automata

April 12, 2023

Contents

Chapter 1

Pushdown automata

A pushdown automata is a type of automation that use a stack structure, with the remaining part the same as the NFA (non-definite finte state automata). With the repect 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.

1.1 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; 2. the character on the top of the stack.

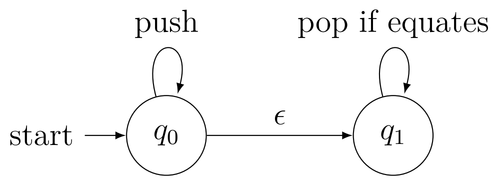
If you wrote out the defintion of state-transition function, it would be something like: $\delta \colon Q \times \Sigma \times Z \to Q$, where Z denote the character in the stack (and of course that Z can equate Q).

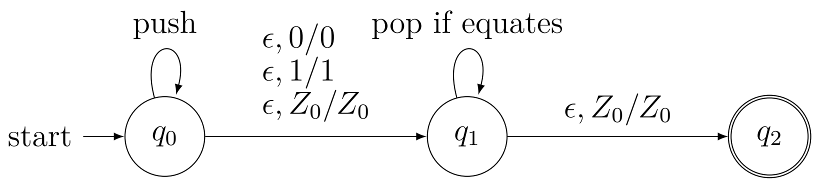
Let's take the pushdown automata that describe the language $L = \{ww^R\}$ as an example.

Example 1.1.1. *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.

1.1.1 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:
>
> $$\delta_0 \colon Q \times \Sigma \times \Gamma \to \mathfrak{P}(Q \times (V \cup T)^*)$$
>
> 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 1.1.2. Let us just skip the blahblah and draw a diagram that shows how a pushdown automaton work.

1.1.2 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 $\overset{*}{\vdash}$ 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 1.1.3 (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.

1.2 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 $w \in L(G) \implies w \in N(P)$.

1.2.1 From grammar to automaton

The idea to prove is that since a deduction in a grammar is $xA\alpha \underset{lm}{\Rightarrow} x\beta\alpha$, with $A \to \beta$ being given, where $x \in T^*$, $\alpha \in (V \cup T)^*$, we use ϵ-transition to deal with $A \to \beta$. 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,

$xA\alpha$
empty x, then ϵ-transit A

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 $A \to \beta$, 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 $B \to \varphi$ 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 1.2.1. Given a grammar whose production is $S \to aAA$, $A \to aS \mid bS \mid a$, 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.

1.2.2 From automaton to grammar

$S \to [q_0 Z_0 p]$ then

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

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

which is indeed a recursive procedure. If the latter variables are all terminative[2], 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

$$A \to aB_1B_2\dots B_n$$

[1]We 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.

[2]It could be this word

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.

From bottom to top to check the variables

Example 1.2.2. 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 $S \to [qZq]$, for there is only one state. If there are n states, then there will be n productions like this one.
2. From $\delta_N(q, i, Z) = (q, ZZ)$, we know that $[qZq] \to i[qZq][qZq]$. However, if there are n states, there will be n^2 productions in this type. It should looks like $[qZq] \to i[qZq_1][q_2Zq]$. The middle two states can be arbitrary. So there should be n^2 in this type. Similiarly, if (q, ZZZ) is contained, there will be n^4 productions out of (q, ZZZ)
3. $\delta_N(q, e, Z) = (q, \epsilon)$, so
$$[qZq] \to e$$

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

$$S \to A$$
$$A \to iAA \mid e$$

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

$$S \to iSS \mid e$$

Chapter 2

The properties of context free grammar

2.1 The Normal Form of grammar

There are two kinds of normal forms that we should know, which are **Chomsky Normal Form** and **Greibach Normal Form**. The definitions have been introduced in previous chapter. You should check them out.

2.2 The simplification of grammar

This section is on its place. However, it has been learned in previous chapter. You may check previous chapter for more info.

2.3 The Pump Lemma of grammar

2.3.1 The content of Pump Lemma

Theorem 2.3.1 (Pump Lemma)**.** Let L be the language of a grammar. Then there exist n that if $|z| \geq n$, then $z = uvwxy$, suit the following conditions:

1. $|vwx| \leq n$. That is to say $|vwx|$ can't be too long.
2. $vx \neq \epsilon$. Since v, x are the pieces to be pumped, v or x should not be zero, that is to say, v and x can be both empty, *and* that is to say $vx \neq \epsilon$.
3. For all $i \geq 0$, uv^iwx^iy is in L.

2.3.2 The application of Pump Lemma

Similarly, the Pump Lemma is used to prove a language L is not a context free language. Let us restate the lemma using the mathematical logic. If L is a context free language then

$$\exists n \in N\ \forall \alpha \in L(\forall uvwxy = \alpha(|vwx| \leq n, vx \neq \epsilon \rightarrow uv^iwx^iy \in L))$$

Let the proposition above be A, then we have L is CFL $\rightarrow A$. Thus, we have $\neg A \rightarrow L$ is not CFL. And $\neg A$ equates

$$\forall n \in N, \exists \alpha \in L, \exists uvwxy = \alpha(\neg(|vwx| \leq n, vx \neq \epsilon \rightarrow uv^iwx^iy \in L))$$

So we have the procedure here, similar to the one in previous chapter about the formal expressions, that we check for every n in $\mathbb{N}$, considering as a random variable[1], and find a $\alpha \in L$, and prove that for all kinds of $uvwxy$, there exists i s.t. uv^iwx^iy is not in L, where we often discuss about the different conditions.

Example 2.3.2. Use pump lemma to show that $L = \{0^n1^n2^n \mid n \geq 1\}$ is not context free language.

There is another example to show that the grammar can't describe the string that have two pairs of equal numbers of symbols.

Example 2.3.3. Let $L = \{0^i1^j2^i3^j\}$. Use pump lemma to prove that L is not context free language.

Mover, since pushdown automata are equivalent to context free grammar, you can easily see that (not prove that) $L = \{ww\}$ is not context free language.

Example 2.3.4. Let $L = \{ww\}$. Use pump lemma to prove that L is not context free language.

Given a n, we shall prove that if $z \in L$, and $z = uvwxy$, we have that uwy does not in L which leads to contradiction. We shall let $z = 0^n1^n0^n1^n$.

1. Let us talk about vwx first. Since $|vwx| \leq n$, we assume that vwx is all in the first block of 0's. Let $|vx|$ be k. Then, $|uwy| = 4n - k$ and moreover, since vwx is all in the first block, uwy starts with $0^{n-k}1^n$ for sure. If $uwy = tt$ for some t, then $|t| = 2n - k/2 \geq 2n - k$, viz., the length of t is longer than that of $0^{n-k}1^n$, and thus t should end with 0. However, uwy ends with 1. If uwy is tt then it should have ended with 0 since t end with 0, which, leads to a contradiction.
2. Suppose that vwx straddles the first block of 0's and the first block of 1's. The same, we assume that uwy can written as tt. Since k, which

[1]not that variable

is the length of vx, is no bigger than n^2, we have $|uwy| \geq 3n$. Thus $|t| \leq 3n/2$. There are two possiblily: 1. vx contains no 1; 2, vx contains at least one 1. For situation 1., the discussion in (1) is also applied. For situation 2., We assert that t does not end with 1^n, for there is at least one 1 in vx and it is deleted from z, while uwy ends with 1^n, which is for sure.

3. The further discussion is omitted here. You can check page 285 of the textbook for help.

2.4 Shin chapter seven

- What is the outline of the chapter?

  1 We learn about how to get Chomsky Normal Form which is very important for us to prove the pump lemma of the context free language.

  2 We talk about the pump lemma of the context free language, and we prove it.

  3 We talk about the close property of the context free language.

2.4.1 Simplification and Chomsky Noraml Form

- What is CNF?

  The grammar with only the productions with the form

$$A \to a \quad \text{or} \quad A \to BC$$

- Why simplification?

  Because we need it to convert the grammar into CNF.

- What to simplify?

  We need to eliminate **useless** symbols and **ϵ-productions** and **unit productions**.

- What is **useful** symbols?

  Symbols that are used in the derivation of a string of the language are **useful**, that is to say if a symbol is useless, it shall not appear during the derivation and thus is indeed useless.

[2]That is because $|vwx| \leq n$

- What is a **generating** symbol?

  Symbols that suits that $A \to w$, where w is in T^* are generating symbols.

- What is a **reachable** symbol?

  Symbols that can be derived from the start variable S.

- What is the procedure to get rid of the useless symbols?

  First, get rid of non-generating symbols.
  Next, get rid of non-reachable symbols.

- Do we have to first get rid of non-generating symbols and then next?

  Yes. You may get wrong answer otherwise.

- Can you prove that?

  Yes. Let us we have a grammar G, and we employ the procedure one, we have G_2, and after next procedure, we have G_1. We need to prove that $V_1 \cup T_1$ are all useful.

  First given a symbol in $V_1 \cup T_1$, X. It should be a symbol after the second procedure, so it should be a symbol that is reachable in G_2. And we can easily notice that X is also generating in G. That is to say $X \underset{G}{\Rightarrow} w$. And we have that $X \underset{G_2}{\Rightarrow} w$. (Since we know that X is generating in G).

  From the fact that X is reachable in G_2, that is to say $S \underset{G_2}{\Rightarrow} \alpha X \beta$, we know that all the symbols that is used in $S \underset{G_2}{\Rightarrow} \alpha X \beta$ are reachable and thus $S \Rightarrow_{G_1} \alpha X \beta$ is true in G_1 since the second procedure does not eliminate the symbols that used in the derivation (that is because they are reachable in G_2). Thus we have proven that if $X \in V_1 \cup T_1$, then we have that $S \Rightarrow \alpha X \beta \Rightarrow \alpha w \beta$

Chapter 3

The closure property of context free grammar

We first make an abstract about the property of the context free language. There is some difference between context free language and formal language. It has been proved that context free language is not closed under intersection and difference. However, the intersection between a context free language and a regular language is always a context free language. Indeed a very intriguing property.

Make an abstract about intersection and homomorphism

3.1 Substitution

A subsitution is a function. It is a function that map the symbol within a string of a language to another **language**. That is to say, $s\colon a \mapsto s(a)$, where $s(a)$ is a language. So it is to say, a s is to make a string in L bigger, by subsititute the symbols with a language.

Who be substituted symbols in a CFL to strings in CFL is still CFL

Definition 3.1.1 (substitutions)**.** A substitution s is a function:

$$s\colon a \mapsto s(a)$$

where $s(a)$ is a language. Moreover, the scope of s can be expanded to L. Let us say $w = a_1a_2\ldots a_n$

$$s(w) = s(a_1)s(a_2)\ldots s(a_n)$$

Theorem 3.1.2 (Closure property under substitution)**.** Given a context free language L, and given an s, if $s(a)$ is context free language for all a in Σ, then $s(L)$ is also a context free language.

Proof. We proof by construction. Given an s and G we can construct a G' such that $L(G') = s(L)$. First let say that the grammar of relative symbol has no common variables viz., G_a of $s(a)$ has independent variables.

Now, since a is a symbol in G, and meanwhile, it is a terminate in G, we subsititute the a with the initial variable S_a in all the production in G. Think about the production that produce $w = a_1 a_2 \dots a_n$. Now it produce $w = S_{a_1} S_{a_2} \dots S_{a_n}$. Further more, the initial variable S_{a_i} can produce a string in $L_{a_i} = s(a_i)$. That is to say after production of G_i, we have $w' = w_1 w_2 \dots w_n$, where $w_i \in s(a_i)$. $w' \in s(L)$ while the production of w' is indeed a production in context free grammar.

So the production of G' is the union of G_{a_i} and the original production of G with terminate being subsituted by the corresponding initial variable. And the variable is the union of all the variables.

Let us look at parse tree. Consider the parse tree that produce w, where the leaf nodes are symbols in Σ. We treat the leaf nodes as the roots of other parse trees. We glue the subtree that produce **a** string G_{a_i} to the leaf node which is exactly a_i. After doing such gluing, we make a parse tree that produce a $w \in s(L)$.

It is indeed true that we should check that $s(L) = L(G)$. But emm... □

3.1.1 The application of substitution theorem

Theorem 3.1.3 (The closure properties of context free language)**.** The context free languages are close under following operations:

1. Union
2. Concatenation
3. Closure and positive closure
4. Homomorphism

where concatenation between two languages is that $L_1 L2 = \{ w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2 \}$ and a homomorphism is function from L to L_2, which suit that $h(w_1 w_2) = h(w_1) h(w_2)$.

3.1.2 The reversal

Context free grammar is closed under reversal, that is to say, if L is context free grammar, then $L^R = \{ w^R \mid w \in L \}$ is context free grammar.

the proof of this property is easy. We shall construct a reverse version of the given grammar G.

3.1.3 Intersection with a regular language

Not closed under intersection but close use intersection with regular language.

exam of $\{ 0^n 1^n 2^n \}$ shows not close.

Example 3.1.4. We already know that $L = \{ 0^n 1^n 2^n \}$ is not a context free language, for that it does not suit the pump theorem. However, $L_1 = \{ 0^i 1^n 2^n \}$, $L_2 = \{ 0^n 1^n 2^i \}$ are context free languages, **and** $L = L_1 \cap L_2$. Consequently, we show that context free language is not closed under intersection.

It is true that for two context free languages L_1, L_2, $L_1 \cap L_2$ could be a language that is not context free language. But we have a theorem to prove that if L is context free, and R is regular, then $L \cap R$ is context free. The proof of the theorem provide a insight into what the nature of intersection is.

Theorem 3.1.5 (Closure property with regular language)**.** If L is a context free language, and R is a regular language, then $L \cap R$ is a context free language.

Proof. The idea is to construct a pushdown automata to accept the language $L \cap R$. The construction is to parrellel a finite state automaton and a pushdown automaton.

Let us say that the finite automaton accept language R is

$$A = (Q_A, \Sigma, \delta_A, q_A, F_A)$$

Let us remind what a finite automaton is. Q_A is the states. Σ is the alphabet, δ_A is the transition function, q_A is the initial state, and F_A is the accepting state. And let us say that the pushdown automaton is

$$P = (Q_P, \Sigma, \Gamma, \delta_P, q_P, Z_0, F_P)$$

where Q_P is the family of the states of pushdown automaton, Σ is the alphabet, Γ is the alphabet in the stack, δ_P is the transiton function, q_P is the initial state of P, Z_0 is the initial character in the stack, F_P is the accepting state.

We construct a new pushdown automaton.

$$(\, Q_A \times Q_P, \Sigma, \Gamma, \delta', (q_A, q_P) Z_0, F' \,)$$

if $\delta_A(q_1, a) = p_1$, $\delta_P(q_2, a, X) = (p_2, \gamma)$, then

$$\delta'(\{ q_1, q_2 \}, a, X) = (\{ p_1, p_2 \}, \gamma)$$

And $\{ p, q \} \in F'$ if and only if p, q are seperately the accepting state in A, P. We can then prove that $w \in L'$ if and only if $w \in L$ and $w \in R$. □

Example 3.1.6. We already know that $L = \{ i^n e^n \}$ is context free language, and that $R = \{ i^* e^* \}$ is regular language.

Consequently, $L \cap R$ is context free language. We can construct the push-down automaton that accepting $L \cap R$.

Theorem 3.1.7 (Other closure properties)**.** Let L be a context free language, R be a regular language.

1. $L - R$ is context free language.
2. $\bar{L}$ could be a language other than context free language.
3. $L_1 - L_2$ could be a language other than context free language.

Proof. 1. $L - R = L \cap \bar{R}$

2. Use contradiction. If $\bar{L}$ is always a context free language. Consider $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$. Since $L_1 \cap L_2$ may be not a context free language, then either $\bar{L}_1$ is not context free language, or $\bar{L}_1 \cup \bar{L}_2$ is context free language while $\overline{\bar{L}_1 \cap \bar{L}_2}$ is not context free language. Consequently, the statement is not true. □