CSE 555 Theory of Computation Class 2 (1/17)

Goran Konjevod

Department of Computer Science and Engineering
Arizona State University

ASU, Spring 2008

Last time: Regular languages, FA

Finite automaton.
Regular language: accepted by a FA.
Computation: step by step, each step completely determined by the current state and input symbol.

Nondeterministic Computation

If A and B are both regular then $A \cup B$ is also.

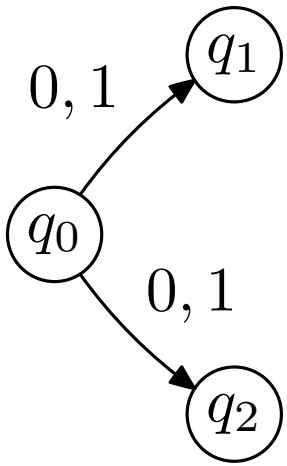
Proof: Given FA M_A (that accepts A) and M_B (that accepts B), construct $M_{A \cup B}$ that accepts $A \cup B$. Construction idea: simulate both machines at the same time, using states that correspond to pairs of a state of M_A and a state of M_B

"Simpler" Union?

Definition of set union: a string belongs to $A \cup B$ iff it belongs to A or to B.

Can a machine do that?

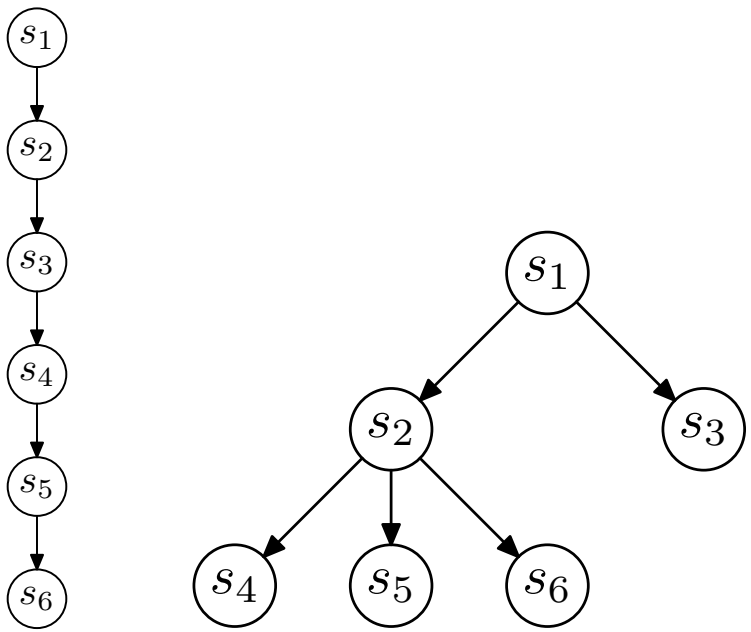
Machine that chooses between two possibilities



Not a FA according to our definitions...
$(Q, \Sigma, \delta, q_0, F)$, where $\delta : Q \times \Sigma$ is a function...
According to our definitions, given a state and a symbol, there is exactly one state to go to...

Solution: change the definition!

What if, given a state and a symbol, there can be more than one state to go to?
Nondeterminism: not necessarily a unique computation path.



(a) (b)

Nondeterministic Finite Automaton Definition

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

- Q is a finite set (of *states*).
- Σ is a finite set (*alphabet*, its elements denoted as *letters* or *symbols*).
- δ is a function $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ (the *transition function*), where $\mathcal{P}(Q)$ is the power set of Q (the set of all subsets of Q).
- q_0 is an element of Q (the *start state*).
- F is a subset of Q (the set of *accepting states*).

What does this mean?

Two problems:

1. When does an NFA accept? (For a DFA: if it ends up in an accepting state.)
2. How do we figure out if the NFA accepts a string? (For a DFA: just simulate it step by step and see.)

Nondeterministic Finite Automaton: remainder of definition

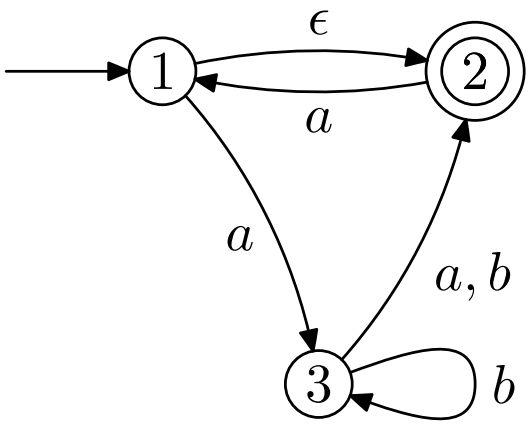
Input: a string over some given alphabet Σ.
Output: Yes/No.
Machine:

1. Reads input one symbol at a time.
2. At each moment, **is** in (any one of) a unique **set of states**.
3. Before reading any input, is in **the set that includes only the start state**.
4. If now is in a set that includes some state q and reads input letter a, then in the next moment, is in a set that includes all states that belong to $\delta(q, a)$.
5. Computation halts when input is exhausted.
6. If halted in a set that includes an **accepting state**, output is Yes, otherwise output is No.

DFA vs. NFA: unique states vs. sets of states

1. When does an NFA accept? Whenever there exists a path from the start state to an accepting state that can be followed while reading the input.
2. How do we figure out if the NFA accepts a string? Simulate it just like a DFA, but instead of the "current state", maintain the "set of current states". (Complexity of this is still linear in the length of the input, but with an extra factor of $|\Sigma|$.)

ϵ-transitions



Nondeterministic Finite Automaton: Complete Definition

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

- Q is a finite set (of *states*).
- Σ is a finite set (*alphabet*, its elements denoted as *letters* or *symbols*).
- δ is a function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ (the *transition function*), where $\mathcal{P}(Q)$ is the power set of Q (the set of all subsets of Q).
- q_0 is an element of Q (the *start state*).
- F is a subset of Q (the set of *accepting states*).

Example of usefulness of NFAs

Let $\Sigma = \{0, 1\}$.
$L = \{w \in \Sigma^* \mid$ the 8th symbol from the end of w is a $0\}$.
Describe a DFA for L.
We need $2^8 = 256$ states!
There is an NFA with only 9 states...
So, from an NFA with $n + 1$ states we may get a DFA with 2^n.
(The worst case.)

DFA vs. NFA: languages

Can NFA accept any languages that DFA cannot?
They do seem more powerful... but the state vs. set-of-states intuition says no..

Equivalence

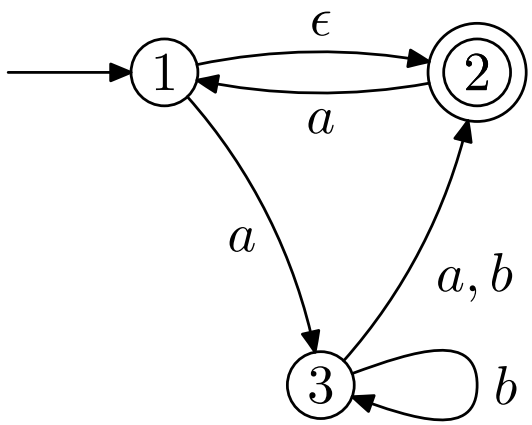
Theorem

Let M be an NFA. Then $L(M)$ is a regular language.

In other words, for every NFA, there exists a DFA that accepts exactly the same set of strings.

Proof: by construction

Example of NFA to DFA conversion



(Conversion done in full on the whiteboard.)

NFA to DFA conversion: symbolic description

From the NFA $N = (Q, \Sigma, \delta, q_0, F)$, we construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$.

1. $Q' = \mathcal{P}(Q)$. (Every state of M is a set of states of N.)
2. For $R \subseteq Q$, let $E(R)$ be the set of states of N that can be reached from states of R by following only ϵ-transitions.
3. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \bigcup_{q \in E(R)} E(\delta(q, a))$.
4. $q_0' = \{q_0\}$.
5. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$.

Regular operations revisited

Theorem

If A and B are regular languages, then

1. *so is $A \cup B$*
2. *so is $A \circ B$*
3. *so is A^**

Proof: easy using NFA—done on the whiteboard.

So far: DFA, NFA, regular languages

Finite automata: deterministic or nondeterministic.
Regular language: accepted by a FA.
DFA and NFA equivalent in terms of power.
NFA sometimes more efficient (compact).

Regular Expressions

Regular languages = languages accepted by FA.
(Computational definition.)
(Next:) Regular languages = languages described by RE (Syntactic definition.)

Basic regular expressions

a, for any $a \in \Sigma$.

ϵ

$\emptyset$

Regular operations

Union
Concatenation
Star

Definition of Regular Expression

R is a regular expression, if one of the following holds:

1. $R = a$ for some $a \in \Sigma$,
2. $R = \epsilon$,
3. $R = \emptyset$,
4. $R = (R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $R = (R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. $R = (R_1^*)$, where R_1 is a regular expression.

Parentheses may be omitted. Precedence order: star, concatenation, union. R represents the language $L(R)$.

Regular expression examples: identities

$R \cup \emptyset = R$.

$R \circ \epsilon = R$

Is $R \cup \epsilon = R$? No!

$R_1(R_2 \cup R_3) = R_1R_2 \cup R_1R_3$.

$(R_1R_2)^*R_1 = R_1(R_2R_1)^*$.

$(R_1 \cup R_2)^* = (R_1^* \cup R_2)^* = (R_1^*R_2)^*R_1^*$

Regular expressions describe exactly regular languages

Theorem

A language is regular if and only if it is described by some regular expression.

Proof.

Need to prove two parts:
1: If a language is described by some regular expression then it is regular.
2: If a language is regular then it is described by some regular expression. □

$L(R)$ is regular.

Proof:
Induction on the structure of the expression.
Basis: cases 1,2,3—verify directly that the language is regular.

Definition of Regular Expression

R is a regular expression, if one of the following holds:

1. $R = a$ for some $a \in \Sigma$,
2. $R = \epsilon$,
3. $R = \emptyset$,
4. $R = (R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $R = (R_1 \circ R_2)$, where R_1 and R_2 are regular expressions,
6. $R = (R_1^*)$, where R_1 is a regular expression.

Part 1: $L(R)$ is regular.

Proof:
Induction on the structure of the expression.
Basis: cases 1,2,3—verify directly that the language is regular.
Inductive step: regular operations preserve regularity (last time).
Done.
A by-product of the proof: method to convert any RE into an NFA.

Part 2: For every regular language there is a RE

Idea: design a procedure to convert a DFA into an equivalent RE.
Intuition:
DFA has many states, and its transitions are labeled by single letters.
RE can be thought of as an NFA with two states, and a single transition labeled by the whole RE.
Transform DFA into RE step by step, reducing the number of states, while possibly making transition labels more complex.

Example: strings with an even number of 1s and 0s.

(Done on the whiteboard.)

Formal description: GNFA

A GNFA (generalized nondeterministic finite automaton) is a 5-tuple $(Q, \Sigma, \delta, q_0, q_1)$, where

- Q is a finite set (of *states*).
- Σ is a finite set (*alphabet*, its elements denoted as *letters* or *symbols*).
- δ is a function $\delta : (Q - \{q_1\})(Q - \{q_0\}) \to \mathcal{R}$ (the *transition function*), where $\mathcal{R}$ is the set of all regular expressions.
- q_0 is an element of Q (the *start state*).
- q_1 is an element of Q (the *accepting state*).

Formal description: DFA to RE conversion

First convert DFA M to GNFA G by adding a new start state (q_0) and accept state (q_1) and transitions as necessary.
Convert(G):

- Let k be the number of states of G.
- If $k = 2$, then G has only the start and accept state connected by transition labeled with some RE R. Return R.
- (If $k > 2$) Select any state $q_r \in Q - \{q_0, q_1\}$. Let $G' = (Q', \Sigma, \delta', q_0, q_1)$, where $Q' = Q - \{q_r\}$, and for any $q_i \in Q' - \{q_1\}$ and any $q_j \in Q' - \{q_0\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

  where $R_1 = \delta(q_i, q_r)$, $R_2 = \delta(q_r, q_r)$, $R_3 = \delta(q_r, q_j)$ and $R_4 = \delta(q_i, q_j)$.
- Call Convert(G').

Designing RE

Give a RE for the following language:

1. $\{w \mid w$ doesn't contain the substring $110\}$.
2. $\{w \mid w \notin \{11, 111\}\}$.

Homework 2

Due 1/30 at the **beginning** of class.