# CSE 322 - Introduction to Formal Methods in Computer Science
# Pushdown Automata

Dave Bacon
*Department of Computer Science & Engineering, University of Washington*

Okay now that we've talked a little about context free grammars, a natural question to ask is: is there some sort of machine which recognizes exactly the context free languages? Well if we relplace "machine" by computational model, then the answer is yes: nondeterministic pushdown automata recognize exactly the context free languages.

So what is a nondeterministic pushdown automata? Well a nondeterministic pushdown automata is a computational model a lot like our good friend the nondeterministic finite automata, but with a crucial difference: a pushdown automata has a memory. Now this isn't just any sort of memory: a nondeterministic finite automata has a *stack* memory. Many of you probably know what a *stack* is. Well at least all of you have stacked up books on top of each other. And when you stack books on top of each other, if you don't allow yourself the Houdini act of pulling out a book from the middle of stack, your stack of books is a last in, first out procedure. That is the last book that you put on the top of the stack is the first that you'll get back if you take a book off the top of the stack. We like to be fancy and call this *LIFO*: last in, first out (as opposed to *FIFO*: first in, first out.)

Okay, so back to pushdown automata. A nondeterministic pushdown automata is like a nondeterministic finite automata, except that it has a stack around. Now the automata can do things like add new symbols to the top of the stack, read symbols off the top of the stack, and remove the top element of the stack. This allows a nondeterministic pushdown automata (NPDA) to have a *memory* of what it has read before and therefore, we will see, recognize languages which a DFAs cannot recognize.

Lets give an informal example. Suppose that we wish to design a NPDA which recognizes the non-regular language

$$L = \{ww^R | w \in \{0,1\}^*\}.$$

This language is not regular because, informally, it needs to remember what it has read in order to verify that the remaining half symbols are the reversal of the first half. Now lets describe informally how a NPDA could recognize such a language. The machine begins by pushing symbols that it reads onto the stack. After pushing each symbol onto the stack, nondeterministically guess (i.e. think $\varepsilon$ transition) that the middle of the string has been reached. At each such guess, read the remaining symbols, at each step popping off the top element of the stack and comparing it with the symbol read. If you get to the end of the string and the stack is empty, and at each step the symbols match, then you should accept.

We see that the reason the NPDA could recognize the language $L$ was that it could use the stack to store the partially read string. While a finite automata has a *memory* which is related to the number of states in the automata, the pushdown automata has a stack with which it can store any amount of data. Now of course, this memory is in some way restricted: it is LIFO afterall, so this isn't necessarily a description of how a *full* computer (whatever that means) works, but, none the less, represents and increase in ability to recognizes languages over finite automata.

## I. FORMAL DEFINITION

Okay lets get formal (words I'm sure you dread hearing!) A nondeterministic pushdown automaton (NPDA) is going to be described by our largest tuple of all time, a six tuple! Formally a nondeterministic pushdown automaton is described by the six tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$. These objects are

1. $Q$ is a finite set representing the states of the NPDA.
2. $\Sigma$ is a finite set which is the input alphabet.
3. $\Gamma$ is a finite set which is the stack alphabet.
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transition function.
5. $q_0 \in Q$ is the start state.
6. $F \subseteq Q$ is the set of accept states.

Okay, so all of these objects should be transparent in what they are, except the transition function $\delta$. So lets discuss $\delta$. First of all lets talk about the input to the transition function. This is $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$. Recall that $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\Gamma_\varepsilon = \Sigma \cup \{\varepsilon\}$. Okay, so first of all lets talk about the function when it neither of the last two inputs are $\varepsilon$. In this case the transition function takes as input the current state, the next symbol in the input string, and the top element of the stack. It then outputs a set of possible new states and new symbols for the top of the stack. Note the the new top of the stack replaces this the old top of the stack. Now what happens if one of the inputs is $\varepsilon$. If the $\varepsilon$ comes from the input alphabet component of the function, this means that the function does not read a symbol from the input. Similarly if the $\varepsilon$ comes from the stack alphabet part of the function, this means that the function does not read from the top of the stack.

Okay, so given the formal definition of a NPDA, lets give a formal definition for the computation. We say that a pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts the input $w$ if $w$ can be written as $w_1 w_2 \dots w_m$ where $w_i \in \Sigma_\varepsilon$, and a sequence of states $r_0, r_1, \dots, r_m \in Q$ exists along with a sequence of strings $s_0, s_1, \dots, s_m \in \Gamma^*$ exists such that

1. $r_0 = q_0$ and $s_0 = \varepsilon$. This is the requirement that the start state is correct and that the stack starts as the empty string.
2. For $i \leq 0 \leq m-1$, 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 $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$. This is the requirement that the evolution is correct according to the transition function. Note that $s_i$ represents the stack at step $i$ of the computation.
3. $r_m \in F$. This is the condition that the final state is an accept state.

## II. EXAMPLE

Lets give an example of a NPDA which recognizes our old friend the non-regular language

$$w = \{0^n 1^n | n \geq 0\}$$

First lets give an informal description of how a pushdown automata would recognize this language. Informally the idea is that as we read of the symbols from the input, we push them onto the stack. After each such push we nondeterministically guess that we have reached the end of the 0s and then begin to pop off the 0s everytime we read a 1. If we make it back to a state where the stack is empty while always reading 1s and popping off 0s, and there are no more symbols to read we accept.

One issue in constructing an NPDA which accepts this language is that out definition of a NPDA doesn't allow us to know that we have the empty stack. To get around this, we will use an extra symbol in the stack alphabet, call it \$, which allows us to keep track of the bottom of the stack.

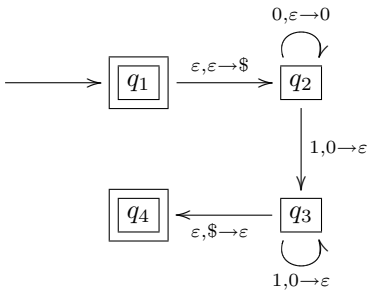
Okay so now lets formally define the NPDA, $(Q, \Sigma, \Gamma, \delta, q_0, F)$:

1. $Q = \{q_1, q_2, q_3, q_4\}$.
2. $\Sigma = \{0, 1\}$.
3. $\Gamma = \{0, 1\}$. We will need two stack symbols, one to keep track of the bottom of the stack and the other to keep track of having read zeros.
4. The transition function is given by the following table:

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

5. $q_0 = q_1$.
6. $F = \{q_1, q_4\}$.

We can also specify this NPDA via something resembling a state diagram. But now instead of each arrow being labeled by a symbol from the input alphabet, each transition is labeled by as $x, y \to z$ symbolizing that when the machines is reading an $x$ from the input it may replace the symbol $y$ on the top of the stack with $z$. Here is that state diagram:




It is helpful to consider how this NPDA acts on some input strings. First consider how this machine acts on 000111. The machine begins in state $q_1$ with the state empty, $\varepsilon$. Then it makes a $\varepsilon$ transition and ends up with this state and symbol $(q_1, \varepsilon)$ and the state and stack $(q_1, \$)$. Now the machine reads 0. The $q_1$ state then dies out, but the $q_2$ state, adds 0 to the stack and thus transitions to $(q_2, 0\$)$. The machine then reads the next 0. This adds another 0 to the stack and the state remains in $q_2$: $(q_2, 00\$)$. Next the machine reads the third 0 and transitions to $(q_2, 000\$)$. Now the machines reads the first 1. This will transition the machine to $q_3$ because the top of the stack is 0, and remove the 0 from the top of the state. This produces the state $(q_3, 00\$)$. Reading the next 1 will pop off another 0 resulting in $(q_3, 0\$)$. The final 1 produces transitions to the state $(q_3, \$)$. But now there is an allowed $\varepsilon$ transition because the symbol on the top of the stack is \$. This transitions to $(q_4, \varepsilon)$. Since we are at the end of the computation and we have a state alive which is an accept state, the machine will accept this string.