Introduction to Formal Languages, Automata and Computability

Pushdown Automata

K. Krithivasan and R. Rama
We have considered the simplest type of automaton, viz., the finite state automaton. We have seen that a finite state automaton has finite amount memory and hence cannot accept type 2 languages like \( \{ a^n b^n | n \geq 1 \} \). In this chapter we consider a class of automata, the pushdown automata, which accept exactly the class of context-free (type 2) languages. The pushdown automaton is a finite automaton with an additional tape, which behaves like a stack. We consider two ways of acceptance and show the equivalence between them.
The equivalence between CFG and pushdown automata is also proved. Let us consider the following language over the alphabet \( \Sigma = \{a, b, c\} : L = \{a^n b^m c^n | n, m \geq 1\} \). To accept this we have an automaton which has a finite control and the input tape which contains the input. Apart from these, there is an additional pushdown tape which is like a stack of plates placed on a spring. Only the top most plate is visible. Plates can be removed from top and added at the top only. In the following example, we have a red plate
and a number of blue plates. The machine is initially in state $q_0$ and initially a red plate is on the stack. When it reads $a$ it adds a blue plate to the stack and remains in state $q_0$. When it sees the $b$, it changes to $q_1$. In $q_1$ it reads $b$’s without manipulating the stack. When it reads a $c$ it goes to state $q_2$ and removes a blue plate. In state $q_2$ it proceeds to read $c$’s and whenever it reads a $c$ it removes a blue plate. Finally in state $q_2$ without reading any input it removes the red plate. The working of the automaton can be summarized by the following table.
Let us see how the automaton treats the input $aabbbccc$.

<table>
<thead>
<tr>
<th>State</th>
<th>Top plate</th>
<th>Input</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$a$</td>
<td></td>
</tr>
<tr>
<td>$q_0$</td>
<td>red</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>remain in state $q_0$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>blue</td>
<td>add blue plate</td>
<td>go to $q_1$</td>
</tr>
<tr>
<td></td>
<td>remain in state $q_0$</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$q_1$</td>
<td>red</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>blue</td>
<td>-</td>
<td>remain in state $q_1$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>red</td>
<td>without waiting for input remove red plate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>blue</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Initially it is $q_0$ and reads a $a$. Top plate is red

\[
\begin{array}{ccccccc}
a & a & b & b & b & c & c \\
\uparrow \\
q_0 & & & & & & R
\end{array}
\]

When it reads the first $a$ in $q_0$ it adds a blue plate to the stack. Now the situation looks as follows:

\[
\begin{array}{ccccccc}
a & a & b & b & b & c & c \\
\uparrow \\
q_0 & & & & & B & R
\end{array}
\]

When its reads the second $a$ the automaton’s instantaneous description (ID) can be represented by
In $q_0$ when it reads a $b$ it goes to $q_1$.

In $q_1$ it reads a $b$ without manipulating the stack.
- In state $q_1$ when it reads a $c$ it removes a blue plate and goes to $q_2$

- In state $q_2$ when it reads a $c$ it removes a blue plate
Now the whole input has been read. The automaton is in $q_2$ and top plate is red. Now without looking for the next input it removes the red plate.

The current situation is represented as
The whole input has been read and the stack has been emptied.
The string is accepted by the automaton if the whole input has been read and the stack has been emptied. This automaton can accept in a similar manner any string of the form $a^n b^m c^n$, $n, m \geq 1$.
Now let us consider the formal definition of a push-down automaton.
Definition A pushdown automaton (PDA) $M = (K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a 7-tuple where
- $K$ is a finite set of states
- $\Sigma$ is a finite set of input symbols
- $\Gamma$ is a finite set of pushdown symbols
- $q_0$ in $K$ is the initial state
- $Z_0$ in $\Gamma$ is the initial pushdown symbol
- $F \subseteq K$ is the set of final states
- $\delta$ is the mapping from $K \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ into finite subsets of $K \times \Gamma^*$
- $\delta(q, a, z)$ contains $(p, \gamma)$ where $p, q \in K$, $a \in \Sigma \cup \{\epsilon\}$, $z \in \Gamma$, $\gamma \in \Gamma^*$ means that when the automaton is in state $q$ and reading $a$ (reading nothing if $a = \epsilon$) and the
top pushdown symbol in $z$, it can go to state $p$ and replace $z$ in the pushdown store by the string $\gamma$. If $\gamma = z_1 \ldots z_n$; $z_1$ becomes the new top symbol of the pushdown store. It should be noted that basically the pushdown automaton is nondeterministic in nature.

An instantaneous description of a PDA is a 3-tuple $(q, w, \alpha)$ where $q$ denotes the current state, $w$ is the portion of the input yet to be read and $\alpha$ denotes the contents of the pushdown store. $w \in \Sigma^*$, $\alpha \in \Gamma^*$ and $q \in K$. By convention the leftmost symbol of $\alpha$ is the top symbol of the stack.
If \((q, a a_1 a_2 \ldots a_n, z z_1 \ldots z_n)\) is an ID and \(\delta(q, a, z)\) contains \((p, B_1 \ldots B_m)\), then the next ID is \((p, a_1 \ldots a_n, B_1 \ldots B_m z_1 \ldots z_n)\), \(a \in \Sigma \cup \{\epsilon\}\).

This is denoted by
\[(q, a a_1 \ldots a_n, z z_1 \ldots z_n) \vdash (p, a_1 \ldots a_n, B_1 \ldots B_m z_1 \ldots z_n).\]
\(\vdash^*\) is the reflexive transitive closure of \(\vdash\). The set of strings accepted by the PDA \(M\) by emptying the pushdown store is denoted as \(\text{Null}(M)\) or \(N(M)\).

\[N(M) = \{w/w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon) \text{ for some } q \in K\}\]

This means that any string \(w\) on the input tape will be accepted by the PDA \(M\) by the empty store, if \(M\) started
in \( q_0 \) with its input head pointing to the leftmost symbol of \( w \) and \( Z_0 \) on its pushdown store, will read the whole of \( w \) and go to some state \( q \) and the pushdown store will be emptied. This is called acceptance by empty store. When acceptance by empty store is considered \( F \) is taken as the empty set. There is another way of acceptance called acceptance by final state. Here when \( M \) is started in \( q_0 \) with \( w \) on the input tape and input tape head pointing to the leftmost symbol of \( w \) and with \( Z_0 \) on the pushdown store, after some moves finally reads the whole input and reaches one of the final states. The pushdown store
need not be emptied in this case. The language accepted by the pushdown automaton by final state is denoted as $T(M)$.

$$T(M) = \{ w/ w \in \Sigma^*, (q_0, w, Z_0) \vdash^* (q_f, \epsilon, \gamma) \text{ for some } q_f \in F \text{ and } \gamma \in \Gamma^* \}.$$

**Example** Let us formally define the pushdown automaton for accepting $\{a^n b^m c^n/n, m \geq 1\}$ described informally earlier.

$M = (K, \Sigma, \Gamma, \delta, q_0, R, \phi)$ where $K = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{B, R\}$ and $\delta$ is given by

$\delta(q_0, a, R) = \{(q_0, BR)\}$

$\delta(q_0, a, B) = \{(q_0, BB)\}$

$\delta(q_0, b, B) = \{(q_1, B)\}$

$\delta(q_1, b, B) = \{(q_1, B)\}$
\[ \delta(q_1, c, B) = \{(q_2, \epsilon)\} \]
\[ \delta(q_2, c, B) = \{(q_2, \epsilon)\} \]
\[ \delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\} \]

The sequence of ID's on input \textit{aabbbcc} is given by,

\[
(q_0, aabbbcc, R) \vdash (q_0, abbbcc, BR) \vdash (q_0, bbbcc, BBR)
\]

\[
\vdash (q_1, bbbcc, BBR) \vdash (q_1, bcc, BBR) \vdash (q_1, cc, BBR)
\]

\[
\vdash (q_2, c, BR) \vdash (q_2, \epsilon, R) \vdash (q_2, \epsilon, \epsilon)
\]

It can be seen that the above PDA is deterministic. The general definition of PDA is nondeterministic. In order that a PDA is deterministic two conditions have to be satisfied.
At any instance, the automaton should not have a choice between reading a true input symbol or $\epsilon$; the next move should be uniquely determined. These conditions may be stated formally as follows: In a deterministic PDA (DPDA),

1. For all $q$ in $K$, $Z$ in $\Gamma$ if $\delta(q, \epsilon, Z)$ is nonempty $\delta(q, a, Z)$ is empty for all $a \in \Sigma$.

2. For all $q$ in $K$, $a$ in $\Sigma \cup \{\epsilon\}$, $Z$ in $\Gamma$, $\delta(q, a, Z)$ contains at most one element.
Theorem \( L \) is accepted by a PDA \( M_1 \) by empty store if and only if \( L \) is accepted by a PDA \( M_2 \) by final state.

Proof (i) Let \( L \) be accepted by a PDA \( M_2 = (K, \Sigma, \Gamma, \delta_2, q_0, Z_0, F) \) by final state. Then construct \( M_1 \) as follows:

\[ M_1 = (K \cup \{q'_0, q_e\}, \Sigma, \Gamma \cup \{X_0\}, \delta_1, q'_0, X_0, \phi) \]

We add two more states \( q'_0 \) and \( q_e \) and one more pushdown symbol \( X_0 \). \( q'_0 \) is the new initial state and \( X_0 \) is the new initial pushdown symbol. \( q_e \) is the erasing state.

\( \delta \) mappings are defined as follows:
1. $\delta_1(q_0', \epsilon, X_0)$ contains $(q_0, Z_0X_0)$
2. $\delta_1(q, a, Z)$ includes $\delta_2(q, a, Z)$ for all $q \in K$, $a \in \Sigma \cup \{\epsilon\}$, $Z \in \Gamma$
3. $\delta_1(q_f, \epsilon, Z)$ contains $(q_e, \epsilon)$ for $q_f \in F$ and $Z \in \Gamma \cup \{X_0\}$
4. $\delta_1(q_e, \epsilon, Z)$ contains $(q_e, \epsilon)$ for $Z \in \Gamma \cup \{X_0\}$

The first move makes $M_1$ go to the initial ID of $M_2$ (except for the $X_0$ in the pushdown store). Using the second set of mappings $M_1$ simulates $M_2$. When $M_2$ reaches a final state using mapping 3, $M_1$ goes to the
erasing state \( q_e \) and using the set of mappings 4, entire pushdown store is erased. If \( w \) is the input accepted by \( M_2 \), we have

\[
(q_0, w, z_0) \xrightarrow{\ast} M_2 (q_f, \varepsilon, \gamma).
\]

This can happen in \( M_1 \) also. \( (q_0, w, Z_0) \xrightarrow{\ast} M_1 (q_f, \varepsilon, \gamma) \).

\( M_1 \) accepts \( w \) as follows:

\[
(q'_0, w, X_0) \vdash (q_0, w, Z_0X_0) \xrightarrow{\ast} (q_f, \varepsilon, \gamma X_0) \xrightarrow{\ast} (q_e, \varepsilon, \varepsilon)
\]

Hence if \( w \) is accepted by \( M_2 \), it will be accepted by \( M_1 \). On the other hand if \( M_1 \) is presented with an input, the first move it can make is using mapping 1 and once
it goes to state $q_e$, it can only erase the pushdown store and has to remain in $q_e$ only. Hence mapping 1 should be used in the beginning and mapping 3 and 4 in the end. Therefore mapping 2 will be used in between and the sequence of moves will be as in the equation 1.

Hence $(q_0, w, Z_0X_0) \overset{*}{\xrightarrow{M_2}} (q_f, \epsilon, \gamma X_0)$ which means $(q_0, w, Z_0) \overset{*}{\xrightarrow{M_2}} (q_f, \epsilon, \gamma)$ and $w$ will be accepted by $M_2$.

(ii) Next we prove that if $L$ is accepted by $M_1$ by empty store, it will be accepted by $M_2$ by final state. Let $M_1 = (K, \Sigma, \Gamma, \delta_1, q_0, Z_0, \phi)$. Then $M_2$ is constructed as follows:

$$M_2 = (K \cup \{q'_0, q_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta_2, q'_0, X_0, \{q_f\})$$
Two more states $q'_0$ and $q_f$ are added to the set of states $K$. $q'_0$ becomes the new initial state and $q_f$ becomes the only final state. One more pushdown symbol $X_0$ is added which becomes the new initial pushdown symbol. The $\delta$ mappings are defined as follows:

1. $\delta_2(q'_0, \epsilon, X_0)$ contains $(q_0, Z_0X_0)$

2. $\delta_2(q, a, Z)$ includes all elements of $\delta_1(q, a, Z)$ for $q \in K$, $a \in \Sigma \cup \{\epsilon\}$, $Z \in \Gamma$

3. $\delta_2(q, \epsilon, X_0)$ contains $(q_f, X_0)$ for each $q \in K$

Mapping 1 makes $M_2$ go to the initial ID of $M_1$ (except for the $X_0$ in the pushdown store). Then using mapping
2, $M_2$ simulates $M_1$. When $M_1$ accepts by emptying the pushdown store, $M_2$ has $X_0$ left on the pushdown store. Using mapping 3, $M_2$ goes to the final state $q_f$. The moves of $M_2$ in accepting an input $w$ can be described as follows:

$$(q'_0, w, X_0) \vdash (q_0, w, Z_0X_0) \vdash^* (q, \epsilon, X_0) \vdash (q_f, \epsilon, X_0)$$

It is not difficult to see that $w$ is accepted by $M_2$ if and only if $w$ is accepted by $M_1$.

It should be noted that $X_0$ is added in the first part for the following reason. $M_2$ may reject an input $w$ by emptying the store and reaching a nonfinal state. If $X_0$
were not there $M_1$ while simulating $M_2$ will empty the store and accept the input $w$. In the second part $X_0$ is added because for $M_2$ to make the last move and reach a final state, a symbol in the pushdown store is required. Thus we have proved the equivalence of acceptance by empty store and acceptance by final state in the case of nondeterministic pushdown automata.

**Remark** The above theorem is not true in the case of deterministic pushdown automata.
Theorem If $L$ is generated by a CFG, then $L$ is accepted by a nondeterministic pushdown automaton by empty store.

Proof Let us assume that $L$ does not contain $\epsilon$ and $L = L(G)$, where $G$ is in Greibach Normal Form. $G = (N, T, P, S)$ where rules in $P$ are of the form $A \rightarrow a\alpha$, $A \in N$, $a \in T$, $\alpha \in N^*$. Then $M$ can be constructed such that $N(M) = L(G)$. $M = (\{q\}, T, N, \delta, q, S, \phi)$ where $\delta$ is defined as follows: If $A \rightarrow a\alpha$ is a rule, $\delta(q, a, A)$ contains $(q, \epsilon)$. $M$ simulates a leftmost derivation in $G$ and the equivalence
\[ L(G) = N(M) \] can be proved using induction. If \( \varepsilon \in L \), then we can have a grammar \( G \) in GNF with an additional rule \( S' \rightarrow \varepsilon \) and \( S' \) will not appear on the right-hand side of any production. In this case, \( M \) can have one \( \varepsilon \)-move defined by \( \delta(q, \varepsilon, S') \) contains \( (q, \varepsilon) \) which will enable it to accept \( \varepsilon \).

**Theorem** If \( L \) is accepted by a PDA, then \( L \) can be generated by a CFG.

**Proof** Let \( L \) be accepted by a PDA by empty store. Construct a CFG \( G = (N, T, P, S) \) as follows:

\[
N = \{[q, Z, p]|q, p \in K, Z \in \Gamma\} \cup \{S\}.
\]

\( P \) is defined as follows:
S \rightarrow [q_0, Z_0, q] \in P \text{ for each } q \text{ in } K. \\
If \delta(q, a, A) \text{ contains } (p, B_1 \ldots B_m) (a \in \Sigma \cup \{\epsilon\}) \text{ is a mapping, then } P \text{ includes rules of the form} \\
[q, A, q_m] \rightarrow a[p, B_1, q_1][q_1, B_2, q_2] \ldots [q_{m-1}, B_m, q_m], \\
q_i \in K, \ 1 \leq i \leq m \\
If \delta(q, a, A) \text{ contains } (p, \epsilon) \text{ then } P \text{ includes } [q, A, p] \rightarrow a \\
Now we show that \( L(G') = N(M)(= L) \). \\
It should be noted that the variables and productions in the grammar \( G \) are defined in such a way that the moves of the PDA are simulated by a leftmost derivation in \( G \). \\
We prove that \\
\[ [q, A, p] \Rightarrow^* x \text{ if and only if } (q, x, A) \vdash^* (p, \epsilon, \epsilon). \]
That is, if the PDA goes from state $q$ to state $p$ after reading $x$ and the stack initially with $A$ on the top ends with $A$ removed from stack (in between the stack can grow and come down). See Figure 1

This is proved by induction on the number of moves of $M$.

(i) If $(q, x, A) \vdash^* (p, \epsilon, \epsilon)$ then $[q, A, p] \Rightarrow x$

**Basis**

If $(q, x, A) \vdash (p, \epsilon, \epsilon) x = a$ or $\epsilon$, where $a \in \Sigma$ and there should be a mapping $\delta(q, x, A)$ contains $(p, \epsilon)$.

In this case by our construction $[q, A, p] \rightarrow x$ is in $P$.

Hence $[q, A, p] \Rightarrow x$. 
Figure 1:
Induction
Suppose the result holds up to \( n - 1 \) steps.
Let \((q, x, A) \vdash^* (p, \varepsilon, \varepsilon)\) in \( n \) steps.
Now we can write \( x = ax'\ a \in \Sigma \cup \{\varepsilon\} \) and the first move is \((q, ax', A) \vdash (q_1, x', B_1 \ldots B_m)\)
This should have come from a mapping \( \delta(q, a, A) \)
contains \((q_1, B_1 \ldots B_m)\) and there is a rule

\[ [q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \ldots [q_m, B_m, q_{m+1}] \text{ in } P. \]  

(2)

The stack contains \( A \) initially and is replaced by \( B_1 \ldots B_m \). Now the string \( x' \) can be written as \( x_1x_2 \ldots x_m \) such that, the PDA completes reading \( x_1 \)
when $B_2$ becomes top of the stack; completes reading $x_2$ when $B_3$ becomes the top of the stack and so on. The situation is is described in the Figure [1]. Therefore $(q_i, x_i, B_i) \vdash^* (q_{i+1}, \epsilon, \epsilon)$ and this happen in less than $n$ steps. So

$$(q_i, B_i, q_{i+1}) \Rightarrow^* x_i \text{ by induction hypothesis.} \quad (3)$$

Putting $q_{m+1} = p$ in (2) we get

$$[q, A, p] \Rightarrow a[q_1, B_1, q_2] \ldots [q_n, B_n, p] \Rightarrow^* ax_1 \ldots x_n = ax' = x \text{ by equation 3) }$$

Therefore $[q, A, p] \Rightarrow^* x$ in $G$. 

(ii) If \([q, A, p] \Rightarrow x\) in \(G\) then \((q, x, A) \vdash^* (p, \varepsilon, \varepsilon)\).

Proof is by induction on the number of steps in the derivation in \(G\).

**Basis**

If \([q, A, p] \Rightarrow x\) then \(x = a\) or \(\varepsilon\) where \(a \in \Sigma\) and \((q, A, p) \rightarrow x\) is a rule in \(P\). This must have come from the mapping \(\delta(q, x, A)\) contains \((p, \varepsilon)\) and hence \((q, x, A) \vdash (p, \varepsilon, \varepsilon)\).

**Induction**

Suppose the hypothesis holds up to \((n - 1)\) steps and suppose \([q, A, p] \Rightarrow x\) in \(n\) steps. The first rule applied in the derivation must be of the form
[q, A, p] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \cdots [q_m, B_m, p] \quad (4)

and \( x \) can be written in the form \( x = a x_1 \ldots x_m \)
such that \([q_i, B_i, q_{i+1}] \Rightarrow x_i\).

This derivation must have taken less than \( n \) steps and
so by the induction hypothesis

\((q_i, x_i, B_i) \vdash^* (q_{i+1}, \epsilon, \epsilon) \quad 1 \leq i \leq m \) and \( q_{m+1} = p \).

(5)

4 must have come from a mapping \( \delta(q, a, A) \) contains

\((q, B_1 \ldots B_m)\)

Therefore
\[(q, ax_1 \ldots x_m, A) \vdash (q_1, x_1 \ldots x_m, B_1 \ldots B_m)\]
\[\vdash^* (q_2, x_2 \ldots x_m, B_2 \ldots B_m)\]
\[\vdash^* (q_3, x_3 \ldots x_m, B_3 \ldots B_m)\]
\[\vdots\]
\[\vdash^* (q_{m-1}, x_{m-1}x_m, B_{m-1}B_m)\]
\[\vdash^* (q_m, x_m, B_m)\]
\[\vdash^* (p, \epsilon, \epsilon)\]
Hence \((q, x, A) \vdash^* (p, \epsilon, \epsilon)\). Having proved that \((q, x, A) \vdash^* (p, \epsilon, \epsilon)\) if and only if \([q, A, p] \Rightarrow^* x\), we can easily see that \(S \Rightarrow [q_0, Z_0, q] \Rightarrow^* w\) if and only if \((q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)\).

This means \(w\) is generated by \(G\) if and only if \(w\) is accepted by \(M\) by empty store. Hence \(L(G) = N(M)\).

Let us illustrate the construction with an example.
Example

Construct a CFG to generate $N(M)$ where

$$M = (\{p, q\}, \{0, 1\}, \{X, Z_0\}, \delta, q, Z_0, \phi)$$

where $\delta$ is defined as follows:

1. $\delta(q, 1, Z_0) = \{(q, XZ_0)\}$
2. $\delta(q, 1, X) = \{(q, XX)\}$
3. $\delta(q, 0, X) = \{(p, X)\}$
4. $\delta(q, \epsilon, Z_0) = \{(q, \epsilon)\}$
5. $\delta(p, 1, X) = \{(p, \epsilon)\}$
6. $\delta(p, 0, Z_0) = \{(q, Z_0)\}$
It can be seen that

\[ N(M) = \{1^n01^n0\}^*, \quad n \geq 1. \]

The machine while reading \(1^n\) adds \(X\)’s to the stack and when it reads a 0, change to state \(p\). In state \(p\) it reads \(1^n\) again removing the \(X\)’s from the stack. When it reads a 0, it goes to \(q\) keeping \(Z_0\) on the stack. It can remove \(Z_0\) by using mapping 4 or repeat the above process several times. Initially also \(Z_0\) can be removed using mapping 4, without reading any input. Hence \(\epsilon\) will also be accepted.

\[ G = (N, T, P, S) \] is constructed as follows:

\[ T = \Sigma \]

\[ N = \{[q, Z_0, q], [q, X, q], [q, Z_0, p], [q, X, p], [p, Z_0, q], [p, X, q], [p, Z_0, p], [p, X, p]\} \cup \{S\} \]
Initial rules are

\[ r_1. \quad S \rightarrow [q, Z_0, q] \]
\[ r_2. \quad S \rightarrow [q, Z_0, p] \]

Next we write the rules for the mappings.

Corresponding to mapping 1, we have the rules

\[ r_3. \quad [q, Z_0, q] \rightarrow 1[q, X, q][q, Z_0, q] \]
\[ r_4. \quad [q, Z_0, q] \rightarrow 1[q, X, p][p, Z_0, q] \]
\[ r_5. \quad [q, Z_0, p] \rightarrow 1[q, X, q][q, Z_0, p] \]
\[ r_6. \quad [q, Z_0, p] \rightarrow 1[q, X, p][p, Z_0, p] \]

Corresponding to mapping 2, we have the rules

\[ r_7. \quad [q, X, q] \rightarrow 1[q, X, q][q, X, q] \]
\[ r_8. \quad [q, X, q] \rightarrow 1[q, X, p][p, X, q] \]
\[ r_9. \quad [q, X, p] \rightarrow 1[q, X, q][q, X, p] \]
\[ r_{10}. \quad [q, X, p] \rightarrow 1[q, X, p][p, X, p] \]
Corresponding to mapping 3, we have the rules
\[ r_{11}. \; [q, X, q] \rightarrow 0[p, X, q] \]
\[ r_{12}. \; [q, X, p] \rightarrow 0[p, X, p] \]
Corresponding to mapping 4 we have the rule
\[ r_{13}. \; [q, Z_0, q] \rightarrow \epsilon \]
Corresponding to mapping 5 we have the rule
\[ r_{14}. \; [p, X, p] \rightarrow 1 \]
Corresponding to mapping 6 we have the rules
\[ r_{15}. \; [p, Z_0, q] \rightarrow 0[q, Z_0, q] \]
\[ r_{16}. \; [p, Z_0, p] \rightarrow 0[q, Z_0, p] \]
So we have ended up with 16 rules. Let us see whether we can remove some useless nonterminals and rules here.
There is no rule with \([p, X, q]\) on the left hand side. So
rules involving it can be removed i.e., \( r_8, r_{11} \). Once \( r_8 \) and \( r_{11} \) are removed, the only rule with \([q, X, q]\) on the left hand side is \( r_7 \) which will create more \([q, X, q]\) whenever applied and the derivation will not terminate. So rules involving \([q, X, q]\) can be removed. i.e., \( r_3, r_5, r_7, r_9 \). Now we are left with rules \( r_1, r_2, r_4, r_6, r_{10}, r_{12}, r_{13}, r_{14}, r_{15}, r_{16} \). If you start with \( r_2, r_6 \) can be applied. \([q, Z_0, p]\) will introduce \([p, Z_0, p]\) in the sentential form. Then \( r_{16} \) can be applied which will introduce \([q, Z_0, p]\) and the derivation will not terminate. Hence \([q, Z_0, p]\) and rules involving it can be removed. i.e., rules \( r_2, r_6, r_{16} \) can be removed. So we end up with rules \( r_1, r_4, r_{10}, r_{12}, r_{13}, r_{14}, r_{15} \). Using nonterminals
contd.

\begin{align*}
A & \text{ for } [q, Z_0, q] \\
B & \text{ for } [q, X, p] \\
C & \text{ for } [p, Z_0, q] \\
D & \text{ for } [p, X, p]
\end{align*}
the rules can be written as

\[
S \rightarrow A \\
A \rightarrow 1BC \\
B \rightarrow 1BD \\
B \rightarrow 0D \\
A \rightarrow \epsilon \\
D \rightarrow 1 \\
C \rightarrow 0A
\]

It can be easily checked that this grammar generates \( \{1^n01^n0\}^* \).