The University of New South Wales

Final Examination

Term 1, 2021

COMP3131/COMP9102

Programming Languages and Compilers

1. **Time Allowed:** 3 hours (including 10 min **reading time**)
2. **Total Number of Pages:** 8 (including cover page and Appendix A)
3. **Total Number of Questions:** 5
4. **Total Marks Available:** 100
5. Marks available for each question are shown in the examination paper.
6. The questions are not of equal value.
7. Answer all questions.
8. Submit your answers via **give** or **Webcms3**.
9. The answers to Q1 can be submitted as jpeg/gif/tiff/png/pdf files but the answers to Q2 – Q5 must be submitted as ASCII text files:
   - Q1: *.*suffix*, where *suffix* is jpeg, gif, tiff, png, or pdf
   - Q2: *.txt
   - Q3: *.txt
   - Q4: *.txt
   - Q5: *.txt
10. No examination materials allowed.

Question 1. Regular Expressions to Finite Automta *[18 marks]*

Consider the following regular expression:

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

(a) Use **T**[illegible]**truction** to convert this regular expression into an NF[illegible]

[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.

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

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

- Binary … @, $\#$, $+$, $-$
- Unary …
- Variabl…
- Bracke…

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.

Question 3. Recursive-Descent LL(1) Parsing *[20 marks]*

Consider the following context-free grammar:

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

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.

Question 4. Attribute Grammars *[17 marks]*

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

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

(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.

Question 5. Code Generation *[30 marks]*

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

- The se[illegible]ciated with a production are evaluated sequentially i[illegible]ner, and
- "#" sta[illegible] concatenation operator.

$S \rightarrow$ **while** B **do** S_1
  $S.begin := getNewLabel();$
  $B.true := getNewLabel();$
  $B.false := S.next;$
  $S_1.next := S.begin;$
  $S.code := emit(S.begin\,':') \# B.code \# emit(B.true\,':') \# S_1.code \# emit('goto'\ S.begin)$

$S \rightarrow \mathbf{ID} = E$
  $S.code := E.code \# emit(\mathbf{ID}.place\,':='\,E.place)$

$E \rightarrow E_1 + E_2$
  $E.place := getNewTemp();$
  $E.code := E_1.code \# E_2.code \# emit(E.place\,':='\,E_1.place\,'+'\,E_2.place)$

$E \rightarrow \mathbf{ID}$
  $E.place := \mathbf{ID}.place;$ // **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, . . [illegible]nd $getNewTemp()$ will return temporary variables t[illegible]alled.
Give t[illegible]$.false$ 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 $B \rightarrow B_1 \ || \ B_2$ 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.

Appendix A. Jasmin assembly (i.e., JVM) instructions for Question 5.


```
public final class JVM {
    // Arithmetic instructions
    IADD = "iadd",
    ISUB = '[illegible]
    IMUL = '[illegible]
    IDIV = '[illegible]

    // Load[illegible] local variable into operand stack)
    ILOAD = [illegible]
    ILOAD_0[illegible]
    ILOAD_1[illegible]
    ILOAD_2 = "iload_2",
    ILOAD_3 = "iload_3",

    // Stores (for storing the top operand in operand stack into a local variable)
    ISTORE = "istore",
    ISTORE_0 = "istore_0",
    ISTORE_1 = "istore_1",
    ISTORE_2 = "istore_2",
    ISTORE_3 = "istore_3",

    // Loads (for loading a constant into operand stack)
    ICONST_M1 = "iconst_m1",
    ICONST_0 = "iconst_0",
    ICONST_1 = "iconst_1",
    ICONST_2 = "iconst_2",
    ICONST_3 = "iconst_3",
    ICONST_4 = "iconst_4",
    ICONST_5 = "iconst_5",

    // Control transfer instructions
    GOTO = "goto",
    IFEQ = "ifeq",
    IFNE = "ifne",
    IFLE = "ifle",
    IFLT = "iflt",
    IFGE = "ifge",
    IFGT = "ifgt",
    IF_ICMPEQ = "if_icmpeq",
    IF_ICMPNE = "if_icmpne",
    IF_ICMPLE = "if_icmple",
    IF_ICMPLT = "if_icmplt",
    IF_ICMPGE = "if_icmpge",
    IF_ICMPGT = "if_icmpgt",

    // Operand stack management instructions
    DUP_X2 = "dup_x2",
    DUP = "dup",
    POP = "pop",
    ...
}
```

==================== **END OF PAPER** ====================