COMP3310/3610: Theory of Computation

Taso Viglas
tasos@it.usyd.edu.au

University of Sydney

July 31, 2007

Computation

- Simple computation models
- Machines or programming models
- Question: "What can we compute?"
- Problems solvable computationally
- Formalize the notion of computation
  actually we need to formalize the notions of 'problem', 'solvable' and 'computationally'

sets and membership

- decision problems
- languages

finite state automata

- regular languages

non-determinism

- non-deterministic automata
- regular operations
- equivalence of automata and regular expressions

Non-regular languages

- the pumping lemma
- pumping lemma examples

decision problems

- languages

Computation

- Decision problems: given a graph $G$ decide whether it is connected or not
- non-decision problems: in the graph $G$, find the shortest path from vertex $s$ to $t$
- in a decision problem, the algorithms receives an input (for example a graph $G$) and responds yes or no (true/false or accept/reject, or 1/0)
- the problem versus the instance of a problem
  the problem: graph connectivity the instance: a specific graph $G$
A decision problem

- Problem: given an undirected graph $G$, decide if it is connected
- Fix an alphabet. For example $\Sigma = \{0, 1\}$
- Input: representation/description of a graph, as a string using the fixed alphabet
- Output is 0/1 (reject/accept the given input

Deciding membership

- Input: can be any string, which will be interpreted as a problem instance
- Example below shows 3 different problems that may interpret input strings their own way
- If an input string does not correspond to a valid instance, just reject it
- Example of a problem definition: the problem of graph connectivity can be defined as the set of strings that represent graphs that are connected

<table>
<thead>
<tr>
<th>Input</th>
<th>Connected</th>
<th>Complete</th>
<th>Prime</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100...1010</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111...010011</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>001011...101</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>00000...0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Notation

- Defining a set: $A = \{ x | x \text{ is a positive natural number with some property}\}$
- Union: $A \cup B$
- Intersection: $A \cap B$
- Complement: $\bar{A}$
- Cartesian Product: $A \times B = \{(x, y) | x \in A, y \in B\}$
- $\Sigma = \{0, 1\}$, then $\Sigma \times \Sigma = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$
- Power set: $\mathcal{P}(A)$
  - $\Sigma = \{0, 1\}$, then $\mathcal{P}(\Sigma) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$

Sets and membership

- $PRIMES = \{ x | x \text{ is the binary representation of an integer that is prime}\}$
- The primality problem: given a binary string $x$, does it belong to the set $PRIMES$?
- $x \in PRIMES$
Languages

- An alphabet is a (finite) set of symbols, \( \Sigma = \{ a, b \} \)
- A word is a concatenation of alphabet symbols
  (a string over a given alphabet), \( w = abba \)
- A language is a set of words over a given alphabet
  \( L = \{ a, aa, abba \} \), or
  \( L = \{ \text{all words starting with an } a \} \)
  \( L = \{ a^n b^n | n \in \mathbb{N} \} \)

Machines and languages

- Let \( L \) be a language
- A machine \( M \) recognizes \( L \)
  if \( M(x) = 1 \) iff \( x \in L \), otherwise
  \( M(x) = 0 \)
- \( M \) accepts all strings in the language \( L \) and rejects all other strings

Finite Automaton

- A finite state machine, reads its input once (left to right) and
  accepts or rejects
- Finite automaton has fixed size memory
- Example of an automaton

Finite automaton: formal definition

- Def: A deterministic finite state automaton (DFA) \( M \) is a
  5-tuple \( M = (Q, \Sigma, \delta, q_0, F) \)
  - \( Q \) finite set of states
  - \( \Sigma \) finite alphabet
  - \( \delta \) transition function, \( \delta : Q \times \Sigma \to Q \)
  - \( q_0 \in Q \) starting state
  - \( F \subseteq Q \) accepting states
A deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string/word $w = w_1 \ldots w_n$ if there exists a sequence $r_0 \ldots r_n$ of states in $Q$ such that
- $r_0 = q_0$
- $\delta(r_i, w_{i+1}) = r_{i+1}$ for all $i = 0, \ldots, n - 1$
- $r_n \in F$

$L(M)$ denotes the language recognized by $M$

**Regular languages**

- (Def 1.16p40) A language is called regular if some finite automaton recognizes it.
- A regular language is completely described by its automaton

**Regular operations**

- (Def 1.23p44) Let $A$ and $B$ be languages. We define the regular operations as follows:
  - Union: $A \cup B = \{ x | x \in A \text{ or } x \in B \}$
  - Concatenation: $A \circ B = \{ xy | x \in A, y \in B \}$
  - Star: $A^* = \{ x_1 x_2 \ldots x_k | k \geq 0, x_i \in A \}$
Regular operations

- Regular languages are closed under the regular operations
- (Thm 1.25p45) The class of regular languages is closed under union
- Proof of closure properties is easier through non-deterministic automata

Non-determinism

- Computation: sequence of steps
- Given the current state and input symbol, the next state is completely determined
- Non-deterministic computation: many possible next states
- example: Non-deterministic FA
  - empty transitions
  - many outgoing arrows for the same symbol
- multiple ways to read through an input
- try all possibilities in parallel. accept if any computation path exists

Examples of NFAs

- alphabet $\Sigma = \{a, b\}$
- $L_1 = \{w | w \text{ contains } 'aba' \text{ as a substring}\}$
- $L_2 = \{w | \text{length of } w \text{ is a multiple of 3}\}$

example of a NFA

$L = \{ \text{all strings whose third symbol from the end is 1} \}$
Definition of an NFA

(Def 1.37p53) a non-deterministic FA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$:
- $Q$ is a finite set of states
- $\Sigma$ is a finite alphabet (set of symbols)
- $\delta: Q \times (\Sigma \cup \{\epsilon\}) \rightarrow P(Q)$ is the transition function
- $q_0 \in Q$ is the starting state
- $F \subseteq Q$ are the final (accepting) states

Two machines are called equivalent if they accept the same language

(Th 1.39p55) Every NFA has an equivalent FA
Proof: by construction

Proof of NFA and DFA equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, construct an equivalent DFA $M = (Q', \Sigma, \delta', q'_0, F')$

First, assume there are no $\epsilon$ arrows
If the number of states in $Q$ is $k$ the all possible subsets of states is $2^k$

$Q' = P(Q)$
for $R \in Q'$, $a \in \Sigma$ define $\delta'(R, a) = \{ q \in Q | q \in \delta(r, a), r \in R \}$

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

$E(q_0) = \{ q \}$, and $F' = \{ R \in Q' | R contains a final state of N \}$

Proof of NFA and DFA equivalence: $\epsilon$ arrows

define $E(R) = \{ all states reachable from R by \epsilon moves only \}$
modify definition $\delta'(R, a) = \{ q \in Q | q \in E(\delta(r, a)), r \in R \}$
start state $q'_0 = \{ E(q_0) \}$
(end of proof)

(Corollary 1.40p56) A language is regular if and only if some NFA accepts it

Closure under regular operations

(Theorems 1.45-49p59) Regular languages are closed under union, concatenation and star
Proof: by construction
Regular expressions

- Assume an alphabet $\Sigma$
- (Definition 1.52) a regular expression $R$ is either:
  - $a \in \Sigma$
  - $\epsilon$ (the empty string)
  - $\emptyset$ the empty regular expression
  - $(R_1 \cup R_2)$, $(R_1 \circ R_2)$, $(R^*)$, where $R, R_1, R_2$ are regular expressions
- examples
  - $0^*10^* = \{w | w$ contains exactly one '1'\}
  - $\Sigma^*1\Sigma^*$
  - $(0 \cup 1)^*1(0 \cup 1)^*$

FAs and regular expressions

- (Theorem 1.54) a language is regular if and only if some regular expression describes it
- proof: by construction, two directions
- (Lemma 1.55) Any regular expression can be converted into an NFA
- (Lemma 1.60) Any DFA can be converted into a regular expression
- proof of the first lemma is straightforward: go through inductive definition of a regular expression and show how to build the NFA for every case
- second lemma requires some more work

Generalized NFA: GNFA

- GNFA, generalized NFA: arrow labels are regular expressions
- start state has arrows to every other state and no incoming arrows
- single accepting state
- only one arrow between two states, no multiple labels

Generalized NFAs

- a GNFA is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{accept})$, where
  - $Q, \Sigma$ finite set of states, and alphabet
  - $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R$
  - $q_{start}$ is the start state
  - $q_{accept}$ is the accept state
DFA → regular expression

- convert DFA to a GNFA such that:
  - start state has arrows to every other state and no incoming arrows
  - single accepting state
  - only one arrow between two states, no multiple labels
- DFA → GNFA with \( k \) states → \( k - 1 \) states → \( \ldots \) 2 states
- Remove any internal state and repair the machine (update labels)
- complete proof by induction (proving that the conversion step is correct)

Non-regular languages

- power of FAs, regular languages
- \( B = \{0^n1^n|n \geq 0\} \)
- \( C = \{w|w \text{ has the same number of 0s and 1s}\} \)
- \( D = \{w|w \text{ has the same number of occurrences of } '01' \text{ and } '10' \text{ as substrings}\} \)
- \( B \) and \( C \) are not regular but \( D \) is regular (why ?)
- limitation: fixed number of states
  - fixed amount of memory
  - proving that languages are not regular: the pumping lemma

DFA → regular expression

- main step: removing states from the GNFA
- remove any internal state and update transition labels

Non-regular languages

- Long paths in DFAs
  - long strings correspond to long (computation) paths
  - long paths must repeat states (visit a state twice)
  - therefore there is a loop in the path
  - consider a DFA with \(|Q| = k\) that accepts a string \( w \) of size \( p \leq k \)
  - on input \( w \) the DFA will visit \( p + 1 \) states
  - \( q_1, \ldots, q_i, \ldots, q_i, \ldots, q_k \)
  - repeat \( q_i, \ldots, q_i \) part. same final state
Non-regular languages

(\textit{Theorem 1.70p78}) Pumping lemma
If \( A \) is regular then it
there exists \( p \) (pumping length) such that,
if \( s \) is any string in \( A \) of length at least \( p \)
then \( s \) may be divided in 3 parts, \( s = xyz \) satisfying:

1. for each \( i \geq 0 \), \( xy^i z \in A \)
2. \(|y| > 0 \) and
3. \(|xy| \leq p \)

Pumping examples -1

\( B = \{0^n|n \geq 0\} \) (ex. 1.73p80)
assume \( B \) is regular and \( p \) is the pumping length, and
\( s = 0^p1^p \)
then \( \exists x, y, z : s = xyz = 0^p1^p \) and \( \forall i \geq 0 : xy^i z \in B \)
three cases for \( i \):

1. if \( y = 0^k \) then \( xy^i z = 0^p+k1^p \in B \) - contradiction
2. if \( y = 1^k \) then \( xy^i z = 0^p1^p+k \in B \) - contradiction
3. if \( y = 0^k1^l \) then \( xy^i z = 0^p+k1^p0^k1^p+l \in B \) - contradiction

Pumping examples -2

\( F = \{ww|w \in \Sigma^*\} \) (ex. 1.75p81)
Let \( p \) be the pumping length and consider \( s = 0^p1^p0^p1 \)
\( s = xyz \) and \(|xy| \leq p \). One case: \( y = 0^k \) and
\( xy^i z = 0^p+k10^p1 \in F \)

note that choice of the string \( s \) is important.
if in this example we choose \( s = 0^p0^p \) the pumping lemma will not work.
Pumping examples -3

- $E = \{ 0^i 1^j | i \geq j \}$ (ex. 1.77p82)
  - consider $s = 0^p 1^p$. cases as first example
  - cases 2 and 3 still give contradictions. but pumping up the zeroes will not give a contradiction $xyz \in E$
  - Pump down $s = xz = 0^{p-k} 1^p$
- note: pumping lemma implies that all FAs accept "short" strings

Two questions

- Is the class of regular languages closed under intersection? prove you claim.
- NFA for a language can be much smaller than the DFA for the same language. can you give an example for such a language?