

Example. Consider the grammar

$$G = (\{S\}, \{a, b\}, P, S)$$

with productions

$$S \longrightarrow \varepsilon \mid aSb$$

that generates $L = \{a^n b^n \mid n \geq 0\}$. The construction gives a PDA M such that $N(M) = L$

Next we prove that PDA accept **only** context-free languages: For a given PDA M we construct an equivalent context-free grammar G .

Theorem. If $L = N(M)$ for some PDA M then there exists a context-free grammar G such that $L = L(G)$.

Proof.

Next we prove that PDA accept **only** context-free languages: For a given PDA M we construct an equivalent context-free grammar G .

Theorem. If $L = N(M)$ for some PDA M then there exists a context-free grammar G such that $L = L(G)$.

Proof. This construction is made complicated by the fact that the PDA may have more than one state. We split the proof in two stages:

Theorem A. If $L = N(M_1)$ for some PDA M_1 then there (effectively) exists a PDA M_2 that has only **one state** such that $L = N(M_2)$.

Theorem B. If $L = N(M_2)$ for some PDA M_2 with **one state**, then there (effectively) exists a context-free grammar G such that $L = L(G)$.

Theorem A. If $L = N(M_1)$ for some PDA M_1 then there (effectively) exists a PDA M_2 that has only **one state** such that $L = N(M_2)$.

Proof.

Theorem A. If $L = N(M_1)$ for some PDA M_1 then there (effectively) exists a PDA M_2 that has only **one state** such that $L = N(M_2)$.

Proof. Idea: The computations of M_1 will be simulated in such a way that the state of M_1 will be stored in the topmost element of the stack in M_2 . So at all times the topmost element of the stack contains both the stack symbol of M_1 and the current state of M_1 .

Then every move of M_1 can be simulated by M_2 since it knows the state and the topmost stack symbol of M_1 . After each step, the next state is written into the new topmost element.

Theorem A. If $L = N(M_1)$ for some PDA M_1 then there (effectively) exists a PDA M_2 that has only **one state** such that $L = N(M_2)$.

Proof. Idea: The computations of M_1 will be simulated in such a way that the state of M_1 will be stored in the topmost element of the stack in M_2 . So at all times the topmost element of the stack contains both the stack symbol of M_1 and the current state of M_1 .

Then every move of M_1 can be simulated by M_2 since it knows the state and the topmost stack symbol of M_1 . After each step, the next state is written into the new topmost element.

But: This is fine as long as at least one new symbol is written in the stack. However, there is one big problem: Where is the next state stored when the stack is only popped and nothing new is written into the stack ? New top of the stack is the symbol that used to be second highest on the stack, and it is not accessible for writing. The new state must have been stored in the stack already when the second highest symbol was pushed into the stack — and this was possibly long before it becomes the topmost symbol.

How do we know long before what is going to be the state of M_1 when a particular stack symbol is revealed to the top of the stack ? Answer: we guess it using non-determinism of M_2 . At the time when the symbol becomes the topmost element of the stack we only need to verify that the earlier guess was correct.

In order to be able to verify the guess it has to be stored also on the stack symbol above. Therefore stack symbols of M_2 need to contain two states of M_1 : one indicates the state of the machine when the symbol is the topmost element of the stack, and the other state that indicates the state of M_1 when the the element below is the topmost element of the stack.

Therefore the stack of M_2 stores triplets

$$Q \times \Gamma \times Q.$$

Symbol $[q, Z, p]$ on top of the stack indicates that that M_1 is in state q with Z is on top of the stack and p will be the state of M_1 when the element below becomes the topmost symbol of the stack.

The stack is kept **consistent**, meaning that the content of the stack will always look like

$$[q_1, Z_1, q_2][q_2, Z_2, q_3][q_3, Z_3, q_4] \dots$$

(The third component of a stack symbol is the same as the first component of the next stack symbol below.)

The values $q_2, q_3, q_4 \dots$ are nondeterministic 'guesses' done by M_2 about the state of M_1 when the corresponding stack elements will get exposed on top of the stack. As the stack shrinks the correctness of the guesses gets verified.

Detailed construction on the blackboard.

Example. Let us construct a one state PDA that is equivalent to the PDA

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

where

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

that recognizes language

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