1 The Regular Operations (Continued)

We will review some of the regular operations.

1.1 Concatenation

Recall that concatenation is defined by:

$$A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$$

How would this works for $A = \{a^n \mid n \geq 0\}$ and $B = \{b^m \mid m \geq 0\}$?

$$A \circ B = \{a^n b^m \mid n, m \geq 0\}$$

We can use NFAs to show that this is the case.

2 Nondeterministic Finite Automata (1.2)

In a deterministic finite automata, when the machine was in a given state and reads the next input symbol, we knew that the next state is; that's why it's called *determinstic*, because it was already determined. **However**, in a *nondeterministic* machine, several choices may exist for the next state at any point. In general, nondeterminism is a *generalization* of determinism; that is, every deterministic finite automaton is automatically a nondeterminism finite automaton.

2.1 The Differences Between DFA and NFA

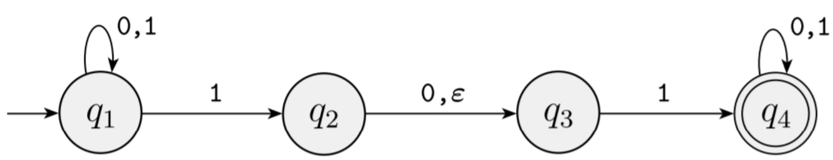

Figure: The nondeterministic finite automaton N_1.

DFA	NFA
• Every state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet. • There is a unique computation path for each input. • Labels on the transition arrows are symbols from the alphabet.	• Not every state in an NFA needs exactly one transition arrow for each symbol. In an NFA, a state may have zero, one, or many exiting arrows for each alphabet symbol. • We may allow several (or zero) alternative computations on the same input. • NFAs may have arrows labeled with members of the alphabet or ϵ. Zero, one, or many arrows may exit from each state with the label ϵ. For example, the above figure has one transition arrow with ϵ as a label.

2.2 NFA Computation

How does an NFA compute? Suppose that we are running an NFA on an input string and come to a state with multiple ways to proceed. Suppose, in fact, that we use the NFA above: N_1. Additionally, suppose that we are at state q_1, and the next input symbol is 1.

- After reading this symbol, the machine **splits into multiple copies of itself** and follows *all* the possibilities in *parallel*. In other words, each copy of the machine takes one of the possible ways to proceed and continues as before.
- If there are subsequent choices, the machine splits again.
- If the next input symbol doesn't appear on any of the arrows exiting the stae occupied by a copy of the machine, that copy of the machine dies.
- If any one of these copies of the machine is in an accept state at the end of the input, the NFA *accepts* the input string.

What happens when we come across a state with an ϵ symbol on an exiting arrow? Well, without reading any input, the machine splits into *multiple* copies, one following each of the exiting ϵ-labeled arrows and one staying at the current state. The machine, then, proceeds nondeterministically as before. So, really, ϵ transitions allow the machine to **transition betweenstates spontaneously** without consuming any input symbols.

2.3 Formal Definition of NFA

Definition 2.1: Nondeterministic Finite Automaton

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. Σ is a finite set called the **alphabet**.
3. $\delta : Q \times \Sigma \cup \{\epsilon\} \mapsto \mathcal{P}(Q)$ is the **transition function**.
4. $q_0 \in Q$ is the **start state**.
5. $F \subseteq Q$ is the **set of accept states** (sometimes also called *final states*).

Remarks:

- $\Sigma \cup \{\epsilon\}$ is sometimes written as Σ_ϵ.
- We say that $\delta(q, c)$ returns a **set** of states; more precisely, a subset of Q. Here, $c \in \Sigma$ or ϵ and $q \in Q$.

2.4 Acceptance in an NFA

We say that an NFA $(Q, \Sigma, \delta, q_0, F)$ accepts a string w in Σ^* if and only if we can write $w = y_1 y_2 \ldots y_m$ where each $y_i \in \Sigma_\epsilon$ and **there is** a sequence of states $r_0, \ldots, r_m \in Q$ such that:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for each $i = 0, 1, \ldots, m-1$
3. $r_m \in F$.