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
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \;\|\|\; label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \;\|\|\; label(B.true) \;\|\|\; S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\|\| \; label(B.true) \;\|\|\; S_1.code$ $\|\| \; gen('\texttt{goto}'\; S.next)$ $\|\| \; label(B.false) \;\|\|\; S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \;\|\|\; B.code$ $\|\| \; label(B.true) \;\|\|\; S_1.code$ $\|\| \; gen('\texttt{goto}'\; begin)$
$S \rightarrow S_1 \; S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \;\|\|\; label(S_1.next) \;\|\|\; S_2.code$

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:

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

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:

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

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:

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

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 ($\hat{}$) operators:

$$E \quad \to \quad E * E \mid E \,\hat{}\, E \mid (E) \mid \mathsf{id}$$

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

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

   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:

$$\begin{array}{lcll} P & \to & S & \{ \quad S.next = newlabel() \\ & & & \phantom{\{} \quad P.code = S.code \parallel label(S.next) \quad \} \\ S & \to & \textbf{while } (B)\ S_1 & \\ S & \to & \textbf{loop } S_1 \textbf{ break on } B \textbf{ else } S_2 \textbf{ endloop} & \end{array}$$

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;
- 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:

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

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.

Formal Languages and Compilers – 2014, session 1

Exercise 1

Let the regular expressions r_1 and r_2 be defined as follows:

$$\begin{array}{rcl} r_1 & = & (a(ba)^*(b \mid \epsilon)^*)^* \\ r_2 & = & (c(dc)^*d^*)^* \end{array}$$

Provide the minimum DFA to recognize the language

$$\mathcal{L} = \{w_1 w_2 \mid w_1 \in \mathcal{L}(r_1) \text{ and } w_2 \in \mathcal{L}(r_2)\}.$$

Exercise 2

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

$$\begin{array}{rcl} B & \rightarrow & B \ \mathsf{or} \ T \mid T \\ T & \rightarrow & T \ \mathsf{and} \ F \mid F \\ F & \rightarrow & \mathsf{not} \ F \mid (B) \mid \mathsf{id} \end{array}$$

where $\{\mathsf{or}, \mathsf{and}, \mathsf{not}, (,), \mathsf{id}\}$ is the set of terminal symbols.

1. Show that $\mathcal{G}$ is not LL(1).
2. Provide a LL(1) grammar $\mathcal{G}'$ that, like $\mathcal{G}$, enforces the usual precedence and associativity of operators, and such that $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})$.

Exercise 3

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

$$\begin{array}{rcl} S & \rightarrow & E \\ E & \rightarrow & TE' \\ E' & \rightarrow & +TE' \mid \epsilon \\ T & \rightarrow & (E) \mid \mathsf{num} \end{array}$$

1. Based on $\mathcal{G}$, provide a L-attributed grammar to compute in $S.val$ the value of the parsed expression. For example, on input $2+3+4$ the attribute $S.val$ should be assigned the number 9.
2. Show the dependency graph and the evaluation for the derivation of $2+3+4$.

Formal Languages and Compilers – 2014, session 2

Exercise 1

Let

$$r_1 = (a(ab)^*b)^*(\epsilon \mid a(ab)^* \mid aabb)$$

$$r_2 = (a(ab)^*b)^*$$

Provide the minimum DFA that recognizes the language $\mathcal{L}(r_1) \setminus \mathcal{L}(r_2)$, namely the set of words belonging to $\mathcal{L}(r_1)$ but not to $\mathcal{L}(r_2)$.

Exercise 2

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

$$\begin{array}{lcl} S & \rightarrow & aA \mid bBc \\ A & \rightarrow & Bd \mid Cc \\ B & \rightarrow & e \mid \epsilon \\ C & \rightarrow & f \mid \epsilon \end{array}$$

Say, justifying your answer, whether $\mathcal{G}$ is LALR or not.

Exercise 3

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

$$\begin{array}{lcll} P & \rightarrow & S & \{ \;\; S.next = newlabel() \\ & & & \quad\;\; P.code = S.code \parallel label(S.next) \quad \} \\ S & \rightarrow & \ldots & \\ S & \rightarrow & \textsf{for}\ (S_1 \ ; B \ ; S_2)\ S_3 & \end{array}$$

where:

- B can be assumed to generate a boolean expression to be dealt with by the usual attributes $B.code$, $B.true$, and $B.false$;
- the intended meaning of the command "$\textsf{for}\ (S_1 \ ; B \ ; S_2)\ S_3$" is the same as the meaning of "$S_1 \ ; \textsf{while}\ (B)\ \{S_3 \ ; S_2; \};$".

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

Formal Languages and Compilers – 2014, session 3

Exercise 1

Let the regular expressions r_1 and r_2 be defined as follows:

$$\begin{aligned} r_1 &= (ab \mid a)^* a \\ r_2 &= a(ba \mid a)^* \end{aligned}$$

Say, justifying your answer, whether $\mathcal{L}(r_1) = \mathcal{L}(r_2)$ or not.

Exercise 2

Let $\mathcal{G}$ be the following ambiguous grammar for the λ-calculus:

$$E \rightarrow \mathsf{v} \mid \lambda\mathsf{v}.\, E \mid EE \mid (E)$$

where E is the single non-terminal symbol, $\lambda\mathsf{v}.\, E$ represents abstraction w.r.t. the variable v in E, and EE represents application.

1. Define an LL(1) grammar $\mathcal{G}'$ such that $\mathcal{L}(\mathcal{G}') = \mathcal{L}(\mathcal{G})$ and the ambiguity of $\mathcal{G}$ is resolved by imposing the following usual conventions:
   - abstraction is right associative;
   - application is left associative;
   - application has higher priority than abstraction.
2. Show the LL(1) parsing table for $\mathcal{G}'$ and the parse tree obtained when parsing the string $\lambda\mathsf{v}_1.\, \lambda\mathsf{v}_2.\, \mathsf{v}_1\mathsf{v}_2\mathsf{v}_1$.

Exercise 3

Let $\mathcal{G}$ be the following LL(1) grammar for arithmetic expressions:

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow (E) \mid \mathsf{num} \end{aligned}$$

1. Enrich $\mathcal{G}$ with synthesized and inherited attributes for the computation in $S.val$ of the value of the parsed expression. For example, on input $1 + 2 * 3$ the attribute $S.val$ should be assigned the number 7.
2. Show the evaluation order for the derivation of the derivation of $1 + 2 * 3$.

Formal Languages and Compilers – 2014, session 4

Exercise 1

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

> "Let $\mathcal{D}$ be a DFA whose states are all final, and let $\mathcal{D}$ recognize the language $\mathcal{L}$. Then $\mathcal{D}$ is the minimum DFA to recognize $\mathcal{L}$."

Exercise 2

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

$$
\begin{array}{lcl}
A & \rightarrow & BC \\
B & \rightarrow & aD \mid bD \\
C & \rightarrow & BCC \mid aCd \mid \epsilon \\
D & \rightarrow & aD \mid bD \mid \epsilon
\end{array}
$$

Provide the LL(1) parsing table for $\mathcal{G}$.

Exercise 3

Let $\mathcal{G}$ be the following grammar for declaration of typed identifiers:

$$
\begin{array}{lcl}
D & \rightarrow & TL \\
T & \rightarrow & BC \\
B & \rightarrow & \mathsf{int} \mid \mathsf{float} \\
C & \rightarrow & [\mathsf{num}]C \mid \epsilon \\
L & \rightarrow & \mathsf{id}M \\
M & \rightarrow & ,\mathsf{id}M \mid \epsilon
\end{array}
$$

1. Enrich $\mathcal{G}$ to provide a translation schema whose side-effect is to add the relative type expression to the symbol-table entry of each of the declared identifiers. Type expressions are generated by the language

   $$T \rightarrow integer \mid float \mid array(n, T)$$

   where n stands for any number. For example, the type expression corresponding to the type declaration $\mathsf{int}\,[3][4]$ is $array(3, array(4, integer))$.
   Assume that $\mathsf{id}.entry$ represents the address of the table entry for the identifier id. Also, assume the existence of the auxiliary function $add.type(addr, t)$ whose effect is to enter the type t into the table entry at address $addr$.

2. Show the evaluation order for the derivation of $\mathsf{int}\,[3][4]\,\mathsf{id}_1, \mathsf{id}_2$.

Formal Languages and Compilers – 2014, session 5

Exercise 1

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

$$\begin{array}{lcl} S & \rightarrow & aA \mid bC \mid \epsilon \\ A & \rightarrow & aB \mid bS \\ B & \rightarrow & aC \mid bA \\ C & \rightarrow & aS \mid bB \end{array}$$

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

Exercise 2

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

$$\begin{array}{lcl} S & \rightarrow & Aa \\ A & \rightarrow & bB \mid c \\ B & \rightarrow & aA \mid \epsilon \end{array}$$

Provide the LALR parsing table for $\mathcal{G}$, and conclude whether $\mathcal{G}$ is LALR or not.

Exercise 3

Let $\mathcal{G}$ be the following grammar for declaration of typed identifiers:

$$\begin{array}{lcl} D & \rightarrow & TL \\ T & \rightarrow & \mathsf{int} \mid \mathsf{float} \mid [T] \mid [\,] \\ L & \rightarrow & \mathsf{id}M \\ M & \rightarrow & ,\mathsf{id}M \mid \epsilon \end{array}$$

1. Enrich $\mathcal{G}$ to provide a translation schema whose side-effect is to add the relative type expression to the symbol-table entry of each of the declared identifiers. Type expressions are generated by the language

   $T \rightarrow integer \mid float \mid list(T) \mid \alpha\_list$

   For example, the types corresponding to the type declarations $[[int]]$ and $[\,]$ are $list(list(integer))$ and, respectively, α_list.
   Assume that $\mathsf{id}.entry$ represents the address of the table entry for the identifier id. Also, assume the existence of the auxiliary function $add.type(addr, t)$ whose effect is to enter the type t into the table entry at address $addr$.

2. Show the evaluation order for the derivation of $[[int]]\,\mathsf{id}_1, \mathsf{id}_2$.