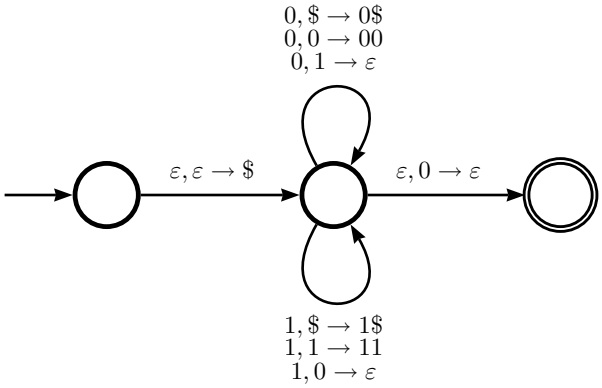
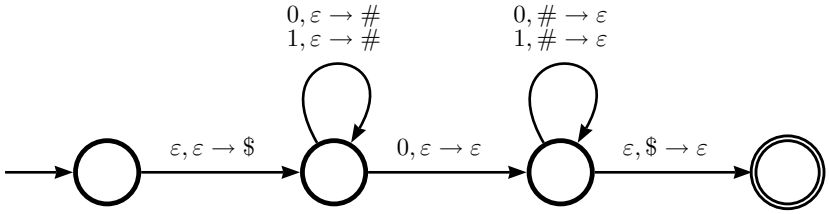
1. **(Designing PDAs)**

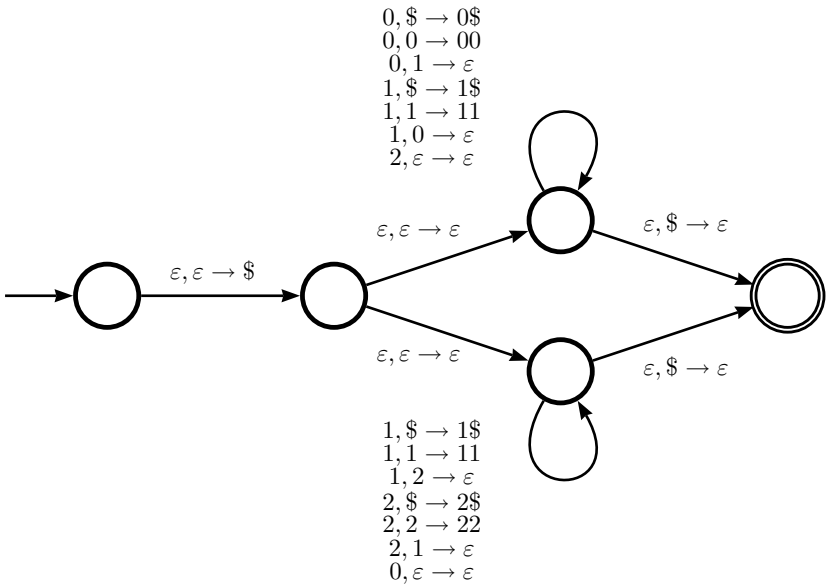
1.1. The PDA uses its stack to keep track of how many more 0's than 1's it has seen (by having $N_0 - N_1$ 0's in the stack) or how many more 1's than 0's it has seen (by having $N_1 - N_0$ 1's in the stack).



1.2. After pushing the standard "rock", the PDA splits itself into two incarnations at every step: one believing that the next input symbol is the middle 0, the other believing that it has not reached the middle symbol yet. If either incarnation finds its belief to be wrong, it dies.



1.3. This PDA has two parts, each of which is similar to the PDA from 1.1. One checks if $N_0(x) = N_1(x)$, ignoring 2's, while the other checks if $N_1(x) = N_2(x)$, ignoring 0's.



2. **(An alternate approach to regular languages)**

2.1. To prove that "$\equiv_A$" in an equivalence relation, we need to show that it possesses the following three properties.

1. Reflexivity:

$$\forall w \in \Sigma^*(xw \in A \iff xw \in A) \implies x \equiv_A x.$$

2. Symmetry:

$$\begin{aligned} x \equiv_A y &\implies \forall w \in \Sigma^*(xw \in A \iff yw \in A) \\ &\implies \forall w \in \Sigma^*(yw \in A \iff xw \in A) \\ &\implies y \equiv_A x. \end{aligned}$$

3. Transitivity:

$$\begin{aligned} (x \equiv_A y) \wedge (y \equiv_A z) &\implies \forall w' \in \Sigma^*(xw' \in A \iff yw' \in A) \wedge \quad (1) \\ &\qquad \forall w'' \in \Sigma^*(yw'' \in A \iff zw'' \in A) \\ &\implies \forall w \in \Sigma^* \left[\begin{array}{c} (xw \in A \iff yw \in A) \\ \wedge \\ (yw \in A \iff zw \in A) \end{array} \right] \quad (2) \\ &\implies \forall w \in \Sigma^*(xw \in A \iff zw \in A) \\ &\implies x \equiv_A z. \end{aligned}$$

Hence "$\equiv_A$" is an equivalence relation. □

Note. Pay particular attention to the step (1) $\implies$ (2). For any $w \in \Sigma^*$, applying each of the two statements from (1) gives us $(xw \in A \iff yw \in A)$ and $(yw \in A \iff zw \in A)$, respectively. Hence, (2).

For illustration, suppose the definition of $x \equiv_A y$ had instead been $\exists w \in \Sigma^*(xw \in A \iff yw \in A)$. This step would then fail because taking an arbitrary $w \in \Sigma^*$ may fail to satisfy one or both of the clauses in (1) if $w \neq w'$ or $w \neq w''$.

2.2. $C_1 = \{\varepsilon\}$
$C_2 = \{a, b\}$
$C_3 = \{aa, ba\}$
$C_4 = \{aaa, baa\}$
$C_5 = \overline{(C_1 \cup C_2 \cup C_3 \cup C_4)}$
Note that C_5 is the infinite class alluded to. Any string $x \in C_5$ satisfies $\forall w \in \Sigma^*(xw \notin A)$. In particular, for any two strings $x, y \in C_5$, $\forall w \in \Sigma^*(xw \notin A \wedge yw \notin A)$, so that it is trivially true that $\forall w \in \Sigma^*(xw \in A \iff yw \in A)$.

2.3. $C_1 = a^*$
$C_2 = a^*b^+$
$C_3 = a^*b^*c^+$
$C_4 = \overline{(C_1 \cup C_2 \cup C_3)}$

2.4. $C_1 = (ab \cup ba)^*$
$C_2 = (ab \cup ba)^*a$
$C_3 = (ab \cup ba)^*b$
$C_4 = \overline{(C_1 \cup C_2 \cup C_3)}$

2.5. The left equivalence classes of $\{0^n1^n : n \geq 0\}$ fall into three types. The first type consists of an infinite number of singleton sets:
$C_0^1 = \{\varepsilon\}$
$C_1^1 = \{0\}$
$C_2^1 = \{00\}$
...
$C_k^1 = \{0^k\}$
...
The second type consists of an infinite number of infinite sets:
$C_0^2 = \{0^n1^n : n > 0\}$
$C_1^2 = \{0^n1^{n-1} : n > 1\}$
$C_2^2 = \{0^n1^{n-2} : n > 2\}$
...
$C_k^2 = \{0^n1^{n-k} : n > k\}$
...
Finally, the third type consists of just one class, which contains everything else:
$C^3 = \overline{\left(\bigcup_{k=0}^{\infty} C_k^1\right) \cup \left(\bigcup_{k=0}^{\infty} C_k^2\right)}$.

2.6. The problem can be mathematically formulated as the following claim.

Claim. $\forall x, y \in \Sigma^*, \forall a \in \Sigma([x]_A = [y]_A \implies [xa]_A - [ya]_A)$.

Proof. Given $[x]_A = [y]_A$, we know by definition that

$$\begin{aligned} & x \equiv_A y \\ \implies \quad & \forall w \in \Sigma^*(xw \in A \iff yw \in A) \\ \implies \quad & \forall w \in \Sigma^*(xaw \in A \iff yaw \in A) \\ \implies \quad & xa \equiv_A ya \end{aligned}$$

from which it follows again by definition that $[xa]_A = [ya]_A$. □

2.7. We will design a DFA for A. The key idea is to create a state for each equivalence class of A. This also gives us a very natural definition of the transition function δ, where we transition from state $[x]_A$ to state $[xa]_A$ upon reading the symbol a, so we can define our DFA as follows. Let $M = (Q, \Sigma, \delta, q_0, F)$, where

$$\begin{aligned} Q &= \{[x_1]_A, [x_2]_A, \ldots, [x_n]_A\} \\ \delta([x]_A, a) &= [xa]_A \text{ for every } [x]_A \in Q \text{ and every } a \in \Sigma. \\ q_0 &= [\varepsilon]_A \\ F &= \{[x]_A : x \in A\} \end{aligned}$$

The first question is why δ is well-defined, that is, why it can take us to only one unique state, even had we chosen a different representative y of $[x]_A$. Fortunately, this is precisely the guarantee of 2.6, so that M is a correctly defined DFA.

The second question is why our definition of F, which certainly guarantees that every string in A is accepted, does not allow M to accept some string $y \notin A$. But such a y would have to be $\equiv_A$ to some $x \in A$, that is, $\forall w \in \Sigma^*(yw \in A \iff xw \in A)$. In particular, we can choose $w = \varepsilon$, so that $y \in A \iff x \in A$, contradicting the facts that $y \notin A$ and $x \in A$. It follows that $L(M) = A$, and hence that A is regular. □

2.8. *This solution was written by Amit Chakrabarti*

Informal description: If A is a regular language over the alphabet Σ, we can build a DFA M to recognize it. We will map strings in Σ^* to states of M in a natural way and show that

$$\text{if two strings } x \text{ and } y \text{ map to the same state, then } x \equiv_A y.$$

In other words, if two strings are *not* equivalent, they will map to distinct states of M. If A in fact has infinitely many left equivalence classes, then we can find infinitely many strings (one from each class) such that no two are equivalent, so they would have to map into an infinite number of different states. However, being a DFA, M only has a finite number of states, giving us a contradiction.

Formal proof:

Given a regular language A, we can first build $M(A) = (Q, \Sigma, \delta, q_0, F)$. Recall the function $\hat{\delta} : Q \times \Sigma^* \to Q$ from class (and the Oct 10 lecture notes): $\hat{\delta}(q, x)$ tells us the state $M(A)$ ends up in when it is started in state q and fed input x. For each string $x \in \Sigma^*$, define $q(x) = \hat{\delta}(q_0, x)$. This is the state that we're going to map x to.

We claim that if $q(x) = q(y)$, then $x \equiv_A y$. As explained in the informal description, this would complete the proof.

Suppose $q(x) = q(y)$. We must show that $\forall w\,(xw \in A \iff yw \in A)$. Suppose w is a string such that $xw \in A$. Then, the machine $M(A)$ must accept xw, so $q(xw) \in F$. But $q(xw) = \hat{\delta}(q(x), w) = \hat{\delta}(q(y), w) = q(yw)$; thus $q(yw) \in F$ as well. So $M(A)$ accepts yw, i.e., $yw \in A$. We have shown that $xw \in A \implies yw \in A$. Clearly, a very similar proof shows that $xw \in A \impliedby yw \in A$, and we are done. □

2.9. We know from 2.5 that the language $\{0^n 1^n : n \geq 0\}$ has infinitely many left equivalences classes. It follows immediately from 2.8 that it cannot be regular. □

2.10. Let $A = \{x \in \{0,1\}^* : x \text{ is a palindrome}\}$. Now, consider another infinite set of strings over the same alphabet given by $\{0^n 1 : n \geq 0\} = \{1, 01, 001, 0001, \dots\}$. Then for any $m, n \geq 0$, $m \neq n$, $0^n 10^n$ is a palindrome (i.e., $\in A$) whereas $0^m 10^n$ is not (i.e., $\notin A$), so by definition $0^n 1 \not\equiv_A 0^m 1$ (for $w = 0^n$). We therefore have infinitely many strings no two of which belong to the same left equivalence class of A, so by the Pigeonhole Principle A must have infinitely many left equivalence classes. It follows from 2.8 that A is nonregular. □