COM364 Automata Theory
Lecture Note*6 - Pushdown Automata

Kurtuluş Küllü

May 2018

Pushdown Automata (PDA) are simply NFA with an infinite stack memory. The stack allows the automaton to remember infinitely many possibilities which wasn't possible with finite number of states. PDA can recognize context-free languages (CFGs), so they are equivalent in power to CFGs. Figure 1 compares the FA model against PDA model. As it can be seen in the figure, the major addition is the stack memory.

We discussed before that a FA cannot recognize $\{0^n1^n \mid n \in \mathbb{N}\}$ because there are infinitely many possibilities for n and a FA cannot remember that value with finite number of states. But, a PDA can recognize this language by writing in its stack the number of 0s it has seen. Such a PDA will work as follows:

- Read symbols from the input like a FA.
- With each 0, push a symbol (0 or X or something else) to the stack.
- Once we start seeing 1s in the input, pop one symbol from the stack for each 1.
- If the stack is empty when we finish the input, accept. If we still have input symbols but the stack becomes empty (more 1s than 0s) or if there are still symbols on the stack but input is finished (more 0s than 1s) the string will be rejected.

Like FA, PDA can be deterministic (DPDA) or non-deterministic (NPDA). However, this time, deterministic and non-deterministic ones are not equal in power. NPDA can recognize some languages that DPDA cannot. The ones that are equivalent to CFGs are non-deterministic ones (NPDA).

Formal Definition

We will be adding the stack and the symbols that can be written to it. Stack symbols don't have to be the same as the input symbols, so now, we have a separate stack alphabet that we will call Γ. Another change is with the transition function (δ). Previously with FA, we only looked at the next input symbol and current state to decide what to do. Now, we also look at the symbol at the top of

*Based on the book "Introduction to the Theory of Computation" by Michael Sipser.

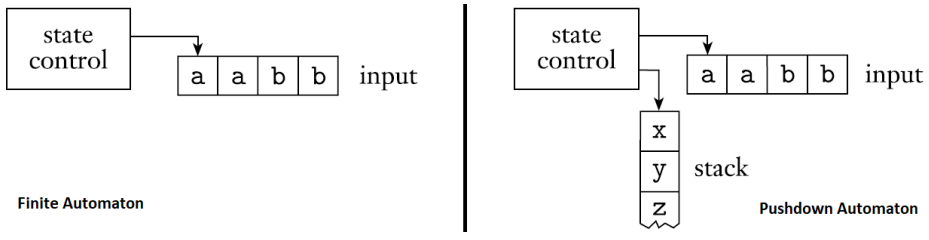

Figure 1: The FA model and PDA model.

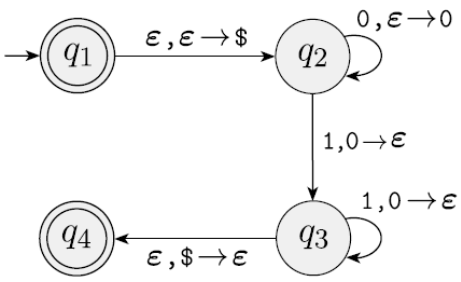

Figure 2: The state diagram of PDA M_1.

the stack. With FA, the result of δ was only the new state (or the set of possible next states). With PDA, we also include stack operations (`push(X)`, `pop()`).

Definition: A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, 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. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transition function (where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$, and P indicates the power set),
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accepting states.

Example: $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, \$\}$, $F = \{q_1, q_4\}$, and δ is given with the table below.

Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

All the empty cells in the table should be considered as $\emptyset$. M_1 is a PDA that can recognize the language $\{0^n 1^n \mid n \in \mathbb{N}\}$. It is given as a state diagram in Figure 2. The state diagram has differences from those we have seen before mainly because of the additional stack operations. Instead of labeling the transitions only with input symbols, we write $a, b \to c$ to indicate that this transition will be applied when the next input is a and top of stack is b (b is popped from the stack), and c is pushed to the stack. In other words, the top of stack was b before and now instead of b we write c. Any of a, b, or c can be ε. If a is ε, the machine may make this transition without reading any symbol from the input. If b is ε, the machine may make this transition without reading and popping any symbol from the stack. If c is ε, the machine does not write any symbol on the stack when going along this transition.

The definition of PDA doesn't give it an automatic mechanism to test if its stack is empty. However, this can be achieved simply by placing a special symbol (like \$) to the stack at the beginning. If we do not use this same symbol for anything else, at any point if we see that symbol at the top of the stack, then we will know that the stack is empty. Go over the state diagram of M_1 in Figure 2 and try to understand how it works by using some example strings in the language such as ε or 000111 and some that are not in the language such as 011, 10, 00011.

Example: A second example PDA is given in Figure 3. Analyze this PDA to understand which strings it accepts and rejects. Finally, try to state the language that it recognizes.

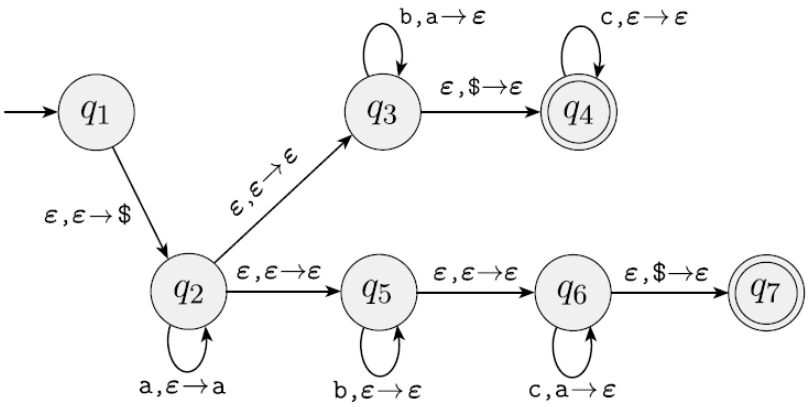

Figure 3: The state diagram of another example PDA.

Example: Give a PDA that recognizes the language $\{ww^R \mid w \in \{0,1\}^*\}$ where w^R means w reversed (written backwards).

Theorem (Equivalance with CFGs): A language is context-free if and only if some PDA recognizes it.

Important Note: We discussed non-context-free languages and the pumping lemma for context free languages at this point in the course. However, these topics are currently omitted in the notes (they will be added later) mainly because they are not in the scope of the second mid-term exam. **But, remember that the pumping lemma for regular languages is included in the exam topics.**

Deterministic PDA

The formal definition for PDA that we have given above allows nondeterminism. DFA and NFA were equivalent in power but for PDA, this is not the case. Nondeterministic PDA (NPDA) are more powerful than deterministic PDA (DPDA). Certain languages cannot be recognized with a DPDA. The set of languages that can be recognized with a DPDA is a proper subset of CFLs and it is called deterministic CFLs (DCFLs).

In a DPDA, at each step of computation, we have at most one possible transition. Defining DPDA is more complex than DFA because a DPDA may read an input symbol without changing the stack or vice versa. So, we need to allow ε-transitions that we didn't allow in DFA. We use two types of ε-transition:

1. ε-input transition: $\delta(q, \varepsilon, x)$; doesn't consume input but changes stack.
2. ε-stack transition: $\delta(q, a, \varepsilon)$; doesn't change stack but consumes input.

A transition with both ($\delta(q, \varepsilon, \varepsilon)$) is allowed only if there's no other transition in the same state that uses a symbol instead of an ε. The main idea with this type of determinism is that the machine doesn't have choice at any point and there's always one possible path for computation.

Definition: A DPDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, 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. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to (Q \times \Gamma_\varepsilon) \cup \{\emptyset\}$ is the transition function (where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$),
5. $q_0 \in Q$ is the start state, and

6. $F \subseteq Q$ is the set of accepting states.

In addition, the transition function δ must satisfy the following condition. For every $q \in Q$, $a \in \Sigma$, and $x \in \Gamma$, exactly one of the values $\delta(q, a, x)$, $\delta(q, \varepsilon, x)$, $\delta(q, a, \varepsilon)$, and $\delta(q, \varepsilon, \varepsilon)$ is not $\emptyset$.

Examples: Consider the PDA given in Figures 2 and 3. Try to decide if they obey the rules above. So, are they deterministic or not? The answer is that the first one, M_1 is deterministic but the second one in Figure 3 is not. Furthermore, it is not possible to design a deterministic PDA that recognizes the same language as the second one because the language that it recognizes $(\{a^i b^j c^k \mid i, j, k \in \mathbb{N} \text{ and } (i = j \text{ or } i = k)\})$ is a CFL but it is not a DCFL.

Example: Consider the language $\{ww^R \mid w \in \{0, 1\}^*\}$. Do you think it is possible to design a DPDA that recognize this language? Try to design one to get an idea. If you can't, think about what the problem is.