

After steps 1,2,3 and 4 we have found an equivalent (except the possible loss of the empty word ε) grammar that does not have ε -productions, unit productions, or useless symbols.

After steps 1,2,3 and 4 we have found an equivalent (except the possible loss of the the empty word ε) grammar that does not have ε -productions, unit productions, or useless symbols.

If the empty word was in the original language and we want to include it in the new grammar we may introduce a **nonrecursive start symbol**. This means that we add a new variable S' , make it the start symbol, and add productions

$$\begin{aligned} S' &\longrightarrow S \\ S' &\longrightarrow \varepsilon \end{aligned}$$

where S is the old start symbol. The new grammar is equivalent to the original one, and it has only one ε -production that can be used only once.

Example. Let us simplify the following grammar:

$$S \longrightarrow Aa \mid CbDS$$

$$A \longrightarrow \varepsilon \mid BA$$

$$B \longrightarrow AA$$

$$C \longrightarrow a$$

$$D \longrightarrow aAD$$

A grammar G is in the **Chomsky normal form** (or CNF) if all productions are of the forms

$$A \longrightarrow BC, \text{ or} \\ A \longrightarrow a$$

where A , B and C are variables, and a is a terminal.

(Right-hand-sides of productions consist of two variables, or one terminal.)

Theorem. Every context-free language L without ε is generated by a grammar that is in the Chomsky normal form.

The conversion from a given context-free grammar to an equivalent CNF grammar is effective.

A grammar G is in the **Chomsky normal form** (or CNF) if all productions are of the forms

$$A \longrightarrow BC, \text{ or} \\ A \longrightarrow a$$

where A , B and C are variables, and a is a terminal.

(Right-hand-sides of productions consist of two variables, or one terminal.)

Theorem. Every context-free language L without ε is generated by a grammar that is in the Chomsky normal form.

The conversion from a given context-free grammar to an equivalent CNF grammar is effective.

Proof. Let G be a grammar that generates L . We may assume that there are no ε -productions or unit productions in G . (Because they can be removed, as we have seen.)

Now all productions with one symbol on the right hand side are terminal productions $A \longrightarrow a$. These are then already in the Chomsky normal form.

The only productions $A \longrightarrow \alpha$ that are not in the Chomsky normal form are **long**, meaning that $|\alpha| \geq 2$.

For every terminal symbol a we introduce a new variable V_a and a production

$$V_a \longrightarrow a$$

Variable V_a hence only derives word a .

(Note that the production $V_a \longrightarrow a$ we add is in the Chomsky normal form.)

The only productions $A \longrightarrow \alpha$ that are not in the Chomsky normal form are **long**, meaning that $|\alpha| \geq 2$.

For every terminal symbol a we introduce a new variable V_a and a production

$$V_a \longrightarrow a$$

Variable V_a hence only derives word a .

(Note that the production $V_a \longrightarrow a$ we add is in the Chomsky normal form.)

Now in each long production $A \longrightarrow \alpha$ we replace every terminal symbol a by the corresponding variable V_a . After this change the grammar generates the same language L as before, and now every long production only contains variables.

Example. The production

$$S \longrightarrow cAbbS$$

will be replaced by production

$$S \longrightarrow V_cAV_bV_bS$$

where V_a and V_b are new variables. New productions $V_a \longrightarrow a$ and $V_b \longrightarrow b$ are also added.

Clearly the new grammar is equivalent to the original one: Every application of the original production

$$S \longrightarrow cAbbS$$

is replaced by a sequence of derivation steps that first uses production

$$S \longrightarrow V_cAV_bV_bS$$

and then applies productions

$$V_c \longrightarrow c \text{ and } V_b \longrightarrow b$$

to replace all occurrences of variables V_c and V_b by terminals c and b .

So far we have constructed an equivalent grammar whose productions are of the forms

$$\begin{aligned} A &\longrightarrow B_1 B_2 \dots B_n \quad \text{and} \\ A &\longrightarrow a \end{aligned}$$

where $n \geq 2$, and $A, B_1, B_2, \dots, B_n$ are variables, and a terminal symbol.

So far we have constructed an equivalent grammar whose productions are of the forms

$$\begin{aligned} A &\longrightarrow B_1 B_2 \dots B_n \text{ and} \\ A &\longrightarrow a \end{aligned}$$

where $n \geq 2$, and $A, B_1, B_2, \dots, B_n$ are variables, and a terminal symbol.

The only productions that are not in CNF are

$$A \longrightarrow B_1 B_2 \dots B_n$$

where $n \geq 3$. For each such production we introduce $n - 2$ new variables $D_1, D_2, \dots, D_{n-2}$, and replace the production by the productions

$$\begin{aligned} A &\longrightarrow B_1 D_1 \\ D_1 &\longrightarrow B_2 D_2 \\ D_2 &\longrightarrow B_3 D_3 \\ &\dots \\ D_{n-3} &\longrightarrow B_{n-2} D_{n-2} \\ D_{n-2} &\longrightarrow B_{n-1} B_n \end{aligned}$$

One application of the original production gives the same result as applying the new productions one after the other. The new grammar is equivalent to the original one.

Remark: The new variables D_i have to be different for different productions.

Example. Let us find a Chomsky normal form grammar that is equivalent to

$$\begin{aligned} S &\longrightarrow SSaA \mid bc \mid c \\ A &\longrightarrow AAA \mid b \end{aligned}$$

Another normal form is so-called **Greibach normal form**, or GNF. A grammar is in the Greibach normal form if all productions are of form

$$A \longrightarrow aB_1B_2 \dots B_n$$

where $n \geq 0$ and $A, B_1, B_2, \dots, B_n$ are variables, and a is a terminal symbol.

(All productions contain exactly one terminal symbol and it is the first symbol on the right hand side of the production.)

Theorem. Every context-free language L without ε is generated by a grammar that is in Greibach normal form.

Proof. Skipped.

Pushdown automata

A **pushdown automaton** (PDA) is an NFA that has access to an **infinite memory, organized as a stack**. It turns out that the family of languages recognized by PDA is exactly the family of context-free languages.

A PDA consists of the following:

- **Stack**. The stack is a word. The PDA has access only to the leftmost symbol of the stack. (This is called the top of the stack.) During one move of the PDA, the leftmost symbol may be removed ("popped" from the stack) and new symbols may be added ("pushed") on the top of the stack.
- **Input tape**. Similar to finite automata: the input word is normally scanned one symbol at a time. But also ε -moves are possible.
- **Finite state control unit**. The control unit is a non-deterministic finite automaton. Transitions may depend on the next input symbol and the topmost stack symbol.

There are two types of moves: normal moves, and ε -moves.

1. **Normal moves**: Depending on

- (a) the current state of the control unit,
- (b) the next input letter, and
- (c) the topmost symbol on the stack

the PDA may

- (A) change the state,
- (B) pop the topmost element from the stack,
- (C) push new symbols to the stack, and
- (D) move to the next input symbol.

2. **ε -moves** don't have (b) and (D), i.e. they are done spontaneously without using the input tape.

Example. Let

$$L = \{a^n b^n \mid n \geq 1\}.$$

A PDA that recognizes L has two states, S_a and S_b , and the stack alphabet contains two symbols, A and Z_0 . In the beginning the machine is in initial state S_a , and the stack contains only one symbol Z_0 , the start symbol of the stack.

Transitions are summarized in this table:

State	Top of stack	Input Symbol		ϵ
		a	b	
S_a	Z_0	Add one A to the stack, stay in state S_a	—	—
S_a	A	Add one A to the stack, stay in state S_a	Remove one A from the stack, go to state S_b	—
S_b	A	—	Remove one A from the stack, stay in state S_b	—
S_b	Z_0	—	—	Remove Z_0 from the stack, stay in state S_b

A word is accepted if and only if the PDA has an empty stack after reading all input symbols.

An **instantaneous description** (ID) records the configuration of a PDA at given time. It is a triple

$$(q, w, \gamma)$$

where

- q is the **state** of the PDA,
- w is the **remaining input**, i.e. the suffix of the original input that has not been used yet, and
- γ is a word, the **content of the stack**. The first letter of γ is the topmost symbol of the stack.

ID contains all relevant information that is needed in subsequent steps of the computation.

We denote

$$(q_1, w_1, \gamma_1) \vdash (q_2, w_2, \gamma_2)$$

if there exists a move that takes the first ID into the second ID.

We denote

$$(q_1, w_1, \gamma_1) \vdash (q_2, w_2, \gamma_2)$$

if there exists a move that takes the first ID into the second ID. We denote

$$(q_1, w_1, \gamma_1) \vdash^* (q_2, w_2, \gamma_2)$$

if there is a sequence of moves (possibly empty) from the first ID to the second ID. We denote

$$(q_1, w_1, \gamma_1) \vdash^+ (q_2, w_2, \gamma_2)$$

if there is a non-empty sequence of moves (at least one move). Finally, we denote

$$(q_1, w_1, \gamma_1) \vdash^n (q_2, w_2, \gamma_2)$$

if the first ID becomes the second ID in exactly n moves.

We operate PDA under two different modes of acceptance, depending on what is more convenient:

- Acceptance by **empty stack**: An input word is accepted if all input letters are consumed and in the end the stack is empty.
- Acceptance by **final state**: We identify some states as final states. An input word is accepted if all input letters are consumed and in the end the PDA is in a final state.

We will see later that the two possible modes of acceptance are **equivalent**: If there is a PDA that recognizes language L by empty stack then there is also a PDA that accepts L by final state, and vice versa. The conversion between the two modes is effective.

Example. The sample PDA to recognize $L = \{a^n b^n \mid n \geq 1\}$ accepts using empty stack.

State	Top of stack	I n p u t S y m b o l		ε
		a	b	
S_a	Z_0	Add one A to the stack, stay in state S_a	—	—
S_a	A	Add one A to the stack, stay in state S_a	Remove one A from the stack, go to state S_b	—
S_b	A	—	Remove one A from the stack, stay in state S_b	—
S_b	Z_0	—	—	Remove Z_0 from the stack, stay in state S_b

An accepting computation on input $aaabbb$:

$(S_a, aaabbb, Z_0) \vdash$

Example. The sample PDA to recognize $L = \{a^n b^n \mid n \geq 1\}$ accepts using empty stack.

State	Top of stack	Input Symbol		ε
		a	b	
S_a	Z_0	Add one A to the stack, stay in state S_a	—	—
S_a	A	Add one A to the stack, stay in state S_a	Remove one A from the stack, go to state S_b	—
S_b	A	—	Remove one A from the stack, stay in state S_b	—
S_b	Z_0	—	—	Remove Z_0 from the stack, stay in state S_b

An accepting computation on input $aaabbb$:

$(S_a, aaabbb, Z_0) \vdash (S_a, aabbb, AZ_0) \vdash (S_a, abbb, AAZ_0) \vdash (S_a, bbb, AAAZ_0)$
 $\vdash (S_b, bb, AAZ_0) \vdash (S_b, b, AZ_0) \vdash (S_b, \varepsilon, Z_0) \vdash (S_b, \varepsilon, \varepsilon)$

Formally, a pushdown automaton M consists of

$$(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

where

- Q is a finite **state set**,
- Σ is the **input alphabet**,
- Γ is the **stack alphabet**,
- $q_0 \in Q$ is the **initial state**,
- $Z_0 \in \Gamma$ is the **start symbol** of the stack,
- $F \subseteq Q$ is the set of **final states**,
- δ is a mapping from

$$Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$$

to finite subsets of

$$Q \times \Gamma^*.$$

Note that $\delta(q, a, Z)$ is a **set** of possible outcomes: PDA are non-deterministic.

1. The interpretation of a transition

$$(p, \gamma) \in \delta(q, a, Z)$$

(where $p, q \in Q$, $a \in \Sigma$, $Z \in \Gamma$, $\gamma \in \Gamma^*$):

In state q , reading input letter a , and Z on the top of the stack, the PDA may go to state p , move to next input symbol, and **replace** Z by γ on top of the stack. The leftmost symbol of γ will be the new top of the stack (if $\gamma \neq \varepsilon$).

This transition means that moves

$$(q, aw, Z\alpha) \vdash (p, w, \gamma\alpha)$$

are allowed (for any $w \in \Sigma^*$, $\alpha \in \Gamma^*$).

2. The interpretation of an ε -transition

$$(p, \gamma) \in \delta(q, \varepsilon, Z)$$

(where $p, q \in Q$, $Z \in \Gamma$, $\gamma \in \Gamma^*$):

In state q and Z on the top of the stack, the PDA may go to state p and replace Z by γ on top of the stack. Note that **no input symbol is consumed**, and the transition can be used regardless of the current input symbol.

This transition means that moves

$$(q, w, Z\alpha) \vdash (p, w, \gamma\alpha)$$

are allowed (for any $w \in \Sigma^*$, $\alpha \in \Gamma^*$).

Let us formally define the two modes of acceptance:

- A word w is accepted by PDA M using **empty stack**, iff

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$$

for some $q \in Q$. Note that q can be any state, final or non-final.

The language recognized using empty stack is denoted by $N(M)$, and it consists of all words accepted using empty stack:

$$N(M) = \{w \in \Sigma^* \mid \exists q \in Q : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

Let us formally define the two modes of acceptance:

- A word w is accepted by PDA M using **empty stack**, iff

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$$

for some $q \in Q$. Note that q can be any state, final or non-final.

The language recognized using empty stack is denoted by $N(M)$, and it consists of all words accepted using empty stack:

$$N(M) = \{w \in \Sigma^* \mid \exists q \in Q : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$$

- A word w is accepted by PDA M using **final states**, iff

$$(q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma)$$

for some $q \in F$, and $\gamma \in \Gamma^*$. Now q has to be a final state, but the stack does not need to be empty.

The language recognized using final states is denoted by $L(M)$, and it consists of all words accepted using final states:

$$L(M) = \{w \in \Sigma^* \mid \exists \gamma \in \Gamma^*, \exists q \in F : (q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma)\}$$

Example. Our sample PDA is

$$M = (\{S_a, S_b\}, \{a, b\}, \{Z_0, A\}, \delta, S_a, Z_0, \emptyset)$$

where

$$\begin{aligned}\delta(S_a, a, Z_0) &= \{(S_a, AZ_0)\} \\ \delta(S_a, a, A) &= \{(S_a, AA)\} \\ \delta(S_a, b, A) &= \{(S_b, \varepsilon)\} \\ \delta(S_b, b, A) &= \{(S_b, \varepsilon)\} \\ \delta(S_b, \varepsilon, Z_0) &= \{(S_b, \varepsilon)\}\end{aligned}$$

and all other sets $\delta(q, a, Z)$ are empty. It does not matter which set we choose as the set of final states, since we use acceptance by empty stack. (Choose, for example $F = \emptyset$.) We have

$$N(M) = \{a^n b^n \mid n \geq 1\}.$$

Example. Let us construct a PDA M such that

$$N(M) = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is a palindrome } \}.$$

Example. Let us construct a PDA M such that

$$N(M) = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is a palindrome} \}.$$

Idea: The PDA will read symbols from the input and push them into the stack. At some point it **guesses** that it is in the middle of the input word, and starts popping letters from the stack and comparing them against the following input letters. If all letters match, and the stack and the input string become empty at the same time, the word was a palindrome.

Example. Let us construct a PDA M such that

$$N(M) = \{w \mid w \in \{a, b\}^* \text{ and } w \text{ is a palindrome} \}.$$

Idea: The PDA will read symbols from the input and push them into the stack. At some point it **guesses** that it is in the middle of the input word, and starts popping letters from the stack and comparing them against the following input letters. If all letters match, and the stack and the input string become empty at the same time, the word was a palindrome.

$$M = (\{q_1, q_2\}, \{a, b\}, \{Z_0, A, B\}, \delta, q_1, Z_0, \emptyset)$$

where δ is

$$\begin{aligned}\delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\} \\ \delta(q_1, a, Z_0) &= \{(q_1, AZ_0), (q_2, AZ_0), (q_2, Z_0)\} \\ \delta(q_1, b, Z_0) &= \{(q_1, BZ_0), (q_2, BZ_0), (q_2, Z_0)\} \\ \delta(q_1, a, A) &= \{(q_1, AA), (q_2, AA), (q_2, A)\} \\ \delta(q_1, b, A) &= \{(q_1, BA), (q_2, BA), (q_2, A)\} \\ \delta(q_1, a, B) &= \{(q_1, AB), (q_2, AB), (q_2, B)\} \\ \delta(q_1, b, B) &= \{(q_1, BB), (q_2, BB), (q_2, B)\} \\ \delta(q_2, b, B) &= \{(q_2, \varepsilon)\} \\ \delta(q_2, a, A) &= \{(q_2, \varepsilon)\} \\ \delta(q_2, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}\end{aligned}$$

State q_1 is used in the first half of the input word, state q_2 in the second half.

$$\begin{aligned}
\delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, a, Z_0) &= \{(q_1, AZ_0), (q_2, AZ_0), (q_2, Z_0)\} \\
\delta(q_1, b, Z_0) &= \{(q_1, BZ_0), (q_2, BZ_0), (q_2, Z_0)\} \\
\delta(q_1, a, A) &= \{(q_1, AA), (q_2, AA), (q_2, A)\} \\
\delta(q_1, b, A) &= \{(q_1, BA), (q_2, BA), (q_2, A)\} \\
\delta(q_1, a, B) &= \{(q_1, AB), (q_2, AB), (q_2, B)\} \\
\delta(q_1, b, B) &= \{(q_1, BB), (q_2, BB), (q_2, B)\} \\
\delta(q_2, b, B) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, a, A) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}
\end{aligned}$$

The three possible outcomes of some transitions have the following roles in accepting computations:

- use the first transition, if the input letter is before the end of the first half of the input word.
- use the second transition, if the input word has even length and the current input letter is the last letter of the first half, and
- use the third transition, if the input word has odd length and the current input letter is exactly in the middle of the input word.

$$\begin{aligned}
\delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, a, Z_0) &= \{(q_1, AZ_0), (q_2, AZ_0), (q_2, Z_0)\} \\
\delta(q_1, b, Z_0) &= \{(q_1, BZ_0), (q_2, BZ_0), (q_2, Z_0)\} \\
\delta(q_1, a, A) &= \{(q_1, AA), (q_2, AA), (q_2, A)\} \\
\delta(q_1, b, A) &= \{(q_1, BA), (q_2, BA), (q_2, A)\} \\
\delta(q_1, a, B) &= \{(q_1, AB), (q_2, AB), (q_2, B)\} \\
\delta(q_1, b, B) &= \{(q_1, BB), (q_2, BB), (q_2, B)\} \\
\delta(q_2, b, B) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, a, A) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}
\end{aligned}$$

Accepting computations for inputs *aba* and *abba*:

$$(q_1, bab, Z_0) \vdash$$

$$(q_1, abba, Z_0) \vdash$$

$$\begin{aligned}
\delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\} \\
\delta(q_1, a, Z_0) &= \{(q_1, AZ_0), (q_2, AZ_0), (q_2, Z_0)\} \\
\delta(q_1, b, Z_0) &= \{(q_1, BZ_0), (q_2, BZ_0), (q_2, Z_0)\} \\
\delta(q_1, a, A) &= \{(q_1, AA), (q_2, AA), (q_2, A)\} \\
\delta(q_1, b, A) &= \{(q_1, BA), (q_2, BA), (q_2, A)\} \\
\delta(q_1, a, B) &= \{(q_1, AB), (q_2, AB), (q_2, B)\} \\
\delta(q_1, b, B) &= \{(q_1, BB), (q_2, BB), (q_2, B)\} \\
\delta(q_2, b, B) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, a, A) &= \{(q_2, \varepsilon)\} \\
\delta(q_2, \varepsilon, Z_0) &= \{(q_2, \varepsilon)\}
\end{aligned}$$

Accepting computations for inputs *aba* and *abba*:

$$(q_1, bab, Z_0) \vdash (q_1, ab, BZ_0) \vdash (q_2, b, BZ_0) \vdash (q_2, \varepsilon, Z_0) \vdash (q_2, \varepsilon, \varepsilon)$$

$$(q_1, abba, Z_0) \vdash (q_1, bba, AZ_0) \vdash (q_2, ba, BAZ_0) \vdash (q_2, a, AZ_0) \vdash (q_2, \varepsilon, Z_0) \vdash (q_2, \varepsilon, \varepsilon)$$

A PDA is called **deterministic** (or DPDA) if every ID has at most one possible move. This means that

- if $\delta(q, \varepsilon, Z)$ is non-empty then $\delta(q, a, Z)$ is empty for every input letter a , and
- All $\delta(q, a, Z)$ and $\delta(q, \varepsilon, Z)$ contain at most one element.

The first condition states that there is no choice between ε -move and non- ε -move. If one can make a move without reading an input letter, then that is the only possible move.

A PDA is called **deterministic** (or DPDA) if every ID has at most one possible move. This means that

- if $\delta(q, \varepsilon, Z)$ is non-empty then $\delta(q, a, Z)$ is empty for every input letter a , and
- All $\delta(q, a, Z)$ and $\delta(q, \varepsilon, Z)$ contain at most one element.

The first condition states that there is no choice between ε -move and non- ε -move. If one can make a move without reading an input letter, then that is the only possible move.

Remark: There exist languages that are recognized by nondeterministic PDA but not by any deterministic PDA. The situation is different than in case of finite automata where determinism was equivalent to non-determinism.

For example, the language of palindromes cannot be recognized by any DPDA.