Chapter 5

Pushdown Automata

A pushdown automaton (PDA) is the first type of representation for a context-free language (CFL) that we will examine.

In this chapter we will construct a pushdown automaton (PDA), simulate input on that automaton, discuss strategy in constructing nondeterministic pushdown automata (NPDA), and then give JFLAP's formal definition of a PDA.

5.1 A Simple Pushdown Automaton

Constructing a PDA is similar to constructing an FSA, as you did in Chapter 1; the main difference is in the transitions. A PDA has additional memory in the form of a stack. Symbols can be pushed onto and off of the stack. A transition must now encode two inputs (what must be read from input and the top of the stack before you can take this transition) and one output (what must be pushed onto the stack if you decide to take this transition).

A transition for a PDA is defined in JFLAP as $X, Y; Z$ where $X$ represents the input symbols to be processed, $Y$ represents the stack symbols that must be on the stack and are popped off the stack, and $Z$ represents the stack symbols that are pushed onto the stack. Note that the comma separates the two inputs (what must be matched if this transition is to be taken) and the semicolon separates the inputs from the output.

For example, the transition $a, ab; cd$ from state $q_1$ to $q_2$ would be represented in JFLAP as shown in Figure 5.1. This transition is interpreted as "if $a$ is the next symbol in the input and $ab$ (with $a$ on top of $b$) is on the top of the stack, then process the input symbol $a$, pop $ab$ off the stack, and push $cd$ (first push $d$, then $c$) onto the stack."
5.1.1 Building the NPDA

Let's now create the NPDA shown in Figure 5.2. Start JFLAP and click on the button labeled Pushdown Automaton. The Editor tab that appears looks identical to the editor for the FSA. There will be only a few differences.

Start by creating four states using the State Creator tool Θ. Add transitions by selecting the Transition Creator tool → and add a transition from q₀ to q₁ (click on q₀ and drag to q₁). When you release the mouse note that three adjacent text fields appear. For the DFA there was only one text field. In the PDA there are three inputs that must be entered: input symbol(s), stack symbol(s) to pop, and stack symbol(s) to push, as shown in Figure 5.3. We want to enter the transition a, Z; aZ. Type "a" and press Tab. Note that the a is entered and the next field is ready for input. Type "Z" (note that this is the capital letter Z), press Tab, type "aZ", and press Enter. Note that the comma and semicolon are automatically added to the transition.

Add the remaining five transitions (leave a field blank for λ); make q₀ the initial state and q₃ the final state.

5.1.2 Simulation of Input

What is the language represented by this PDA? Let's step through a simulation of an input string. Select Input : Step with Closure and enter the input string aaabbbbbb. The initial configuration is shown in Figure 5.4 and includes the current state q₀, the input string, and the stack (shown below the input string). The stack is always "empty" when a trace starts. The stack actually has
a bottom-of-stack marker $Z$ (the capital letter $Z$) so there is a symbol to test to see if the stack is empty or not. Be careful! The $Z$ can be removed, but if you do, you cannot check to see if the stack is empty.

Click **Step** and the first $a$ in the input string has been processed (it is now shaded gray), the $Z$ was popped off the stack, but then put back on with an $a$ on top of it. The transition from $q_0$ to $q_1$ could also have been written as $a; \lambda; a$. (See Section 5.3 for different definitions of PDAs.)

Click **Step** two more times and there will be a total of three $a$’s on top of the stack; one for each $a$ in the input string. Click **Step** three more times, and each $a$ is matched with a $b$. For each $b$ greyed-out an $a$ is popped off the stack.

There are still two more $b$’s to be processed. Click **Step** two more times and the $b$’s are processed and the string is accepted.

What is the language accepted by this PDA? Try testing additional strings if you don’t see it right away. For example, does this PDA accept $bbaaa$, $aabb$, $aaabb$, $ababb$, and $bbb$? Don’t forget that you can test many input strings at the same time by selecting the multiple simulation input method.

The language accepted by this PDA is those strings with $a$’s first followed by $b$’s such that there is at least one $a$ and there are more $b$’s than $a$’s. Another way to write this is $\{a^nb^m \mid n > 0 \text{ and } n < m\}$. This example is also stored in the file ex5.1.

### 5.2 Nondeterministic PDA

A PDA is nondeterministic if there is a state with more than one choice of transition for its move. Remember that an FSA is nondeterministic if there is a state with either a $\lambda$-transition or two outgoing transitions from the same state with the same symbol. For an NPDA to be nondeterministic, there is a state with either a $\lambda$-transition $(\lambda, \lambda; \lambda)$ or two outgoing transitions from the same state with the same symbols for the input symbol and the symbols to pop. For example, the two transitions $a, b; \lambda$ and $a, a; \lambda$ outgoing from the same state do not make a PDA nondeterministic since the symbol to pop from the stack is different. The two transitions $a, ba; b$ and $a, ba; \lambda$ outgoing from the same state do make a PDA nondeterministic.
We will build in stages an NPDA using JFLAP that accepts the language \( \{ a^n b^m \mid n > 0, m \leq 3n \} \). First think about how you would build a deterministic PDA for this language. Is it possible? For each \( a \) processed, you must know how many \( b \)'s (1, 2, or 3) will be processed. But you do not know the number of \( b \)'s per \( a \) unless you can examine the whole string. A PDA can examine and process only one input symbol at a time. Thus, no deterministic PDA can be built that accepts this language. We must build an NPDA. In this NPDA there will be choices of paths that can be taken, such that for a valid input string, one of those paths will lead to acceptance.

Let's now build the NPDA. First let's discuss the possible strings in the language. They include \( aabb, aabb, aabbb, aabbbb \), and \( aabbbb \). All the \( a \)'s come first and for each \( a \) there must be one to three \( b \)'s. It is also a good idea to discuss strings not in the language. They include \( bba (a \)'s not first), \( bab (a \)'s not first), \( aabbb (too many b \)'s), \( aa (there must be at least one b) \), and \( bb (there must be at least one a) \).

Since all the \( a \)'s come first, create states that push all the \( a \)'s onto the stack. This NPDA is shown in Figure 5.5.

How do we continue so that for each \( a \) there will be one to three \( b \)'s? After all the \( a \)'s are pushed on we must have three paths that pop one \( a \) and end up at the same point: one path for reading one \( b \), one path for reading two \( b \)'s, and one path for reading three \( b \)'s. These three paths are added in Figure 5.6, with all three paths ending at \( q_2 \). Note in each path that only one \( a \) is popped from the stack.

At the point where the three paths end, we need to have three loops: one for matching one \( a \) with one \( b \), one for one \( a \) with two \( b \)'s, and one with one \( a \) with three \( b \)'s. These three loops are added in Figure 5.7. Note that only two transitions were added, as one transition can be used for the loop for two \( b \)’s and three \( b \)’s.

When the stack is empty, the string is accepted if all the input has been processed. Add one final state and arc as shown in Figure 5.8. This example is also stored in file ex5.2.

Which states are nondeterministic? In JFLAP select Test : Highlight Nondeterminism to show which states are nondeterministic.

There are several path choices in this NPDA. Let's simulate a string on the NPDA. Select Input : Step with Closure and enter the input string \( a^4 b^{10} \).
5.2. NONDETERMINISTIC PDA

Figure 5.6: Pop one $a$ and process 1, 2, or 3 $b$'s.

Figure 5.7: Three loops added.

Figure 5.8: A nondeterministic PDA.

With nondeterminism, the number of configurations can grow at a rapid rate. Click on Step nine times and see all the configurations! It is helpful in tracing NPDAs to remove configurations
you think are unlikely to succeed, and freeze selected configurations so you can focus on a smaller number of configurations.

Let us restart the trace by clicking on Reset. Click on Step four times and all the a’s are pushed onto the stack. Click on Step two more times and there are now four possible configurations, as shown in Figure 5.9.

Can any of these configurations no longer possibly succeed? The rightmost configuration has already pushed two a’s off the stack and there are eight b’s left, so this configuration will eventually lead to failure since there are more than three b’s for each remaining a. Although JFLAP will automatically remove this configuration later when it fails, it can be removed now by selecting it first and then selecting Remove. This will reduce the number of configurations later to examine as this configuration would have generated several more configurations, all that would eventually fail. Continue stepping through, trying to keep the number of choices down by removing or freezing some configurations. There are several possible solutions, as there are several combinations of matching two of the a’s with three b’s each and two of the a’s with two b’s each.

## 5.3 Definition of an NPDA in JFLAP

JFLAP gives a general definition of an NPDA $M$ to be defined as a septuple $M = (Q, \Sigma, \Gamma, \delta, q_s, Z, F)$ where:

- $Q$ is a finite set of states $\{q_i|i \text{ is a nonnegative integer}\}$
- $\Sigma$ is the finite input alphabet
- $\Gamma$ is the finite stack alphabet
- $\delta$ is the transition function, $\delta : Q \times \Sigma^\ast \times \Gamma^\ast \rightarrow \text{finite subsets of } Q \times \Gamma^\ast$
- $q_s \in Q$ is the initial state
- $Z$ is the start stack symbol (must be capital Z)
- $F \subseteq Q$ is the set of final states

The definition of $\delta$ in JFLAP is general to allow JFLAP to work with many definitions of an NPDA. Zero or more symbols can be read as input, zero or more symbols can be popped from the stack, and zero or more symbols can be pushed onto the stack. In defining an NPDA you do not need to define the input symbols and stack symbols explicitly, but they are determined based on the labels you enter in the transactions. The bottom-of-stack marker is set in JFLAP to $Z$ and
cannot be changed. JFLAP assumes this marker is the only symbol on the stack when a simulation begins.

You might want to restrict the definition of $\delta$ for a particular NPDA to a subset of what JFLAP allows. For example, consider the following restrictions: Require the input symbol to be either a single symbol or $\lambda$ and require exactly one symbol to be popped from the stack. In that case, the definition of $\delta$ would be:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \text{finite subsets of } Q \times \Gamma^*$$

If you want to use this definition of $\delta$, then when you build your NPDA with JFLAP make sure all the transitions follow this restricted rule. JFLAP does not check this for you.

Here is an example showing the nonpictorial representation of the NPDA $M$ in Figure 5.2. Using the formal definition of an NPDA in JFLAP, $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ where:

$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, Z\}$

$\delta = \{(q_0, a, Z, q_1, aZ), (q_1, a, a, q_1, a\alpha), (q_1, b, a, q_2, \lambda), (q_2, b, a, q_2, \lambda), (q_2, b, Z, q_3, Z), (q_3, b, Z, q_3, Z)\}$

$q_0 = q_0$

$F = \{q_3\}$

The $\delta$ moves are described as five-tuples where the first field is the state the transition originates in, the second field is the input symbol(s) to process, the third field is the stack symbol(s) to pop, the fourth field is the state at which the transition ends, and the fifth field is the stack symbol(s) to push onto the stack.

5.4 Summary

In Section 5.1 we learned how to edit and simulate an NPDA. The main difference between editing an NPDA and an FSA is the label on a transition. The transition format is $X, Y; Z$ where $X$ represents the input symbols to be processed, $Y$ represents the stack symbols that must be on the stack and are popped off the stack, and $Z$ represents the stack symbols that are pushed onto the stack. The configuration of an NPDA includes the current state, the current input left to be processed, and the current stack contents. A special symbol, capital $Z$, is the bottom-of-stack marker.
In Section 5.2 we built an NPDA for the language \( \{a^n b^m \mid n > 0, m \leq 3n\} \), a language that cannot be recognized by a deterministic PDA. We then traced it with the string \( a^{4}b^{10} \) and saw how quickly the nondeterminism led to an exponential number of configurations.

In Section 5.3 we presented JFLAP's formal definition of an NPDA. JFLAP's definition is general so that it accommodates many definitions of a PDA. For example, you may prefer to use the definition in which a single symbol is always popped from the stack. (Popping multiple symbols or \( \lambda \) is not allowed.)

### 5.5 Exercises

1. Load the following files containing NPDA. For each NPDA, list five strings that are accepted, list five strings that are not accepted, and determine the language represented.

   (a) ex5.5a
   (b) ex5.5b
   (c) ex5.5c
   (d) ex5.5d
   (e) ex5.5e
   (f) ex5.5f

2. Modify the NPDA from Section 5.1 to accept the following languages.

   (a) \( \{a^n b^m \mid n > 0 \text{ and } n = m\} \)
   (b) \( \{a^n b^m \mid n > 0 \text{ and } m = 2n\} \)
   (c) \( \{a^n b^m \mid n > 0 \text{ and } n > m\} \)

3. Generate a set of strings for testing the NPDA from Section 5.2 that will convince you this NPDA is correct. You should create a minimal set of strings that covers all cases of strings that are accepted and not accepted.

4. The following NPDA are incorrect. Load the files and determine why they do not represent the corresponding language. List strings they should accept and do not, or strings they should not accept but do. Then correct the NDPA.

   (a) ex5.5Wrong-a should represent the language \( \{a^n b^m c^n \mid n < m + p, n > 0, m > 0, p > 0\} \).
   (b) ex5.5Wrong-b should represent the language \( \{a b^n c^n \mid n \geq 0, m \geq 0\} \).
   (c) ex5.5Wrong-c should represent the language \( \{(a)^n b^m c^n \mid n \text{ is odd, } n > 0, m > 0\} \).