Theoretical Computer Science
Context-Free Languages

Jonas Hübotter

Outline

Overview

Representations of context-free languages

Overview

Representations of context-free languages

- Context-Free Grammar (CFG)

Overview

Representations of context-free languages

- Context-Free Grammar (CFG)
- Pushdown Automaton (PDA)

Variables

Definition 1

Given a grammar $G = (V, \Sigma, P, S)$, a variable $X \in V$ is

- generative if $\exists X \rightarrow_G^* w \in \Sigma^*$

Variables

Definition 1

Given a grammar $G = (V, \Sigma, P, S)$, a variable $X \in V$ is

- generative if $\exists X \rightarrow_G^* w \in \Sigma^*$;
- reachable if $\exists S \rightarrow_G^* X$

Variables

Definition 1

Given a grammar $G = (V, \Sigma, P, S)$, a variable $X \in V$ is

- generative if $\exists X \rightarrow_G^* w \in \Sigma^*$;
- reachable if $\exists S \rightarrow_G^* X$; and
- helpful if it is generative and reachable.

Inductive Definition

Given a context-free grammar $G = (V, \Sigma, P, S)$ with $V = \{A_1, \ldots, A_k\}$,

Inductive Definition

Given a context-free grammar $G = (V, \Sigma, P, S)$ with
$V = \{A_1, \ldots, A_k\}$,
productions $A_i \to w_0 A_{i_1} w_1 \ldots w_{n-1} A_{i_n} w_n$

Inductive Definition

Given a context-free grammar $G = (V, \Sigma, P, S)$ with
$V = \{A_1, \ldots, A_k\}$,
productions $A_i \rightarrow w_0 A_{i_1} w_1 \ldots w_{n-1} A_{i_n} w_n$
correspond to

$$u_1 \in L_G(A_{i_1}) \wedge \cdots \wedge u_n \in L_G(A_{i_n}) \\ \implies w_0 u_1 w_1 \ldots w_{n-1} u_n w_n \in L_G(A_i).$$

Inductive Definition

Given a context-free grammar $G = (V, \Sigma, P, S)$ with
$V = \{A_1, \dots, A_k\}$,
productions $A_i \to w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$
correspond to

$$\begin{aligned} & u_1 \in L_G(A_{i_1}) \land \dots \land u_n \in L_G(A_{i_n}) \\ & \implies w_0 u_1 w_1 \dots w_{n-1} u_n w_n \in L_G(A_i). \end{aligned}$$

Hence, $L(G) = L_G(S)$.

Inductive Definition

Given a context-free grammar $G = (V, \Sigma, P, S)$ with
$V = \{A_1, \dots, A_k\}$,
productions $A_i \to w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$
correspond to

$$u_1 \in L_G(A_{i_1}) \wedge \dots \wedge u_n \in L_G(A_{i_n}) \implies w_0 u_1 w_1 \dots w_{n-1} u_n w_n \in L_G(A_i).$$

Hence, $L(G) = L_G(S)$.

Productions produce words top-down,
inductive definition *produces* words bottom-up.

Decomposition Lemma

Lemma 2 (Decomposition Lemma)

Any derivation of length n of β *from* $\alpha_1\alpha_2$ *may split* β *into two separately derivable parts* β_1 *and* β_2 *at any position.*

Decomposition Lemma

Lemma 2 (Decomposition Lemma)

Any derivation of length n of β from $\alpha_1\alpha_2$ may split β into two separately derivable parts β_1 and β_2 at any position. Formally:

$$\alpha_1\alpha_2 \rightarrow^{n}_{G} \beta \iff \exists \beta_1, \beta_2, n_1, n_2.\ \beta = \beta_1\beta_2 \wedge n = n_1 + n_2 \wedge \alpha_1 \rightarrow^{n_1}_{G} \beta_1 \wedge \alpha_2 \rightarrow^{n_2}_{G} \beta_2.$$

Syntax Tree

Definition 3

A syntax tree of a derivation $\rightarrow_G$ given $G = (V, \Sigma, P, S)$ is a tree where

Syntax Tree

Definition 3

A syntax tree of a derivation $\rightarrow_G$ given $G = (V, \Sigma, P, S)$ is a tree where

1. every leaf is labeled with a symbol in $\Sigma \cup \{\epsilon\}$

Syntax Tree

Definition 3

A syntax tree of a derivation $\rightarrow_G$ given $G = (V, \Sigma, P, S)$ is a tree where

1. every leaf is labeled with a symbol in $\Sigma \cup \{\epsilon\}$;
2. every inner node is labeled with $A \in V$,
   assuming its children are $X_1, \ldots, X_n \in V \cup \Sigma \cup \{\epsilon\}$,
   $A \rightarrow X_1 \ldots X_n \in P$

Syntax Tree

Definition 3

A syntax tree of a derivation $\to_G$ given $G = (V, \Sigma, P, S)$ is a tree where

1. every leaf is labeled with a symbol in $\Sigma \cup \{\epsilon\}$;
2. every inner node is labeled with $A \in V$,
   assuming its children are $X_1, \ldots, X_n \in V \cup \Sigma \cup \{\epsilon\}$,
   $A \to X_1 \ldots X_n \in P$; and
3. a leaf labeled ϵ is an only child of its parent.

Syntax Tree

Definition 3

A syntax tree of a derivation $\rightarrow_G$ given $G = (V, \Sigma, P, S)$ is a tree where

1. every leaf is labeled with a symbol in $\Sigma \cup \{\epsilon\}$;
2. every inner node is labeled with $A \in V$,
   assuming its children are $X_1, \ldots, X_n \in V \cup \Sigma \cup \{\epsilon\}$,
   $A \rightarrow X_1 \ldots X_n \in P$; and
3. a leaf labeled ϵ is an only child of its parent.

The border of a syntax tree is the labels of its leafs concatenated from left to right.

Syntax Tree

Definition 3

A syntax tree of a derivation $\rightarrow_G$ given $G = (V, \Sigma, P, S)$ is a tree where

1. every leaf is labeled with a symbol in $\Sigma \cup \{\epsilon\}$;
2. every inner node is labeled with $A \in V$,
   assuming its children are $X_1, \ldots, X_n \in V \cup \Sigma \cup \{\epsilon\}$,
   $A \rightarrow X_1 \ldots X_n \in P$; and
3. a leaf labeled ϵ is an only child of its parent.

The border of a syntax tree is the labels of its leafs concatenated from left to right.

$$A \rightarrow_G^* w \iff w \in L_G(A)$$
$$\iff \exists \text{ syntax tree with root } A \text{ and border } w.$$

Syntax Tree

Definition 4

- A CFG G is ambiguous if $\exists w \in L(G)$ that has two distinct syntax trees.
- A CFL L is inherently ambiguous if every CFG G with $L(G) = L$ is ambiguous.

Chomsky Normal Form

Definition 5 (Chomsky Normal Form)

All productions are of the form $A \to a$ or $A \to BC$ for $a \in Sigma$ and $A, B, C \in V$.

Chomsky Normal Form

Definition 5 (Chomsky Normal Form)

All productions are of the form $A \rightarrow a$ or $A \rightarrow BC$ for $a \in Sigma$ and $A, B, C \in V$.

Algorithm to convert a CFG to Chomsky Normal Form ($\mathcal{O}(|P|^2)$)

1. replace every $a \in \Sigma$ occurring in a production with length > 1 by a non-terminal

Chomsky Normal Form

Definition 5 (Chomsky Normal Form)

All productions are of the form $A \rightarrow a$ or $A \rightarrow BC$ for $a \in Sigma$ and $A, B, C \in V$.

Algorithm to convert a CFG to Chomsky Normal Form ($\mathcal{O}(|P|^2)$)

1. replace every $a \in \Sigma$ occurring in a production with length > 1 by a non-terminal
2. replace $A \rightarrow B_1 \dots B_k$ (where $k > 2$) with $A \rightarrow B_1 C_2, C_2 \rightarrow B_2, \dots, C_k \rightarrow B_k$

Chomsky Normal Form

Definition 5 (Chomsky Normal Form)

All productions are of the form $A \to a$ or $A \to BC$ for $a \in Sigma$ and $A, B, C \in V$.

Algorithm to convert a CFG to Chomsky Normal Form ($\mathcal{O}(|P|^2)$)

1. replace every $a \in \Sigma$ occurring in a production with length > 1 by a non-terminal
2. replace $A \to B_1 \dots B_k$ (where $k > 2$) with $A \to B_1 C_2, C_2 \to B_2, \dots, C_k \to B_k$
3. remove ϵ-productions (i.e. $A \to \epsilon$)

Chomsky Normal Form

Definition 5 (Chomsky Normal Form)

All productions are of the form $A \to a$ or $A \to BC$ for $a \in Sigma$ and $A, B, C \in V$.

Algorithm to convert a CFG to Chomsky Normal Form ($\mathcal{O}(|P|^2)$)

1. replace every $a \in \Sigma$ occurring in a production with length > 1 by a non-terminal
2. replace $A \to B_1 \dots B_k$ (where $k > 2$) with $A \to B_1 C_2, C_2 \to B_2, \dots, C_k \to B_k$
3. remove ϵ-productions (i.e. $A \to \epsilon$)
4. remove chain productions (i.e. $A \to B$)

Other Normal Forms

Definition 6 (Greibach Normal Form)

All productions are of the form $A \to aA_1 \dots A_n$ for $a \in Sigma$ and $A_1, \dots, A_n \in V$.

Other Normal Forms

Definition 6 (Greibach Normal Form)

All productions are of the form $A \rightarrow aA_1 \ldots A_n$ for $a \in Sigma$ and $A_1, \ldots, A_n \in V$.

Definition 7 (Backus-Naur Normal Form)

Allows the use of regular expressions in productions (in addition to symbols).

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Algorithm ($\mathcal{O}(|w|^3)$)

Given $G = (V, \Sigma, P, S)$ in Chomsky normal form and $w = a_1 \ldots a_n \in \Sigma^*$.

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Algorithm ($\mathcal{O}(|w|^3)$)

Given $G = (V, \Sigma, P, S)$ in Chomsky normal form and $w = a_1 \dots a_n \in \Sigma^*$.
Define $V_{ij} = \{A \in V \mid A \to_G^* a_i \dots a_j\}$ for $i \leq j$ as the set of all initial symbols that may be used to derive $a_i \dots a_j$.

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Algorithm ($\mathcal{O}(|w|^3)$)

Given $G = (V, \Sigma, P, S)$ in Chomsky normal form and
$w = a_1 \dots a_n \in \Sigma^*$.
Define $V_{ij} = \{A \in V \mid A \rightarrow_G^* a_i \dots a_j\}$ for $i \leq j$ as the set of all initial symbols that may be used to derive $a_i \dots a_j$.
Then $w \in L_G(A) \iff A \in V_{1n}$.

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Algorithm ($\mathcal{O}(|w|^3)$)

Given $G = (V, \Sigma, P, S)$ in Chomsky normal form and
$w = a_1 \dots a_n \in \Sigma^*$.
Define $V_{ij} = \{A \in V \mid A \rightarrow_G^* a_i \dots a_j\}$ for $i \leq j$ as the set of all initial symbols that may be used to derive $a_i \dots a_j$.
Then $w \in L_G(A) \iff A \in V_{1n}$.

Recursive definition of V_{ij}:

- base: $V_{ii} = \{A \in V \mid (A \rightarrow a_i) \in P\}$

Cocke-Younger-Kasami Algorithm (CYK)

Solves the word problem for CFGs.

Algorithm ($\mathcal{O}(|w|^3)$)

Given $G = (V, \Sigma, P, S)$ in Chomsky normal form and
$w = a_1 \dots a_n \in \Sigma^*$.
Define $V_{ij} = \{A \in V \mid A \rightarrow_G^* a_i \dots a_j\}$ for $i \leq j$ as the set of all initial symbols that may be used to derive $a_i \dots a_j$.
Then $w \in L_G(A) \iff A \in V_{1n}$.

Recursive definition of V_{ij}:

- base: $V_{ii} = \{A \in V \mid (A \rightarrow a_i) \in P\}$
- step:

$$V_{ij} = \{A \in V \mid {}^{\exists i \leq k < j, B \in V_{ik}, C \in V_{(k+1)j}.}_{(A \rightarrow BC) \in P}\}$$

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ;
- an initial state $q_0 \in Q$

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ;
- an initial state $q_0 \in Q$;
- an initial stack element $Z_0 \in \Gamma$

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ;
- an initial state $q_0 \in Q$;
- an initial stack element $Z_0 \in \Gamma$;
- a (partial) transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ;
- an initial state $q_0 \in Q$;
- an initial stack element $Z_0 \in \Gamma$;
- a (partial) transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$; and
- a set of terminal (accepting) states $F \subseteq Q$.

PDA

Definition 8

A pushdown automaton (PDA) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) stack alphabet Γ;
- an initial state $q_0 \in Q$;
- an initial stack element $Z_0 \in \Gamma$;
- a (partial) transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$; and
- a set of terminal (accepting) states $F \subseteq Q$.

Graphically, transitions are denoted as $a, Z/\alpha$ where $a \in \Sigma$ is the input, $Z \in \Gamma$ is the top stack element, and $\alpha \in \Gamma^*$ replaces Z in the new stack.

PDA

Definition 9

The configuration of a PDA M is a triple (q, w, α) where $q \in Q$ is its state, $w \in \Sigma^*$ is its remaining input, and $\alpha \in \Gamma^*$ is its stack.

PDA

Definition 9

The configuration of a PDA M is a triple (q, w, α) where $q \in Q$ is its state, $w \in \Sigma^*$ is its remaining input, and $\alpha \in \Gamma^*$ is its stack.

The initial configuration of M on input $w \in \Sigma^*$ is (q_0, w, Z_0).

PDA

Definition 9

The configuration of a PDA M is a triple (q, w, α) where $q \in Q$ is its state, $w \in \Sigma^*$ is its remaining input, and $\alpha \in \Gamma^*$ is its stack.

The initial configuration of M on input $w \in \Sigma^*$ is (q_0, w, Z_0).

Definition 10

The transition relation of a PDA M is

$$\begin{aligned} (q, aw, Z\alpha) \rightarrow_M (q', w, \beta\alpha) &\quad \text{if } (q', \beta) \in \delta(q, a, Z) \\ (q, w, Z\alpha) \rightarrow_M (q', w, \beta\alpha) &\quad \text{if } (q', \beta) \in \delta(q, \epsilon, Z). \end{aligned}$$

PDA

Definition 11

PDA M accepts $w \in \Sigma^*$ with final state if

$$(q_0, w, Z_0) \rightarrow^*_M (f, \epsilon, \gamma) \quad \text{for } f \in F, \gamma in \Gamma^*.$$

So, $L_F(M) = \{w \in \Sigma^* \mid \exists f \in F, \gamma \in \Gamma^*.\ (q_0, w, Z_0) \rightarrow^*_M (f, \epsilon, \gamma)\}$.

PDA

Definition 11

PDA M accepts $w \in \Sigma^*$ with final state if

$$(q_0, w, Z_0) \to_M^* (f, \epsilon, \gamma) \quad \text{for } f \in F, \gamma in \Gamma^*.$$

So, $L_F(M) = \{w \in \Sigma^* \mid \exists f \in F, \gamma \in \Gamma^*. \ (q_0, w, Z_0) \to_M^* (f, \epsilon, \gamma)\}$.

Definition 12

PDA M accepts $w \in \Sigma^*$ with empty stack if

$$(q_0, w, Z_0) \to_M^* (q, \epsilon, \epsilon) \quad \text{for } q \in Q.$$

So, $L_\epsilon(M) = \{w \in \Sigma^* \mid \exists q \in Q. \ (q_0, w, Z_0) \to_M^* (q, \epsilon, \epsilon)\}$.

PDA

Definition 11

PDA M accepts $w \in \Sigma^*$ with final state if

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \quad \text{for } f \in F, \gamma in \Gamma^*.$$

So, $L_F(M) = \{w \in \Sigma^* \mid \exists f \in F, \gamma \in \Gamma^*. \ (q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma)\}$.

Definition 12

PDA M accepts $w \in \Sigma^*$ with empty stack if

$$(q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon) \quad \text{for } q \in Q.$$

So, $L_\epsilon(M) = \{w \in \Sigma^* \mid \exists q \in Q. \ (q_0, w, Z_0) \rightarrow_M^* (q, \epsilon, \epsilon)\}$.

Both accepting conditions are equally powerful.

Lemmas

Lemma 13 (Extension Lemma)

Every derivation may occur as a sub-derivation of a larger derivation

Lemmas

Lemma 13 (Extension Lemma)

Every derivation may occur as a sub-derivation of a larger derivation:

$$(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta).$$

Lemmas

Lemma 13 (Extension Lemma)

Every derivation may occur as a sub-derivation of a larger derivation:

$$(q, u, \alpha) \rightarrow^n_M (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow^n_M (q', u'v, \alpha'\beta).$$

Lemma 14 (Decomposition Lemma)

Every derivation that empties the stack can be divided into sub-derivations that each remove a single symbol from the stack

Lemmas

Lemma 13 (Extension Lemma)

Every derivation may occur as a sub-derivation of a larger derivation:

$$(q, u, \alpha) \to_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \to_M^n (q', u'v, \alpha'\beta).$$

Lemma 14 (Decomposition Lemma)

Every derivation that empties the stack can be divided into sub-derivations that each remove a single symbol from the stack: Given $(q, w, Z_1 \ldots Z_k) \to_M^n (q', \epsilon, \epsilon)$

Lemmas

Lemma 13 (Extension Lemma)

Every derivation may occur as a sub-derivation of a larger derivation:

$$(q, u, \alpha) \to_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \to_M^n (q', u'v, \alpha'\beta).$$

Lemma 14 (Decomposition Lemma)

Every derivation that empties the stack can be divided into sub-derivations that each remove a single symbol from the stack: Given $(q, w, Z_1 \ldots Z_k) \to_M^n (q', \epsilon, \epsilon)$*,*
then $\forall i \in [1, k].\ \exists u_i, p_i, n_i$ *such that*

$$(p_{i-1}, u_i, Z_i) \to_M^{n_i} (p_i, \epsilon, \epsilon)$$

with $w = u_1 \ldots u_k$*,* $q = p_0$*,* $q_k = p_k$*, and* $n = \sum_{i=1}^{k} n_i$*.*

CFG $\to$ PDA

Given CFG $G = (V, \Sigma, P, S)$,

CFG $\rightarrow$ PDA

Given CFG $G = (V, \Sigma, P, S)$,

1. bring all productions into the form

$$A \rightarrow bB_1 \dots B_k \quad \text{for } b \in \Sigma \cup \{\epsilon\}$$

CFG $\rightarrow$ PDA

Given CFG $G = (V, \Sigma, P, S)$,

1. bring all productions into the form

$$A \rightarrow bB_1 \dots B_k \quad \text{for } b \in \Sigma \cup \{\epsilon\}$$

2. define the PDA $M = (\{q\}, \Sigma, V, q, S, \delta)$

CFG $\rightarrow$ PDA

Given CFG $G = (V, \Sigma, P, S)$,

1. bring all productions into the form

$$A \rightarrow bB_1 \dots B_k \quad \text{for } b \in \Sigma \cup \{\epsilon\}$$

2. define the PDA $M = (\{q\}, \Sigma, V, q, S, \delta)$ with

$$\delta(q, b, A) = \{(q, \beta) \mid (A \rightarrow b\beta) \in P\}.$$

CFG $\to$ PDA

Given CFG $G = (V, \Sigma, P, S)$,

1. bring all productions into the form

$$A \to bB_1 \dots B_k \quad \text{for } b \in \Sigma \cup \{\epsilon\}$$

2. define the PDA $M = (\{q\}, \Sigma, V, q, S, \delta)$ with

$$\delta(q, b, A) = \{(q, \beta) \mid (A \to b\beta) \in P\}.$$

Then, $L(G) = L_\epsilon(M)$.

PDA $\rightarrow$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$,

PDA $\rightarrow$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

PDA $\to$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

PDA $\rightarrow$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

We define the productions P as

- $\forall q \in Q.\ S \rightarrow [q_0, Z_0, q]$

PDA $\to$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

We define the productions P as

- $\forall q \in Q.\ S \to [q_0, Z_0, q]$ and
- $\forall (r_0, Z_1 \dots Z_k) \in \delta(q, b, Z).\ \forall r_1, \dots, r_k \in Q.$

PDA $\to$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

We define the productions P as

- $\forall q \in Q.\ S \to [q_0, Z_0, q]$ and
- $\forall (r_0, Z_1 \dots Z_k) \in \delta(q, b, Z).\ \forall r_1, \dots, r_k \in Q.$

$$[q, Z, r_k] \to b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k].$$

PDA $\rightarrow$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

We define the productions P as

- $\forall q \in Q.\ S \rightarrow [q_0, Z_0, q]$ and
- $\forall (r_0, Z_1 \ldots Z_k) \in \delta(q, b, Z).\ \forall r_1, \ldots, r_k \in Q.$

$$[q, Z, r_k] \rightarrow b[r_0, Z_1, r_1][r_1, Z_2, r_2] \ldots [r_{k-1}, Z_k, r_k].$$

We observe that

$$[q, Z, r_k] \rightarrow_G^* w \iff (q, w, Z) \rightarrow_M^* (r_k, \epsilon, \epsilon).$$

PDA $\rightarrow$ CFG

Given PDA $G = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$, define CFG $G = (V, \Sigma, P, S)$.

We define $V = Q \times \Gamma \times Q \cup \{S\}$ where each $[q, Z, p] \in V$ describes all possibilities of going from state $q \in Q$ to state $p \in Q$ while $Z \in \Gamma$ is the top stack element.

We define the productions P as

- $\forall q \in Q.\ S \rightarrow [q_0, Z_0, q]$ and
- $\forall (r_0, Z_1 \dots Z_k) \in \delta(q, b, Z).\ \forall r_1, \dots, r_k \in Q.$

$$[q, Z, r_k] \rightarrow b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k].$$

We observe that

$$[q, Z, r_k] \rightarrow_G^* w \iff (q, w, Z) \rightarrow_M^* (r_k, \epsilon, \epsilon).$$

So, $L(G) = L_\epsilon(M)$.

Closure Properties

Theorem 15

Given the context-free languages L, L_1, L_2, then the following are also centext-free languages:

Closure Properties

Theorem 15

Given the context-free languages L, L_1, L_2, then the following are also centext-free languages:

- $L_1 L_2$

Closure Properties

Theorem 15

Given the context-free languages L, L_1, L_2*, then the following are also centext-free languages:*

- $L_1 L_2$;
- $L_1 \cup L_2$

Closure Properties

Theorem 15

Given the context-free languages L, L_1, L_2, then the following are also centext-free languages:

- $L_1 L_2$;
- $L_1 \cup L_2$; *and*
- L^*.

Closure Properties

Theorem 15

Given the context-free languages L, L_1, L_2, then the following are also centext-free languages:

- $L_1 L_2$;
- $L_1 \cup L_2$; *and*
- L^*.

Theorem 16

Given the deterministic context-free language L, then $\bar{L}$ is deterministic context-free.

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ *be context-free.*

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ *be context-free. Then there exists some* $n > 0$ *such that every* $z \in L$ *with* $|z| \geq n$ *can be decomposed into* $z = uvwxy$

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ *be context-free. Then there exists some* $n > 0$ *such that every* $z \in L$ *with* $|z| \geq n$ *can be decomposed into* $z = uvwxy$ *such that*

- $vx \neq \epsilon$

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ be context-free. Then there exists some $n > 0$ such that every $z \in L$ with $|z| \geq n$ can be decomposed into $z = uvwxy$ such that

- $vx \neq \epsilon$;
- $|vwx| \leq n$

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ be context-free. Then there exists some $n > 0$ such that every $z \in L$ with $|z| \geq n$ can be decomposed into $z = uvwxy$ such that

- $vx \neq \epsilon$;
- $|vwx| \leq n$; *and*
- $\forall i \geq 0.\ uv^iwx^iy \in L$.

Pumping Lemma

Lemma 17 (Pumping Lemma for context-free languages)

Let $L \subseteq \Sigma^*$ *be context-free. Then there exists some* $n > 0$ *such that every* $z \in L$ *with* $|z| \geq n$ *can be decomposed into* $z = uvwxy$ *such that*

- $vx \neq \epsilon$;
- $|vwx| \leq n$; *and*
- $\forall i \geq 0.\ uv^i wx^i y \in L$.

A necessary condition for context-free languages.

Pumping Lemma

Example 18 (proof structure)

Assume L is context-free.
Let $n > 0$ be a Pumping Lemma number.

Pumping Lemma

Example 18 (proof structure)

Assume L is context-free.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvwxy$ with $vx \neq \epsilon$ and $|vwx| \leq n$.

Pumping Lemma

Example 18 (proof structure)

Assume L is context-free.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvwxy$ with $vx \neq \epsilon$ and $|vwx| \leq n$.
Then, $\forall i \geq 0.\ uv^iwx^iy \in L$.

Pumping Lemma

Example 18 (proof structure)

Assume L is context-free.
Let $n > 0$ be a Pumping Lemma number.
Choose $z \in L$ with $|z| \geq n$.
Define $z = uvwxy$ with $vx \neq \epsilon$ and $|vwx| \leq n$.
Then, $\forall i \geq 0.\ uv^i wx^i y \in L$.
Now, use the last statement to find a contradiction separating all possible cases for v and x.