E0 224 Computational Complexity Theory **Indian Institute of Science, Bangalore**
Fall 2014 **Department of Computer Science and Automation**

Lecture 2: Aug 11, 2014

Lecturer: Chandan Saha <chandan@csa.iisc.ernet.in> *Scribe: Bibaswan Kumar Chatterjee*

2.1 Introduction

To study the various questions of computation and its efficiency, there has been a need to define a formal mathematical model that captures our intuitive notion of computation. We say that a function is computable if we can apply a finite set of mathematical rules, that depends on the function being computed but not on the input size of the function, to output the value of the function for a given input using a piece of paper to scribe on.

When we say functions, we mean Boolean functions, i.e. $f : \{0,1\}^* \to \{0,1\}$ Even if we need to compute a function $g : \{0,1\}^* \to \{0,1\}^*$, we can break into multiple Boolean functions g_i where each $g_i : \{0,1\}^* \to \{0,1\}$ is a Boolean function that computes the i^{th} bit of the output string.

2.2 Turing Machines

2.2.1 Introduction

A Turing Machine is a formal mathematical model that captures our notion of computation.

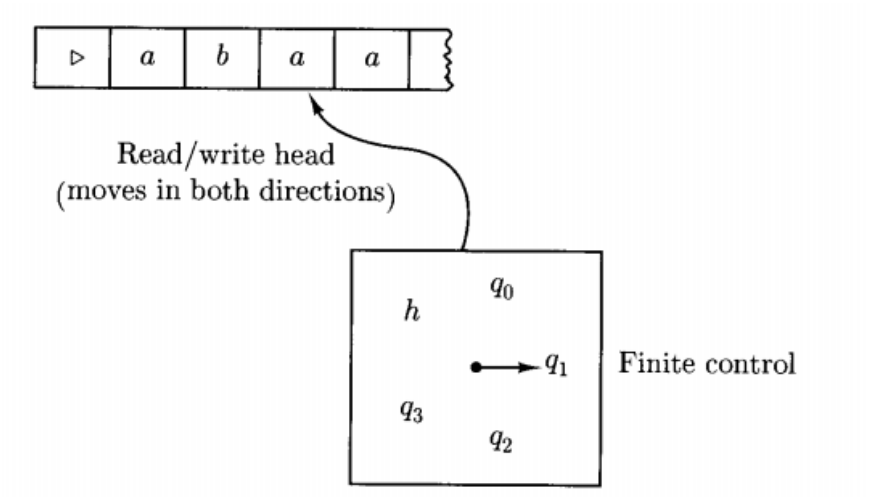

Figure 2.1: A Turing Machine [M2]

In the definition we follow, a Turing machine can be informally thought of as a hypothetical machine consisting of a single unidirectional infinite tape of cells, a read/write head which can read or write a character on a cell of the tape and can move in either direction left or right at a time, a finite set of states and a transition table. The transition table consists of a set of rules which allow the Turing machine to compute a Boolean function f(x) when the input x is supplied on the infinite tape and it is started. Each rule involves a set of elementary operations where the Turing machine reads a single character from a cell in the tape and based on the character writes a character on the cell and move the read/write head one cell to left or right. This continues till the machine ends computing f(x) and writes either 0 or 1 on the tape and halts.

We now give a formal definition of a Turing Machine.

Definition 2.1. *A Turing Machine M is described by a 3-tuple* $< \Gamma, Q, \delta >$ *where*
$\Gamma =$ *a finite set of alphabets,*
$Q =$ *a finite set of states,*
$\delta : \Gamma \times Q \to \Gamma \times Q \times \{L, R\}$ *is a mapping which says if M is in state* $q \in Q$ *and the symbol* $\sigma \in \Gamma$ *is being read from the tape, and* $\delta(q, \sigma) = (q', \sigma', z)$ *where* $z \in \{L, R\}$ *then at the next step the symbol* σ *will be replaced by* σ' *and M will move to state* q' *and the read/write head will move Left or Right as given by* z.
M has two special states q_{start} *and* q_{halt}*, M starts computation from state* q_{start} *and M is said to have halted if it has reached state* q_{halt}.

The above definition of a Turing Machine can be tweaked in various ways by say considering k tapes instead of only one tape or by considering a tape where each cell is indexed and can be accessed randomly. But it can be easily verified that these changes to the definition do not yield a substantially different model as the above model can simulate any of these new models. As long as we are wiling to ignore polynomial factors in the running time of the Turing Machine, the above model is as powerful as any other model we can come up with by tweaking the formal definition.

2.3 Computing with Turing Machine

2.3.1 How does a Turing Machine compute a function? : An Example

To illustrate how a Turing Machine computes a Boolean function, let us define a Boolean function $f : \{0,1\}^* \to \{0,1\}$ where $f(x) = 1$ if x is *odd* else 0. We construct a Turing Machine to compute $f(x)$ as follows:
$M =< \Gamma, Q, \delta >$ where
$\Gamma = \{\$, \#, 0, 1\}$
$Q = \{q_{start}, q_{halt}, q_{right}, q_{test}, q_0, q_1\}$
δ is defined by the table below:

IF	THEN
$q_{start}, \$$	$q_{right}, \$, R$
$q_{right}, 1$	$q_{right}, 1, R$
$q_{right}, 0$	$q_{right}, 0, R$
$q_{right}, \$$	$q_{test}, \$, L$
$q_{test}, 0$	$q_0, 0, R$
$q_{test}, 1$	$q_1, 1, R$
$q_0, \$$	$q_0, \$, R$
$q_1, \$$	$q_1, \$, R$
$q_0, \#$	$q_{halt}, 0, R$
$q_1, \#$	$q_{halt}, 1, R$

Here input has to be supplied on the tape in the form $\$x\$\#$. $\$$ and $\#$ are special symbols such that $\$$ marks the beginning and end of input and $\#$ marks the position after the input where the output symbol will be written. Once M starts with state q_{start} it switches to q_{right}. The state q_{right} is responsible for traversing to the end of the input string. Once the end of the input is reached, M switches to the state q_{test} which checks the last bit of x. If the last bit is found to be 1, then that information is remembered by switching to state q_1 else it switches to state q_0. Based on whether the state is q_1 or q_0, the corresponding output symbol is written over $\#$ and M halts. For example if M is supplied with input $\$100110\$\#$ it will halt with output $\$100110\$0$ on the tape.

Definition 2.2. *Let $f : \{0,1\}^* \to \{0,1\}^*$*
We say that a Turing Machine M computes f if on every input $x \in \{0,1\}^*$, $M(x) = f(x)$ where $M(x)$ is the content on the tape of M after M has reached q_{halt}.

A Boolean function $f : \{0,1\}^* \to \{0,1\}$ can be viewed as a subset of $\{0,1\}^*$ where the subset $S = \{x | x \in \{0,1\}^*$ and $f(x) = 1\}$. Thus we can define a language $L \subseteq \{0,1\}^*$ by a Boolean function f as $L = \{x | x \in \{0,1\}^*$ and $f(x) = 1\}$.

Thus using the computability of Boolean functions by a Turing Machine we can say whether a Turing Machine *accepts* the strings of a language.

Definition 2.3. *We say that a Turing Machine M decides a language $L \subseteq \{0,1\}^*$, if on every input $x \in \{0,1\}^*$, M halts with $M(x) = 1$ if $x \in L$ otherwise it halts with $M(x) = 0$.*

Definition 2.4. *Let $T : \mathbb{N} \to \mathbb{N}$ and $f : \{0,1\}^* \to \{0,1\}^*$. We say that A Turing Machine M computes f in time $T(n)$ if on every input $x \in \{0,1\}^*$, $M(x) = f(x)$ and number of basic operations of M on input x is bounded by $T(|x|)$.*

2.4 Properties of Turing Machine

2.4.1 Robustness of Definition

There are possible ways to tweak the definition of Turing Machine we gave above with the hope of making it more powerful. But we can show that such changes to the definition imparts no more power to a Turing Machine than the one defined above except changing the the running time by polynomial factor in n. For example we can modify the above definition to consider Turing Machines as follows:

1. **Increase the alphabet size**
   Say we define a Turing Machine whose alphabet set we restrict only to the set $\Gamma' = \{0, 1, \$, \#\}$ instead of Γ ($\Gamma' \subseteq \Gamma$). It can be shown that for every $f : \{0,1\}^* \to \{0,1\}$ and $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a Turing Machine M using alphabet Γ, it is computable in time $4log|\Gamma|T(n)$ by a Turing Machine $\overline{M}$ using alphabet Γ'.

2. **Use k tapes instead of one**
   Suppose we consider a definition of Turing Machine $\overline{M}$ which has k infinite tapes instead of one. Out of these k tapes, one is for input, one is for output and the rest $k-2$ tapes are 'work' tapes. Again it can be shown that, for every $f : \{0,1\}^* \to \{0,1\}$ and $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a Turing Machine $\overline{M}$ using k tapes, then it is computable in time $5kT(n)^2$ in a single-tape Turing Machine M as defined above.

3. **Use one bi-directional infinite tape instead of unidirectional**
   Suppose we consider Turing Machines which have bi-directional infinite tape instead of unidirectional infinite tape as considered in our definition. Again we can show that, for every $f : \{0,1\}^* \to \{0,1\}$ and $T : \mathbb{N} \to \mathbb{N}$, if f is computable in time $T(n)$ by a Turing Machine $\overline{M}$ using a bi-directional infinite tape, then it is computable in time $4T(n)$ in a Turing Machine M using a single unidirectional infinite tape.

2.4.2 Turing Machines as strings

We can represent a Turing Machine as a string. This is possible as we can encode the description of a Turing Machine as a sequence of zeros and ones. Specifically, if we encode the list of inputs and outputs of the transition function of a Turing Machine following some convention, then we are done as there will be enough information to simulate the Turing Machine from this encoding. Since this encoding is a binary string, this encoding can be given as an input to another Turing Machine. Due to the binary representation of a Turing Machine we automatically get the following properties:

1. **Every string $\alpha \in \{0,1\}^*$ represents some Turing Machine**
   For every $\alpha \in \{0,1\}^*$, α can either be the valid encoding of some Turing Machine, in which case α will represent the Turing Machine M_α otherwise, it is not a valid encoding of a Turing Machine, in which case we can map it to some trivial Turing Machine M^* which halts immediately and outputs 0 on any input.
2. **Every Turing Machine is represented by infinitely many strings**
   This is possible if we allow our encoding to end with say arbitrary number of ones or arbitary numer of zeroes. Thus we can add arbitrary redundant information to a Turing Machine encoding to make infinitely many representations of the same Turing Machine. This is somewhat similar to adding comments to a program in any programming language.

2.4.3 Universal Turing Machine

There exists a Turing Machine U that simulate any Turing Machine M on any input x, if it is supplied with the encoding of M and x as input. This might sound counter-intuitive as U can have fixed alphabet size and number of states and the Turing Machine to be simulated can have arbitrarily larger number of states and alphabet size. But it is still possible as we can encode any number of states or characters by binary strings that can be easily understood by the Universal Turing Machine U.
Infact it can be shown that, there exists a construction of U such that, for every $x, \alpha \in \{0,1\}^*$, $U(x,\alpha) = M_\alpha(x)$, where M_α denotes the Turing Machine represented by α. Moreover, if M_α halts on input x within T steps then $U(x,\alpha)$ halts within $CTlogT$ steps, where C is a number independent of $|x|$ and depending only on the size of M_α's alphabet size, number of tapes, and number of states.

In context of mordern day computing, a Universal Turing Machine is like an interpreter for a programming language written in the same language.

2.4.4 Uncomputability

There exists functions that cannot be computed by a Turing Machine. Infact there exists uncomputable Boolean functions. Since Boolean functions also define languages, such languages corresponding to uncomputable Boolean functions are called *undecidable* languages. We will prove the existence of one such language.

Theorem 2.5. *Let L be a language $L = \{\alpha \in \{0,1\}^* : M_\alpha(\alpha) = 1\}$. Let $L_{uc} = \overline{L} = \{0,1\}^* \setminus L$. L_{uc} is undecidable.*

Proof. The language L_{uc} is defined such that for every $\alpha \in \{0,1\}^*$ if $\alpha \in L_{uc}$ then either $M_\alpha(\alpha) = 0$ or $M_\alpha(\alpha)$ enters an infinite loop.
Now, suppose for sake of contradiction, there exists a Turing Machine M that decides L_{uc}. Then for every $\beta \in \{0,1\}^*$, $M(\beta) = 1 \rightarrow \beta \in L_{uc}$ and $M(\beta) = 0 \rightarrow \beta \notin L_{uc}$.
Now let $\lfloor M \rfloor$ be the encoding of M. Then if we give $\lfloor M \rfloor$ as input to M, we get a contradiction as if $M(\lfloor M \rfloor) = 1$ then $\lfloor M \rfloor \in L_{uc}$ but by definition of L_{uc}, $\lfloor M \rfloor \in L_{uc} \rightarrow M(\lfloor M \rfloor) = 0$ or $M(\lfloor M \rfloor)$ enters an infinite loop. Similarly,

if $M(\lfloor M \rfloor) = 0$ then $\lfloor M \rfloor \notin L_{uc}$ but by definition of L_{uc}, $\lfloor M \rfloor \notin L_{uc} \rightarrow M(\lfloor M \rfloor) = 1$. Thus M cannot exist. So the language L_{uc} is *undecidable*. □

References

[M2] H.R. LEWIS and C.H. PAPADIMITRIOU, "Elements of The Theory of Computation," *Prentice-Hall*, 1998

[M1] S. ARORA and B. BARAK "Computational Complexity: A Mordern Approach," *Cambridge University Press*, 2009