Chapter 4
Pushdown automata and context-free languages

Introduction

- The language $a^n b^n$ cannot be accepted by a finite automaton
- On the other hand, $L_k = \{a^n b^n \mid n \leq k\}$ is accepted for any given n.
- Finite memory, infinite memory, extendable memory.
- Pushdown (stack) automata: LIFO memory.

4.1 Pushdown automata

- Input tape and read head,
- finite set of states, among which an initial state and a set of accepting states,
- a transition relation,
- an unbounded pushdown stack.

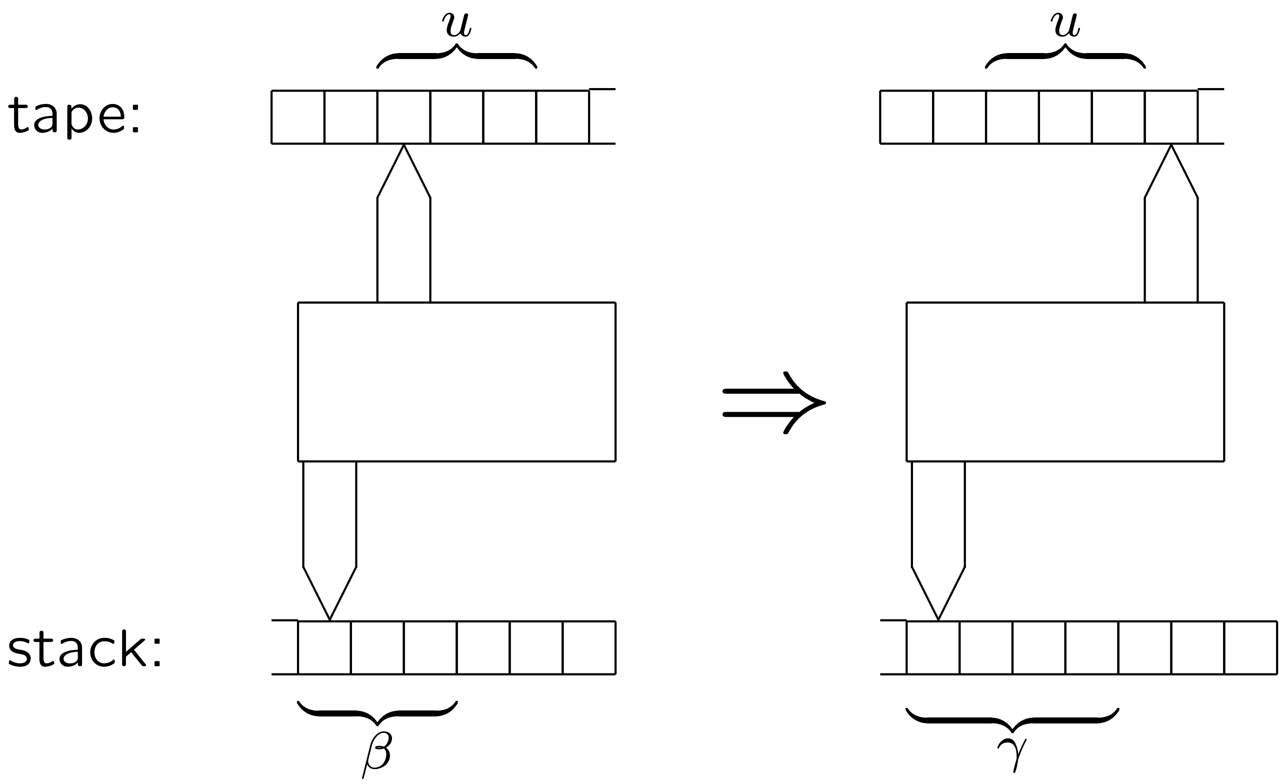
Formalization

7-tuple $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$, where

- Q is a finite set of states,
- Σ is the *input alphabet*,
- Γ is the *stack alphabet*,
- $Z \in \Gamma$ is the *initial stack symbol*,
- $s \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states,
- $\Delta \subset ((Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Gamma^*))$ is the transition relation.

Transitions

$$((p, u, \beta), (q, \gamma)) \in \Delta$$



Executions

The configuration (q', w', α') is *derivable in one step* from the configuration (q, w, α) by the machine M (notation $(q, w, \alpha) \vdash_M (q', w', \alpha')$) if

- $w = uw'$ (the word w starts with the prefix $u \in \Sigma^*$),
- $\alpha = \beta\delta$ (before the transition, the top of the stack read from left to right contains $\beta \in \Gamma^*$),
- $\alpha' = \gamma\delta$ (after the transition, the part β of the stack has been replaced by γ, the first symbol of γ is now the top of the stack),
- $((q, u, \beta), (q', \gamma)) \in \Delta$.

A configuration C' is *derivable in several steps* from the configuration C by the machine M (notation $C \vdash_M^* C'$) if there exist $k \geq 0$ and intermediate configurations $C_0, C_1, C_2, \ldots, C_k$ such that

- $C = C_0$,
- $C' = C_k$,
- $C_i \vdash_M C_{i+1}$ for $0 \leq i < k$.

An *execution of a pushdown automaton* on a word w is a sequence of configurations

$$(s, w, Z) \vdash (q_1, w_1, \alpha_1) \vdash \cdots \vdash (q_n, \varepsilon, \gamma)$$

where s is the initial state, Z is the initial stack symbol, and ε represents the empty word.

A word w is *accepted* by a pushdown automaton $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ if

$$(s, w, Z) \vdash_M^* (p, \varepsilon, \gamma), \text{with } p \in F.$$

Examples

$$\{a^n b^n \mid n \geq 0\}$$

- $Q = \{s, p, q\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{A\}$,
- $F = \{q\}$ and Δ contains the transitions

$$(s, a, \varepsilon) \to (s, A)$$
$$(s, \varepsilon, Z) \to (q, \varepsilon)$$
$$(s, b, A) \to (p, \varepsilon)$$
$$(p, b, A) \to (p, \varepsilon)$$
$$(p, \varepsilon, Z) \to (q, \varepsilon)$$

The automaton $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ described below accepts the language

$$\{ww^R\}$$

- $Q = \{s, p, q\}$,
- $\Sigma = \{a, b\}$,
- $\Gamma = \{A, B\}$,
- $F = \{q\}$ and Δ contains the transitions

$$(s, a, \varepsilon) \rightarrow (s, A)$$
$$(s, b, \varepsilon) \rightarrow (s, B)$$
$$(s, \varepsilon, \varepsilon) \rightarrow (p, \varepsilon)$$
$$(p, a, A) \rightarrow (p, \varepsilon)$$
$$(p, b, B) \rightarrow (p, \varepsilon)$$
$$(p, \varepsilon, Z) \rightarrow (q, \varepsilon)$$

Context-free languages

Definition:
A language is context-free if there exists a context-free grammar that can generate it.

Examples

The language $a^n b^n$, $n \geq 0$, is generated by the grammar whose rules are

1. $S \rightarrow aSb$

2. $S \rightarrow \varepsilon$.

The language containing all words of the form ww^R is generated by the grammar whose productions are

1. $S \to aSa$
2. $S \to bSb$
3. $S \to \varepsilon$.

The language generated by the grammar whose productions are

1. $S \to \varepsilon$
2. $S \to aB$
3. $S \to bA$
4. $A \to aS$
5. $A \to bAA$
6. $B \to bS$
7. $B \to aBB$

is the language of the words that contain the same number of a's and b's in any order

Relation with pushdown automata

Theorem
A language is context-free if and only if it is accepted by a pushdown automaton.

Properties of context-free languages

Let L_1 and L_2 be two context-free languages.

- The language $L_1 \cup L_2$ is context-free.
- The language $L_1 \cdot L_2$ is context-free.
- L_1^* is context-free.
- $L_1 \cap L_2$ and $\overline{L_1}$ are not necessarily context-free!
- If L_R is a regular language and if the language L_2 is context-free, then $L_R \cap L_2$ is context-free.

Let $M_R = (Q_R, \Sigma_R, \delta_R, s_R, F_R)$ be a deterministic finite automaton accepting L_R and let $M_2 = (Q_2, \Sigma_2, \Gamma_2, \Delta_2, Z_2, s_2, F_2)$ be a pushdown automaton accepting the language L_2. The language $L_R \cap L_2$ is accepted by the pushdown automaton $M = (Q, \Sigma, \Gamma, \Delta, Z, s, F)$ for which

- $Q = Q_R \times Q_2$,
- $\Sigma = \Sigma_R \cup \Sigma_2$,
- $\Gamma = \Gamma_2$,
- $Z = Z_2$,
- $s = (s_R, s_2)$,
- $F = (F_R \times F_2)$,

- $(((q_R, q_2), u, \beta), ((p_R, p_2), \gamma)) \in \Delta$ if and only if

  $(q_R, u) \vdash^*_{M_R} (p_R, \varepsilon)$ (the automaton M_R can move from the state q_R to the state p_R, while reading the word u, this move being done in one or several steps) and

  $((q_2, u, \beta), (p_2, \gamma)) \in \Delta_2$ (The pushdown automaton can move from the state q_2 to the state p_2 reading the word u and replacing β by γ on the stack).

4.3 Beyond context-free languages

- There exist languages that are not context-free (for cardinality reasons).

- We would like to show that some specific languages are not context-free.

- For this, we are going to prove a form of pumping lemma.

- This requires a more abstract notion of derivation.

Example

1. $S \rightarrow SS$
2. $S \rightarrow aSa$
3. $S \rightarrow bSb$
4. $S \rightarrow \varepsilon$

Generation of $aabaab$:

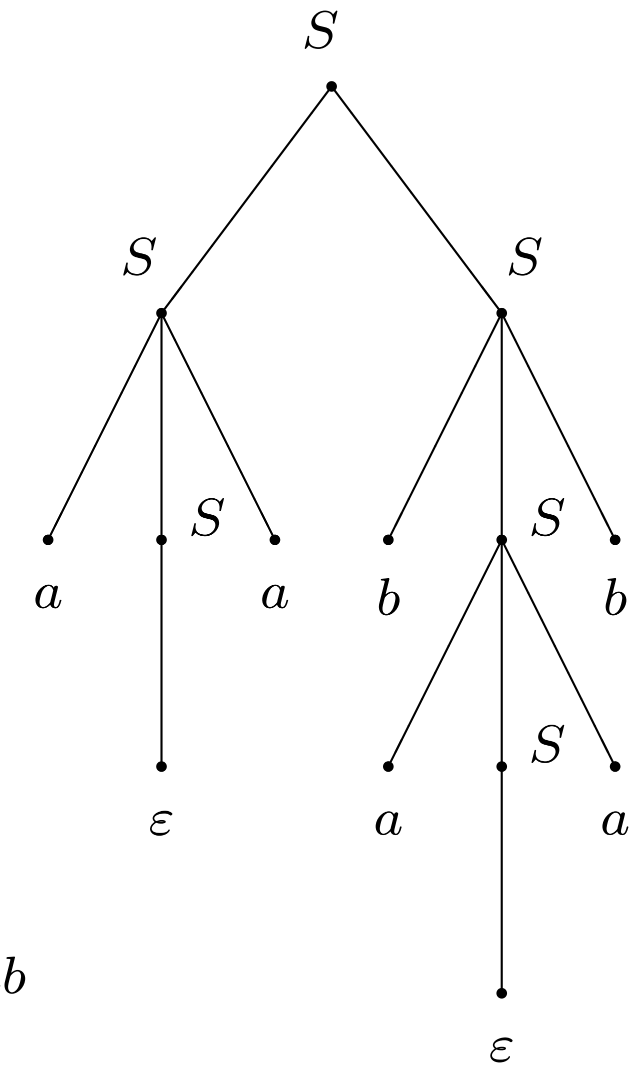
$$S \Rightarrow SS \Rightarrow aSaS \Rightarrow aaS$$
$$\Rightarrow aabSb \Rightarrow aabaSab \Rightarrow aabaab$$

$$S \Rightarrow SS \Rightarrow SbSb \Rightarrow SbaSab$$
$$\Rightarrow Sbaab \Rightarrow aSabaab \Rightarrow aabaab$$

and 8 other ways.

We need a representation of derivations that abstract from the order in which production rules are applied.

The notion of parse tree



Parse tree for $aabaab$

Definition

A parse tree for a context-free grammar $G = (V, \Sigma, R, S)$ is a tree whose nodes are labeled by elements of $V \cup \varepsilon$ and that satisfies the following conditions.

- The root is labeled by the start symbol S.
- Each interior node is labeled by a non-terminal.
- Each leaf is labeled by a terminal symbol or by ε.

- For each interior node, if its label is the non-terminal A and if its direct successors are the nodes $n_1, n_2, \ldots, n_k$ whose labels are respectively $X_1, X_2, \ldots, X_k$, then

$$A \rightarrow X_1 X_2 \ldots X_k$$

must be a production of G.

- If a node is labeled by ε, then this node must be the only successor of its immediate predecessor (this last constraints aims only at preventing the introduction of unnecessary copied of ε in the parse tree).

Generated word

The word generated by a parse tree is the one obtained by concatenating its leaves from left to right

Theorem
Given a context-free grammar G, a word w is generated by G ($S \overset{*}{\underset{G}{\Rightarrow}} w$) if and only if there exists a parse tree for the grammar G that generates w.

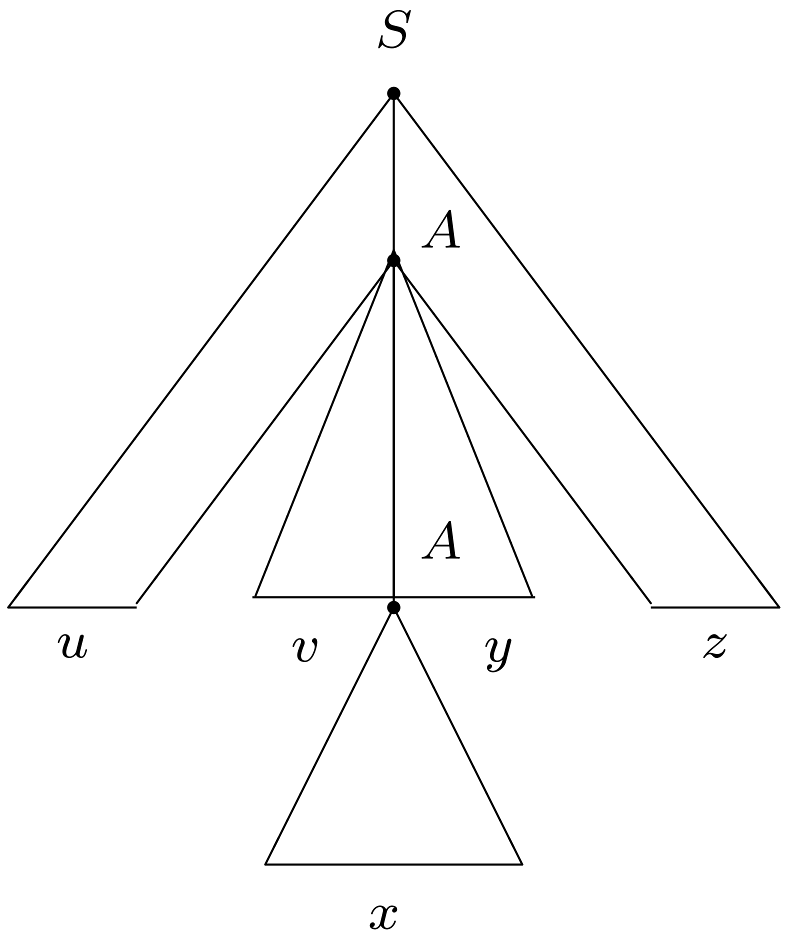
The pumping lemma

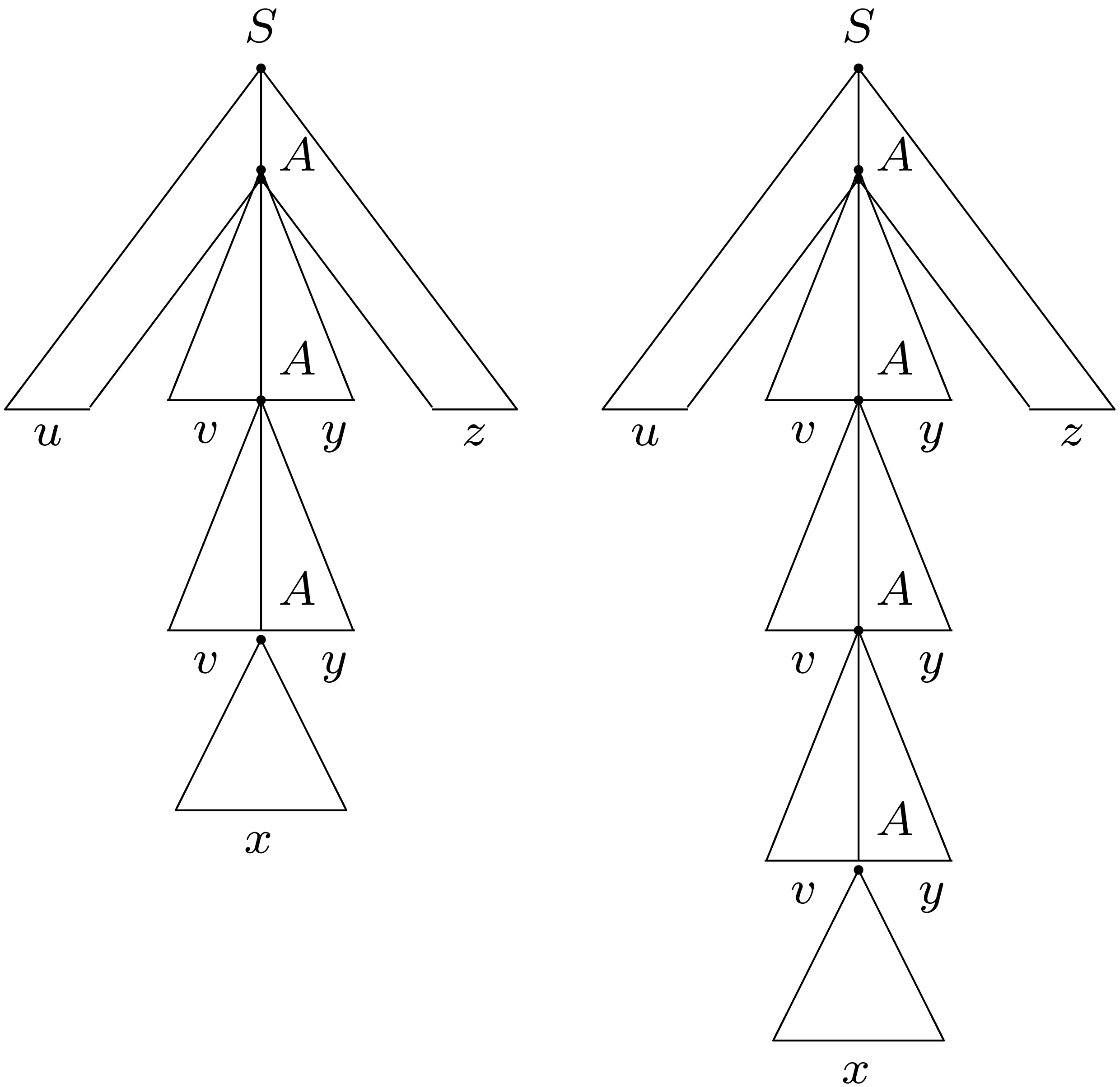
Lemma

Let L be a context-free language. Then there exists, a constant K such that for any word $w \in L$ satisfying $|w| \geq K$ can be written $w = uvxyz$ with v or $y \neq \varepsilon$, $|vxy| \leq K$ and $uv^n xy^n z \in L$ for all $n \geq 0$.

Proof

A parse tree for G generating a sufficiently long word must contain a path on which *the same non-terminal appears at least twice*.


S
A
A
u
v
y
z
x


S
A
A
u
v
y
z
A
v
y
x
S
A
A
u
v
y
z
A
v
y
A
v
y
x

Choice of K

- $p = max\{|\alpha|, A \rightarrow \alpha \in R\}$
- The maximal length of a word generated by a tree of depth i is p^i.
- We choose $K = p^{m+1}$ where $m = |\{V - \Sigma\}|$.
- Thus $|w| > p^m$ and the parse tree contains paths of length $\geq m + 1$ that must include the same non terminal at least twice.
- Going back up one of these paths, a given non terminal will be seen for the second time after having followed at most $m + 1$ arcs. Thus one can choose vxy of length at most $p^{m+1} = K$.
- Note: v and y cannot both be the empty word for all paths of length greater than $m + 1$. Indeed, if this was the case, the generated word would be of length less than p^{m+1}.

Applications of the pumping lemma

$L = \{a^n b^n c^n\}$ is not context-free.

Proof

There is no decomposition of $a^n b^n c^n$ in 5 parts u, v, x, y and z (v or y nonempty) such that, for all $j > 0$, $uv^j xy^j z \in L$. Thus the pumping lemma is not satisfied and the language cannot be context-free.

- v and y consist of the repetition of a unique letter. Impossible

- v and y include different letters. Impossible.

1. There exist two context-free languages L_1 and L_2 such that $L_1 \cap L_2$ is not context-free :

   - $L_1 = \{a^n b^n c^m\}$ is context-free,
   - $L_2 = \{a^m b^n c^n\}$ is context-free, but
   - $L_1 \cap L_2 = \{a^n b^n c^n\}$ is not context-free !

2. The complement of a context-free language is not necessarily context-free. Indeed, the union of context-free languages is always a context-free language. Thus, if the complement was context-free, so would be intersection:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

Algorithms for context-free languages

Let L be a context-free language (defined by a grammar or a pushdown automaton).

1. Given a word w, there exists an algorithm for checking whether $w \in L$.

2. There exists an algorithm for checking if $L = \emptyset$.

3. There is no algorithm for checking if $L = \Sigma^*$.

4. If L' is also a context-free language, there is no algorithm that can check if $L \cap L' = \emptyset$.

Theorem

Given context-free grammar G, there exists an algorithm that decides if a word w belongs to $L(G)$.

Proof

- Pushdown automaton? No, since these are nondeterministic and contain transitions on the empty word.

- Idea: bound the length of the executions. This will be done in the context of grammars (bound on the length of derivations).

Hypothesis: bounded derivations

To check if $w \in L(G)$:

1. One computes a bound k on the number of steps that are necessary to derive a word of length $|w|$.
2. One then explores systematically all derivations of length less than or equal to k. There is a finite number of such derivations.
3. If one of these derivations produces the word w, the word is in $L(G)$. If not, the word cannot be produced by the grammar and is not in $L(G)$.

Grammars with bounded derivations

Problem:

$$A \to B$$
$$B \to A$$

Solution: Grammar satisfying the following constraints

1. $A \to \sigma$ with σ terminal, or
2. $A \to v$ with $|v| \geq 2$.
3. Exception: $S \to \varepsilon$

Bound: $2 \times |w| - 1$

Obtaining a grammar with bounded derivations

1. Eliminate rules of the form $A \to \varepsilon$.

If $A \to \varepsilon$ and $B \to vAu$ one adds the rule $B \to vu$. The rule $A \to \varepsilon$ can then be eliminated.

If one eliminates the rule $S \to \varepsilon$, one introduces a new start symbol S' and the rules $S' \to \varepsilon$, as well as $S' \to \alpha$ for each production of the form $S \to \alpha$.

2. Eliminating rules of the form $A \to B$.

For each pair of non-terminals A and B one determines if $A \stackrel{*}{\Rightarrow} B$.

If the answer is positive, for each production of the form $B \to u$ ($u \notin V - \Sigma$), one adds the production $A \to u$.

All productions of the form $A \to B$ can then be eliminated.

Theorem
Given a context-free grammar G, there exists an algorithm for checking if $L(G) = \emptyset$.

- Idea: search for a parse tree for G.
- One builds parse trees in order of increasing depth.
- The depth of the parse trees can be limited to $|V - \Sigma|$.

Deterministic pushdown automata

Two transitions $((p_1, u_1, \beta_1), (q_1, \gamma_1))$ and $((p_2, u_2, \beta_2), (q_2, \gamma_2))$ are compatible if

1. $p_1 = p_2$,

2. u_1 and u_2 are compatible (which means that u_1 is a prefix of u_2 or that u_2 is a prefix of u_1),

3. β_1 and β_2 are compatible.

A pushdown automaton is deterministic if for every pair of compatible transitions, theses transitions are identical.

Deterministic context-free languages

Let L be a language defined over the alphabet Σ, the language L is deterministic context-free if and only if it is accepted by a deterministic pushdown automaton.

- Not all context-free languages are deterministic context-free.
- $L_1 = \{wcw^R \mid w \in \{a, b\}^*\}$ is deterministic context-free.
- $L_2 = \{ww^R \mid w \in \{a, b\}^*\}$ is context-free, but not deterministic context-free.

Properties of deterministic context-free languages

If L_1 and L_2 are deterministic context-free languages,

- $\Sigma^* - L_1$ is also deterministic context-free.
- There exists context-free languages that are not deterministic context-free.
- The languages $L_1 \cup L_2$ and $L_1 \cap L_2$ are not necessarily deterministic context-free.

Applications of context-free languages

- Description and syntactic analysis of programming languages.
- Restriction to deterministic context-free languages.
- Restricted families of grammars:LR.