In this lecture we introduce Pushdown Automata, a computational model equivalent to context free languages.

A pushdown automata is an NFA augmented with an infinitely large stack.

The additional memory enables recognition of some non regular languages.
Informal Description

A Pushdown Automata (PDA) can write an unbounded number of Stack Symbols on the stack and read these symbols later.

Writing a symbol onto the stack is called \textit{pushing} and it pushes all symbols on the stack one stack cell down.

Removing a symbol off the stack is called \textit{popping} and every symbol on the stack moves one stack cell up.

Note: A PDA can access only the stack’s topmost symbol (LIFO).

A PDA Recognizing $L = \{0^n1^n\}$

This PDA reads symbols from the input.
As each 0 is read, it is pushed onto the stack.
As each 1 is read, a 0 is popped from the stack.
If the stack becomes empty exactly when the last 1 is read – accept.
Otherwise – reject.

Nondeterministic PDAs

A Nondeterministic PDA allows nondeterministic transitions.
Nondeterministic PDA-s are \textit{strictly stronger} than deterministic PDA-s.
In this respect, the situation is not similar to the situation of DFA-s and NFA-s.
Nondeterministic PDA-s are \textit{equivalent to CFL-s}.
A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

1. $Q$ is a finite set called the states.
2. $\Sigma$ is the input alphabet.
3. $\Gamma$ is the stack alphabet.
4. $\delta: Q \times \Sigma \times \Gamma \rightarrow P(Q, \Gamma)$ is the transition function.
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accepting states.

Consider the expression

Assume that the PDA is in state $q \in Q$, the next input symbol is $\sigma \in \Sigma$, and the top symbol on the stack is $\gamma \in \Gamma$.

The next transition may either depend on the input symbol $\sigma$ and the stack symbol $\gamma$, or only on the input symbol $\sigma$, or only on the stack symbol $\gamma$, or on none of them. This choice is formally expressed by the argument of the transition function as detailed in the next slides.

Each step of the automaton is atomic, meaning it is executed in a single in devisable time unit. For descriptive purposes only, each step is divided into two separate sub-steps:

Sub-step1: A symbol may be read from the input, a symbol may be read and popped off the stack.

Sub-step2: A state transition is carried out and a stack symbol may be pushed on the stack.
Transition Function – 1st Sub-step

If the transition depends both on $\sigma$ and $\gamma$ we write $\delta(q,\sigma,\gamma)$. In this case $\sigma$ is consumed and $\gamma$ is removed from the stack.

If the transition depends only on $\sigma$ we write $\delta(q,\sigma,\varepsilon)$, $\sigma$ is consumed and the stack does not change.

Finally, if the transition depends neither on $\sigma$, nor on $\gamma$, we write $\delta(q,\varepsilon,\varepsilon)$. In this case $\sigma$ is not consumed and the stack is not changed.

PDA - The Transition Function

The range of the transition function is $P(Q,\Gamma_e)$:

The power set of the Cartesian product of the set of PDA states and the stack alphabet.

Using pairs means that $\delta$ determines:

1. The new state to which the PDA moves.
2. The new stack symbol pushed on the stack.

Using the power set means that the PDA is nondeterministic: At any given situation, it may make a **nondeterministic transition**.

Finally, the use of $\Gamma_e = \Gamma \cup \{\varepsilon\}$ means that at each transition the PDA may either push a stack symbol onto the stack or not, if the value is $\varepsilon$. 
The definition of a PDA does not give a special way to check emptiness.

One way to do it is to augment the stack alphabet with a special “emptiness” marker, the symbol $. (Note: There is nothing special about $ any other symbol not in the original \( \Gamma \) can do.)

The computation is started by an \( \varepsilon \) transition in which $ is pushed on the stack.

If the end marker $ is found on the stack at the end of the computation, it is popped by a single additional \( \varepsilon \) transition after which the automaton “knows” that the stack is empty.

The label of each transition represents the 1\(^{st}\) sub-step (left of arrow) and pushed stack symbol (right of the arrow).

The $ symbol, pushed onto the stack at the beginning of the computation, is used as an “empty” marker.
A PDA Recognizing $L = \{0^n1^n\}$

The PDA accepts either if the input is empty, or if scanning the input is completed and the PDA is at $q_4$.

CFLG-s and PDA-s are Equivalent

**Theorem:**
A language is CFL if and only if there exists a PDA accepting it.

**Lemma-**
For any CFL $L$, there exists a PDA $P$ such that $L = L(P)$.

**Proof Idea**
Since $L$ is a CFL there exists a CFG $G$ such that $L = L(G)$. We will present a PDA $P$, that recognizes $L$.

The PDA $P$ starts with a word $w \in \Sigma^*$ on its input.

In order to decide whether $w \in L(G)$, $P$ simulates the derivation of $w$.

**Proof Idea (cont.)**
Recall that a derivation is a sequence of strings, where each string contains variables and terminals. The first string is always the start symbol of $G$ and each string is obtained from the previous one by a single activation of some rule.
Proof Idea (cont.)

A string may allow activation of several rules and the PDA $P$ non deterministically guesses the next rule to be activated.

The initial idea for the simulation is to store each intermediate string on the stack. Upon each production, the string on the stack is before production is transformed to the string after production.

Unfortunately, this idea does not quite work since at any given moment, $P$ can only access the top symbol on the stack.

To overcome this problem, the stack holds only a suffix of each intermediate string where the top symbol is the variable to be substituted during the next production.

The Intermediate String $aaSbb$

Informal Description of $P$

Push the marker $\$ and the start symbol $S$ on the stack.

Repeat

If the top symbol is a variable $V$ – Replace $V$ by the right hand side of some non deterministically chosen rule whose left hand side is $V$.

.....
**Informal Description of \( P \)**

Push the marker $ and the start symbol \( S \) on the stack.

Repeat

......

If the top symbol is a terminal compare it with the next symbol on the input. If equal – advance the input and pop the variable else – reject.

---

**The Proof**

We start by defining **Extended Transitions**: Assume that PDA \( P \) is in state \( q \) , it reads \( a \in \Sigma \) from the input and pops \( s \in \Gamma \) from the stack and then moves to state \( r \) while pushing \( u = u_1,u_2,\ldots,u_l \) onto the stack.

This is denoted by \( (r,u) \in \delta(q,a,s) \).

Next, extended transitions are implemented.

---

**Implementing Extended Trans.**

Add states \( q_1,q_2,\ldots,q_{j-1} \).

Set the transition function \( \delta \) as follows:

Add \( (q_1,u_l) \) to \( \delta(q,a,s) \).

Set \( \delta(q_1,\varepsilon,\varepsilon) = \{q_2,u_{l-1}\} \),

\[
\delta(q_2,\varepsilon,\varepsilon) = \{q_2,u_{l-2}\},
\]

......

\[
\delta(q_{j-1},\varepsilon,\varepsilon) = \{r,u_1\} \quad \text{(see next slide)}
\]
Implementing Extended Trans.

Let $G$ be an arbitrary CFG. Now we are ready to construct the PDA, $P$ such that that $L(P) = L(G)$. The states of $P$ are a

$Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$ where $E$ contains all states needed to implement the extended transitions presented in the previous slide.

The PDA $P$ is presented on the next slide:

The Result PDA

Example

Consider the following CFG:

\[ S \rightarrow aTb \mid b \]
\[ T \rightarrow Ta \mid \epsilon \]
CFLG-s and PDA-s are Equivalent

**Theorem (rerun):**
A language is CFL if and only if there exists a PDA accepting it.

**Lemma**-
For any PDA $P$, there exists a CFG $G$ such that $L(P) = L(G)$.

**Proof Idea**
The proof has two parts:

**Part1:** Use the PDA $P$ to construct the grammar $G$ - interesting and intuitive (Here!).

**Part2:** Prove by induction that $L(P) = L(G)$, That is, show that for every $w \in \Sigma^*$, $w \in L(P)$, if and only if $w \in L(G)$ - Complex (Here?).

**Construction of $G$**
First change the PDA $P$ as follows:
1. The PDA $P$ has a single accept state $q_{accept}$.
2. The PDA $P$ empties its stack before accepting.
3. Each transition either pushes a symbol on the Stack or pops a symbol, but not both.
(WE already know how to do it).

**The Variables of $G$ and the Idea**
For each pair of states $P, p$ and $q$, the grammar $G$s has a variable $A_{pq}$.

The idea of the construction: $w \in \Sigma^*$ can be derived from the variable $A_{pq}$ *if and only if* $w$ takes PDA $P$ from $p$, with an empty stack to $q$ with an empty stack.
Properties of $G$

**Note:** if $w$ takes $P$ from $p$, with an empty stack to $q$ with an empty stack then

\[ w \text{ takes } P \text{ from } p, \text{ with any stack content to } q \text{ with the same stack content.} \]

**Also note:** The first step of $P$ from $p$, to $q$ with $w$ must be a push step.

---

Rules of $G$

**Assume** $w$ takes $P$ from $p$, with an empty stack to $q$ with an empty stack. There are 2 possibilities:

1. During $P$‘s transition from $p$ to $q$ with an empty stack, the stack is never empty.
2. There exists some intermediate state $r$ in which the stack is empty.

---

Properties of $G$

**Also Note:** We do not know if there exists even a single $w \in \Sigma^*$, such that $w$ takes $P$ from $p$ with an empty stack to $q$ with an empty stack. The grammar $G$ should be obtained from $P$ by syntactical transformations and not by trying to simulate $P$.

---

Rules of $G$ – First Type

The first type of rules is designed to enable a derivation of $w$ from $G$ iff $w$ takes $P$ from $p$ to $q$ where the stack is not empty in any intermediate state. If such a $w$ exists, its first symbol should induce a push step of some stack symbol $t$ (since at $p$ the stack id empty). For a similar reason, the last symbol of $w$ must induce a pop step of the same symbol $t$.
Rules of $G$ – First Type

The rules of the first type are designed to enable a derivation of $w$ that takes $P$ from $p$ to $q$, if such a $w$ exists:

Let $a$, be the input symbol inducing $P$'s transition from $p$ to $r$ while pushing stack symbol $t$.

Let $b$ be the input symbol inducing $P$'s transition from $s$ to $q$ while popping the same symbol $t$.

Add to $G$ the rule $A_{pq} \rightarrow aA_{rs}b$.

Rules of $G$ – Second Type

The second type of rules is designed to enable a derivation of $w$ from $G$ if $w$ takes $P$ from $p$ to $q$ where the stack is empty on some intermediate state.

Since we cannot guess which will the intermediate state be, we add to $G$ the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ for any state $r \in Q$.

Rules of $G$ – Third Type

So far, all the rules have a variable on the right hand side.

One can see that if $w$ takes $P$ from $p$, with an empty stack to $q$ with an empty stack where the stack is not empty in any intermediate step, then $A_{pq} \xrightarrow{*} \sigma_0 A_{i_1i_1} \sigma_1 ... A_{i_ii} \sigma_i$ is a derivation of $G$.

Rules of $G$ – Third Type

Each $\sigma_j$ in the derivation

$A_{pq} \xrightarrow{*} \sigma_0 A_{i_1i_1} \sigma_1 ... A_{i_ii} \sigma_i$

Is an input symbol and $w = \sigma_0 \sigma_1 ... \sigma_l$.

The variables of $G A_{i_1i_1} A_{i_2i_2} ... A_{i_ii}$ correspond to all intermediate states which $P$ visits on its computation from $p$, with an empty stack to $q$ with an empty stack with input $w$. 
In order to derive \( w \), we add rules of a third type: for any state \( p \) of \( P \) add the rule \( \epsilon \overset{}{\rightarrow} pp \).

This completes the construction of \( G \).

For each \( Q_{srqp} \in \Gamma \), add the rule \( \overset{}{\rightarrow} pp \) to \( G \).

For each \( Q_{rqp} \in \Gamma \), add the rule \( \overset{}{\rightarrow} pp \) to \( G \).

The Proof

For any \( w \in \Sigma^* \), we have to prove that \( G \) generates \( w \) if and only if \( P \) accepts it. If we prove just one direction we may end up generating words that \( P \) does not accept or with \( G \) generating words that \( P \) does not accept.

Claim

For each \( p \in P \), with an empty stack to with an empty stack.

If \( \forall \Delta_{bd} \in \Sigma \), the rule \( \Delta_{bd} \rightarrow \epsilon \) to \( G \).

For each \( p \in P \), \( s \in T \), and stack.

Rules of \( G \) - A Formal Definition

In order to derive \( w \), we add rules of a third type.
**Proof**

By induction on the length (number of steps) of the derivation of \(x\).

**Basis:** The derivation has 1 step

The only rule whose left side has no variables \(A_{pp} \rightarrow \epsilon\), obviously \(\epsilon\) takes \(P\) from \(p\), with an empty stack to \(p\) with an empty stack.

**The Induction Step**

Assume the claim is true for derivations of \(k\) steps, \(k \geq 1\) and let \(D = A_{pq} \xrightarrow{k+1} x\) be a derivation of \(k + 1\) steps. Consider the first step of \(D\). Its form is either \(A_{pq} \rightarrow aA_{rs}b\) or \(A_{pq} \rightarrow A_{pr}A_{rq}\).

**The Induction Step - case** \(A_{pq} \rightarrow aA_{rs}b\)

In this case Assume the claim is true for \(x = ayb\) and \(E = A_{rs} \xrightarrow{k} y\) is a derivation of \(y\) with \(k\) steps.

By the IH \(y\) takes \(P\) from \(r\), with an empty stack to \(s\) with an empty stack. Since \(A_{pq} \rightarrow aA_{rs}b\) is a rule of \(G\), we can conclude \((r,t) \in \delta(p,a,\epsilon)\) and \((q,\epsilon) \in \delta(s,b,t)\). QED

**The Induction Step - case** \(A_{pq} \rightarrow A_{pr}A_{rq}\)

Exercise to the reader.
Claim

If $x$ can bring $P$ from $p$, with an empty stack to $q$ with an empty stack then $A_{pq}$ generates $x$.

Proof

Read in the book.

Wrap Up

In this lecture we introduced Pushdown Automata, and proved that a language $L$ is CFL if and only if there exists a PDA $P$ recognizing $L$. 