1 Informal description

Push down automata are very similar to finite automata, but we equip the state machine with a stack for reading and writing data. The automata still operates by scanning the input, left to right, one character at a time. The automata terminates when it has read the last character of the input. An example can be seen below: transitions are labeled by $a, b/c$, where $a \in \Sigma$ is a value of the input, and $b, c \in \Gamma$, where $\Gamma$, called the “tape alphabet”, is the set of characters that can pushed and popped from the stack. It is reasonable to assume that $\Sigma \subseteq \Gamma$. The notation $b/c$ means that you can take this transition if the character at the top of the stack is $b$, and, in doing so, you replace the $b$ with a $c$. Note that you can only take a transition $a, b/c$ if the next character of the input is a AND the character at the top of the stack is a $b$. If we don’t wish to put anything new onto the stack, we can use a transition of the form $a, b/\Lambda$. In this case, we would pop a $b$, and the number of elements on the stack would be reduced by one. Similarly, we also allow the machine to ignore the input or the stack. The former is denoted by $\Lambda, a/b$, and the latter is denoted by $a, \Lambda/c$. We can ignore both by $\Lambda, \Lambda/c$. Such transitions can be taken regardless of the values of the next input character and the character at the top of the stack. We also allow to push multiple characters onto the stack at once (though we do not allow multiple pops at once). For example, $a, b/bc$ would pop 1 $b$, push 1 $b$, and then push 1 $c$. If the stack initially only had content $b$, then, after this transition, it would contain $bc$; we will always write (left to right) the content of the stack from bottom to top.

Empty stack: We don’t have any explicit mechanism for testing the stack to see if it is empty. Instead, if that is something we care to do, we can create a transition at the start that pushes a special symbol onto the stack, and we can later interpret as an indicator that the stack is empty. This can be seen in Figure 1 below, where $\$ plays that role. Note that we do not read an input character in that transition; we only start processing input after we’ve initialized our stack.

Termination: The automata terminates when the last character of the input is read. It accepts if and only if it terminates in an accept state. Just as with finite state automata, we assume there is a trap state for rejecting that is not made explicit: if it is ever impossible to make a transition, and there is still input that hasn’t been processed, then the machine is assumed to transition into a reject state and to stay there. We stress that we do not care about the content of the stack when deciding whether to terminate, and, in particular, we can terminate while there is content on the stack. (This is different from what is described in the course textbook.) Note that in the case where we ignore the input tape, we also delay termination by one transition. So, we can take many transitions of the form $\Lambda, a/b$, and these transitions do not “use up” any of the input. We will next walk through the example in Figure 1 below, which will demonstrate this point.

Example: We walk through the NPDA in Figure 1, using input $abba$. 

1-1
• There is only one transition we can take from the start state. We transition to even, the input tape still holds abba, and the stack holds $.

• Since the first input character is $a$, the only legal transition is to $>_a$. The input tape holds bba and the stack holds $a$.

• Since the top of the stack is now $a$, the only legal transition is $b, a/\Lambda$, which leaves us in the same state. (Note we cannot take the transition labeled $\Lambda, \$/\$, because the top of the stack was $a$.) This transition removes the $a$ at the top of the stack. The input tape now holds $ba$, and the stack now holds $\$. 

• The only legal transition is the one labeled $\Lambda, \$/\$, to state even. The remaining input is still $ba$, since the $\Lambda$ in that transition does not use up an input character. This transition pops and pushes $\$, so the stack still holds $\$.

• We transition to state $>_b$, the input tape holds $a$ and the stack holds $\$b$.

• We transition using $a, b/\Lambda$, remaining in the same state. The input tape is now empty, and the stack now holds $\$.

• We now have a choice to make. We can terminate and reject, or we can transition one more time using $\Lambda, \$/\$ and then accept. Recall that the definition of non-determinism says that a string is in the language as long as there exists some sequence of choices that leads to accept. So, in this case, the string is in the language.

![Diagram of the NFA accepting $L = \{0^n | n \geq 0\}$](image)

Figure 1: $M_L$ accepting language $L = \{0^n | n \geq 0\}$

2 Formal notation

A NPDA can be denoted by $(Q, \Sigma, \Gamma, \delta, q_0, Q_A)$, where $Q$ is the set of states, $\Sigma$ is the input alphabet, $\Gamma$ is the tape alphabet (which might contain $\Sigma$), $\delta$ is a transition function, detailed below, $q_0$ is a
special start state, and \( Q_A \subseteq Q \) is a set of accept states. The function \( \delta \) maps a state, an input character, and a character read from the stack, to a state and a sequence of characters to be written to the stack. However, in the non-deterministic case, note that it might map the same input onto multiple outputs. We therefore let the co-domain be the power set of \( Q \times \Gamma^* \). Formally, then, \( \delta \) is a function \( \delta : Q \times \Sigma \times \Gamma \rightarrow 2^{Q \times \Gamma^*} \).

A machine accepts string \( w \) if and only if \( w \) can be written as \( w_1 w_2 \cdots w_n \), where each \( w_i \in \Sigma \cup \{\Lambda\} \), and there exists a sequence of states \( r_0, r_1, \ldots, r_n, r_i \in Q \), and a sequence of strings \( s_0, \ldots, s_n, s_i \in \Gamma^* \), such that

1. \( r_0 = q_0, s_0 = \Lambda, \) and \( r_m \in Q_A \).
2. \( \forall i \in \{1, \ldots, n\}, \exists \alpha \in \Gamma \cup \{\Lambda\}, \beta, \gamma, \in \Gamma^* \), such that \( s_{i-1} = \alpha \gamma, s_i = \beta \gamma, \) and \( (r_i, \beta) \in \delta(r_{i-1}, w_{i-1}, \alpha) \).

Intuitively, \( w_1 \cdots w_n \) denote the input string, but possibly “padded” with internal \( \Lambda \) values to account for places that we might take a transition that doesn’t read any input. The first condition states that we start in the start state with an empty stack, and we terminate in an accept state. The second condition says that we transition through some valid sequence of states, maintaining valid stack content.