CS112: Theory of Computation (LFA)

Lecture 3: Nondeterminism

Dumitru Bogdan

Faculty of Computer Science
University of Bucharest

March 8, 2023

Table of contents

Section 1

Previously on CS112

DFA

Definition

A finite automaton is 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. $\delta : Q \times \Sigma \to Q$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

DFA Computation

Now we formalise finite automaton's computation as follows: Let $M = (Q, \Sigma, \delta, q_o, F)$ be a finite automaton and let $w = w_1 w_2 \dots w_n$ be a string where each w_i is a member of Σ.

Definition

Then M **accepts** w if a sequence of states $r_0, r_1, \dots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \dots, n-1$
3. $r_n \in F$

Regular Language

Definition

A language is called a regular language if some finite automaton recognizes it.

Regular operations

Definition

Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star** as follows:

- Union: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- Concatenation: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
- Star: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$

empty string

empty string ϵ is always a member of A^*, no matter what A is.

Closure under union

Theorem

The class of regular languages is closed under the union operation, meaning that if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Closure under union

Proof idea:

- Because A_1 and A_2 are regular, we know that some finite automaton M_1 recognizes A_1 and some finite automaton M_2 recognizes A_2
- To prove $A_1 \cup A_2$ is regular we need a finite automaton called M that recognize $A_1 \cup A_2$. This is a proof by **construction**
- This FA M must accept an input string if either M_1 or M_2 accepts it. So we simulate somehow M_1 and M_2
- Cannot be done in sequential order because once a symbol has been read then it is gone
- So we simulate M_1 and M_2 simultaneously by remembering the pair of states
- If size (i.e., number of states) of M_1 is k_1 and size of M_2 is k_2 then we have $k_1 \times k_2$ pairs

Section 2

Context setup

Context setup

Corresponding to Sipser 1.2

Generalization of determinism

- So far in our discussion, every step of a computation follows in a unique way from the preceding step
- When the machine is in a given state and reads the next input symbol, we know what the next state will be
- We call this a **deterministic** computation
- In a more general approach, a **nondeterministic** machine has several choices for the next state at any point
- Since it is a generalization it means that every deterministic finite automaton (**DFA**) is automatically a nondeterministic finite automaton (**NFA**)

Section 3

Nondeterminism

Generalization of determinism

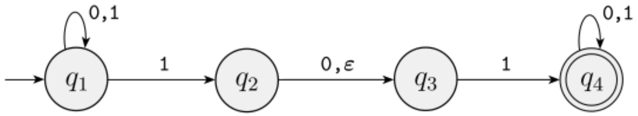

Figure: The nondeterministic finite automaton N_1

Differences between NFA and DFA

- While a DFA always has exactly one exiting transition arrow, an NFA may have zero, one, or many exiting arrows for each alphabet symbol
- in a DFA, labels on the transition arrows are symbols from the alphabet, while an NFA can have the ϵ label.

How does an NFA compute

- When reading a symbol and there is only one way to proceed we have the DFA situation
- When reading a symbol and there are many ways to proceed (multiple arrows with the same symbol) the machine splits into multiple copies of itself and follows all the possibilities in parallel.
- If a state with an ϵ symbol on an exiting arrow is encountered, without reading any input, the machine splits into multiple copies (following each ϵ labeled arrow)

Intuition 1

Nondeterminism may be viewed as a kind of **parallel computation** wherein multiple independent “processes” or “threads” can be running concurrently.

When the NFA splits to follow several choices, that corresponds to a process “forking” into several children, each proceeding separately. If at least one of these processes accepts, then the entire computation accepts.

Intuition 2

Another way to think of a nondeterministic computation is as a **tree of possibilities**.

The root of the tree corresponds to the start of the computation. Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices. The machine accepts if at least one of the computation branches ends in an accept state

Intuition 2

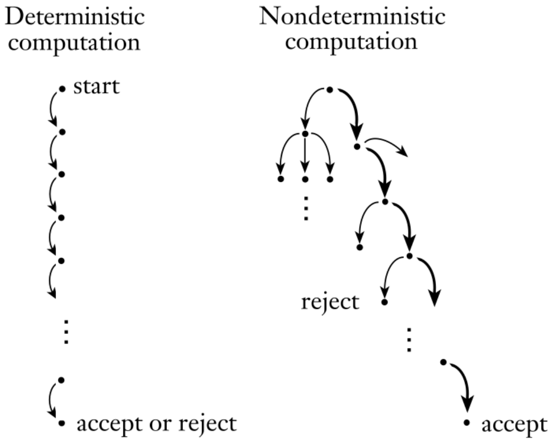

Figure: Deterministic and nondeterministic computations with an accepting branch

Why bother with NFAs?

NFA are useful in several respects:

- constructing NFAs is sometimes easier than directly constructing DFA because they are much smaller
- Every NFA can be converted into an equivalent DFA
- NFAs is a good introduction to nondeterminism in more powerful computational models because they are easy to understand

Example

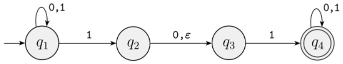

Figure: N_1 NFA

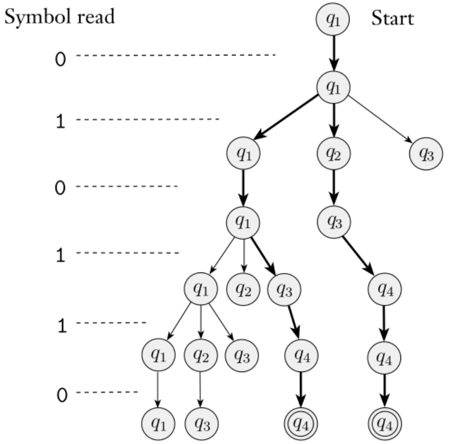

Figure: The computation of N_1 on input 010110

Example

Let A be the language consisting of all strings over $\{0, 1\}$ containing 1 in the third position from the end (e.g., 000100 is in A but 0011 is not)

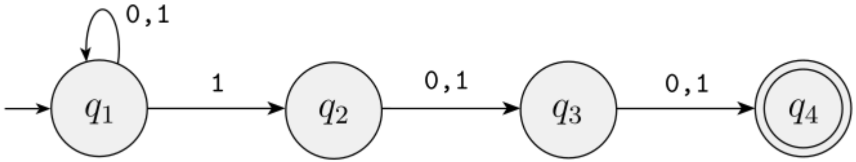

Figure: The NFA N_2 recognizing A

Example

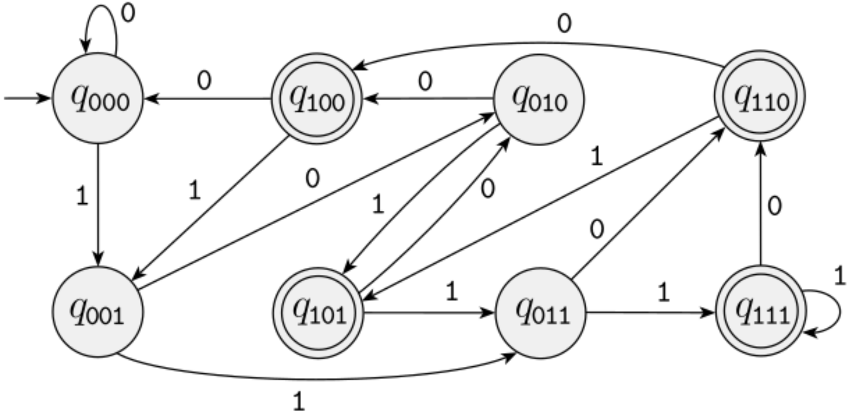

Figure: The DFA M_2 recognizing A

Example

Let A be the language consisting of all strings over $\{0\}$ having the form 0^k where k is a multiple of 2 or 3 (e.g., ϵ, 00, 000, 000000 but not 0 or 00000)

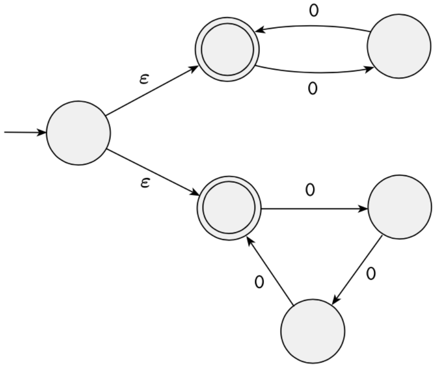

Figure: The NFA N_3 recognizing A

Example

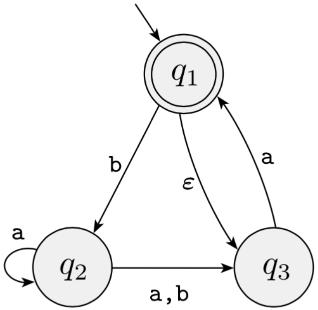

Figure: The NFA N_4

Does N_4 accepts ϵ, *a*, *baba*? Does it accepts *bb*?

Formal definition

- The formal definition of a nondeterministic finite automaton is similar to that of a deterministic finite automaton.
- However, transition function is the key difference

Formal definition

Definition

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

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accepted states

We denote $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$ and $\mathcal{P}(Q)$ as the power set of Q.

Example

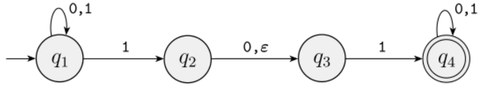

Figure: The NFA N_1

Formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$ where:

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. δ is given as

	0	1	ϵ
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_1 is the start state
5. $F = \{q_4\}$

Formal definition

Now we formalise NFA's computation as follows:

Definition

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA and w a string over alphabet Σ. Then we say that N **accepts** w if we can write w as $w = y_1 y_2 \dots y_m$ where each y_i is a member of Σ_ϵ and a sequence of states $r_0, r_1, \dots, r_m$ exists in Q with three conditions:

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

Observe that $\delta(r_i, y_{i+1})$ is a set of allowable next states.

Section 4

Equivalence of NFAs and DFAs

Equivalence of NFAs and DFAs

- NFAs appear to have more power than DFAs, so we might expect that NFAs recognize more languages.
- But deterministic and nondeterministic finite automata recognize the same class of languages
- This is important because describing an NFA for a given language sometimes is much easier than describing a DFA for that language

Definition

Two machines are equivalent if they recognize the same language

Equivalence of NFAs and DFAs

Theorem

Every NFA has an equivalent DFA.

Proof idea:

- The idea is to convert the NFA into an equivalent DFA that **simulates** the NFA.
- In the examples of NFAs we kept track of the various branches of the computation
- If k is the number of states of the NFA, it has 2^k subsets of states
- So the DFA simulating the NFA will have 2^k states
- Now we need to figure out which will be the start state and accept states of the DFA, and what will be its transition function

Equivalence of NFAs and DFAs I

Theorem

Every NFA has an equivalent DFA.

Equivalence of NFAs and DFAs II

Proof.

Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. First we do our construction on an easy case when N has no ϵ arrows.

1. $Q' = \mathcal{P}(Q)$. Ever state of M is a set o states of N.
2. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \textit{ for some } r \in R\}$. If R is a state of M, it is also a set of states of N. When M reads symbol a in state R it shows where a takes each state on R. Because each state can go to a set o states we take the union of all this sets. More concise:
$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$
3. $q_0 = \{q_0\}$. M starts in the state corresponding to the collection containing just the start state of N
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$. The machine M accepts if one of the possible states that N could be in at this point is an accept state.

Equivalence of NFAs and DFAs

Now we need to consider the ϵ arrows. For this we define:

$E(R) = \{q | q$ can be reached from R by traveling along 0 or more ϵ arrows$\}$

All we have to do now is to modify the transition function of M so that we can reach the states when also going along ϵ arrows:

$\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a))$ *for some* $r \in R\}$

Additionally, we need to modify the start state of M to visit all possible states that can be reached from the start state of N along the ϵ arrows. So $q'_0 = E(\{q_0\})$

Equivalence of NFAs and DFAs

Corollary

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

Proof.

Hint: $\Leftrightarrow$ type of proof. We use the above theorem and another one from the previous lecture $\square$

Example

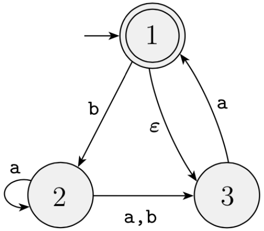

Figure: N_4 NFA

Let's convert the the following NFA $N_4 = (\{1,2,3\},\{a,b\},\delta,1,\{1\})$ to a DFA named M.

Example

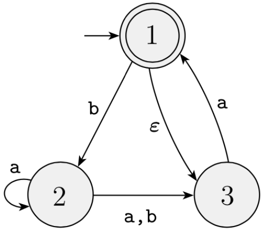

Figure: N_4 NFA

- First we need to determine M's states.
- Since N_4 has three states, $\{1,2,3\}$, we get eight states:

$$M = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

Example

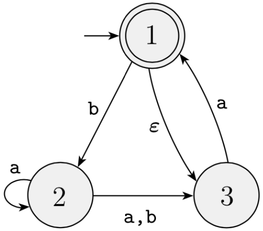

Figure: N_4 NFA

- Next, we determine the start and accept states of M.
- The start state is $E(\{1\})$, the set of states that are reachable from 1 by traveling along ϵ arrows, plus 1 itself. An ϵ arrow goes from 1 to 3, so $E(\{1\}) = \{1, 3\}$.
- The accept states are those containing N_4's accept state: $\{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$

Example

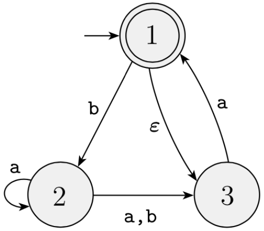

Figure: N_4 NFA

- At last, we determine M's transition function.
- In M state $\{2\}$ goes to $\{2, 3\}$ on input a. State $\{2\}$ goes on state $\{3\}$ on input b
- State $\{1\}$ goes on $\emptyset$ because no a arrows
- State $\{1, 2\}$ goes to $\{2, 3\}$ because 2 points to both 2 and 3 on M

Example

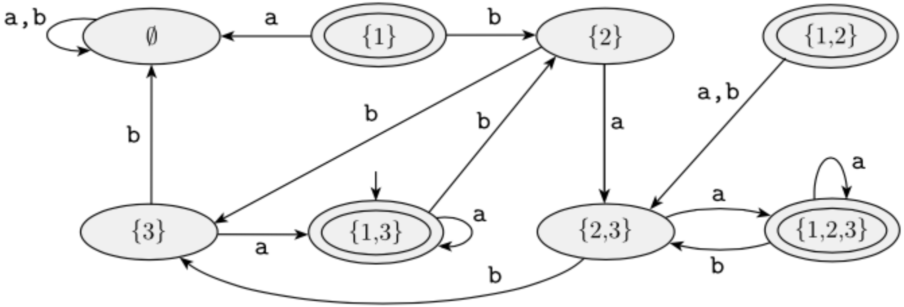

Figure: M DFA corresponding to NFA N_4

Example

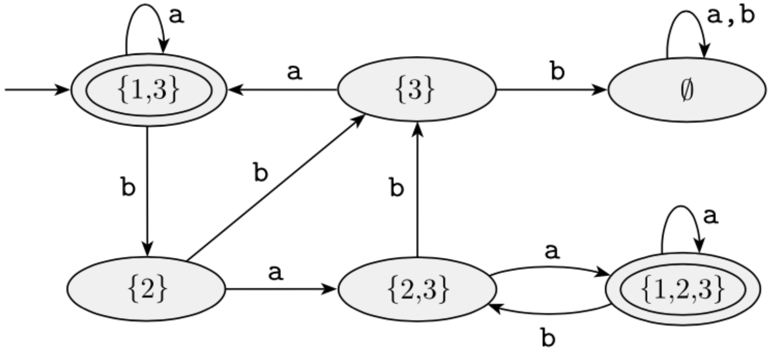

Figure: DFA M after removing unnecessary states