6 Pushdown Automata

We will now consider a new notion of automata *Pushdown Automata* (PDA). PDAs are finite automata with a stack, i.e. a data structure which can be used to store an arbitrary number of symbols (hence PDAs have an infinite set of states) but which can be only accessed in a *last-in-first-out* (LIFO) fashion. The languages which can be recognized by PDA are precisely the context free languages.

6.1 What is a Pushdown Automaton?

A Pushdown Automaton $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is given by the following data

- A finite set Q of states,
- A finite set Σ of symbols (the alphabet),
- A finite set Γ of stack symbols,
- A transition function
$$\delta \in Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}_{\text{fin}}(Q \times \Gamma^*)$$
Here $\mathcal{P}_{\text{fin}}(A)$ are the finite subsets of a set, i.e. this can be defined as
$$\mathcal{P}_{\text{fin}}(A) = \{X \mid X \subseteq A \wedge X \text{is finite.}\}$$
- An initial state $q_0 \in Q$,
- An initial stack symbol $Z_0 \in \Gamma$,
- A set of final states $F \subseteq Q$.

As an example we consider a PDA P which recognizes the language of even length palindromes over $\Sigma = \{0,1\}$ — $L = \{ww^R \mid w \in \{0,1\}^*\}$. Intuitively, this PDA pushes the input symbols on the stack until it *guesses* that it is in the middle and then it compares the input with what is on the stack, popping of symbols from the stack as it goes. If it reaches the end of the input precisely at the time when the stack is empty, it accepts.

$$P_0 = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1,\#\}, \delta, q_0, \#, \{q_2\})$$

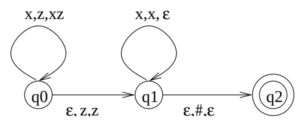
where δ is given by the following equations:

$$\begin{array}{lcl}
\delta(q_0, 0, \#) & = & \{(q_0, 0\#)\} \\
\delta(q_0, 1, \#) & = & \{(q_0, 1\#)\} \\
\delta(q_0, 0, 0) & = & \{(q_0, 00)\} \\
\delta(q_0, 1, 0) & = & \{(q_0, 10)\} \\
\delta(q_0, 0, 1) & = & \{(q_0, 01)\} \\
\delta(q_0, 1, 1) & = & \{(q_0, 11)\} \\
\delta(q_0, \epsilon, \#) & = & \{(q_1, \#)\} \\
\delta(q_0, \epsilon, 0) & = & \{(q_1, 0)\} \\
\delta(q_0, \epsilon, 1) & = & \{(q_1, 1)\} \\
\delta(q_1, 0, 0) & = & \{(q_1, \epsilon)\} \\
\delta(q_1, 1, 1) & = & \{(q_1, \epsilon)\} \\
\delta(q_1, \epsilon, \#) & = & \{(q_2, \epsilon)\} \\
\delta(q, w, z) & = & \{\} \qquad \text{everywhere else}
\end{array}$$

To save space we may abbreviate this by writing:

$$\begin{array}{lcl}
\delta(q_0, x, z) & = & \{(q_0, xz)\} \\
\delta(q_0, \epsilon, z) & = & \{(q_1, z)\} \\
\delta(q_1, x, x) & = & \{(q_1, \epsilon)\} \\
\delta(q_1, \epsilon, \#) & = & \{(q_2, \epsilon)\} \\
\delta(q, x, z) & = & \{\} \qquad \text{everywhere else}
\end{array}$$

where $q \in Q, x \in \Sigma, z \in \Gamma$. We obtain the previous table by expanding all the possibilities for q, x, z.
We draw the transition diagram of P by labelling each transition with a triple x, Z, γ with $x \in \Sigma, Z \in \Gamma, \gamma \in \Gamma^*$:



6.2 How does a PDA work?

At any time the state of the computation of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is given by:

- the state $q \in Q$ the PDA is in,
- the input string $w \in \Sigma^*$ which still has to be processed,
- the contents of the stack $\gamma \in \Gamma^*$.

Such a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$ is called an Instantanous Description (ID).
We define a relation $\vdash_P \subseteq \text{ID} \times \text{ID}$ between IDs which describes how the PDA can change from one ID to the next one. Since PDAs in general are nondeterministic this is a relation (not a function), i.e. there may be more than one possibility.
There are two possibilities for $\vdash_P$:

1. $(q, xw, z\gamma) \vdash_P (q', w, \alpha\gamma)$ if $(q', \alpha) \in \delta(q, x, z)$
2. $(q, w, z\gamma) \vdash_P (q', w, \alpha\gamma)$ if $(q', \alpha) \in \delta(q, \epsilon, z)$

In the first case the PDA reads an input symbol and consults the transition function δ to calculate a possible new state q' and a sequence of stack symbols α which replace the currend symbol on the top z.
In the second case the PDA ignores the input and silently moves into a new state and modifies the stack as above. The input is unchanged.
Consider the word 0110 — what are possible sequences of IDs for P_0 starting with $(q_0, 0110, \#)$?

$$\begin{array}{rll}
(q_0, 0110, \#) & \vdash_{P_0} (q_0, 110, 0\#) & \text{1. with } (q_0, 0\#) \in \delta(q_0, 0, \#) \\
& \vdash_{P_0} (q_0, 10, 10\#) & \text{1. with } (q_0, 10) \in \delta(q_0, 1, 0) \\
& \vdash_{P_0} (q_1, 10, 10\#) & \text{2. with } (q_1, 1) \in \delta(q_0, \epsilon, 1) \\
& \vdash_{P_0} (q_1, 0, 0\#) & \text{1. with } (q_1, \epsilon) \in \delta(q_1, 1, 1) \\
& \vdash_{P_0} (q_1, \epsilon, \#) & \text{1. with } (q_1, \epsilon) \in \delta(q_1, 0, 0) \\
& \vdash_{P_0} (q_2, \epsilon, \epsilon) & \text{2. with } (q_2, \epsilon) \in \delta(q_1, \epsilon, \#)
\end{array}$$

We write $(q, w, \gamma) \vdash_P^* (q', w', \gamma)$ if the PDA can move from (q, w, γ) to (q', w', γ') in a (possibly empty) sequence of moves. Above we have shown that

$$(q_0, 0110, \#) \vdash_{P_0}^* (q_2, \epsilon, \epsilon).$$

However, this is not the only possible sequence of IDs for this input. E.g. the PDA may just guess the middle wrong:

$$\begin{array}{lll} (q_0, 0110, \#) & \vdash_{P_0} (q_0, 110, 0\#) & 1.\ \text{with } (q_0, 0\#) \in \delta(q_0, 0, \#) \\ & \vdash_{P_0} (q_1, 110, 0\#) & 2.\ \text{with } (q_1, 0) \in \delta(q_0, \epsilon, 0) \end{array}$$

We have shown $(q_0, 0110, \#) \vdash_{P_0}^* (q_1, 110, 0\#)$. Here the PDA gets stuck there is no state after $(q_1, 110, 0\#)$.
If we start with a word which is not in the language L (like 0011) then the automaton will always get stuck before reaching a final state.

6.3 The language of a PDA

There are two ways to define the language of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ $(L(P) \subseteq \Sigma^*)$ because there are two notions of acceptance:

Acceptance by final state

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \gamma) \wedge q \in F\}$$

That is the PDA accepts the word w if there is any sequence of IDs starting from (q_0, w, Z_0) and leading to (q, ϵ, γ), where $q \in F$ is one of the final states. Here it doesn't play a role what the contents of the stack are at the end.

In our example the PDA P_0 would accept 0110 because $(q_0, 0110, \#) \vdash_{P_0}^* (q_2, \epsilon, \epsilon)$ and $q_2 \in F$. Hence we conclude $0110 \in L(P_0)$.

On the other hand since there is no successful sequence of IDs starting with $(q_0, 0011, \#)$ we know that $0011 \notin L(P_0)$.

Acceptance by empty stack

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}$$

That is the PDA accepts the word w if there is any sequence of IDs starting from (q_0, w, Z_0) and leading to (q, ϵ, ϵ), in this case the final state plays no role.

If we specify a PDA for acceptance by empty stack we will leave out the set of final states F and just use $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$.

Our example automaton P_0 also works if we leave out F and use acceptance by empty stack.

We can always turn a PDA which use one acceptance method into one which uses the other. Hence, both acceptance criteria specify the same class of languages.

6.4 Deterministic PDAs

We have introduced PDAs as nondeterministic machines which may have several alternatives how to continue. We now define Deterministic Pushdown Automata (DPDA) as those which never have a choice.
To be precise we say that a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is deterministic (is a DPDA) iff

$$|\delta(q, x, z)| + |\delta(q, \epsilon, z)| \leq 1$$

Remember, that $|X|$ stands for the number of elements in a finite set X.
That is: a DPDA may get stuck but it has never any choice.
In our example the automaton P_0 is not deterministic, e.g. we have $\delta(q_0, 0, \#) = \{(q_0, 0\#)\}$ and $\delta(q_0, \epsilon, \#) = \{(q_1, \#)\}$ and hence $|\delta(q_0, 0, \#)| + |\delta(q_0, \epsilon, \#)| = 2$.
Unlike the situation for finite automata, there is in general no way to translate a nondeterministic PDA into a deterministic one. Indeed, there is no DPDA which recognizes the language L ! **Nondeterministic PDAs are more powerful than deterministic PDAs.**
However, we can define a similar language L' over $\Sigma = \{0, 1, \$\}$ which can be recognized by a deterministic PDA:

$$L' = \{w\$w^R \mid w \in \{0, 1\}^*\}$$

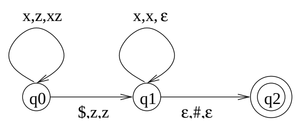
That is L' contains palindroms with a marker \$ in the middle, e.g. $01\$10 \in L'$.
We define a DPDA P_1 for L':

$$P_1 = (\{q_0, q_1, q_2\}, \{0, 1, \$\}, \{0, 1, \#\}, \delta', q_0, \#, \{q_2\})$$

where δ is given by:

$$\begin{array}{lcl} \delta(q_0, x, z) & = & \{(q_0, xz) \qquad x \in \{0, 1\}\} \\ \delta(q_0, \$, z) & = & \{(q_1, z)\} \\ \delta(q_1, x, x) & = & \{(q_1, \epsilon)\} \\ \delta(q_1, \epsilon, \#) & = & \{(q_2, \epsilon)\} \\ \delta(q, x, z) & = & \{\} \qquad \text{everywhere else} \end{array}$$

Here is its transition graph:



We can check that this automaton is deterministic. In particular the 3rd and 4th line cannot overlap because # is not an input symbol.
Different to PDAs in general the two acceptance methods are not equivalent for DPDAs — acceptance by final state makes it possible to define a bigger class of langauges. Hence, we shall always use acceptance by final state for DPDAs.

6.5 Context free grammars and push-down-automata

Theorem 6.1 *For a language $L \subseteq \Sigma^*$ the following is equivalent:*

1. *L is given by a CFG G, $L = L(G)$.*
2. *L is the language of a PDA P, $L = L(P)$.*

To summarize: Context Free Languages (CFLs) can be described by a Context Free Grammar (CFG) and can be processed by a pushdown automaton.
We will he only show how to construct a PDA from a grammar - the other direction is shown in [HMU01] (6.3.2, pp. 241).
Given a CFG $G = (V, \Sigma, S, P)$, we define a PDA

$$P(G) = (\{q_0\}, \Sigma, V \cup \Sigma, \delta, q_0, S)$$

where δ is defined as follows:

$$\delta(q_0, \epsilon, A) = \{(q_0, \alpha) \mid A \to \alpha \in P\}$$

for all $A \in V$.

$$\delta(q_0, a, a) = \{(q_0, \epsilon)\}$$

for all $a \in \Sigma$.

We haven't given a set of final states because we use acceptance by empty stack.
Yes, we use only one state!
Take as an example $G = (\{E, T, F\}, \{(,), a, +, *\}, E, P)$ where

$$\begin{aligned} P = \{E &\to T \mid E + T \\ T &\to F \mid T * F \\ F &\to a \mid (E) \end{aligned}$$

we define

$$P(G) = (\{q_0\}, \{(,), a, +, *\}, \{E, T, F, (,), a, +, *\}, \delta, q_0, E)$$

with

$$\begin{aligned} \delta(q_0, \epsilon, E) &= \{(q_0, T), (q_0, E + T)\} \\ \delta(q_0, \epsilon, T) &= \{(q_0, F), (q_0, T * F)\} \\ \delta(q_0, \epsilon, F) &= \{(q_0, a), (q_0, \texttt{(}E\texttt{)})\} \\ \delta(q_0, \texttt{(}, \texttt{(}) &= \{(q_0, \epsilon)\} \\ \delta(q_0, \texttt{)}, \texttt{)}) &= \{(q_0, \epsilon)\} \\ \delta(q_0, \texttt{a}, \texttt{a}) &= \{(q_0, \epsilon)\} \\ \delta(q_0, \texttt{+}, \texttt{+}) &= \{(q_0, \epsilon)\} \\ \delta(q_0, \texttt{*}, \texttt{*}) &= \{(q_0, \epsilon)\} \end{aligned}$$

How does the $P(G)$ accept `a + (a*a)`?

$$\begin{aligned} (q_0, \texttt{a + (a*a)}, E) &\vdash (q_0, \texttt{a + (a*a)}, E\texttt{+}T) \\ &\vdash (q_0, \texttt{a + (a*a)}, T\texttt{+}T) \\ &\vdash (q_0, \texttt{a + (a*a)}, F\texttt{+}T) \\ &\vdash (q_0, \texttt{a + (a*a)}, a\texttt{+}T) \\ &\vdash (q_0, \texttt{+ (a*a)}, \texttt{+}T) \\ &\vdash (q_0, \texttt{ (a*a)}, T) \\ &\vdash (q_0, \texttt{ (a*a)}, F) \\ &\vdash (q_0, \texttt{ (a*a)}, \texttt{(}E\texttt{)}) \\ &\vdash (q_0, \texttt{ a*a)}, E\texttt{)}) \\ &\vdash (q_0, \texttt{ a*a)}, T\texttt{)}) \\ &\vdash (q_0, \texttt{ a*a)}, T * F\texttt{)}) \\ &\vdash (q_0, \texttt{ a*a)}, F * F\texttt{)}) \\ &\vdash (q_0, \texttt{ a*a)}, a * F\texttt{)}) \\ &\vdash (q_0, \texttt{ *a)}, \texttt{*}F\texttt{)}) \\ &\vdash (q_0, \texttt{ a)}, F\texttt{)}) \\ &\vdash (q_0, \texttt{ a)}, \texttt{a)}) \\ &\vdash (q_0, \texttt{)}, \texttt{)}) \\ &\vdash (q_0, \epsilon, \epsilon) \end{aligned}$$

Hence `a + (a*a)` $\in L(P(G))$.
This example hopefully already illustrates the general idea:

$$\begin{aligned} w &\in L(G) \\ &\iff \\ S &\Rightarrow \ldots \Rightarrow w \\ &\iff \\ (q_0, w, S) &\vdash \cdots \vdash (q_0, \epsilon, \epsilon) \\ &\iff w \in L(P(G)) \end{aligned}$$

The automaton we have constructed is very non-deterministic: Whenever we have a choice between different rules the automaton may silently choose one of the alternative.