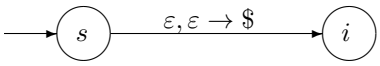
Context-Free Grammars (CFG) $\rightarrow$ Pushdown Automata (PDA)

Algorithm. Let us show how:

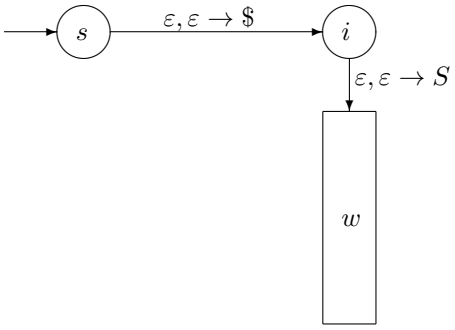
- for each context-free grammar (CFG),
- to design a pushdown automaton (PDA) that accepts exactly all the words generated by the given CFG.

We start at the start state s.

The first thing we do is push the dollar sign into the stack, and go into an intermediate state i:



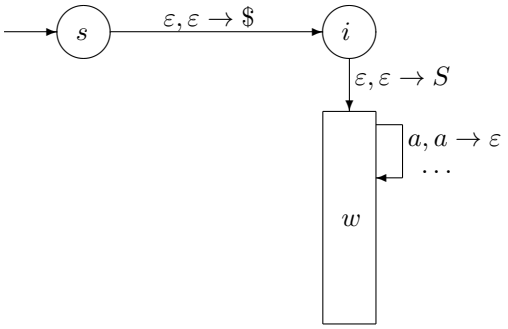
Then, we push the starting variable into the stack, and go to a special *working state* w:



For each terminal symbol a, we add a transition $a, a \rightarrow \varepsilon$, meaning that:

- if we are in the state w we read the symbol a, and the same symbol a is on top of the stack,

- we pop this symbol from the stack:



Finally, for each rule of the type $A \to pqr$ (where each of the symbols p, q, and r is either a variable or a terminal symbol):

- if we are in the state w, and we have A on top of the stack,
- we replace the letter A with pqr.

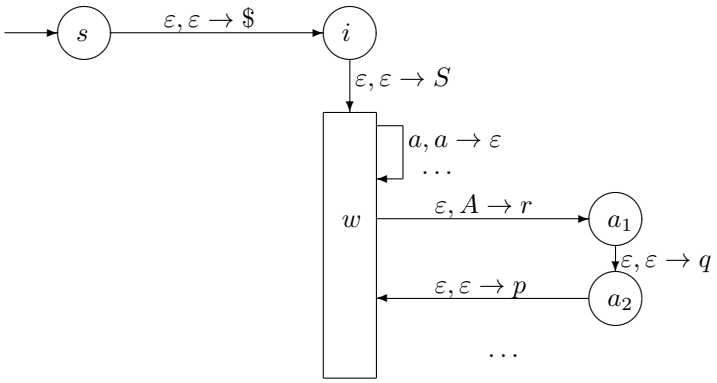
To get the word pqr when popping a stack, we need to place then in the stack in reverse order:

- first r,
- then q,
- and finally, p.

Each transition of the PDA can only push one symbol into the stack. So, to place these three symbols into the stack, we need three transitions:

- first, we use the rule $\varepsilon, A \to r$ and go into the first auxiliary state;
- then, we use the rule $\varepsilon, \varepsilon \to q$ and go into the second auxiliary state;
- finally, we use the rule $\varepsilon, \varepsilon \to p$ and go back into the working state.

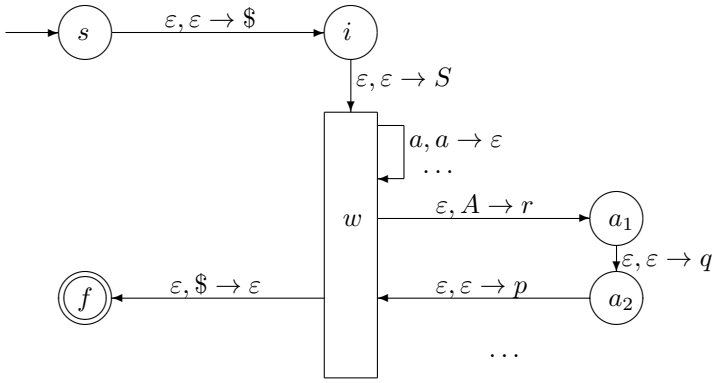
The resulting PDA will look as follows:



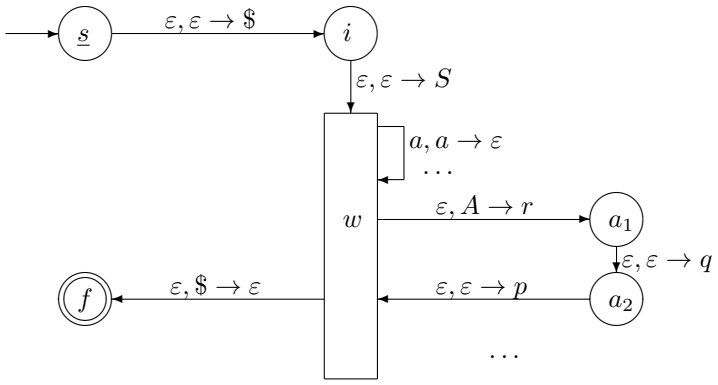
Similarly:

- if we have a rule $A \to p$ with just one symbol in the right-hand side, we need only one transition $\varepsilon, A \to p$, and there is no need for auxiliary states;
- if we have a rule $A \to pq$ with two symbols in the right-hand side, we need two transitions $\varepsilon, A \to q$ and $\varepsilon, \varepsilon \to p$ and one auxiliary state, etc.

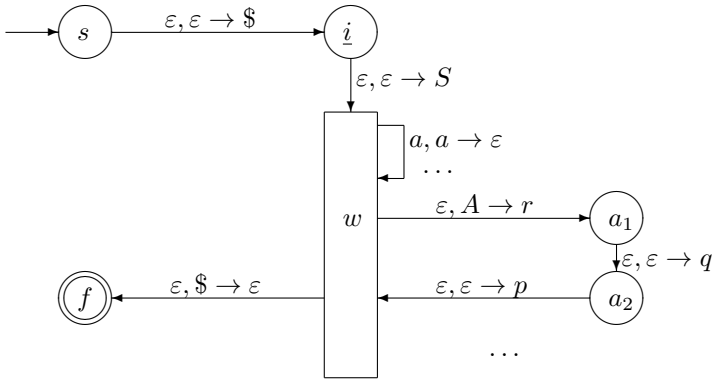
Finally, when we are done, and only the dollar sign is left in the stack, we pop the dollar sign and go to the final state f:



How to transform derivation in a grammar to acceptance by the push-down automaton – general idea: first step. We start at the start state s with an empty stack:

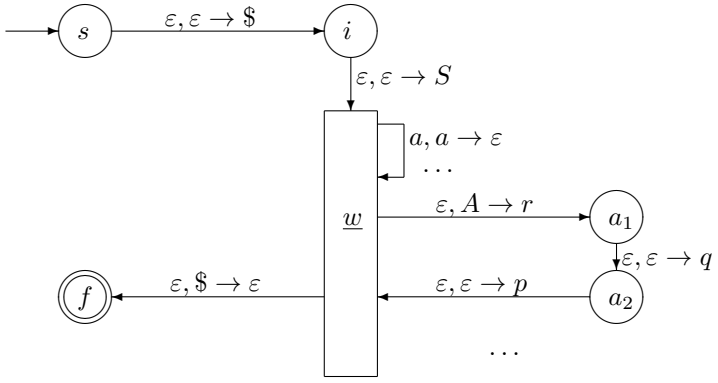


We want to get to the final state f. The only way to leave the start state s is to use the rule $\varepsilon, \varepsilon \to \$$, i.e., to push the dollar sign into the stack and to go to the intermediate state i:



Now, the stack contains the dollar sign:

How to transform derivation in a grammar to acceptance by the push-down automaton – general idea: second step. We are now in the intermediate state i, and we want to go to the final state. The only way to leave the intermediate state is to use the rule $\varepsilon, \varepsilon \to S$, i.e., to push the starting variable (which we denoted S) into the stack and to go to the working state w:

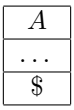


Now, the stack contains two symbols:

How to transform derivation in a grammar to acceptance by the push-down automaton – general idea: following steps. We want to get to the final state f. The only way to get there is to use the rule $\varepsilon, \$ \to \varepsilon$, and the only way to use this rule is to have the dollar sign on top of the stack. So, to get to the final state, we need to get rid of all the symbols which are currently placed in the stack on top of the dollar sign.

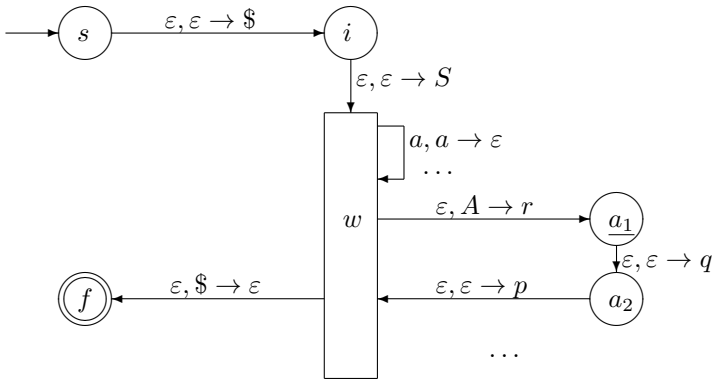
In general, on top of the stack can be either a variable or a terminal symbol. Let us consider these two cases one by one.

What if we have a variable on top of the stack. Let us first consider the case when the top symbol of the stack is a variable, e.g., variable A:

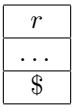


The only way to pop the variable A from the stack is to use one the transitions corresponding to a rule $A \to \ldots$ of the grammar – actually, we need to use the same rule that is used in the derivation of our rule in the grammar to get rid of this variable A. For example, if we use the rule $A \to pqr$, then, as we mentioned earlier, we need to perform three transitions.

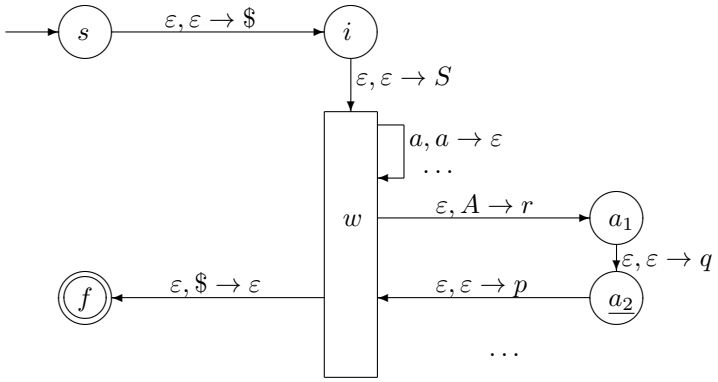
First, we use the rule $\varepsilon, A \to r$, i.e., pop A, push r, and go into the auxiliary state a_1:



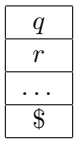
The stack will now take the form:



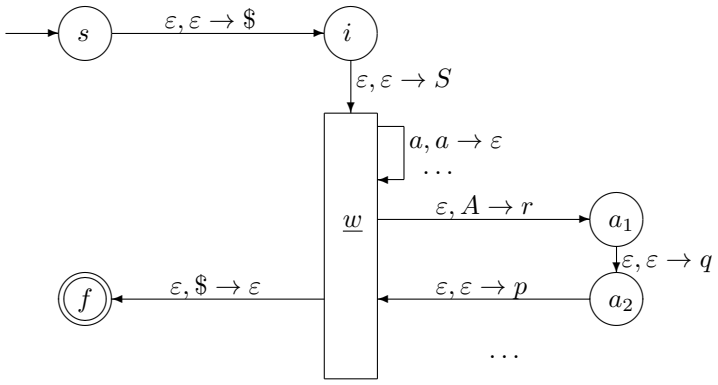
Then, we use the rule $\varepsilon, \varepsilon \to q$, i.e., we push q into the stack and go to the state a_2:



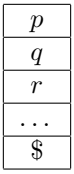
The stack now has the form:



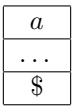
Finally, we use the rule $\varepsilon, \varepsilon \to p$, i.e., we push p and go back to the working state w:



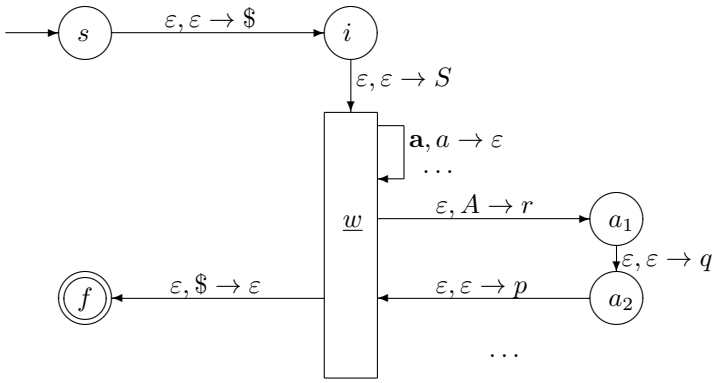
The stack now has the form:



What if we have a terminal symbol on top of the stack? What if we have a terminal symbol – e.g., a – on top of the stack?

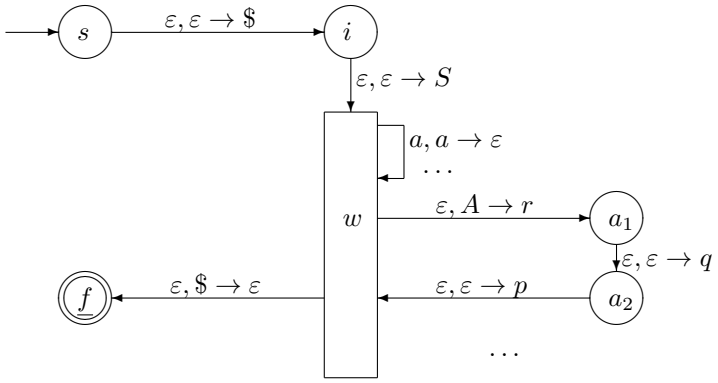


In this case, the only way to pop this symbol is to use the corresponding rule $a, a \to \varepsilon$, i.e., to read the corresponding letter a, and to pop a from the stack:



The stack now does not have a anymore:

How to transform derivation in a grammar to acceptance by the push-down automaton – general idea: final step. At the end, when we are in the working state with only the dollar sign in the stack, we can use the rule $\varepsilon, \$ \to \varepsilon$ to pop the dollar sign and to get to the final state f:

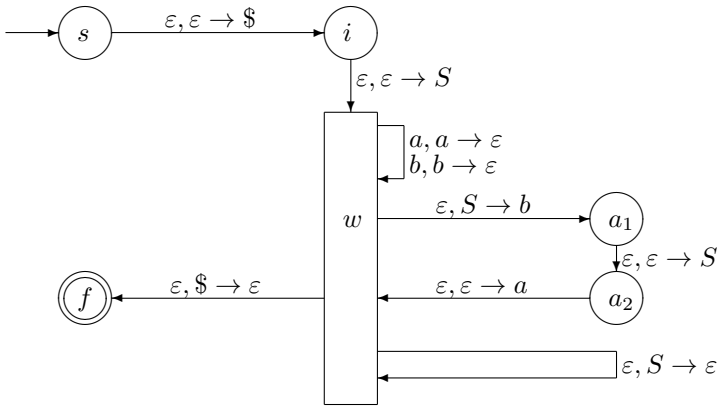


Now, we are in the final state with an empty stack, so the word is accepted.

Example. Let us follow the above procedure on the example of the grammar for the language

$$\{a^n b^n, n = 0, 1, 2, \ldots\} = \{\Lambda, ab, aabb, \ldots\}$$

with two terminal symbols a and b and two rules $S \to \varepsilon$ and $S \to aSb$:



How to transform derivation in a grammar to acceptance by the push-down automaton: example. Let us show how this is done on the example of the word *aabb* generated by the above automaton:

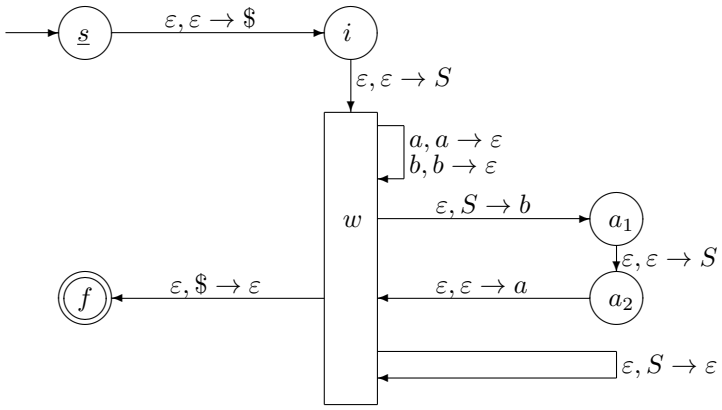
$$\underline{S} \to a\underline{S}b \to aa\underline{S}bb \to aabb.$$

To make this derivation clearer, let us mark the variable S corresponding to different transitions by subscripts:

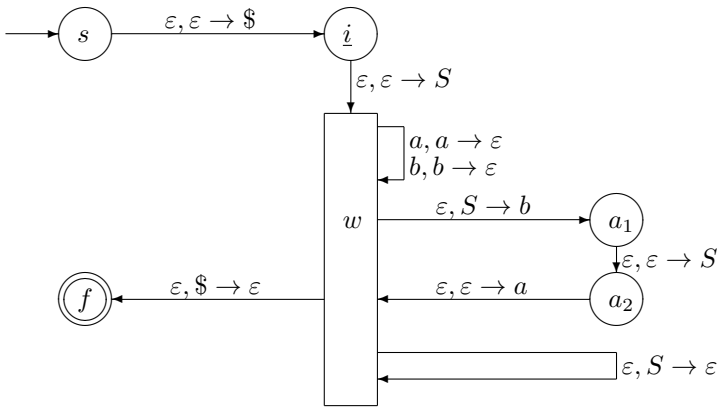
- we start with the first occurrence S_1 of the variable S;

- we then use the rule $S_1 \to aS_2b$ whose right-hand side contains the second occurrence S_2 of the variable S;
- this occurrence, in its turn, gets transformed into $S_2 \to aS_3b$ for yet another occurrence S_3 of the same variable S; so far, the derivation takes the form
$$S_1 \to aS_2b \to aaS_3bb;$$
- finally, to the occurrence S_3, we apply the rule $S_3 \to \varepsilon$, resulting in the desired derivation:
$$S_1 \to aS_2b \to aaS_3bb \to aabb.$$

Let us now trace what our pushdown automaton will do. We start in the state s with an empty stack:

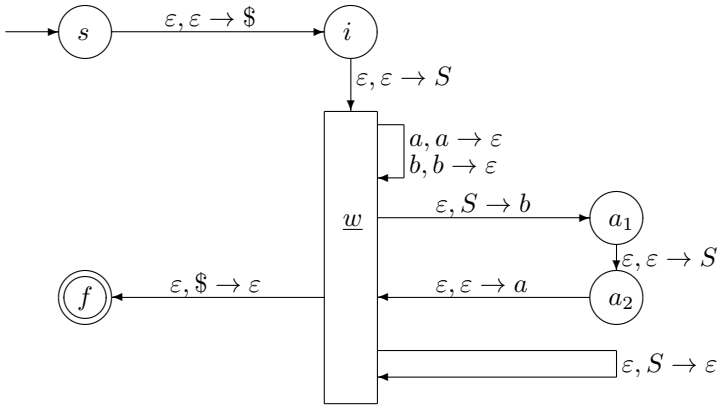


The only thing we can do when in the state s is push the dollar sign into the stack and get to the intermediate state i:



The contents of the stack is as follows:

When we are in the state i, the only thing we can do is push the starting variable S (which corresponds to the first occurrence S_1 of this variable) into the stack and go into the working state w;



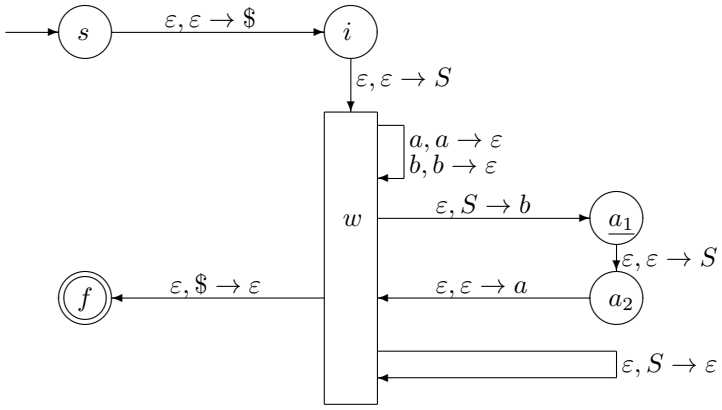
Now, the stack contains the starting variable on top of the dollar sign:

Now that we are in the working state, we can start following the rules that were used to derive the word $aabb$. The first rule was $S \to aSb$, or, to be precise, $S_1 \to aS_2b$. As we have mentioned, this rule is implemented in three steps:

- first, we pop S (that corresponds to the first occurrence S_1) and push the last symbol of the right-hand side – in this cases, the letter b – into the stack, getting into the auxiliary state a_1;
- then, we push S (that corresponds to the second occurrence S_2) into the stack, getting into the auxiliary state a_2;
- finally, we push a into the stack, and go back to the working state w.

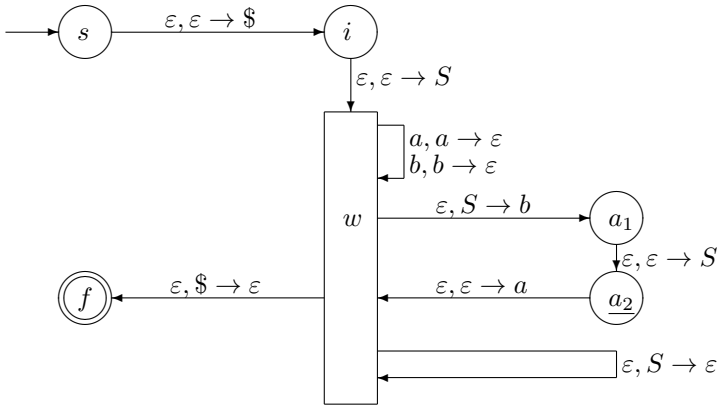
Let us illustrate this step by step.

First, we pop S, push b, and go into the state a_1:

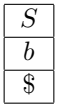


The stack will now have b instead of S:

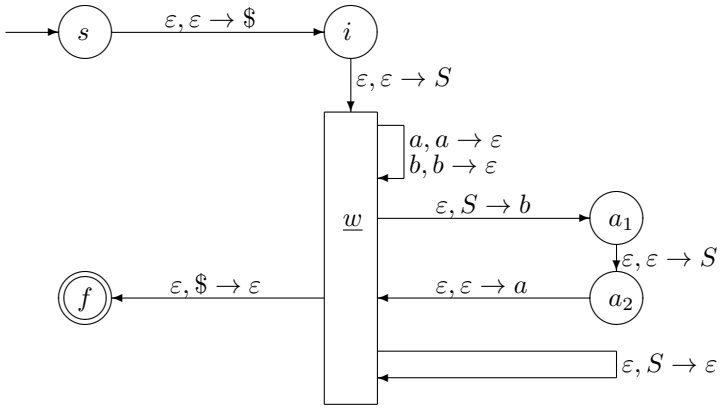
Then, we push S (corresponding to S_2) into the stack and go into the state a_2:



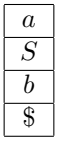
The stack will now have S on top of its previous contents:



Finally, we push a into the stack, and go back to the working state:



The stack will now have letter a at the top:

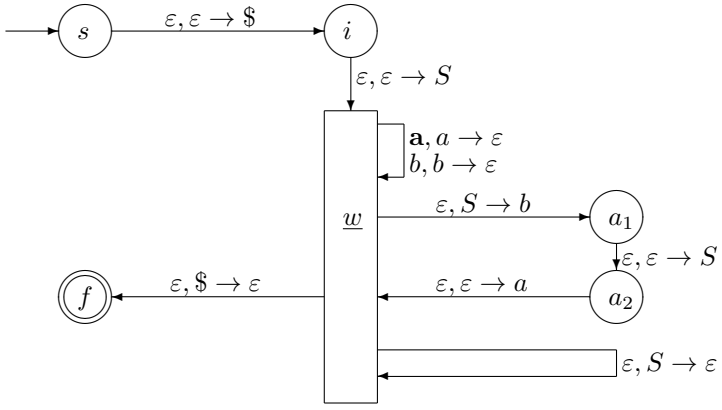


Now, the letter a is top of the stack. The only thing we can do if a terminal symbol is on top of the stack is use one of the rules of the type $x, x \to \varepsilon$ where x stands for the corresponding terminal symbol.

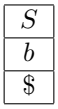
In our case:

- since the terminal symbol on top of the stack is the letter a,
- we need to use the rule $a, a \to \varepsilon$,

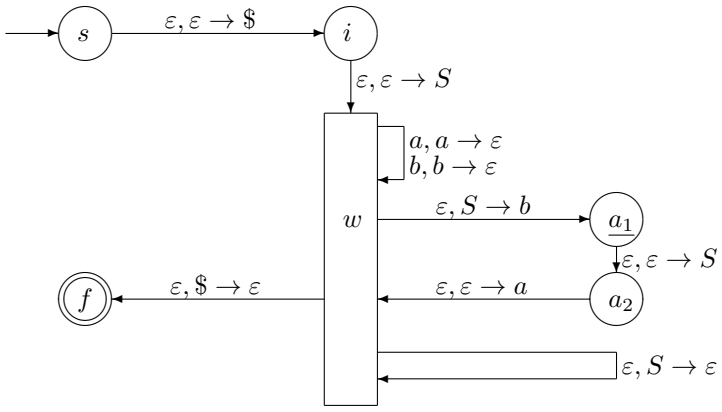
i.e., we read the letter a from the original word **a**abb and pop the top symbol a from the stack:



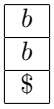
After this popping, the variable S (corresponding to S_2) will be on top of the stack:



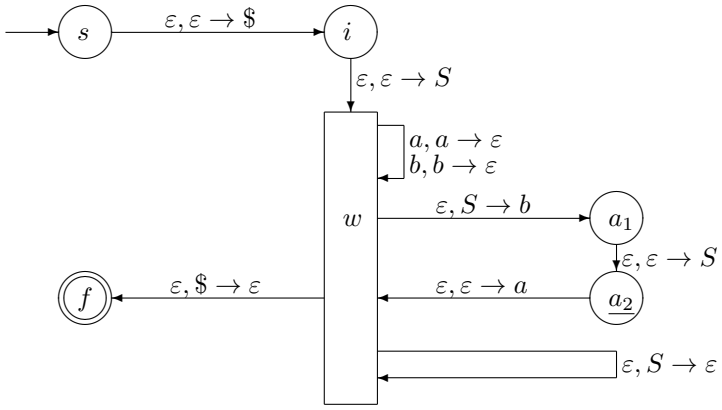
According to the original derivation of the word $aabb$, to get rid of the second occurrence S_2 of the variable S, we also use the rule $S \to aSb$, or, to be precise $S_2 \to aS_3b$. So, similarly to what we have before when we used this rule, first, we pop S, push b, and go to the state a_1:



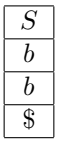
Now, we have b instead of S on top of the stack:



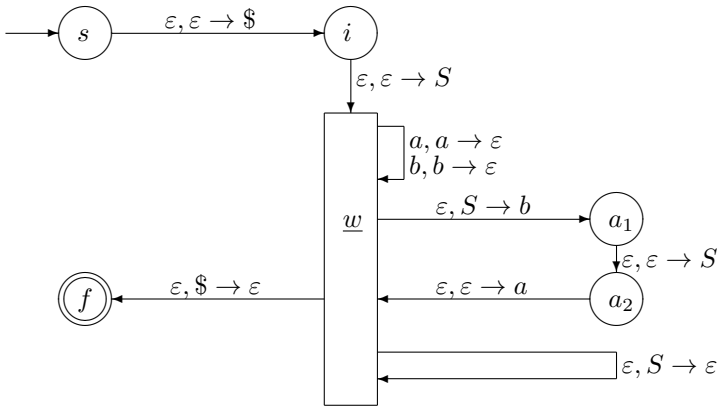
After that, we push S (that corresponds to the third occurrence S_3) and go to state a_2:



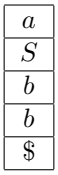
The stack now has the form:



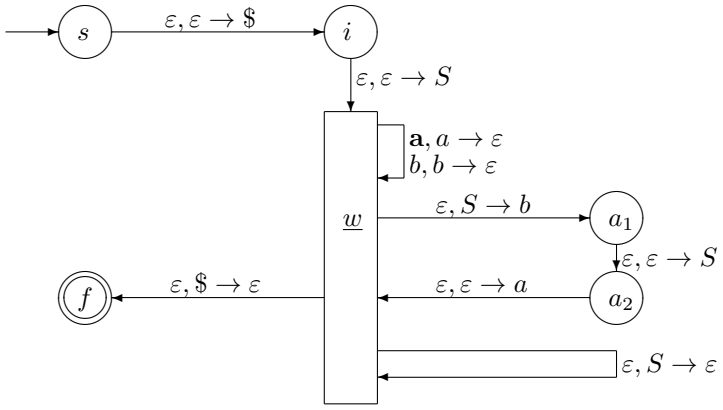
Finally, we push a into the stack and go back to the working state w:



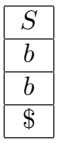
Now, the stack has the following form:



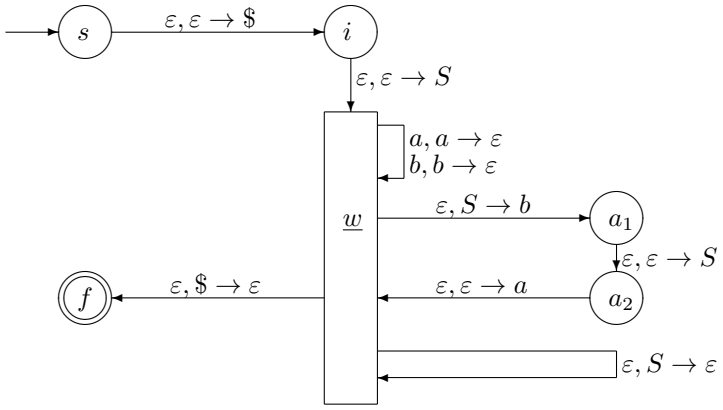
Now again, we have a terminal symbol a on top of the stack, so the only thing we can do is use the rule $a, a \to \varepsilon$: we read the second letter a of the word $a\mathbf{a}bb$ (the first one we have already read, so the cursor points to the second one) and pop a from the stack. As a result, we get the following state:



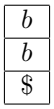
The stack has the following form:



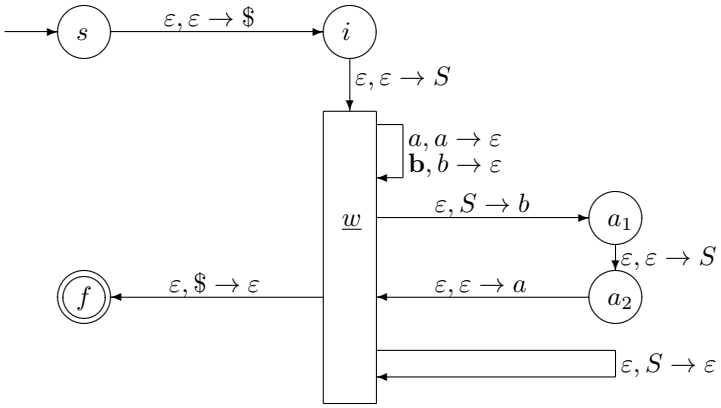
On top of the stack is the variable S corresponding to the third occurrence S_3. To get rid of this variable, in the original derivation of the sequence, we used the rule $S \to \varepsilon$ – or, to be more precise, $S_3 \to \varepsilon$. This rule of the grammar corresponds to the transition $\varepsilon, S \to \varepsilon$ of the pushdown automaton, i.e., we pop S from the stack:



The stack now takes the following form:

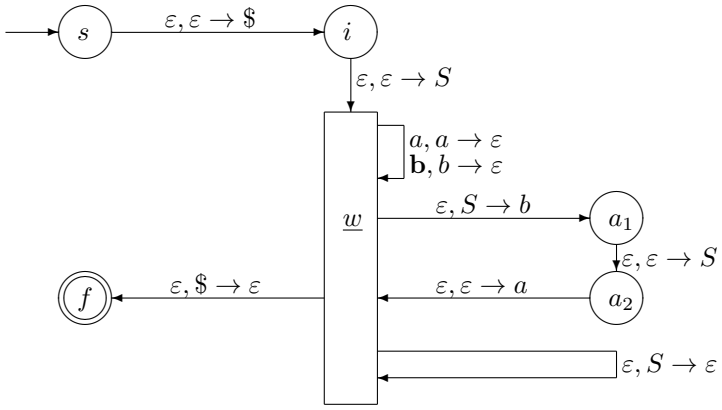


There is a terminal symbol on top of the stack – in this case, the symbol b. We want an empty stack at the end. The only way to get rid of b is to use the rule $b, b \to \varepsilon$, i.e., to read the next letter b from the word $aa\mathbf{b}b$, and to pop b from the stack:



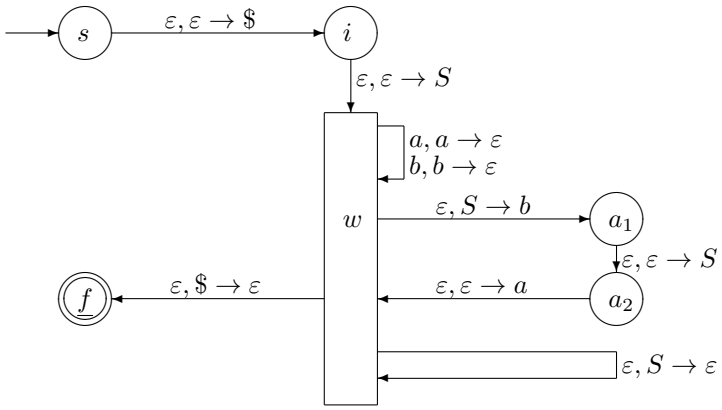
Now, the stack has the following form:

Again, we have a terminal symbol b on top of the stack, so we again use the rule $b, b \to \varepsilon$, i.e., we read the next letter b of the word $aab\mathbf{b}$, and we pop b from the stack:



Now, the stack only contains the dollar sign:

We have read all the letters of the original word, and all we have in the stack is the dollar sign. So now, we can use the rule $\varepsilon, \$ \to \varepsilon$ to pop the dollar sign and to go to the final state:



Now, we are in the final state f with the empty stack. This means that the word *aabb* is accepted by this pushdown automaton.

A graphical description of the transitions. Let us describe, step by step, how the state of the automaton and the contents of the stack change.

To make the description clearer, in addition to the columns that show the current state and the current stack, we also have auxiliary columns corresponding to reading a symbol. In these auxiliary columns, we only describe the symbol which is read, we do not copy the state of the automaton or the state

of the stack – instead, we place a dash indicating that no change was made yet. The changes caused by reading the symbol are described in the following non-auxiliary column:

read							a			
state	s	i	w	a_1	a_2	w	$-$	w	a_1	a_2
stack		$\$$	S $\$$	b $\$$	S b $\$$	a S b $\$$	$-$	S b $\$$	b b $\$$	S b b $\$$

read		a			b		b		
state	w	$-$	w	w	$-$	w	$-$	w	f
stack	a S b b $\$$	$-$	S b b $\$$	b b $\$$	$-$	b $\$$	$-$	$\$$	

Note. Notice that this pushdown automaton is non-deterministic: when we have a variable S on top of the stack, we could, in principle, apply:

- either the transition corresponding to the rule $S \to \varepsilon$
- or the transitions corresponding to the rule $S \to aSb$.

Practice.

- Practice showing that some other words from the language $\{a^n b^n\}$ – e.g., the empty string and the word ab – are also accepted by this pushdown automaton.
- Design a PDA corresponding to another grammar, and trace how a word generated by this grammar will be accepted by this automaton.