

## Lecture 4: Non-Determinism

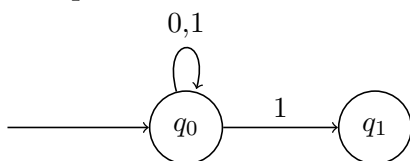
*Lecturer: Zvi Galil**Author: Austin Peng***Contents**

<b>1</b>	<b>Non-Determinism</b>	<b>2</b>
<b>2</b>	<b>Unary Language</b>	<b>4</b>
<b>3</b>	<b>Nondeterministic Finite Automaton</b>	<b>5</b>
3.1	Formal Definition Of A Nondeterministic Finite Automaton . . . . .	5
3.2	Equivalence Of NFAs and DFAs . . . . .	5

# 1 Non-Determinism

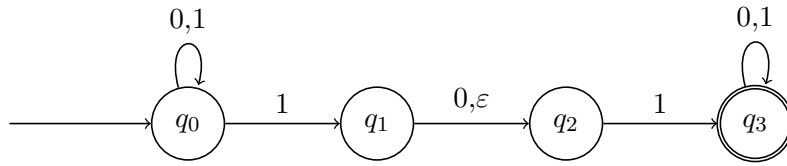
- Non-determinism looks strange and impractical, and in some sense it is, but it is very important.
- You will find out at the end of the course why the most important problem in computer science involves non-determinism and non-deterministic computation.
- Our definition of finite automaton so far is deterministic. This means, when we are at some state, and we see a symbol, we go to exactly one other state. Also, from every state, the transition is well defined for every symbol in the alphabet.
- 2 main differences between a non-deterministic finite automaton (NFA) and deterministic finite automaton (DFA).
  - from a state, when we see a symbol, we can go to 0 or more states (in a DFA, when we see a symbol, we go to exactly 1 state)
  - we have " $\epsilon$ "-transitions, which are transitions that we can take without seeing any symbol.
- The starting, accepting, and states all work the same. The big difference is in  $\delta$  (transition function).

**Example 1.** Consider the following NFA:



- If we are at  $q_0$  and see a 1, we do not have to go to  $q_1$ .
- Rather, think of it like we are in many states at once. (referred to as "guessing")
- In other words, we say that from  $q_0$ , we can go to several states.
- The automaton is a genius and knows where to go. It has the foresight to think, "I see 1 several times, I stay at  $q_0$ , but I will transition to  $q_1$  at just the right time."
- The choice of the 1 to transition is because the automaton can see the future.
- Note: this is abstract and not practical

**Example 2.** Consider the following NFA:



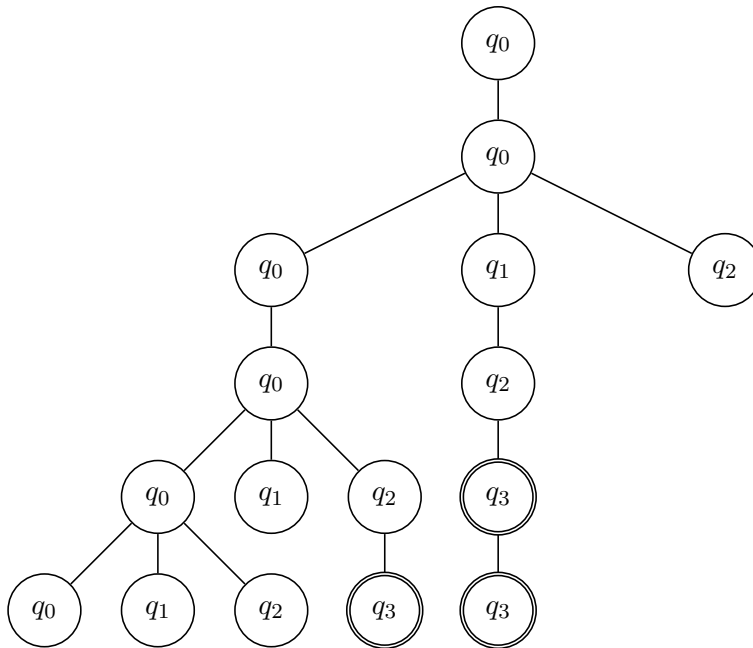
Suppose at  $q_1$  and we see a 1. There is no transition with 1, so we "abort". At  $q_1$  we see  $\varepsilon$  and move to  $q_2$ . Note: we always see  $\varepsilon$ , so without doing anything at  $q_1$ , we "slide" to  $q_2$ .

Below is a transition table describing the NFA.

$Q \backslash \text{symbol}$	0	1	$\varepsilon$
$\rightarrow q_0$	$q_0$	$q_0, q_1$	-
$q_1$	$q_2$	-	$q_2$
$q_2$	-	$q_3$	-
$q_3$	$q_3$	$q_3$	-

Think of split timelines every time we have to go to multiple states. From  $q_0$  seeing 1, we can go back to  $q_0$  or go to  $q_1$ . Note that we accept the input if at least one of the timelines ends at an accepting state.

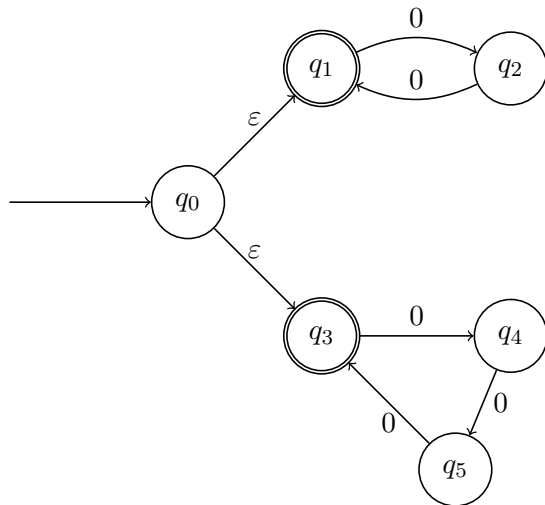
Below is a decision parse tree for input  $x = 01011$



The nodes at each layer represent the states we are in at that time (given the input). At  $q_0$ , we see 0 and can only go back to  $q_0$ . At  $q_1$ , we see 1 and can go to  $q_0$ ,  $q_1$ , or slide with  $\varepsilon$  to  $q_2$ . And repeat for the remaining binary values in the input string  $x$ . Once we finish, as long as one state in the final depth is accepting, the string  $x$  is accepted. This is the reason why we call this "guessing".

## 2 Unary Language

**Example 3.**  $M_1$



This NFA guesses if the input string length is divisible by either 2 or 3 by guessing to go higher  $q_1$  or lower  $q_3$ .

### 3 Nondeterministic Finite Automaton

#### 3.1 Formal Definition Of A Nondeterministic Finite Automaton

**Definition 1.** A **nondeterministic finite automaton** is a 5-tuple  $Q, \Sigma, \delta, q_0, F$  where:

- $Q$  is a finite set of states
- $\Sigma$  is a finite alphabet
- $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$  is the transition function, where  $\mathcal{P}(Q)$  is the power set of  $Q$
- $q_0 \in Q$  is the start state
- $F \subseteq Q$  is the set of accepting states

Note: the only change in the definition compared to a DFA is the transition function  $\delta$ .  $\delta$  is now a set of states.

We say  $M$  accepts  $w$  if  $w$  can be written as  $w = y_1 \dots y_n$  such that  $y_i \in \Sigma_\varepsilon$  and there are states  $r_0, r_1, \dots, r_m$  such that  $r_0 = q_0, r_m \in F$ , and  $r_{i+1} \in \delta(r_i, y_{i+1})$  for  $0 \leq i \leq m-1$ .

#### 3.2 Equivalence Of NFAs and DFAs

**Corollary 1.** A language is regular if and only if some nondeterministic finite automaton recognizes it.

**Theorem 1.** Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

*Proof Idea.* As a **corollary**, the set of languages NFAs can accept ( $N$ ) is equal to the set of languages that DFAs can accept ( $D$ ). The first part of the proof is to note that every DFA is an NFA, so  $D \subseteq N$ . Then to show that  $N \subseteq D$ , we say for each NFA  $N_i \in N$  we can construct an equivalent DFA, implying that  $N_i \in D$ . Together this implies that  $N = D$ .

Note: in terms of power of these two computing structures, this proof shows that there is no difference. But in terms of size and simplicity, NFAs have the advantage (since an equivalent DFA must have at least  $2^k$  states).

**Theorem 2.** Let  $L_k$  be the regular language over  $\{0, 1\}$  which contains all strings which have a 1 as the  $k$ 'th character from the right end of the string. Any DFA that recognizes  $L_k$  must contain at least  $2^k$  states.

*Proof.* Suppose that  $D$  is a DFA for  $L_k$  containing strictly fewer than  $2^k$  states. Then, by the pigeonhole principle, there must be a state  $q$  such that two different binary strings of length  $k$ ,  $x$ , and  $y$ , both cause the machine to end in state  $q$ . But if  $x$  and  $y$  are different, there is at least one index  $i$  such that  $x[i] = 0$  and  $y[i] = 1$  or vice versa. But then  $x0^{i-1}$  and  $y0^{i-1}$  cause the machine to end in the same state, yet one must be accepted and the other must be rejected. This is a contradiction, so our assumption that there was a DFA for  $L_k$  with strictly fewer than  $2^k$  states is incorrect.  $\square$