1 Introducing Nondeterminism

1.1 Informal Overview

Nondeterminism
Michael Rabin and Dana Scott (1959)

Figure 1: Michael Rabin

Figure 2: Dana Scott

Nondeterminism
Given a current state of the machine and input symbol to be read, the next state is not uniquely determined.

Comparison to DFAs

Nondeterministic Finite Automata (NFA)
NFAs have 3 features when compared with DFAs.

1. Ability to take a step without reading any input symbol
2. A state may have no transition on a particular symbol
3. Ability to transition to more than one state on a given symbol

---

ϵ-Transitions
Transitions without reading input symbols

Example 1. The British spelling of "color" is "colour". In a web search application, you may want to recognize both variants.

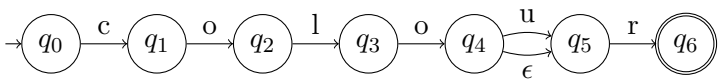


Figure 3: NFA with ϵ-transitions

No transitions

Example 2.

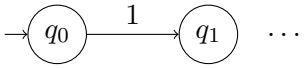


Figure 4: No 0-transition out of initial state

In the above automaton, if the string starts with a 0 then the string has no computation (i.e., rejected).

Multiple Transitions

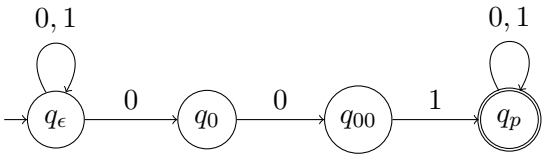


Figure 5: q_ϵ has two 0-transitions

1.2 Nondeterministic Computation

Parallel Computation View

At each step, the machine "forks" a thread corresponding to one of the possible next states.

- If a state has an ϵ-transition, then you fork a new process for each of the possible ϵ-transitions, without reading any input symbol
- If the state has multiple transitions on the current input symbol read, then fork a process for each possibility
- If from current state of a thread, there is no transition on the current input symbol then the thread dies

Parallel Computation View: An Example

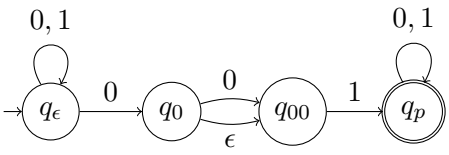


Figure 6: Example NFA

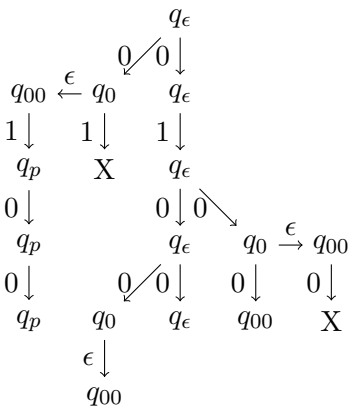
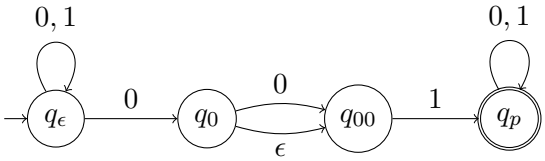


Figure 7: Computation on 0100

Nondeterministic Acceptance
Parallel Computation View

Input is *accepted* if after reading all the symbols, one of the live threads of the automaton is in a final/accepting state. If none of the live threads are in a final/accepting state, the input is *rejected*.



0100 is accepted because one thread of computation is $q_\epsilon \xrightarrow{0} q_0 \xrightarrow{\epsilon} q_{00} \xrightarrow{1} q_p \xrightarrow{0} q_p \xrightarrow{0} q_p$

Computation: Guessing View

The machine magically guesses the choices that lead to acceptance

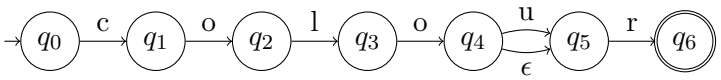


Figure 8: NFA M_{color}

After seeing "colo" the automaton guesses if it will see the british or the american spelling. If it guesses american then it moves without reading the next input symbol.

Observations: Guessing View

- If there is a sequence of choices that will lead to the automaton (not "dying" and) ending up in an accept state, then those choices will be magically guessed
- On the other hand, if the input will not be accepted then no guess will lead the to automaton being in an accept state
  - On the input "colobr", whether automaton M_{color} guesses british or american, it will not proceed when it reads 'b'.

---

2 Formal Definitions

2.1 NFAs

Nondeterministic Finite Automata (NFA)
Formal Definition

Definition 3. A nondeterministic finite automaton (NFA) is $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is the finite set of states
- Σ is the finite alphabet
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the powerset of Q
- $q_0 \in Q$ initial state
- $F \subseteq Q$ final/accepting states

Example of NFA

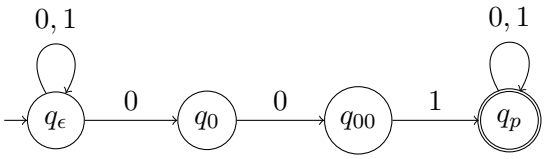


Figure 9: Transition Diagram of NFA

Formally, the NFA is $M_{001} = (\{q_\epsilon, q_0, q_{00}, q_p\}, \{0,1\}, \delta, q_\epsilon, \{q_p\})$ where δ is given by

$$\begin{array}{lll} \delta(q_\epsilon, 0) = \{q_\epsilon, q_0\} & \delta(q_\epsilon, 1) = \{q_\epsilon\} & \delta(q_0, 0) = \{q_{00}\} \\ \delta(q_{00}, 1) = \{q_p\} & \delta(q_p, 0) = \{q_p\} & \delta(q_p, 1) = \{q_p\} \end{array}$$

δ is $\emptyset$ in all other cases.

2.2 Nondeterministic Computation

Computation

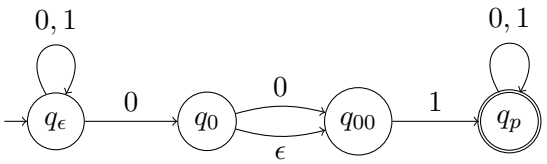
Definition 4. For an NFA $M = (Q, \Sigma, \delta, q_0, F)$, string w, and states $q_1, q_2 \in Q$, we say $q_1 \xrightarrow{w}_M q_2$ if there is one thread of computation on input w from state q_1 that ends in q_2. Formally, $q_1 \xrightarrow{w}_M q_2$ if there is a sequence of states $r_0, r_1, \ldots r_k$ and a sequence $x_1, x_2, \ldots x_k$, where for each i, $x_i \in \Sigma \cup \{\epsilon\}$, such that

- $r_0 = q_1$,
- for each i, $r_{i+1} \in \delta(r_i, x_{i+1})$,
- $r_k = q_2$, and
- $w = x_1 x_2 x_3 \cdots x_k$

Differences with definition for DFA

- Since δ gives a set of states, for each i, r_{i+1} is required to be in $\delta(r_1, x_{i+1})$, and not *equal* to it (as is the case for DFAs)
- Allowing/inserting ϵ in to the input sequence

Example Computation



$q_\epsilon \xrightarrow{0100}_M q_p$ because taking $r_0 = q_\epsilon$, $r_1 = q_0$, $r_2 = q_{00}$, $r_3 = q_p$, $r_4 = q_p$, $r_5 = q_p$, and $x_1 = 0$, $x_2 = \epsilon$, $x_3 = 1$, $x_4 = 0$, $x_5 = 0$, we have

- $x_1 x_2 \cdots x_5 = 0\epsilon 100 = 0100$
- $r_{i+1} \in \delta(r_i, x_{i+1})$

Acceptance/Recognition

Definition 5. For an NFA $M = (Q, \Sigma, \delta, q_0, F)$ and string $w \in \Sigma^*$, we say M *accepts* w iff $q_0 \xrightarrow{w}_M q$ for some $q \in F$.

Definition 6. The *language accepted or recognized* by NFA M over alphabet Σ is $\mathbf{L}(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. A language L is said to be *accepted/recognized* by M if $L = \mathbf{L}(M)$.

Useful Notation

Definition 7. For an NFA $M = (Q, \Sigma, \delta, q_0, F)$, string w, and state $q \in Q$, we say $\hat{\delta}_M(q, w)$ to denote states of all the active threads of computation on input w from q. Formally,

$$\hat{\delta}_M(q, w) = \{q' \in Q \mid q \xrightarrow{w}_M q'\}$$

We could say M accepts w iff $\hat{\delta}_M(q_0, w) \cap F \neq \emptyset$.

Observation 1
For NFA M, string w and state q_1 it could be that

- $\hat{\delta}_M(q_1, w) = \emptyset$
- $\hat{\delta}_M(q_1, w)$ has more than one element

Observation 2
However, the following proposition about DFAs continues to hold for NFAs

> For NFA M, strings u and v, and states q_1, q_2, $q_1 \xrightarrow{uv}_M q_2$ iff there is a state q such that $q_1 \xrightarrow{u}_M q$ and $q \xrightarrow{v}_M q_2$

Example

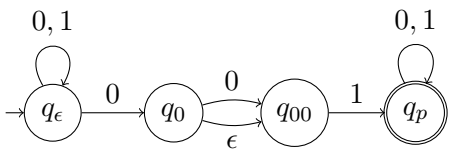


Figure 10: Example NFA

$\hat{\delta}_M(q_\epsilon, 0100) = \{q_p, q_{00}, q_\epsilon\}$

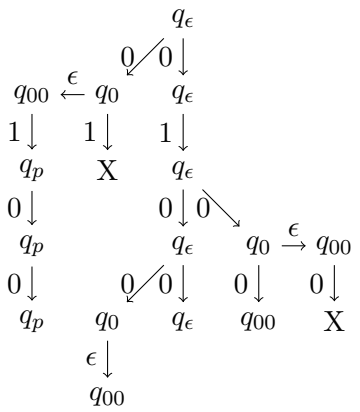


Figure 11: Computation on 0100

2.3 Examples

Example I

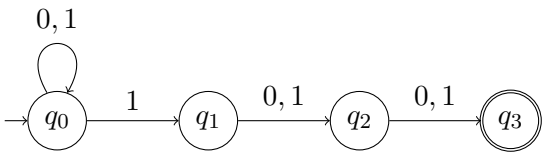


Figure 12: Automaton accepts strings having a 1 three positions from end of input

The automaton "guesses" at some point that the 1 it is seeing is 3 positions from end of input.

Example II

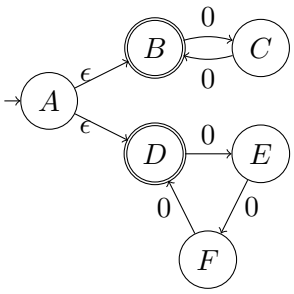


Figure 13: NFA accepting strings where the length is either a multiple 2 or 3

The NFA "guesses" at the begining whether it will see a multiple of 2 or 3, and then confirms that the guess was correct.

Example III

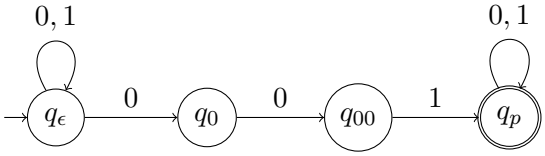


Figure 14: NFA accepting strings with 001 as substring

At some point the NFA "guesses" that the pattern 001 is starting and then checks to confirm the guess.

---

3 Power of Nondeterminism

3.1 Overview

Using Nondeterminism

When designing an NFA for a language

- You follow the same methodology as for DFAs, like identifying what needs to be remembered
- But now, the machine can "guess" at certain steps

3.2 Examples

Back to the Future

Problem
For $\Sigma = \{0, 1, 2\}$, let

$$L = \{w\#c \mid w \in \Sigma^*,\ c \in \Sigma,\ \text{and } c \text{ occurs in } w\}$$

So $1011\#0 \in L$ but $1011\#2 \notin L$. Design an NFA recognizing L.

Solution

- Read symbols of w, i.e., portion of input before # is seen
- Guess at some point that current symbol in w is going to be the same as 'c'; store this symbol in the state
- Read the rest of w
- On reading #, check that the symbol immediately after is the one stored, and that the input ends immediately after that.

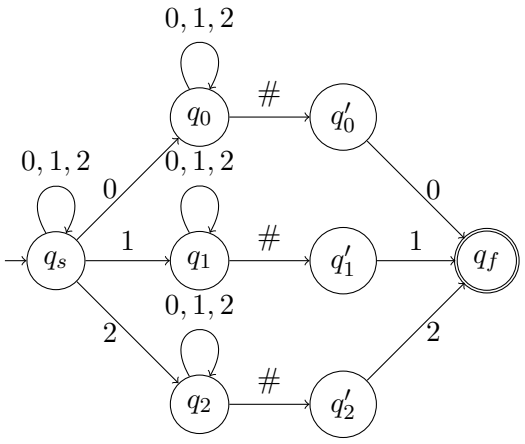


Figure 15: $L(M) = \{w\#c \mid c \text{ occurs in } w\}$

Pattern Recognition

Problem
For alphabet Σ and $u \in \Sigma^*$, let

$$L_u = \{w \in \Sigma^* \mid \exists v_1, v_2 \in \Sigma^*.\ w = v_1 u v_2\}$$

That is, L_u is all strings that have u as a substring.

Solution

- Read symbols of w
- Guess at some point that the string u is going to be seen
- Check that u is indeed read
- After reading u, read the rest of w

To do this, the automaton will remember in its state what prefix of u it has seen so far; the initial state will assume that it has not seen any of u, and the final state is one where all the symbols of u have been observed.

Formally, we can define this automaton as follows. Let $u = a_1 a_2 \cdots a_n$. The NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{\epsilon, a_1, a_2 a_2, a_1 a_2 a_3, \ldots, a_1 a_2 \cdots a_n = u\}$. Thus, every prefix of u is a state of NFA M.
- $q_0 = \epsilon$,
- $F = \{u\}$,
- And δ is given as follows

$$\delta(q, a) = \begin{cases} \{\epsilon\} & \text{if } q = \epsilon,\ a \neq a_1 \\ \{\epsilon, a_1\} & \text{if } q = \epsilon,\ a = a_1 \\ \{a_1 a_2 \cdots a_{i+1}\} & \text{if } q = a_1 \cdots a_i\ (1 \leq i < n),\ a = a_{i+1} \\ \{u\} & \text{if } q = u \\ \emptyset & \text{otherwise} \end{cases}$$

See Example III above for a concrete case.

---

1 k-positions from the end

Problem
For alphabet $\Sigma = \{0, 1\}$,

$$L_k = \{w \in \Sigma^* \mid \exists v_1, v_2 \in \Sigma^*.\ w = v_1 1 v_2 \text{ and } |v_2| = k - 1\}$$

That is, L_k is all strings that have a 1 k positions from the end.

Solution

- Read symbols of w
- Guess at some point that there are only going to be k more symbols in the input
- Check that the first symbol after this point is a 1, and that we see $k - 1$ symbols after that
- Halt and accept no more input symbols

The states need to remember that how far we are from the end of the input; either very far (initial state), or less that k symbols from end.

Formally, $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = \{q_i \mid 0 \leq i \leq k\}$. The subscript of the state counts how far we are from the end of the input; q_0 means that there can be many symbols left before the end, and q_i $(i > 1)$ means there are $k - i$ symbols left to read.
- $q_0 = q_0$
- $F = \{q_k\}$,
- And δ is given as follows

$$\delta(q, a) = \begin{cases} \{q_0\} & \text{if } q = q_0,\ a = 0 \\ \{q_0, q_1\} & \text{if } q = q_0,\ a = 1 \\ \{q_{i+1}\} & \text{if } q = q_i (1 \leq i < k) \\ \emptyset & \text{otherwise} \end{cases}$$

See Example I above for a concrete case.

Observe that this automaton has only $k + 1$ states, whereas we proved in lecture 3 that any DFA recognizing this language must have size at least 2^k. Thus, NFAs can be exponentially smaller than DFAs.

Proposition 8. *There is a family of languages L_k (for $k \in \mathbb{N}$) such that the smallest DFA recognizing L_k has at least 2^k states, whereas there is an NFA with only $O(k)$ recognizing L_k.*

Proof. Follows from the observations above. □

Halving a Language

Definition 9. For a language L, define $\frac{1}{2}L$ as follows.

$$\frac{1}{2}L = \{x \mid \exists y.\ |x| = |y| \text{ and } xy \in L\}$$

In other words, $\frac{1}{2}L$ consists of the first halves of strings in L

Example 10. If $L = \{001, 0000, 01, 110010\}$ then $\frac{1}{2}L = \{00, 0, 110\}$.

Recognizing Halves of Regular Languages

Proposition 11. *If L is recognized by a DFA M then there is a NFA N such that $L(N) = \frac{1}{2}L$.*

Proof Idea

On input x, need to check if x is the first half of some string $w = xy$ that is accepted by M.

- "Run" M on input x; let M be in state q_i after reading all of x
- *Guess a string y such that $|y| = |x|$*
- Check if M reaches a final state on reading y from q_i

How do you guess a string y of equal length to x using finite memory? Seems to require remembering the length of x!

Fixing the Idea

Problem and Fix(?)

- How do you guess a string y of equal length to x using finite memory? Guess one symbol of y as you read one symbol of x!
- How do you "run" M on y from q_i, if you cannot store all the symbols of y? Run M on y as you guess each symbol, without waiting to finish the execution on x!
- If we don't first execute M on x, how do we know the state q_i from which we have to execute y from? Guess it! And then check that running M on x does indeed end in q_i, your guessed state.

New Algorithm

On input x, NFA N

1. Guess state q_i and place "left finger" on (initial state of M) q_0 and "right finger" on q_i
2. As characters of x are read, N moves the left finger along transitions dictated by x and *simultaneously* moves the right finger along nondeterministically chosen transitions labelled by some symbol
3. Accept if after reading x, left finger is at q_i (state initially guessed for right finger) *and* right finger is at an accepting state

Things to remember: initial guess for right finger, and positions of left and right finger.

Algorithm on Example

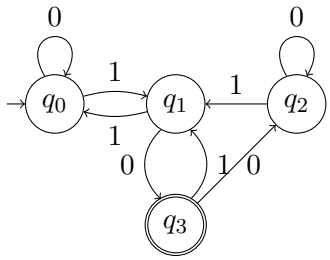


Figure 16: DFA M

$100010 \in L$ and so $x = 100 \in \frac{1}{2}L$
NFA N execution on $x = 100$ is

String Read	Left Finger		Right Finger
ϵ	q_0	↱	$\boxed{q_2}$
1	q_1		q_2
10	q_3	=?	q_1
100	$\boxed{q_2}$	↲	$\boxed{q_3}$
			↑ accept?

Formal Construction of NFA N
States and Initial State

Given $M = (Q, \Sigma, \delta, q_0, F)$ recognizing L define $N = (Q', \Sigma, \delta', q_0', F')$ that recognizes $\frac{1}{2}L$

- $Q' = Q \times Q \times Q \cup \{s\}$, where $s \notin Q$
  - s is a new start state
  - Other states are of the form ⟨left finger, initial guess, right finger⟩; "initial guess" records the initial guess for the right finger
- $q_0' = s$
- Transitions

  $\delta'(s, \epsilon) = \{\langle q_0, q_i, q_i\rangle \mid q_i \in Q\}$
  "Guess" the state q_i that the input will lead to
  $\delta'(\langle q_i, q_j, q_k\rangle, a) = \{\langle q_l, q_j, q_m\rangle \mid \delta(q_i, a) = q_l,$
  $\exists b \in \Sigma.\ \delta(q_k, b) = q_m\}$
  b is the guess for the next symbol of y and initial guess does not change
- $F' = \{\langle q_i, q_i, q_j\rangle \mid q_i \in Q, q_j \in F\}$
