Takeaway Notes: Push-Down Automata

Contents

1 Introduction

This writeup is meant to provide key takeaways regarding push-down machines or automata. It is largely a distillation of the textbook and online video material, but seen through my individual lens in terms of emphasis and approach. Think of the textbook, the videos, this writeup, and the class meetings as if they were four portholes into the same subject. By exposing yourself to the same material in four different ways, the hope is that you walk away from this in a way that fulfills the objectives of the course.

2 Formal Definition

Informally, a *push-down automaton* (PDA) is a finite-state automaton with a stack. However, those three words "with a stack" are quite loaded, and as always we turn to a formal definition to keep things clear and unambiguous.

A given PDA P can be written as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

As before, the key is in the definitions of these symbols:

- Q is a finite set of states
- Σ is an alphabet—the alphabet of the language that is accepted by P
- Γ is the *stack alphabet*—this part is new
- δ is P's *transition function* (more on this shortly)
- $q_0 \in Q$ is P's start state
- $Z_0 \in \Gamma$ is P's stack's *start symbol*
- $F \subseteq Q$ is P's set of accepting or final states

Of these symbols, Q, Σ, and Z_0 are analogous to their FSA equivalents. δ is analogous as well but is quite distinct in definition: δ is of the form $\delta(q, a, X) = \{(p, \gamma), \ldots\}$.

- $q \in Q$ is the current state
- Either $a \in \Sigma$ or $a = \epsilon$ is the next symbol on the input string
- $X \in \Gamma$ is the current symbol at the top of the stack
- $p \in Q$ is the destination state of the transition
- $\gamma \in \Gamma^*$ is the string that replaces X in the stack, upon transition

Note that, formally, this machine does not define a "push" or "pop" as we know it. Instead, a "push" is represented by $\gamma = \alpha X$ ($\alpha \in \Gamma$ of course) and a "pop" is represented by $\gamma = \epsilon$. "Leave the stack alone" is represented by $\gamma = X$.

2.1 The Stack has a Separate Alphabet

The provision that the stack has a distinct alphabet Γ invites some interesting possibilities. Note that there is no restriction at all between Σ and Γ: they can be identical, totally different, overlapping, different sizes, etc. Analogies can be drawn here with, say, the separation of V and T in a grammar, or possibly having a "parallel" set of states. A contrast can also be made between this and real-world stacks, such as those in a microprocessor. Those stacks accumulate CPU words, exactly like main memory. This situation appears to enforce the condition of having $\Sigma = \Gamma$, if we consider these alphabets to consist of every possible value that a CPU word can take. In any case, formally speaking, Γ is distinct, and this is important to remember.

2.2 The Machine is Nondeterministic

The presence of ϵ as a possible value for a in $\delta(q, a, X)$ as well as the absence of any stipulation on δ's coverage (i.e., there is no restriction that a result $\{(p, \gamma) \ldots\}$ *must* be defined for every possible combination of $\delta(q, a, X)$) shows that this PDA is *nondeterministic*. Further, there is no restriction that $\delta(q, a, X)$ have a single (p, γ) (it is defined as returning a set, after all)—yet another characteristic of a nondeterministic machine. As such, we apply the same rules for nondeterminism as we did with FSAs:

- When multiple possibilities apply to the same situation, we imagine that the PDA is running through all of them in parallel.
- When we encounter a combination of state, input symbol, and top-of-stack symbol for which there is no $\delta(q, a, X) = \{(p, \gamma) \ldots\}$ defined, then we may assume that the machine "dies" or "crashes" and thus does not accept the input string in question.

Interestingly, unlike with FSAs, it turns out that a deterministic PDA is *materially different* from a nondeterministic one: i.e., the sets of languages they can accept are actually different. Nondeterministic PDAs can accept any context-free language; deterministic ones can only accept a *subset* of CFLs.

2.3 A Sample PDA

For example, a PDA that accepts strings of the form ww^R (i.e., the language of "even" palindromes) would formally be defined as follows:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

…where $Q = \{A, B, C\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, Z_0\}$, $q_0 = A$, and $F = \{C\}$. δ would be defined as follows:

$$\delta(A, 0, Z_0) = \{(A, 0Z_0)\}$$
$$\delta(A, 1, Z_0) = \{(A, 1Z_0)\}$$
$$\delta(A, 0, 0) = \{(A, 00)\}$$
$$\delta(A, 0, 1) = \{(A, 01)\}$$
$$\delta(A, 1, 0) = \{(A, 10)\}$$
$$\delta(A, 1, 1) = \{(A, 11)\}$$
$$\delta(A, \epsilon, Z_0) = \{(B, Z_0)\}$$
$$\delta(A, \epsilon, 0) = \{(B, 0)\}$$
$$\delta(A, \epsilon, 1) = \{(B, 1)\}$$
$$\delta(B, 0, 0) = \{(B, \epsilon)\}$$
$$\delta(B, 1, 1) = \{(B, \epsilon)\}$$
$$\delta(B, \epsilon, Z_0) = \{(C, Z_0)\}$$

Graphical notations exist for this, and they typically resemble the finite state notation but with an additional stack read/write for each state transition.

3 Related Ideas

3.1 Instantaneous Descriptions (IDs)

The current status of a PDA during execution is called an *instantaneous description* (ID), consisting of a triple (q, w, γ) where q is the current state, w is the remaining input, and γ is the current content of the stack.

Instead of speaking of the δ transitions that are applied as a PDA executes (the way we describe FSA execution), we instead think of PDAs as going from one ID to another:

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

(the $\vdash$ symbol is colloquially called a "turnstile")

To represent multiple moves from starting to ending IDs, we add a $*$ to the turnstile. When it is necessary to specify the machine involved, say P, we add that as a subscript:

$$(q, w, X) \vdash_P^* (p, v, Y)$$

The notion of IDs allows us to express a key intuition about PDAs formally, which is that *data that a PDA never looks at cannot affect its computation.* Kind of like "out of sight out of mind." Formally, this is stated as such:

- If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA and $(q, x, \alpha) \vdash_P^* (p, y, \beta)$, then for any strings $w \in \Sigma^*$ and $\gamma \in \Gamma^*$:
$$(q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma)$$
- If $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is a PDA and $(q, xw, \alpha) \vdash_P^* (p, yw, \beta)$, then:
$$(q, x, \alpha) \vdash_P^* (p, y, \beta)$$

Note how the second statement is like a partial converse of the first one. Unused input can be added or removed harmlessly, but the same cannot be said of a trailing shared stack substring (can you see why?).

3.2 Acceptance by Final State

One way to define the language of a PDA is to require that its strings end in a final/accepting state, just like with FSAs:

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \alpha)\}$$

This condition for accepting a language puts FSAs strictly within the capabilities of a PDA (and thus regular languages strictly within the set of context-free languages), because in this sense, an FSA is simply a PDA that accepts a language by final state *and* ignores its stack. Intuitively, one can see how the languages accepted by that type of machine must strictly be contained within the set of all context-free languages. It just so happens that this set of languages is precisely the regular set.

3.3 Acceptance by Empty Stack

The stack feature of PDAs facilitates a new condition for acceptance: acceptance by empty stack. This effectively defines a new, alternate language that a given PDA P may accept (N here stands for "null stack"):

$$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$$

For the exact same P, $L(P)$ and $N(P)$ are not necessarily the same. However, the set of all languages acceptable by final state and the set of all languages acceptable by empty stack *are* the same, because there is a reliable, algorithmic way to convert a machine P into P' such that $L(P) = N(P')$ or $N(P) = L(P')$.

3.4 Equivalence of PDAs and CFGs

As we have seen with the rest of the course, the key idea here is equivalency to a class of languages. And indeed the class of languages that PDAs recognize is precisely the class of languages that context-free grammars (CFGs) recognize, namely all context-free languages.

Exact details are elided here, but not surprisingly, the proof of equivalency lies in the ability to algorithmically convert any PDA to a corresponding CFG and vice versa. The conversion algorithm from PDAs to CFGs is particularly interesting, making clever use of symbols to represent corresponding productions in the target CFG.

3.5 Deterministic Pushdown Automata (DPDA)

One last key concept in the area of PDAs is the notion of the *deterministic* pushdown automaton, or DPDA. A DPDA's determinism is reflected in these conditions:

- $\delta(q, a, X)$ has at most a single result for any $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, and $X \in \Gamma$.
- If $\delta(q, a, X)$ is not empty for some $a \in \Sigma$, then $\delta(q, \epsilon, X)$ must be empty.

The key characteristic of DPDAs is the language class that they recognize: the class consists of CFLs with unambiguous grammars. They can't recognize *all* CFLs with unambiguous grammars (e.g., $S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$) though. The relationship is solely an *if* and not an *if and only if*: if P is a deterministic push-down automaton, then the language it recognizes has an unambiguous grammar. However, if a language has an unambiguous grammar, it isn't necessarily true that a DPDA P exists which will recognize it.

4 Important Skills

What should one be able to *do* with this thought framework? The key skills are all about equivalencies, and equivalencies are shown by the ability to perform generalized conversions from one category to another:

- Converting a PDA from acceptance by final state to acceptance by empty stack and vice versa: this shows that both types of PDAs recognize the exact same class of languages.

- Converting a PDA into a CFG and vice versa: this again shows that both models for representing languages pertain to the exact same class of languages—in this case, context-free.

5 Big Picture Points

Things you shouldn't forget even years after taking this course:

- The capability that finally brings finite-state automatons to a higher level of power is the addition of a stack. This addition creates the family of machines called *push-down automata* (PDAs).
- PDAs accept a language either by final state (like FSAs) or by empty stack. For PDAs in general, both forms of acceptance stay within the same class of languages.
- This class of languages turns out to be the context-free languages—the exact same class generated by context-free grammars. This can be proven because a PDA can be converted into a CFG and vice versa.
- PDAs are nondeterministic by default; when they are deterministic, they accept a proper *subset* of CFLs. This proper subset shares the property that these languages have unambiguous grammars. However, not all languages with unambiguous grammars are part of this subset.