Assignment 15, Authomata Theory

Oleg Sivokon

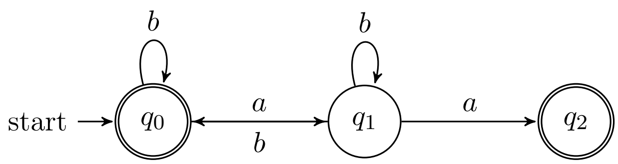
<2016-01-01 Fri>

Contents

1 Problems

1.1 Problem 1

1. Show right-linear grammar equivalent to the automaton given below:



2. Build a finite automaton for the language given by the grammar rules:

$$
\begin{aligned}
S &\rightarrow bS \mid aR \\
R &\rightarrow bS \mid aaR \mid aa \mid b\ .
\end{aligned}
$$

1.1.1 Answer 1

First, I will extract the δ function from the given automaton. Then, I'll use the cases of this function to generate the grammar:

δ	a	b	ϵ
q_0	$\{q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_1, q_0\}$	-
q_2	-	-	$\{q_2\}$

Rename the variables: $q_0 = S$, $q_1 = X$ and $q_2 = Y$, then the resulting grammar is given by:

$$
\begin{aligned}
S &\rightarrow aX \mid bS \mid \epsilon \\
X &\rightarrow aY \mid bX \mid bS \\
Y &\rightarrow \epsilon\ .
\end{aligned}
$$

All rules in this grammar are either of the form $V \to t$ or $v \to tW$, where V and W are variables and t is a terminal.

1.1.2 Answer 2

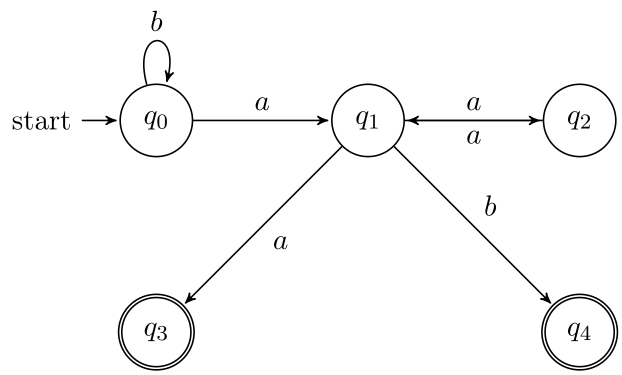
First, I will modify the grammar to make it easier to construct the δ for the desired automaton by introducing new rules: $X \to aR$, $Y \to a$ and $Z \to b$. Now we can rewrite the grammar:

$$
\begin{aligned}
&S \to bS \mid aR \\
&R \to bS \mid aX \mid aY \mid Z \\
&X \to aR \\
&Y \to a \\
&Z \to b\ .
\end{aligned}
$$

Renaming the variables: $S = q_0$, $R = q_1$, $X = q_2$, $Y = q_3$ and $Z = q_4$ gives the following δ function:

δ	a	b
q_0	$\{q_1\}$	$\{q_0\}$
q_1	$\{q_2, q_3\}$	$\{q_0, q_4\}$
q_2	$\{q_1\}$	-
$*q_3$	-	-
$*q_4$	-	-

Having δ we can build the automaton:



1.2 Problem 2

Given a language $L = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$, show context-free grammar G which accepts it.

1.2.1 Answer 3

L contains an empty string, thus G must have the $S \to \epsilon$ rule. Also, for any a is must add a b somewhere in the word. There are only two possible ways to do that: either a is before b or the other way around. Thus, we need to add two more rules: $S \to aSbS$ and $S \to bSaS$. Using structural induction on the length of the word generated by S we can also show that for any word in L there is a derivation in G. Hence G is given by:

$$S \to aSbS \mid bSaS \mid \epsilon \, .$$

1.3 Problem 3

Show a context-free grammar G accepting the language L defined as:

$$L = \{dw_1dw_2d\dots w_nd \mid n \geq 4 \\ \wedge \forall k : w_k \in \{a,b,c\}^* \\ \wedge \exists k : (2 \geq k \geq n-2 \wedge \#_c(w_{k+2}) = |w_k|)\}$$

1.3.1 Answer 4

The words in L will have to start with at least two repetitions of dw, followed by the part with a simplified requirement: there will be no restrictin on the value of k. Since context-free grammars are closed under concatenation, I will construct the grammar in steps.

1. G_1 is the grammar for two repetitions of dw:

$$\begin{aligned} &S \to dX \\ &W \to aW \mid bW \mid cW \mid \epsilon \\ &X \to WdY \\ &Y \to WdW\ . \end{aligned}$$

2. G_2 is the grammar that counts the number of letters in the k^{th} word and ensures that $k+2^{th}$ word has as many c letters.

$$\begin{aligned} &S \to dCT \\ &W \to aW \mid bW \mid cW \mid \epsilon \\ &C \to aCc \mid bCc \mid cCc \mid dWd \\ &T \to TWd \mid d\ . \end{aligned}$$

C variable is used to count the number of letters in the k^{th} word. T variable is used to add zero or more repetitions of dw after the $k+2^{th}$ word was found. Followed by the final d. W is variable for generating arbitrary long wrods w_i.

Now, we can combine both grammars:

$$
\begin{aligned}
&S \to dX \\
&W \to aW \mid bW \mid cW \mid \epsilon \\
&X \to WdY \\
&Y \to WdWZ \\
&Z \to dCT \\
&C \to aCc \mid bCc \mid cCc \mid dWd \\
&T \to TWd \mid d\ .
\end{aligned}
$$

1.4 Problem 4

Given the grammar G:

$$
\begin{aligned}
&S \to ABC \mid bB \mid D \\
&A \to a \mid \epsilon \\
&B \to bB \mid \epsilon \\
&C \to c \\
&D \to Da \mid aDc \mid Dc \mid ac \mid a \mid c\ .
\end{aligned}
$$

1. Is G unambiguous?
2. Give an alternative description to $L(G)$.

1.4.1 Answer 5

G is ambigous, it is possible to derive ac via:

1. $S \to ABC$, $A \to a$, $B \to \epsilon$ and $C \to c$.
2. $S \to D$, $D \to ac$.

1.4.2 Answer 6

$L(G)$ is actually regular. If you look at all derivations from S separately, then ABC is equivalent to $(a+\epsilon)b^*c$, bB is equivalent to b^+. And D is equivalent to $(a+c)^+$. The later can be proved by induction on the word length generated by D.

Base step: The word of length 1 can be generated by D, sinc it produces both a and c terminals.

Inductive step: Suppose we can derive the word $(a+c)^+$ of length $n-1$ using D rule, then the word of length n would be generated by either the $D \to Da$ or $D \to Dc$ rule.

Hence, by induction, D generates the language $(a+c)^+$.

Now $L(G) = (a+\epsilon)b^*c + b^+ + (a+c)^+$, since regular languages are closed under union.

1.5 Problem 5

Give a context-free grammar accepting the language

$$
\begin{aligned}
L = \{x{+}yz \mid\ & |x| = |y| \\
\mid\ & x, y \in 0, 1^* \\
\mid\ & |x| \mod 2 = 0 \iff z = \mathbf{e} \\
\mid\ & |x| \mod 2 = 1 \iff z = \mathbf{o}\}
\end{aligned}
$$

1.5.1 Answer 7

The grammar G accepting L can be given as follows:

$$\begin{aligned}
&S \to Oo \mid Ee \\
&X \to 00 \mid 01 \mid 10 \mid 11 \\
&E \to XEX \mid X + X \\
&O \to XOX \mid 0+0 \mid 0+1 \mid 1+0 \mid 1+1\ .
\end{aligned}$$

O variable is responsible for generating odd x and y while E is responsible for generating even sums. X generates all possible pairs of zeros and ones. It is easy to get convinced that E will generate all and only even sums (since concatenating X arbitrary number of times will produce only words of even length). Similarly, E will generate all and only odd sums.

Only odd sums can terminate in o and only even sums can terminate in e.

```
even_binary --> "00" ; "11" ; "01" ; "10" ;
                "00" , even_binary ;
                "10" , even_binary ;
                "01" , even_binary ;
                "11" , even_binary.

odd_binary --> "0" ; "1" ; "1", even_binary ; "0", even_binary.

odd_sums --> odd_binary , "+" , odd_binary.

even_sums --> even_binary , "+" , even_binary.

sums --> odd_sums, "o" ; even_sums, "e" .

print_helper(E) :-
    string_codes(X, E),
    (phrase(sums, E) ->
         format('\\item ~w \\textit{accepted}\n', [X])
     ;
     format('\\item ~w \\textit{rejected}\n', [X])).

assignment_15 :-
    format('$$\\begin{itemize}\n', []),
    Candidates = [`101+001o`, `1111+0000e`, `101+001`,
                  `1111+0000`, `abcd`, `1010101`,
                  `0101+1010o`, `001+110e`],
    maplist(print_helper, Candidates),
    format('\\end{itemize}$$', []).
```

101+001o *accepted*

1111+0000e *accepted*

101+001 *rejected*

1111+0000 *rejected*

abcd *rejected*

1010101 *rejected*

0101+1010o *rejected*

001+110e *rejected*