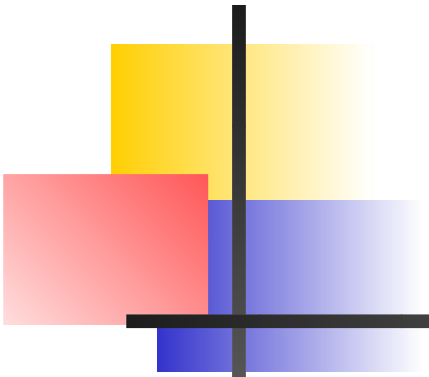


# Regular Languages

**Definition:** A language is called a **regular language** if some finite automaton recognizes it

# Regular Operations

**Definition:** Let $A$ and $B$ be regular languages, we define the following operations:

**Union:** $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.

**Concatenation:** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.

**Star (Kleene Closure):** $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

# $A \cup B$ is a Regular Language

**Theorem:** If $A_1$ and $A_2$ are regular languages, then so is $A \cup B$.[a]

[a] We postpone the proofs of the other regular operations until we discussed nondeterminism.

# $A \cup B$ is a Regular Language

**PROOF**

Let $M_1$ recognize $A_1$, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
$M_2$ recognize $A_2$, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct $M$ to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the ***Cartesian product*** of sets $Q_1$ and $Q_2$ and is written $Q_1 \times Q_2$. It is the set of all pairs of states, the first from $Q_1$ and the second from $Q_2$.
2. $\Sigma$, the alphabet, is the same as in $M_1$ and $M_2$. In this theorem and in all subsequent similar theorems, we assume for simplicity that both $M_1$ and $M_2$ have the same input alphabet $\Sigma$. The theorem remains true if they have different alphabets, $\Sigma_1$ and $\Sigma_2$. We would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.
3. $\delta$, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let
   $$\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big).$$
   Hence $\delta$ gets a state of $M$ (which actually is a pair of states from $M_1$ and $M_2$), together with an input symbol, and returns $M$'s next state.
4. $q_0$ is the pair $(q_1, q_2)$.
5. $F$ is the set of pairs in which either member is an accept state of $M_1$ or $M_2$. We can write it as
   $$F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}.$$
   This expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. (Note that it is *not* the same as $F = F_1 \times F_2$. What would that give us instead?[3])

# Nondeterminism

Up to now we have only considered machines where, given a state and given an input symbol, the next state is uniquely defined - ***deterministic machines (DFA)***

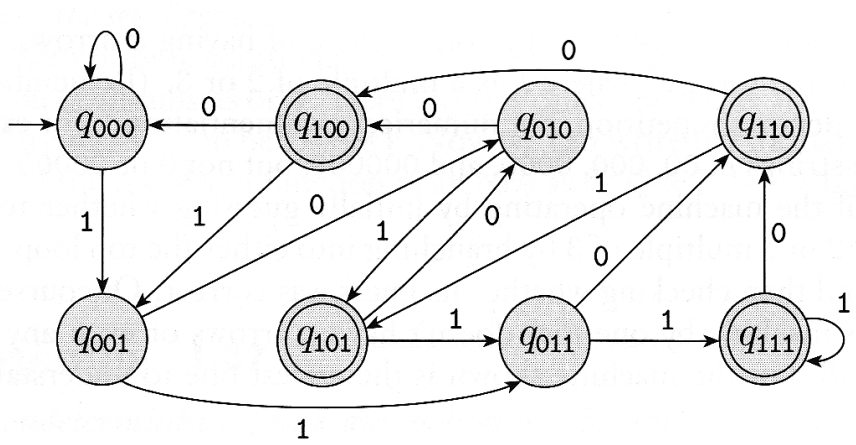
However, we could conceive of machines that, given a state and a particular input symbol, have a choice of states to move to - ***nondeterministic machines (NFA)***

We will see that nondeterminism does not add to the power of the NFA to recognize larger sets of languages, but nondeterminism adds to the expressiveness of the machines, i.e., it is much easier to build a nondeterministic machine for a particular regular language than a deterministic machine.

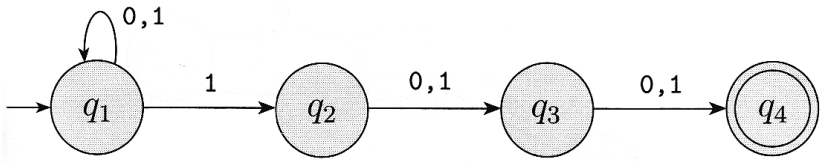
# NFA

**Example:** Let $A$ be the language consisting of all strings over $\Sigma = \{0, 1\}$ containing a $1$ in the third position from the end (e.g. $00100, 0101, 1100 \in A$).

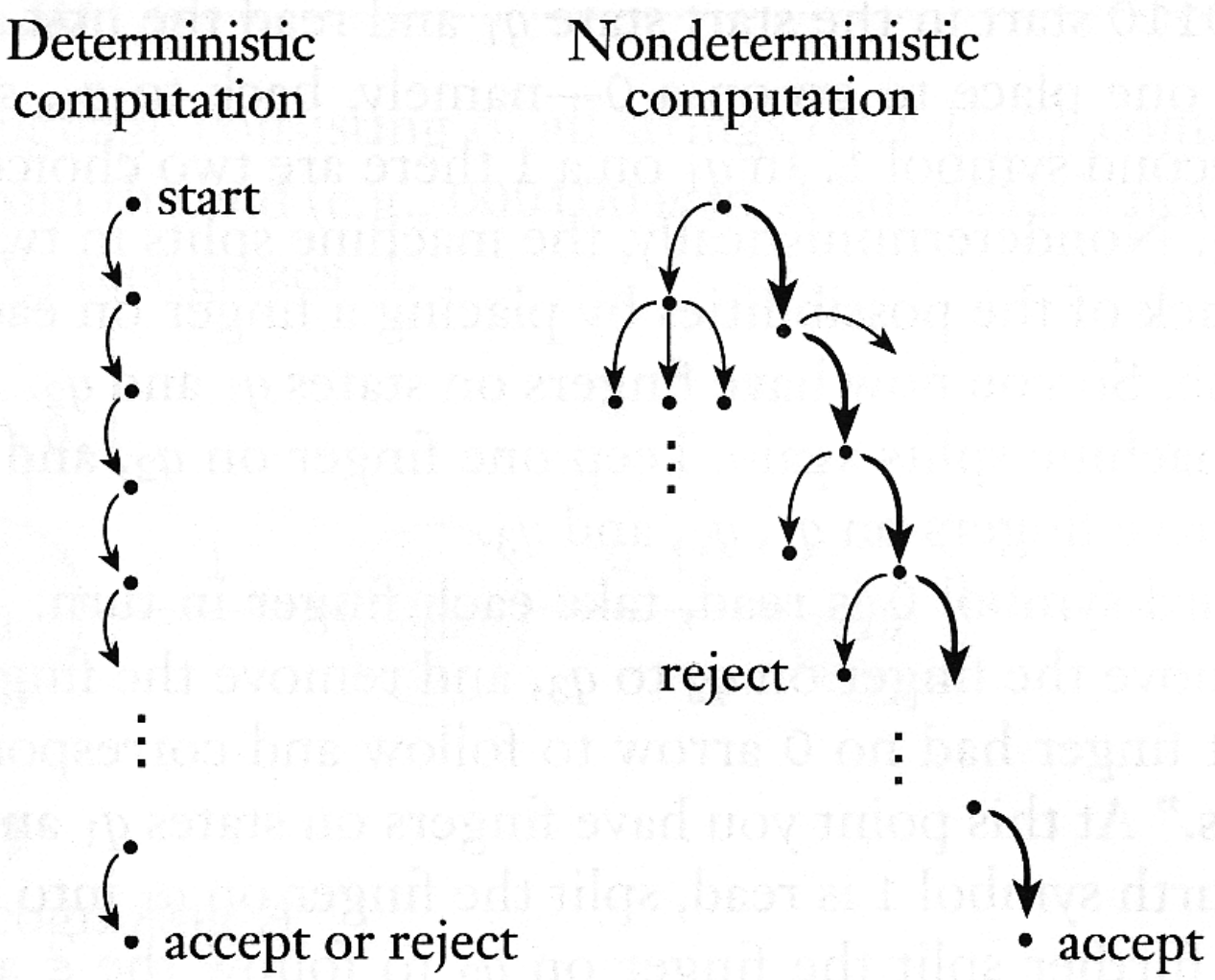
DFA:




NFA:

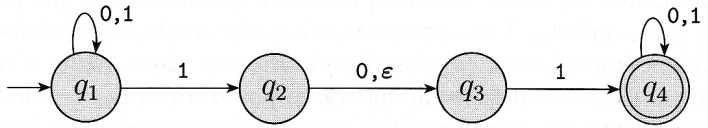


# Computing with NFAs

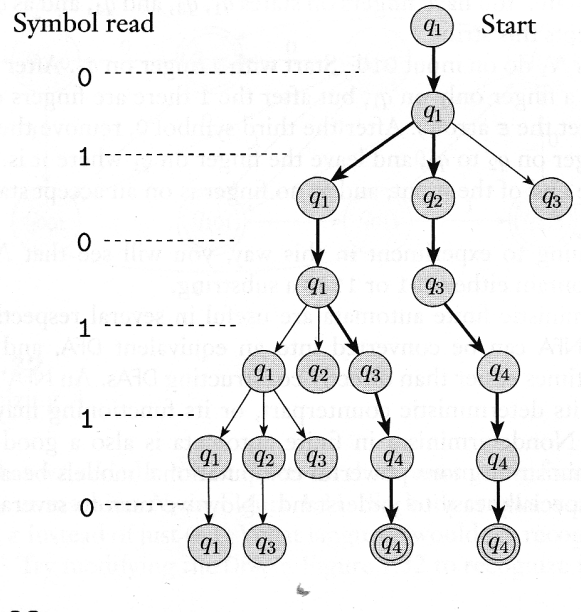


# Computing with NFAs

**Example:** Consider the computation of the following NFA,




with input string $s = 010110$,



# Formal Def. of NFA

**Definition:** a **nondeterministic** **finite automaton** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. $Q$ is a finite set called the **states**,
2. $\Sigma$ is a finite set called the **alphabet**,
3. $\delta : \ Q \times \Sigma_\epsilon \rightarrow P(Q)$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states**.

# DFA $\equiv$ NFA

We say that two machines are ***equivalent*** if they recognize the same language.

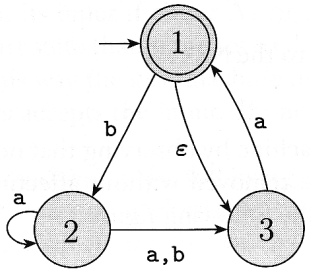
**Theorem:** Every NFA has an equivalent DFA.

**Proof Sketch:** For every NFA we can construct a DFA that simulates it. We do so by introducing new states that remove multiple transitions on the same input symbol as well as transitions on the empty symbol $\epsilon$.[a]

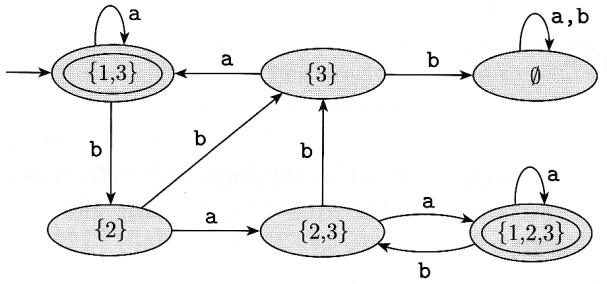
[a] For a formal proof see pp. 55 (1st & 2nd eds.)

# DFA $\equiv$ NFA

**Example:** Consider the NFA,




and its equivalent DFA,




Consider input strings $bba$, $baa$, and $a$.

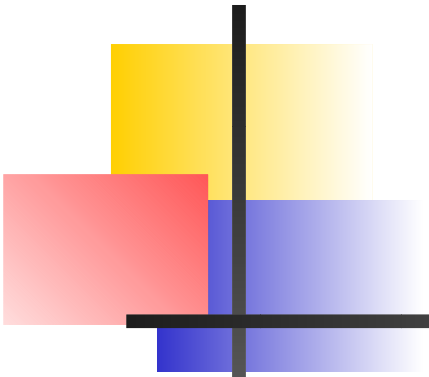
# Regular Languages & NFAs

**Theorem:** A language is regular if and only if some NFA recognizes it

**Proof Sketch:**

**If a language is regular then some NFA recognizes it.** By definition, if a DFA recognizes a language, then it is regular. It follows that there will be some NFAs that are equivalent to the recognizing DFA. Therefore, if a language is regular, then there will be some NFA that recognizes it.

**If some NFA recognizes a language then the language is regular.** Any NFA can be converted into an equivalent DFA. By definition this DFA recognizes the same language as the NFA. The language a DFA recognizes is regular. This implies that if an NFA recognizes a language, then the language is regular.



# More on NFAs

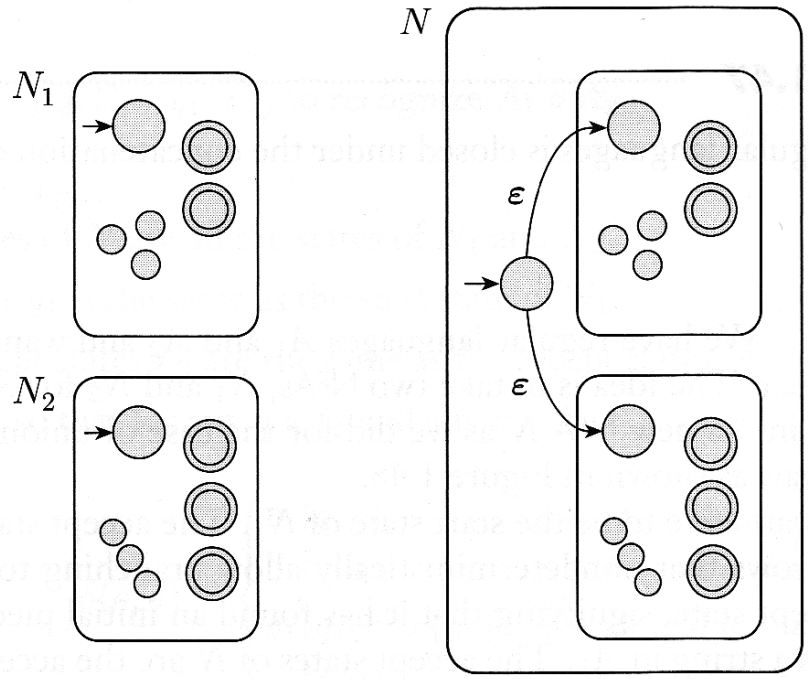
NFAs facilitate proofs by allowing an easier construction of machines that recognize regular languages.

Let’s revisit our regular operations.

# Union - NFA

**Theorem:** The class of regular languages is closed under the union operation.

**Proof Sketch:** Let $N_1$ and $N_2$ be the NFAs that recognize the languages $A_1$ and $A_2$, respectively, we can then construct an NFA $N$ that recognizes the language $A_1 \cup A_2$ as follows:[a]

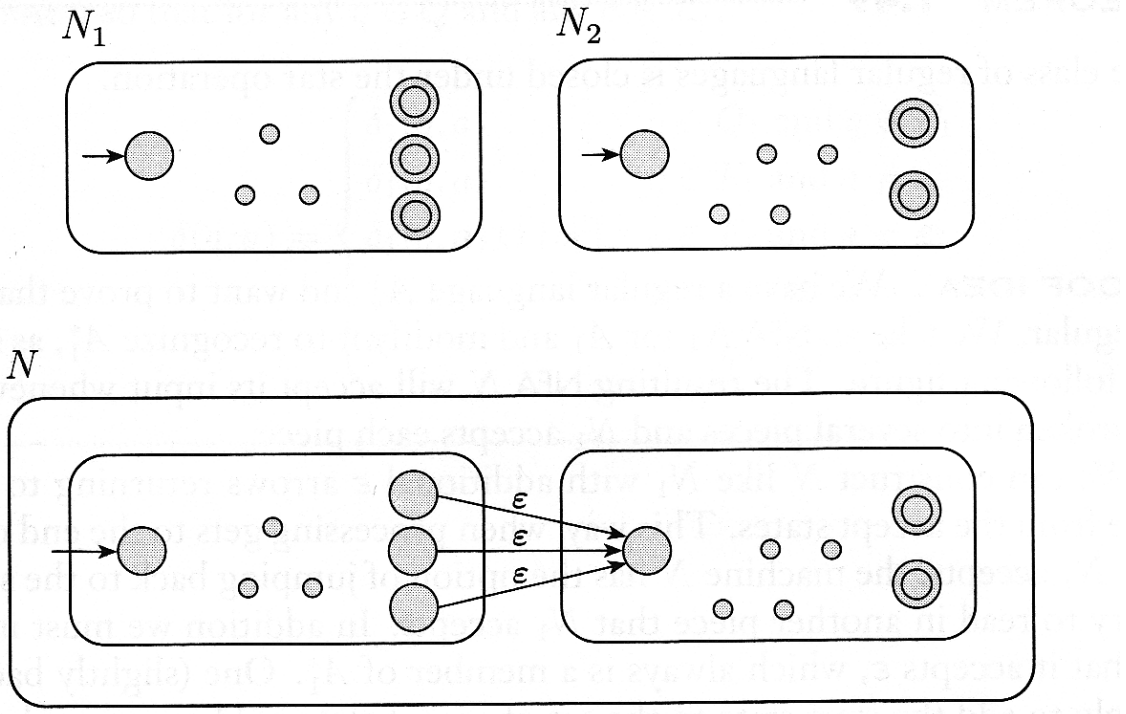



[a] Formal proofs of this and the following theorems appear in the text pp59ff 1st & 2nd eds.

# Concatenation - NFA

**Theorem:** The class of regular languages is closed under the concatenation operation.

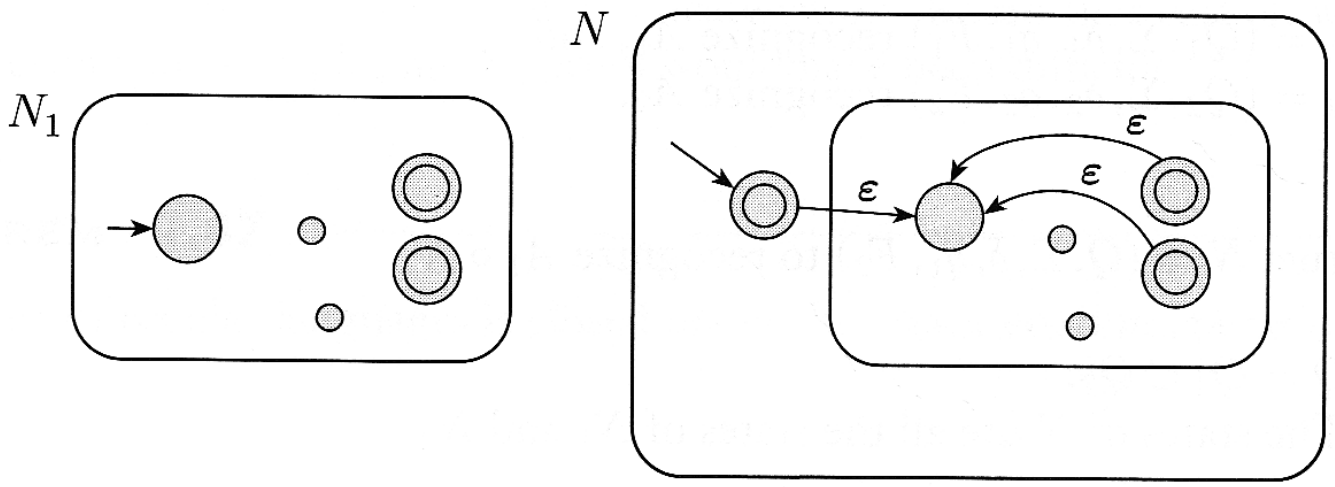
**Proof Sketch:** Let $N_1$ and $N_2$ be the NFAs that recognize the languages $A_1$ and $A_2$, respectively, we can then construct an NFA $N$ that recognizes the language $A_1 \circ A_2$ as follows:



# Star - NFA

**Theorem:** The class of regular languages is closed under the kleene closure operation.

**Proof Sketch:** Let $N_1$ be the NFA that recognizes the language $A_1$, we can then construct an NFA $N$ that recognizes the language $A_1^*$ as follows:



# Nonregular Languages

Are there languages that FAs cannot recognize? Yes!

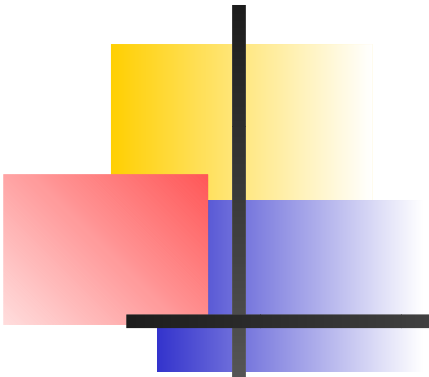
The prototypical language in this class of nonregular languages is the language

$$\{0^n 1^n \mid 0, 1 \in \Sigma \text{ and } n \geq 0\}.$$

The problem with this language is that the FA has to keep track of $i$ 0's and make sure that for every string in the language there are corresponding number of $i$ 1's.

For the general case FAs cannot perform this task.

Can we prove this? Yes, the ***pumping lemma***.



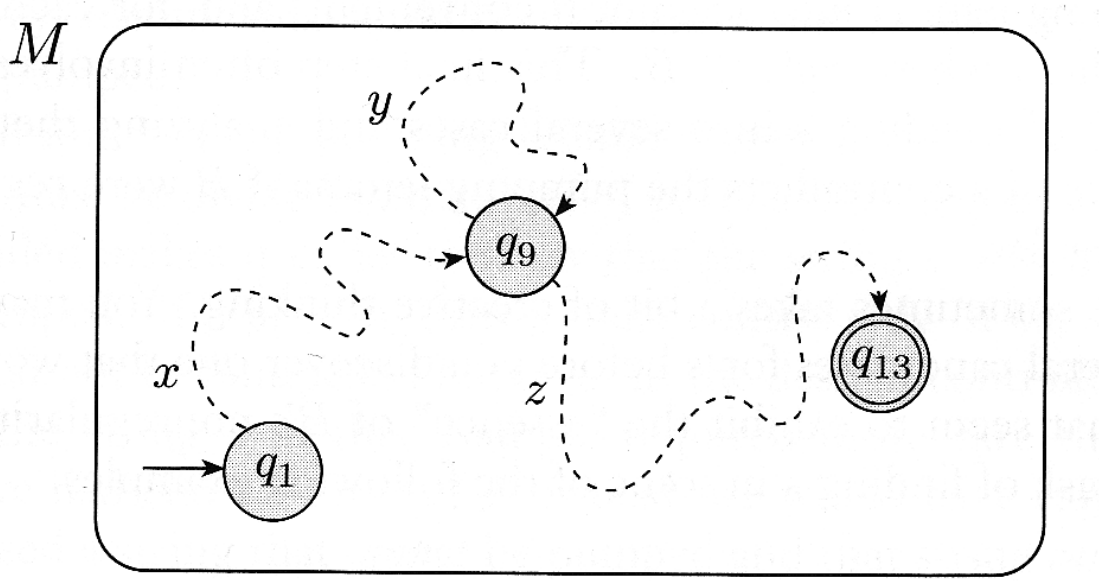
# The Pumping Lemma

The pumping lemma is based on the fact that if we pick a string with more symbols in it than there are states then there has to be a loop in the sequence of states. Given this loop we can then walk around this loop as many times as we please and for regular languages the strings generated are members of the original language of the machine. It is interesting to observe that for nonregular languages this pumping does not work! This gives us a way to prove that a language is not regular using contradiction.

# The Pumping Lemma

**Thereom (The Pumping Lemma):** If $A$ is a regular language, then there is a number $p$ (the ***pumping length***) where, if $s$ is any string in $A$ of length at least $p$, then $s$ may be divided into three pieces, $s = xyz$, such that

1. for each $i \geq 0, xy^iz \in A$,
2. $| y | > 0$, and
3. $| xy | \leq p$.



# PL - Example

**EXAMPLE 1.73**

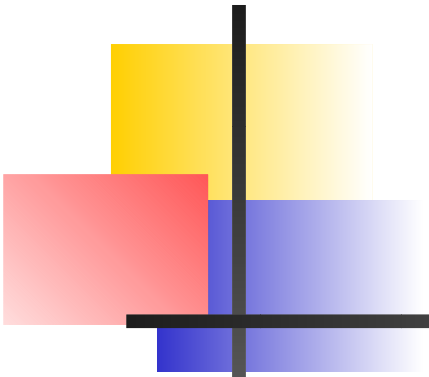
Let $B$ be the language $\{0^n1^n \mid n \geq 0\}$. We use the pumping lemma to prove that $B$ is not regular. The proof is by contradiction.

Assume to the contrary that $B$ is regular. Let $p$ be the pumping length given by the pumping lemma. Choose $s$ to be the string $0^p1^p$. Because $s$ is a member of $B$ and $s$ has length more than $p$, the pumping lemma guarantees that $s$ can be split into three pieces, $s = xyz$, where for any $i \geq 0$ the string $xy^iz$ is in $B$. We consider three cases to show that this result is impossible.

1. The string $y$ consists only of 0s. In this case the string $xyyz$ has more 0s than 1s and so is not a member of $B$, violating condition 1 of the pumping lemma. This case is a contradiction.
2. The string $y$ consists only of 1s. This case also gives a contradiction.
3. The string $y$ consists of both 0s and 1s. In this case the string $xyyz$ may have the same number of 0s and 1s, but they will be out of order with some 1s before 0s. Hence it is not a member of $B$, which is a contradiction.

Thus a contradiction is unavoidable if we make the assumption that $B$ is regular, so $B$ is not regular. Note that we can simplify this argument by applying condition 3 of the pumping lemma to eliminate cases 2 and 3.

In this example, finding the string $s$ was easy, because any string in $B$ of length $p$ or more would work. In the next two examples some choices for $s$ do not work, so additional care is required. ■



# Proof Tools

The material we have developed up to this point gives us the following tools to prove properties of languages:

- In order to prove that a language is regular we construct a FA (either DFA or NFA since they are equivalent) that recognizes the language.
- In order to prove that a language is nonregular, we assume that it is regular and then show that this assumption leads to a contradiction in the pumping lemma.