# Context-Free Languages and Automata

**Before:** We had a class of languages (the regular languages) which contained exactly those languages recognized by a particular kind of automata (the finite automata).

**Now:** We have a more general class of languages (the context-free languages), and we would like to get a similar kind of characterization by automata.

The appropriate kind of automaton is called a *pushdown automaton*.

# Pushdown Automata

**Idea:** Reduce limitation of finite memory by adding an auxiliary memory device: the *pushdown store* or *stack*.

- The stack can hold an arbitrarily long string of symbols, but it can only be accessed in LIFO fashion.
- A finite-state control is still used for the "program".
- We will start out by considering *nondeterministic* pushdown automata.

# Using a Pushdown Automaton

A nondeterministic pushdown automaton can recognize

$$\{ww^R \mid w \in \Sigma^*\}$$

- Read symbols and push them onto the stack.
- Guess nondeterministically when $w$ has been read.
- Read symbols and compare them with symbols popped from the stack.
- Accept if no mismatch and the stack is empty when the end of input is reached.

# Formal Definition of Pushdown Automaton

**Def:** *(Sipser, Def. 2.13)* A *pushdown automaton* is a six-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where

1. $Q$ is the set of *states*,
2. $\Sigma$ is the *input alphabet*,
3. $\Gamma$ is the *stack alphabet*,
4. $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the *transition function*,
5. $q_0$ is the *start state*, and
6. $F \subseteq Q$ is the set of *accept states*.

# Notes

- $(r, b) \in \delta(q, w, a)$ *means:* "When in state $q$ and stack has the form $at$ for some $t \in \Gamma^*$, then consume input $w$, set stack to $bt$, and go to state $r$."
- The automaton is nondeterministic:
  - $\delta(q, w, a)$ might be $\phi$, or might contain multiple $(r, b)$.
  - The input $w$ consumed might be $\epsilon$.
  - If $a = \epsilon$, then nothing is popped from the stack.
  - If $b = \epsilon$, then nothing is pushed on the stack.
- There is no direct "stack empty" test, but we can work around by initially pushing a special marker \$.

# Computation of a PDA

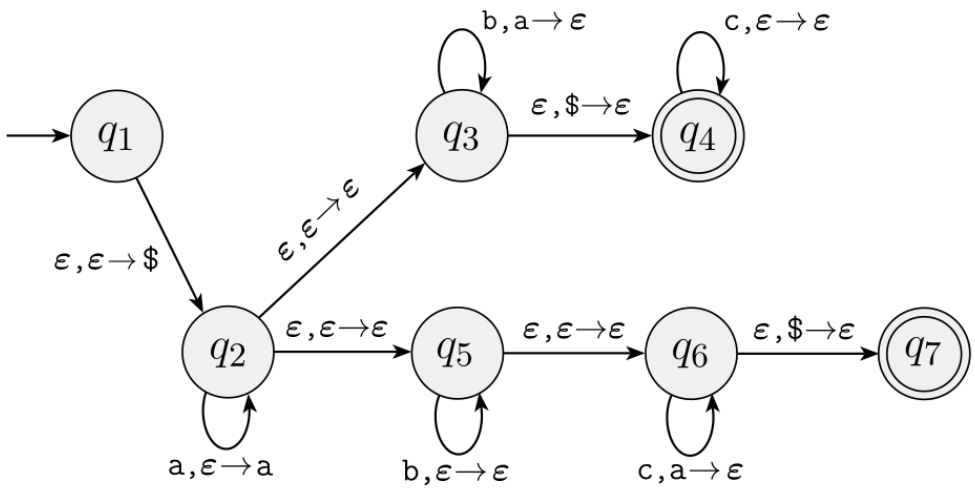
**Def.** A PDA $P$ *accepts* input $w$ if $w$ can be written as $w = w_1 w_2 \dots w_m$, where each $w_i \in \Sigma_e$, and there exist states $r_0, r_1, \dots r_m \in Q$ and strings $s_0, s_1, \dots, s_m \in \Gamma^*$, such that

1. $r_0 = q_0$ and $s_0 = \epsilon$.
2. For $i = 0, \dots, m-1$, there exist $a_i, b_i \in \Gamma_\epsilon$ and $t \in \Gamma^*$, such that $s_i = at$, $s_{i+1} = bt$, and $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$.
3. $r_m \in F$.

# Example (Sipser 2.16)

A PDA that recognizes the language:

$$\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$$



# NFA's and PDAs

Every NFA "is" a PDA, which simply ignores its stack.

Details?

# PDA's from CFG's

**Lemma:** *(Sipser, 2.21)* If a language is context-free, then some pushdown automaton recognizes it.

**Proof Idea:** We construct a pushdown automaton from a grammar that generates the language.

- The stack maintains a sentential form that we need to derive the remainder of the input string.
- An input symbol on top of the stack is *matched* against the input string.
- A variable on top of the stack is *expanded* by the RHS of a nondeterministically chosen rule.
- If the stack becomes empty ($ on top), go to the accept state.

# Some Details

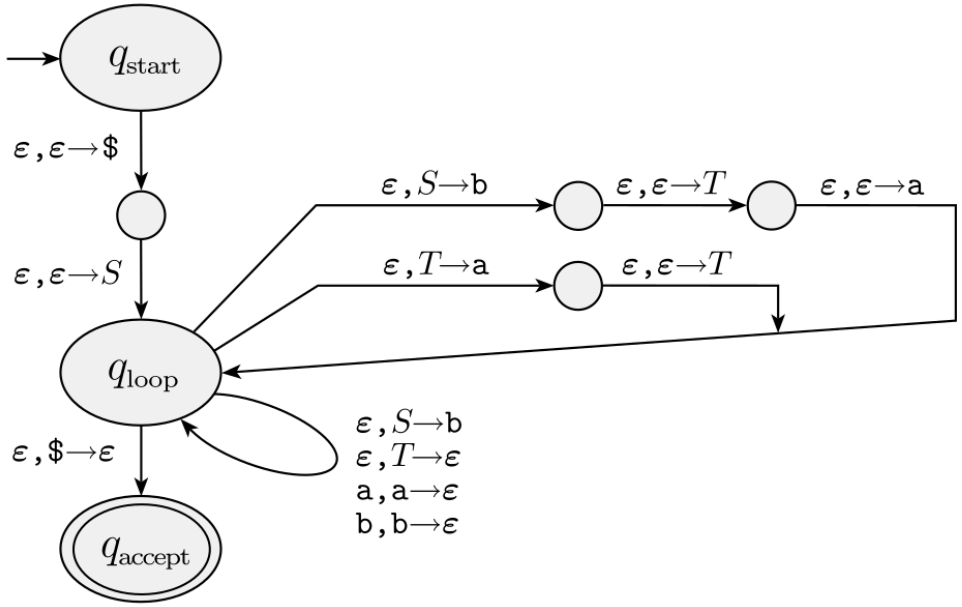
There are three "main states" for the PDA: $q_{start}, q_{loop}, q_{accept}$.

- From $q_{start}$, the stack is initialized to $S\$$ and the new state is $q_{loop}$.
- From $q_{loop}$, there are:
  - Transitions to "match" input against terminals on top of the stack and return to $q_{loop}$.
  - Transitions to expand a nonterminal on top of the stack and return to $q_{loop}$.
  - A transition to $q_{accept}$ when $\$$ is on top of the stack.

Chains of transitions (with "extra" states) are used to push sequences of symbols onto the stack.

# Example (Sipser 2.25)

$$
\begin{aligned}
S &\rightarrow aTb \mid b \\
T &\rightarrow Ta \mid \epsilon
\end{aligned}
$$



# CFG's from PDA's

**Lemma:** *(Sipser, 2.27)* If a pushdown automaton recognizes a language, then it is context-free.

**Proof Idea:** Given a PDA $P$, we construct an equivalent CFG $G$.

- Variables of $G$ will be $A_{pq}$, where $p$ and $q$ are states of $P$.
- *Idea:* The rules of $G$ are designed so that:
  - $A_{pq} \stackrel{*}{\Rightarrow} w$ if and only if $w$ can bring $P$ from state $p$ with empty stack to state $q$ with empty stack.

# Some Assumptions

We assume:

- $P$ has exactly one accept state: $q_{\text{accept}}$.
- $P$ empties its stack before accepting.
- Each transition of $P$ either pushes one symbol or pops one symbol, but not both (or neither).

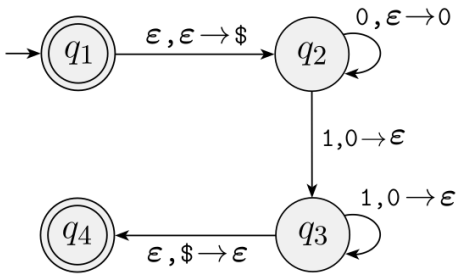
These assumptions are without loss of generality, because we can modify $P$ to arrange that they are satisfied. *(How?)*

# Definition of $G$

- $V = \{A_{pq} \mid p, q \in Q\}$
- $S = A_{q_0,q_{\text{accept}}}$.
- For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\epsilon$: if $(r, u) \in \delta(p, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, u)$, then put $A_{pq} \to aA_{rs}b$ in $G$. Represents computations from $p$ to $q$ that start by pushing $u$ and end by popping $u$.
- For each $p, q, r \in Q$, put rule $A_{pr} \to A_{pq}A_{qr}$ in $G$. Represents computations from $p$ to $q$ via intermediate $r$ with empty stack.
- For each $p \in Q$, put rule $A_{pp} \to \epsilon$ in $G$.

# Example

Let's try this on the PDA of Sipser, 2.15:




- Rules of form $A_{pq} \to a\ A_{rs}\ b$:

$$
\begin{array}{lcll}
A_{14} & \to & A_{23} & \text{(push/pop \$)} \\
A_{23} & \to & 0\ A_{22}\ 1 & \text{(push/pop 0)} \\
A_{23} & \to & 0\ A_{23}\ 1 & \text{(push/pop 0)}
\end{array}
$$

Only two transitions push something in $\Gamma$!

- Rules of form $A_{pr} \to A_{pq}A_{qr}$: there are $4^3 = 64$ of them but none end up deriving any strings (in this example).
- Rules of form $A_{pp} \to \epsilon$: there are 4 of them, but only one is useful: $A_{22} \to \epsilon$.

# Correctness ("forward" direction)

**Claim:** *(Sipser, 2.30)* If $A_{pq} \stackrel{*}{\Rightarrow} x$, then $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack.

**Proof.** By induction on the number of steps $k$ in the derivation $A_{pq} \stackrel{*}{\Rightarrow} x$ (Note: must have $k \geq 1$).

- **Basis:** $k = 1$. The only possibility is $A_{pp} \rightarrow \epsilon$.
- **Induction:** Assume true for all derivations of length $\leq k$, for some $k \geq 1$, and prove true for all derivations of length $\leq k + 1$.

Possibilities for first step are:

1. $A_{pq} \stackrel{*}{\Rightarrow} aA_{rs}b$.
   Then $x = ayb$, where $A_{rs} \stackrel{*}{\Rightarrow} y$ by a derivation of length $k$. By the Ind. Hyp., $y$ can bring $P$ from $r$ with empty stack to $s$ with empty stack. Because $A_{pq} \to aA_{rs}b$ is a rule, $P$ can read $a$ in state $r$ and push $u$, and read $b$ in state $s$ and pop $u$.

2. $A_{pq} \stackrel{*}{\Rightarrow} A_{pr}A_{rq}$.
   Then $x = yz$, where $A_{pr} \stackrel{*}{\Rightarrow} y$ and $A_{rq} \stackrel{*}{\Rightarrow} z$, both by derivations of length $\leq k$. By the Ind. Hyp., $y$ can bring $P$ from $p$ to $r$ with empty stack and $z$ can bring $P$ from $r$ to $q$ with empty stack.

# Correctness ("backward" direction)

**Claim:** *(Sipser, 2.31)* If $x$ can bring $P$ from $p$ with empty stack to $q$ with empty stack, then $A_{pq} \stackrel{*}{\Rightarrow} x$.

**Proof.** By induction on the number of steps $k$ (could be 0) in the computation of $P$.

- **Basis:** $k = 0$. The computation starts and ends in the same state $p$, so $G$ has a rule $A_{pp} \to \epsilon$.
- **Induction Step:** Assume true for computations of length $\leq k$, for some $k \geq 0$, and prove true for computations of length $\leq k + 1$.

**Cases:** (1) the stack never becomes empty before the end of the computation, or (2) it becomes empty somewhere in the middle of the computation as well.

1. If it never becomes empty, then for some $r, a, y, b, s, u$, $x = ayb$, the first step reads $a$, pushes $u$ and goes to $r$, and the last step (from state $s$) reads $b$ and pops $u$. Then $A_{pq} \stackrel{*}{\Rightarrow} x$ by a derivation whose first step uses $A_{pq} \to aA_{rs}b$ and the remainder is a derivation $A_{rs} \stackrel{*}{\Rightarrow} y$, which exists by applying the induction hypothesis to the computation from $r$ to $s$ that reads $y$.

2. If it becomes empty in the middle, then $x = yz$, where $y$ brings $P$ from $p$ to some $r$ with an empty stack, and $z$ brings $P$ from $r$ to $q$ with an empty stack. Then $A_{pq} \stackrel{*}{\Rightarrow} x$ by a derivation that starts with $A_{pq} \rightarrow A_{pr}A_{rs}$ and continues with derivations $A_{pr}A_{rs} \stackrel{*}{\Rightarrow} yA_{rs} \stackrel{*}{\Rightarrow} yz$, which exist by applying the induction hypothesis to the computations from $p$ to $r$ and from $r$ to $s$.

# Intersection of a Regular and a CF Language

**Prop.** The intersection of a regular language and a context-free language is context-free.

**Proof Idea?**

# Intersection of a Regular and a CF Language

**Prop.** The intersection of a regular language and a context-free language is context-free.

**Proof Idea:** Product of DFA and PDA. Suppose

$$\begin{aligned} M_1 &= (Q_1, \Sigma, \delta_1, q_1, F_1) \\ P_2 &= (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2), \end{aligned}$$

where DFA $M_1$ recognizes $L_1$ and PDA $P_2$ recognizes $L_2$. Construct

$$P = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), F_1 \times F_2)$$

where

$$\delta((r_1, r_2), a, u) = \begin{cases} \{((\delta_1(r_1, a), s), v) \mid (s, v) \in \delta_2(r_2, a, u)\}, \\ \qquad \text{if } a \neq \epsilon \\ \{((r_1, s), v) \mid (s, v) \in \delta_2(r_2, \epsilon, u)\}, \\ \qquad \text{if } a = \epsilon \end{cases}$$

If $a \neq \epsilon$ then $M_1$ and $P_2$ take a joint step. If $a = \epsilon$, then $P_2$ takes a step and $M_1$ does nothing.

**Claim:** $P$ accepts $w$ if and only if $M_1$ accepts $w$ and $P_2$ accepts $w$.