Theory of Computation

(Feodor F. Dragan)
Department of Computer Science
Kent State University

Spring, 2018

Before we go into details,

what are the two fundamental questions
in theoretical Computer Science?

1. **Can** a given problem **be solved** by computation?
2. **How efficiently** can a given problem be solved by computation?

We focus on *problems* rather than on specific *algorithms* for solving problems.

To answer both questions mathematically, we need to start by formalizing the notion of "computer" or "machine".

So, course outline breaks naturally into three parts:

1. Models of computation (*Automata theory*)
   - Finite automata
   - Push-down automata
   - Turing machines
2. What can we compute? (*Computability Theory*)
3. How efficient can we compute? (*Complexity Theory*)

We start with overview of the first part

Models of Computations or Automata Theory

First we will consider more restricted models of computation

- Finite State Automata
- Pushdown Automata

Then,

- (universal) Turing Machines

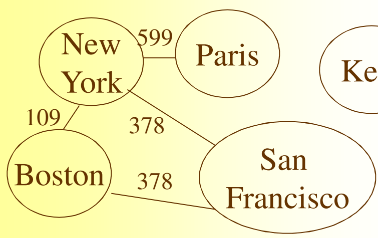
We will define "*regular expressions*" and "*context-free grammars*" and will show their close relation to Finite State Automata and to Pushdown Automata.

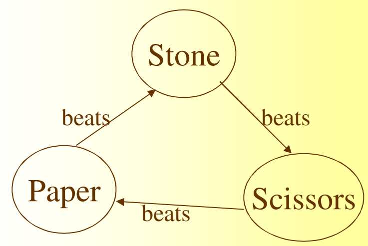
Used in compiler construction (lexical analysis)

Used in linguistic and in programming languages (syntax)

Graphs

- $G=(V,E)$
- vertices (V), edges (E)
- labeled graph, undirected graph, directed graph
- subgraph
- path, cycle, simple path, simple cycle, directed path
- connected graph, strongly connected digraph
- tree, root, leaves
- degree, outdegree, indegree





Strings and Languages

- *Alphabet* (any finite set of symbols)

  $\Sigma_1=\{0,1\}$

  $\Sigma_2=\{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$

  $\Gamma=\{0,1,x,y,z\}$

- A *string* over an alphabet (a finite sequence of symbols from that alphabet)

  $w_1=01001$ over Σ_1

  $w_2=abracadabra$ over Σ_2

  $|w_1|$ is the length of string w_1 (=5)

  ε is an empty string

  the reverse of w_1 is $w_1^R=10010$

  substring, concatenation ($\overbrace{xx\cdots x}^{k}=x^k$)

Strings and Languages

- A *language* is a set of strings over a given alphabet (Σ_1).

$$L_1 = \{w: \ w \text{ contains 001 as a substring } \}$$

$$L_2 = \{w: \ |w| \text{ is even } \}$$

$\varepsilon \in L_2$

$|L_1|$ is the length of L_1 (this is an infinite language)

- Usual set operations as union and intersection can be applied to languages. $L_1 = \{11, 001\}$, $L_2 = \{0, 10\}$ ➔ $L_1 \cup L_2 = \{0, 10, 11, 001\}$, $L_1 \cap L_2 = \{\ \}$

concatenation of two languages

$$L_1 L_2 = \{110, 1110, 0010, 00110\}$$

$$L_2 L_1 = \{011, 0001, 1011, 10001\}$$

Definitions, theorems, proofs

- ***Definitions*** describe the objects and notations
  - Defining some object we must make clear what constitutes that object and what does not.
- ***Mathematical statements*** about objects and notions
  - a statement which expresses that some object has a certain property
  - it may or may not be true, but must be precise
- A ***proof*** is a convincing logical argument that a statement is true
- A ***theorem*** is a mathematical statement proved true
  - this word is reserved for statements of special interest
  - statements that are interesting only because they assist in the proof of another, more significant statement, are called ***lemmas***
  - a theorem or its proof may allow us to conclude easily that another, related statements are true; these statements are called ***corollaries*** of the theorem

An example

- ***Definitions:***
  - A graph *G=(V,E),* a node *v*, an edge *(v,u),* # of edges |*E*|,
  - incident, the degree *d(v)* of a node *v,*
  - sum, even number.

- ***Theorem:*** For every graph *G*, the sum of the degrees of all the nodes in *G* is *2*|*E*|.
- ***Corollary:*** For every graph *G,* the sum of the degrees of all the nodes in *G* is an even number.

OR

- ***Lemma:*** For every graph *G*, the sum of the degrees of all the nodes in *G* is *2*|*E*|.
- ***Theorem:*** For every graph *G,* the sum of the degrees of all the nodes in *G* is an even number.
- ***Proof:*** (easy)

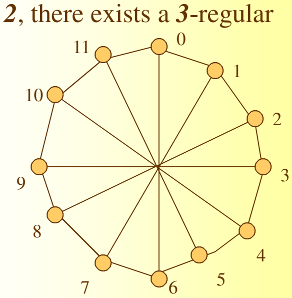
Types of proofs:

Proof by construction

- If theorem states that a particular type of object exists.
- A way to prove such a theorem is by demonstrating how to construct the object.
- A way to disprove a "theorem" is to construct an object that contradicts that statement (called a *counterexample*).
- ***Definition:*** A graph is ***k*-regular** if every node in the graph has degree ***k***
- ***Theorem:*** For each even number ***n*** greater than **2**, there exists a ***3***-regular graph with ***n*** nodes.
- ***Proof:*** Construct a graph *G=(V,E)* as follows.

$$V = \{0,1,\ldots,n-1\}$$

$$E = \{\{i,i+1\}:\ for\ \ 0 \le i \le n-2\} \cup \{\{n-1,0\}\}$$
$$\cup \{\{i,i+n/2\}:\ for\ \ 0 \le i \le n/2-1\}$$



Proof by induction

- Prove a statement ***S(X)*** about a family of objects ***X*** (e.g., integers, trees) in two parts:

1. *Basis:* Prove for one or several small values of ***X*** directly.
2. *Inductive step:* Assume ***S(Y)*** for ***Y*** ``smaller than" ***X***;
   prove ***S(X)*** using that assumption.

Theorem: A binary tree with n leaves has $2n-1$ nodes.

Proof:
- formally, *S(T):* **if** T is a binary tree with n leaves, **then** T has $2n - 1$ nodes.
- induction is on the *size* = # of nodes in T.

Basis: if T has 1 node, it has 1 leaf. $1=2-1$, so OK

Induction: Assume $S(U)$ for trees with fewer nodes that T.
- T must be a root plus two subtrees U and V
- If U and V have u and v leaves, respectively, and T has t leaves, then $u + v = t$.
- By the induction hypothesis, U and V have $2u - 1$ and $2v - 1$ nodes, respectively.
- Then T has $1 + (2u - 1) + (2v - 1)$ nodes
  - $= 2 (u + v) - 1$
  - $= 2 t - 1$, proving inductive step.

If-And-Only-If Proofs

- Often, a statement we need to prove is of the form "***X* if and only if *Y***". We are often required to do two things:

1. Prove the *if-part:* Assume Y and prove X.
2. Prove the *only-if-part:* Assume X and prove Y.

Remember:

- the *if* and *only-if* parts are *converses* of each other.
- one part, say "if X then Y", says nothing about whether Y is true when X is false.
- an equivalent form to "if X then Y" is "if not Y then not X": the latter is the *contrapositive* of the former.

Equivalence of Sets

- many important facts in language theory are of the form that two sets of strings, described in two different ways, are really the same set.
- to prove sets S and T are the same, prove: x is in S if and only if x is in T. That is
  - Assume x is in S; prove x is in T.
  - Assume x is in T; prove x is in S.

Example: Balanced Parentheses

- Here are two ways that we can define ``*balanced parentheses*'':

1. *Grammatically*:

   a) The empty string ε is balanced.

   b) If *w* is balanced, then *(w)* is balanced.

   c) If *w* and *x* are balanced, then so is *wx*.

2. *By Scanning :* *w* is balanced if and only if:

   a) *w* has an equal number of left and right parentheses.

   b) Every prefix of *w* has at least as many left as right parentheses.

- Call these **GB** and **SB** properties, respectively.

Theorem: A string of parentheses *w* is **GB** if and only if it is **SB**.

If part of the proof

- An induction on |*w*| (length of *w*). Assume *w* is **SB**; prove it is **GB**.

Basis: If $w = \varepsilon$ (length = 0), then *w* is **GB** by rule (a).

- Notice that we do not even have to address the question of whether ε is **SB** (it is, however).

Induction: Suppose the statement ``**SB** implies **GB**'' is true for strings shorter than *w*.

- *Case 1:* *w* is not ε, but has no nonempty prefix that has an equal number of (and).
  Then *w* must begin with (and end with) ; i.e., *w = (x).*
  - *x* must be **SB** (why?).
  - By the **IH**, *x* is **GB**.
  - By rule (b), *(x)* is **GB**; but *(x)* = *w*, so *w* is **GB**.
- *Case 2:* *w* = *xy*, where *x* is the shortest, nonempty prefix of *w* with an equal number of (and), and y is not ε .
  - *x* and *y* are both **SB** (why?).
  - By the **IH**, *x* and *y* are **GB**.
  - *w* is **GB** by rule (c).

Only-If part of the proof

• An induction on |*w*| (length of *w*). Assume *w* is **GB**; prove it is **SB**.

Basis: If $w = \varepsilon$ (length = 0), then clearly *w* is **SB**.

Induction: Suppose the statement ``**GB** implies **SB**" is true for strings shorter than *w,* and assume that *w* is not ε .

- *Case 1: w* is **GB** because of rule (b); i.e., *w = (x)* and *x* is **GB**.
  - by the **IH**, *x* is **SB**.
  - Since *x* has equal numbers of ('s and)'s, so does *(x).*
  - Since *x* has no prefix with more)'s than ('s, so does *(x).*
- *Case 2: w* is not ε and is **GB** because of rule (c); i.e., *w = xy*, and *x* and *y* are **GB**.
  - By the **IH**, *x* and *y* are **SB**.
  - (Aside) Trickier than it looks: we have to argue that neither *x* nor *y* could be ε , because if one were, the other would be *w*, and this rule application could not be the one that first shows *w* to be **GB**.
  - *xy* has equal numbers of ('s and)'s because *x* and *y* both do.
  - If *w* had a prefix with more)'s than ('s, that prefix would either be a prefix of *x* (contradicting the fact that *x* has no such prefix) or it would be *x* followed by a prefix of *y* (contradicting the fact that *y* also has no such prefix).
  - (Aside) Above is an example of *proof by contradiction*. We assumed our conclusion about *w* was false and showed it would imply something that we know is false.

Finite Automata

• An important way to describe certain simple, but highly useful languages called ``*regular languages*."

- A *graph* with a finite number of *nodes*, called *states*.
- *Arcs* are *labeled* with one or more *symbols* from some *alphabet*.
- One state is designated the *start state* or *initial state*.
- Some states are *final states* or *accepting states*.
- The *language of the FA* is the *set of strings* that label paths that go from the start state to some accepting state.

Example

• This FA scans HTML documents, looking for a list of what could be title-author pairs, perhaps in a reading list for some literature course.

• It accepts whenever it finds the end of a list item.

• In an application, the strings that matched the title (before ' **by** ') and author (after) would be stored in a table of title-author pairs being accumulated.