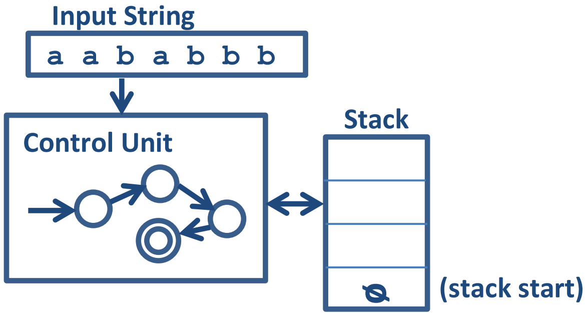
10 - Pushdown Automata

We have seen that *finite automata* are limited in that they are only capable of accepting *regular languages*. Such languages (and machines) are useful for lexical scanning, as we have seen. But for parsing, we found it useful to build a machine capable of recognizing a *context-free language*. Finite automata aren't adequate for this task because they have no "memory", beyond their states, which are finite in number.

<u>Pushdown Automata</u> (PDA) add a **stack** to the existing notion of a finite state machine, giving them an infinite (albeit simple) memory mechanism:



A Nondeterministic Pushdown Acceptor (NPDA) is defined by the septuple:

- Q – a finite set of states
- Σ – an input alphabet
- Γ – a stack alphabet
- δ – transitions Q x (Σ∪{λ}) x Γ → Q x Γ*
- s – initial state ϵ Q
- ⦰ – start stack symbol ϵ Γ
- F – set of final states ⊆ Q

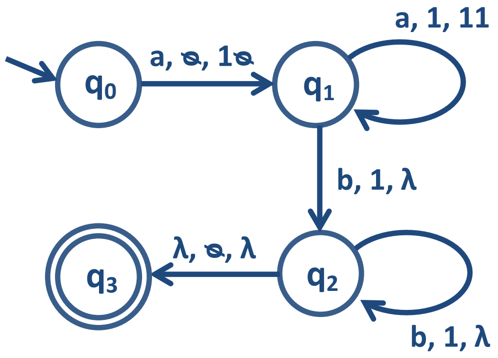
For example, transition δ (q,a,c) = (q',w) would mean:

1. the machine is currently in state q
2. consume (read in) the next symbol a from the input string
3. pop the symbol at the top of the stack
4. move to state q'
5. push a new string onto the top of the stack

Example 1 – Construct a NPDA for the language { a^nb^n ; n≥1 }

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a,b\}$
$\Gamma = \{\varnothing,1\}$
$z = \varnothing$
$s = q_0$
$F = \{q_3\}$

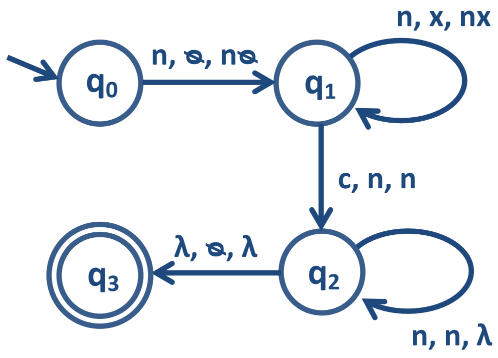
$\delta(q_0,a,\varnothing) = \{ (q_1,1\varnothing) \}$
$\delta(q_1,a,1) = \{ (q_1,11) \}$
$\delta(q_1,b,1) = \{ (q_2,\lambda) \}$
$\delta(q_2,b,1) = \{ (q_2,\lambda) \}$
$\delta(q_2,\lambda,\varnothing) = \{ (q_3,\lambda) \}$



Example 2 – Construct a NPDA for the language { wcw^R ; $w\epsilon\{a,b\}^+$ }

$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a,b,c\}$
$\Gamma = \{\varnothing,a,b,c\}$
$z = \varnothing$
$s = q_0$
$F = \{q_3\}$

$\delta(q_0,n,\varnothing) = \{ (q_1,n\varnothing) \}$ $n \in \{a,b\}$
$\delta(q_1,n,x) = \{ (q_1,nx) \}$ $n,x \in \{a,b\}$
$\delta(q_1,c,n) = \{ (q_2,n) \}$ $n \in \{a,b\}$
$\delta(q_2,n,n) = \{ (q_2,\lambda) \}$ $n \in \{a,b\}$
$\delta(q_2,\lambda,\varnothing) = \{ (q_3,\lambda) \}$



NPDAs accept all strings for which there is *some* path to an "accept" state.

The distinction between "deterministic" and "non-deterministic" PDAs is a bit different than it is for FAs. PDAs are only non-deterministic if there is more than one choice for a given scenario. Note that in a PDA, the input string isn't the only factor in a state transition; the symbol at the top of the stack also plays a role. By this definition, ex. 1 (above) is deterministic. Finally, unlike FAs, "deterministic" and "non-deterministic" PDAs aren't equivalent. NPDAs have more expressive power than DPDAs.

We will focus on NPDAs, because they are equivalent to CFGs.