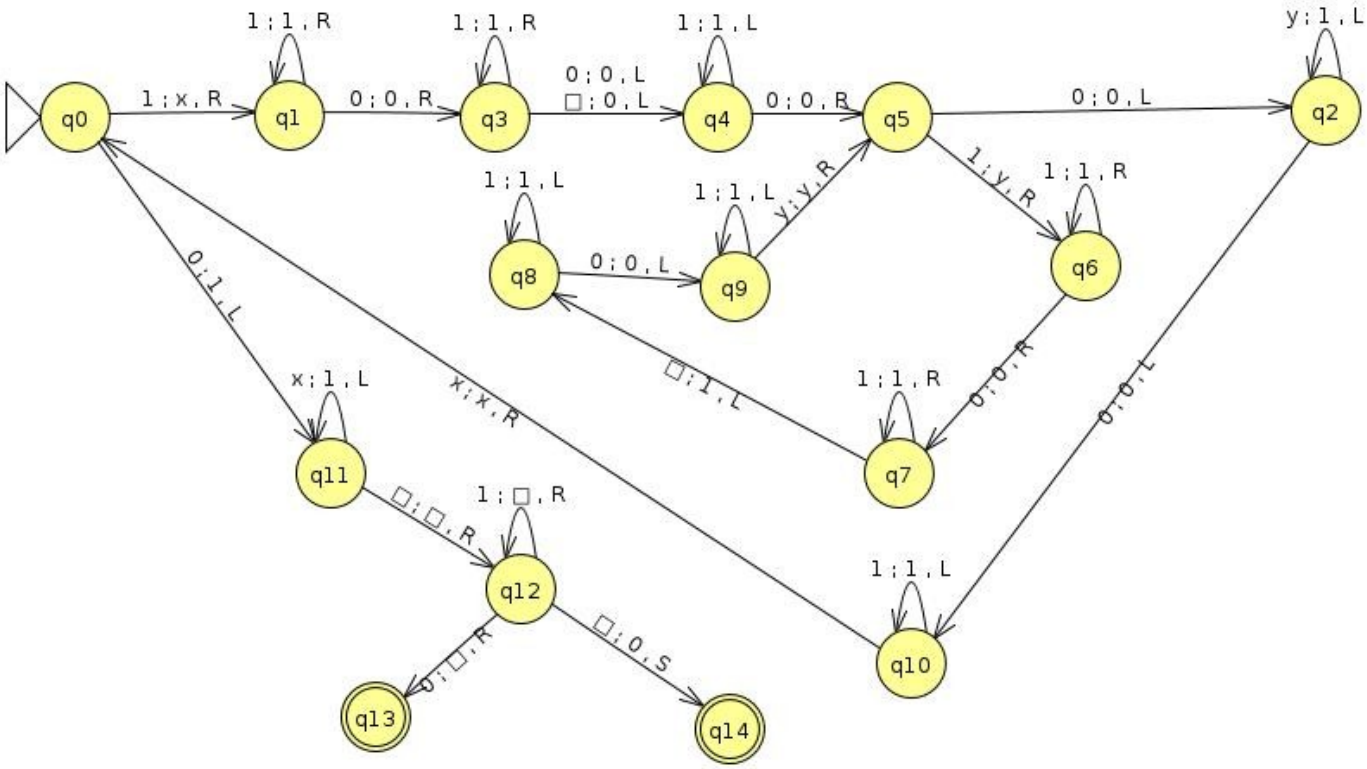
SESSIONAL II - TOC
MUHAMMAD HAMAD
MS-21354
MS-CS

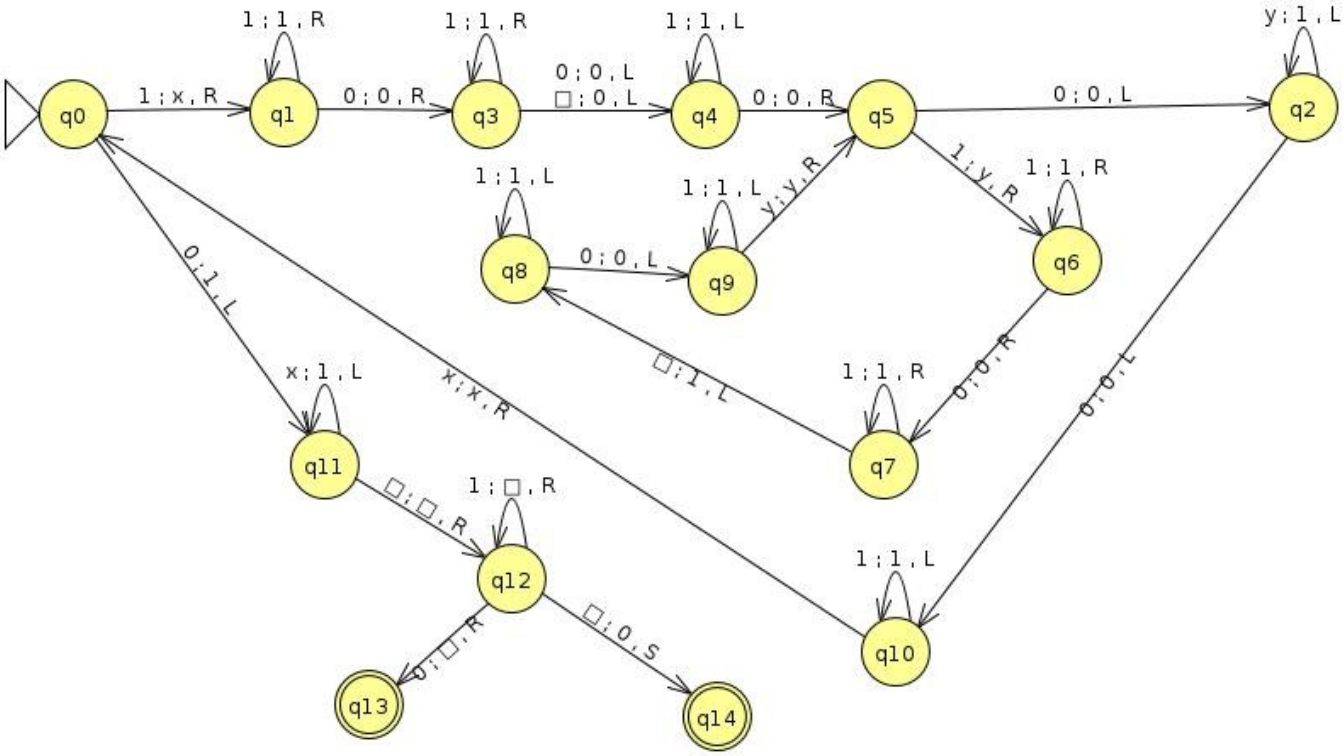
QUESTION NO 1:

TURING MACHINE FOR X/Y



QUESTION NO 2 :

TURING MACHINE FOR X*Y



QUESTION 3 :

PDA TO CFG CONVERSION

Here we give an example of how to create a CFG generating the language accepted by a PDA (by empty stack). The example PDA accepts the language $\{0^n1^n\}$. Formally, the PDA is $(Q, \Sigma, \Gamma, \delta, q, Z, F)$ where $Q = \{q, r\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{Z, X\}$, δ is defined by:

state	input	stack	new state	stack
q	0	Z	q	XZ
q	0	X	q	XX
q	1	X	r	s
r	1	X	r	s
r	s	Z	r	s

Since the PDA accepts by empty stack, the final set F is irrelevant.

The construction defines a CFG $tt = (V, T, P, S)$ where V contains a start symbol S as well as a symbol $[sY\,t]$ for every combination of stack symbol Y and states s and t. Thus

$V = \{S, [qZq], [qZr], [qXq], [qXr], [rZq], [rZr], [rXq], [rXr]\}$. $T = \{0, 1\}$, the input alphabet of the PDA. S is the start symbol for the grammar.

The intuitive meaning of variables like $[qXq]$ is that the represent the language (set of strings) that label paths from q to q that have the net effect of popping X off the stack (without going deeper into the stack).

The productions P of tt have two forms. First, for the start symbol S we add productions to the "[startState, startStackSymbol, state]" variable for every state in the PDA. The language generated by S will correspond to the set of strings labeling paths from S to any other state that have the net effect of emptying the stack (popping off the starting stack symbol).

For our example PDA we get the two productions:

$$S \rightarrow [qZq] \,/\, [qZr]$$

(recall that the "/" separates alternate right-hand-sides for the variable).

Next, for each transition in the PDA of the form $\delta(s, a, Y)$ contains (t, s) (i.e. state s goes to t while reading symbol a from the input and popping Y off the stack) we add the production

$$[sYt] \rightarrow a$$

to the grammar (we have three transitions of this form in the PDA, all into state r). This corresponds to the fact that there is a path from s to t labeled by a that has the net effect of popping Y off the stack.

After this stage we have the following productions (with all non-terminals listed even if they don't have any productions):

$$\begin{aligned}
S &\rightarrow [qZq][qZr] \\
[qZq] &\rightarrow \\
[qZr] &\rightarrow \\
[qXq] &\rightarrow \\
[qXr] &\rightarrow 1 \\
[rZq] &\rightarrow \\
[rZr] &\rightarrow s \\
[rXq] &\rightarrow \\
[rXr] &\rightarrow 1
\end{aligned}$$

These "push nothing" transitions are just a special case of the general rule:

> If there is a transition from s to t that reads a from the input, Y from the stack, and pushes k symbols $Y_1 Y_2 \cdots Y_k$ onto the stack, add all productions of the form $[sY\,s_k] \rightarrow a[tY_1 s_1][s_1 Y_2 s_2] \cdots [s_{k-1} Y_k s_k]$ to the grammar (for all combinations of states $s_1, s_2, \ldots, s_k$).

This expresses the intuition that the PDA can go from s to s_k with a net effect of popping Y by first going from s to t while popping Y pushing the Y_i's on the stack, and then taking a path that pops off each Y_i in turn.

In the "push nothing" case, this results in RHS's that are single terminals as above.

Lets apply the general rule to the $\delta(q, 0, Z) = (q, XZ)$ transition. In this case, we are pushing the string XZ of length 2 on the stack, and need to add all productions of the form:

$$[qZs_2] \rightarrow 0[qXs_1][s_1 Z_0 s_2]$$

where s_1 and s_2 can be any combinations of q and/or r. In other words, we add the productions: $[qZq] \rightarrow 0[qXq][qZq] \,/\, 0[qXr][rZq]$

$$[qZr] \rightarrow 0[qXq][qZr] \;/\; 0[qXr][rZr]$$

to the grammar.

Repeating this for the $\delta(q, 0, X) = (q, XX)$ transition gives us:

$$[qXq] \rightarrow 0[qXq][qXq] \;/\; 0[qXr][rXq]$$
$$[qXr] \rightarrow 0[qXq][qXr] \;/\; 0[qXr][rXr]$$

Collecting all of these productions together, we get (again listing all variables even if they have no productions):

$$\begin{aligned}
S &\rightarrow [qZq][qZr] \\
[qZq] &\rightarrow 0[qXq][qZq] \;/\; 0[qXr][rZq] \\
[qZr] &\rightarrow 0[qXq][qZr] \;/\; 0[qXr][rZr] \\
[qXq] &\rightarrow 0[qXq][qXq] \;/\; 0[qXr][rXq] \\
[qXr] &\rightarrow 1 \;/\; 0[qXq][qXr] \;/\; 0[qXr][rXr] \\
[rZq] &\rightarrow \\
[rZr] &\rightarrow s \\
[rXq] &\rightarrow \\
[rXr] &\rightarrow 1
\end{aligned}$$

To verify that the language generated by the grammar is the same as the language accepted by the PDA (by empty-stack) one would use two proofs by induction, one to

show that every word in the language of the PDA can be generated by the grammar, and another to show that every word that can be generated by the grammar is accepted by the PDA. Instead of doing those formal proofs here (perhaps they would make a good exercise?), we will look at an example string to gain further insight into why the construction works.

Consider the string 0011 in the language. The PDA accepts the string by pushing two X's on

the stack while reading 0's and then popping them off while reading 1's. To derive 0011 in the grammar we do the following:

$$S \Rightarrow [qZr] \Rightarrow 0[qXr][rZr] \Rightarrow 00[qXr][rXr][rZr] \Rightarrow 001[rXr][rZr] \Rightarrow 0011[rZr] \Rightarrow 0011$$ Consider

all the productions again. If we ever generate a $[rZq]$ or $[rXq]$ symbol, then we can't remove it (since those symbols have no productions). Therefore we can remove those symbols and the productions containing them from the grammar.

$$\begin{aligned}
S &\rightarrow [qZq][qZr] \\
[qZq] &\rightarrow 0[qXq][qZq] \\
[qZr] &\rightarrow 0[qXq][qZr] \;/\; 0[qXr][rZr] \\
[qXq] &\rightarrow 0[qXq][qXq] \\
[qXr] &\rightarrow 1 \;/\; 0[qXq][qXr] \;/\; 0[qXr][rXr] \\
[rZr] &\rightarrow s \\
[rXr] &\rightarrow 1
\end{aligned}$$

Now the only production for $[qZq]$ produces another $[qZq]$ symbol, so if we ever generate the $[qZq]$ variable we will never be able to get to a string of just terminals. Similarly, the only production for $[qXq]$ produces more $[qXq]$'s. Removing these two variables (and the productions that use them) simplifies the grammar to:

$$\begin{aligned}
S &\rightarrow [qZr] \\
[qZr] &\rightarrow 0[qXr][rZr] \\
[qXr] &\rightarrow 1 \;/\; 0[qXr][rXr] \\
[rZr] &\rightarrow s \\
[rXr] &\rightarrow 1
\end{aligned}$$

Variable $[rZr]$ generates only s, so it is a no-op and can be deleted. However, unlike the previous simplifications we must keep a modified versions (with $[rZR$ replaced by s) of the productions using $[rZr]$. Similarly, variable $[rXr]$ generates only the terminal 1, so we can replace all uses of it with 1.

$$\begin{aligned}
S &\rightarrow [qZr] \\
[qZr] &\rightarrow 0[qXr] \\
[qXr] &\rightarrow 1 \;/\; 0[qXr]1
\end{aligned}$$