Ministry of Higher Education
El-Shorouk Academy
Higher Institute for Computer & Information Technology

Department : Computer Science
Instructor : Dr. Ahmed elabbassy
Reviewer1 : Dr. Khaled Menshawy
Reviewer2 : Dr. M. El-Hennawy

Acad. Year	:	2017 / 2018
Term	:	Second
Year	:	4th
No. Questions	:	4
Date	:	24/05/2018
Time Allowed	:	2 Hours
Max. Marks	:	80 Marks
No. of pages	:	2

4203 Compiler Theory (Final Exam)-Sol

Answer all of the following questions:

Question 1 (20 Marks)

1.1 Consider the following grammar for Boolean expressions (where id stands for the terminal identifier"):

Bexp → ***Bexp*** **or** ***Bterm*** | ***Bterm***
Bterm → ***Bterm*** **and** ***Bfact*** | ***Bfact***
Bfact → **not** ***Bfact*** | **(** ***Bexp*** **)** | **id**

a) What is the start symbol in the above grammar?

E

b) What are the terminal and the non-terminal symbols in the above grammar?

Terminals: id ( ) or and not

non-terminal: *E T F*

c) Construct a leftmost derivation for the following sentence: not id and id

E → *T*
→ *T* **and** *F*
→ *F* **and** *F*
→ **not** *F* **and** *F*
→ **not id and** *F*
→ **not id and id**

d) Eliminate the Left-Recursion from the above grammar.

E → *T*{`or` *T*}
T → *F*{`and` *F*}
F → {`not`}*B*
B → `true` | `false` | (*E*)

1.2 What do we mean when we say that a grammar is "ambiguous"? Mention two techniques for eliminating ambiguity from a grammar

An Ambiguous Grammar

- **A grammar that generates a string with *two distinct parse trees***

Two Basic Methods dealing with Ambiguity

- **One is to state a rule that *specifies in each ambiguous case which of the parse trees (or syntax trees) is the correct one,* called a disambiguating rule.**
  - **The advantage: it corrects the ambiguity without changing (and possibly complicating) the grammar.**

- **The disadvantage: the syntactic structure of the language is no longer given by the grammar alone.**

- **Change the grammar into a form that forces the construction of the correct parse tree, thus removing the ambiguity.**

1.3 Explain why Top Down parsers cannot handle Left Recursive Grammars.

On a left-recursive grammar there is a possibility that the algorithm will try to expand a production without consuming any of its input. In this case, guessing right does not help as it due to the recursion the tree is continuously being expanded without any advanced in the input symbols? Thus the parsing process never terminates.

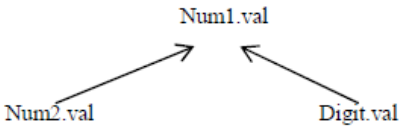
Question 2 (20 Marks)

2.1 Given the following BNF grammar and associated semantic rules:

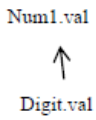
Grammar Rule	Semantic Rules
$number_1 \rightarrow number_2\ digit$	$number_1.val = number_2.val * 10 + digit.val$
$number \rightarrow digit$	$number.val = digit.val$
$digit \rightarrow 0$	$digit.val = 0$
$digit \rightarrow 1$	$digit.val = 1$
$digit \rightarrow 2$	$digit.val = 2$
$digit \rightarrow 3$	$digit.val = 3$
$digit \rightarrow 4$	$digit.val = 4$
$digit \rightarrow 5$	$digit.val = 5$
$digit \rightarrow 6$	$digit.val = 6$
$digit \rightarrow 7$	$digit.val = 7$
$digit \rightarrow 8$	$digit.val = 8$
$digit \rightarrow 9$	$digit.val = 9$

a) Draw the dependency graph for the semantic rules

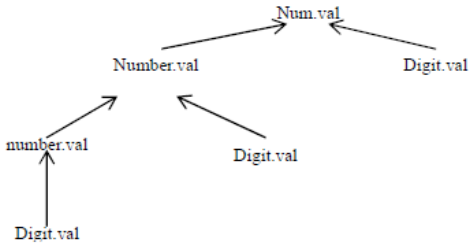
number1.val = number2.val * 10 +digit.val

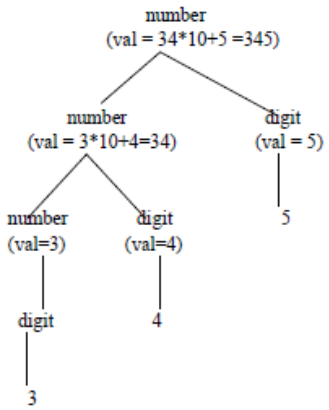


number.val = digit.val



b) Draw the parse tree and dependency graph for the string 785

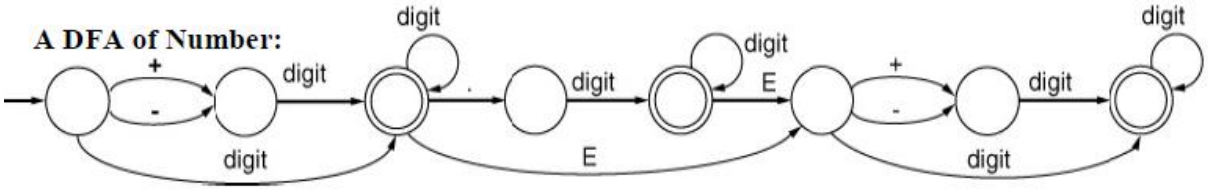




2.2 Draw DFA's that accept the following:

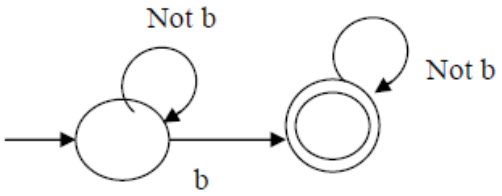
a) All strings that represent numeric constants in scientific notation

***nat* = [0-9]+ *signedNat* = (+|-)?*nat* *number* = *signedNat*(“ . ”*nat*)? (E *signedNat*)**

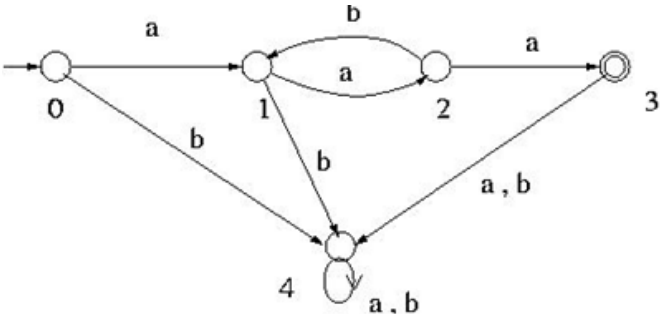


b) All strings that contain exactly one b over the alphabet {a, b, c}

– **(a|c)*b(a|c)***



c) The regular expression a(ab)*aa, given the alphabet { a, b }.



accepts a(ab)*aa

2.3 What is the role of the following sections in a procedure activation record?

a) Space for bookkeeping information

b) Space for local temporaries

Question 3 (20 Marks)

3.1 Given the grammar rule for an if-statement:

If-stmt* → if (*exp*) *statement

| if (*exp*) *statement* else *statement*

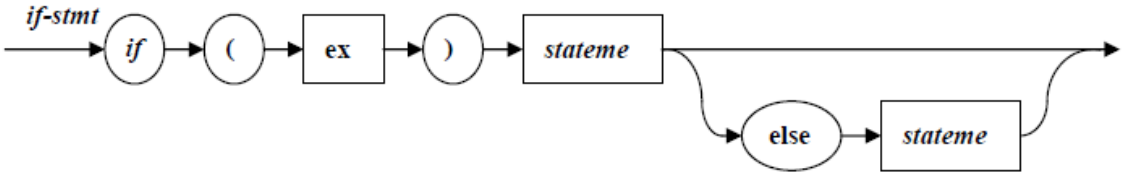
a) Translate this rule into EBNF

The EBNF of the if-statement

If-stmt → if (*exp*) *statement* [else *statement*]

Square brackets of the EBNF are translated into a test in the code for if-stmt.

b) Draw the syntax diagram of EBNF of part (a).



c) Write pseudo-code to parse this grammar by recursive descent

```
procedure if-stmt;
    begin
      match( if );
      match( ( );
      exp;
```

```
        match( ) );
        statement;
        if token = else then
          match (else);
          statement;
        end if;
      end if-stmt;

    procedure match( expectedToken);
    begin
      if token = expectedToken then
        getToken;
      else
        error;
      end if;
    end match
```

c- Write pseudo-code to parse this grammar by recursive descent

```
procedure if-stmt;
    begin
      match( if );
      match( ( );
      exp;
      match( ) );
      statement;
      if token = else then
        match (else);
        statement;
      end if;
    end if-stmt;
```

```
procedure match( expectedToken);
begin
  if token = expectedToken then
    getToken;
  else
    error;
  end if;
end match
```

3.2 Mention a situation where the First set computation is required.

(2) It is difficult to decide when to use the choice A →α and the choice A →β; if both α and β begin with non-terminals. Such a decision problem requires the computation of the First Sets.

3.3 Given the following arithmetic Expression:

a * b + a * b * c

a) Write down the corresponding three-address code.

b) Show the triple representation for this code.

Question 4 (20 Marks)

4.1 Consider the following grammar

Stmt-sequence → stmt; stmt-sequence | stmt

Stmt→ s

a) Left factor this grammar

Stmt-sequence→stmt stmt-seq'

Stmt-seq'→; stmt-sequence | ε

b) Construct First and Follow sets for the non terminal of the resulting grammar. You are asked to show the computation process.

The first and follow set

First Sets	Follow Sets
First(stmt-sequence)={s} **First(stmt)={s}** **First(stmt-seq')={;, ε}**	**Follow(stmt-sequence)={$}** **Follow(stmt)={;}** **Follow(stmt-seq')={$}**

c) Construct the LL(1) parsing table for the resulting grammar

the LL(1) parsing table

M[N,T]	S	;	$
Stmt-sequence	**Stmt-sequence →stmt stmt-seq'**		
Stmt	**stmt→s**		
Stmt-seq'		**Stmt-seq' →; stmt-sequence**	**Stmt-seq' →ε**

d) Show the action of corresponding LL(1) parser given the input string s; s ; s

Step	Parsing Stack	Input	Action
1	*stmt-sequence* **$**	**s;s;s $**	**Stmt-sequence →stmt stmt-seq'**
2	**stmt stmt-seq' $**	**s;s;s $**	**stmt→s**
3	**s stmt-seq' $**	**s;s;s $**	**Match**
4	**stmt-seq' $**	**;s;s $**	**Stmt-seq' →; stmt-sequence**
5	**;stmt-sequence $**	**;s;s $**	**Match**
6	**stmt-sequence $**	**s;s $**	**Stmt-sequence →stmt stmt-seq'**
7	**stmt stmt-seq' $**	**s;s $**	**stmt→s**
8	**s stmt-seq' $**	**s;s $**	**Match**
9	**stmt-seq' $**	**;s $**	**Stmt-seq' →; stmt-sequence**
10	**;stmt-sequence $**	**;s $**	**Match**
11	**stmt-sequence $**	**s $**	**Stmt-sequence →stmt stmt-seq'**
12	**stmt stmt-seq' $**	**s $**	**stmt→s**
13	**s stmt-seq' $**	**s $**	**Match**
14	**stmt-seq' $**	**$**	**Stmt-seq' →ε**
15	**$**	**$**	**accept**

4.2 What is meant by regular expression?

a language or technique for writing rules that describe tokens

4.3 Write regular expressions for the following:

a) All strings of digits such that all 2's occur before all 9's

(noNine)* (noTwo)*

b) Strings of a's and b's that contain an even number of a's and an even number of b's

no regular expression

this case cannot be represented by a regular expression regular expression is not able to count

c) All strings that contain at most one b over the alphabet {a, b, c}

(a|c)*|(a|c)*b(a|c)* **(a|c)*(b|ε)(a|c)***

(End of Questions-Good Luck)