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

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

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	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

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