COL226: Programming Languages

II semester 2015-16

Sat 13 Feb 2016 Minor 1 60 minutes Max Marks 40

1. **[7 marks]** Prove that for any two functions $f : \texttt{'a} -> \texttt{'b}$, $g : \texttt{'b} -> \texttt{'c}$ and any list $L : \texttt{'a list}$

$$map\ (g\ o\ f)\ L = (map\ g)\ (map\ f\ L)$$

where $map\ f\ [\,] = [\,]$ and $map\ f\ (h :: t) = (f\ h) :: (map\ f\ t)$

Solution.

The proof is by structural induction on list L.

Basis

$$map\ (g\ o\ f)\ [\,] = [\,],\ \textit{using base case of map function definition, i.e., } map\ f\ [\,] = [\,]. \tag{1}$$

$$(map\ g)(map\ f\ [\,]) = (map\ g)\ [\,], \qquad \textit{using } map\ f\ [\,] = [\,] \tag{2}$$

$$= [\,], \qquad \textit{using } map\ f\ [\,] = [\,]. \tag{3}$$

Induction Hypothesis

Assume for any two functions $f : \texttt{'a}- >' \texttt{b}, g : \texttt{'b}- >' \texttt{c}$, and a list $T : \texttt{'a list}$ of length $k \geq 0$,

$$map\ (g\ o\ f)\ T = (map\ g)(map\ f\ T)$$

Induction Step

Let $L = x :: T$ be any list of length $k + 1$.

$$\begin{array}{lll} & map\ (g\ o\ f)\ L & \\ = & map\ (g\ o\ f)\ (x :: T) & \\ = & ((g\ o\ f)\ x) :: (map\ (g\ o\ f)\ T) & \{\text{definition of } map\} \\ = & (g\ (f\ x)) :: (map\ (g\ o\ f)\ T) & \{\text{definition of } o\} \\ = & (g\ (f\ x)) :: (map\ g\ (map\ f\ T)) & \{\text{Induction Hypothesis}\} \\ = & (map\ g)(f\ x ::\ map\ f\ T) & \{\text{definition of } map\} \\ = & (map\ g)(map\ f\ L) & \{\text{definition of } map\} \end{array}$$

2. **[5+5 = 10 marks]** A 'wise man' once claimed, "*A picture is worth a thousand words*". However it is not clear what a million words are worth. Consider the language

$$L = \{\{a,b\}^k c \{a,b\}^k \mid 1 \le k \le 1000000\}$$

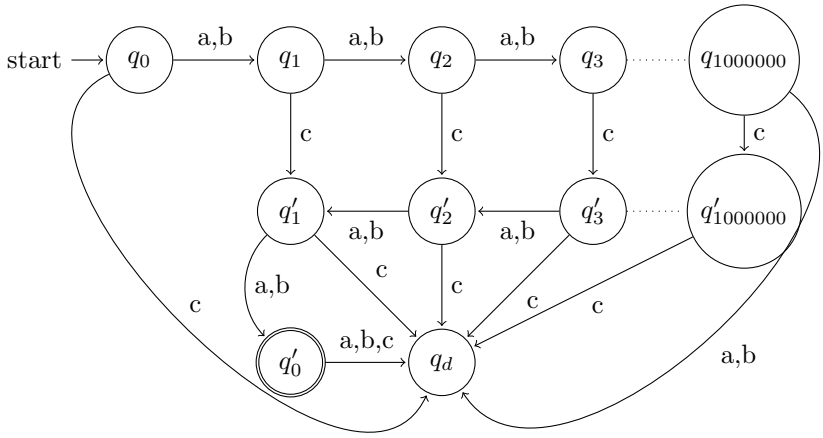
over the alphabet $A = \{a, b, c\}$.

(a) Design a deterministic finite automaton which accepts the language L.

(b) Design a deterministic finite automaton which accepts the language L^*.

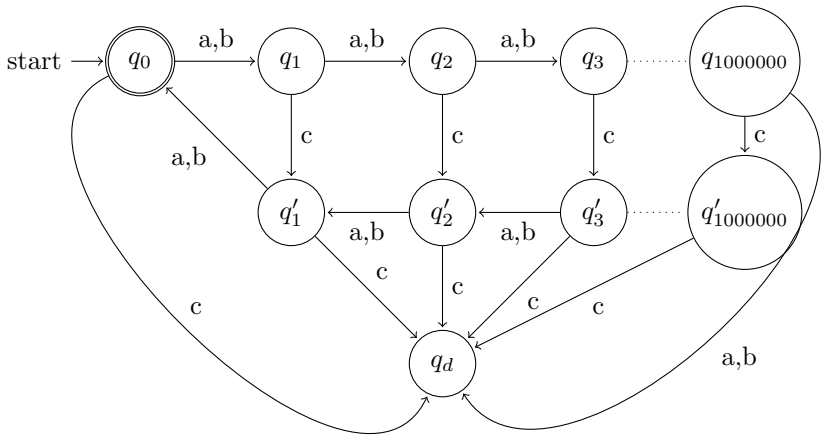
Solution.

(a) Let $\mathcal{D} = \langle Q, A, q_0, \delta, \{q_0'\}\rangle$ where $Q = \{q_k, q_k' \mid 0 \le k \le 10^6\} \cup \{q_d\}$, $A = \{a, b, c\}$, $\delta : Q \times A \to Q$ with $\delta(q_i, a) = \delta(q_i, b) = q_{i+1}$, for $0 \le i < 10^6$, $\delta(q_i, c) = q_i'$, for $1 \le i \le 10^6$ and $\delta(q_i', a) = \delta(q_i', b) = q_{i-1}'$, for $1 < i \le 10^6$, $\delta(q_0', a) = \delta(q_0', b) = \delta(q_0', c) = q_d$, $\delta(q_d, a) = \delta(q_d, b) = \delta(q_d, c) = q_d$, and $\delta(q_0, c) = \delta(q_{10^6}, a) = \delta(q_{10^6}, b) = q_d$
If you desperately want a state diagram for the above DFA, here it is:



(b) Let $\mathcal{D}^* = \langle Q, A, q_0, \delta, \{q_0\}\rangle$ where $Q = \{q_k, q_k' \mid 1 \le k \le 10^6\} \cup \{q_0, q_d\}$, $\delta : Q \times A \to Q$ with $\delta(q_i, a) = \delta(q_i, b) = q_{i+1}$, for $0 \le i < 10^6$, $\delta(q_i, c) = q_i'$, for $1 \le i \le 10^6$ and $\delta(q_{i+1}', a) = \delta(q_{i+1}', b) = q_i'$, for $1 < i < 10^6$, $\delta(q_1', a) = \delta(q_1', b) = q_0$ and all other cases go to q_d a dead state. Note that q_0 is both start state and accepting state.

If you desperately want a state diagram for the above DFA, here it is:



3. **[10 marks]** Let $RE(\mathtt{A})$ denote the set of regular expressions over the alphabet $\mathtt{A}$ such that none of the regular expression operators occur as symbols in $\mathtt{A}$. That is, the set $\{\varepsilon, \emptyset, *, (,), ., |\}$ is disjoint from $\mathtt{A}$.

   (a) Define $RE(\mathtt{A})$ in EBNF.

   (b) Define a context-free grammar that generates $RE(\mathtt{A})$.

   Solution.

   (a) Assume $\mathtt{A} = \{a_1, \ldots, a_n\}$

$$
\begin{array}{lcl}
\mathit{RegExp} & ::= & \mathit{EmptyRegExp} \mid \mathit{NonemptyRegExp} \\
\mathit{SimpleRegExp} & ::= & \text{“}\emptyset\text{”} \mid \text{“}\varepsilon\text{”} \\
\mathit{Alphabet} & ::= & a_1 \mid \cdots \mid a_n \\
\mathit{NonemptyRegExp} & ::= & \mathit{Alphabet} \mid \\
 & & \mathit{NonemptyRegExp}\text{“}\{.\text{”}\mathit{NonemptyRegExp}\text{“}\}\text{”} \mid \\
 & & \mathit{NonemptyRegExp}\text{“}\{|\text{”}\mathit{NonemptyRegExp}\text{“}\}\text{”} \mid \\
 & & \mathit{NonemptyRegExp}\text{“}*\text{”} \mid \\
 & & \text{“}(\text{”}\mathit{NonemptyRegExp}\text{“})\text{”}
\end{array}
$$

   (b) Let $O = \{\varepsilon, \emptyset, *, (,), ., |\}$. Then the set of terminal symbols is given by $T = \mathtt{A} \cup O$. $G = \langle N, T, P, S\rangle$ and the sets N, P and S are as shown below in the rewrite rules.

      i. Since the question does not specify any precedence between operators the corresponding CFG may be specified unambiguously by a fully-parenthesized grammar as follows.

$$
\begin{array}{lcl}
S & \longrightarrow & \emptyset \mid \varepsilon \mid R \\
R & \longrightarrow & A \mid (R.R) \mid (R|R) \mid (R*) \\
A & \longrightarrow & a_1 \mid \cdots \mid a_n
\end{array}
$$

      ii. Alternatively we may define the precedence of operators using the usual convention that unary operators have the highest precedence followed by the binary operators. We may assume further that all binary operators are left associative and that . has a higher precedence than |. Then we have the following rules.

$$
\begin{array}{lcl}
S & \longrightarrow & \emptyset \mid \varepsilon \mid R \\
R & \longrightarrow & D \mid D|R \\
D & \longrightarrow & K \mid K.D \\
K & \longrightarrow & F \mid F* \\
F & \longrightarrow & A \mid (S) \\
A & \longrightarrow & a_1 \mid \cdots \mid a_n
\end{array}
$$

4. **[7+6=13 marks]** Let $O = \{\oplus, \odot, \otimes\}$ be three right-associative infix binary operators in an expression language which has following precedence rules.

   (a) $\odot$ has the highest precedence and $\oplus$ has the lowest precedence with $\otimes$ having precedence strictly in between $\odot$ and $\oplus$.

   (b) ~~In any expression, between two neighbouring occurrences of the same operator, the left one has higher precedence than the one on the right.~~

   Let $I = \{a, b, c\}$ be the set of identifier tokens and $B = \{(,)\}$ is the set of brackets that are allowed for grouping sub-expressions together. Then $T = I \cup O \cup B$ is the set of terminal tokens.

   (a) Design a context-free grammar to generate all valid expressions in this language which allows for any expressions (with or without the use of brackets) to be unambiguously parsed in such a way that the precedence rules are respected.

   (b) Design attribute grammar rules to transform every expression into a unique expression belonging to the language generated by the rules

$$\begin{array}{lcl} S & \longrightarrow & I \mid (S \oplus S) \mid (S \otimes S) \mid (S \odot S) \\ I & \longrightarrow & a \mid b \mid c \end{array}$$

   Solution.

   (a) The required grammar is $G = \langle \{S, P, M, D, I\}, \{a, b, c\}, \mathcal{P}, S \rangle$ where the set $\mathcal{P}$ of productions is as follows.

$$\begin{array}{lcl} S & \longrightarrow & P \mid P \oplus S \\ P & \longrightarrow & M \mid M \otimes P \\ M & \longrightarrow & D \mid D \odot M \\ D & \longrightarrow & I \mid (S) \\ I & \longrightarrow & a \mid b \mid c \end{array}$$

   The above grammar is for right-associative binary operators with the given precedence. It will generate many expressions with (some or no) parentheses and the rules of precedence will determine how the grouping needs to be done.

   (b) The grammar G' given in question (4b) generates only a subset of the strings generated by the solution G in answer (4a). The task now is to give attribute definitions to the rules in G such that every expression (with some or no) parentheses is converted into a fully parenthesised expression with the right kind of grouping as intended in the string. Since such a translation fully parenthesizes every compound expression generated by G, there would be no need to use the precedence rules when evaluating the expression. We synthesize a single attribute *str* for each non-terminal symbol

$$\begin{array}{rlcll} 1. & S & \longrightarrow & P & \triangleright \; S.str := P.str \\ 2. & S_0 & \longrightarrow & P \oplus S_1 & \triangleright \; S_0.str := \text{“(”} P.str \oplus S_1.str \text{“)”} \\ 3. & P & \longrightarrow & M & \triangleright \; P.str := M.str \\ 4. & P_0 & \longrightarrow & M \otimes P_1 & \triangleright \; P_0.str := \text{“(”} M.str \oplus P_1.str \text{“)”} \\ 5. & M & \longrightarrow & D & \triangleright \; M.str := D.str \\ 6. & M_0 & \longrightarrow & D \odot M_1 & \triangleright \; M_0.str := \text{“(”} D.str \oplus M_1.str \text{“)”} \\ 7. & D & \longrightarrow & I & \triangleright \; D.str := I.str \\ 8. & D & \longrightarrow & (S) & \triangleright \; D.str := S.str \\ 9. & I & \longrightarrow & a & \triangleright \; I.str := \text{“a”} \\ 10. & I & \longrightarrow & b & \triangleright \; I.str := \text{“b”} \\ 11. & I & \longrightarrow & c & \triangleright \; I.str := \text{“c”} \end{array}$$

   Notice that rule 8. removes existing parentheses (whether they are redundant or not) and the rules 2., 4., 6. ensure that there are exactly as many pairs of parentheses as there are occurrences of binary operators in the expression.