

CSL853: Complexity Theory

Ragesh Jaiswal, CSE, IIT Delhi

Turing Machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain.
 $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

Modeling Computation: Turing machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain. $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

Starting configuration:

- The input tape has $\triangleright$ in the leftmost tape cell, followed by the input, and followed by blank symbols everywhere else.
- All work/output tapes have $\triangleright$ in the leftmost tape cell and blank symbols everywhere else.
- All tape heads are at the left end of their respective tapes.
- The state of M is q_{start} .

Modeling Computation: Turing machines

k -tape Turing Machine

A k -tape Turing machine M may be described by a 3-tuple, (Γ, Q, δ) consisting of:

- A set Γ of the symbols that M 's tapes can contain. $\Gamma = \{\triangleright, \square, 0, 1\}$ and these are called *alphabets* of M . $\triangleright$ is called the *start* symbol and $\square$ is called the *blank* symbol.
- A set Q of M 's states. (Alternatively the set of states that M 's register can be in). Q contains a designated start state q_{start} and a designated halt state q_{halt} .
- A function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ describing the transition rule. δ is called the *transition function*.

- Starting configuration:

- The input tape has $\triangleright$ in the leftmost tape cell, followed by the input, and followed by blank symbols everywhere else.
- All work/output tapes have $\triangleright$ in the leftmost tape cell and blank symbols everywhere else.
- All tape heads are at the left end of their respective tapes.
- The state of M is q_{start} .

- Transition: Transitions are executed using function δ .

- Halting: M halts off it enters the state q_{halt} . Moreover, after it enters q_{halt} , δ makes sure that no further modification to the tape is done.

Modeling Computation: Turing machines

Definition (Computing a function and running time)

Let $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ and let $T : \mathbb{N} \rightarrow \mathbb{N}$ be some functions, and let M be a Turing machine. We say that M computes f in $T(n)$ -time if for every $x \in \{0, 1\}^*$, if M is initialized to the start configuration on input x , then after at most $T(|x|)$ steps it halts with $f(x)$ written on its output tape. We say that M computes f if it computes f in $T(n)$ time for some function $T : \mathbb{N} \rightarrow \mathbb{N}$.

Definition (Time constructible functions)

We say that a function $T : \mathbb{N} \rightarrow \mathbb{N}$ is time constructible if $T(n) \geq n$ and there is a TM M that computes the function $x \rightarrow \text{Bin}(T(|x|))$ in time $T(n)$. ($\text{Bin}(T(|x|))$ denotes the binary representation of the number $T(|x|)$.)

Modeling Computation: Turing machines

- Consider the following function $PAL : \{0, 1\}^* \rightarrow \{0, 1\}$:

$$PAL(x) = \begin{cases} 1 & \text{if } x \text{ is a palindrome} \\ 0 & \text{otherwise} \end{cases}$$

Theorem

There is a TM M that computes the function PAL in $(3n)$ -time.

Modeling Computation: Turing machines

Alternate definitions

- There are other definitions of Turing Machine. So, why does the definition does not matter much as far our study of computation is concerned?
- For any function f and any TM M , as per a different definition, that computes f , there is a TM M' (as per our definition) that computes f *almost as efficiently* as M .
- Examples:

Theorem

For every $f : \{0, 1\}^* \rightarrow \{0, 1\}$ and time-constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using alphabet Γ then it is computable in time $4 \log |\Gamma| T(n)$ by a TM M' using the alphabet $\{\triangleright, \square, 0, 1\}$.

Modeling Computation: Turing machines

Alternate definitions

Theorem

For every $f : \{0, 1\}^* \rightarrow \{0, 1\}$ and time-constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using alphabet Γ then it is computable in time $4 \log |\Gamma| T(n)$ by a TM M' using the alphabet $\{\triangleright, \square, 0, 1\}$.

Theorem

For every $f : \{0, 1\}^* \rightarrow \{0, 1\}$, time-constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a TM M using k tapes (plus additional input and output tapes) then it is computable in time $5kT(n)^2$ by a TM M' using only a single work tape (plus additional input and output tapes).

Modeling Computation: Turing machines

Alternate definitions

Theorem

Define a bidirectional TM to be a TM whose tapes are infinite in both directions. For every $f : \{0,1\}^* \rightarrow \{0,1\}^*$ and time constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a bidirectional TM M then it is computable in time $4T(n)$ by a standard (unidirectional) TM M' .

Modeling Computation: Turing machines

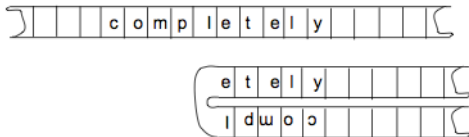
Alternate definitions

Theorem

Define a *bidirectional TM* to be a TM whose tapes are infinite in both directions. For every $f : \{0,1\}^* \rightarrow \{0,1\}^*$ and time constructible $T : \mathbb{N} \rightarrow \mathbb{N}$, if f is computable in time $T(n)$ by a bidirectional TM M then it is computable in time $4T(n)$ by a standard (unidirectional) TM M' .

Source: Arora-Barak

M 's tape is infinite in both directions:



$\tilde{M}$ uses a larger alphabet to represent it on a standard tape:



Modeling Computation: Turing machines

Universal Turing Machine

- Claim 1: A turing machine may be encoded into a string in $\{0, 1\}^*$.
- Claim 2: There is an encoding such that every string in $\{0, 1\}^*$ represents some Turing machine.

Modeling Computation: Turing machines

Universal Turing Machine

Theorem (Efficient Universal Turing Machine)

There exists a TM U such that for every $x, \alpha \in \{0, 1\}^*$, $U(x, \alpha) = M_\alpha(x)$, where M_α denotes the TM represented by α .

Furthermore, if M_α halts on input x within T steps then $U(x, \alpha)$ halts within $CT \log T$ steps, where C is a number independent of $|x|$ and depending only on M_α 's alphabet size, number of tapes, and number of states.

- Are there functions that are not computable?

Modeling Computation: Turing machines

Recursively enumerable and recursive

Definition (Recursively enumerable)

A language L is said to be *Recursively Enumerable* if there is a TM M such that $L(M) = L$.

Definition (Recursive)

A language L is said to be *recursive* or *decidable* if there is a TM M such that for every $x \in L$, $M(x)$ halts and accepts and for every $x \notin L$, $M(x)$ halts and rejects.

- Are there functions that are not recursively enumerable?
- Are there functions that are recursively enumerable but not decidable?

Modeling Computation: Turing machines

Recursively enumerable and recursive

Theorem

Consider the following function $UC : \{0, 1\}^* \rightarrow \{0, 1\}$:

$$UC(\alpha) = \begin{cases} 0 & \text{if } M_\alpha(\alpha) \text{ halts in a finite number of steps} \\ & \text{and outputs 1} \\ 1 & \text{otherwise} \end{cases}$$

L_{UC} is not recursively enumerable.

	0	1	00	01	10	11	...	α	...
0	0	1	*	0	1	0		$M_0(\alpha)$	
1	1	*	0	1	*	1		...	
00	*	0	1	0	1	*			
01	1	*	0	0	*	0			
...									
α	$M_\alpha(0)$	...						$M_\alpha(\alpha) = 1 - M_\alpha(\alpha)$	
...									

Source: Arora-Barak

Modeling Computation: Turing machines

Recursively enumerable and recursive

Theorem

Consider the following function $UC : \{0,1\}^* \rightarrow \{0,1\}$:

$$UC(\alpha) = \begin{cases} 0 & \text{if } M_\alpha(\alpha) \text{ halts in a finite number of steps} \\ & \text{and outputs 1} \\ 1 & \text{otherwise} \end{cases}$$

L_{UC} is not recursively enumerable.

Theorem

Consider the following function $HALT$:

$$HALT(x, \alpha) = \begin{cases} 1 & \text{if } M_\alpha(x) \text{ halts in a finite number of steps} \\ 0 & \text{otherwise} \end{cases}$$

L_{HALT} is not decidable.

Modeling Computation: Turing machines

Recursively enumerable and recursive

- L_{UC} is not recursively enumerable.
- L_{HALT} is recursively enumerable but not decidable.

Theorem (Rice's Theorem)

Every non-trivial property of recursively enumerable languages is undecidable.

End