LECTURE 1

Overview

(1) Understanding computation and computability
(2) Study finitary representations for languges and machines
(3) Understanding capabilities of abstract machines

Algorithms and Procedures (intuitive)

Procedure: finite sequence of instructions that can be carried out mechanically, say by a computer program

Algorithm: a procedure that always halts is an algorithm

Example 1

Determine if $i > 1$ is a prime

1. set $j = 2$
2. if $j \geq i$ then halt; i is a prime
3. if i/j is an integer then halt; i is not a prime
4. $j \leftarrow j+1$
5. go to 2

Example 2

A perfect number is one for which the sum of its divisors, except for itself, equals the number:

$1+2+4 \neq 8$	8 is not a perfect number
$1+2+3 = 6$	6 is a perfect number
	28, 496, 8128 are perfect numbers

Define a procedure for determining if $\exists$ a perfect number $> i$:

1. $j = i + 1$;
2. check if j is perfect, if so halt.
3. $j \leftarrow j+1$
4. go to 2

Example 3

If $P \neq NP$ then ....

Algorithm — Example 1 always halts & answers yes or no.

Procedure — Example 2 halts if there are an infinite number of perfect numbers

Not a procedure — Example 3

Mathematical preliminaries

Sets: $S = \{a, b, dd\}$; $S = \{i : i > 0 \;\&\; i \text{ is even}\}$

S = set of measurable subsets.

$A \subseteq B$ A is a subset of B.

Power set. $S = \{a, b, c\}$

$2^S = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$

Cardinality (size) of a set. $|S| = 3$

Note $|2^S| = 2^{|S|}$

Functions: $f : S_1 \to S_2$

If domain of f is all of S_1 it is a total function

" " " " is a subset of S_1 it is a partial function

Graphs & trees: normal usage

vertices, edges, digraph etc.

Relation subset of $S_1 \times S_2$

Proof by contradiction (in text)

Proofs

Proof by induction

Example 1

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Basis: $n = 1$ $\qquad \sum_{i=1}^{1} i^2 = 1$ trivially true.

Inductive step.

Assume true for n.

Prove true for $n+1$.

$$\sum_{i=1}^{n+1} i^2 = \sum_{i=1}^{n} i^2 + (n+1)^2 = \frac{n(n+1)(2n+1)}{6} + (n+1)^2$$

$$= \frac{n(n+1)(2n+1) + 6(n+1)^2}{6}$$

$$= \frac{(n+1)\left[n(2n+1) + 6(n+1)\right]}{6}$$

$$= \frac{(n+1)(n+2)(2n+3)}{6} = \frac{(n+1)\left((n+1)+1\right)\left(2(n+1)+1\right)}{6}$$

Q.E.D.

Example 2

$P(n,r) =$ number of ways of arranging r of n distinct objects.

{ note we know it should be
$P(n,r) = \frac{n!}{(n-r)!}$ }

First. show $P(n,n) = n!$ by induction

Basis $P(1,1) = 1 = 1!$ OK ✓

Assume $P(n-1, n-1) = (n-1)!$ (Prove true for n; $n \geq 2$)

To arrange n distinct objects in order, single out a special object and arrange remaining $n-1$ objects first. For each ordered arrangement of the $n-1$ objects, there are n positions for the special object. ($n-2$ between + 2 end positions)

$\therefore$ By "product rule"

$P(n,n) = n \times P(n-1, n-1) = n \cdot (n-1)! = n!$

Can then argue that

$P(n,n) = P(n,r) \times P(n-r, n-r)$

$\therefore P(n,r) = \frac{P(n,n)}{P(n-r,n-r)} = \frac{n!}{(n-r)!}$

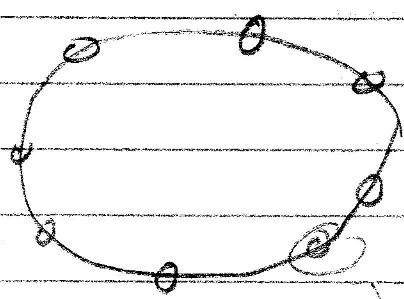
Example 3

All horses are the same color

Basis : clearly true for 1 horse ✓

Assume true for n
Show true for $n+1$

Put $n+1$ horses in a circle



$n+1$ horses

remove 1

By induction n (remaining) are the same color.
Remove another horse and repeat.

$\Rightarrow$ all horses are the same color.

Problem with proof ?

Diagonalization arguments

Prove that there are not a countable # of real numbers. (ie. an uncountable #) in [0, 1].
Use binary representation.

$x_1 = .\,x_{11}\ x_{12}\ x_{13}\ x_{14} \cdots$

$x_2 = .\,x_{21}\ x_{22}\ x_{23}\ x_{24} \cdots$

$\vdots \qquad \vdots$

$x_n = .\,x_{n,1}\ x_{n,2}\ x_{n_3} \cdots$

$\vdots$

Consider

Let $y_{ii} = \overline{x_{ii}}$

$\therefore\ .\,y_{11}\ y_{22}\ y_{33} \cdots$ is not in list.

(Proof by contradiction!)

Languges

Def. Any finite set of symbols is an alphabet or vocabulary

$$V = \{A, B, C, D, \ldots, Z\}$$
$$V = \{0, 1\}$$
$$V = \{\square, IF, THEN, ELSE\}$$

(Note that we are also using metasymbols $\{$, $\}$ in description)

Note: (We will normally use Σ for an alphabet later)

Symbols from an alphabet can be concatenated to form sentences.

Def A sentence (or string or word) over alphabet V is a finite ordered sequence of symbols.

$V = \{A, B, C, \ldots, Z\}$ ALPHA is a word or string

$V = \{0, 1\}$ 01101 is a string

Note: two strings can be concatenated to form another string:

concatenate(ALPHA, BETA) = ALPHABETA

Let α be a string. $|\alpha|$ is the length of α; number of symbols in the string.

The "empty" string is denote λ or ϵ and has length 0.

Let V be an alphabet. Then V^* is the set of all strings over V, including the empty string.

$V^+ = V^* - \{\lambda\}$

Let $V = \{a, b, c, d\}$ What is V^*?

Can you specify a procedure to generate V^*?

<u>Def</u> A <u>language</u> over V is a set of strings over V.
Alternatively a <u>language</u> is $L \subseteq V^*$

<u>Examples</u> $V = \{a, b\}$

$\{a, aba, aa\}$ a finite language

$\{a^n b^n \mid n \geq 1\}$ an infinite language

Finite representation for languages?

(A) Recognition point of view.

① Give a <u>procedure</u> which <u>halts</u> for sentences in the language and either does not terminate or says <u>no</u> for sentences not in the language.

② Give an <u>algorithm</u> as above.

The procedure <u>recognizes</u> the language

Note: $\exists$ languages that can be recognized by procedures but not by algorithms

(B) Generation point of view.
Systematically generate (enumerate) all sentences of the language.

Given a procedure to recognize L, we can give a procedure for generating L. (Note that some care must be taken since the procedure for recognizing might not halt).

Proof

List in some order all strings x_i over the alphabet of L. Run the recognizing procedure for a number of steps as indicated.

	steps ①	②	③	④	⑤	...
x_1	1	3	6	10	15	
x_2	2	5	9	14		
x_3	4	8	13			
x_4	7	12				
x_5	11					
⋮						

1. run 1 step on x_1
2. run 1 step on x_2
3. run 2 steps on x_1
4. run 1 step on x_3

⋮

Given a procedure for generating L, we can give a procedure for recognizing L. What is it?

Def A language L that can be generated by a procedure is said to be a <u>recursively enumerable set</u> or R.E.
A language L that can be recognized by an algorithm is said to be <u>recursive</u>

Informal Def. A grammar is a method to generate the strings of a language.
Informally, a grammar has:

1. start symbol S
2. V_N: non-terminal symbols
3. V_T: terminal symbols
4. finite set of productions or rewrite rules

Example 1

$S \to ABC$

$AS \to Ba$

$AB \to ac$

$C \to a$

$aC \to c$

$S \Rightarrow ABC$

$\Rightarrow acC$

$\Rightarrow aca$ ⇝ all terminals

∴ a string in the language.

Note: language $\subseteq V_T^*$

Example 2

$S \to 0S1$

$S \to 01$

$L \stackrel{?}{=} \{0^n 1^n \mid n \geq 1\}$

Example 3

$S \to$ <noun phrase> □ <verb phrase>

<noun phrase> $\to$ <article> □ <noun>

<article> $\to$ a | the

<noun> $\to$ boy | girl

<verb phrase> $\to$ is □ <adjective>

<adjective> $\to$ good | bad

the boy is bad

Formal Definition

Def. A phrase structure grammar (or unrestricted grammar) G is a 4-tuple $G = \langle N, T, P, S \rangle$ where:

N is a finite set of nonterminals including S ($S \in N$).

S is the start symbol.

T is a finite set of terminals with $T \cap N = \phi$.

P is a finite set of productions or rewrite rules

$$\alpha \rightarrow \beta$$

where $\alpha \in V^+$ and $\beta \in V^*$ with $V = N \cup T$.

Define a relation $\Rightarrow$ on $V^* \times V^*$ called directly derives by $x \Rightarrow y$

iff $x = \gamma \alpha \delta$, $\gamma, \delta \in V^*$

$y = \gamma \beta \delta$

and $\alpha \rightarrow \beta \in P$

Let $\Rightarrow^*$ be the relation that is the reflexive transitive closure of $\Rightarrow$.

Def. For a grammar G, the set of sentential forms is $S(G) = \{ \alpha \in V^* \mid S \Rightarrow^* \alpha \}$

Def. For a grammar G, the language generated by G

$$L(G) = S(G) \cap T^*$$
$$= \{ w \in T^* \mid S \Rightarrow^* w \}$$

<u>Example</u> $G = (N, T, P, S)$

$N = \{S, B, C\}$, $T = \{a, b, c\}$

Productions
1. $S \rightarrow aSBC$
2. $S \rightarrow aBC$
3. $CB \rightarrow BC$
4. $aB \rightarrow ab$
5. $bB \rightarrow bb$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

What is $L(G)$?

$L(G) = \{a^n b^n c^n \mid n \geq 1\}$

By putting restrictions on the types of production rules we obtain various specialized grammars.

The most general grammar is called a phrase structure grammar or type 0 grammar.
Sometimes type 0 is defined by

$$\alpha \rightarrow \beta \quad , \quad \alpha \in V^* N V^* \quad , \quad \beta \in V^*$$

or $\alpha \rightarrow \beta$, $\alpha \in N^+$, $\beta \in V^*$

<u>Def</u> A grammar is said to be context sensitive <u>with</u> erasing if productions are of the form

$$\varphi A \psi \rightarrow \varphi \alpha \psi \quad , \quad \varphi, \psi \in V^* , \quad A \in N, \quad \alpha \in V^*$$

φ is left context

ψ is right context.

Note: context sensitive with erasing is equivalent to Type 0. (equivalent with respect to languages)

Type 0 languages are recursively enumerable (r.e)

<u>Def</u> A grammar is type 1 or context sensitive <u>without</u> erasing if each production is of the form $\varphi A \psi \to \varphi \alpha \psi$ with $\varphi, \psi \in V^*$
$A \in N$, $\alpha \in V^+$.

<u>Def</u> A grammar is <u>monotonic</u> if for each production $\alpha \to \beta$, we have $|\beta| \geq |\alpha|$
(also called just context sensitive).

<u>Lemma</u> Type 1 is equivalent to monotonic

Context sensitive is sometimes defined as monotonic.

Note: A context sensitive language
[$L = L(G)$ or $L = L(G) \cup \{\lambda\}$ where G is a context sensitive grammar]

is recursive

Type 2 Grammars

If every rule $\alpha \to \beta$ is of the form

$$A \to \alpha \quad \text{with } A \in N \text{ and } \alpha \in V^*$$

This is also called context-free (Note $A \to \lambda$ is OK)

Example

$S \to ABD$ $\qquad B \to bC$

$A \to AAC$ $\qquad C \to c$

$A \to ab$ $\qquad D \to a$

Example

$S \to 0S1$ $\qquad$ Note that $L(G) = \{0^n 1^n \mid n \geq 0\}$

$S \to \lambda$ $\qquad$ for this example

Type 3 Grammars

If each rule is of the form:

$A \to xB$ $\qquad x \in T^*,\ A, B \in N$

$A \to x$

it is called a right linear grammar.

for $\left.\begin{matrix} A \to Bx \\ A \to x \end{matrix}\right\}$ it is a left linear grammar.

If a grammar is left linear or right linear it is a regular grammar.

Note 1. Could also define as $A \to aB$ where $a \in T$.

Note 2. A language L is of type i if $\exists$ grammar G s.t $L(G) = L$ and G is of type i.

The Chomsky Hierarchy

Grammars	Languages	Automata
Type 0 phrase structure, unrestricted, context sensitive with erasing	recursively enumerable sets (r.e.)	Turing machines non-det or det
Type 1 monotonic context sensitive	recursive ⊃ context sensitive	nondet linear bounded automata
Type 2 context-free	context-free	non-det p.d.a
LR(k)	det. context-free	det p.d.a
Type 3 regular left-linear right-linear	regular set "set defined by a regular expression"	finite state automata non-det. or det.