

PUSHDOWN AUTOMATA

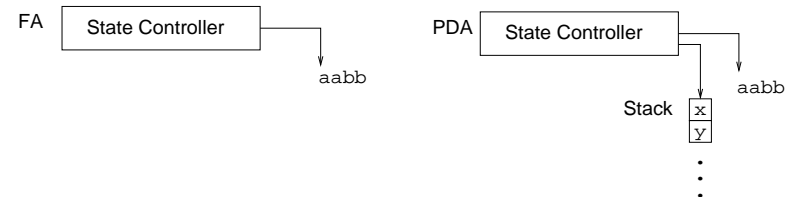
CHAPTER 2.2

Introduction

Pushdown Automaton (PDA) - not a personal digital assistant
- like an NFA with an extra piece of memory, a stack
- equivalent in power to a CFG

So, we have two ways to show that a language is context-free:

1. Write a CFG for it
2. Write a PDA for it



A PDA can recognize $L = \{0^n 1^n \mid n \geq 0\}$

Formal Definition of a PDA

Note: A PDA may use different alphabets for input and for the stack.

Σ - input alphabet

Γ - stack alphabet

Note: We can also use ϵ in the input or on the stack

ϵ in the input or the stack

ϵ in the input causes the machine to change state without reading a symbol of the input

ϵ on the stack causes the machine to change state without reading a symbol from the stack

Formal Definition of a PDA

The transition function: The machine changes state depending on its

1. current state
2. input symbol
3. symbol at the top of the stack

Depending on these values, the machine changes state AND the stack top. That is

1. new state
2. new stack top

$$\delta: Q \times \Sigma \cup \{\epsilon\} \times \Gamma \cup \{\epsilon\} \rightarrow Q \times \Gamma \cup \{\epsilon\}$$

Formal Definition of a PDA

The range is not quite right. The PDA is nondeterministic, so there may be several possible (*state, stack top symbol*) pairs that could be next.

Therefore, the range is really $P(Q \times \Gamma \cup \{\varepsilon\})$.

Example: states $Q = \{q_1, q_2\}$, stack alphabet $\Gamma = \{0, 1\}$

$$Q \times \Gamma \cup \{\varepsilon\} = \{(q_1, 0), (q_1, 1), (q_1, \varepsilon), (q_2, 0), (q_1, 1), (q_1, \varepsilon)\}$$

$$P(Q \times \Gamma \cup \{\varepsilon\}) = \{\emptyset, \{(q_1, 0)\}, \{(q_1, 1)\}, \dots, \{(q_1, 0), (q_1, 1)\}, \{(q_1, 0), (q_1, \varepsilon)\}, \dots\}$$

Formal Definition of a PDA

A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q, \Sigma, \Gamma,$ and F are all finite sets and

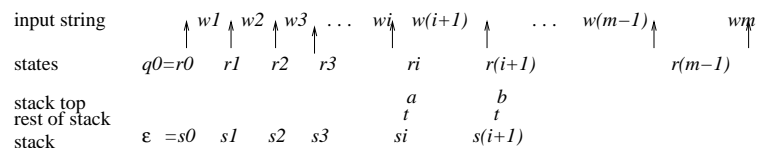
1. Q is the set of states
2. Σ is the input alphabet
3. Γ is the stack alphabet
4. $\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\varepsilon\} \rightarrow P(Q \times \Gamma \cup \{\varepsilon\})$ is the transition function
5. q_0 is the start state
6. $F \subseteq Q$ is the set of accept states

Formal Definition of a PDA

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ computes as follows:

It accepts input w if w can be written as $w = w_1, w_2, \dots, w_m$ where each $w_i \in \Sigma \cup \{\varepsilon\}$ and there is a sequence of states $r_0, r_1, \dots, r_m \in Q$ and a sequence of strings in the stack $s_0, s_1, \dots, s_m \in \Gamma^*$ that satisfy the following conditions

1. $r_0 = q_0$ and $s_0 = \varepsilon$
2. For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$ where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma \cup \{\varepsilon\}$ and $t \in \Gamma^*$
3. $r_m \in F$



Examples

Example 2.14: Give a formal definition of the PDA that recognizes $L = \{0^n 1^n \mid n \geq 0\}$

Let $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$ where

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. $\Gamma = \{0, \$\}$
4. $F = \{q_4\}$
5. start state is: q_1
6. $\delta :$

Input	0			1			ε		
Stack Top	0	\$	ε	0	\$	ε	0	\$	ε
q1									
q2									
q3									
q4									

State Diagram for a PDA

We write state transition

$$q_i \xrightarrow{a,b \rightarrow c} q_j$$

to signify that when the machine is in state q_i reading a from the input, and b is at the top of the stack, it may pop the b at the top of the stack, push a c on the stack top in its place, and transition to state q_j .

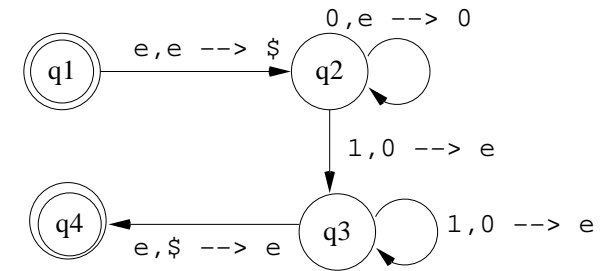
If $q_i \xrightarrow{\varepsilon, b \rightarrow c} q_j$ make the transition without reading a symbol from the input.

If $q_i \xrightarrow{a, \varepsilon \rightarrow c} q_j$ make the transition without reading and popping a symbol from the stack.

If $q_i \xrightarrow{a, b \rightarrow \varepsilon} q_j$ make the transition without pushing a symbol on the stack.

State Diagram for a PDA

Example 2.14: $L = \{0^n 1^n \mid n \geq 0\}$



State Diagram for a PDA

Example 2.16: $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

DO IN CLASS

State Diagram for a PDA

Example 2.18: $L = \{ww^R \mid w \in \{0, 1\}^*\}$

DO IN CLASS

Equivalence of PDAs with CFGs

Of course, we must show that PDAs describe the same set of languages that CFGs do. We know, by definition that CFLs are languages generated by CFGs.

Theorem 2.20: A language is CF iff some PDA recognizes it.

Lemma 2.21: If a language is CF, then some PDA recognizes it.

Lemma 2.27: If a some PDA recognizes a language, then the language is CF.

We will prove 2.21, but not 2.27 (in text).

Equivalence of PDAs with CFGs

Lemma 2.21: If a language is CF, then some PDA recognizes it.

Proof Sketch:

Let A be a CFL, then (by definition) there is a CFG, G , that generates it. That is, $L(G) = A$. We need to show how to convert G into an equivalent PDA P .

That is, assuming that A is a CFL, we know that by definition, there is a CFG, G , that generates it.

We must show how to get the PDA P from G .

Recall what a derivation is: Each step of the derivation yields an intermediate string which is made up of variables and terminals.

Equivalence of PDAs with CFGs

Proof Sketch (cont):

A problem with testing G to see if it generates a particular string w is deciding which substitution rule to use. Therefore, for each rule choice in G have a branch in PDA P (remember, the PDA is nondeterministic).

The PDA starts by writing the start variable of G to the stack.

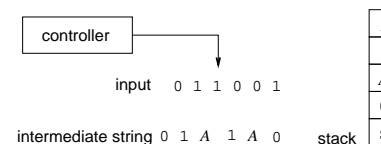
The PDA keeps the intermediate string on the stack.

The PDA goes through a series of intermediate strings making one substitution after another. Eventually, the PDA might produce a string (on the stack) of all terminals. In this case, it accepts the string if it matches the string G generated, otherwise, it rejects it.

Use a leftmost derivation. When get a terminal symbol on the stack top, match it immediately to the input string.

Equivalence of PDAs with CFGs

Proof Sketch (cont):

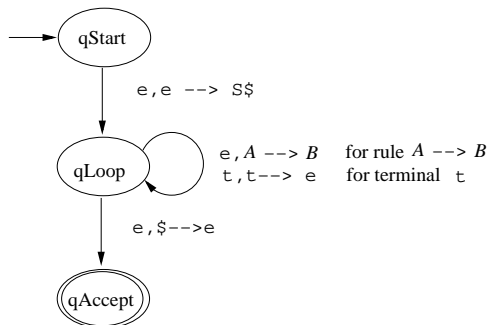


Informal description of PDA P :

1. Place a marker symbol (\$) on the stack and then the start variable of the CFG
2. Repeat
 - (a) If the top of the stack is a variable symbol A , select one of the subrules for A and substitute the rhs of it for A .
 - (b) If the top of the stack is a terminal symbol, t , read the next symbol from the input and compare it to t . If they match, continue, otherwise this PDA fails.
 - (c) If the top of the stack is the marker (\$), enter the accept state. Accept the input string if all the input has been read.

Equivalence of PDAs with CFGs

Proof Sketch (cont):



Equivalence of PDAs with CFGs

Example 2.25: Construct a PDA P for the following CFG G :

$$S \rightarrow a T b \mid b$$

$$T \rightarrow T a \mid \epsilon$$

Equivalence of PDAs with CFGs

The other way: Lemma 2.27: If a PDA recognizes a language, then the language is context-free.

Proof Idea: Have a PDA P and we want to write a CFG G that generates all the strings that P accepts.

We will leave. See p. 119.

In summary: Theorem 2.20 states that PDAs recognize the same class of languages that CFGs can generate. (The class of CFLs).

Where do regular languages fit in?

Every regular language is recognized by a FA, and all FAs are PDAs that ignore their stack. Therefore, all regular languages are also CFLs.

Non-context-free Languages

Guess what? There are languages that are not context-free. There is a version of the Pumping Lemma for CFLs. Section 2.3

Theorem 2.34: Pumping Lemma for CFLs

If A is a CFL, then there is a number p (the pumping length) where, if s is any string in A of at least length p , then s may be divided into 5 parts $s = wxyz$ satisfying the conditions

1. For each $i \geq 0$, $w^i x y^i z \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$

Non-context-free Languages

Theorem 2.34: Pumping Lemma for CFLs

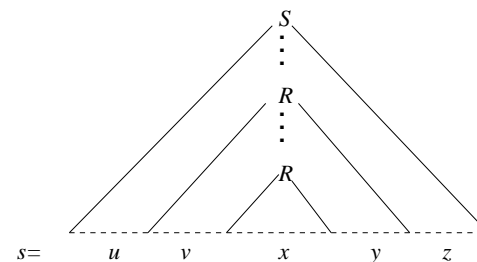
If A is a CFL, then there is a number p (the pumping length) where, if s is any string in A of at least length p , then s may be divided into 5 parts $s = uvxyz$ satisfying the conditions

Proof Sketch:

Let A be CFL and let G be a CFG that generates it. We must show that any sufficiently long string s in A can be pumped and the resulting string will be in A .

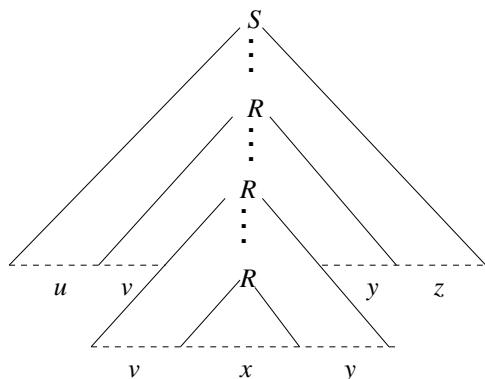
Non-context-free Languages

Roughly, $p \simeq |V|$, the number of variables in the grammar. Let $|s| > p$, then the parse tree for s must contain a variable more than once. Let this be R :



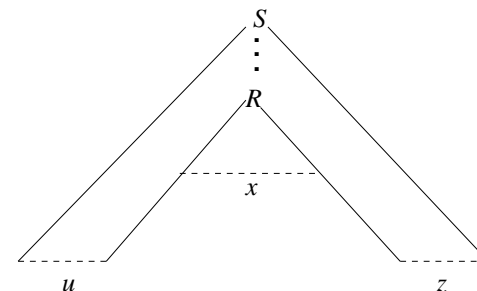
Non-context-free Languages

The substitution for R in terms of itself can be repeated as much as we like:



Non-context-free Languages

or it can be removed and the terminal substitution for R used directly:



Non-context-free Languages

Example 2.36: Show that $B = \{a^n b^n c^n \mid n \geq 0\}$ is not a CFL.

DO IN CLASS

Non-context-free Languages

Example 2.38: Show that $D = \{ww \mid w \in \{0,1\}^*\}$ is not a CFL.

DO IN CLASS