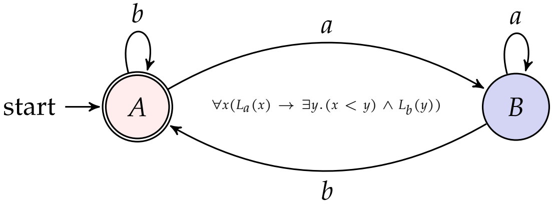
CS 208: Automata Theory and Logic

Lecture 6: Context-Free Grammar

Ashutosh Trivedi



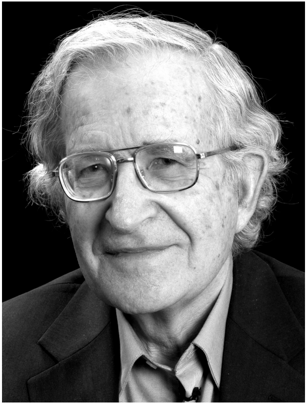
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

Context-Free Grammars

Pushdown Automata

Properties of CFLs

Context-Free Grammars



Noam Chomsky
(linguist, philosopher, logician, and activist)

" A grammar can be regarded as a device that enumerates the sentences of a language. We study a sequence of restrictions that limit grammars first to Turing machines, then to two types of systems from which a phrase structure description of a generated language can be drawn, and finally to finite state Markov sources (finite automata). "

Grammars

A (formal) grammar consists of

1. A finite set of rewriting rules of the form

   $$\phi \to \psi$$

   where ϕ and ψ are strings of symbols.
2. A special "initial" symbol S (S standing for sentence);
3. A finite set of symbols stand for "words" of the language called terminal vocabulary;
4. Other symbols stand for "phrases" and are called non-terminal vocabulary.

Grammars

A (formal) grammar consists of

1. A finite set of rewriting rules of the form

$$\phi \to \psi$$

   where ϕ and ψ are strings of symbols.
2. A special "initial" symbol S (S standing for sentence);
3. A finite set of symbols stand for "words" of the language called terminal vocabulary;
4. Other symbols stand for "phrases" and are called non-terminal vocabulary.

Given such a grammar, a valid sentence can be generated by

1. starting from the initial symbol S,
2. applying one of the rewriting rules to form a new string ϕ by applying a rule $S \to \phi_1$,
3. and apply another rule to form a new string ϕ_2 and so on,
4. until we reach a string ϕ_n that consists only of terminal symbols.

Examples

Consider the grammar

$$S \rightarrow AB \quad (1)$$
$$A \rightarrow C \quad (2)$$
$$CB \rightarrow Cb \quad (3)$$
$$C \rightarrow a \quad (4)$$

where $\{a, b\}$ are terminals, and $\{S, A, B, C\}$ are non-terminals.

Examples

Consider the grammar

$$
\begin{aligned}
S &\to AB & (1)\\
A &\to C & (2)\\
CB &\to Cb & (3)\\
C &\to a & (4)
\end{aligned}
$$

where $\{a, b\}$ are terminals, and $\{S, A, B, C\}$ are non-terminals.
We can derive the phrase "ab" from this grammar in the following way:

$$
\begin{aligned}
S &\to AB, \text{ from (1)}\\
&\to CB, \text{ from (2)}\\
&\to Cb, \text{ from (3)}\\
&\to ab, \text{ from (4)}
\end{aligned}
$$

Examples

Consider the grammar

$$
\begin{aligned}
S &\rightarrow NounPhrase \quad VerbPhrase && (5)\\
NounPhrase &\rightarrow SingularNoun && (6)\\
SingularNoun \quad VerbPhrase &\rightarrow SingularNoun \quad comes && (7)\\
SingularNoun &\rightarrow John && (8)
\end{aligned}
$$

We can derive the phrase "John comes" from this grammar in the following way:

$$
\begin{aligned}
S &\rightarrow NounPhrase \quad VerbPhrase, \text{ from (1)}\\
&\rightarrow SingularNoun \quad VerbPhrase, \text{ from (2)}\\
&\rightarrow SingularNoun \quad comes, \text{ from (3)}\\
&\rightarrow John \quad comes, \text{ from (4)}
\end{aligned}
$$

Types of Grammars

Depending on the rewriting rules we can characterize the grammars in the following four types:

1. type 0 grammars with no restriction on rewriting rules;
2. type 1 grammars have the rules of the form
$$\alpha A \beta \to \alpha \gamma \beta$$
where A is a nonterminal, α, β, γ are strings of terminals and nonterminals, and γ is non empty.
3. type 2 grammars have the rules of the form
$$A \to \gamma$$
where A is a nonterminal, and γ is a string (potentially empty) of terminals and nonterminals.
4. type 3 grammars have the rules of the form
$$A \to aB \text{ or } A \to a$$
where A, B are nonterminals, and a is a string (potentially empty) of terminals.

Types of Grammars

Depending on the rewriting rules we can characterize the grammars in the following four types:

1. Unrestricted grammars with no restriction on rewriting rules;
2. Context-sensitive grammars have the rules of the form

$$\alpha A \beta \to \alpha \gamma \beta$$

   where A is a nonterminal, α, β, γ are strings of terminals and nonterminals, and γ is non empty.
3. Context-free grammars have the rules of the form

$$A \to \gamma$$

   where A is a nonterminal, and γ is a string (potentially empty) of terminals and nonterminals.
4. Regular grammars have the rules of the form

$$A \to aB \text{ or } A \to a$$

   where A, B are nonterminals, and a is a string (potentially empty) of terminals.

Types of Grammars

Depending on the rewriting rules we can characterize the grammars in the following four types:

1. Unrestricted grammars with no restriction on rewriting rules;
2. Context-sensitive grammars have the rules of the form

   $$\alpha A \beta \rightarrow \alpha \gamma \beta$$

   where A is a nonterminal, α, β, γ are strings of terminals and nonterminals, and γ is non empty.
3. Context-free grammars have the rules of the form

   $$A \rightarrow \gamma$$

   where A is a nonterminal, and γ is a string (potentially empty) of terminals and nonterminals.
4. Regular grammars have the rules of the form

   $$A \rightarrow aB \text{ or } A \rightarrow a$$

   where A, B are nonterminals, and a is a string (potentially empty) of terminals. (also left-linear grammars)

Do regular grammars capture regular languages?

– Regular grammars to finite automata
– Finite automata to regular grammars

Context-Free Languages: Syntax

Definition (Context-Free Grammar)

A context-free grammar is a tuple $G = (V, T, P, S)$ where

- V is a finite set of variables (nonterminals, nonterminals vocabulary);
- T is a finite set of terminals (letters);
- $P \subseteq V \times (V \cup T)^*$ is a finite set of rewriting rules called productions,
  - We write $A \to \beta$ if $(A, \beta) \in P$;
- $S \in V$ is a distinguished start or “sentence” symbol.

Context-Free Languages: Syntax

Definition (Context-Free Grammar)

A context-free grammar is a tuple $G = (V, T, P, S)$ where

- V is a finite set of variables (nonterminals, nonterminals vocabulary);
- T is a finite set of terminals (letters);
- $P \subseteq V \times (V \cup T)^*$ is a finite set of rewriting rules called productions,
  - We write $A \to \beta$ if $(A, \beta) \in P$;
- $S \in V$ is a distinguished start or "sentence" symbol.

Example: $G_{0^n1^n} = (V, T, P, S)$ where

- $V = \{S\}$;
- $T = \{0, 1\}$;
- P is defined as

$$\begin{aligned} S &\to \varepsilon \\ S &\to 0S1 \end{aligned}$$

- $S = S$.

Context-Free Languages: Semantics

Derivation:

- Let $G = (V, T, P, S)$ be a context-free grammar.
- Let $\alpha A \beta$ be a string in $(V \cup T)^* V (V \cup T)^*$
- We say that $\alpha A \beta$ yields the string $\alpha\gamma\beta$, and we write $\alpha A \beta \Rightarrow \alpha\gamma\beta$ if

$$A \rightarrow \gamma \text{ is a production rule in } G.$$

- For strings $\alpha, \beta \in (V \cup T)^*$, we say that α derives β and we write $\alpha \overset{*}{\Rightarrow} \beta$ if there is a sequence $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup T)^*$ s.t.

$$\alpha \rightarrow \alpha_1 \rightarrow \alpha_2 \cdots \alpha_n \rightarrow \beta.$$

Context-Free Languages: Semantics

Derivation:

- Let $G = (V, T, P, S)$ be a context-free grammar.
- Let $\alpha A \beta$ be a string in $(V \cup T)^* V (V \cup T)^*$
- We say that $\alpha A \beta$ yields the string $\alpha \gamma \beta$, and we write $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if

$$A \to \gamma \text{ is a production rule in } G.$$

- For strings $\alpha, \beta \in (V \cup T)^*$, we say that α derives β and we write $\alpha \stackrel{*}{\Rightarrow} \beta$ if there is a sequence $\alpha_1, \alpha_2, \ldots, \alpha_n \in (V \cup T)^*$ s.t.

$$\alpha \to \alpha_1 \to \alpha_2 \cdots \alpha_n \to \beta.$$

Definition (Context-Free Grammar: Semantics)

The language $L(G)$ accepted by a context-free grammar $G = (V, T, P, S)$ is the set

$$L(G) = \{w \in T^* \ : \ S \stackrel{*}{\Rightarrow} w\}.$$

CFG: Example

Recall $G_{0^n1^n} = (V, T, P, S)$ where

- $V = \{S\}$;
- $T = \{0, 1\}$;
- P is defined as

$$\begin{aligned} S &\rightarrow \varepsilon \\ S &\rightarrow 0S1 \end{aligned}$$

- $S = S$.

The string $000111 \in L(G_{0^n1^n})$, i.e. $S \stackrel{*}{\Rightarrow} 000111$ as

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111.$$

Prove that $0^n 1^n$ is accepted by the grammar $G_{0^n 1^n}$.

The proof is in two parts.

- First show that every string w of the form $0^n 1^n$ can be derived from S using induction over w.
- Then, show that for every string $w \in \{0, 1\}^*$ derived from S, we have that w is of the form $0^n 1^n$.

CFG: Example

Consider the following grammar $G = (V, T, P, S)$ where

- $V = \{E, I\}$; $T = \{a, b, 0, 1\}$; $S = E$; and
- P is defined as

$$\begin{aligned} E &\rightarrow I \mid E + E \mid E * E \mid (E) \\ I &\rightarrow a \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

The string $(a1 + b0 * a1) \in L(G)$, i.e. $E \stackrel{*}{\Rightarrow} (a1 + b0 * a1)$ as

$$E \Rightarrow (E) \Rightarrow (E + E) \Rightarrow (I + E) \Rightarrow (I1 + E) \Rightarrow (a1 + E) \stackrel{*}{\Rightarrow} (a1 + b0 * a1).$$

CFG: Example

Consider the following grammar $G = (V, T, P, S)$ where

- $V = \{E, I\}$; $T = \{a, b, 0, 1\}$; $S = E$; and
- P is defined as

$$
\begin{aligned}
E &\rightarrow I \mid E + E \mid E * E \mid (E) \\
I &\rightarrow a \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

The string $(a1 + b0 * a1) \in L(G)$, i.e. $E \overset{*}{\Rightarrow} (a1 + b0 * a1)$ as

$$
\begin{aligned}
E &\Rightarrow (E) \Rightarrow (E + E) \Rightarrow (I + E) \Rightarrow (I1 + E) \Rightarrow (a1 + E) \overset{*}{\Rightarrow} (a1 + b0 * a1). \\
E &\Rightarrow (E) \Rightarrow (E + E) \Rightarrow (E + E * E) \Rightarrow (E + E * I) \overset{*}{\Rightarrow} (a1 + b0 * a1).
\end{aligned}
$$

CFG: Example

Consider the following grammar $G = (V, T, P, S)$ where

- $V = \{E, I\}$; $T = \{a, b, 0, 1\}$; $S = E$; and
- P is defined as

$$\begin{aligned} E &\rightarrow I \mid E + E \mid E * E \mid (E) \\ I &\rightarrow a \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

The string $(a1 + b0 * a1) \in L(G)$, i.e. $E \stackrel{*}{\Rightarrow} (a1 + b0 * a1)$ as

$$\begin{aligned} E &\Rightarrow (E) \Rightarrow (E + E) \Rightarrow (I + E) \Rightarrow (I1 + E) \Rightarrow (a1 + E) \stackrel{*}{\Rightarrow} (a1 + b0 * a1). \\ E &\Rightarrow (E) \Rightarrow (E + E) \Rightarrow (E + E * E) \Rightarrow (E + E * I) \stackrel{*}{\Rightarrow} (a1 + b0 * a1). \end{aligned}$$

Leftmost and rightmost derivations:

1. Derivations are not unique
2. Leftmost and rightmost derivations
3. Define $\Rightarrow_{lm}$ and $\Rightarrow_{rm}$ in straightforward manner.
4. Find leftmost and rightmost derivations of $(a1 + b0 * a1)$.

Exercise

Consider the following grammar:

$$
\begin{aligned}
S &\rightarrow AS \mid \varepsilon. \\
S &\rightarrow aa \mid ab \mid ba \mid bb
\end{aligned}
$$

Give leftmost and rightmost derivations of the string *aabbba*.

Parse Trees

- A CFG provide a structure to a string
- Such structure assigns meaning to a string, and hence a unique structure is really important in several applications, e.g. compilers
- Parse trees are a successful data-structures to represent and store such structures

Parse Trees

- A CFG provide a structure to a string
- Such structure assigns meaning to a string, and hence a unique structure is really important in several applications, e.g. compilers
- Parse trees are a successful data-structures to represent and store such structures
- Let's review the Tree terminology:
  - A tree is a directed acyclic graph (DAG) where every node has at most incoming edge.

Parse Trees

- A CFG provide a structure to a string
- Such structure assigns meaning to a string, and hence a unique structure is really important in several applications, e.g. compilers
- Parse trees are a successful data-structures to represent and store such structures
- Let's review the Tree terminology:
  - A tree is a directed acyclic graph (DAG) where every node has at most incoming edge.
  - Edge relationship as parent-child relationship
  - Every node has at most one parent, and zero or more children
  - We assume an implicit order on children ("from left-to-right")
  - There is a distinguished root node with no parent, while all other nodes have a unique parent
  - There are some nodes with no children called leaves—other nodes are called interior nodes
  - Ancestor and descendent relationships are closure of parent and child relationships, resp.

Parse Tree

Given a grammar $G = (V, T, P, S)$, the parse trees associated with G has the following properties:

1. Each interior node is labeled by a variable in V.
2. Each leaf is either a variable, terminal, or ε. However, if a leaf is ε it is the only child of its parent.
3. If an interior node is labeled A and has children labeled $X_1, X_2, \ldots, X_k$ from left-to-right, then

$$A \to X_1 X_2 \ldots X_k$$

is a production is P. Only time X_i can be ε is when it is the only child of its parent, i.e. corresponding to the production $A \to \varepsilon$.

Reading exercise

– Give parse tree representation of previous derivation exercises.

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?
- Are parse trees unique?

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?
- Are parse trees unique?
- Answer is no. A grammar is called ambiguous if there is at least one string with two different leftmost (or rightmost) derivations.

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?
- Are parse trees unique?
- Answer is no. A grammar is called ambiguous if there is at least one string with two different leftmost (or rightmost) derivations.
- There are some inherently ambiguous languages.

$$L = \{a^n b^n c^m d^m \; : \; n, m \geq 1\} \cup \{a^n b^m c^n d^m \; : \; n, m \geq 1\}.$$

Write a grammar accepting this language. Show that the string $a^2 b^2 c^2 d^2$ has two leftmost derivations.

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?
- Are parse trees unique?
- Answer is no. A grammar is called ambiguous if there is at least one string with two different leftmost (or rightmost) derivations.
- There are some inherently ambiguous languages.

$$L = \{a^n b^n c^m d^m \ : \ n, m \geq 1\} \cup \{a^n b^m c^n d^m \ : \ n, m \geq 1\}.$$

  Write a grammar accepting this language. Show that the string $a^2b^2c^2d^2$ has two leftmost derivations.
- There is no algorithm to decide whether a grammar is ambiguous.

Reading exercise

- Give parse tree representation of previous derivation exercises.
- Are leftmost-derivation and rightmost derivation parse-trees always different?
- Are parse trees unique?
- Answer is no. A grammar is called ambiguous if there is at least one string with two different leftmost (or rightmost) derivations.
- There are some inherently ambiguous languages.

$$L = \{a^n b^n c^m d^m \; : \; n, m \geq 1\} \cup \{a^n b^m c^n d^m \; : \; n, m \geq 1\}.$$

  Write a grammar accepting this language. Show that the string $a^2 b^2 c^2 d^2$ has two leftmost derivations.
- There is no algorithm to decide whether a grammar is ambiguous.
- What does that mean from application side?

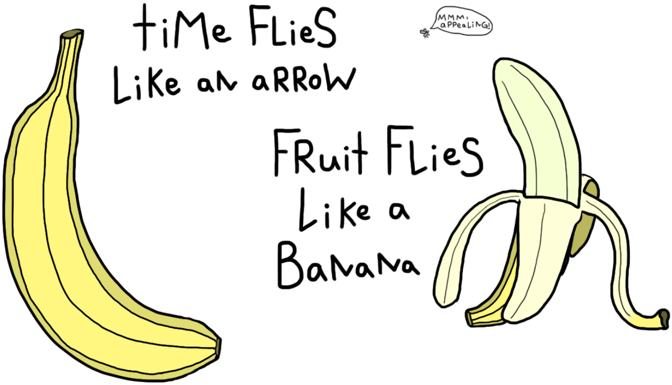
In-class Quiz

Write CFGs for the following languages:

1. Strings ending with a 0
2. Strings containing even number of 1's
3. palindromes over $\{0, 1\}$
4. $L = \{a^i b^j : i \leq 2j\}$ or $L = \{a^i b^j : i < 2j\}$ or $L = \{a^i b^j : i \neq 2j\}$
5. $L = \{a^i b^j c^k : i = k\}$
6. $L = \{a^i b^j c^k : i = j\}$
7. $L = \{a^i b^j c^k : i = j + k\}$.
8. $L = \{w \in \{0, 1\}^* : |w|_a = |w|_b\}$.
9. Closure under union, concatenation, and Kleene star
10. Closure under substitution, homomorphism, and reversal

Syntactic Ambiguity in English

Syntactic Ambiguity in English



—Anthony G. Oettinger

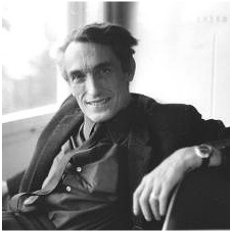
Context-Free Grammars

Pushdown Automata

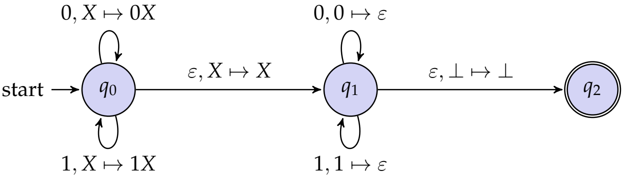
Properties of CFLs

Pushdown Automata

Anthony G. Oettinger

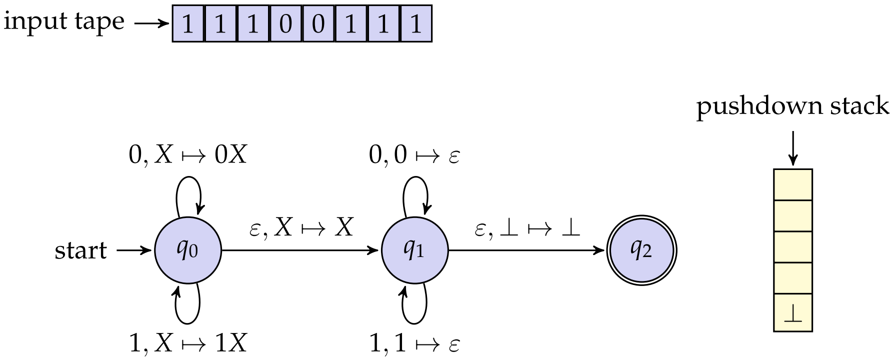


M. P. Schutzenberger

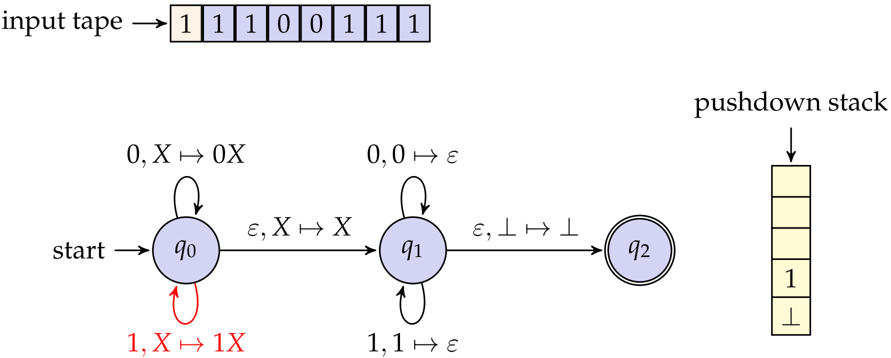


- Introduced independently by Anthony G. Oettinger in 1961 and by Marcel-Paul Schützenberger in 1963
- Generalization of ε-NFA with a "stack-like" storage mechanism
- Precisely capture context-free languages
- Deterministic version is not as expressive as non-deterministic one
- Applications in program verification and syntax analysis

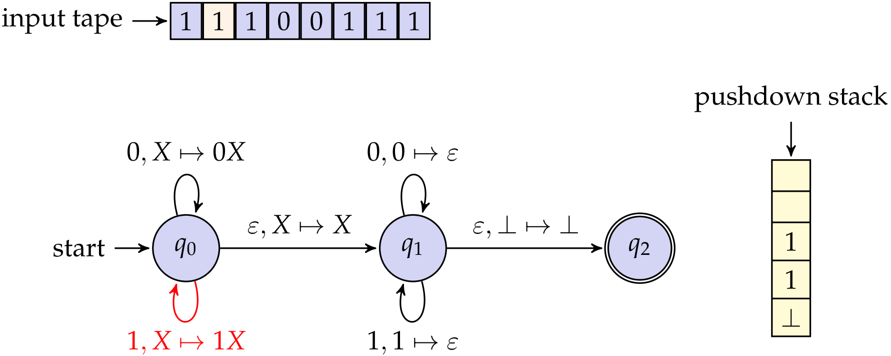
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



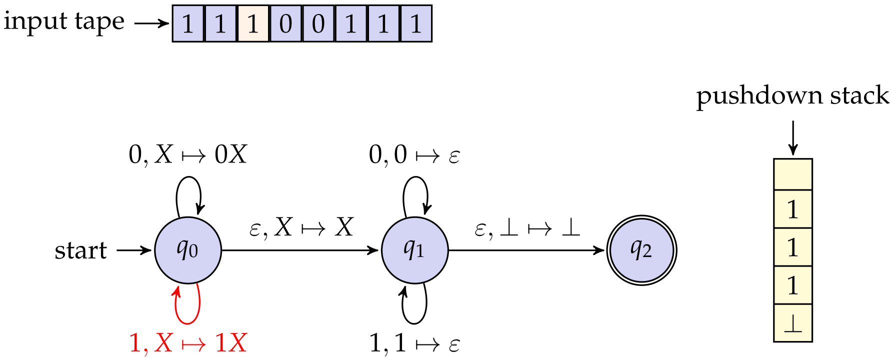
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



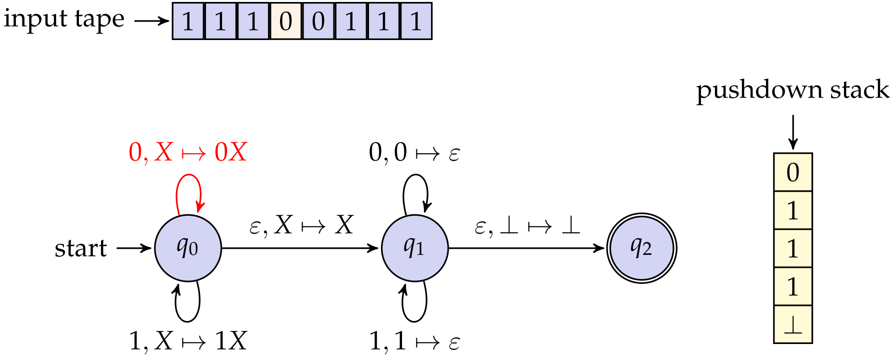
Example 1: $L = \{w\overline{w} \;:\; w \in \{0,1\}^*\}$



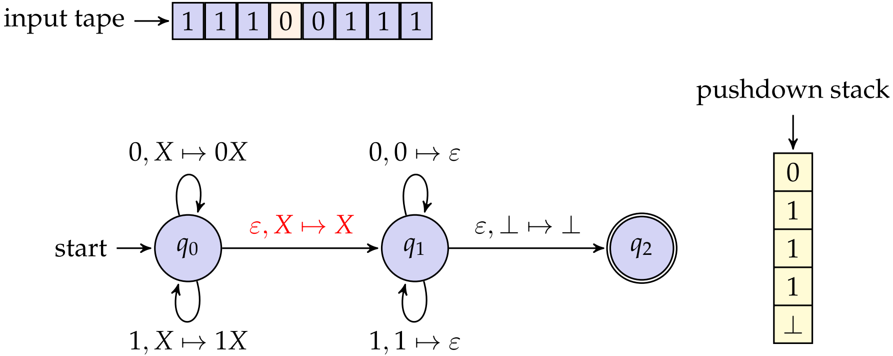
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



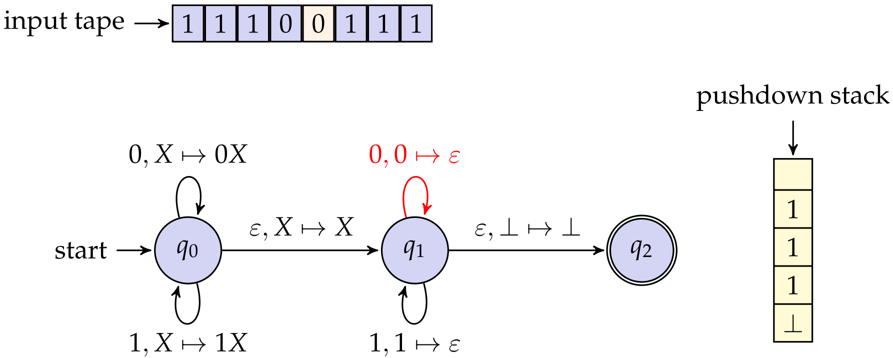
Example 1: $L = \{w\overline{w} \; : \; w \in \{0,1\}^*\}$



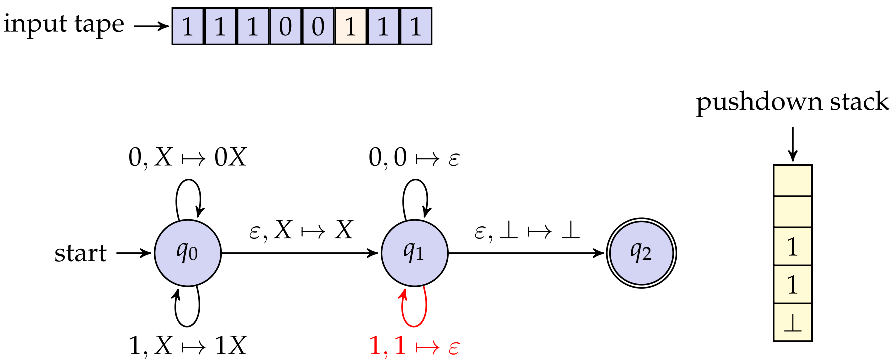
Example 1: $L = \{w\overline{w} \;:\; w \in \{0,1\}^*\}$



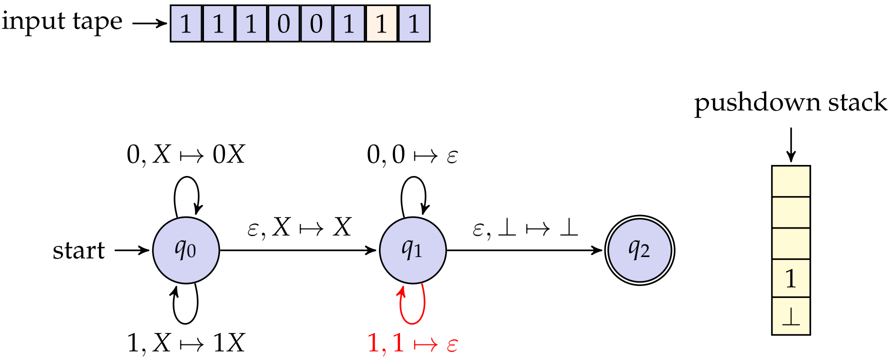
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



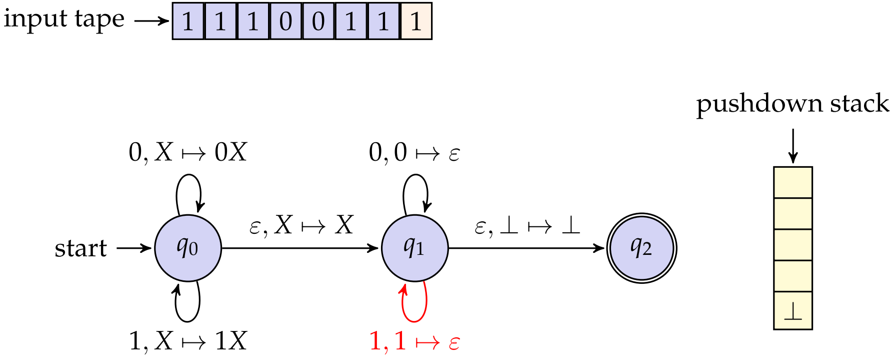
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



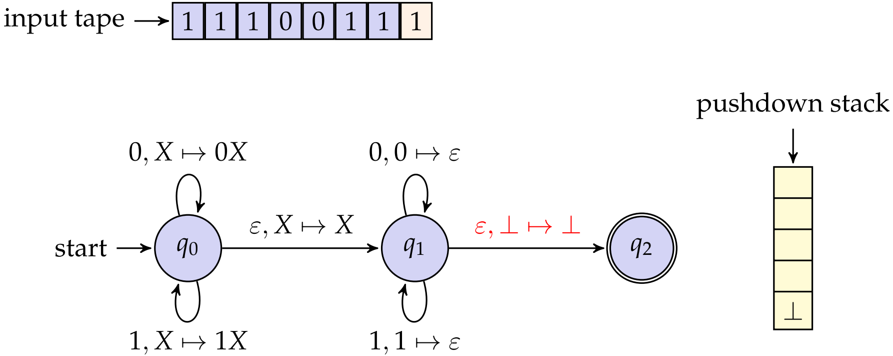
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



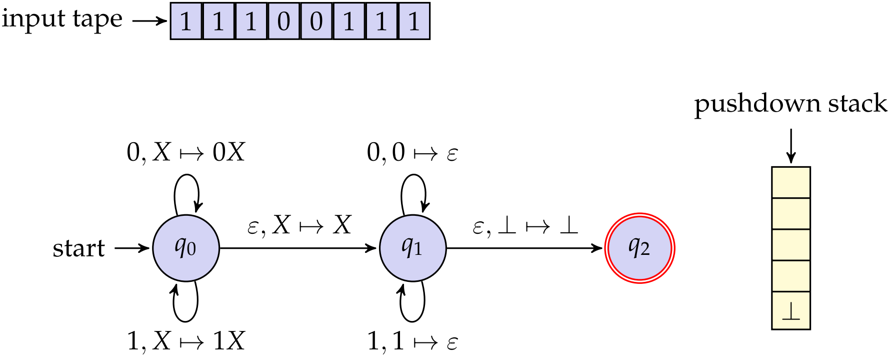
Example 1: $L = \{w\overline{w} \ : \ w \in \{0,1\}^*\}$



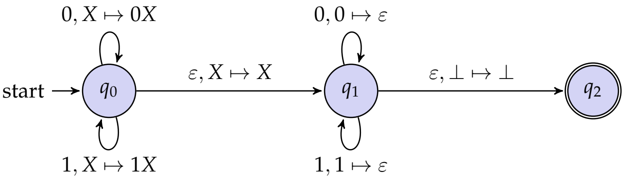
Example 1: $L = \{w\overline{w} \;:\; w \in \{0,1\}^*\}$



Example 1: $L = \{w\overline{w} \;:\; w \in \{0,1\}^*\}$



Pushdown Automata



A pushdown automata is a tuple $(Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ where:

- Q is a finite set called the states;
- Σ is a finite set called the alphabet;
- Γ is a finite set called the stack alphabet;
- $\delta : Q \times \Sigma \times \Gamma \to 2^{Q \times \Gamma^*}$ is the transition function;
- $q_0 \in Q$ is the start state;
- $\bot \in \Gamma$ is the start stack symbol;
- $F \subseteq Q$ is the set of accepting states.

Semantics of a PDA

- Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ be a PDA.
- A configuration (or instantaneous description) of a PDA is a triple (q, w, γ) where
  - q is the current state,
  - w is the remaining input, and
  - $\gamma \in \Gamma^*$ is the stack contents, where written as concatenation of symbols form top-to-bottom.
- We define the operator $\vdash$ (derivation) such that if $(p, \alpha) \in \delta(q, a, X)$ then

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta),$$

  for all $w \in \Sigma^*$ and $\beta \in \Gamma^*$. The operator $\perp^*$ is defined as transitive closure of $\perp$ in straightforward manner.
- A run of a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ over an input word $w \in \Sigma^*$ is a sequence of configurations

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

  such that for every $0 \leq i < n-1$ we have that $(q_i, w_i, \beta_i) \vdash (q_{i+1}, w_{i+1}, \beta_{i+1})$ and $(q_0, w_0, \beta_0) = (q_0, w, \perp)$.

Semantics: acceptance via final states

1. We say that a run

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

   is accepted via final state if $q_n \in F$ and $w_n = \varepsilon$.
2. We say that a word w is accepted via final states if there exists a run of P over w that is accepted via final state.
3. We write $L(P)$ for the set of words accepted via final states.
4. In other words,

$$L(P) = \{w \;:\; (q_0, w, \bot) \vdash^* (q_n, \varepsilon, \beta) \text{ and } q_n \in F\}.$$

Semantics: acceptance via final states

1. We say that a run

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

   is accepted via final state if $q_n \in F$ and $w_n = \varepsilon$.
2. We say that a word w is accepted via final states if there exists a run of P over w that is accepted via final state.
3. We write $L(P)$ for the set of words accepted via final states.
4. In other words,

$$L(P) = \{w \,:\, (q_0, w, \bot) \vdash^* (q_n, \varepsilon, \beta) \text{ and } q_n \in F\}.$$

5. Example $L = \{w\overline{w} \,:\, w \in \{0,1\}^*\}$ with the notion of configuration, computation, run, and acceptance.

Semantics: acceptance via empty stack

1. We say that a run

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

   is accepted via empty stack if $\beta_n = \varepsilon$ and $w_n = \varepsilon$.
2. We say that a word w is accepted via empty stack if there exists a run of P over w that is accepted via empty stack.
3. We write $N(P)$ for the set of words accepted via empty stack.
4. In other words

$$N(P) = \{w \, : \, (q_0, w, \perp) \vdash^* (q_n, \varepsilon, \varepsilon)\}.$$

Semantics: acceptance via empty stack

1. We say that a run

$$(q_0, w_0, \beta_0), (q_1, w_1, \beta_1), \ldots, (q_n, w_n, \beta_n)$$

   is accepted via empty stack if $\beta_n = \varepsilon$ and $w_n = \varepsilon$.
2. We say that a word w is accepted via empty stack if there exists a run of P over w that is accepted via empty stack.
3. We write $N(P)$ for the set of words accepted via empty stack.
4. In other words

$$N(P) = \{w \ : \ (q_0, w, \perp) \vdash^* (q_n, \varepsilon, \varepsilon)\}.$$

Is $L(P) = N(P)$?

Equivalence of both notions

Theorem

For every language defind by a PDA with empty stack semantics, there exists a PDA that accept the same language with final state semantics, and vice-versa.

Proof.

- Final state to Empty stack
  - Add a new stack symbol, say $\perp'$, as the start stack symbol, and in the first transition replace it with $\perp\perp'$ before reading any symbol. (How? and Why?)
  - From every final state make a transition to a sink state that does not read the input but empties the stack including $\perp'$.

Equivalence of both notions

Theorem

For every language defind by a PDA with empty stack semantics, there exists a PDA that accept the same language with final state semantics, and vice-versa.

Proof.

- Final state to Empty stack
  - Add a new stack symbol, say $\bot'$, as the start stack symbol, and in the first transition replace it with $\bot\bot'$ before reading any symbol. (How? and Why?)
  - From every final state make a transition to a sink state that does not read the input but empties the stack including $\bot'$.
- Empty Stack to Final state
  - Replace the start stack symbol $\bot'$ and $\bot\bot'$ before reading any symbol. (Why?)
  - From every state make a transition to a new unique final state that does not read the input but removes the symbol $\bot'$.

□

Formal Construction: Empty stack to Final State

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$ be a PDA. We claim that the PDA $P' = (Q', \Sigma, \Gamma', \delta', q_0', \bot', F')$ is such that $N(P) = L(P')$, where

1. $Q' = Q \cup \{q_0'\} \cup \{q_F\}$
2. $\Gamma' = \Gamma \cup \{\bot'\}$
3. $F' = \{q_F\}$.
4. δ' is such that
   - $\delta'(q, a, X) = \delta(q, a, X)$ for all $q \in Q$ and $X \in \Gamma$,
   - $\delta'(q_0', \varepsilon, \bot') = \{(q_0, \bot\bot')\}$ and
   - $\delta'(q, \varepsilon, \bot') = \{(q_F, \bot')\}$ for all $q \in Q$.

Formal Construction: Final State to Empty Stack

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ be a PDA. We claim that the PDA $P' = (Q', \Sigma, \Gamma', \delta', q'_0, \bot')$ is such that $L(P) = N(P')$, where

1. $Q' = Q \cup \{q'_0\} \cup \{q_F\}$
2. $\Gamma' = \Gamma \cup \{\bot'\}$
3. δ' is such that
   - $\delta'(q, a, X) = \delta(q, a, X)$ for all $q \in Q$ and $X \in \Gamma$,
   - $\delta'(q'_0, \varepsilon, \bot') = \{(q_0, \bot\bot')\}$ and
   - $\delta'(q, \varepsilon, X) = \{(q_F, \varepsilon)\}$ for all $q \in Q$ and $X \in \Gamma$.
   - $\delta'(q_F, \varepsilon, X) = \{(q_F, \varepsilon)\}$ for all $X \in \Gamma$.

Expressive power of CFG and PDA

Theorem

A language is context-free if and only if some pushdown automaton accepts it.

Proof.

1. For an arbitrary CFG G give a PDA P_G such that $L(G) = L(P_G)$.

Expressive power of CFG and PDA

Theorem

A language is context-free if and only if some pushdown automaton accepts it.

Proof.

1. For an arbitrary CFG G give a PDA P_G such that $L(G) = L(P_G)$.
   - Leftmost derivation of a string using the stack
   - One state PDA accepting by empty stack
   - Proof via a simple induction over size of an accepting run of PDA

Expressive power of CFG and PDA

Theorem

A language is context-free if and only if some pushdown automaton accepts it.

Proof.

1. For an arbitrary CFG G give a PDA P_G such that $L(G) = L(P_G)$.
   - Leftmost derivation of a string using the stack
   - One state PDA accepting by empty stack
   - Proof via a simple induction over size of an accepting run of PDA
2. For an arbitrary PDA P give a CFG G_P such that $L(P) = L(G_P)$.

Expressive power of CFG and PDA

Theorem

A language is context-free if and only if some pushdown automaton accepts it.

Proof.

1. For an arbitrary CFG G give a PDA P_G such that $L(G) = L(P_G)$.
   - Leftmost derivation of a string using the stack
   - One state PDA accepting by empty stack
   - Proof via a simple induction over size of an accepting run of PDA
2. For an arbitrary PDA P give a CFG G_P such that $L(P) = L(G_P)$.
   - Modify the PDA to have the following properties such that each step is either a "push" or "pop", and has a single accepting state and the stack is emptied before accepting.
   - For every state pair of P define a variable A_{pq} in P_G generating strings such that PDA moves from state p to state q starting and ending with empty stack.
   - Three production rules

$$A_{pq} = aA_{rs}b \text{ and } A_{pq} = A_{pr}A_{rq} \text{ and } A_{pp} = \varepsilon.$$

From CFGs to PDAs

Given a CFG $G = (V, T, P, S)$ consider PDA $P_G = (\{q\}, T, V \cup T, \delta, q, S)$ s.t.:

- for every $a \in T$ we have

$$\delta(q, a, a) = (q, \varepsilon), \text{ and}$$

- for variable $A \in V$ we have that

$$\delta(q, \varepsilon, A) = \{(q, \beta) \ : \ A \to \beta \text{ is a production of } P\}.$$

Then $L(G) = N(P_G)$.

From CFGs to PDAs

Given a CFG $G = (V, T, P, S)$ consider PDA $P_G = (\{q\}, T, V \cup T, \delta, q, S)$ s.t.:

- for every $a \in T$ we have

$$\delta(q, a, a) = (q, \varepsilon), \text{ and}$$

- for variable $A \in V$ we have that

$$\delta(q, \varepsilon, A) = \{(q, \beta) \ : \ A \to \beta \text{ is a production of } P\}.$$

Then $L(G) = N(P_G)$.

Example. Give the PDA equivalent to the following grammar

$$\begin{aligned} I &\to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ E &\to I \mid E * E \mid E + E \mid (E). \end{aligned}$$

From CFGs to PDAs

Theorem

We have that $w \in N(P)$ if and only if $w \in L(G)$.

Proof.

– (If part). Suppose $w \in L(G)$. Then w has a leftmost derivation

$$S = \gamma_1 \Rightarrow_{lm} \gamma_2 \Rightarrow_{lm} \cdots \Rightarrow_{lm} \gamma_n = w.$$

It is straightforward to see that by induction on i that $(q, w, S) \vdash^* (q, y_i, \alpha_i)$ where $w = x_i y_i$ and $x_i \alpha_i = \gamma_i$.

□

From CFGs to PDAs

Theorem

We have that $w \in N(P)$ if and only if $w \in L(G)$.

Proof.

- (Only If part). Suppose $w \in N(P)$, i.e. $(q, w, S) \vdash^* (q, \varepsilon, \varepsilon)$.
  We show that if $(q, x, A) \vdash^* (q, \varepsilon, \varepsilon)$ then $A \Rightarrow^* x$ by induction over number of moves taken by P.
  - Base case. $x = \varepsilon$ and $(q, \varepsilon) \in \delta(q, \varepsilon, A)$. It follows that $A \to \varepsilon$ is a production in P.
  - inductive step. Let the first step be $A \to Y_1 Y_2 \dots Y_k$. Let $x_1 x_2 \dots x_k$ be the part of input to be consumed by the time $Y_1 \dots Y_k$ is popped out of the stack.
    It follows that $(q, x_i, Y_i) \vdash^* (q, \varepsilon, \varepsilon)$, and from inductive hypothesis we get that $Y_i \Rightarrow x_i$ if Y_i is a variable, and $Y_i = x_i$ is Y_i is a terminal. Hence, we conclude that $A \Rightarrow^* x$.

□

From PDAs to CFGs

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, \{q_F\})$ with restriction that every transition is either pushes a symbol or pops a symbol form the stack, i.e. $\delta(q, a, X)$ contains either (q', YX) or (q', ε).
Consider the grammar $G_p = (V, T, P, S)$ such that

- $V = \{A_{p,q} \ : \ p, q \in Q\}$
- $T = \Sigma$
- $S = A_{q_0, q_F}$
- and P has transitions of the following form:
  - $A_{q,q} \rightarrow \varepsilon$ for all $q \in Q$;
  - $A_{p,q} \rightarrow A_{p,r} \, A_{r,q}$ for all $p, q, r \in Q$,
  - $A_{p,q} \rightarrow a \, A_{r,s} \, b$ if $\delta(p, a, \varepsilon)$ contains (r, X) and $\delta(s, b, X)$ contains (q, ε).

From PDAs to CFGs

Given a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, \{q_F\})$ with restriction that every transition is either pushes a symbol or pops a symbol form the stack, i.e. $\delta(q, a, X)$ contains either (q', YX) or (q', ε).
Consider the grammar $G_p = (V, T, P, S)$ such that

- $V = \{A_{p,q} \; : \; p, q \in Q\}$
- $T = \Sigma$
- $S = A_{q_0,q_F}$
- and P has transitions of the following form:
  - $A_{q,q} \rightarrow \varepsilon$ for all $q \in Q$;
  - $A_{p,q} \rightarrow A_{p,r} \; A_{r,q}$ for all $p, q, r \in Q$,
  - $A_{p,q} \rightarrow a \; A_{r,s} \; b$ if $\delta(p, a, \varepsilon)$ contains (r, X) and $\delta(s, b, X)$ contains (q, ε).

We have that $L(G_p) = L(P)$.

From PDAs to CFGs

Theorem

If $A_{p,q} \Rightarrow^* x$ then x can bring the PDA P from state p on empty stack to state q on empty stack.

Proof.

We prove this theorem by induction on the number of steps in the derivation of x from $A_{p,q}$.

- Base case. If $A_{p,q} \Rightarrow^* x$ in one step, then the only rule that can generate a variable free string in one step is $A_{p,p} \rightarrow \varepsilon$.
- Inductive step. If $A_{p,q} \Rightarrow^* x$ in $n+1$ steps. The first step in the derivation must be $A_{p,q} \rightarrow A_{p,r}A_{r,q}$ or $A_{p,q} \rightarrow a\ A_{r,s}\ b$.
  - If it is $A_{p,q} \rightarrow A_{p,r}A_{r,q}$, then the string x can be broken into two parts x_1x_2 such that $A_{p,r} \Rightarrow^* x_1$ and $A_{r,q} \Rightarrow^* x_2$ in at most n steps. The theorem easily follows in this case.
  - If it is $A_{p,q} \rightarrow aA_{r,s}b$, then the string x can be broken as ayb such that $A_{r,s} \Rightarrow^* y$ in n steps. Notice that from p on reading a the PDA pushes a symbol X to stack, while it pops X in state s and goes to q.

□

From CFGs to PDAs

Theorem

If x can bring the PDA P from state p on empty stack to state q on empty stack then $A_{p,q} \Rightarrow^* x$.

Proof.

We prove this theorem by induction on the number of steps the PDA takes on x to go from p on empty stack to q on empty stack.

- Base case. If the computation has 0 steps that it begins and ends with the same state and reads ε from the tape. Note that $A_{p,p} \Rightarrow^* \varepsilon$ since $A_{p,p} \rightarrow \varepsilon$ is a rule in P.
- Inductive step. If the computation takes $n + 1$ steps. To keep the stack empty, the first step must be a "push" move, while the last step must be a "pop" move. There are two cases to consider:
  - The symbol pushed in the first step is the symbol popped in the last step.
  - The symbol pushed if the first step has been popped somewhere in the middle.

□

Context-Free Grammars

Pushdown Automata

Properties of CFLs

Deterministic Pushdown Automata

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ is deterministic if

- $\delta(q, a, X)$ has at most one member for every $q \in Q$, $a \in \Sigma$ or $a = \varepsilon$, and $X \in \Gamma$.
- If $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$ then $\delta(q, \varepsilon, X)$ must be empty.

Deterministic Pushdown Automata

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ is deterministic if

- $\delta(q, a, X)$ has at most one member for every $q \in Q$, $a \in \Sigma$ or $a = \varepsilon$, and $X \in \Gamma$.
- If $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$ then $\delta(q, \varepsilon, X)$ must be empty.

Example. $L = \{0^n 1^n \ : \ n \geq 1\}$.

Deterministic Pushdown Automata

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \bot, F)$ is deterministic if

- $\delta(q, a, X)$ has at most one member for every $q \in Q$, $a \in \Sigma$ or $a = \varepsilon$, and $X \in \Gamma$.
- If $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$ then $\delta(q, \varepsilon, X)$ must be empty.

Example. $L = \{0^n 1^n \; : \; n \geq 1\}$.

Theorem

Every regular language can be accepted by a deterministic pushdown automata that accepts by final states.

Deterministic Pushdown Automata

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ is deterministic if

- $\delta(q, a, X)$ has at most one member for every $q \in Q$, $a \in \Sigma$ or $a = \varepsilon$, and $X \in \Gamma$.
- If $\delta(q, a, X)$ is nonempty for some $a \in \Sigma$ then $\delta(q, \varepsilon, X)$ must be empty.

Example. $L = \{0^n 1^n \ : \ n \geq 1\}$.

Theorem

Every regular language can be accepted by a deterministic pushdown automata that accepts by final states.

Theorem (DPDA $\neq$ PDA)

There are some CFLs, for instance $\{w\overline{w}\}$ that can not be accepted by a DPDA.

Chomsky Normal Form

A Context-free grammar (V, T, P, S) is in Chomsky Normal Form if every rule is of the form

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a. \end{aligned}$$

where A, B, C are variables, and a is a nonterminal. Also, the start variable S must not appear on the right-side of any rule, and we also permit the rule $S \rightarrow \varepsilon$.

Theorem

Every context-free language is generated by a CFG in Chomsky normal form.

Chomsky Normal Form

A Context-free grammar (V, T, P, S) is in Chomsky Normal Form if every rule is of the form

$$\begin{array}{rcl} A & \rightarrow & BC \\ A & \rightarrow & a. \end{array}$$

where A, B, C are variables, and a is a nonterminal. Also, the start variable S must not appear on the right-side of any rule, and we also permit the rule $S \rightarrow \varepsilon$.

Theorem

Every context-free language is generated by a CFG in Chomsky normal form.

Reading Assignment: How to convert an arbitrary CFG to Chomsky Normal Form.

Pumping Lemma for CFLs

Theorem

For every context-free language L there exists a constant p (that depends on L) such that
for every string $z \in L$ of length greater or equal to p,
there is an infinite family of strings belonging to L.

Pumping Lemma for CFLs

Theorem

For every context-free language L there exists a constant p (that depends on L) such that
for every string $z \in L$ of length greater or equal to p,
there is an infinite family of strings belonging to L.

Why? *Think parse Trees!*

Pumping Lemma for CFLs

Theorem

For every context-free language L there exists a constant p (that depends on L) such that
for every string $z \in L$ of length greater or equal to p,
there is an infinite family of strings belonging to L.

Why? *Think parse Trees!*

Let L be a CFL. Then there exists a constant n such that if z is a string in L of length at least n, then we can write $z = uvwxy$ such that

- $|vwx| \leq n$
- $vx \neq \varepsilon$,
- *For all* $i \geq 0$ *the string* $uv^iwx^iy \in L$.

Pumping Lemma for CFLs

Theorem

Let L be a CFL. Then there exists a constant n such that if z is a string in L of length at least n, then we can write $z = uvwxy$ such that i) $|vwx| \leq n$, ii) $vx \neq \varepsilon$, and iii) for all $i \geq 0$ the string $uv^iwx^iy \in L$.

– Let G be a CFG accepting L. Let b be an upper bound on the size of the RHS of any production rule of G.

Pumping Lemma for CFLs

Theorem

Let L be a CFL. Then there exists a constant n such that if z is a string in L of length at least n, then we can write $z = uvwxy$ such that i) $|vwx| \leq n$, ii) $vx \neq \varepsilon$, and iii) for all $i \geq 0$ the string $uv^iwx^iy \in L$.

- Let G be a CFG accepting L. Let b be an upper bound on the size of the RHS of any production rule of G.
- What is the upper bound on the length strings in L with parse-tree of height $\ell + 1$?

Pumping Lemma for CFLs

Theorem

Let L be a CFL. Then there exists a constant n such that if z is a string in L of length at least n, then we can write $z = uvwxy$ such that i) $|vwx| \leq n$, ii) $vx \neq \varepsilon$, and iii) for all $i \geq 0$ the string $uv^iwx^iy \in L$.

- Let G be a CFG accepting L. Let b be an upper bound on the size of the RHS of any production rule of G.
- What is the upper bound on the length strings in L with parse-tree of height $\ell + 1$? Answer: b^ℓ.
- Let $N = |V|$ be the number of variables in G.
- What can we say about the strings z in L of size greater than b^N?

Pumping Lemma for CFLs

Theorem

Let L be a CFL. Then there exists a constant n such that if z is a string in L of length at least n, then we can write $z = uvwxy$ such that i) $|vwx| \leq n$, ii) $vx \neq \varepsilon$, and iii) for all $i \geq 0$ the string $uv^iwx^iy \in L$.

- Let G be a CFG accepting L. Let b be an upper bound on the size of the RHS of any production rule of G.
- What is the upper bound on the length strings in L with parse-tree of height $\ell + 1$? Answer: b^ℓ.
- Let $N = |V|$ be the number of variables in G.
- What can we say about the strings z in L of size greater than b^N?
- Answer: in every parse tree of z there must be a path where a variable repeats.
- Consider a minimum size parse-tree generating z, and consider a path where at least a variable repeats, and consider the last such variable.
- Justify the conditions of the pumping Lemma.

Applying Pumping Lemma

Theorem (Pumping Lemma for Context-free Languages)

$L \in \Sigma^*$ *is a context-free language*
$\Longrightarrow$
there exists $p \geq 1$ *such that*
for all strings $z \in L$ *with* $|z| \geq p$ *we have that*
there exists $u, v, w, x, y \in \Sigma^*$ *with* $z = uvwxy$, $|vx| > 0$, $|vwx| \leq p$ *such that*
for all $i \geq 0$ *we have that*
$uv^iwx^iy \in L$.

Applying Pumping Lemma

Theorem (Pumping Lemma for Context-free Languages)

$L \in \Sigma^*$ *is a context-free language*
$\Longrightarrow$
there exists $p \geq 1$ *such that*
for all strings $z \in L$ *with* $|z| \geq p$ *we have that*
there exists $u, v, w, x, y \in \Sigma^*$ *with* $z = uvwxy$, $|vx| > 0$, $|vwx| \leq p$ *such that*
for all $i \geq 0$ *we have that*
$uv^iwx^iy \in L$.

Pumping Lemma (Contrapositive)

For all $p \geq 1$ *we have that*
there exists strings $z \in L$ *with* $|z| \geq p$ *such that*
for all $u, v, w, x, y \in \Sigma^*$ *with* $z = uvwxy$, $|vx| > 0$, $|vwx| \leq p$ *we have that*
there exists $i \geq 0$ *such that*
$uv^iwx^iy \notin L$.
$\Longrightarrow$
$L \in \Sigma^*$ *is not a context-free language.*

Example

Prove that the following languages are not context-free:

1. $L = \{0^n 1^n 2^n \ : \ n \geq 0\}$
2. $L = \{0^i 1^j 2^k \ : \ 0 \leq i \leq j \leq k\}$
3. $L = \{ww \ : \ w \in \{0,1\}^*\}$.
4. $L = \{0^n \ : \ n \text{ is a prime number}\}$.
5. $L = \{0^n \ : \ n \text{ is a perfect square}\}$.
6. $L = \{0^n \ : \ n \text{ is a perfect cube}\}$.

Closure Properties

Theorem

Context-free languages are closed under the following operations:

1. *Union*
2. *Concatenation*
3. *Kleene closure*
4. *Homomorphism*
5. *Substitution*
6. *Inverse-homomorphism*
7. *Reverse*

Closure Properties

Theorem

Context-free languages are closed under the following operations:

1. *Union*
2. *Concatenation*
3. *Kleene closure*
4. *Homomorphism*
5. *Substitution*
6. *Inverse-homomorphism*
7. *Reverse*

Reading Assignment: Proof of closure under these operations.

Intersection and Complementaion

Theorem

Context-free languages are not closed under intersection and complementation.

Proof.

- Consider the languages

$$L_1 = \{0^n 1^n 2^m : n, m \geq 0\}, \text{ and}$$
$$L_2 = \{0^m 1^n 2^n : n, m \geq 0\}.$$

- Both languages are CFLs.
- What is $L_1 \cap L_2$?

Intersection and Complementaion

Theorem

Context-free languages are not closed under intersection and complementation.

Proof.

- Consider the languages

$$L_1 = \{0^n 1^n 2^m \ : \ n, m \geq 0\}, \text{ and}$$
$$L_2 = \{0^m 1^n 2^n \ : \ n, m \geq 0\}.$$

- Both languages are CFLs.
- What is $L_1 \cap L_2$?
- $L = \{0^n 1^n 2^n \ : \ n \geq 0\}$ and it is not a CFL.
- Hence CFLs are not closed under intersection.

Intersection and Complementaion

Theorem

Context-free languages are not closed under intersection and complementation.

Proof.

- Consider the languages

$$\begin{aligned} L_1 &= \{0^n 1^n 2^m \ : \ n, m \geq 0\}, \text{ and} \\ L_2 &= \{0^m 1^n 2^n \ : \ n, m \geq 0\}. \end{aligned}$$

- Both languages are CFLs.
- What is $L_1 \cap L_2$?
- $L = \{0^n 1^n 2^n \ : \ n \geq 0\}$ and it is not a CFL.
- Hence CFLs are not closed under intersection.
- Use De'morgan's law to prove non-closure under complementation.

□