Push Down Automata and Deterministic Push Down Automata

A Word from the Instructor

I have noticed that the definition of a push-down automaton (PDA) differs from textbook to textbook. I checked four sources recently, and they all differ. However, all sources give equivalent definitions, that is, in each case the every context-free language is accepted by some PDA, and the language accepted by any PDA is context-free. I was seeking the "official" definition of a PDA, and I conclude that there is no consensus among experts. The definition I like best is on the Wikipedia page, which has a friendly informal explanation of a PDA, complete with explantory figures, as well as rigorous definitions of the terms. For that reason, my definition of PDA is essentially the same as the one on Wikipedia.

You should read the Wikipedia article `https://en.wikipedia.org/wiki/Pushdown_automaton` at least down to the end of the section called **PDA and context-free languages.** I have posted the first five pages of that article as the handout `wikipda.pdf`. (Do not read past page 5.) Every CFL is accepted by some PDA and the language accepted by any PDA is context-free. You also need to know that 2-PDA (2 stacks) have *Turing power*.

We will emphasize *deterministic* push-down automata (DPDA). They have their own Wikipedia article, `https://en.wikipedia.org/wiki/Deterministic_pushdown_automaton`. DPDA have practical importance. For example, an LALR parser is a DPDA with output.

Definition of a PDA

A PDA is an ordered 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, z, F)$ where

- Q is a finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- δ is the *transition relation*, sometimes called the *transition function*, which is a a finite subset of $\Big[Q \times \Sigma \cup \{\varepsilon\} \times \Gamma\Big] \times \Big[Q \times \Gamma^*\Big]$.
- $q_0 \in Q$ is the start state.
- $z \in \Gamma$ is the start stack symbol.
- $F \subseteq Q$ is the set of final states.

Instantaneous Descriptions. We define an *instantaneous description* (ID) of M to be an ordered triple (q, w, γ) where

- $q \in Q$, is the current state of M,
- $w \in \Sigma^*$ is the unread input string,
- γ is the current stack.

We write the input string from left to right and the stack from top to bottom. Thus, if $w = ab$, the first symbol of w is a, while if $\gamma = xyz$, the top symbol is x and the bottom symbol is z. The transition relation δ can be defined to be a finite set of ordered pairs. For clarity, we insert an arrow between the first and second member of a pair. Thus we write a member of δ as $(q, a, x) \mapsto (q', \gamma)$, for $q, q' \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in \Gamma$, and $\gamma \in \Gamma^*$. A *start* ID of M is a triple (q_0, w, z), where $w \in \Sigma^*$. The initial stack contains just the start stack symbol z.

The Wikipedia page gives two possibilities for an accepting ID, *empty stack* and *final state*. We will use the stricter rule, *empty stack and final state*. In any case, the entire input file must have been read. Thus an *accepting* ID is an ordered triple $(q, \varepsilon, \varepsilon)$, where $q \in F$.

Transitions and Computations. Each step of a computation of M changes one ID of M to another, guided by one member of δ. Suppose the current ID is $I = (q, aw, x\gamma)$, for $q \in Q$, $a \in \Sigma \cup \varepsilon$, $w \in \Sigma^*$, $x \in \Gamma$, and $\alpha \in \Gamma^*$, and the step is guided by $(q, a, x) \mapsto (q', \alpha)$. Then M pops the symbol x off the stack. Then either reads a symbol or not, then pushes α onto the stack. The ID after the step is $I' = (q', w, \alpha\gamma)$. We write $I \vdash_M I'$.

Language Accepted by M. We write $\vdash_M^*$ to be the reflective transitive closure of $\vdash_M$. We say M *accepts* $w \in \Sigma^*$ if $(q_0, w, z) \vdash_M^* (f, \varepsilon, \varepsilon)$ for some $f \in F$. Let $L(M)$ be the language consisting of all strings accepted by M.

Example 1. We define a PDA M which accepts $L = \{a^n b^n : n \geq 0\}$.

Let $M = \big(\{q_0, q_1, q_2\}, \{a, b\}, \{z, a\}, \delta, q_0, z, \{q_2\}\big)$ where δ consists of the following transitions.

1. $(q_0, \varepsilon, z) \mapsto (q_2, \varepsilon)$
2. $(q_0, a, z) \mapsto (q_0, az)$
3. $(q_0, a, a) \mapsto (q_0, aa)$
4. $(q_0, b, a) \mapsto (q_1, \varepsilon)$
5. $(q_1, b, a) \mapsto (q_1, \varepsilon)$
6. $(q_1, \varepsilon, z) \mapsto (q_2, \varepsilon)$

We now give a computation of M which accepts the string *aaabbb*.

$(q_0, aaabbb, z) \overset{2}{\vdash}_M (q_0, aabbb, az) \overset{3}{\vdash}_M (q_0, abbb, aaz) \overset{3}{\vdash}_M (q_0, bbb, aaaz) \overset{4}{\vdash}_M (q_1, bb, aaz) \overset{5}{\vdash}_M (q_1, b, az) \overset{5}{\vdash}_M (q_1, \varepsilon, z) \overset{6}{\vdash}_M (q_2, \varepsilon, \varepsilon)$

Example 2. Palindromes.

Let L be the language of palindromes over $\{a, b\}$. That is, $L = \{w \in \{a, b\}^* : w^R = w\}$.

δ consists of the following transitions.

1. $(q_0, a, z) \mapsto (q_0, az)$
2. $(q_0, b, z) \mapsto (q_0, bz)$
3. $(q_0, a, a) \mapsto (q_0, aa)$
4. $(q_0, b, a) \mapsto (q_0, ba)$
5. $(q_0, a, b) \mapsto (q_0, ab)$
6. $(q_0, b, b) \mapsto (q_0, bb)$
7. $(q_0, \varepsilon, z) \mapsto (q_1, z)$

8. $(q_0, a, z) \mapsto (q_1, z)$
9. $(q_0, b, z) \mapsto (q_1, z)$
10. $(q_0, \varepsilon, a) \mapsto (q_1, a)$
11. $(q_0, a, a) \mapsto (q_1, a)$
12. $(q_0, b, a) \mapsto (q_1, a)$
13. $(q_0, \varepsilon, b) \mapsto (q_1, b)$
14. $(q_0, a, b) \mapsto (q_1, b)$
15. $(q_0, b, b) \mapsto (q_1, b)$
16. $(q_1, a, a) \mapsto (q_1, \varepsilon)$
17. $(q_1, b, b) \mapsto (q_1, \varepsilon)$
18. $(q_1, \varepsilon, z) \mapsto (q_2, \varepsilon)$

Deterministic Push-Down Automata

In practice, it is impossible to write a program to emulate a machine that is not deterministic. Here, we focus our attention on deterministic push-down automata, DPDA. Not every CFL is accepted by a DPDA. If L is accepted by a DPDA, we say it is a *deterministic* context-free language, or DCFL.

A PDA is may not query either the stack or the input string to determine whether it is empty. Since a PDA is non-deterministic, it can always correctly guess whether the stack or the input is empty. This solution is not availabel to a DPDA.

We solve the problem for the stack by insisting that there is always a special symbol at the bottom of the stack, the start stack symbol, up until the last step, when it is popped off. Similarly, when we design a DPDA, we can insist that every input string ends of an "end-of-file" marker, sometimes abbreviated **eof**; we use the dollar sign "**$**" for end-of-file.

Definition of a Deterministic Push-Down Automaton

A DPDA is defined to be a PDA which has two restrictive properties:
1. The δ is single valued. That is, for any $(q, a, x) \in Q \times \Sigma \cup \{\varepsilon\} \times \Gamma)$ there is a most one value of $\delta(q, a, x) \in Q \times \Gamma*$.
2. If $\delta(q, \varepsilon, x)$ is defined for some $q \in Q$, $x \in \Gamma$, then $\delta(q, a, x)$ is not defined for any $a \in \Sigma$.

Observe that the PDA in Example 2 above is not a DPDA, since, for example, $\delta(q_0, a, a)$ is simultaneously (q_0, aa) and (q_2, a).

The justification for the second property is less obvious. A DPDA computes pop= x first, then uses x to decide what to do next. If $\delta(q, \varepsilon, x)$ and $\delta(q, a, x)$ are both defined for some $a \in \Sigma$, M does not know whether to read the next input symbol. A PDA, on the other hand, correctly guesses what to do.

Finite Lookahead

We say that a DPDA has *k-lookahead* if it is able to look at the next k input symbols without reading them. When we study LALR parsing, we will assume that the parser has 1-lookahead.