Theory of Computation
Context-Free Languages

Context-Free Grammars

- Here is an example of a context-free grammar G_1:

$$\begin{array}{lcl} A & \longrightarrow & \mathtt{0}A\mathtt{1} \\ A & \longrightarrow & B \\ B & \longrightarrow & \# \end{array}$$

- Each line is a substitution rule (or production).
- A, B are variables.
- $\mathtt{0}, \mathtt{1}, \#$ are terminals.
- The left-hand side of the first rule (A) is the start variable.

Grammars and Languages

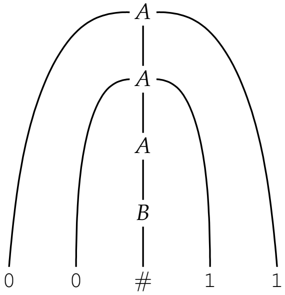
- A grammar describes a language.
- A grammar generates a string of its language as follows.
  1. Write down the start variable.
  2. Find a written variable and a rule whose left-hand side is that variable.
  3. Replace the written variable with the right-hand side of the rule.
  4. Repeat steps 2 and 3 until no variable remains.
- Any language that can be generated by some context-free grammar is called a context-free language.

Grammars and Languages

- For example, consider the following derivation of the string `00#11` generated by G_1:

$$A \Rightarrow \texttt{0}A\texttt{1} \Rightarrow \texttt{00}A\texttt{11} \Rightarrow \texttt{00}B\texttt{11} \Rightarrow \texttt{00\#11}$$

- We also use a parse tree to denote a string generated by a grammar:



Context-Free Grammars – Formal Definition

Definition

A context-free grammar is a 4-tuple (V, Σ, R, S) where

- V is a finite set of variables;
- Σ is a finite set of terminals where $V \cap \Sigma = \emptyset$;
- R is a fintie set of rules. Each rule consists of a variable and a string of variables and terminals; and
- $S \in V$ is the start variable.

- Let u, v, w are strings of variables and terminals, and $A \longrightarrow w$ a rule. We say uAv yields uwv (written $uAv \Rightarrow uwv$).
- u derives v (written $u \overset{*}{\Longrightarrow} v$) if $u = v$ or there is a sequence $u_1, u_2, \ldots, u_k$ $(k \geq 0)$ that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$.
- The language of the grammar is $\{w \in \Sigma^* : S \overset{*}{\Longrightarrow} w\}$.

Context-Free Languages – Examples

Example

Consider $G_3 = (\{S\}, \{(,)\}, R, S)$ where R is

$$S \longrightarrow (S) \mid SS \mid \epsilon.$$

- $A \longrightarrow w_1 \mid w_2 \mid \cdots \mid w_k$ stands for

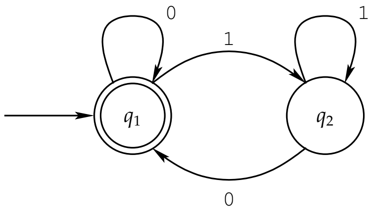
$$\begin{array}{ccc} A & \longrightarrow & w_1 \\ A & \longrightarrow & w_2 \\ & \vdots & \\ A & \longrightarrow & w_k \end{array}$$

- Examples of the strings generated by G_3: $\epsilon, (), (())(), \ldots$.

Context-Free Languages – Examples

- From a DFA M, we can construct a context-free grammar G_M such that the language of G is $L(M)$.
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Define $G_M = (V, \Sigma, P, S)$ where
  - $V = \{R_i : q_i \in Q\}$ and $S = \{R_0\}$; and
  - $P = \{R_i \longrightarrow \mathtt{a}R_j : \delta(q_i, \mathtt{a}) = q_j\} \cup \{R_i \longrightarrow \epsilon : q_i \in F\}$.
- Recall M_3 and construct $G_{M_3} = (\{R_1, R_2\}, \{\mathtt{0}, \mathtt{1}\}, P, \{R_1\})$ with

$$\begin{array}{rcl} R_1 & \longrightarrow & \mathtt{0}R_1 \mid \mathtt{1}R_2 \mid \epsilon \\ R_2 & \longrightarrow & \mathtt{0}R_1 \mid \mathtt{1}R_2. \end{array}$$

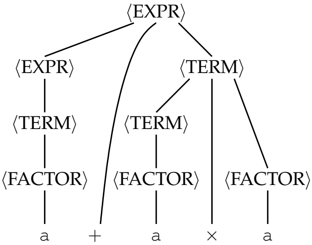


Example

Consider $G_4 = (V, \Sigma, R, \langle\text{EXPR}\rangle)$ where

- $V = \{\langle\text{EXPR}\rangle, \langle\text{TERM}\rangle, \langle\text{FACTOR}\rangle\}$, $\Sigma = \{\mathtt{a}, +, \times, (,)\}$; and
- R is

$$
\begin{array}{rcl}
\langle\text{EXPR}\rangle & \longrightarrow & \langle\text{EXPR}\rangle + \langle\text{TERM}\rangle \mid \langle\text{TERM}\rangle \\
\langle\text{TERM}\rangle & \longrightarrow & \langle\text{TERM}\rangle \times \langle\text{FACTOR}\rangle \mid \langle\text{FACTOR}\rangle \\
\langle\text{FACTOR}\rangle & \longrightarrow & (\langle\text{EXPR}\rangle) \mid \mathtt{a}
\end{array}
$$



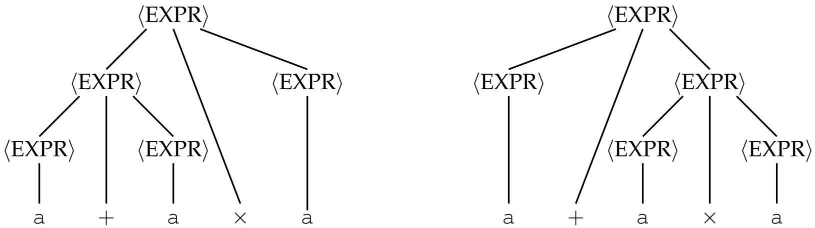
Ambiguity

Example

Consider G_5:

$$\langle\text{EXPR}\rangle \longrightarrow \langle\text{EXPR}\rangle + \langle\text{EXPR}\rangle \mid \langle\text{EXPR}\rangle \times \langle\text{EXPR}\rangle \mid (\langle\text{EXPR}\rangle) \mid \mathtt{a}$$

- We have two parse trees for $\mathtt{a} + \mathtt{a} \times \mathtt{a}$.



- If a grammar generates the same in different ways, the string is derived ambiguously in that grammar.
- If a grammar generates some string ambiguously, it is ambiguous.

Ambiguity – Formal Definition

Definition

A string is derived ambiguously in a grammar if it has two or more different leftmost derivations. A grammar is ambiguous if it generates some string ambiguously.

- A derivation is a leftmost derivation if the leftmost variable is the one replaced at every step.
- Two leftmost derivations of $\mathtt{a} + \mathtt{a} \times \mathtt{a}$:

$$\begin{array}{rcl} \langle\text{EXPR}\rangle & \Rightarrow & \langle\text{EXPR}\rangle\times\langle\text{EXPR}\rangle \Rightarrow \langle\text{EXPR}\rangle+\langle\text{EXPR}\rangle\times\langle\text{EXPR}\rangle \Rightarrow \\ & & \mathtt{a}+\langle\text{EXPR}\rangle\times\langle\text{EXPR}\rangle \Rightarrow \mathtt{a}+\mathtt{a}\times\langle\text{EXPR}\rangle \Rightarrow \mathtt{a}+\mathtt{a}\times\mathtt{a} \\ \langle\text{EXPR}\rangle & \Rightarrow & \langle\text{EXPR}\rangle+\langle\text{EXPR}\rangle \Rightarrow \mathtt{a}+\langle\text{EXPR}\rangle \Rightarrow \\ & & \mathtt{a}+\langle\text{EXPR}\rangle\times\langle\text{EXPR}\rangle \Rightarrow \mathtt{a}+\mathtt{a}\times\langle\text{EXPR}\rangle \Rightarrow \mathtt{a}+\mathtt{a}\times\mathtt{a} \end{array}$$

- If a language can only be generated by ambiguous grammars, we call it is inherently ambiguous.
  - $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i = j \text{ or } j = k\}$ is inherently ambiguous.

Chomsky Normal Form

Definition

A context-free grammar is in Chomsky normal form if every rule is of the form

$$\begin{array}{rcl} S & \longrightarrow & \epsilon \\ A & \longrightarrow & BC \\ A & \longrightarrow & \mathtt{a} \end{array}$$

where $\mathtt{a}$ is a terminal, S is the start variable, A is a variable, and B, C are non-start variables.

- A normal form means a uniform representation.
  - conjunctive normal form, negative normal form, etc.

Theorem

Any context-free language is generated by a context-free grammar in Chomsky normal form.

Chomsky Normal Form

Proof.

Given a context-free grammar for a context-free language, we will convert the grammar into Chomsky normal form.

- (start variable) Add a new start variable S_0 and a rule $S_0 \longrightarrow S$.
- (ϵ-rules) For each ϵ-rule $A \longrightarrow \epsilon (A \neq S_0)$, remove it. Then for each occurrence of A on the right-hand side of a rule, add a new rule with that occurrence deleted.
  - $R \longrightarrow uAvAw$ becomes $R \longrightarrow uAvAw \mid uvAw \mid uAvw \mid uvw$.
- (unit rules) For each unit rule $A \longrightarrow B$, remove it. Add the rule $A \longrightarrow u$ for each $B \longrightarrow u$.
- For each rule $A \longrightarrow u_1 u_2 \cdots u_k (k \geq 3)$ and u_i is a variable or terminal, replace it by $A \longrightarrow u_1 A_1$, $A_1 \longrightarrow u_2 A_2, \ldots,$ $A_{k-2} \longrightarrow u_{k-1} u_k$.
- For each rule $A \longrightarrow u_1 u_2$ with u_1 a terminal, replace it by $A \longrightarrow U_1 u_2$, $U_1 \longrightarrow u_1$. Similarly when u_2 is a terminal. $\square$

Chomsky Normal Form – Example

- Consider G_6 on the left. We add a new start variable on the right.

$$\begin{array}{rcl} S & \longrightarrow & ASA \mid \mathtt{a}B \\ A & \longrightarrow & B \mid S \\ B & \longrightarrow & \mathtt{b} \mid \epsilon \end{array} \qquad \begin{array}{rcl} \underline{S_0} & \longrightarrow & S \\ S & \longrightarrow & ASA \mid \mathtt{a}B \\ A & \longrightarrow & B \mid S \\ B & \longrightarrow & \mathtt{b} \mid \epsilon \end{array}$$

- Remove $B \longrightarrow \epsilon$ (left) and then $A \longrightarrow \epsilon$ (right):

$$\begin{array}{rcl} S_0 & \longrightarrow & S \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \underline{\mathtt{a}} \\ A & \longrightarrow & B \mid S \mid \underline{\epsilon} \\ B & \longrightarrow & \mathtt{b} \end{array} \qquad \begin{array}{rcl} S_0 & \longrightarrow & S \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid \underline{SA} \mid \underline{AS} \mid \underline{S} \\ A & \longrightarrow & B \mid S \\ B & \longrightarrow & \mathtt{b} \end{array}$$

- Remove $S \longrightarrow S$ (left) and then $S_0 \longrightarrow S$ (right):

$$\begin{array}{rcl} S_0 & \longrightarrow & S \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & B \mid S \\ B & \longrightarrow & \mathtt{b} \end{array} \qquad \begin{array}{rcl} S_0 & \longrightarrow & \underline{ASA} \mid \underline{\mathtt{a}B} \mid \underline{\mathtt{a}} \mid \underline{SA} \mid \underline{AS} \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & B \mid S \\ B & \longrightarrow & \mathtt{b} \end{array}$$

Chomsky Normal Form – Example

- Remove $A \longrightarrow B$ (left) and then $A \longrightarrow S$ (right):

$$\begin{array}{rcl} S_0 & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & S \mid \underline{\mathtt{b}} \\ B & \longrightarrow & \mathtt{b} \end{array} \qquad \begin{array}{rcl} S_0 & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S & \longrightarrow & ASA \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & \underline{\mathtt{b}} \mid \underline{ASA} \mid \underline{\mathtt{a}B} \mid \underline{\mathtt{a}} \mid \underline{SA} \mid \underline{AS} \\ B & \longrightarrow & \mathtt{b} \end{array}$$

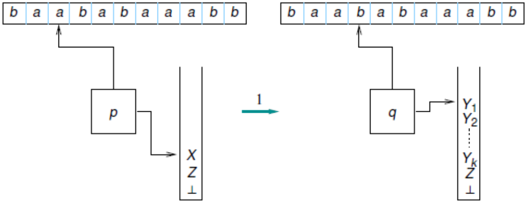
- Remove $S_0 \longrightarrow ASA$, $S \longrightarrow ASA$, and $A \longrightarrow ASA$:

$$\begin{array}{rcl} S_0 & \longrightarrow & \underline{AA_1} \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ S & \longrightarrow & \underline{AA_1} \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & \mathtt{b} \mid \underline{AA_1} \mid \mathtt{a}B \mid \mathtt{a} \mid SA \mid AS \\ B & \longrightarrow & \mathtt{b} \\ \underline{A_1} & \longrightarrow & SA \end{array}$$

- Add $U \longrightarrow a$:

$$\begin{array}{rcl} S_0 & \longrightarrow & AA_1 \mid \underline{UB} \mid \mathtt{a} \mid SA \mid AS \\ S & \longrightarrow & AA_1 \mid \underline{UB} \mid \mathtt{a} \mid SA \mid AS \\ A & \longrightarrow & \mathtt{b} \mid AA_1 \mid \underline{UB} \mid \mathtt{a} \mid SA \mid AS \\ B & \longrightarrow & \mathtt{b} \\ A_1 & \longrightarrow & SA \\ \underline{U} & \longrightarrow & \mathtt{a} \end{array}$$

Schematic of Pushdown Automata



Each step of the PDA looks like:

- Read current symbol and advance head;
- Read and pop top-of-stack symbol;
- Push in a string of symbols on the stack;
- Change state.

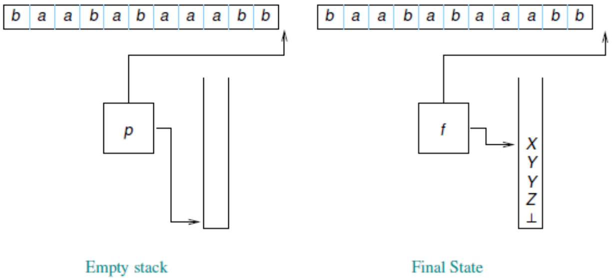
Each transition is of the form

$$(p, a, X) \to (q, Y_1 Y_2 \ldots Y_k)$$

Three Mechanisms of Acceptance

Accept if input is consumed and

- Stack is empty (Acceptance by Empty Stack),
- PDA is in a final state (Acceptance by Final State),
- PDA is in a final state and stack is empty (Acceptance by Final State and Empty Stack).



It turns out that the three notions of acceptance are equivalent.

Pushdown Automata

- Consider $L = \{\texttt{0}^n\texttt{1}^n : n \geq 0\}$.
- We have the following table:

Language	Automata
Regular	Finite
Context-free	Pushdown

- A pushdown automaton is a finite automaton with a stack.
  - A stack is a last-in-first-out storage.
  - Stack symbols can be pushed and poped from the stack.
- Computation depends on the content of the stack.
- It is not hard to see L is recognized by a pushdown automaton.

Pushdown Automata

- Consider $L = \{\mathtt{0}^n\mathtt{1}^n : n \geq 0\}$.
- We have the following table:

Language	Automata
Regular	Finite
Context-free	Pushdown

- A pushdown automaton is a finite automaton with a stack.
  - A stack is a last-in-first-out storage.
  - Stack symbols can be pushed and poped from the stack.
- Computation depends on the content of the stack.
- It is not hard to see L is recognized by a pushdown automaton.

Pushdown Automata – Formal Definition

Definition

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

- Q is the set of states;
- Σ is the input alphabet;
- Γ is the stack alphabet;
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function;
- $q_0 \in Q$ is the start state; and
- $F \subseteq Q$ is the accept states.

- Recall $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$.
- We consider nondeterministic pushdown automata in the definition. It convers deterministic pushdown automata.
- Deterministic pushdown automata are strictly less powerful.
  - There is a langauge recognized by only nondeterministic pushdown automata.

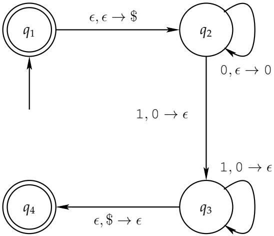
Computation of Pushdown Automata

- A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w if w can be written as $w = w_1 w_2 \cdots w_m$ with $w_i \in \Sigma_\epsilon$ and there are sequences of states $r_0, r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots, s_m \in \Gamma^*$ such that
  - $r_0 = q_0$ and $s_0 = \epsilon$;
    - ★ M starts with the start state and the empty stack.
  - For $0 \leq i < m$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, $s_i = at$, and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$.
    - ★ On reading w_{i+1}, M moves from r_i with stack at to r_{i+1} with stack bt.
    - ★ Write $c, a \to b (c \in \Sigma_\epsilon$ and $a, b \in \Gamma_\epsilon)$ to denote that the machine is reading c from the input and replacing the top of stack a with b.
  - $r_m \in F$.
    - ★ M is at an accept state after reading w.
- The language recognized by M is denoted by $L(M)$.
  - That is, $L(M) = \{w : M \text{ accepts } w\}$.
- For convenience, we extend δ to $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma^*)$

Pushdown Automata – Example

- Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ where
  - $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$; and
  - δ is the following table:

input	0			1			ϵ		
stack	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$				$\{(q_4, \epsilon)\}$	
q_4									

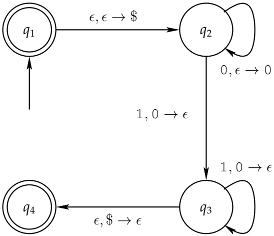


- $L(M_1) = \{0^n 1^n : n \geq 0\}$

Pushdown Automata – Example

- Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ where
  - $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$; and
  - δ is the following table:

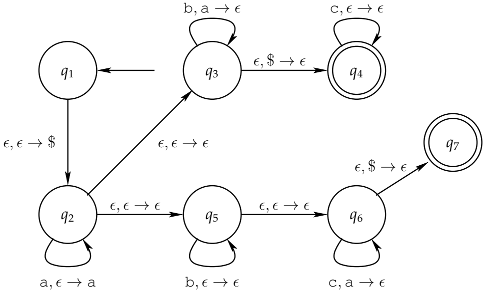
input	0			1			ϵ		
stack	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \epsilon)\}$					
q_3				$\{(q_3, \epsilon)\}$				$\{(q_4, \epsilon)\}$	
q_4									



- $L(M_1) = \{0^n 1^n : n \geq 0\}$

Pushdown Automata – Example

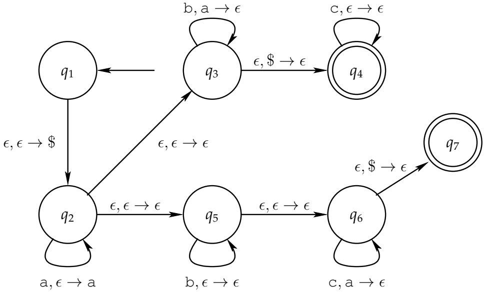
- Consider the following pushdown automaton M_2:



- $L(M_2) = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i, j, k \geq 0 \text{ and}, i = j \text{ or } i = k\}$

Pushdown Automata – Example

- Consider the following pushdown automaton M_2:



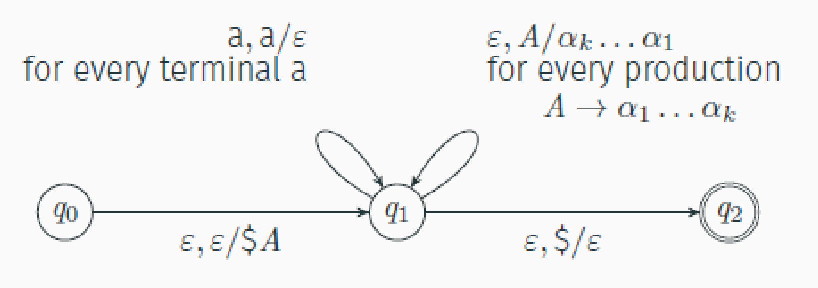
- $L(M_2) = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : i, j, k \geq 0 \text{ and, } i = j \text{ or } i = k\}$

Context-Free Grammars ⇒ Pushdown Automata

- Idea: Use PDA to simulate derivations
- Example: $G : A \to 0A1 \mid B;\ \ B \to \#$
- Derivation: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$
- Rule:
  - Write the start symbol A onto the stack
  - Rewrite variable on top of stack (in reverse) according to production
  - Pop top terminal if it matches input

input	00#11	00#11	00#11	00#11	00#11
stack	$\$A$	$\$1A0$	$\$1A$	$\$11A0$	$\$11A$
	00#11	00#11	00#11	00#11	00#11
	$\$11B$	$\$11\#$	$\$11$	$\$1$	$\$$

Context-Free Grammars ⇒ Pushdown Automata



Lemma

If a language is context-free, some pushdown automaton recognizes it.

Proof.

Let $G = (V, \Sigma, R, S)$ be a context-free grammar generating the language. Define

$P = (\{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}, \ldots\}, \Sigma, V \cup \Sigma \cup \{\$\}, \delta, q_{\text{start}}, \{q_{\text{accept}}\})$ where

- $\delta(q_{\text{start}}, \epsilon, \epsilon) = \{(q_{\text{loop}}, S\$)\}$
- $\delta(q_{\text{loop}}, \epsilon, A) = \{(q_{\text{loop}}, w) : A \longrightarrow w \in R\}$
- $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \epsilon)\}$
- $\delta(q_{\text{loop}}, \epsilon, \$) = \{(q_{\text{accept}}, \epsilon)\}$

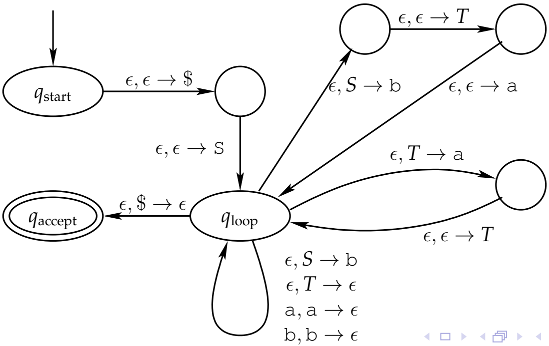
Note that $(r, u_1 u_2 \cdots u_l) \in \delta(q, a, s)$ is simulated by $(q_1, u_l) \in \delta(q, a, s)$, $\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\}, \ldots, \delta(q_{l-1}, \epsilon, \epsilon) = \{(r, u_1)\}$. $\square$

Example

Example

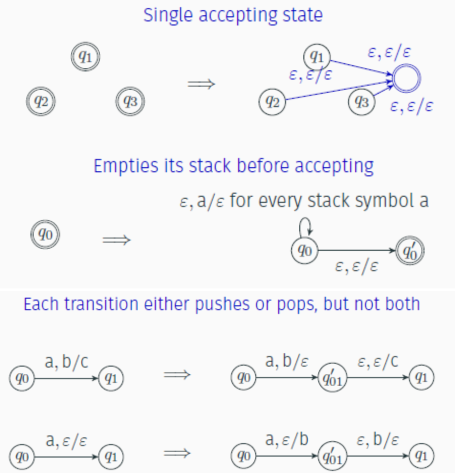
Find a pushdown automaton recognizing the language of the following context-free grammar:

$$\begin{array}{rcl} S & \longrightarrow & \mathtt{a}T\mathtt{b} \mid \mathtt{b} \\ T & \longrightarrow & T\mathtt{a} \mid \epsilon \end{array}$$



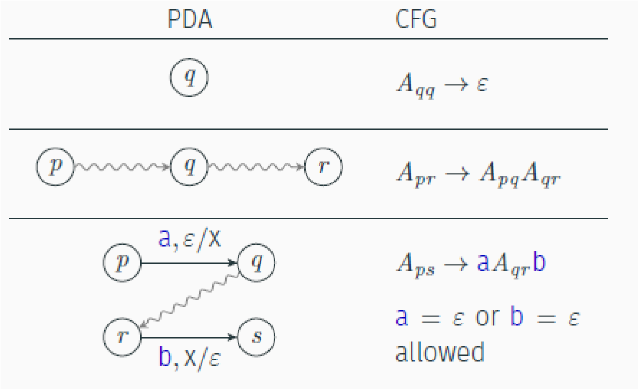
Simplified PDA

- Has a single accepting state
- Empties its stack before accepting
- Each transition is either a push, or a pop, but not both

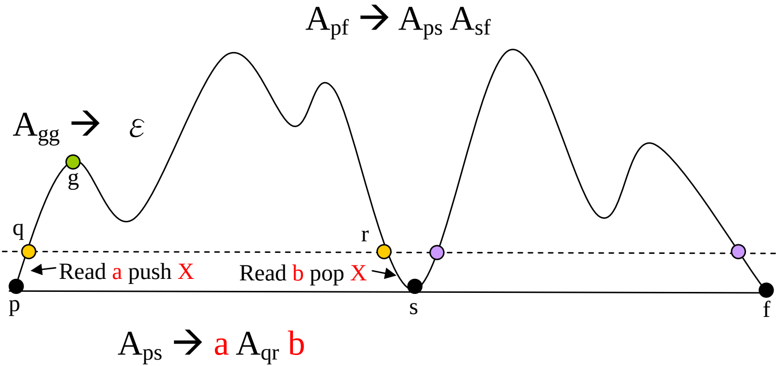


Pushdown Automata ⇒ Context-Free Grammars

- Key Idea: For every pair (q, r) of states in PDA, introduce variable A_{qr} in CFG so that A_{qr} generates all strings that allow the PDA to go from q to r (with empty stack both at q and at r)



Pushdown Automata ⇒ Context-Free Grammars



Pushdown Automata $\Rightarrow$ Context-Free Grammars

Lemma

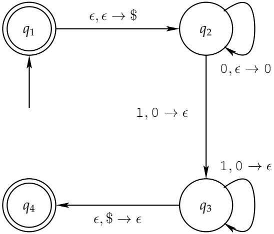
If a pushdown automaton recognizes a language, the language is context-free.

Proof.

Without loss of generality, we consider a pushdown automaton that has a single accept state q_{accept} and empties the stack before accepting. Moreover, its transition either pushes or pops a stack symbol at any time. Let $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$. Define the context-free grammar $G = (V, \Sigma, R, S)$ where

- $V = \{A_{pq} : p, q \in Q\}$, $S = A_{q_0, q_{\text{accept}}}$; and
- R has the following rules:
  - For each $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$, then $A_{pq} \longrightarrow aA_{rs}b \in R$.
  - For each $p, q, r \in Q$, $A_{pq} \longrightarrow A_{pr}A_{rq} \in R$.
  - For each $p \in Q$, $A_{pp} \longrightarrow \epsilon \in R$.

Example



- We write $A_{i,j}$ for $A_{q_i q_j}$.
- Consider the following context-free grammar:

$$\begin{array}{lcll}
A_{14} & \to & A_{23} & \text{since } (q_2, \$) \in \delta(q_1, \epsilon, \epsilon) \text{ and } (q_4, \epsilon) \in \delta(q_3, \epsilon, \$) \\
A_{23} & \to & 0A_{23}1 & \text{since } (q_2, 0) \in \delta(q_2, 0, \epsilon) \text{ and } (q_3, \epsilon) \in \delta(q_3, 1, 0) \\
A_{23} & \to & 0A_{22}1 & \text{since } (q_2, 0) \in \delta(q_2, 0, \epsilon) \text{ and } (q_3, \epsilon) \in \delta(q_2, 1, 0) \\
A_{22} & \to & \epsilon &
\end{array}$$

Pushdown Automata ⇒ Context-Free Grammars

Lemma

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

Proof.

Prove by induction on the length k of derivation.

- $k = 1$. The only possible derivation of length 1 is $A_{pp} \Rightarrow \epsilon$.
- Consider $A_{pq} \stackrel{*}{\Longrightarrow} x$ of length $k + 1$. Two cases for the first step:
  - $A_{pq} \Rightarrow aA_{rs}b$. Then $x = ayb$ with $A_{rs} \stackrel{*}{\Longrightarrow} y$. By IH, y brings P from r to s with empty stack. Moreover, $(r, t) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$ since $A_{pq} \longrightarrow aA_{rs}b \in R$. Let P start from p with empty stack, P first moves to r and pushes t to the stack after reading a. It then moves to s with t in the stack. Finally, P moves to q with empty stack after reading b and popping t.
  - $A_{pq} \Rightarrow A_{pr}A_{rq}$. Then $x = yz$ with $A_{pr} \stackrel{*}{\Longrightarrow} y$ and $A_{rq} \stackrel{*}{\Longrightarrow} z$. By IH, P moves from p to r, and then r to q. □

Pushdown Automata $\Rightarrow$ Context-Free Grammars

Lemma

If x can bring P from p with empty stack to q with empty stack, A_{pq} generates x in G.

Proof.

Prove by induction on the length k of computation.

- $k = 0$. The only possible 0-step computation is to stay at the same state while reading ϵ. Hence $x = \epsilon$. Clearly, $A_{pp} \stackrel{*}{\Longrightarrow} \epsilon$ in G.
- Two possible cases for computation of length $k + 1$.
  - The stack is empty only at the beginning and end of the computation. If P reads a, pushes t, and moves to r from p at step 1, $(r, t) \in \delta(q, a, \epsilon)$. Similarly, if P reads b, pops t, and moves to q from s at step $k + 1$, $(q, \epsilon) \in \delta(s, b, t)$. Hence $A_{pq} \longrightarrow aA_{rs}b \in G$. Let $x = ayb$. By IH, $A_{rs} \stackrel{*}{\Longrightarrow} y$. We have $A_{pq} \stackrel{*}{\Longrightarrow} x$.
  - The stack is empty elsewhere. Let r be a state where the stack becomes empty. Say y and z are the inputs read during the computation from p to r and r to q respectively. Hence $x = yz$. By IH, $A_{pr} \stackrel{*}{\Longrightarrow} y$ and $A_{rq} \stackrel{*}{\Longrightarrow} z$. Since $A_{pq} \longrightarrow A_{pr}A_{rq} \in G$. We have $A_{pq} \stackrel{*}{\Longrightarrow} x$. $\square$

Theorem

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

Corollary

Every regular language is context-free.

Pumping Lemma

Theorem

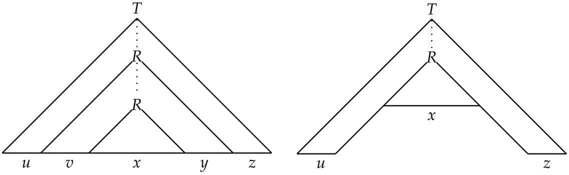
If A is a context-free language, then there is a number p (the puming length) such that for every $s \in A$ with $|s| \geq p$, there is a partition $s = uvxyz$ that

1. *for each $i \geq 0$, $uv^ixy^iz \in A$;*
2. $|vy| > 0$*; and*
3. $|vxy| \leq p$.

Proof.

Let $G = (V, \Sigma, R, T)$ be a context-free grammar for A. Define b to be the maximum number of symbols in the right-hand side of a rule. Observe that a parse tree of height h has at most b^h leaves and hence can generate strings of length at most b^h.
Choose $p = b^{|V|+1}$. Let $s \in A$ with $|s| \geq p$ and τ the smallest parse tree for s. Then the height of $\tau \geq |V| + 1$. There are $|V| + 1$ variables along the longest branch. A variable R must appear twice.

Pumping Lemma



(a) Smallest parse tree τ (b) A parse tree if $|vy| = 0$

Figure: Pumping Lemma

Proof. (cont'd).

From Figure (a), we see $uv^ixy^iz \in A$ for $i \geq 0$.
Suppose $|vy| = 0$. Then Figure (b) is a smaller parse tree than τ. A contradiction. Hence $|vy| > 0$.
Finally, recall R is in the longest branch of length $|V| + 1$. Hence the subtree R generating vxy has height $\leq |V| + 1$. $|vxy| \leq b^{|V|+1} = p$. □

Pumping Lemma – Examples

Example

Show $B = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n : n \geq 0\}$ is not a context-free language.

Proof.

Let p be the pumping length. $s = \mathtt{a}^p\mathtt{b}^p\mathtt{c}^p \in B$. Consider a partition $s = uvxyz$ with $|vy| > 0$. There are two cases:

- v or y contain more than one type of symbol. Then $uv^2xy^2z \notin B$.
- v and y contain only one type of symbol. Then one of $\mathtt{a}$, $\mathtt{b}$, or $\mathtt{c}$ does not appear in v nor y (pigeon hole principle). Hence $uv^2xy^2z \notin B$ for $|vy| > 0$.

$\square$

Pumping Lemma – Examples

Example

Show $C = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k : 0 \leq i \leq j \leq k\}$ is not a context-free language.

Proof.

Let p be the pumping length and $s = \mathtt{a}^p\mathtt{b}^p\mathtt{c}^p \in C$. Consider any partition $s = uvxyz$ with $|vy| > 0$. There are two cases:

- v or y contain more than one type of symbol. Then $uv^2xy^2z \notin C$.
- v and y contain only one type of symbol. Then one of a, b, or c does not appear in v nor y. We have three subcases:
  - a does not appear in v nor y. $uxz \notin C$ for it has more a then b or c.
  - b does not appear in v nor y. Since $|vy| > 0$, a or c must appear in v or y. If a appears, $uv^2xy^2z \notin C$ for it has more a than b. If c appears, $uxy \notin C$ for it has more b than c.
  - c does not appear in v nor y. $uv^2xy^2z \notin C$ for it has less c than a or b.

□

Pumping Lemma – Examples

Example

Show $D = \{ww : w \in \{\mathtt{0}, \mathtt{1}\}^*\}$ is not a context-free language.

Proof.

Let p be the pumping length and $s = \mathtt{0}^p\mathtt{1}^p\mathtt{0}^p\mathtt{1}^p$. Consider a partition $s = uvxyz$ with $|vy| > 0$ and $|vxy| \leq p$.

If $\mathtt{0}\cdots\mathtt{0}\overbrace{\mathtt{0}\cdots\mathtt{0}\mathtt{1}\cdots\mathtt{1}}^{vxy}\mathtt{1}\cdots\mathtt{1}\mathtt{0}^p\mathtt{1}^p$, uv^2xy^2z moves $\mathtt{1}$ into the second half and thus $uv^2xy^2z \notin D$. Similarly, uv^2xy^2z moves $\mathtt{0}$ into the first half if

$\mathtt{0}^p\mathtt{1}^p\mathtt{0}\cdots\mathtt{0}\overbrace{\mathtt{0}\cdots\mathtt{0}\mathtt{1}\cdots\mathtt{1}}^{vxy}\mathtt{1}\cdots\mathtt{1}$.

It remains to consider $\mathtt{0}^p\mathtt{1}\cdots\mathtt{1}\overbrace{\mathtt{1}\cdots\mathtt{1}\mathtt{0}\cdots\mathtt{0}}^{vxy}\mathtt{0}\cdots\mathtt{0}\mathtt{1}^p$. But then $uxz = \mathtt{0}^p\mathtt{1}^i\mathtt{0}^j\mathtt{1}^p$ with $i < p$ or $j < p$ for $|vy| > 0$. $uxz \notin D$. □