

# Computability Notes

## Lecture 1

In this section of the course we will explore the notion of computation. Till now we have seen problems which have been solvable, that is, for which there exist procedures. One would like to ask whether there exist problems which cannot be solved? Are there problems that can be solved but are ‘intractable’? What do we mean by ‘computation’? In answering these questions we will present a standard model of computation which intuitively corresponds to our notion of computation. We will also prove some basic results about this model.

This area of mathematics was motivated by questions such as the problem posed by David Hilbert, whether there exists a procedure to determine the truth or falsity of any mathematical proposition. This question was resolved by Kurt Gödel who proved that no such procedure could exist by constructing a formula whose very definition stated that it could not be proved or disproved. Other related questions are about intelligence of machines. Alan Turing gave a famous test to determine intelligence, which is based on a machine’s ability to imitate a human.

**Definition.** An *alphabet* is a non-empty, finite set of symbols. For example  $A = \{0, 1, 2, a, b, c\}$  is an alphabet. The set of symbols on a keyboard is also an alphabet.  $A^*$  is the set of strings over the alphabet  $A$ .

Let  $A = \{a_0, \dots, a_r\}$  be an alphabet; let  $W \subseteq A^*$ ; and let  $P$  be a ‘procedure’. For now we can consider a procedure to be a program or recipe with a finite description. We will be more precise later.

### Definitions:

1.  $P$  is a *decision procedure* for  $W$  iff  $\forall x \in A^*$ ,  $P$ , given  $x$ , halts and returns  $\square$  if  $x \in W$  and  $a_0$  otherwise.
2.  $W$  is *decidable* if there exists a decision procedure for  $W$ .

3.  $P$  is an *enumeration procedure* for  $W$  iff  $P$  outputs exactly the the elements of  $W$  (in any order with repetitions allowed).
4.  $W$  is *enumerable* if there exists an enumeration procedure for  $W$ .

*Example.*  $A^*$  is enumerable.

**Proof.** Let  $A = \{a_0, a_1, \dots, a_n\}$ . Output the words of length  $0, 1, 2, \dots$ . We can do this in lexicographic order.

*Example.* PRIME?, SAT?, CYCLE? (in a graph), NEG-WT-CYCLE? etc. are all decidable.

**Proof.** We gave algorithms for them and algorithms are ‘procedures’.

**Theorem 1** *Every decidable set is enumerable.*

**Proof.** Let  $W \subseteq A^*$  be a decidable set with decision procedure  $P$ . Enumerate as follows: enumerate  $A^*$  separately and output  $x \in A^*$  iff  $x \in W$ , which can be checked using  $P$ .

**Theorem 2**  *$W \subseteq A^*$  is decidable iff  $W$  and  $A^* \setminus W$  are enumerable.*

**Proof.** ( $\Rightarrow$ ) If  $W$  is decidable, say by procedure  $P$ , then we can decide  $A^* \setminus W$  using  $P$  and inverting its output. By Theorem 1,  $W$  and  $A^* \setminus W$  are enumerable.

( $\Leftarrow$ ) Assume that  $P$  and  $P'$  are enumeration procedures for  $W$  and  $A^* \setminus W$  respectively. Given  $x \in A^*$ , start the enumerations  $P$  and  $P'$ . At some point, one of  $P$  or  $P'$  will output  $x$ , and  $x \in W$  if  $P$  outputs  $x$  and  $x \notin W$  if  $P'$  outputs  $x$ . This gives a decision procedure for  $W$  (and  $A^* \setminus W$ ).

**Definition.** A function  $f : A^* \rightarrow A^*$  is *computable* iff there is a procedure which, given  $x \in A^*$ , returns  $f(x)$ .

*Example.* Length of a List, Modular Exponentiation, DFS tree, Sorting etc. all are computable functions.

**Theorem 3** *Let  $A$  be an alphabet,  $f : A^* \rightarrow A^*$  be a function, and  $\# \notin A$ . Then the following are equivalent.*

1.  $f$  is computable.
2.  $F = \{x\#f(x) \mid x \in A^*\}$  is decidable.
3.  $F$  is enumerable.

*Proof.* (2  $\Rightarrow$  3): Theorem 1

(3  $\Rightarrow$  1): Given  $x \in A^*$ , enumerate  $F$ . Eventually  $x\#f(x)$  will appear. Output  $f(x)$ .

(1  $\Rightarrow$  2) For input  $x\#y$ , compute  $f(x)$ . Now  $x\#y \in F$  iff  $f(x) = y$ .

**Register Machine:** A register machine is a model of computation. It consists of an arbitrary number of registers, where each register can store a word, an element of  $A^*$  (where  $A = \{a_0, a_1, \dots, a_r\}$  is the alphabet). The instruction set of the register machine has five types of instructions:

1.  $L : \text{let } R_i = R_i + a_j.$   
Here  $L \in \mathbb{N}$  is the *label* of the instruction and  $i, j \in \mathbb{N}$ . The semantics of this instruction is: Append the symbol  $a_j$  to the end of the string in register  $R_i$ .
2.  $L : \text{let } R_i = R_i - a_j.$   
The semantics is: if the string in  $R_i$  ends with the symbol  $a_j$  then remove the symbol, else skip.
3.  $L : \text{Print}.$   
The semantics is: Print the word in  $R_0$ .
4.  $L : \text{Halt}.$   
The semantics is: Halt the machine.
5.  $L : \text{If } R_i = \square \text{ then } L' \text{ else } L_0 \text{ or } L_1 \text{ or } \dots \text{ or } L_r.$   
The semantics is: if  $R_i$  is empty goto instruction  $L'$ . If  $R_i$  ends in  $a_k$  goto  $L_k$ , for  $k = 0, \dots, r$ .

**Definition.** A *register program* is a finite sequence of instructions  $\alpha_0, \alpha_1, \dots, \alpha_k$ , such that

- a.  $\alpha_i$  has label  $i$ .
- b. Jumps have labels  $\leq k$ .

c. Only  $\alpha_k$  is a halt instruction.

The starting instruction for the program is  $\alpha_0$ . All registers are initially empty except  $R_0$  which contains the input.

What can we say about the power of such register machines? Clearly, each register program gives rise to a procedure. However, there is a famous thesis of Church and Turing regarding the power of register machines and other equivalent models of computation like Turing machines, the lambda calculus, general recursive functions, etc. The Church-Turing thesis says that no physically realizable notion of computation is more powerful than a register machine, or a Turing machine, or the lambda calculus, or general recursive functions, etc. While there cannot be any proof of this thesis, about 70 years of research supports this claim.

## Lecture 2

In the previous lecture we saw the definition of a register machine and register programs. We begin with an example.

Example 1: Consider a register machine over the alphabet  $A = \{1\}$ . Lets write a Program to check if the input in  $R_0$  has even number of 1's or not.

```
0 if  $R_0 = \square$  then 6 else 1
1  $R_0 = R_0 - 1$ 
2 if  $R_0 = \square$  then 5 else 3
3  $R_0 = R_0 - 1$ 
4 goto 0 (i.e. if  $R_0 = \square$  then 0 else 0)
5  $R_0 = R_0 + 1$ 
6 Print
7 Halt
```

From now on, we will assume that any algorithm, which we can give with pseudo code, can be turned into a register program.

Some Notations,

**P: X** → **halt** - means register program P started with input X in  $R_0$  eventually halts.

**P: X**  $\rightarrow \infty$  - means register program P started with input X in  $R_0$  never halts.

**P: X**  $\rightarrow y$  - means register program P started with input X in  $R_0$  halts and outputs only y.

Notice that, P(as given in example 1): X  $\rightarrow$  halt.

Also P'(as defined below): X  $\rightarrow \infty$ .

```
P' = 0 goto 0
      1 Halt
```

As before, for  $W \subseteq A^*$ , we define

- a. P enumerates W, if P on input  $\square$ , prints all words in W.
- b. W is R-enumerable, if  $\exists P$  s.t. P enumerates W.
- c. P decides W if  $\forall x \in W$ , P:  $x \rightarrow \square$  and P:  $x \rightarrow y$  ( $y \neq \square$ ) otherwise.
- d. W is R-decidable if  $\exists P$  s.t. P decides W.
- e. for  $f : A^* \rightarrow A^*$ , P computes f if and only if  $\forall x \in A^*$ , P:  $x \rightarrow f(x)$ .
- f. f is R-computable if  $\exists P$  which computes f.

## Limits of Decidability

Are there sets that are not enumerable ? Are there functions that are not computable ? YES! we will give simple cardinality argument.

**Theorem 4** *There exists functions which are not computable. Also there exists sets which are not enumerable.*

**Proof:** Consider all the functions from  $\mathbb{N} \rightarrow \{0, 1\}$ . Notice that there are uncountably many such functions. But how many programs exists? Only countably many! (They are enumerable, as we've seen.) Therefore there exists functions which cannot be computed by any program.

Here is another proof, allowing us to introduce a powerful technique known as *diagonalization*. Suppose that there are a countable number of functions; then the functions can be put into the following tabular form.

	0	1	2	3	4	.	.
$f_0$	<u>1</u>	0	0	1	0	.	.
$f_1$	0	<u>0</u>	0	1	0	.	.
$f_2$	1	1	<u>1</u>	1	1	.	.
$f_3$	1	0	0	<u>0</u>	1	.	.
$f_4$	0	0	1	0	<u>0</u>	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Each entry  $x_{ij}$  of the table represents the value  $f_i(j)$ . Let's define a function  $g$  s.t.  $g(i) = 1 - f_i(i)$ . Now function  $g$  is also in the list, therefore  $g = f_n$  for some  $n$ . But  $g(n) = 1 - f_n(n)$  and  $g(n) = f_n(n)$ , a contradiction.

### Lecture 3

Recall that we can totally order the programs over  $A = \{a_0, \dots, a_r\}$ , e.g., lexicographically.

Suppose program  $P$  is the  $n$ th program in the ordering, then we can define  $g_P := a_0 a_0 \dots a_0$  (a string of  $n$   $a_0$ 's). This numbering is an example of Gödel numbering.

Let us define the language **HALT'** :=  $\{g_P | P : g_P \rightarrow \text{halt}\}$ . Note that **HALT'** corresponds to a subset of  $\mathbb{N}$ .

**Theorem 5** *The language **HALT'** :=  $\{g_P | P : g_P \rightarrow \text{halt}\}$  is not decidable.*

**Proof:** By diagonalization. Suppose **HALT'** is decidable. Then  $\exists P_n$  which decides **HALT'**. That is, the following holds  $\forall P$

$$\begin{aligned} P : g_P \rightarrow \text{halts} &\implies P_n : g_P \rightarrow \square \\ P : g_P \rightarrow \infty &\implies P_n : g_P \rightarrow a_0 \end{aligned}$$

But from  $P_n$ , we can construct  $P_m$ , so that

$$\begin{aligned} P : g_P \rightarrow \text{halts} &\implies P_m : g_P \rightarrow \infty \\ P : g_P \rightarrow \infty &\implies P_m : g_P \rightarrow \text{halts} \end{aligned}$$

In other words

$$P : g_P \rightarrow \text{halts} \iff P_m : g_P \rightarrow \infty$$

Note that the above holds for any  $P$ . But see what happens when we instantiate  $P$  with  $P_m$  (diagonalization!). We get

$$P_m : g_{P_m} \rightarrow \text{halts} \iff P_m : g_{P_m} \rightarrow \infty$$

This is a contradiction. So our initial assumption that **HALT'** must be decidable was wrong. The proof will be complete once we mention how to construct  $P_m$  from  $P_n$ . First, we can assume that  $P_n$  prints an answer and halts, therefore, we construct  $P_m$  so that when  $P_n$  prints  $\square$ ,  $P_m$  does not halt, otherwise it does halt. This is accomplished by replacing the only **halt** instruction in  $P_n$ , which we assume is the  $k$ th instruction by:

```

k if  $R_0 = \square$  then  $k$  else  $k + 1$ 
k+1 halt

```

■

To see in more detail why this is diagonalization, consider the following table.

	0	1	2	3	4	5
	$\square$	$a_0$	$a_0a_0$	$a_0a_0a_0$	$a_0a_0a_0a_0$	$a_0a_0a_0a_0a_0$
$P_0$	H	$\infty$	$\infty$	H	H	H
$P_1$	$\infty$	$\infty$	H	$\infty$	H	H
$P_2$	H	H	$\infty$	$\infty$	H	$\infty$
$P_3$	H	$\infty$	$\infty$	$\infty$	$\infty$	H
$P_4$	H	H	$\infty$	H	$\infty$	H
$P_5$	$\infty$	$\infty$	H	$\infty$	H	H
$P_6$	$\infty$	$\infty$	$\infty$	H	$\infty$	$\infty$

Notice that program  $P_m$ , by construction, differs from every program along the diagonal of the above table. This leads to a contradiction.

Now we consider the classic Halting Problem.

$$\mathbf{HALT} := \{g_P | P : \square \rightarrow \text{halts}\}$$

**Theorem 6** *The language **HALT** is undecidable.*

**Proof:** By reduction from **HALT'**: we will show that if **HALT** is decidable, then so is **HALT'**, a contradiction. Therefore, **HALT** is undecidable.

Given program  $P$ , we construct  $P^+$  from  $P$ , as follows. Program  $P^+$  is as follows.

0	Let $R_0 = R_0 + a_0$
1	Let $R_0 = R_0 + a_0$
2	Let $R_0 = R_0 + a_0$
.	
.	
.	
$ g_P  - 1$	Let $R_0 = R_0 + a_0$
	P with all labels $l$ replaced by $l +  g_P $

Notice that  $P^+$  writes the Gödel number  $g_P$  in register  $R_0$ , and then behaves just program  $P$ . Therefore:

$$P : g_P \rightarrow \text{halts} \iff P^+ : \square \rightarrow \text{halts}$$

By the definition of **HALT** and **HALT'**, we can see that

$$P \in \mathbf{HALT}' \iff P^+ \in \mathbf{HALT}$$

Therefore, if **HALT** is decidable, then so is **HALT'**. But, since **HALT'** is not decidable, neither is **HALT**. (This is an example of reduction.) ■

**Theorem 7** ***HALT**, **HALT'** are enumerable.*

**Proof:** We want a program which prints out the Gödel number of all the programs which halt. We cannot simply test programs for termination, because if a program does not halt, we cannot determine this by running it. Similarly, we cannot cycle through all the programs running each one a finite number of steps, because there are an infinite number of programs.

We use the following clever idea. We go through rounds, where in each round  $i$ , we run the first  $i$  programs for  $i$  steps. So if program 1,000,000 halts in a billion steps, we will find that out in the billionth round, and will print its Gödel number. If some program does not halt, we will never write it down. ■

**Theorem 8** *The set  $\{P|g_P : P \notin \mathbf{HALT}\}$  is not enumerable.*

**Proof:** We have already shown that **HALT** is enumerable, but not decidable. Note that the above set is the complement of **HALT**. But if both **HALT** and its complement are enumerable, then, as we have seen, both are decidable, a contradiction. ■