NAME: ______________________________

STUDENT ID: ______________________________

SIGNATURE: ______________________________

The University of New South Wales

Final Examination

November/December 2000

COMP3131/COMP9102

Parsing and Translation
and
Compiling Techniques and Programming Languages

Time allowed: **2 hours**

Total number of questions: **5**

Answer **all** questions

The questions are **not** of equal value

Each question must be answered in a separate book

No examination materials

Answers must be written in ink.

Question 1. Context-Free Grammars (CFGs) *[20 marks]*

Consider the following CFG:

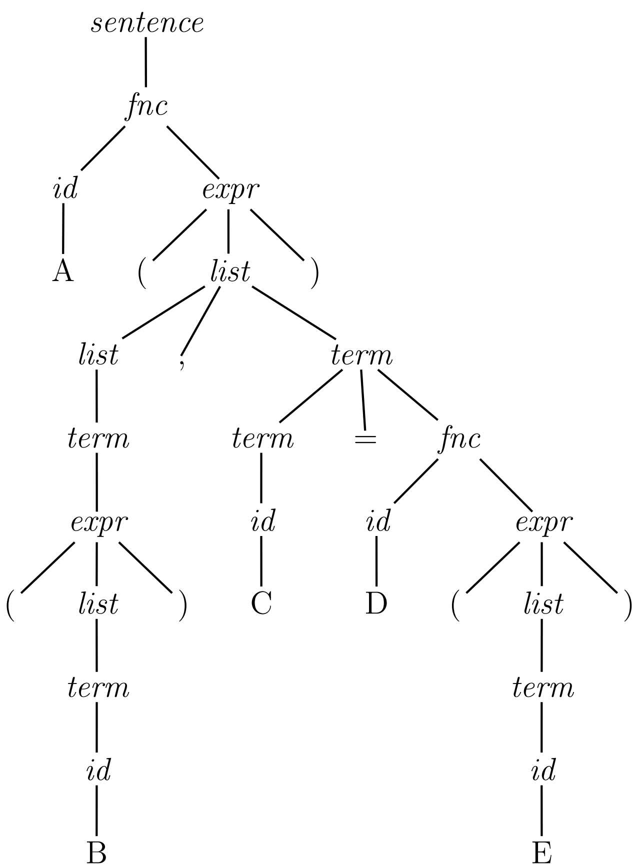
$$
\begin{array}{lcl}
\textit{sentence} & \rightarrow & \textit{fnc} \\
\textit{fnc} & \rightarrow & \textit{id expr} \\
\textit{id} & \rightarrow & \textsf{\textbf{A}} \mid \textsf{\textbf{B}} \mid \textsf{\textbf{C}} \mid \textsf{\textbf{D}} \mid \textsf{\textbf{E}} \\
\textit{expr} & \rightarrow & \text{“(”}\ \textit{list}\ \text{“)”} \\
\textit{list} & \rightarrow & \textit{list}\ \text{“,”}\ \textit{term} \mid \textit{term} \\
\textit{term} & \rightarrow & \textit{term}\ \text{“=”}\ \textit{fnc} \mid \textit{expr} \mid \textit{id}
\end{array}
$$

(a) Give a parse tree for **A** ((**B**), **C** = **D** (**E**)).

(b) Give a leftmost derivation for **A** (**B**, **C**).

(c) Give a revised grammar that is free of the left recursion.

(a)



(b)

$$
\begin{aligned}
\textit{sentence} &\Rightarrow_{\text{lm}} \textit{fnc} \\
&\Rightarrow_{\text{lm}} \textit{id expr} \\
&\Rightarrow_{\text{lm}} \textit{id expr} \\
&\Rightarrow_{\text{lm}} \textit{id} \ (\ \textit{list}\ ) \\
&\Rightarrow_{\text{lm}} \textit{id} \ (\ \textit{list},\ \textit{term}\ ) \\
&\Rightarrow_{\text{lm}} \textit{id} \ (\ \textit{term},\ \textit{term}\ ) \\
&\Rightarrow_{\text{lm}} \textit{id} \ (\ \textit{id},\ \textit{term}\ ) \\
&\Rightarrow_{\text{lm}} \textit{id} \ (\ \textit{id},\ \textit{id}\ )
\end{aligned}
$$

(c)

$$
\begin{array}{lcl}
\textit{sentence} & \rightarrow & \textit{fnc} \\
\textit{fnc} & \rightarrow & \textit{id expr} \\
\textit{id} & \rightarrow & \textbf{A} \mid \textbf{B} \mid \textbf{C} \mid \textbf{D} \mid \textbf{E} \\
\textit{expr} & \rightarrow & \text{“(”}\ \textit{list}\ \text{“)”} \\
\textit{list} & \rightarrow & \textit{term}\ (\ \text{“,”}\ \textit{term}\ )^* \\
\textit{term} & \rightarrow & (\textit{expr} \mid \textit{id}\ )\ (\ \text{“=”}\ \textit{fnc}\ )^*
\end{array}
$$

Question 2. Regular Expressions and Finite Automata *[20 marks]*

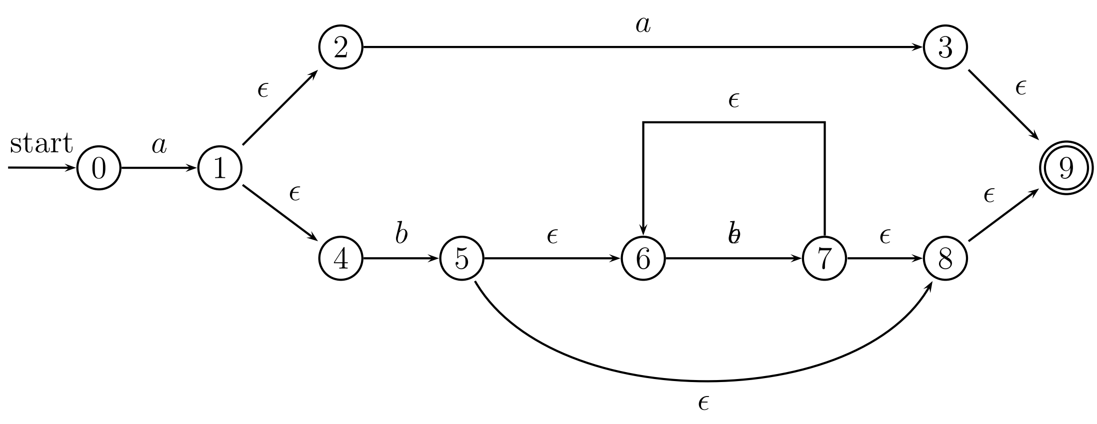
Let the regular expression $a(a|b^+)$ be given.

(a) Use Thompson's construction to convert the regular expression into an Nondeterministic Finite Automaton (NFA).

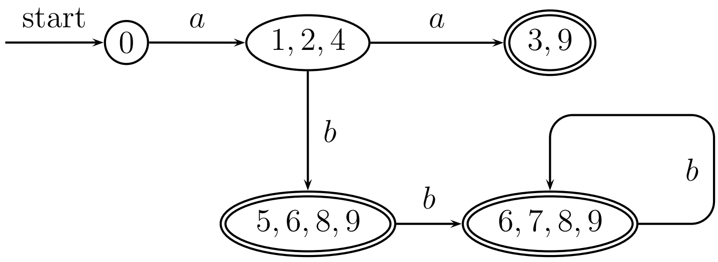
(b) Use the subset construction to convert the NFA from (a) into a Deterministic Finite Automaton (DFA).

(c) Is it true that a DFA constructed from an NFA with n states using the subset construction has at most 2^n states? Why or why not?

(a)



(b)



(c) True. The set of DFA states is a subset of the set of all NFA states.

Question 3. Top-Down Parsing *[30 marks]*

Consider the following grammar:

$$
\begin{array}{lcl}
\mathit{lexpr} & \rightarrow & \mathit{atom} \mid \mathit{list} \\
\mathit{atom} & \rightarrow & \textbf{id} \mid \textbf{num} \\
\mathit{list} & \rightarrow & \text{“(”}\ \mathit{lexpr\text{-}seq}\ \text{“)”} \\
\mathit{lexpr\text{-}seq} & \rightarrow & \mathit{lexpr}\ \mathit{lexpr\text{-}seq\text{-}tail} \\
\mathit{lexpr\text{-}seq\text{-}tail} & \rightarrow & \mathit{lexpr}\ \mathit{lexpr\text{-}seq\text{-}tail} \mid \epsilon
\end{array}
$$

(a) Construct FIRST sets for all nonterminals and all production right sides.

(b) Construct FOLLOW sets for all nonterminals.

(c) Construct SELECT sets for all productions.

(d) Construct the LL(1) parsing table for the grammar.

(e) Is this grammar LL(1). Why or why not?

(a)

$$
\begin{array}{lcl}
\mathsf{FIRST}(\mathit{lexpr}) & = & \{\textbf{id}, \textbf{num}, (\} \\
\mathsf{FIRST}(\mathit{atom}) & = & \{\textbf{id}, \textbf{num}\} \\
\mathsf{FIRST}(\mathit{list}) & = & \{(\} \\
\mathsf{FIRST}(\mathit{lexpr\text{-}seq}) & = & \{\textbf{id}, \textbf{num}, (\} \\
\mathsf{FIRST}(\mathit{lexpr\text{-}seq\text{-}tail}) & = & \{\epsilon, \textbf{id}, \textbf{num}, (\} \\
\mathsf{FIRST}(\textbf{id}) & = & \{\textbf{id}\} \\
\mathsf{FIRST}(\textbf{num}) & = & \{\textbf{num}\} \\
\mathsf{FIRST}((\textbf{list})) & = & \{(\} \\
\mathsf{FIRST}(\epsilon) & = & \{\epsilon\}
\end{array}
$$

(b)

$$
\begin{array}{lcl}
\mathsf{FOLLOW}(\mathit{lexpr}) & = & \mathsf{FOLLOW}(\mathit{atom}) = \mathsf{FOLLOW}(\mathit{list}) = \{\textbf{id}, \textbf{num}, (,), \$\} \\
\mathsf{FOLLOW}(\mathit{lexpr\text{-}seq}) & = & \{)\} \\
\mathsf{FOLLOW}(\mathit{lexpr\text{-}seq\text{-}tail}) & = & \{)\}
\end{array}
$$

(c)

$$
\begin{aligned}
\mathsf{SELECT}(lexpr \rightarrow atom) &= \{\textbf{id}, \textbf{num}\} \\
\mathsf{SELECT}(lexpr \rightarrow list) &= \{(\} \\
\mathsf{SELECT}(atom \rightarrow \textbf{id}) &= \{\textbf{id}\} \\
\mathsf{SELECT}(atom \rightarrow \textbf{num}) &= \{\textbf{num}\} \\
\mathsf{SELECT}(list \rightarrow (lexpr\text{-}seq)) &= \{(\} \\
\mathsf{SELECT}(lexpr\text{-}seq \rightarrow lexpr lexpr\text{-}seq\text{-}tail) &= \{\textbf{id}, \textbf{num}, (\} \\
\mathsf{SELECT}(lexpr\text{-}seq\text{-}tail \rightarrow lexpr lexpr\text{-}seq\text{-}tail) &= \{\textbf{id}, \textbf{num}, (\} \\
\mathsf{SELECT}(lexpr\text{-}seq\text{-}tail \rightarrow \epsilon) &= \{)\}
\end{aligned}
$$

(d)

	id	**num**	(	)	$
lexpr	*atom*	*atom*	*list*		
atom	**id**	**num**			
list			(*lexpr-seq*)		
lexpr-seq	*lexpr lexpr-seq*	*lexpr lexpr-seq*	*lexpr lexpr-seq*		
lexpr-seq-tail	*lexpr lexpr-seq-tail*	*lexpr lexpr-seq-tail*	*lexpr lexpr-seq-tail*	ϵ	

(e) LL(1) because no table entry contains more than one production.

Question 4. Attribute Grammars *[20 marks]*

Consider the following grammar for unsigned decimal numbers:

$$
\begin{aligned}
number &\rightarrow digit\ number \mid digit \\
digit &\rightarrow \textbf{0} \mid \textbf{1} \mid \textbf{2} \mid \textbf{3} \mid \textbf{4} \mid \textbf{5} \mid \textbf{6} \mid \textbf{7} \mid \textbf{8} \mid \textbf{9}
\end{aligned}
$$

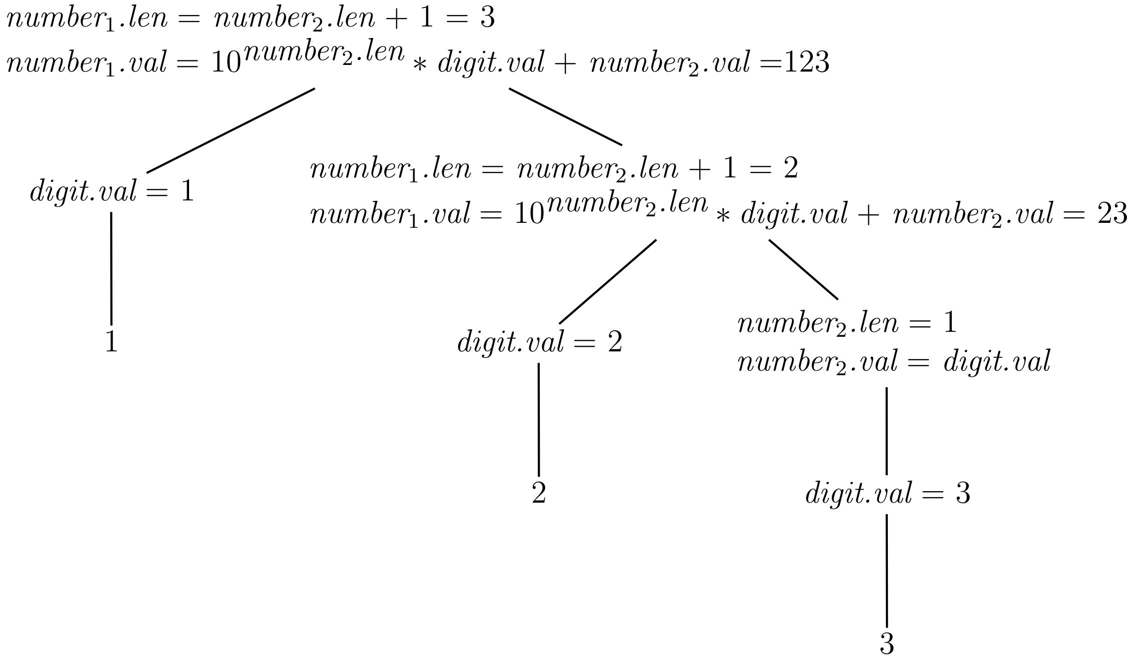
(a) Give an attribute grammar for the integer value of a number.

(b) Draw the decorated parse tree for **123**.

(a)

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

(b)


$number_1.len = number_2.len + 1 = 3$
$number_1.val = 10^{number_2.len} * digit.val + number_2.val = 123$
$digit.val = 1$
1
$number_1.len = number_2.len + 1 = 2$
$number_1.val = 10^{number_2.len} * digit.val + number_2.val = 23$
$digit.val = 2$
2
$number_2.len = 1$
$number_2.val = digit.val$
$digit.val = 3$
3

Question 5. Code Generation *[10 marks]*

Consider the following Jasmin code:

```
.source Test.java
.class  Test
.super java/lang/Object

.method static add(II)I
.limit stack 2
.limit locals 2
.var 0 is a I from Label0 to Label1
.var 1 is b I from Label0 to Label1

Label0:
        iload_0
        iload_1
        iadd
Label1:
        ireturn
.end method

.method public static main([Ljava/lang/String;)V
.limit stack 2
.limit locals 1
.var 0 is argv [Ljava/lang/String; from Label0 to Label1

Label0:
        iconst_1
        iconst_2
        invokestatic Test/add(II)I
        pop
Label1:
        return
.end method
```

(a) Give a Java program that can be compiled to the above Jasmin code.

(b) Show the contents of the operand stack for the main method just before and after the instruction `invokestatic Test/add(II)I` is executed.

(a)

```
class Test {
static int add(int a, int b) {
   return a + b;
}

public static void main(String argv[]) {
  add(1, 2);
  }
}
```

(b)

```
BEFORE:
+------------
|  1      2
+-----------
AFTER:
+------------
|  3
+-----------
```