1 Push-down Automata

A push-down automaton is a finite automaton with an additional last-in first-out push-down stack; anything read from the stack is immediately destroyed. Push-down automata are partway to a Turing machine. Push-down automaton are nondeterministic and can recognize context-free languages. They are important for parsing and compiling.

1.1 Formalism

Formally, a push-down automaton is a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$.

- All the components are similar to a nondeterministic finite automaton and have similar meanings, except for Γ which is the *stack alphabet*.
- Also, the set Δ of transitions is a little more complicated than in a fsa because it is necessary to consider what happens to the stack.

Thus we have the following in a push-down automaton $M = (K, \Sigma, \Gamma, \Delta, s, F)$:

K	a finite set of states
Σ	an input alphabet
Γ	a stack alphabet
$s \in K$	an initial state
$F \subseteq K$	a set of final states
Δ	a transition relation, a finite subset of $(K \times (\Sigma \cup \{\epsilon\}) \times \Gamma^*) \times (K \times \Gamma^*)$

1.2 Transitions

The meaning of a transition $((p, a, \beta), (q, \gamma)) \in \Delta$:

M in state p with β at the top of the stack can

- read an a from the input tape (and a may be ϵ),
- replace β on top of the stack by γ, and
- enter state q.

- Thus β is popped from the stack and γ is pushed in its place.

- The stack is written with topmost symbols to the left.

The triple (p, a, β) can be regarded as the *condition* of the transition and the pair (q, γ) can be regarded as the *action*.

See Handout 5, How Does a Push-Down Automaton Work?

Push-down automata are *nondeterministic*.

1.3 Configurations

- A *configuration* of a push-down automaton is a member of $K \times \Sigma^* \times \Gamma^*$.
- If (q, w, α) is a configuration, then
- q tells the current state of the pda,
- w tells the remaining input symbols to be read,
- and α gives the push-down stack, topmost symbols to the left.

1.4 Yields in one step

If (p, x, α) and (q, y, ζ) are configurations, then $(p, x, \alpha) \vdash_M (q, y, \zeta)$ (*yields in one step*) if there is a transition $((p, a, \beta), (q, \gamma))$ in Δ such that $x = ay$, $\alpha = \beta\nu$, and $\zeta = \gamma\nu$ for some $\nu \in \Gamma^*$. Note that the pda has no test for an empty stack; it can only test what's on top of the stack.

1.5 Yields

$\vdash_M^*$ is the transitive reflexive closure of $\vdash$, so that $C_1 \vdash_M^* C_2$ if configuration C_2 can be obtained from C_1 by zero or more "yields in one step" operations. $\vdash_M^*$ is read as *yields*.

1.6 Language recognized by a pda

A push-down automaton M *accepts* an input $w \in \Sigma^*$ iff $(s, w, \epsilon) \vdash_M^* (p, e, e)$ for some state $p \in F$. Thus at some time, the stack must be empty and the input must be all read.

$L(M)$ is the set of strings accepted by M. $L(M)$ is called the *language recognized by* the pda M. Later we will see that a language L is context-free iff there is a pda M such that $L = L(M)$.

1.7 Example pda

This one is from the text. $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where:

$$
\begin{array}{lcll}
K & = & \{s, f\} & \\
\Sigma & = & \{a, b, c\} & \\
\Gamma & = & \{a, b\} & \\
F & = & \{f\} & \\
 & & ((s, a, e), (s, a)) & \text{push } a \\
 & & ((s, b, e), (s, b)) & \text{push } b \\
\Delta & = & ((s, c, e), (f, e)) & \text{found middle of string} \\
 & & ((f, a, a), (f, e)) & \text{pop } a\text{, match input} \\
 & & ((f, b, b), (f, e)) & \text{pop } b\text{, match input}
\end{array}
$$

Here's how it works on the input *abbcbba*:

$(s, abbcbba, e) \vdash_M (s, bbcbba, a) \vdash_M (s, bcbba, ba) \vdash_M (s, cbba, bba) \vdash_M (f, bba, bba) \vdash_M (f, ba, ba) \vdash_M (f, a, a) \vdash_M (f, e, e)$.

Thus $(s, abbcbba, e) \vdash_M^* (f, e, e)$, so M accepts the input *abbcbba*. In general, M recognizes the language $\{wcw^R : w \in \{a, b\}^*\}$.

Note that in order to show that M rejects an input, it is necessary to consider all possible computation sequences.

1.8 Another example

$M = (K, \Sigma, \Gamma, \Delta, s, F)$ where:

$$
\begin{array}{lcll}
K & = & \{s, f\} & \\
\Sigma & = & \{a, b\} & \\
\Gamma & = & \{a, b\} & \\
F & = & \{f\} & \\
 & & ((s, a, e), (s, a)) & \text{push } a \\
 & & ((s, b, e), (s, b)) & \text{push } b \\
\Delta & = & ((s, e, e), (f, e)) & \text{guess middle of string} \\
 & & ((f, a, a), (f, e)) & \text{pop } a\text{, match input} \\
 & & ((f, b, b), (f, e)) & \text{pop } b\text{, match input}
\end{array}
$$

Here's how it works on the input *abbbba*:

$(s, abbbba, e) \vdash_M (s, bbbba, a) \vdash_M (s, bbba, ba) \vdash_M (s, bba, bba) \vdash_M (f, bba, bba) \vdash_M (f, ba, ba) \vdash_M (f, a, a) \vdash_M (f, e, e)$.

Thus $(s, abbbba, e) \vdash_M^* (f, e, e)$, so M accepts the input *abbbba*.

- In general, M recognizes the language $\{ww^R : w \in \{a, b\}^*\}$, but M has to guess the middle of the string nondeterministically.

- If M guesses wrong, the computation will not lead to acceptance, but M is nondeterministic, so it accepts the input if there is at least one accepting computation.

1.9 Problems

How might a push-down automaton recognize the language $\{a^n b^n : n \geq 0\}$?

Can you figure out in general how a push-down automaton might recognize the language consisting of all strings of a and b that contain an equal number of a and b? Can you figure out a context-free grammar for this language? Can you prove that your grammar is correct?