Design and Implementation of a Compiler
Midterm Exam
Spring 2012

1. True / False Write a T if the statement is true or an F if the statement is false.

a. The language defined by a grammar can never anything except sentences derivable from that grammar.

b. All valid sentences of a grammar are sentential forms of that grammar.

c. Every LL(1) grammar is unambiguous.

d. Every unambiguous grammar is LL(1).

e. Nested comments can be recognized by a regular expression.

f. If you have a DFA and that DFA has precisely one cycle, and if that cycle is a self-loop (ie, a cycle with a path length of one), then that DFA describes a language of infinite size.

g. Consider a parse tree for some sentence of a certain grammar. All non-leaf nodes in this parse tree are non-terminals.

h. Compare the following two regular expressions: a*(a | b)* and (a | b)*. Here, the | symbol represents the OR operator. Is it true that these two expressions define the same language?

i. The following grammar is LL(1):

A → BC | CB
B → aBb | x
C → aCb | y

j. LL parsers are often implemented with a bottom-up recursive-descent parser.

k. A grammar is not LL(1) if two productions for the same nonterminal have a common prefix.

l. Tokens in lexical analysis are usually defined by regular expressions.

m. Every sentence of a context free language is a sentential form of the language.

n. NFA's are more efficient to simulate than DFA's.

o. DFA's are more efficient to create from regular expressions than NFA's.

p. The root of a parse tree is a terminal symbol.

q. Equivalent grammars define the same language.

r. Inherited attributes are a subset of all synthesized attributes.

s. Let Rev be the operator that reverses strings. For example, Rev(abc)=cba. Let R be any regular expression. Rev(R) is the set of strings denoted by R, with each string reversed. True or False: Rev(R) a regular set.

t. Compared to BNF, EBNF grammars are better-suited for efficient recursive descent parsing.

u. An LL grammar cannot have right recursion.

v. After left-factoring and left-recursion removal all LL grammars can be made into LL(1) grammars.

2. Consider the set of all languages definable by each of the following formats.

A. CFG	**E**. EBNF	**L**. regular expression	**R**. LL(2)
B. BNF	**H**. NFA	**M**. rectangular grammar	**S**. LL(*)
C. CSG	**J**. OSG	**P.** rectangular expression	**T**. LR
D. DFA	**K**. regular-grammar	**Q**. LL(1)	**V**. Type 0 grammars

W. An NFA for which there is a unique final state, and every-other state has either 1 or 2 successors
X. A BNF grammar with semantic actions added
Y. Those grammars parsable with a recursive descent parser having the ability to look ahead 2 tokens.
Z. GNF - Greibach Normal Form (but including the possibility of the empty sentence, if desired).

a. Group these choices (by letter) into groups of equivalent expressive power – in other words, into groups which are capable of defining the same set of languages. You will also have an additional group, titled "Doesn't Exist" (because I added some imaginary names to the list.) Your answer should look like this:
將上面 A~Z 選項，可以定義同一種語言的選項填入一樣的類別裡。另外因爲 A~Z 選項裡有些爲虛構的，所以可以看到答案裡應該會有一組額外的類別名爲 "Doesn't Exist"。你的答案必須寫像下面這樣的格式:

Doesn't Exist	G1	G2	G3	G4	. . .	G *n*
list of letters	*list of letters*	*list of letters*	*list of letters*	*list of letters*		*list of letters*

b. Here is a subset of seven choices from above:

A. CFG	**D**. DFA	**Q**. LL(1)	**V**. Type 0 grammars
C. CSG	**J**. OSG	**T**. LR	

Order these choices according to increasing expressivity. If any of these choices don't exist, then don't include them in the list. So your answer should be something like:
(將上述 7 個的選項根據表達式遞增的關係來排序，如果這些選項裡有不存在的則不需要填入排序內。你的答案必須寫像下面這樣的格式。

SomeLetter	<	SomeLetter	<	SomeLetter	=	SomeLetter	<	SomeLetter	<	SomeLetter

- In the above sample answer, notice that there are only 6 letters. This is the answer that you would get if you believe that one of the seven choices doesn't exist.
- Also, in the above example, notice that there is an = sign between the third and fourth choices. This is the answer that you would get if you thought that two of the choices had equal expressivity.
- This means that you answer must include the **<** or **=** symbols between your letters.

(切記以上的格式只是一個例子，排序出來的 letter 數不一定只有 6 個，可能是 7 個或更少。 "=" 不一定只有一個，且不一定出現於第 3 個 letter 和第 4 個 letter 之間。)

Let me be more clear: I don't mean that your answer must have 6 letters. I don't mean that your answer must have an = between the 3rd and 4th letters. No. The above is just an example of the format of the answer. Another example of the format would be:

SomeLetter	<	SomeLetter	=	SomeLetter	=	SomeLetter

In this alternative answer, you are saying that only four of the letters are real grammars. And you are saying that they only belong to two groups, because the 2nd, 3rd, and 4th have the same expressivity. (therefore, of course, the 2nd, 3rd, and 4th can be listed in any order).

3. Multiple choice. Write the letter**s** of **all valid answers** to each question.
In these grammars: **S** is the **start** symbol; **uppercase letters** are **nonterminals**; **lowercase letters** are **terminals**; and all other characters are special symbols. For regular expressions, the special symbols for the lex format. For grammars, they are BNF format.

a. Consider the language defined by the regular expression (a|b)*b+ . Which of the following regular expressions also define that language?

A. (a*b+) | (b*b+) **B**. (ab | bb)* b* **C**. (a | b | ba)*b+ **D.** none of these

b. Consider the language defined by the grammar: S → E {xE} E→ y
Which of the following grammars also define that language?

A. S→ y | SxS **B**. S→ E | ExS E→ y **C**. S→Sxy | y **D.** none of these

c. Consider the grammar answers in the previous question. Which of them are ambiguous?

A. S→ y | SxS **B**. S→ E | ExS E→ y **C**. S→Sxy | y **D.** none of these

d. Consider the following FORTRAN statement or statement fragment: DO 10 I = 1.100
How many tokens does this statement have?

A. 1 **B**. 2 **C**. 3 **D**. 4 **E**. 5 **F**. 6 **G**. 7 **H**. 8 **J**. It is an illegal statement

4. Specify minimum-sized BNF grammar for each of the following. Or, if no such grammar exists, then just say: “It doesn’t exist”. (Or, if you think it does exist, but you don’t know how to write it, you can get partial credit by saying: “It exists”.)
根據以下的敘述定義出 minimum-sized BNF grammar。 如果沒有語法存在，那只要寫“It doesn’t exist”。(如果你覺得語法存在，但你不知道如何寫出來，你寫“It exists”，將得到部分的分數。)

a. L = { wcw | w is in (a | b) * } (ie, w is a string of a’s and b’s that repeats with a **c** in between)

b. L = { wcw^R | w is in (a | b) * } where w^R stands for w reversed. In problem #1 part s, this was called Rev(w).

c. L = {$a^n b^m c^m d^n$ | n≥1 *and* m≥ 1}

d. L = {$a^n b^n c^m d^m$ | n≥1 *and* m≥ 1}

e. L = {$a^n b^m c^n d^m$ | n≥1 *and* m≥ 1}

f. L = { $a^n b^n c^n$ | n≥0 }

g. L = { $a^n b^n$ | n≥1 }

5. Consider the following grammar: S→ aSS | Sb | a | λ

a. Give a leftmost derivation of the string abba

b. Construct the parse tree for abba

6. Convert the following grammar to an equivalent, minimum-sized EBNF format:

stmt-sequence → stmt **';'** stmt-sequence
| stmt
stmt → **a**

7. Give an example of a grammar with:

a. An unreachable terminal

b. A terminal that is reachable but that derives no input string

8. Consider the following partial grammar, in EBNF format:

start → memvar
| method
memvar → MODIFIER ID ID ['=' expr] ';'
method → MODIFIER ID ID '(' args ')' morestuff

In this grammar, terminals are indicated by UPPER-CASE LETTERS or by quotes (').
Notice that the [and the] don't have quotes because they are special symbols.
Also, in this grammar, there are some nonterminals that are not shown.

a. Choose the *smallest* category to which the above grammar fragment belongs:

A. LL **B**. LL(1) **C**. LL(2) **D**. LL(3) **E**. LL(4) **F**. LL(5) **G**. LL(6)
H. LR **I**. LR(1) **J**. LR(2) **K**. LR(3) **L**. LR(4) **M**. LR(5) **N**. LR(6)
O. None of the above

b. What is the full list of valid tokens that will belong to the prediction set?

9. Consider that a certain production has the following format:

$A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$

where no β_i begins with an A.

Convert the above grammar fragment into a non-left-recursive form

10. Prove that $S \rightarrow SS | a$ is ambiguous

11. Consider the following grammar (where terminals are lower-case):

S → iEtS | iEtSeS | a

E → b

a. What is the left-factored equivalent?

b. Is the grammar ambiguous?

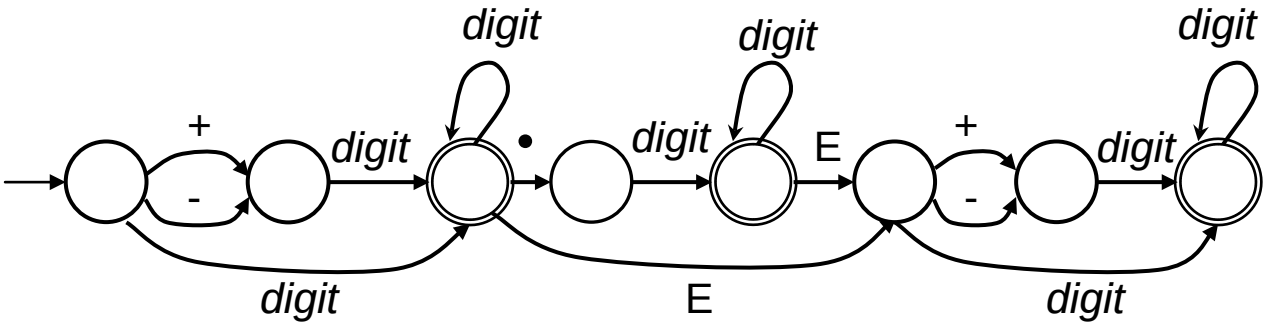
12. Given the partial answers shown below, fill in the outside portion to obtain the indicated composition (by adding states):

給定部分的答案展示如下。將外面的部分填寫完成，以得到指定的 composition。(composition: 表示結合律的意思，就是將 A 和 B 這兩個 state 用其他 state 串在一起。)

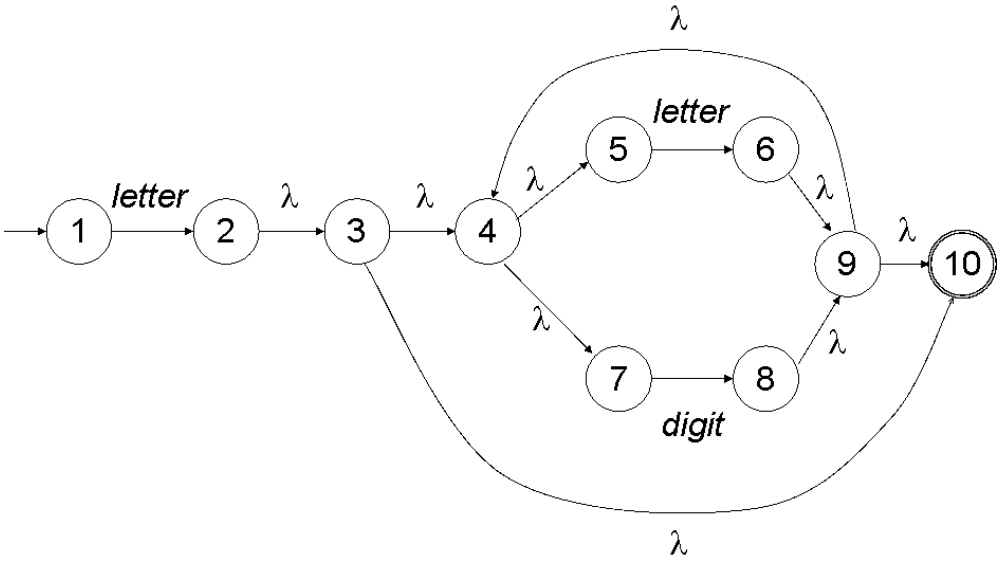
a. To obtain A | B ：

b. To obtain A* ：

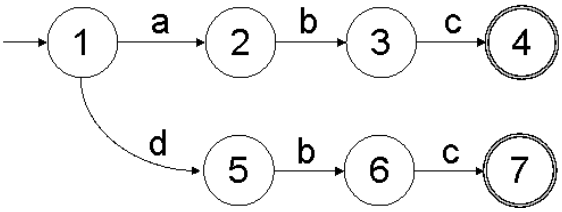
13. Write the regular expression for the following DFA, in lex format.



14. Write the equivalent, minimum-state DFA for the following:



15. Suppose that we wished to reduce the following DFA to a smaller number of states, by means of the algorithm in the book.



a. For the first partition, what would the sets of states be?

b. For this first partition draw the augmented DFA, with the trap state for the errors. All edges on your new DFA must be labeled.

c. What is the final, simplified DFA, after finishing the algorithm?

16. Draw the transition table for the transition diagram on the right: