

Context-Free Languages and Pushdown Automata

Context-Free Grammars

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

$$A \longrightarrow 0A1$$

$$A \longrightarrow B$$

$$B \longrightarrow \#$$

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

Grammars and Languages

$$\begin{aligned}A &\longrightarrow 0A1 \\A &\longrightarrow B \\B &\longrightarrow \#\end{aligned}$$

- 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.
- For example, consider the following *derivation* of the string $00\#11$ generated by G_1 :

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

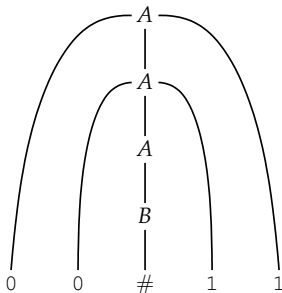
- Any language that can be generated by some context-free grammar is called a *context-free language*.

Grammars and Languages

- With respect to the following *derivation* of the string $00\#11$ generated by G_1 :

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 00B11 \Rightarrow 00\#11$$

we also use a *parse tree* to denote a string generated by a grammar:



Definition 1

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

- V is a finite set of *variables* (also called *non-terminals*);
 - Σ is a finite set of *terminals* where $V \cap \Sigma = \emptyset$;
 - R is a finite set of *production 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 $A \rightarrow w$ a rule. We say uAv yields uwv (written $uAv \Rightarrow uwv$).
 - u derives v (written $u \Rightarrow^* v$) if $u = v$ or there is a sequence u_1, u_2, \dots, u_k ($k \geq 0$) that $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$.
 - The *language* of the grammar is $\{w \in \Sigma^* : S \Rightarrow^* w\}$.

Example 2 (Balanced Parentheses)

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

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

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

$$A \longrightarrow w_1$$

$$A \longrightarrow w_2$$

$$\vdots$$

$$A \longrightarrow w_k$$

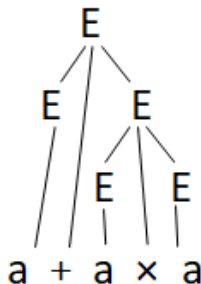
- Examples of the strings generated by G_3 : $\epsilon, (), (())(), \dots$

Parse Trees vs. Derivation Sequences

Consider the following grammar: $E \rightarrow E + E \mid E \times E \mid (E) \mid a$

The following two derivation sequences have the same parse tree.

- $E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E \times E \Rightarrow a + E \times a \Rightarrow a + a \times a$
- $E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow a + E \times E \Rightarrow a + a \times E \Rightarrow a + a \times a$



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
 - $P = \{R_i \rightarrow aR_j : \delta(q_i, a) = q_j\} \cup \{R_i \rightarrow \epsilon : q_i \in F\}$.
- Recall M_3 and construct $G_{M_3} = (\{R_1, R_2\}, \{0, 1\}, P, \{R_1\})$ with

$$\begin{aligned}R_1 &\longrightarrow 0R_1 \mid 1R_2 \mid \epsilon \\R_2 &\longrightarrow 0R_1 \mid 1R_2.\end{aligned}$$

- The above is a *right-linear grammar* for which the right-hand-side contains at most one variable at the end of the rule.

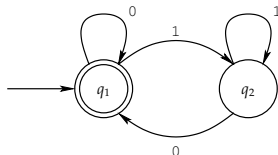


Figure: M_3

Subclasses of Context-Free Grammars

- Right-Linear Grammar

$$\begin{aligned}R_1 &\longrightarrow 0R_1 \mid 1R_2 \mid \epsilon \\R_2 &\longrightarrow 0R_1 \mid 1R_2\end{aligned}$$

- Left-Linear Grammar

$$\begin{aligned}R_1 &\longrightarrow R_10 \mid R_21 \mid \epsilon \\R_2 &\longrightarrow R_10 \mid R_21\end{aligned}$$

- Linear Grammar

$$R_1 \longrightarrow 0R_11 \mid \epsilon$$

Note: Left- and Right-Linear Grammars only generate regular languages, while Linear Grammar could generate non-regular languages such as $\{0^n1^n \mid n \geq 0\}$.

- How about rules contain both RL and LL rules? (Can you use such to generate $\{0^n1^n \mid n \geq 0\}$?)

$$\begin{aligned}R_1 &\longrightarrow R_21 \mid \epsilon \\R_2 &\longrightarrow 0R_1\end{aligned}$$

Context-Free vs. Context-Sensitive Grammars

- 1 Context-Free Rules: $A \rightarrow \beta, \beta \in (V \cup \Sigma)^*$
- 2 Context-Sensitive Rules: $\alpha A \gamma \rightarrow \alpha \beta \gamma, \alpha, \gamma \in (V \cup \Sigma)^*, \beta \in (V \cup \Sigma)^+$
- **Similarity:** both replace A by β .
- **Difference:** in (2), replacing A by β could only take place if A is surrounded by (in the context of) α and γ .
- Context-sensitive grammars are more powerful than context-free grammars.

The Chomsky Hierarchy

Grammars	Rules	Languages	Automata
Type 3 / Right-linear	$A \rightarrow aB, A \rightarrow \epsilon$	Regular	DFA/NFA
Type 2 / CFG	$A \rightarrow \alpha$	CFL	PDA
Type 1 / CSG	$\alpha A \gamma \rightarrow \alpha \beta \gamma, \beta > 0$	CSL	LBA
Type 0 / Unrestricted	$\alpha A \gamma \rightarrow \beta$	r.e.	Turing Machine

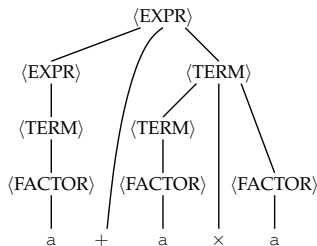
Context-Free Languages – Examples

Example 3 (Fragment of C Grammar)

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

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

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

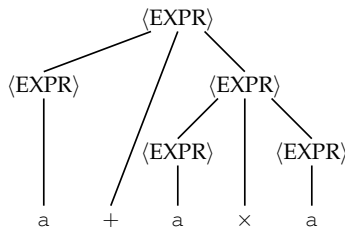
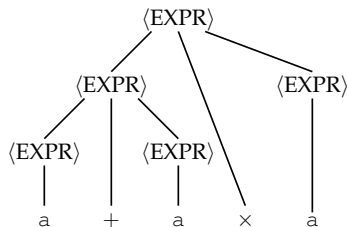


Example 4 (Fragment of C Grammar)

Consider G_5 :

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

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



- If a grammar generates (w.r.t. parse trees) the same in different ways, the string is derived *ambiguously* in that grammar.
- If a grammar generates some string ambiguously, it is *ambiguous*.

Ambiguity

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

$$\begin{aligned} \langle \text{EXPR} \rangle &\Rightarrow \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow \\ &a + \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow a + a \times \langle \text{EXPR} \rangle \Rightarrow a + a \times a \\ \langle \text{EXPR} \rangle &\Rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \Rightarrow a + \langle \text{EXPR} \rangle \Rightarrow \\ &a + \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow a + a \times \langle \text{EXPR} \rangle \Rightarrow a + a \times a \end{aligned}$$

Theorem 5

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

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

Chomsky Normal Form (CNF)

Definition 6

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

$$\begin{aligned} S &\longrightarrow \epsilon \\ A &\longrightarrow BC \\ A &\longrightarrow a \end{aligned}$$

where a is a terminal, S is the start variable, A is a variable, and B, C are non-start variables.

- RHS is (1) ϵ (only from S), (2) exactly two non-start variables, (3) exactly one terminal.

Theorem 7

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.

- 1 **(start variable)** Add a new start variable S_0 and a rule $S_0 \rightarrow S$.
- 2 **(ϵ -rules)** For each ϵ -rule $A \rightarrow \epsilon (A \neq S_0)$, remove it. Then for each occurrence of A on the right-hand side of a rule, add a new rule with that occurrence deleted.
 - $R \rightarrow uAvAw$ becomes $R \rightarrow uAvAw \mid uvAw \mid uAvw \mid uvw$.
- 3 **(unit rules)** For each unit rule $A \rightarrow B$, remove it. Add the rule $A \rightarrow u$ for each $B \rightarrow u$.
- 4 For each rule $A \rightarrow u_1u_2 \cdots u_k (k \geq 3)$ and u_i is a variable or terminal, replace it by $A \rightarrow u_1A_1, A_1 \rightarrow u_2A_2, \dots, A_{k-2} \rightarrow u_{k-1}u_k$.
- 5 For each rule $A \rightarrow u_1u_2$ with u_1 a terminal, replace it by $A \rightarrow U_1u_2, U_1 \rightarrow u_1$. Similarly when u_2 is a terminal. □

Chomsky Normal Form – Example

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

$$\begin{array}{l} S \longrightarrow ASA \mid aB \\ A \longrightarrow B \mid S \\ B \longrightarrow b \mid \epsilon \end{array} \qquad \begin{array}{l} \underline{S_0} \longrightarrow S \\ \underline{S} \longrightarrow ASA \mid aB \\ A \longrightarrow B \mid S \\ B \longrightarrow b \mid \epsilon \end{array}$$

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

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

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

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

Chomsky Normal Form – Example

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

$$\begin{array}{l} S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A \rightarrow S \mid \underline{b} \\ B \rightarrow b \end{array} \quad \begin{array}{l} S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A \rightarrow \underline{b} \mid \underline{ASA} \mid \underline{aB} \mid \underline{a} \mid \underline{SA} \mid \underline{AS} \\ B \rightarrow b \end{array}$$

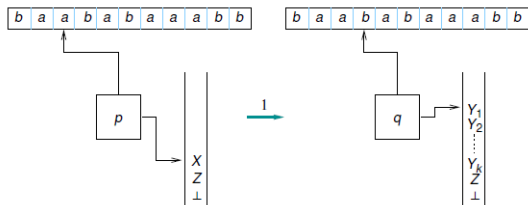
- Remove $S_0 \rightarrow ASA$, $S \rightarrow ASA$, and $A \rightarrow ASA$:

$$\begin{array}{l} S_0 \rightarrow \underline{AA_1} \mid aB \mid a \mid SA \mid AS \\ S \rightarrow \underline{AA_1} \mid aB \mid a \mid SA \mid AS \\ A \rightarrow b \mid \underline{AA_1} \mid aB \mid a \mid SA \mid AS \\ B \rightarrow b \\ \underline{A_1} \rightarrow SA \end{array}$$

- Add $U \rightarrow a$:

$$\begin{array}{l} S_0 \rightarrow AA_1 \mid \underline{UB} \mid a \mid SA \mid AS \\ S \rightarrow AA_1 \mid \underline{UB} \mid a \mid SA \mid AS \\ A \rightarrow b \mid AA_1 \mid \underline{UB} \mid a \mid SA \mid AS \\ B \rightarrow b \\ A_1 \rightarrow SA \\ \underline{U} \rightarrow a \end{array}$$

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.

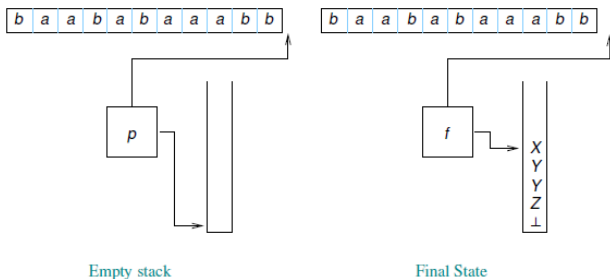
Each transition is of the form

$$(p, a, X) \rightarrow (q, Y_1 Y_2 \dots Y_k)$$

Three Mechanisms of Acceptance

Accept if input is consumed and

- 1 Stack is empty (*Acceptance by Empty Stack*),
- 2 PDA is in a final state (*Acceptance by Final State*),
- 3 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 $L = \{0^n 1^n : n \geq 0\}$.
- 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 popped 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 $L = \{0^n 1^n : n \geq 0\}$.
- 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 popped from the stack.
- Computation depends on the content of the stack.
- It is not hard to see L is recognized by a pushdown automaton.

Definition 8

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*;
 - $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ 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 converts deterministic pushdown automata.
 - **Deterministic pushdown automata are strictly less powerful.**
 - For convenience, we often extend δ to $Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma^*)$, i.e., allowing $a \in \Gamma_\epsilon$ in the stack to be replaced by $x \in \Gamma^*$.

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 $r_0, r_1, \dots, r_m \in Q$ and strings $s_0, s_1, \dots, s_m \in \Gamma^*$ (representing contents of the stack) such that

$$(r_0, s_0) \xrightarrow{w_1, -} (r_1, s_1) \cdots (r_i, at) \xrightarrow{w_{i+1}, a \rightarrow b} (r_{i+1}, bt) \cdots \xrightarrow{w_m, -} (r_m, s_m)$$

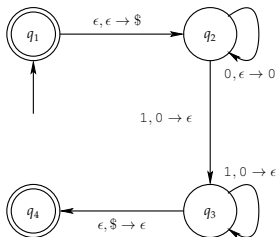
where

- $r_0 = q_0$ and $s_0 = \epsilon$;
- 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 $c, a \rightarrow b$ ($c \in \Sigma_\epsilon$ 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 .
- $r_m \in F$.
- The language recognized by M is denoted by $L(M)$.
 - That is, $L(M) = \{w : M \text{ accepts } w\}$.

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									

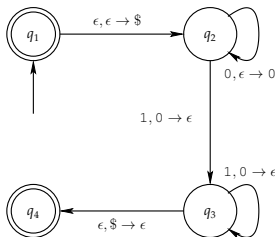


- $L(M_1) = \{0^n 1^n : n \geq 0\}$

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:

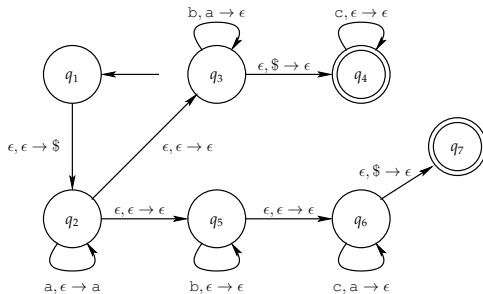
input	0			1			ϵ		
stack	0	\$	ϵ	0	\$	ϵ	0	\$	ϵ
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$			$\{(q_3, \epsilon)\}$			
q_3						$\{(q_3, \epsilon)\}$			$\{(q_4, \epsilon)\}$
q_4									



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

Pushdown Automata – Example

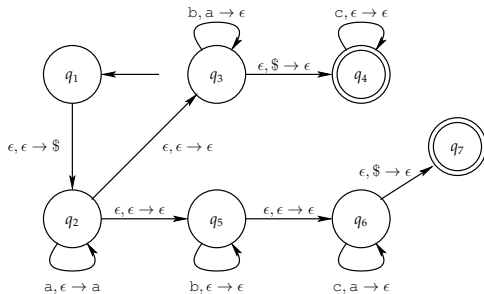
- Consider the following pushdown automaton M_2 :



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

Pushdown Automata – Example

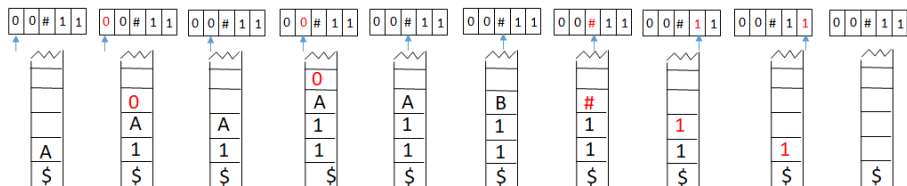
- Consider the following pushdown automaton M_2 :



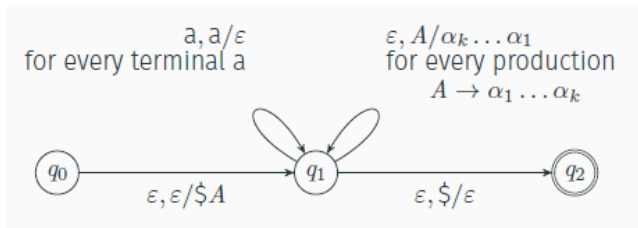
- $L(M_2) = \{a^i b^j c^k : i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Context-Free Grammars \Rightarrow Pushdown Automata

- Idea: Use PDA to simulate derivations
- Example: $G : A \rightarrow 0A1 \mid B; B \rightarrow \#$
- 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



Context-Free Grammars \Rightarrow Pushdown Automata



Note

- The above construction seems to suggest that the number of states in a PDA is not very "important".
- In fact, we can turn an arbitrary PDA into an equivalence one with a single state, such PDA are sometimes called "stateless" PDA.
 - The "state" information can be incorporated into a "stack symbol", in a way how a programming language handles the "call-return" mechanism in a function call.

Context-Free Grammars \Rightarrow Pushdown Automata

Lemma 9

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

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

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

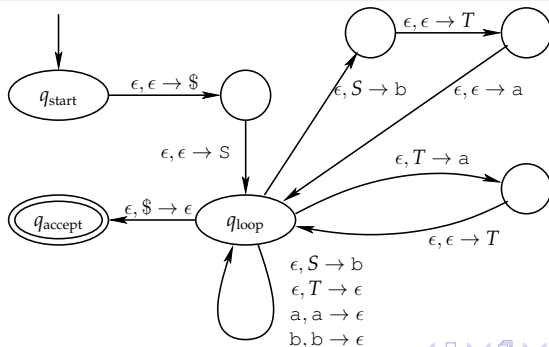
Note that $(r, u_1 u_2 \cdots u_l) \in \delta(q, a, s)$ is simulated by $(q_1, u_1) \in \delta(q, a, s)$,
 $\delta(q_1, \epsilon, \epsilon) = \{(q_2, u_{l-1})\}, \dots, \delta(q_{l-1}, \epsilon, \epsilon) = \{(r, u_1)\}$. □

Example

Example 10

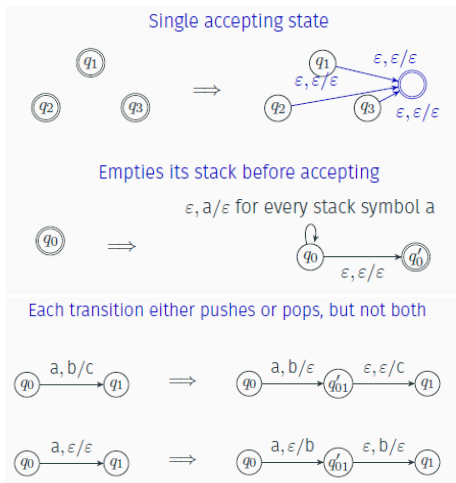
Find a pushdown automaton recognizing the language of the following context-free grammar:

$$\begin{aligned} S &\longrightarrow aTb \mid b \\ T &\longrightarrow Ta \mid \epsilon \end{aligned}$$





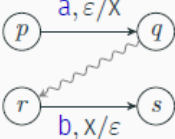
Simplified PDA

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



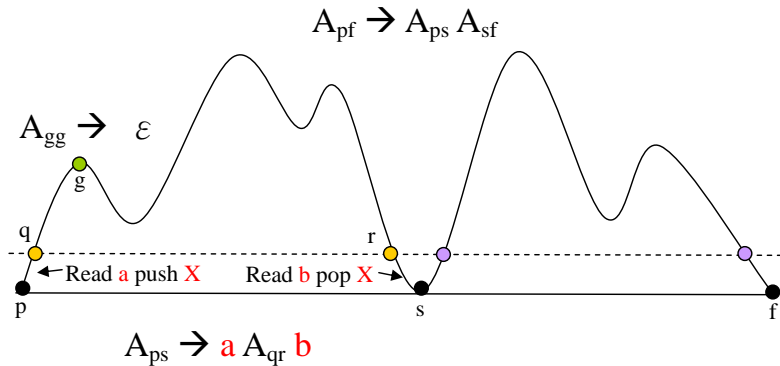
Pushdown Automata \Rightarrow Context-Free Grammars

- Key Idea: For every pair (q, r) of states in PDA, introduce variable A_{qr} in CFG so that
 - $A_{qr} \xRightarrow{*} w$ iff PDA goes from q to r reading w (with empty stack both at q and at r)

PDA	CFG
	$A_{qq} \rightarrow \varepsilon$
	$A_{pr} \rightarrow A_{pq}A_{qr}$
	$A_{ps} \rightarrow aA_{qr}b$ $a = \varepsilon$ or $b = \varepsilon$ allowed

Pushdown Automata \Rightarrow Context-Free Grammars

- Type 1: $A_{ps} \rightarrow aA_{qr}b$
- Type 2: $A_{pf} \rightarrow A_{ps}A_{sf}$
- Type 3: $A_{gg} \rightarrow \epsilon$



Pushdown Automata \Rightarrow Context-Free Grammars

Lemma 11

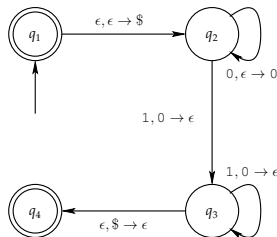
If a pushdown automaton recognizes a language, the language is context-free.

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 $A_{pq} \rightarrow aA_{rs}b \in R$.
 - For each $p, q, r \in Q$, $A_{pq} \rightarrow A_{pr}A_{rq} \in R$.
 - For each $p \in Q$, $A_{pp} \rightarrow \epsilon \in R$.

Example



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

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

Pushdown Automata \Rightarrow Context-Free Grammars

Lemma 12

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 $A_{pq} * x$ of length $k + 1$. Two cases for the first step:
 - $A_{pq} \Rightarrow aA_{rs}b$. Then $x = ayb$ with $A_{rs} * y$. 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 $A_{pq} \rightarrow aA_{rs}b \in R$. 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 $A_{pr} * y$ and $A_{rq} * z$. By IH, P moves from p to r , and then r to q . □

Pushdown Automata \Rightarrow Context-Free Grammars

Lemma 13

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, $A_{pp}*\epsilon$ 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 $A_{pq} \rightarrow aA_{rs}b \in G$. Let $x = ayb$. By IH, $A_{rs}*y$. We have $A_{pq}*x$.
 - 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, $A_{pr}*y$ and $A_{rq}*z$. Since $A_{pq} \rightarrow A_{pr}A_{rq} \in G$. We have $A_{pq}*x$. □

Theorem 14

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

Corollary 15

Every regular language is context-free.

- When we say PDA, we mean “nondeterministic PDA”
- Deterministic PDA (DPDA) are less powerful, they only accept deterministic context-free languages (DCFL).
- DPDA cannot accept $\{ww^R \mid w \in \{0, 1\}^*\}$ or $\overline{\{a^n b^n c^n \mid n \geq 0\}}$.
- The equivalence problem is undecidable for PDA, yet it is decidable for DPDA.
- $\text{Regular} \subsetneq \text{DCFL} \subsetneq \text{CFL}$
- DCFLs are not closed under union, intersection, concatenation, but are closed under complement.

Pumping Lemma for CFLs

Theorem 16

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

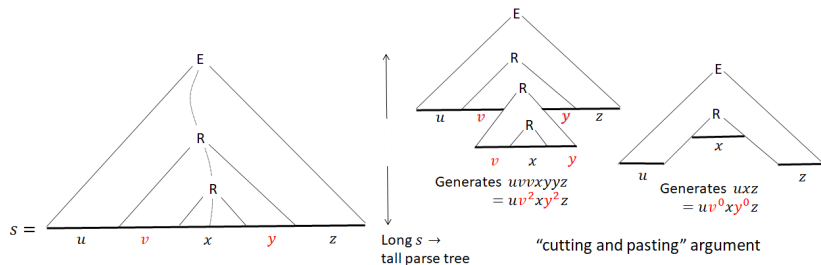
- 1 for each $i \geq 0$, $uv^i xy^i z \in A$;
- 2 $|vy| > 0$; and
- 3 $|vxy| \leq p$.

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 for CFLs



(Fig. from M. Sipser’s class notes)

Proof. (cont’d).

From Figure (a), we see $uv^i xy^i z \in A$ 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 $\leq |V| + 1$. $|vxy| \leq b^{|V|+1} = p$. \square

Pumping Lemma – Examples

Example 17

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

Proof.

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

- v or y contain more than one type of symbol, e.g.,

$\overbrace{aaaa}^u \overbrace{aab}^v \overbrace{bbbbcccccc}^{xyz}$. Then $uv^2xy^2z \notin B$.

- v and y contain only one type of symbol, e.g.,

$\overbrace{aaa}^u \overbrace{aa}^v \overbrace{ab}^x \overbrace{bb}^y \overbrace{bbbcccccc}^z$. Then one of a , b , or c does not appear in v nor y (pigeon hole principle). Hence $uv^2xy^2z \notin B$ for $|vy| > 0$. □

Pumping Lemma – Examples

Example 18

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

Proof.

Let p be the pumping length and $s = a^p b^p c^p \in C$. 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 than 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 19

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

Proof.

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

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

$0^p 1^p 0 \dots 0 \overbrace{0 \dots 0 1 \dots 1}^{vxy} 1 \dots 1$.

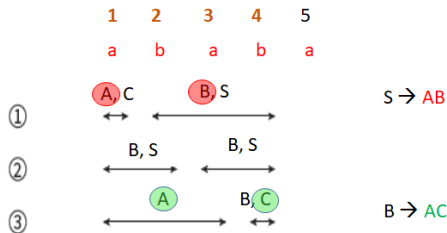
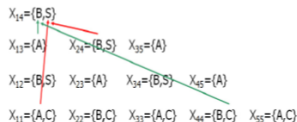
It remains to consider $0^p 1 \dots 1 \overbrace{1 \dots 1 0 \dots 0}^{vxy} 0 \dots 0 1^p$. But then $uxz = 0^p 1^i 0^j 1^p$ with $i < p$ or $j < p$ for $|vy| > 0$. $uxz \notin D$. □

Testing Membership (Cocke-Younger-Kasami Algo.)

- Test " $w = a_1 \dots a_n \in L(G)?$ ", assuming G in CNF.
- Algorithm (CYK) is a good example of dynamic programming and runs in time $O(n^3)$, where $n = |w|$.
 - We construct an n -by- n lower triangular array of sets of variables.
 - $X_{ij} = \{A \mid A \xRightarrow{*} w_{i,j}\}$, where $w_{i,j} = a_i \dots a_j$. Finally, ask if $S \in X_{1n}$.
- Basis: $X_{ii} = \{A \mid A \rightarrow a_i \text{ is a production}\}$
- To compute X_{ij} inductively, try all possible ways of splitting $a_i \dots a_j$ into substrings.

Example

Grammar: $S \rightarrow AB, A \rightarrow BC \mid a, B \rightarrow AC \mid b, C \rightarrow a \mid b$
 String $w = ababa$



$\{x^n y^n \mid n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}$ Not a DPDA Language

Theorem 20

CFL $L = \{x^n y^n \mid n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}$ cannot be acceptable by a DPDA.

Proof.

Assume, otherwise, that DPDA M accepts L . We construct a new DPDA M_0 which consists of "two modified copies" M_1 and M_2 of M in the following way:

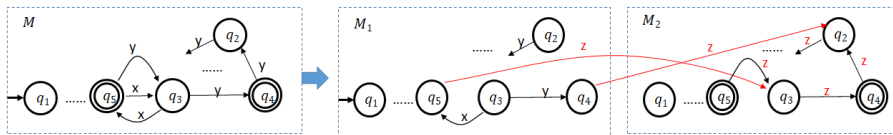
- the initial state of M_0 is the initial state of M_1 , and the final states of M_0 are the final states of M_2 ,
- remove all x transitions from M_2 ,
- replace all the y transitions of M_2 with z transitions,
- for each y transition emanating from an accept state of M_1 , remove that transition and add a z transition to its "copy" in M_2 ,
- remove all x transitions emanating from accept states of M_1 ,
- update on the stack remains unchanged.



$\{x^n y^n \mid n \geq 0\} \cup \{x^n y^{2n} \mid n \geq 0\}$ Not a DPDA Language

Proof.

- Claim: $L(M_0) \subseteq x^* y^* z^*$
 - The prefix before entering M_2 must be accepted by M_1 . Hence, the prefix must be of the form $x^n y^n$ or $x^n y^{2n}$.
- If M_0 accepts $x^n y^n z^i$ ($i \geq 1$), then M must accept $x^n y^{n+i}$, only possible if $i = n$.
- Hence, M_0 accepts $\{x^n y^n z^n \mid n \geq 1\}$ – a contradiction. □



Deterministic Context-Free Languages

- Deterministic PDA: in state q reading a with x at top of stack, at most one transition can apply.
 - if $q \xrightarrow{\epsilon, \epsilon \rightarrow \epsilon} p$ exists, it is the only transition in q ;
 - $q \xrightarrow{a, x \rightarrow y} p$ cannot co-exist with either $q \xrightarrow{a, \epsilon \rightarrow z} p'$ or $q \xrightarrow{a, x \rightarrow z} p''$.

Theorem 21

$\text{Reg} \subsetneq \text{DCFL} \subsetneq \text{CFL}$

Theorem 22

- DCFLs are closed under complement, union/intersection with regular languages.
- DCFLs are not closed under union, intersection, concatenation.

Closure Properties of DCFL

Proof.

- (Complementation - YES) [Proof Idea] swap accept/non-accept states but need to make sure that DPDA reads the entire string.
 - $L = \{a^n b^n c^n \mid n \geq 0\}$ is not CF, yet its complement \bar{L} is CF (Why?).
So $\overline{\{a^n b^n c^n \mid n \geq 0\}}$ is NOT a DCFL.

- (Union/Concatenation with Regular - YES):

Proof Idea

DPDA \times DFA \rightarrow DPDA.

- (Union - NO) $\bar{L} =$

① $\{a^i b^j c^k \mid i \neq j\} \cup \{a^i b^j c^k \mid i \neq k\} \cup \{a^i b^j c^k \mid j \neq k\} \cup$

② $\{\text{anything with } ba, cb, ca\}$

Each of the above four sub-languages is DCFL. However, (1) is not DCFL; otherwise, \bar{L} is DCFL.

- (Intersection - NO): $L \cup M = \overline{\bar{L} \cap \bar{M}}$.



Closure Properties of DCFL

Proof.

- (Concatenation - NO):

- Let $L_1 = \{a^i b^j c^k \mid i \neq j\}$ and $L_2 = \{a^i b^j c^k \mid j \neq k\}$; both are DCFLs.
- $L_3 = 0L_1 \cup L_2$ is a DCFL.
- Claim: $A = 0^*L_3$ is not a DCFL.
 - Suppose A is a DCFL. Then $A \cap 0a^*b^*c^*$ is a DCFL.
 - $A \cap 0a^*b^*c^* = 0L_1 \cup 0L_2$ (a DCFL) implies $L_1 \cup L_2$ is a DCFL. However, $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}} = \overline{\{a^n b^n c^n \mid n \geq 0\}} - \{\text{anything with } ba, cb, ca\}$
 - As $\overline{\{a^n b^n c^n \mid n \geq 0\}}$ is not a DCFL, $L_1 \cup L_2$ is not a DCFL – a contradiction.



Note:

- If L is a DCFL and R is regular, LR is always a DCFL but RL may not be a DCFL
- DCFLs are interesting as they are closed under complementation, but not closed under union, intersection, concatenation.

Using Idea behind PDA \Rightarrow CFG to Show Closure with Regular Sets

Theorem 23

CFLs are closed under intersection with regular languages.

Proof.

Let $G = (V, \Sigma, R, S)$ be a CFG in CNF and $N = (Q, \Sigma, \delta, q_0, \{q_f\})$ be an NFA with a unique accept state. We construct $G' = (V', \Sigma, R', S')$ as follows. A variable in V' is of the form (q_i, A, q_j) , where

$q_i, q_j \in Q, A \in V$

- $S' = (q_0, S, q_f)$,
- $(q_i, S, q_j) \rightarrow \epsilon$ if $S \rightarrow \epsilon$ in R , and $q_j \in \delta(q_i, \epsilon)$,
- $(q_i, A, q_j) \rightarrow a$ if $A \rightarrow a$ in R , and $q_j \in \delta(q_i, a)$,
- $(q_i, A, q_j) \rightarrow (q_i, B, q_k)(q_k, C, q_j)$ if $A \rightarrow BC$ in $R, \forall q_k \in Q$.

Claim: $(q_i, A, q_j) \xRightarrow{*} w$ in G' iff $q_i \xrightarrow{w} q_j$ in N and $A \xRightarrow{*} w$ in G . □

Testing Emptiness

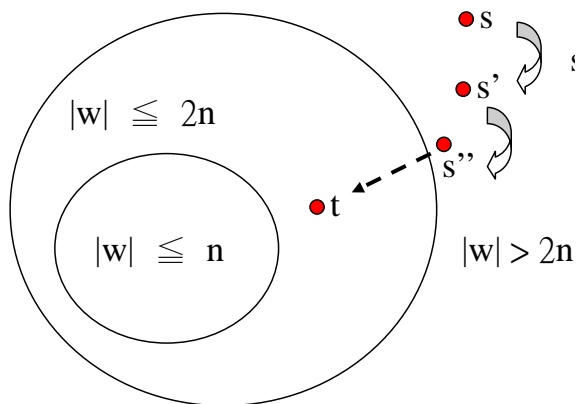
Given a CFG $G = (V, \Sigma, P, S)$ in CNF, construct a set $T = \{A \mid A \xRightarrow{*} w, w \in \Sigma^*\}$ iteratively in the following way:

- 1 Let $T = \{A \mid A \rightarrow a \in P, a \in \Sigma_\epsilon\}$.
- 2 For all rules $B \rightarrow CD \in P$, if $C, D \in T$, then $T = T \cup \{B\}$.
- 3 Repeat Step (2) until no more variable is added to T .

Claim: $S \in T$ iff $L(G) \neq \emptyset$.

Testing Infiniteness

- The idea is essentially the same as for regular languages.
- Use the pumping lemma constant n . If there is a string in the language of length between n and $2n - 1$, then the language is infinite; otherwise not.



Closure of CFLs Under Inverse Homomorphism

Consider a homomorphism $h(0) = aba, h(1) = bc$. Suppose PDA P accepts $ababc$. The following is the way how P' accepts $h^{-1}(ababc) = 01$.

- Each state of P' is of the form $[q, z]$, where q is a state of P and $z \in \{a, b, c\}^*$.
- P' starts in state $[q_0, \epsilon]$, upon reading input 0, P' moves to $[q_0, aba]$; then simulate P 's computation on aba as follows
 - $[q_0, \epsilon] \xrightarrow{0, \epsilon \rightarrow \epsilon} [q_0, aba] \xrightarrow{\epsilon, \alpha \rightarrow \beta} [q_1, ba] \xrightarrow{\epsilon, -} [q_2, a] \xrightarrow{\epsilon, -} [q_3, \epsilon]$
 - in the above, $[q_0, aba] \xrightarrow{\epsilon, \alpha \rightarrow \beta} [q_1, ba]$ simulates $(q_1, \alpha \rightarrow \beta) \in \delta(q_0, a)$ (i.e., specified in the transition function of P).
- in state $[q_3, \epsilon]$, upon reading input 1, P' moves to $[q_3, bc]$; then simulate P 's computation on bc as follows
 - $[q_3, \epsilon] \xrightarrow{1, \epsilon \rightarrow \epsilon} [q_3, bc] \xrightarrow{\epsilon, -} [q_4, c] \xrightarrow{\epsilon, -} [q_5, \epsilon]$, where q_5 is an accept state.
- Hence, P' accepts 01.
- P' updates the stack in the same way as P does.