

CS 483 - Introduction to the Theory of Computation

Lecture 7

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

$$S \rightarrow \varepsilon \mid aSb$$

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

$$S \rightarrow \varepsilon \mid aSb$$

A non-regular grammar for $L = \{ww^R \mid w \in \{a, b\}^*\}$?

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

$$S \rightarrow \varepsilon \mid aSb$$

A non-regular grammar for $L = \{ww^R \mid w \in \{a, b\}^*\}$?

$$S \rightarrow \varepsilon \mid aSa \mid bSb$$

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

$$S \rightarrow \varepsilon \mid aSb$$

A non-regular grammar for $L = \{ww^R \mid w \in \{a, b\}^*\}$?

$$S \rightarrow \varepsilon \mid aSa \mid bSb$$

A non-regular grammar for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$?

Non-regular Language Revisited Again

A non-regular grammar for $L = \{a^n b^n \mid n \in \mathbb{N}\}$?

$$S \rightarrow \varepsilon \mid aSb$$

A non-regular grammar for $L = \{ww^R \mid w \in \{a, b\}^*\}$?

$$S \rightarrow \varepsilon \mid aSa \mid bSb$$

A non-regular grammar for $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$?

$$S \rightarrow AR \mid LC$$

$$A \rightarrow \varepsilon \mid aA$$

$$C \rightarrow \varepsilon \mid cC$$

$$R \rightarrow \varepsilon \mid bRc$$

$$L \rightarrow \varepsilon \mid aLb$$

Context-Free Grammar

A grammar $G = (N, \Sigma, P, S)$ is a **context-free grammar (CFG)** if every rule in P has the form

$$A \rightarrow \alpha$$

for $A \in N$ and $\alpha \in (N \cup \Sigma)^*$.

The languages generated by CFGs are called **context-free languages (CFL)**.

More Examples of CFG

The grammar:

$$S \rightarrow SS \mid a$$

More Examples of CFG

The grammar:

$$S \rightarrow SS \mid a$$

generates the language

$$\{a^n \mid n \geq 1\}$$

More Examples of CFG

The grammar:

$$S \rightarrow \varepsilon \mid \text{"}\varepsilon\text{"} \mid a \mid b \mid S \cup S \mid SS \mid S^* \mid (S)$$

generates all regexes on the alphabet $\{a, b\}$.

More Examples of CFG

The grammar:

$$S \rightarrow \varepsilon \mid SS \mid (S)$$

generates all valid nested parentheses,

e.g. $(()())()$

More Examples of CFG

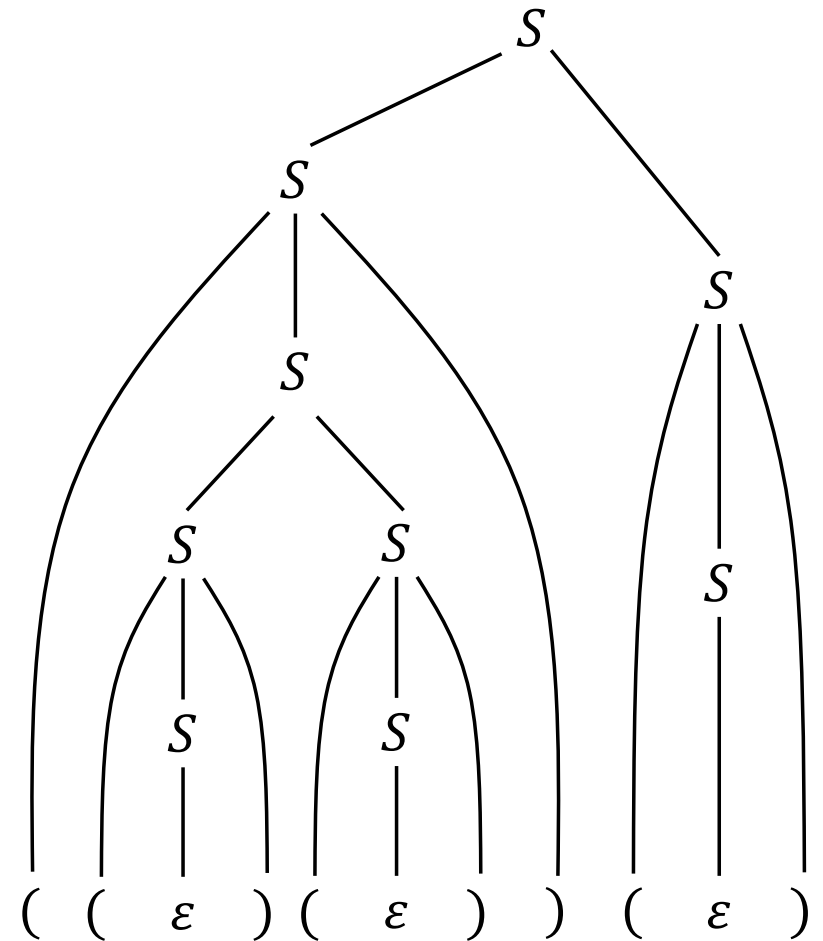
The grammar:

$$S \rightarrow \varepsilon \mid SS \mid (S)$$

generates all valid nested parentheses,

e.g. $((0(0)))0$

parsing tree:



More Examples of CFG

The grammar:

$$S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$$

generates all valid nested parentheses and brackets,

e.g. $[O[O]]O$

but not $[()]$

More Examples of CFG

The grammar with

$$N = \{\langle \text{expr} \rangle, \langle \text{term} \rangle, \langle \text{factor} \rangle, \langle \text{number} \rangle\},$$

$$\Sigma = \{0, 1, \dots, 9, +, \times, (,)\},$$

$$S = \langle \text{expr} \rangle$$

and production rules:

$$\langle \text{expr} \rangle \rightarrow \langle \text{term} \rangle \mid \langle \text{term} \rangle + \langle \text{expr} \rangle$$

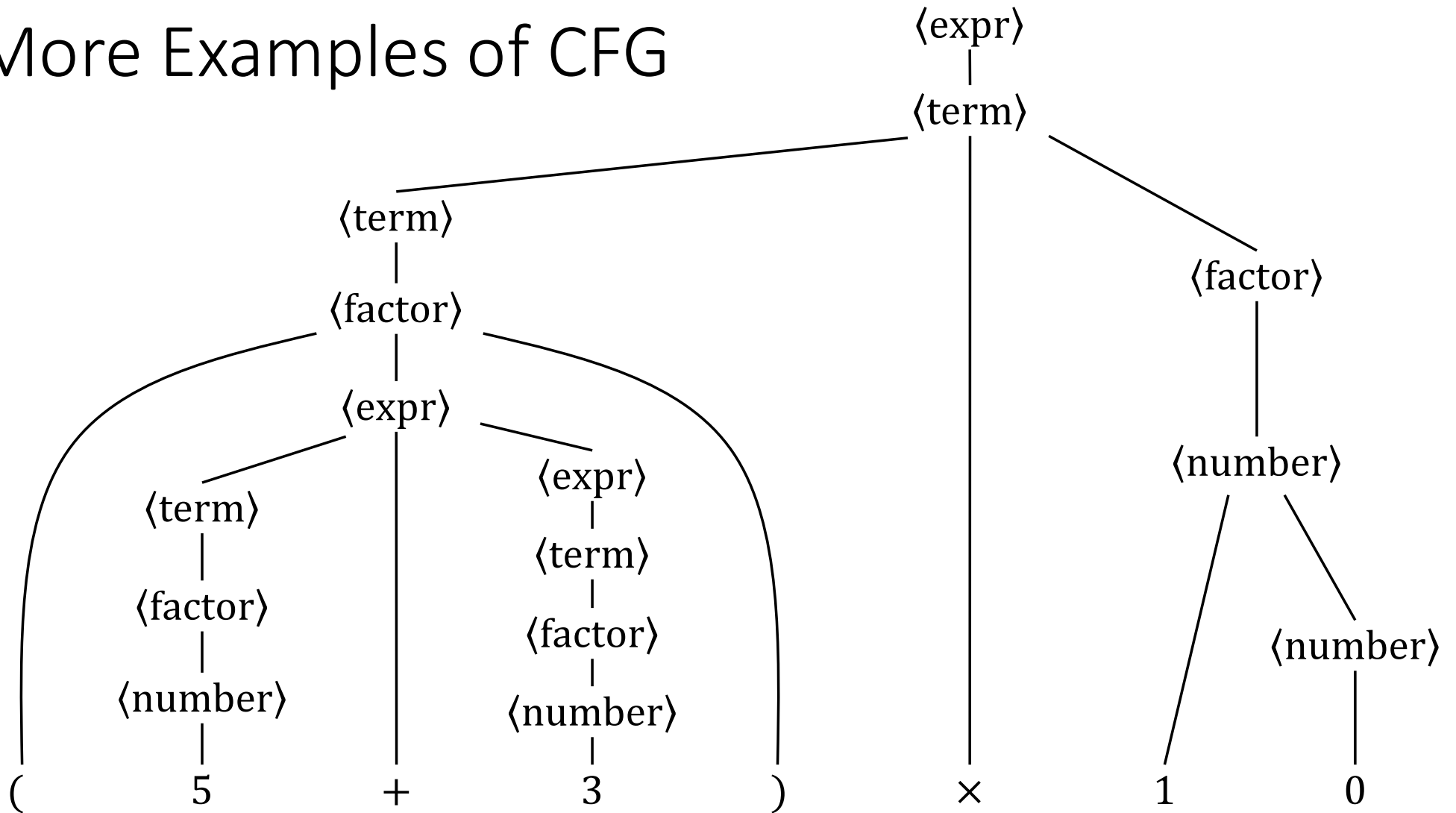
$$\langle \text{term} \rangle \rightarrow \langle \text{factor} \rangle \mid \langle \text{term} \rangle \times \langle \text{factor} \rangle$$

$$\langle \text{factor} \rangle \rightarrow \langle \text{number} \rangle \mid (\langle \text{expr} \rangle)$$

$$\langle \text{number} \rangle \rightarrow 0 \mid 1 \mid \dots \mid 9 \mid 0\langle \text{number} \rangle \mid 1\langle \text{number} \rangle \mid \dots \mid 9\langle \text{number} \rangle$$

generates all arithmetic expressions with addition and multiplication.

More Examples of CFG



More Examples of CFG

Programming languages are **almost** CFLs:

$\langle \text{program} \rangle \rightarrow \langle \text{statement} \rangle \mid \langle \text{statement} \rangle \langle \text{program} \rangle$

$\langle \text{statement} \rangle \rightarrow \langle \text{assignment} \rangle \mid \langle \text{if_stmt} \rangle \mid \langle \text{for_stmt} \rangle \mid \dots$

$\langle \text{assignment} \rangle \rightarrow \langle \text{variable_name} \rangle = \langle \text{expr} \rangle;$

$\langle \text{if_stmt} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \{ \langle \text{program} \rangle \}$

...

More Examples of CFG

Programming languages are **almost** CFLs:

$$\begin{aligned}\langle \text{program} \rangle &\rightarrow \langle \text{statement} \rangle \mid \langle \text{statement} \rangle \langle \text{program} \rangle \\ \langle \text{statement} \rangle &\rightarrow \langle \text{assignment} \rangle \mid \langle \text{if_stmt} \rangle \mid \langle \text{for_stmt} \rangle \mid \dots \\ \langle \text{assignment} \rangle &\rightarrow \langle \text{variable_name} \rangle = \langle \text{expr} \rangle; \\ \langle \text{if_stmt} \rangle &\rightarrow \text{if } \langle \text{expr} \rangle \{ \langle \text{program} \rangle \} \\ &\dots\end{aligned}$$

The resulted programs can be **parsed** but not necessarily **compiled or run**.

See e.g.: [Full grammar for Python3](#)

More Examples of CFG

Most natural languages are **almost** CFLs:

$\langle \text{sentence} \rangle \rightarrow \langle \text{noun phrase} \rangle \langle \text{verb phrase} \rangle.$

$\langle \text{noun phrase} \rangle \rightarrow \langle \text{noun}' \rangle \langle \text{prep phrase} \rangle$

$\langle \text{verb phrase} \rangle \rightarrow \langle \text{verb}' \rangle \langle \text{prep phrase} \rangle$

$\langle \text{prep phrase} \rangle \rightarrow \langle \text{prep} \rangle \langle \text{noun}' \rangle$

$\langle \text{noun}' \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \mid \langle \text{article} \rangle \langle \text{adj} \rangle \langle \text{noun} \rangle$

$\langle \text{verb}' \rangle \rightarrow \langle \text{verb} \rangle \mid \langle \text{verb} \rangle \langle \text{noun}' \rangle$

$\langle \text{article} \rangle \rightarrow \varepsilon \mid a \mid an \mid the$

...

More Examples of CFG

Most natural languages are **almost** CFLs:

$\langle \text{sentence} \rangle \rightarrow \langle \text{noun phrase} \rangle \langle \text{verb phrase} \rangle.$

$\langle \text{noun phrase} \rangle \rightarrow \langle \text{noun}' \rangle \langle \text{prep phrase} \rangle$

$\langle \text{verb phrase} \rangle \rightarrow \langle \text{verb}' \rangle \langle \text{prep phrase} \rangle$

$\langle \text{prep phrase} \rangle \rightarrow \langle \text{prep} \rangle \langle \text{noun}' \rangle$

$\langle \text{noun}' \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \mid \langle \text{article} \rangle \langle \text{adj} \rangle \langle \text{noun} \rangle$

$\langle \text{verb}' \rangle \rightarrow \langle \text{verb} \rangle \mid \langle \text{verb} \rangle \langle \text{noun}' \rangle$

$\langle \text{article} \rangle \rightarrow \varepsilon \mid a \mid an \mid the$

...

The resulted sentences are **syntactically** correct but not necessarily **semantically** correct.

How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

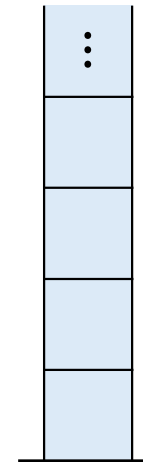
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A **stack**.

[() [()]] ()



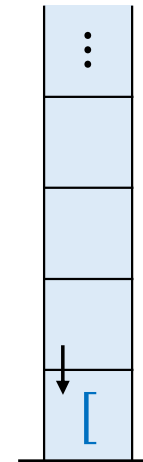
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()
↑



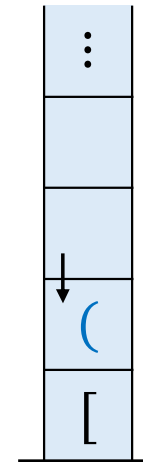
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()
 ↑



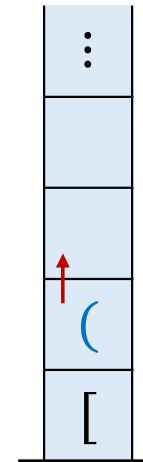
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()
 ↑



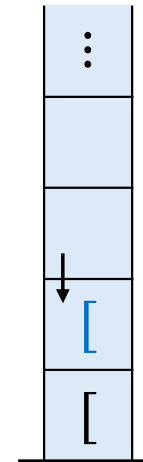
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()
 ↑



How to compute CFL?

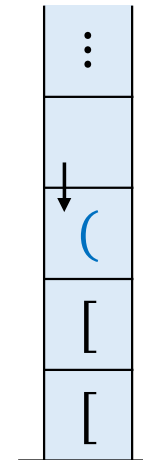
Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()

↑



How to compute CFL?

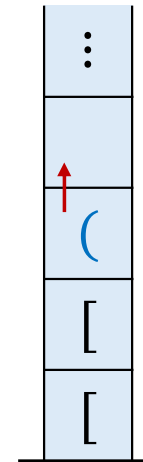
Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()

↑



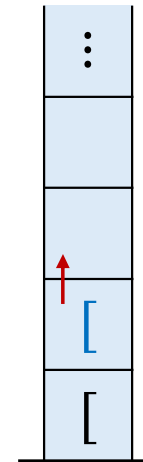
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()
 ↑



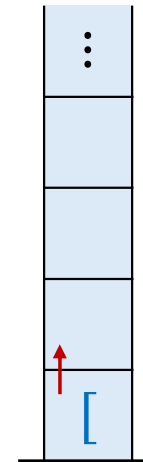
How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()] **[** ()
 ↑



How to compute CFL?

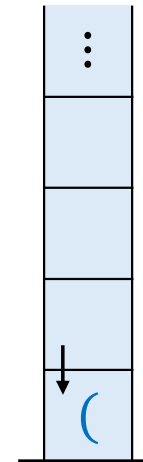
Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

[() [()]] ()

↑

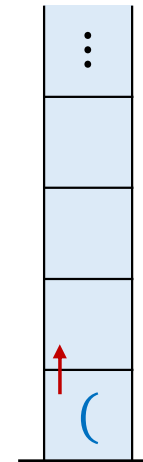


How to compute CFL?

Example: $S \rightarrow \varepsilon \mid SS \mid (S) \mid [S]$

What algorithm can recognize all valid nested parentheses and brackets, that reads the input in one pass?

A stack.

$$[\quad (\quad) \quad [\quad (\quad) \quad] \quad] \quad (\quad)$$


Pushdown Automaton

A pushdown automaton (PDA) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- Γ is another finite set of symbols (stack alphabet);

Pushdown Automaton

A pushdown automaton (PDA) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- Γ is another finite set of symbols (stack alphabet);
- δ is the transition function:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times \Gamma^*}$$

Pushdown Automaton

A pushdown automaton (PDA) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- Γ is another finite set of symbols (stack alphabet);
- δ is the transition function:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times \Gamma^*}$$

The diagram illustrates the components of the transition function δ . Arrows point from the parts of the function signature to descriptive labels:

- From Q to "current state"
- From $(\Sigma \cup \{\varepsilon\})$ to "next symbol"
- From $(\Gamma \cup \{\varepsilon\})$ to "top of stack (pop)"
- From $2^{Q \times \Gamma^*}$ to "new state" (via an arrow pointing to Q) and "new stack content (push)" (via an arrow pointing to Γ^*)

Pushdown Automaton

A pushdown automaton (PDA) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- Γ is another finite set of symbols (stack alphabet);
- δ is the transition function:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times \Gamma^*}$$

and the values of δ are **finite subsets** of $Q \times \Gamma^*$

Pushdown Automaton

A pushdown automaton (PDA) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ where:

- Q is a finite set of states;
- Σ is a finite set of symbols (alphabet);
- Γ is another finite set of symbols (stack alphabet);
- δ is the transition function:

$$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times \Gamma^*}$$

and the values of δ are finite subsets of $Q \times \Gamma^*$

- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the set of accepting states.

Pushdown Automaton

The input $w \in \Sigma^*$ is accepted by the PDA M if and only if there exist:

- an ε -padded $y = y_1 \cdots y_m$ such that $y = w$
- a sequence of states $r_0, r_1, \dots, r_m$ where $r_0 = q_0$ and $r_m \in F$
- a sequence of strings $s_0, s_1, \dots, s_m \in \Gamma^*$ representing the [stack](#):

Pushdown Automaton

The input $w \in \Sigma^*$ is accepted by the PDA M if and only if there exist:

- an ε -padded $y = y_1 \cdots y_m$ such that $y = w$
- a sequence of states $r_0, r_1, \dots, r_m$ where $r_0 = q_0$ and $r_m \in F$
- a sequence of strings $s_0, s_1, \dots, s_m \in \Gamma^*$ representing the **stack**:
 - $s_0 = s_m = \varepsilon$
 - $(r_{i+1}, \textcolor{blue}{b}) \in \delta(r_i, y_{i+1}, \textcolor{red}{a})$ for some $a \in \Gamma \cup \{\varepsilon\}$, $b \in \Gamma^*$ and $t \in \Gamma^*$ such that
$$s_i = \textcolor{red}{a}t, \quad s_{i+1} = \textcolor{blue}{b}t$$

Examples of Pushdown Automata

Example 1:

$$L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$$

Examples of Pushdown Automata

Example 1:

$$L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$$

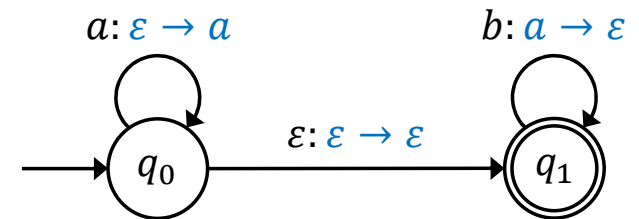
- $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a\}$, $F = \{q_1\}$
- Non-empty values of δ :
 - $\delta(q_0, a, \varepsilon) = \{(q_0, a)\}$ (read a and push a)
 - $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\}$ (transit to second part)
 - $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$ (read b and pop a)

Examples of Pushdown Automata

Example 1:

$$L_1 = \{a^n b^n \mid n \in \mathbb{N}\}$$

- $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a\}$, $F = \{q_1\}$
- Non-empty values of δ :
 - $\delta(q_0, a, \varepsilon) = \{(q_0, a)\}$ (read a and push a)
 - $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\}$ (transit to second part)
 - $\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$ (read b and pop a)



Examples of Pushdown Automata

Example 2:

$$L_2 = \{ww^R \mid w \in \{a, b\}^*\}$$

Examples of Pushdown Automata

Example 2:

$$L_2 = \{ww^R \mid w \in \{a, b\}^*\}$$

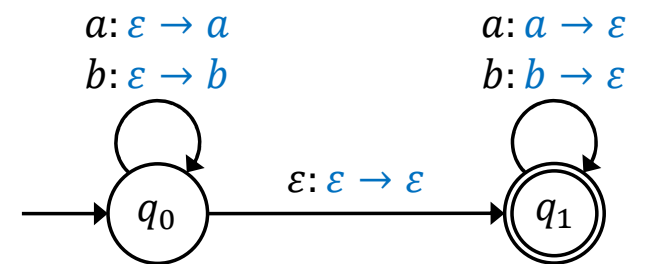
- $Q = \{q_0, q_1\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b\}$, $F = \{q_1\}$

- Non-empty values of δ :

- $\delta(q_0, a, \varepsilon) = \{(q_0, a)\}$, $\delta(q_0, b, \varepsilon) = \{(q_0, b)\}$

- $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \varepsilon)\}$

- $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$, $\delta(q_1, b, b) = \{(q_1, \varepsilon)\}$



Examples of Pushdown Automata

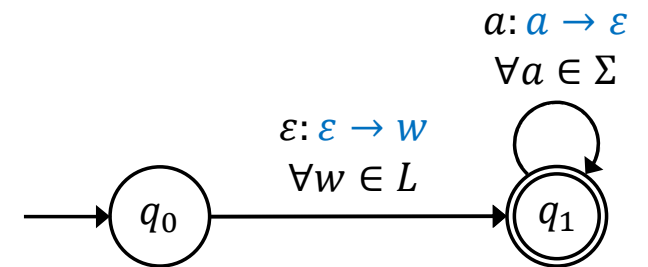
Example 2.5: What if the values of δ can be infinite?

Examples of Pushdown Automata

Example 2.5: What if the values of δ can be infinite?

$L \subseteq \Sigma^*$ is an arbitrary language.

- $Q = \{q_0, q_1\}, \Gamma = \Sigma, F = \{q_1\}$
- Non-empty values of δ :
 - $\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, w) \mid w \in L\}$
 - $\delta(q_1, a, a) = \{(q_1, \varepsilon)\}$ for each $a \in \Sigma$



Examples of Pushdown Automata

Example 3:

$$L_3 = \{w \in \{a, b\}^* \mid w \text{ contains same numbers of } a \text{ and } b\}$$

We will continue this example in the next lecture.