NAME: ______________________________

STUDENT ID: ______________________________

SIGNATURE: ______________________________

The University of New South Wales

Final Examination

June 2021

COMP3131/COMP9102

Programming Languages and Compilers

Time allowed: **2 hours**

Total number of questions: **5**

Answer **all** questions

The questions are **not** of equal value

Marks for this paper total **100**

This paper may **not** be retained by the candidate

No examination materials

Answers must be written in ink.

Question 1. Regular Expressions and Finite Automata *[15 marks]*

Consider the following regular expression:

$$(a|b)^*a(a|\epsilon)$$

(a) Use **Thompson's construction** to convert this regular expression into an NFA.

[7 marks]

(b) Use the **subset construction** to convert the NFA of (a) into a DFA.

[7 marks]

(c) Convert the DFA of (b) into a minimal-state DFA.

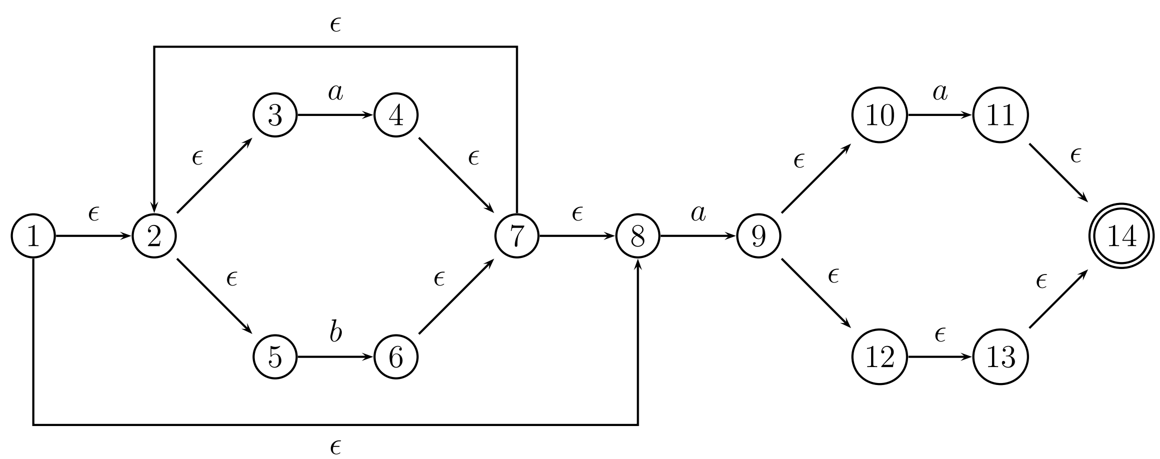
[4 marks]

You are required to apply exactly Thompson's construction algorithm in (a) and the subset construction algorithm in (b) to solve those two problems.

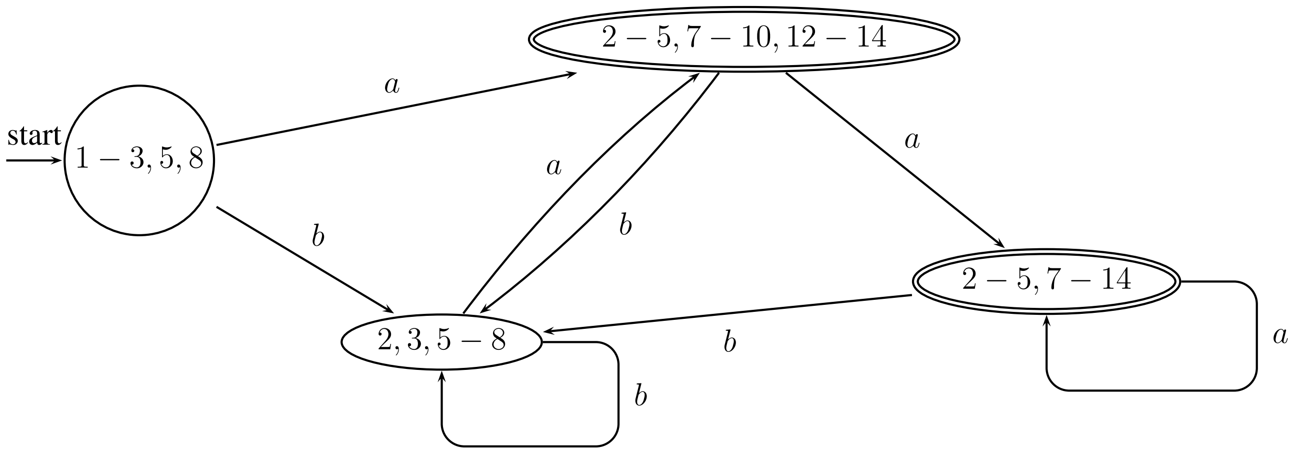
Accepted files to submit via give or Webcms3: *.jpeg, *.gif, *.tiff, *.png or *.pdf.

********* ANSWER *********

(a)



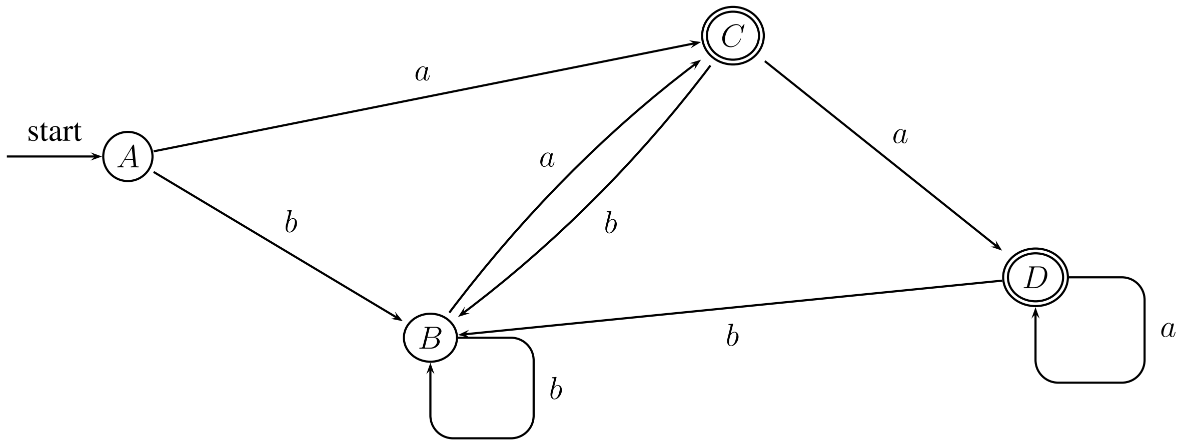
(b)



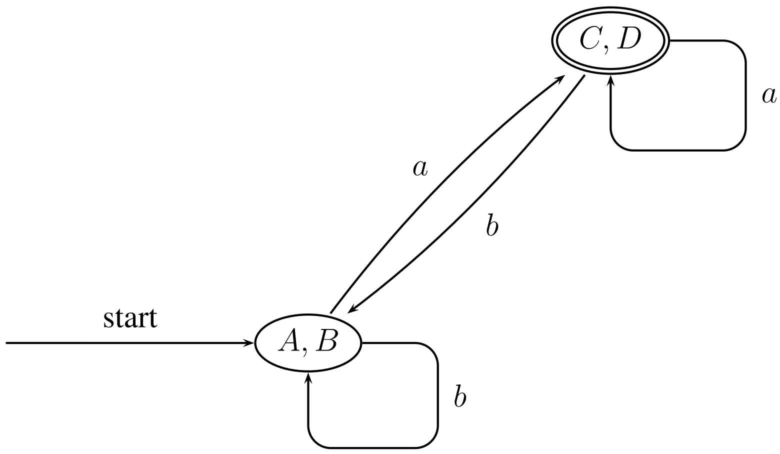
start
1 − 3, 5, 8
2 − 5, 7 − 10, 12 − 14
2, 3, 5 − 8
2 − 5, 7 − 14
a
b
a
b
a
b
b
a

(c)

Renaming the states of the DFA yields:



The minimal-state DFA is:



Question 2. Context-Free Grammars *[15 marks]*

Assume that arithmetic expressions are built up from the following terminals:

- Binary infix operators: @, #, +, $-$
- Unary prefix operator: $\sim$
- Variables: X, Y, Z
- Brackets: [,]

Operator $\sim$ has the highest precedence, followed by @ and #, which have equal precedence. Operators + and $-$ have the lowest precedence. Operators @, # and + are left associative but $-$ is right associative. Brackets are used to group expressions in the usual manner.

Write a context-free grammar for this language.

You are not allowed to use the regular operators, *, $^+$, and ?, in your grammar.

Accpeted file to submit via give or Webcms3: *.txt.

********* ANSWER *********

```
<exp> -->  <exp> +  <term>
       | <term> - <expr>
       | <term>

<term> --> <term> @ <factor>
        | <term> # <factor>
        | <factor>

<factor> --> ~ <factor>
          | [ <exp> ]
          | <var>

<var> --> X | Y | Z
```

Question 3. Recursive Descent Parsing *[20 marks]*

Consider the following context-free grammar:

$$
\begin{array}{llcl}
1. & S & \rightarrow & BA \\
2. & A & \rightarrow & aBA \\
3. & & | & \epsilon \\
4. & B & \rightarrow & DC \\
5. & C & \rightarrow & cDC \\
6. & & | & \epsilon \\
7. & D & \rightarrow & xSf \\
8. & & | & gCg
\end{array}
$$

where the set of nonterminals is $\{S, A, B, C, D\}$ and the set of terminals is $\{a, c, x, f, g\}$.

(a) Compute the FIRST sets for all nonterminal symbols.

[4 marks]

(b) Compute the FOLLOW sets for all nonterminal symbols.

[6 marks]

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

[4 marks]

(d) Is the grammar LL(1)? Justify your answer in a few sentences.

[2 marks]

(e) The string gx is NOT syntactically legal (since it is NOT in the language defined by the grammar). Explain concisely how this can be detected by an LL(1) table-driven parser for the language.

[4 marks]

Note: In your answer, you can write E or e as an abbreviation for ϵ.

Accepted file to submit via give or Webcms3: *.txt.

********* ANSWER *********

********* ANSWER *********

(a)

FIRST(S)	$\{x, g\}$
FIRST(A)	$\{a, \epsilon\}$
FIRST(B)	$\{x, g\}$
FIRST(C)	$\{c, \epsilon\}$
FIRST(D)	$\{x, g\}$

(b)

FOLLOW(S)	$\{f, \$\}$
FOLLOW(A)	$\{f, \$\}$
FOLLOW(B)	$\{a, f, \$\}$
FOLLOW(C)	$\{a, f, g, \$\}$
FOLLOW(D)	$\{a, c, f, g, \$\}$

(c)

	a	c	d	x	f	g	$
S				P1		P1	
A	P2				P3		P3
B				P4		P4	
C	P6	P5			P6	P6	P6
D				P7		P8	

(d) Yes. Each entry contains at most one production.

(e)

```
Stack     Input      Production           Left Derivation
$         gx
$         gx         S->BA                S=>BA
$AB       gx         B->DC                S=>DCA
$ACD      gx         D->gCg               S=>gCgCA
$ACgCg    advance
#ACgC     x          blank ===> error
```

Question 4. Attribute Grammars *[15 marks]*

Consider the following context-free grammar that generates regular expressions:

$$
\begin{array}{llll}
1. & R & \rightarrow & a \\
2. & & | & b \\
3. & & | & \epsilon \\
4. & & | & R_1 R_2 \\
5. & & | & R_1 \mid R_2 \\
6. & & | & (R_1)^* \\
7. & & | & (R_1)
\end{array}
$$

(a) Define an attribute grammar that records the maximum number of **nested** Kleene star operators of a regular expression R in its attribute $R.depth$. For example, ab has depth 0, a^* has depth 1 and $a^*|(b^*|a)^*$ has depth 2.

[14 marks]

(b) Is $R.depth$ inherited or synthesized? Explain your answer.

[3 marks]

Note: In your answer, you can write R_1, R_2 and * as $R1$, $R2$ and $*$, respectively.

Accepted file via **give**: *.txt.

This is straightforward except that adding an inherited attribute can be tricky.

********* ANSWER *********

(a)

1.	R	$\rightarrow$	a	$\{R.depth = 0;\}$
2.		$\mid$	b	$\{R.depth = 0;\}$
3.		$\mid$	ϵ	$\{R.depth = 0;\}$
4.		$\mid$	R_1R_2	$\{$ **if** $R_1.depth > R_2.depth$ $R.depth = R_1.depth$ **else** $R.depth = R_2.depth$ $\}$
5.		$\mid$	$R_1 \mid R_2$	$\{$ **if** $R_1.depth > R_2.depth$ $R.depth = R_1.depth$ **else** $R.depth = R_2.depth$ $\}$
6.		$\mid$	$(R_1)^*$	$R.depth = R_1.depth + 1$
7.		$\mid$	(R_1)	$R.depth = R_1.depth$

(b) Synthesised as it is propagated to a node from its children

Question 5. Code Generation *[30 marks]*

Consider the following attribute grammar for generating "short-circuit" code, where

- The semantic rules associated with a production are evaluated sequentially in a top-down manner, and
- "#" stands for the string concatenation operator.

$S \rightarrow$ **while** B **do** S_1
$S.begin := getNewLabel();$
$B.true := getNewLabel();$
$B.false := S.next;$
$S_1.next := S.begin;$
$S.code := emit(S.begin\ ':') \ \#\ B.code \ \#\ emit(B.true\ ':') \ \#\ S_1.code \ \#\ emit('goto'\ S.begin)$
$S \rightarrow \mathbf{ID} = E$
$S.code := E.code \ \#\ emit(\mathbf{ID}.place\ ':='\ E.place)$
$E \rightarrow E_1 + E_2$
$E.place := getNewTemp();$
$E.code := E_1.code \ \#\ E_2.code \ \#\ emit(E.place\ ':='\ E_1.place\ '+'\ E_2.place)$
$E \rightarrow \mathbf{ID}$
$E.place := \mathbf{ID}.place;$ // $\mathbf{ID}.place$ is the lexeme of the **ID**
$E.code := ''$ // no code generated
$B \rightarrow B_1$ && B_2
$B_1.true := getNewLabel();$
$B_1.false := B.false;$
$B_2.true := B.true;$
$B_2.false := B.false;$
$B.code := B_1.code \ \#\ emit(B_1.true\ ':') \ \#\ B_2.code$
$B \rightarrow !\ B_1$
$B_1.true := B.false;$
$B_1.false := B.true;$
$B.code := B_1.code$
$B \rightarrow \mathbf{ID}_1 > \mathbf{ID}_2$
$B.code := emit('\mathbf{if}'\ \mathbf{ID}_1.place > \mathbf{ID}_2.place\ 'goto' B.true) \ \#\ emit('goto' B.false)$

Note that this grammar is ambiguous but that does not affect the following questions.

Consider the following **while** loop:

while (! (a > b && x > y))
r = p + q;

(a) Draw the AST (Abstract Syntax Tree) for the **while** loop. *[5 marks]*

Continued onto next page

Question 5 continued from Page 6

(b) Suppose that $S.next$ = L666, $getNewLabel()$ will return labels L1, L2, ... when called, and $getNewTemp()$ will return temporary variables t1, t2, ... when called. Give the $B.true$ and $B.false$ attributes for all the B nodes in the AST of (a).

[7 marks]

(c) Give the $S.code$ attribute for the root node S in the AST of (a). In other words, give the code generated for the **while** loop according to this attribute grammar.

[7 marks]

(d) The production $B \rightarrow B_1$ && B_2 given in the grammar serves to define conditional AND expressions. Suppose we replace this production with $B \rightarrow B_1 \parallel B_2$ so that conditional OR expressions are considered instead. Give the semantic rules for the new production to generate short-circuit code for conditional OR expressions.

[5 marks]

(e) Give the semantic rules for the new production that defines `do-while` statements:

$$S \;\rightarrow\; \textbf{do}\ S_1\ \textbf{while}\ B$$

In a `do-while` statement, S_1 will be executed at least once.

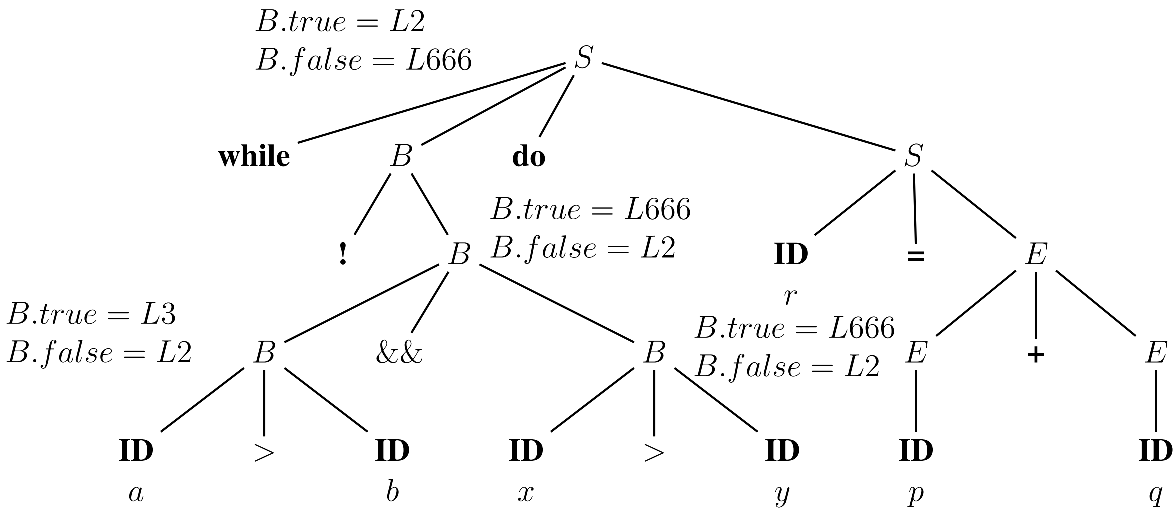
[6 marks]

Note: In your answer, you can write S_1, B_1 and B_2 as $S1$, $B1$ and $B2$, respectively.

Accepted file to submit via give or Webcms3: *.txt.

********* ANSWER *********

(a) and (b)



(c)

```
L1:
     if a > b goto L3
       goto  L2
L3:
     if x > y goto L666
       goto  L2
L2:
     r = p + q
     goto L1
```

(d)

$$
\boxed{\begin{array}{l}
B \rightarrow B_1 \parallel B_2 \\
\qquad\qquad B_1.true := B.true; \\
\qquad\qquad B_1.false := getNewLabel(); \\
\qquad\qquad B_2.true := B.true; \\
\qquad\qquad B_2.false := B.false; \\
\qquad\qquad B.code := B_1.code \bowtie emit(B_1.false\,'\,:') \bowtie B_2.code
\end{array}}
$$

(e)

$S \to$ **do** S_1 **while** B

$S.begin := getNewLabel();$
$B.true := S.begin();$
$B.false := S.next;$
$S_1.next := S.begin;$
$emit(S.begin\ ':') \bowtie S_1.code \bowtie B.code$