Nondeterministic Finite Automata

Nondeterminism gives a machine multiple options for its moves.

Nondeterministic Finite Automata

In a ***nondeterministic*** finite automaton (NFA), for each state there can be zero, one, two, or more transitions corresponding to a particular symbol.

If NFA gets to state with more than one possible transition corresponding to the input symbol, we say it ***branches***.

If NFA gets to a state where there is no valid transition, then that branch ***dies***.

NFA Acceptance

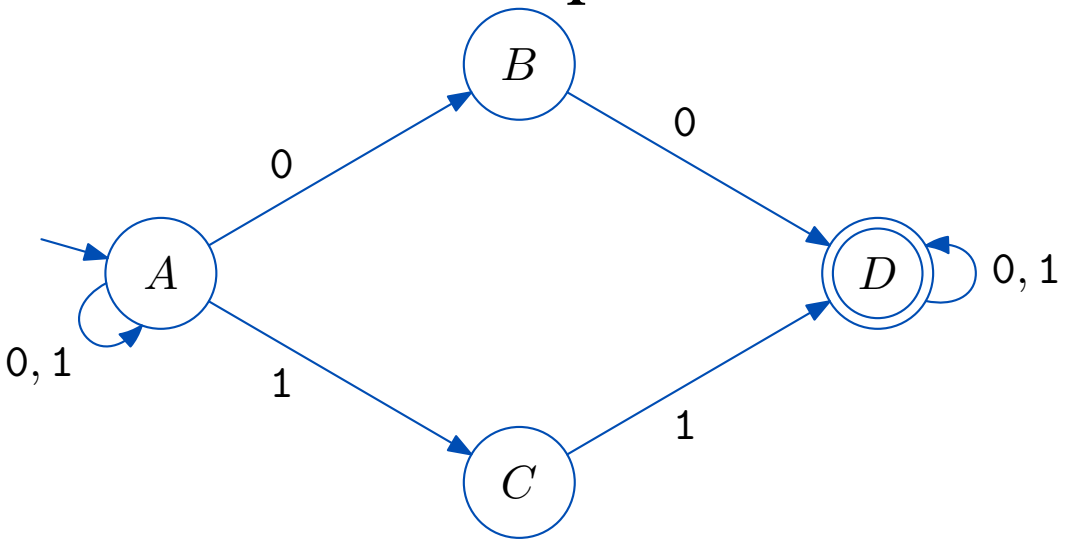
An NFA ***accepts*** the input string if ***there exists*** some choice of transitions that leads to ending in an accept state.

Thus, one accepting branch is enough for the overall NFA to accept, but every branch must reject for the overall NFA to reject.

This is a ***model*** of computation. We write ***DFA*** to specify a deterministic finite automaton (the one defined earlier). If type doesn't matter, we now just write ***FA***.

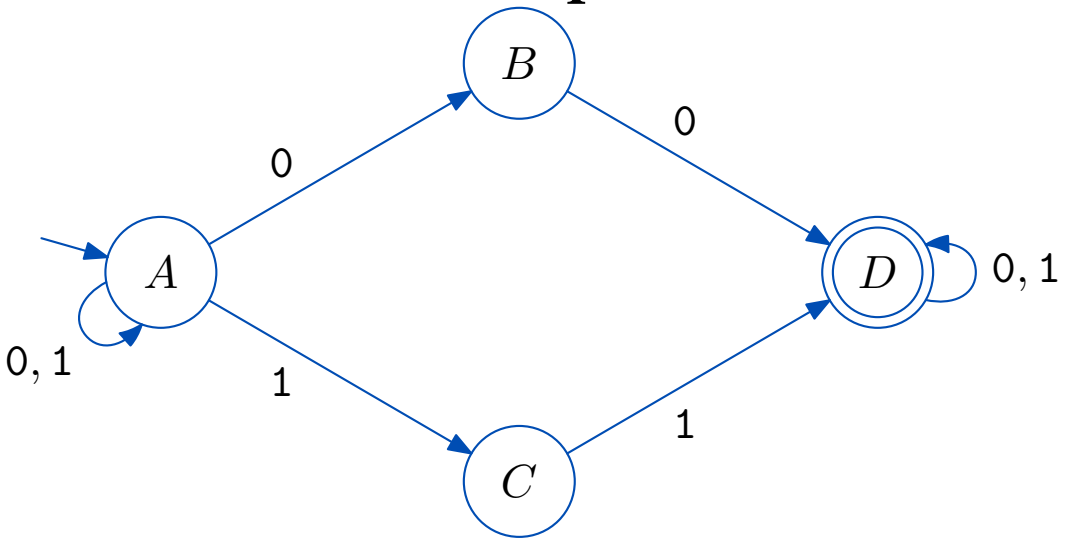
Example

What does this NFA accept?



Example: Doubles

What does this NFA accept?



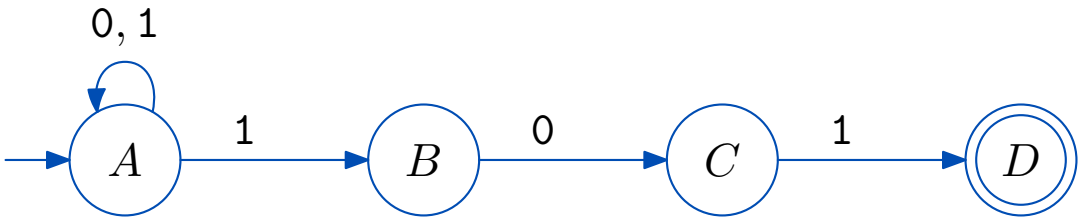
It accepts any binary string that contains `00` or `11` as a substring.

Example: Ending of Strings

An NFA that accepts all binary strings that end with `101`.

Example: Ending of Strings

An NFA that accepts all binary strings that end with `101`.

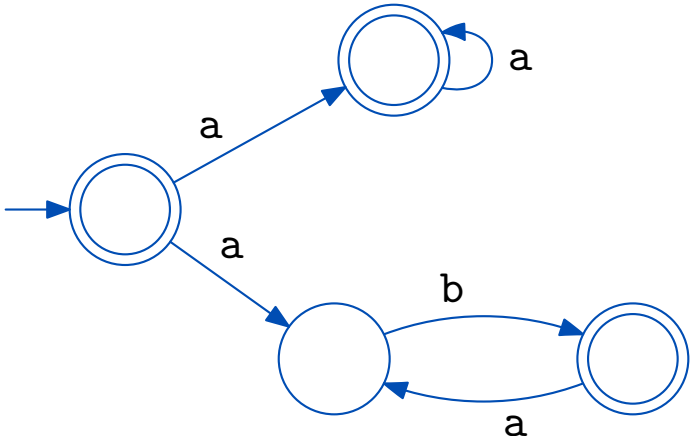


Example: Simultaneous Patterns

An NFA for $\mathtt{a}^* + (\mathtt{ab})^*$

Example: Simultaneous Patterns

An NFA for $\mathtt{a}^* + (\mathtt{ab})^*$

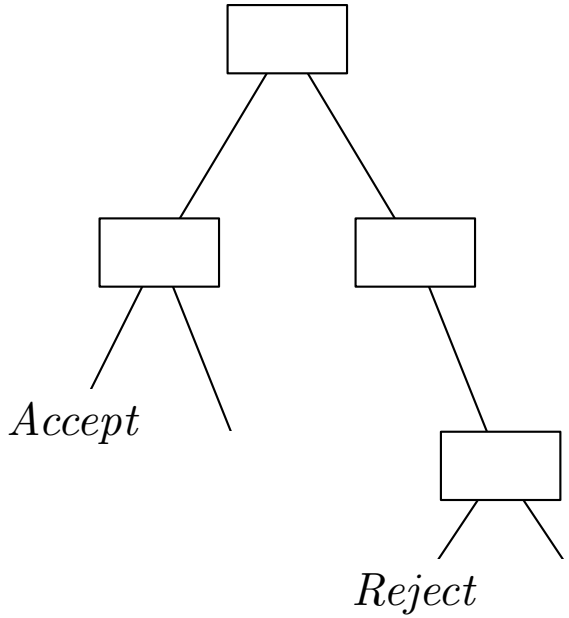


Nondeterminism as "Guess and Verify"

There are many ways to view nondeterminism. One way is the "***guess and verify***" idea: We assume the NFA is clairvoyant and always guesses correctly the next state to go to. However, the NFA must "check" its guesses.

Nondeterminism via Computation Tree

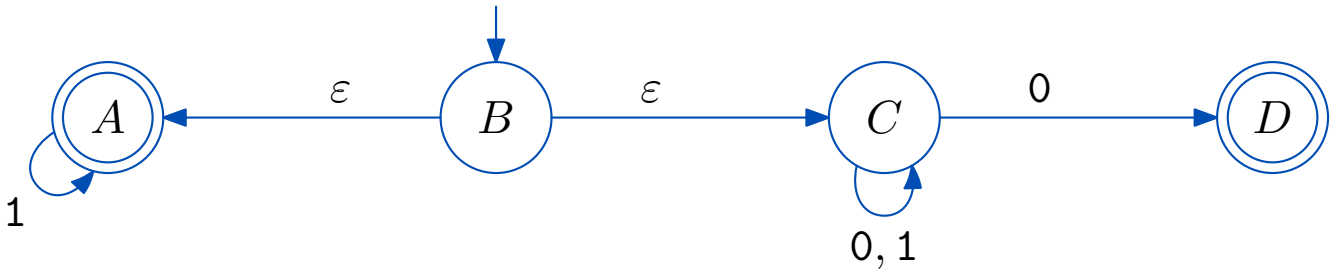
We can think of nondeterminism as a tree growing downwards, with the children of a node its possible successors. (The input is accepted exactly when at least one of the branches ends in an accept state.)



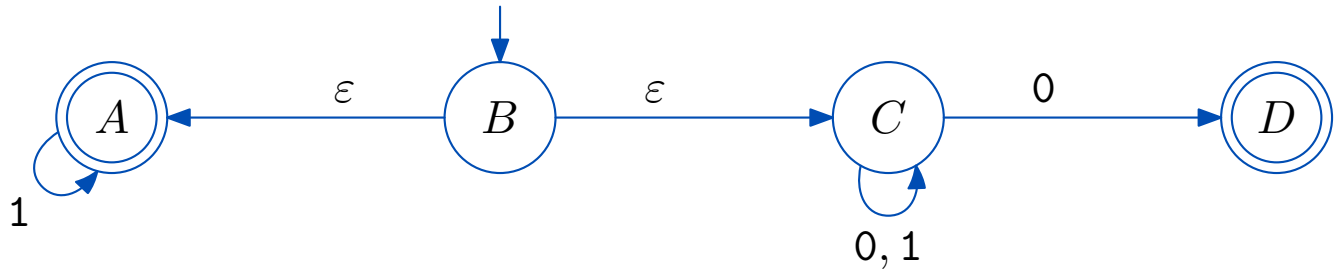
More Nondeterminism: ε-transitions

We also allow ε-transitions: arrows labeled with the empty string. These allow the NFA to change state without consuming an input symbol.

Example



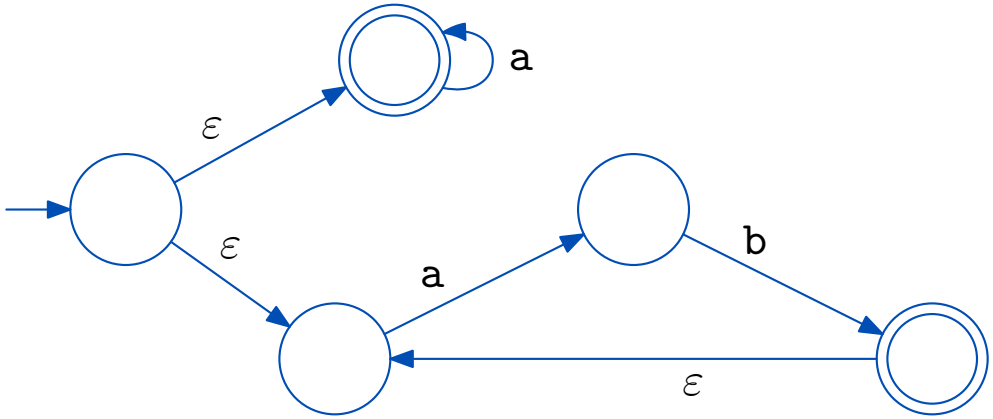
Example



Accepts all binary strings where the last symbol is `0` or that contain only `1`'s.

Another Example

Here is another NFA for $a^* + (ab)^*$:



Formal Definition

Formally, an NFA is a 5-tuple $(Q, \Sigma, q_0, T, \delta)$ where as before:

- Q is finite set of states;
- Σ is alphabet of input symbols;
- q_0 is start state;
- T is subset of Q giving the accept states;

and

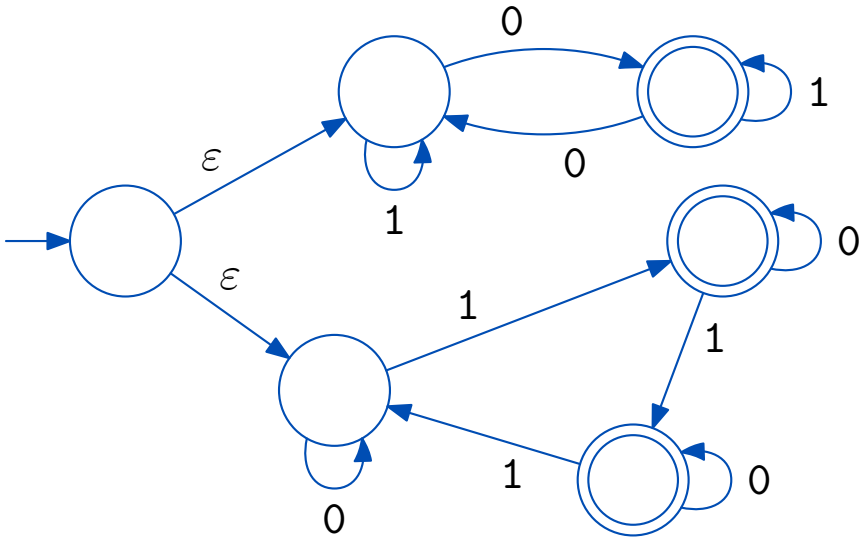
- δ is the transition function.

Now the transition function specifies a set of states rather than a state: it maps $Q \times \Sigma$ to $\{$ subsets of Q $\}$.

Practice

Give an NFA for the set of all binary strings that have either the number of `0`'s odd, or the number of `1`'s not a multiple of 3, or both.

Solution to Practice



Summary

A nondeterministic finite automaton (NFA) can have zero, one, or multiple transitions corresponding to a particular symbol. It is defined to accept the input if there exists some choice of transitions that cause the machine to end up in an accept state. Nondeterminism can also be viewed as a tree, or as a "guess-and-verify" concept. You can also have ε-transitions, where the NFA can change state without consuming an input symbol.