is an example:

(8 points)

```
switch(i) {
    case 1 : c = '1';
             break;
    case 2 : c = '2';
             break;
    default: c = 'e';
             break;
}
```

Write a context-free grammar for accepting switch case statements like the one above. Your grammar must be unambiguous. IMPORTANT: You should use STATEMENT as a non-terminal without attempting to expand it any further. (In other words, STATEMENT is expanded by other parts of the grammar which you are not responsible for. Please note that STATEMENT non-terminal does not include the SEMICOLON token at its end.)

Here are the assumptions:

- A token ID that recognizes identifiers and variables such as i is available.
- A token NUM that recognizes integer numbers such as 1 or 2 is available.
- Tokens switch, case, default, and break are available to recognize those keywords.
- Switch statements can only accept variables like i as arguments. The only variable type that is passed to a switch statement is an integer.
- Each switch statement must have at least one (non-default) case.
- Each case statement should be followed by a value for the identifier in the switch.
- The body of each case statement is zero or more STATEMENTs which must be followed by a BREAKSTATEMENT.
- The presence of "default" is optional.
- Use descriptive non-terminal and terminal names like SWITCH (non-terminal) and SEMICOLON (terminal). (**Caution:** The TA *will not grade your answer* if you use non-descriptive names!!)

SWITCH → switch LPARENTHESIS ID RPARENTHESIS LCURLY CASES RCURLY
CASES → CASE CASES DEFAULTQUESTION
CASES → ε
CASE → case NUMBER COLON STATEMENTLIST BREAKSTATEMENT
STATEMENTLIST → STATEMENTLIST STATEMENT SEMICOLON
STATEMENTLIST → ε
DEFAULTQUESTION → default COLON STATEMENTLIST BREAKSTATEMENT
DEFAULTQUESTION → ε
BREAKSTATEMENT → break SEMICOLON