II.6.1. Push Down Automata

Motivation

- There are context free languages which are not regular, so they can’t be recognised by a NFA.
- The problem is that when parsing a language such as 
  \( L = \{ww^R \mid w \in \{a, b\}^*\} \), when checking the second half of the word, one needs to know the first half of the word in reverse order.
- An NFA has only finitely many states, and therefore only finite memory.
- We can repair the problem by adding a stack to memory, which allows us to record in case of \( L \) the first half of the word.
- The resulting machine will be called a \textbf{push-down automaton} (PDA).

Architecture of a PDA

- A PDA consists of
  - An input type, containing the input word.
  - The input tape will be read only from left to right.
  - A finite state \( q \).
  - A stack.
II.6.1. Push Down Automata

Instructions of a PDA

- A PDA has two kinds of instruction:
  - An empty move instructions which depending on the state, the top symbol of the stack, without looking at the next symbol on the input tape chooses a new state, a possibly empty sequence of stack symbols to replace the top symbol on the stack, and keeps the pointer to the input tape.
  - An ordinary instruction, which depending on the state, the top symbol of the stack, and the next input symbol on the tape chooses a new state, a possibly empty sequence of stack symbols to replace the top symbol on the stack, and moves the pointer on the input tape forward.

Execution of a PDA

- The automaton starts with the head on the left most symbol of the input tape, with some initial state and a start symbol on the stack.
- It will then do the following non-deterministically:
  - If the stack is empty, it will get stuck.
  - Otherwise, depending on the state $s$ and the top stack symbol $x$ it will do one of the following:
    - If $s$ and $x$ match an empty move transition then replace the top stack symbol by the new symbols to be put on the stack, switch to a new state and keep the position on the input tape as before.
    - If the next symbol on the input tape is $y$ and $s, x, y$ match an ordinary transition then replace the top stack symbol by the new symbols to be put on the stack, switch to a new state and keep the position on the input tape as before.
- If none of the above instructions matches, the PDA gets stuck.
- The PDA will stop if
  - it gets stuck (because of empty stack or having no operation possible) while still reading the word,
  - it wants to read a letter and has reached the end of the word.
- There are two kinds of languages defined from a PDA:
  - The language accepted by a final state.
    A word is accepted by final state, if the PDA reads the complete word and reaches a state which is final (accepting).
  - The language accepted by empty stack.
    A word is accepted by empty stack, if the PDA reads the complete word and then obtains empty stack.
II.6.1. Push Down Automata

Acceptance by Final State vs Empty State

- Clearly, acceptance by final state is natural generalisation of a NFA.
- The standard LL($k$), LR($k$), LALR parsing algorithms use a deterministic PDA accepting by final state.
- CFG correspond in a natural way to non-deterministic PDA which accept by empty stack.
- We will see that the languages accepted by PDAs by empty stack and accepted by PDAs by final state are equivalent.
- Therefore PDAs accepting by empty state correspond to intermediate machines for proving the equivalence of context free languages and languages accepted by final state by a PDA.

Convention Regarding Stacks

- When writing $w$ for the stack, the top element will be the left most element of $w$.

Definition of PDA

A push down automaton (PDA) $P = (T, Q, \Gamma, q_0, Z_0, F, \delta)$ consists of:

- an input alphabet $T$;
- a finite set of states $Q$;
- a stack alphabet $\Gamma$;
- a start state $q_0 \in Q$;
- an start stack symbol $Z_0 \in \Gamma$;
- a set of accepting or final states $F \subseteq Q$;
- a transition relation $\delta \subseteq Q \times \Gamma \times (T \cup \{\epsilon\}) \times Q \times \Gamma^*$, where we write $(q, Z) \xrightarrow{a} (q', w)$ for $(q, Z, a, q', w) \in \delta$. 
Interpretation of the Components

- The input alphabet $T$ will be the alphabet on the tape; so the PDA will look at elements of $T^*$ and check whether it accepts them or not;
- The stack will consist of elements of the stack alphabet $\Gamma$.
- The PDA will start in start state $q_0$ with stack consisting of the start stack symbol $Z_0$.
- $F$ will be the set of final states in case we consider the language accepted by final state.
- \((q, Z) \xrightarrow{a} (q', w)\) means that the PDA can, when in state $q$, and if the next tape symbol is $a$, move on the tape once to the right, change to state $q'$, and replace the top stack symbol $Z$ by $w$.
- \((q, Z) \xrightarrow{\epsilon} (q', w)\) means that the PDA can, when in state $q$ and having top symbol $Z$ on the stack, make a move without looking at the next letter, and switch to state $q'$, and replace the top symbol on the stack by $w$.

Example

- We define a PDA which accepts the language
  $$L := \{ww^R \mid w \in \{a, b\}^*\}$$
- We will use the stack in order to record the first half of the word, so need the stack symbols $a, b$.
- In addition we need the bottom symbol of the stack $Z_0$. After having finished parsing the word, if we read this symbol the PDA will move to an accepting state.
- There are 3 states:
  - Initial state $q_0$.
    - In this state the PDA will read the first half of the word, and push it onto the stack.
  - An intermediate state $q_1$.
    - When reading the second half of the word, the PDA will be in this state and pop symbols from the stack.
  - A final accepting state.
- So our PDA will be
  $$\left(\{a, b\}, \{q_0, q_1, q_2\}, \{Z_0, a, b\}, q_0, Z_0, \{q_2\}, \delta\right)$$

Transitions

We have the following transitions in $\delta$:

- Initially the PDA is in state $q_0$, sees stack symbol $Z_0$, and then reads the first symbol on the tape and pushes it on the stack:
  $$\begin{align*}
  (q_0, Z_0) & \xrightarrow{a} (q_0, aZ_0) \\
  (q_0, Z_0) & \xrightarrow{b} (q_0, bZ_0)
  \end{align*}$$
- When reading future letters, the PDA will, when in state $q_0$, push them on the stack:
  $$\begin{align*}
  (q_0, a) & \xrightarrow{a} (q_0, aa) \\
  (q_0, b) & \xrightarrow{a} (q_0, ab) \\
  (q_0, a) & \xrightarrow{b} (q_0, ba) \\
  (q_0, b) & \xrightarrow{b} (q_0, bb)
  \end{align*}$$
The PDA will guess when it has reached the middle of the stack, and then switch silently (making a $\epsilon$-transition, i.e. a transition without reading the next symbol) to state $q_1$ without modifying the stack. This can happen at the beginning (without having read a symbol, so stack is $Z_0$), or when it has already pushed some symbol on the stack, so the stack symbol can be any of $a, b, Z_0$:

$$( q_0, Z_0 ) \xrightarrow{\epsilon} ( q_1, Z_0 )$$
$$( q_0, a ) \xrightarrow{\epsilon} ( q_1, a )$$
$$( q_0, b ) \xrightarrow{\epsilon} ( q_1, b )$$

When in state $q_1$, the PDA compares whether the letters it reads are identical to those it read in the first part, so whether the letter it reads is identical to the letter on the stack. If yes it empties the stack.

$$\begin{align*}
(q_1, a) & \xrightarrow{a} (q_1, \epsilon) \\
(q_1, b) & \xrightarrow{b} (q_1, \epsilon)
\end{align*}$$

When the stack is emptied, while in state $q_1$, the PDA can move to the accepting state $q_2$. It will as well empty the stack.

If there are more letters to be read, the PDA will get stuck in that state, because there will be no transitions in this state.

$$( q_1, Z_0 ) \xrightarrow{\epsilon} ( q_2, \epsilon )$$

Presentation of the PDA by Table

<table>
<thead>
<tr>
<th>PDA</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminals</td>
<td>$a, b$</td>
</tr>
<tr>
<td>states</td>
<td>$q_0, q_1, q_2$</td>
</tr>
<tr>
<td>stack alphab.</td>
<td>$Z_0, a, b$</td>
</tr>
<tr>
<td>start state</td>
<td>$q_0$</td>
</tr>
<tr>
<td>start stack</td>
<td>$Z_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_2$</td>
</tr>
<tr>
<td>transitions</td>
<td></td>
</tr>
<tr>
<td>$q_0, Z_0$</td>
<td>$\xrightarrow{a} (q_0, aZ_0)$</td>
</tr>
<tr>
<td>$q_0, a$</td>
<td>$\xrightarrow{a} (q_0, a)$</td>
</tr>
<tr>
<td>$q_0, b$</td>
<td>$\xrightarrow{b} (q_0, b)$</td>
</tr>
<tr>
<td>$q_1, Z_0$</td>
<td>$\xrightarrow{\epsilon} (q_1, Z_0)$</td>
</tr>
<tr>
<td>$q_1, a$</td>
<td>$\xrightarrow{a} (q_1, a)$</td>
</tr>
<tr>
<td>$q_1, b$</td>
<td>$\xrightarrow{b} (q_1, b)$</td>
</tr>
<tr>
<td>$q_1, \epsilon$</td>
<td>$\xrightarrow{\epsilon} (q_1, \epsilon)$</td>
</tr>
</tbody>
</table>
II.6.1. Push Down Automata

Graphical Presentation

We can present a PDA as well graphically by a transition diagram with
- states, start state, final states represented as before,
- transitions labelled with an expression
  \[ a, Z/w \]
  where \( a \in T \), \( Z \in \Gamma \), \( w \in \Gamma^* \),
where a transition from state \( q \) to \( q' \) labelled by \( a, Z/w \) stands for
\[ (q, Z) \xrightarrow{a} (q', w) \]
- The alphabet \( T \) is implicitly given by the elements of the alphabet appearing in the transitions.

Example in Graphical Notation

Configuration of a PDA

- The complete state of a PDA is given by its stack and an element of \( Q \).
- We call this the configuration of a PDA.

Definition

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \) be a PDA.
A configuration of \( P \) is given by an element \((q, z)\) where \( q \in Q \), \( z \in \Gamma^* \).

Transition of Configurations

- We extend the relation \((q, Z) \xrightarrow{a} (q', z)\) for \( q, q' \in Q \), \( Z \in \Gamma \), \( z \in \Gamma^* \), \( a \in T \cup \{\epsilon\} \) to a one step relation
  \[ (q, z) \xrightarrow{a}^1 (q', z') \]
  between configurations \((q, z), (q', z')\) and \( w \in T^* \) expressing that:
  - If the PDA has configuration \((q, z)\), then it can make a one step movement using \( a \) to configuration \((q', z')\).
- Furthermore we define an \( n \)-step transition relation
  \[ (q, z) \xrightarrow{w}^n (q', z') \]
  and a transition relation expressing that we can move from one configuration to another in arbitrarily many steps
  \[ (q, z) \xrightarrow{w}^* (q', z') \]
One Step Transition of Configuration

**Definition**

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \) be a PDA. For \( q, q' \in Q, Z \in \Gamma, z \in \Gamma^* \), define the one step transition relation between configurations \((q, z)\) and \((q', z')\) and \(a \in T \cup \{\epsilon\}\), written as \((q, z) \xrightarrow{a} (q', z')\) by:

\[
(q, z) \xrightarrow{a} (q', z') \\
\text{If } (q, z) \xrightarrow{} (q', z) , \text{ then } \forall z' \in Z^*(q, Zz') \xrightarrow{a} (q', zz')
\]

Since \( \xrightarrow{a} \) extends \( \xrightarrow{} \), we often write \((q, z) \xrightarrow{a} (q', z')\) instead of \((q, z) \xrightarrow{a} (q', z')\).

**n-Step Transition of Configuration**

**Definition**

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \) be a PDA. We define for \( n \in \mathbb{N} \) the n-step transition relation between configurations \((q, z)\), \((q', z')\) and \(w \in T^*\), written as \((q, z) \xrightarrow{w}^n (q', z')\) as follows:

\[
(q, z) \xrightarrow{\epsilon}^0 (q, z) \\
(q, z) \xrightarrow{a}^1 (q', z') \\
(q, z) \xrightarrow{w}^n (q'', z'') \\
\text{then } (q, z) \xrightarrow{aw}^{n+1} (q'', z'')
\]

---

**Transition of Configuration**

**Definition**

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \) be a PDA. We define the transition relation between configurations \((q, z)\) and \((q', z')\) and \(w \in T^*\), written as \((q, z) \xrightarrow{w} (q', z')\) as follows:

\[
(q, z) \xrightarrow{w}^* (q', z') \iff \exists n \in \mathbb{N}. (q, z) \xrightarrow{w}^n (q', z')
\]

---

**Language Accepted by a PDA**

**Definition**

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \) be a PDA.

- The **language accepted by final state of** \( P \), denoted by \( L_{\text{final}}(P) \), is defined as
  \[
  L_{\text{final}}(P) := \{ w \in T^* \mid (q_0, Z_0) \xrightarrow{w}^* (q, z) \text{ for some } q \in F, z \in \Gamma^* \}
  \]

A **final-state PDA** is a PDA \( P \) for which we define its language to be \( L(P) = L_{\text{final}}(P) \).
II.6.1. Push Down Automata

Language Accepted by a PDA

Definition (Cont)

- The **language accepted by empty stack of** $P$, denoted by $L_{\text{empty}}(P)$, is defined as

$$L_{\text{empty}}(P) := \{ w \in T^* \mid (q_0, Z_0) \xrightarrow{w}^* (q, \epsilon) \text{ for some } q \in Q \}$$

An **empty-stack PDA** is a PDA $P$ for which we define its language to be $L(P) = L_{\text{empty}}(P)$.

II.6.2. Equivalence of Final-State and Empty-Stack-PDAs

Equivalence Acceptance by Final/Empty Stack

Theorem

Let $L$ be a language. The following are equivalent:

1. $L = L(P)$ for a final state PDA $P$.
2. $L = L(P)$ for an empty stack PDA $P$.

Proof Idea of (1) $\Rightarrow$ (2)

Let $P = (T, Q, \Gamma, q_0, Z_0, F, \delta)$. The idea for constructing an empty stack PDA $P'$ is as follows:

- $P'$ operates essentially as $P$.
- If $P$ reaches an accepting state, then $P'$ can switch to a special state. In that state it empties using $\epsilon$-transition the stack and and therefore accepts the string.
- Full details can be found in Additional Material.
II.6.2. Equivalence of Final-State and Empty-Stack-PDAs

Proof of (2) ⇒ (1)

Let \( P = (T, Q, \Gamma, q_0, Z_0, F, \delta) \).
The idea for constructing a final state PDA \( P' \) is as follows:

- \( P' \) keeps a special stack symbol \( Z'_0 \) at the bottom of the stack.
- It operates as \( P \), until it observes that the top stack symbol is \( Z'_0 \).
- This indicates that \( P \) would have reached an empty stack and therefore accepted the string.
- Therefore \( P' \) moves into a special accepting state \( q_{\text{final}} \) and terminates.
- Full details can be found in the Additional Material.

II.6.3. Equivalence of CFG and PDA

Equivalence Acceptance by Final/Empty Stack

We are going to show that the context free languages are exactly the languages which can be recognised by a (non-deterministic) empty stack PDA.
Since empty stack PDA are final state PDA recognise the same languages, the context free languages are exactly the languages which can be recognised by a PDA.

Theorem

Let \( L \) be a language. The following are equivalent:

1. \( L = L(G) \) for a CFG \( G \).
2. \( L = L(P) \) for an empty stack PDA \( P \).
Corollary

Because empty stack and final state PDAs are equivalent, from the theorem follows immediately the following corollary:

**Corollary**

Let \( L \) be a language. The following are equivalent:

1. \( L = L(G) \) for a CFG \( G \).
2. \( L = L(P) \) for an empty stack PDA \( P \).
3. \( L = L(P) \) for a final state PDA \( P \).

Proof of the Theorem

We will only show \((1) \Rightarrow (2)\), \((2) \Rightarrow (1)\) is quite sophisticated.

Proof of \((1) \Rightarrow (2)\)

Assume \( G = (T, N, S, P) \) is a CFG. We need to construct a PDA which simulates \( G \).

There are two ways of constructing such a PDA, one follows the LL-parsing method, one uses the LR-parsing method.

**LL-parsing** stands for left-to-right parsing based on a leftmost derivation.

- It constructs a leftmost derivation top down, and is therefore an example of a top down parser.
- We take as example the grammar with the rules
  
  \[
  \begin{align*}
  S & \rightarrow AC \\
  A & \rightarrow aAb \\
  A & \rightarrow ab \\
  C & \rightarrow cCd \\
  C & \rightarrow cd
  \end{align*}
  \]

- We consider a left-most derivation
  
  \[
  S \Rightarrow AC \Rightarrow aAbC \Rightarrow aabbC \Rightarrow aabbcCd \Rightarrow aabbcddd
  \]

- LL-parsing will result in a PDA with a single state. The configuration can be given by its stack.

We will introduce in the main slides LL-parsing and in the Additional material LR-parsing.

Deterministic versions of them are the basis for many parsers.

In the additional material you will find the full proof of \((1) \Rightarrow (2)\) by showing for the PDA based on the LL parser that it is equivalent to the CFG.
II.6.3. Equivalence of CFG and PDA

Example of LL Parsing

\[ S \rightarrow AC \]
\[ A \rightarrow aAb \quad A \rightarrow ab \]
\[ C \rightarrow cCd \quad C \rightarrow cd \]

\[ S \Rightarrow AC \Rightarrow aAbC \Rightarrow aAbCcCd \Rightarrow aAbccd \]

- We start on our PDA with stack \( S \) for the start symbol.
- So we have stack \( S \). We guess that the rule is \( S \rightarrow AC \), and replace the top symbol \( S \) on the stack by \( AC \).
- So we have stack \( AC \) (the top symbol of the stack is the left most one).
- We guess that the rule for the top symbol is \( A \rightarrow aAb \), and replace the top symbol \( A \) on the stack by \( aAb \).
- So we have stack \( aAbC \) (the top of the stack is the symbol most to the left).

\[ S \rightarrow AC \]
\[ A \rightarrow aAb \quad A \rightarrow ab \]
\[ C \rightarrow cCd \quad C \rightarrow cd \]

\[ S \Rightarrow AC \Rightarrow aAbC \Rightarrow aAbCcCd \Rightarrow aAbccd \]

- We have now stack \( aAbC \). We can accept the letter \( a \), and remove this symbol \( a \) from the stack.
- So we have stack \( AbC \).
- We guess that the rule is \( A \rightarrow ab \), and replace the top symbol \( A \) on the stack by \( ab \).
- So we have stack \( abbC \).
- Now we can 3 times accept a letter, namely \( a, b, b \) and remove them from the stack.
- So we have stack \( C \).

\[ S \rightarrow AC \]
\[ A \rightarrow aAb \quad A \rightarrow ab \]
\[ C \rightarrow cCd \quad C \rightarrow cd \]

\[ S \Rightarrow AC \Rightarrow aAbC \Rightarrow aAbCcCd \Rightarrow aAbccd \]

- We have now stack \( aAbC \). We can accept the letter \( a \), and remove this symbol \( a \) from the stack.
- So we have stack \( AbC \).
- We guess that the rule is \( A \rightarrow ab \), and replace the top symbol \( A \) on the stack by \( ab \).
- So we have stack \( abbC \).
- Now we can 3 times accept a letter, namely \( a, b, b \) and remove them from the stack.
- So we have stack \( C \).
**LL(k) Parsing**

- We had to guess which production to use.
- LL(k) is an algorithm to decide by using a lookahead of the next k symbols, which production to use.
- For instance when the stack was AC we could have guessed by knowing that the next two letters are a, a, that we need to use the production A → aAb and not A → ab, since the latter would give as next two letters ab.
- Usually only a lookahead of 1 symbol is used, but most standard grammars (e.g. Java) have no LL(1) grammars.

**Invariant**

- The invariant kept above was that when having consumed string w and obtained stack v, then we have S ⇒* wv.

**Resulting PDA**

- We obtain a PDA with a single state. Therefore we omit the state, and ignore the start state, set of states, final states, the state in configurations, and have transitions
  \[ Z \xrightarrow{a} z \]
  for \( Z \in \Gamma \), \( a \in T \), \( z \in \Gamma^* \).
- The stack symbols were the alphabet used in strings occurring in the derivations, i.e. \( T \cup N \).
- The start stack symbol was S.

**Resulting PDA**

- The PDA has the following productions:
  - If the top symbol is a non-terminal A, and there was a rule \( A \rightarrow w \), we could replace A by w, and make an \( \epsilon \)-transition. So we have in this case a transition
    \[ A \xrightarrow{\epsilon} w \]
  - If the top symbol was a terminal a, then we could accept letter a and pop the symbol from the stack. So we have transitions
    \[ a \xrightarrow{a} \epsilon \]
The empty stack PDA derived from $G = (T, N, S, P)$ is as follows:

<table>
<thead>
<tr>
<th>PDA</th>
<th>$P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminals</td>
<td>$T$</td>
</tr>
<tr>
<td>states</td>
<td>single state</td>
</tr>
<tr>
<td>stack alphab.</td>
<td>$T \cup N$</td>
</tr>
<tr>
<td>start stack</td>
<td>$S$</td>
</tr>
</tbody>
</table>
| transitions | $A \overset{\epsilon}{\rightarrow} w$ if $A \rightarrow w \in P$
              | $a \overset{a}{\rightarrow} \epsilon$ if $a \in T$ |

In the additional material there is a proof $L(P) = L(G)$. This can be found in the Additional Material.