

A pushdown automata is a type of automation that use a stack structure, with the remaining part the same as the NFA (non-definite finite state automata). With the respect to the stack the pushdown automata has gained a more greater ability to describe the language. We will in the further section prove that pushdown automata has just the same ability describing as Context free grammar.

0.1 The definition of PDA

What is new in the pushdown automata is that there is a stack that we can operate when we receive a character. When it receive a character of Σ , the automata is going to next state based on three object: 1. the current state; 2. the received character of Σ ; 3. the character on the top of the stack.

If you wrote out the definition of state-transition function, it would be something like: $\delta: Q \times \Sigma \times Z \rightarrow Q$, where Z denote the character in the stack (and of course that Z can equate ϵ).

Let's take the pushdown automata that describe the language $L = \{ww^R\}$ as an example.

Example 0.1.1. *It is clear that we should push a into the stack if we receive a , before we finish going through the string w . And it is the same clear that we should get the top of the stack and check if it is the same as the received character after we go through the string w .*

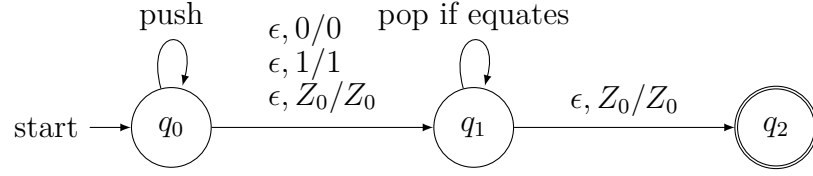
The tricky one is that we have to check these two kinds of situation at the same time, for that every time we get a character, it could be that the w is done or not.

We can use NFA with two state to construct the pushdown automaton we need.

We say that the automata is at q_0 saying that we are at w , and the state q_1 is saying that we are at w^R . And before we receive a character we use a ϵ transition from q_0 to q_1 , after figuring out which, all is clear.

0.1.1 The precise definition of the pushdown automata

A pushdown automaton is a seven tuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.



Γ is the set of the character that appears in the stack

Z_0 is the initial character that located in the stack.

Moreover, the state-transition function should be like:

$$\delta_0: Q \times \Sigma \times \Gamma \rightarrow \mathfrak{P}(Q \times (V \cup T)^*)$$

for that the the pushdown automata is built on a NFA.

Also, the element on the top of the stack is always popped. If you want the stack to remain un-changed, you can define something like $\delta(q_i, a, \alpha) = \{(q_j, \alpha)\}$

Example 0.1.2. Let us just skip the blahblah and draw a diagram that shows how a pushdown automaton work.

0.1.2 Instantaneous Descriptions of a Pushdown automata

We introduce the instantaneous descriptions of PDA to show how a PDA accept a string.

A string is given and the string is the input of the PDA. What will happen in the PDA is that part of the string is received and state changes and stack is pushed into character or the other way. Then the information in the middle of the procedure contains three parts: 1. the state; 2. the string in the stack; 3. the remaining string.

1. the state
2. the string in stack
3. the remaining string

So we can describe the middle state of the whole PDA with a three tuple— (q, w, γ) , where $q \in Q$, $w \in \Sigma^*$, $\gamma \in \Gamma^*$. We use \vdash to indicate that an ID can transit to the other. Then we know that if $\delta(q, a, X)$ contains (p, α) we will know that

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

It is very similar to the function $\hat{\delta}$ in the finite state automata. And it is very similar to the symbol \Rightarrow in the context free grammar. Similarly we use \vdash^* to

indicate that one ID can transit to the other after zero or one or more than one character are received.

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

Theorem 0.1.3 (Deduction principle). $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA. If $(q, x, \alpha) \vdash^* (p, y, \beta)$, then

$$(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma)$$

Proof. The proof is trivial. □

However, it is worth noting that the converse of the thm is not true.

0.2 Pushdown automata and Context Free Grammar

Pushdown automata and context free grammar have the same ability to describe a language. In order to prove that, we shall prove that given an automaton we can construct context free grammar such that $w \in N(P) \implies w \in L(G)$, and that given a context free grammar we can construct an automaton such that $w \in L(G) \implies w \in N(P)$.

0.2.1 From grammar to automaton

The idea to prove is that since a deduction in a grammar is $xA\alpha \xRightarrow{lm} x\beta\alpha$, with $xA\alpha$ empty x , then ϵ -transit $A \rightarrow \beta$ being given, where $x \in T^*$, $\alpha \in (V \cup T)^*$, we use ϵ -transition to deal with $A \rightarrow \beta$. Before dealing with A , we pop all the terminates in the stack, that is if $xA\alpha$ is in the stack, what we receive is the corresponding terminate, and we pop the terminate to get $A\alpha$ in the stack, that is to make A at the top of the stack. We have something like:

$$\delta(q, a, a) = (q, \epsilon)$$

so that we pop the terminate. Moreover, we use a ϵ -transition to achieve $A \rightarrow \beta$, that is

$$\delta(q, A, \epsilon) = (q, \beta)$$

which we complete that $xA\alpha \Rightarrow x\beta\alpha$. You may have noticed that there can be only one state.

Now $\beta\alpha$ is in the stack. If $\beta\alpha = yB\gamma$, then we do the same procedure to make $B\gamma$ in the stack and make another generation to produce $\varphi\gamma$ if $B \rightarrow \varphi$ is given, until there is no variable and therefore no terminate in the stack (cause we keep inputing terminates to pop them out).

Consequently, we have that $\alpha \in T^*(\alpha \in L(G) \implies \alpha \in N(P))$. And therefore we construct an automaton from a grammar. Note that the automaton has only one state and that it is empty-stack accepting.

Example 0.2.1. Given a grammar whose production is $S \rightarrow aAA$, $A \rightarrow aS \mid bS \mid a$, construct an automaton.

While it is a theorem to prove that G and P here are equivalent, we have no room nor time for it.

0.2.2 From automaton to grammar

The procedure of transforming an automaton to grammar is concerning with **Greibach Normal Form**. For example, given an automaton P , let $(r_n, Y_1Y_2 \dots Y_n)$ be contained in $\delta(q, a, X)^1$, that is to say if we are at q , we receive a , X is at the top of the stack, then we go to state r_n and X is popped, $Y_1Y_2 \dots$ is pushed.

What will happen if we receive a at state q , viewing the automaton as a grammar? We actually have

$$[qXr_n] \rightarrow a[qY_1r_1][r_1Y_2r_2] \dots [r_{n-1}Y_nr_n]$$

which is indeed a recursive procedure. If the latter variables are all terminative², then we will produce a string that is accepted by the automaton.

Note that r_i is arbitrarily chosen and it may produce many **useless** variables consequently, and that the grammar is in **Greibach Normal Form**, where every production is like

$$A \rightarrow aB_1B_2 \dots B_n$$

Indeed, the procedure looks like some kind of algorithm in computer, while it actually is. Anyway, this is the main idea of the transformation.

Let the original automaton be P , and let the grammar constructed here be G . It is true that we should check $L(P) = N(G)$. Since that we have poor time here, let us just skip it.

Example 0.2.2. Transfer the automaton to check (if) and (else) into grammar. $P_N = (\{q\}, \{i, e\}, \{Z\}, \delta_N, q, Z)$. $\delta_N(q, i, Z) = (q, ZZ)$, $\delta_N(q, e, Z) = (q, \epsilon)$.

The production of the grammar G are as follows:

1. The only production for S is $S \rightarrow [qZq]$, for there is only one state. If there are n states, then there will be n productions like this one.

¹We use $[pXq]$ to denote that we are at p and after X is gone from the stack, the state goes to q . View $[pXq]$ as variable.

²It could be this word

$S \rightarrow [q_0Z_0p]$ then

From bottom to top to check the variables

2. From $\delta_N(q, i, Z) = (q, ZZ)$, we know that $[qZq] \rightarrow i[qZq][qZq]$. However, if there are n states, there will be n^2 productions in this type. It should look like $[qZq] \rightarrow i[qZq_1][q_2Zq]$. The middle two states can be arbitrary. So there should be n^2 in this type. Similarly, if (q, ZZZ) is contained, there will be n^4 productions out of (q, ZZZ)
3. $\delta_N(q, e, Z) = (q, \epsilon)$, so

$$[qZq] \rightarrow e$$

We can use A instead of $[qZq]$. So the production are

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow iAA \mid e \end{aligned}$$

And it is easy to see that A is equivalent to S . Then we can substitute A with S . Then we got:

$$S \rightarrow iSS \mid e$$