CS21
Decidability and Tractability

Lecture 2
January 6, 2017
Outline

• Finite Automata
Terminology

• finite alphabet $\Sigma$ : a set of symbols
• language $L \subseteq \Sigma^*$ : subset of strings over $\Sigma$
• a machine takes an input string and either
  – accepts, rejects, or
  – loops forever
• a machine recognizes the set of strings that lead to accept
• a machine decides a language $L$ if it accepts $x \in L$ and rejects $x \notin L$
What goes inside the box?

- We want the **simplest** mathematical formalization of computation possible.
- **Strategy:**
  - endow box with a feature of computation
  - try to **characterize** the languages decided
  - identify language we “know” real computers can decide that machine cannot
  - add new feature to overcome limits
Finite Automata

• simple model of computation
• reads input from left to right, one symbol at a time
• maintains state: information about what seen so far (“memory”)
  – finite automaton has finite # of states: cannot remember more things for longer inputs
• 2 ways to describe: by diagram, or formally
FA diagrams

- (single) start state
- States: 0, 1, 0, 1, 0, 1
- Alphabet: $\Sigma = \{0, 1\}$
- Transition for each symbol
- (several) accept states
- Read input one symbol at a time; follow arrows; accept if end in accept state
FA operation

• Example of FA operation:

input: 0 1 0 1
not accepted
Example of FA operation:

input: 1 0 1
accepted

What language does this FA recognize?

$L = \{x : x \in \{0,1\}^*, x_1 = 1\}$
Example FA

• What language does this FA recognize?
  \[ L = \{ x : x \in \{0,1\}^*, \text{x has even \# of 1s} \} \]

• illustrates fundamental feature/limitation of FA:
  – “tiny” memory
  – in this example only “remembers” 1 bit of info.
Example FA

\[ \Sigma = \{A,B,C\} \]

\[ \{Q,N,D\} \]

Try:
AC
CBCC
AA
BBBBBBB
CCBC

“35 cents”
FA formal definition

A finite automaton is a 5-tuple

\[(Q, \Sigma, \delta, q_0, F)\]

- \(Q\) is a finite set called the **states**
- \(\Sigma\) is a finite set called the **alphabet**
- \(\delta: Q \times \Sigma \rightarrow Q\) is a function called the **transition function**
- \(q_0\) is an element of \(Q\) called the **start state**
- \(F\) is a subset of \(Q\) called the **accept states**
FA formal definition

• Specification of this FA in formal terms:

  – \( Q = \{ \text{even, odd} \} \)
  – \( \Sigma = \{ 0, 1 \} \)
  – \( q_0 = \text{even} \)
  – \( F = \{ \text{even} \} \)

  \[ \text{function } \delta: \]
  \[ \delta(\text{even, 0}) = \text{even} \]
  \[ \delta(\text{even, 1}) = \text{odd} \]
  \[ \delta(\text{odd, 0}) = \text{odd} \]
  \[ \delta(\text{odd, 1}) = \text{even} \]
Formal description of FA operation

finite automaton

\[ M = (Q, \Sigma, \delta, q_0, F) \]

accepts a string

\[ w = w_1w_2w_3\ldots w_n \in \Sigma^* \]

if \exists sequence \( r_0, r_1, r_2, \ldots, r_n \) of states for which

- \( r_0 = q_0 \)
- \( \delta(r_i, w_{i+1}) = r_{i+1} \) for \( i = 0, 1, 2, \ldots, n-1 \)
- \( r_n \in F \)
What now?

• We have a model of computation
  (Maybe this is it. Maybe everything we can do with real computers we can do with FA…)

• try to characterize the languages FAs can recognize
  – investigate closure under certain operations

• show that some languages not of this type
Characterizing FA languages

• We will show that the set of languages recognized by FA is **closed** under:
  – union “C = (A ∪ B)”
  – concatenation “C = (A ° B)”
  – star “ C = A* ”

• Meaning: if A and B are languages recognized by a FA, then C is a language recognized by a FA
Characterizing FA languages

• union “C = (A ∪ B)”
  \[(A ∪ B) = \{x : x \in A \text{ or } x \in B \text{ or both}\}\]

• concatenation “C = (A ° B)”
  \[(A ° B) = \{xy : x \in A \text{ and } y \in B\}\]

• star “C = A*” (note: ε always in A*)
  \[A^* = \{x_1x_2x_3\ldots x_k : k \geq 0 \text{ and each } x_i \in A\}\]
Concatenation attempt

\[(A \circ B) = \{xy : x \in A \text{ and } y \in B\}\]

What label do we put on the new transitions?
Concatenation attempt

- Need it to happen “for free”: label with $\varepsilon$ (?)
- allows construct with multiple transitions with the same label (!?)

\[ \varepsilon = \]

\[ = \]
Nondeterministic FA

• We will make life easier by describing an additional feature (nondeterminism) that helps us to “program” FAs

• We will prove that FAs with this new feature can be simulated by ordinary FA
  – same spirit as programming constructs like procedures

• The concept of nondeterminism has a significant role in TCS and this course.
NFA diagrams

(single) start state

states

transitions:
• may have several with a given label (or none)
• may be labeled with $\varepsilon$

• At each step, several choices for next state
NFA operation

• Example of NFA operation:

Input: 0 1 0

Not accepted
NFA operation

• Example of NFA operation:

input: 1 1 0

accepted

alphabet
\( \Sigma = \{0,1\} \)
NFA operation

• One way to think of NFA operation:

• string $x = x_1 x_2 x_3 \ldots x_n$ accepted if and only if
  – there exists a way of inserting $\varepsilon$’s into $x$
    
    $x_1 \varepsilon \varepsilon x_2 x_3 \ldots \varepsilon x_n$
    
  – so that there exists a path of transitions from the start state to an accept state
NFA formal definition

A nondeterministic FA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:

- $Q$ is a finite set called the states
- $\Sigma$ is a finite set called the alphabet
- $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to \wp(Q)$ is a function called the transition function
- $q_0$ is an element of $Q$ called the start state
- $F$ is a subset of $Q$ called the accept states
NFA formal definition

- Specification of this NFA in formal terms:
  - $Q = \{s_1, s_2, s_3, s_4\}$
  - $\Sigma = \{0, 1\}$
  - $q_0 = s_1$
  - $F = \{s_4\}$
  - $\delta(s_1, 0) = \{s_1\}$
  - $\delta(s_1, 1) = \{s_1, s_2\}$
  - $\delta(s_1, \varepsilon) = \{\}$
  - $\delta(s_2, 0) = \{s_3\}$
  - $\delta(s_2, 1) = \{\}$
  - $\delta(s_2, \varepsilon) = \{s_3\}$
  - $\delta(s_3, 0) = \{\}$
  - $\delta(s_3, 1) = \{s_4\}$
  - $\delta(s_3, \varepsilon) = \{\}$
  - $\delta(s_4, 0) = \{s_4\}$
  - $\delta(s_4, 1) = \{s_4\}$
  - $\delta(s_4, \varepsilon) = \{\}$
Formal description of NFA operation

NFA $M = (Q, \Sigma, \delta, q_0, F)$
accepts a string $w = w_1w_2w_3\ldots w_n \in \Sigma^*$
if $w$ can be written (by inserting $\epsilon$’s) as:

$$y = y_1y_2y_3\ldots y_m \in (\Sigma \cup \{\epsilon\})^*$$

and $\exists$ sequence $r_0, r_1, \ldots, r_m$ of states for which

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

• Recall: want to show the set of languages recognized by NFA is closed under:
  
  – union “C = (A ∪ B)”
  
  – concatenation “C = (A ° B)”
  
  – star “C = A* ”
Closure under union

$C = (A \cup B) = \{x : x \in A \text{ or } x \in B\}$
Closure under concatenation

\[ C = (A \circ B) = \{xy : x \in A \text{ and } y \in B\} \]
Closure under star

\[ C = A^* = \{x_1 x_2 x_3 \ldots x_k : k \geq 0 \text{ and each } x_i \in A\} \]
NFA, FA equivalence

**Theorem**: a language $L$ is recognized by a **FA** if and only if $L$ is recognized by a NFA.

Must prove *two* directions:

$(\Rightarrow)$ L is recognized by a FA implies $L$ is recognized by a NFA.

$(\Leftarrow)$ L is recognized by a NFA implies $L$ is recognized by a FA.

(usually one is easy, the other more difficult)
NFA, FA equivalence

\( \iff \) L is recognized by a FA implies L is recognized by a NFA

**Proof**: a finite automaton is a nondeterministic finite automaton that happens to have no \( \varepsilon \)-transitions, and for which each state has exactly one outgoing transition for each symbol.
NFA, FA equivalence

\( \Leftarrow \) L is recognized by a NFA implies L is recognized by a FA.

**Proof**: we will build a FA that simulates the NFA (and thus recognizes the same language).

– alphabet will be the same

– what are the states of the FA?