Formal Languages and Compilers – 2013, session 1

[8 marks] Exercise 1

Let $\mathcal{N}$ be the NFA over the alphabet $\{a, b\}$ with initial state A, final state D, and transition table as drawn below. Provide the minimum DFA equivalent to $\mathcal{N}$.

	a	b
A	$\{B, C\}$	$\{D\}$
B	$\{A, C\}$	$\{D\}$
C	$\{A, B\}$	$\{D\}$
D	$\emptyset$	$\emptyset$

[12 marks] Exercise 2

Let $\mathcal{G}$ be the following grammar:

$$B \quad \rightarrow \quad \mathsf{not}\, B \mid B \Rightarrow B \mid (B) \mid \mathsf{id}$$

where B is the single non-terminal symbol.

1. Show that $\mathcal{G}$ is ambiguous.
2. Provide the SLR parsing table for $\mathcal{G}$, list all the conflicts found, and state how each of them can be resolved to get the usual associativity and precedence of the involved operators:
   - "not" has higher precedence than "$\Rightarrow$";
   - "$\Rightarrow$" is right associative, i.e. $\mathsf{id}_1 \Rightarrow \mathsf{id}_2 \Rightarrow \mathsf{id}_3$ stands for $\mathsf{id}_1 \Rightarrow (\mathsf{id}_2 \Rightarrow \mathsf{id}_3)$.
3. Using the modified SLR table, show the parsing steps on input

   $$\mathsf{id} \Rightarrow \mathsf{not}\,\mathsf{id} \Rightarrow \mathsf{id}$$

   and draw the resulting parse tree.

[5 marks] Exercise 3

Extend the syntax-directed definition of Fig. 6.36 to deal with the control-flow construct generated by

$$S \quad \rightarrow \quad \textbf{repeat}\ S_1\ \textbf{until}\ B$$

whose intended meaning is as follows. First S_1 is executed. If B is false in the resulting state, then the execution of the whole command is over, otherwise **repeat** S_1 **until** B is executed again.

[3 marks] Exercise 4

Let $\mathcal{L}$ be defined as follows:

$$\mathcal{L} = \{a^n b^m a^n b^m \mid n, m \geq 0\}$$

Say whether or not $\mathcal{L}$ is a context-free language. Justify your answer.

[2 marks] Exercise 5

Let $\mathcal{P}$ be the following program:

```
A: begin
    proc B;
        begin
            C: begin...end
            D: begin...end
        end
    E: begin
        F: begin...end
        proc T;
            begin
                G: begin...end
                H: begin...end
            end
    end
end
```

- Draw the scoping tree for $\mathcal{P}$.
- Show the computation and the resulting stack chain pointer at the end of the following sequence of calls:

$$A \Downarrow E \Downarrow T \Downarrow H$$

label to which control flows if B is true, and $B.false$, the label to which control flows if B is false. With a statement S, we associate an inherited attribute $S.next$ denoting a label for the instruction immediately after the code for S. In some cases, the instruction immediately following $S.code$ is a jump to some label L. A jump to a jump to L from within $S.code$ is avoided using $S.next$.

The syntax-directed definition in Fig. 6.36-6.37 produces three-address code for boolean expressions in the context of if-, if-else-, and while-statements.

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**.$code$
$S \rightarrow$ **if** $(B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** $(B) S_1$ **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\text{'goto'}\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** $(B) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen(\text{'goto'}\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

Figure 6.36: Syntax-directed definition for flow-of-control statements.

We assume that $newlabel()$ creates a new label each time it is called, and that $label(L)$ attaches label L to the next three-address instruction to be generated.[8]

[8] If implemented literally, the semantic rules will generate lots of labels and may attach more than one label to a three-address instruction. The backpatching approach of Section 6.7

Formal Languages and Compilers – 2013, session 2

[12 marks] Exercise 1

Let $\mathcal{G}$ be defined as follows:

$$\begin{array}{rcl} S & \rightarrow & aS \mid aSb \mid T \\ T & \rightarrow & aTa \mid a \end{array}$$

1. Show that $\mathcal{G}$ is not LALR(1).
2. Provide a LALR(1) grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})$.
3. Using the LALR(1) parsing table for $\mathcal{G}'$, show the parsing steps on input $aaabb$ and draw the resulting parse tree.

[8 marks] Exercise 2

Provide the minimum DFA to recognize the language generated by the following regular expression:

$$(ba)^*(b \mid a \mid \epsilon)(ba)^*(b^* \mid \epsilon).$$

[7 marks] Exercise 3

Let $\mathcal{G}$ be the following grammar for binary numbers:

$$\begin{array}{rcl} S & \rightarrow & L \\ L & \rightarrow & LB \mid B \\ B & \rightarrow & 0 \mid 1 \end{array}$$

1. Add attribution rules to $\mathcal{G}$ so that the attribute $S.val$ of the start symbol contains the decimal value of the generated binary number.
2. Show the evaluation of $S.val$ for the derivation of 101.

[3 marks] Exercise 4

Let $\mathcal{G}$ be the following grammar:

$$\begin{array}{rcl} S & \rightarrow & Aa \mid Bb \\ A & \rightarrow & aAb \mid ab \\ B & \rightarrow & aBbb \mid abb \end{array}$$

Explain why $\mathcal{G}$ is neither SLR nor LALR(1).

Formal Languages and Compilers – 2013, session 3

[15 marks] Exercise 1

Let $\mathcal{G}$ be the following grammar for arithmetic expressions with multiplication ($*$) and exponentiation ($\widehat{\ }$) operators:

$$E \quad \rightarrow \quad E * E \mid E \,\widehat{\ }\, E \mid (E) \mid \mathsf{id}$$

1. List all the conflicts of the SLR parsing table for $\mathcal{G}$, and state how each of them can be resolved under the following usual assumptions:
   - the multiplication operator is left associative;
   - the exponentiation operator is right associative;
   - exponentiation has higher precedence than multiplication.
2. Provide a LALR grammar $\mathcal{G}'$ where the ambiguity of $\mathcal{G}$ is resolved as said above.
3. Show the LALR parsing steps on input

   $$\mathsf{id} \,\widehat{\ }\, \mathsf{id} \,\widehat{\ }\, \mathsf{id} * \mathsf{id}$$

   and draw the resulting parse tree.

[8 marks] Exercise 2

Let r_1 and r_2 be defined as follows:

$r_1 = (ba)^*(a^*b^*|a^*)$

$r_2 = (ba)^*(b^* \mid a^*\epsilon).$

Provide the minimum DFAs to recognize $\mathcal{L}(r_1)$ and $\mathcal{L}(r_2)$, respectively, and say whether $\mathcal{L}(r_1) = \mathcal{L}(r_2)$ or not.

[4 marks] Exercise 3

Let $\mathcal{D}$ be the following partially specified syntax-directed definition for flow-of-control statements:

$$P \rightarrow S \qquad \{ \; S.next = newlabel() \quad P.code = S.code \parallel label(S.next) \; \}$$

$$S \rightarrow \textbf{while } (B)\ S_1$$

$$S \rightarrow \textbf{loop } S_1 \textbf{ break on } B \textbf{ else } S_2 \textbf{ endloop}$$

where:

- B can be assumed to generate a boolean expression and to have associated attributes $B.code$, $B.true$, and $B.false$ with the usual meaning;
- the semantics of the **while**-command is the usual one;
- the intended meaning of the **loop**-command is as follows. First S_1 is executed. If B is true in the resulting state then the execution of the whole command is over, otherwise S_2 is executed and then the **loop**-command is executed again.

Add attribution rules to $\mathcal{D}$ to get the translation of statements to code.

[3 marks] Exercise 4

Say, justifying your answer, whether the following statement is true or not:

> "Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be regular languages. Then $\mathcal{L}_1 \cap \mathcal{L}_2$ is a regular language."

Formal Languages and Compilers – 2013, session 4

[15 marks] Exercise 1

Let $\mathcal{G}$ be the following grammar for regular expressions over $\{a, b\}$ with concatenation ($\bullet$), alternation ($+$), and Kleene-star (*) operators:

$$R \quad \rightarrow \quad R + R \mid R \bullet R \mid R^* \mid (R) \mid a \mid b$$

1. List all the conflicts of the SLR parsing table for $\mathcal{G}$, and state how each of them can be resolved under the following usual assumptions:
   - concatenation and alternation are both left associative;
   - the Kleene-star operator has highest precedence;
   - the concatenation operator has the second highest precedence;
   - the alternation operator has the lowest precedence.

   For example, $r = a + b^* \bullet a$ stands for $a + ((b^*) \bullet (a))$.
2. Provide a SLR grammar $\mathcal{G}'$ where the ambiguity of $\mathcal{G}$ is resolved as said above.
3. Show the SLR parsing steps on input r and draw the resulting parse tree.

[8 marks] Exercise 2

Let $\mathcal{G}$ be the following grammar:

$$\begin{array}{lcl} S & \rightarrow & aB \mid bA \\ A & \rightarrow & \epsilon \\ B & \rightarrow & bA \mid aC \\ C & \rightarrow & aC \mid bA \end{array}$$

Provide the minimum DFA to recognize $\mathcal{L}(\mathcal{G})$.

[4 marks] Exercise 3

Let $\mathcal{D}$ be the following partially specified syntax-directed definition for statements:

$$
\begin{array}{lcll}
P & \rightarrow & S & \{ \quad S.next = newlabel() \\
 & & & \quad P.code = S.code \parallel label(S.next) \quad \} \\
S & \rightarrow & \textbf{while } (B)\ S_1 & \\
S & \rightarrow & \textbf{repeat } N \textbf{ times } S_1 \textbf{ endrepeat} &
\end{array}
$$

where:

- B can be assumed to generate a boolean expression and to have associated attributes $B.code$, $B.true$, and $B.false$ with the usual meaning;
- the semantics of the **while**-command is the usual one;
- N can be assumed to generate a natural number and to have an attribute $N.val$ containing the value of the generated number;
- the intended meaning of the **repeat**-command is that S_1 is executed $N.val$ times.

Add attribution rules to $\mathcal{D}$ to get the translation of statements to code.

[3 marks] Exercise 4

Let $\mathcal{L} = \{a^j b^k a^{j-k} \mid j, k \geq 0 \text{ and } j > k\}$. Say whether $\mathcal{L}$ is a context-free language or not.

Formal Languages and Compilers – 2013, session 5

[15 marks] Exercise 1

Let $\mathcal{G}$ be defined as follows:

$$
\begin{array}{lcl}
S & \rightarrow & bS \mid Sd \mid AB \\
A & \rightarrow & bAa \mid ba \\
B & \rightarrow & cBd \mid cd
\end{array}
$$

1. Show that $\mathcal{G}$ is not SLR.
2. Provide a LALR(1) grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})$.
3. Using the LALR(1) parsing table for $\mathcal{G}'$, show the parsing steps on input $bbbacd$ and draw the resulting parse tree.

[8 marks] Exercise 2

Provide the minimum DFA to recognize the language generated by the following regular expression:

$$(a(ba)^*b)^*(ab \mid a(ab)^* \mid \epsilon).$$

[4 marks] Exercise 3

Let $\mathcal{G}$ be an SLR grammar for arithmetic expressions with infix addition and infix multiplication operators.

1. Add attribution rules to $\mathcal{G}$ so that the synthesised attribute of the start symbol of $\mathcal{G}$ contains the translation of the generated expressions from infix to prefix notation. For example, on input $id_1 + id_2 * id_3$, the synthesised attribute should be the string $+id_1 * id_2 id_3$.
2. Show the evaluation for the derivation of $id_1 + id_2 * id_3$.

[3 marks] Exercise 4

Let $\mathcal{L} = \{a^j b^h c^k \mid j, h, k \geq 0 \text{ and } j + h = k\}$. Say whether $\mathcal{L}$ is a context-free language or not.