CS 6810: Theory of Computing **Fall 2021**

Lecture 2: Aug 31, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Oscar So*

1 Introduction

This lecture focuses on the Universal Turing Machine, NP, NP-Completeness, and Reductions.

2 Universal Turing Machines (UTM)

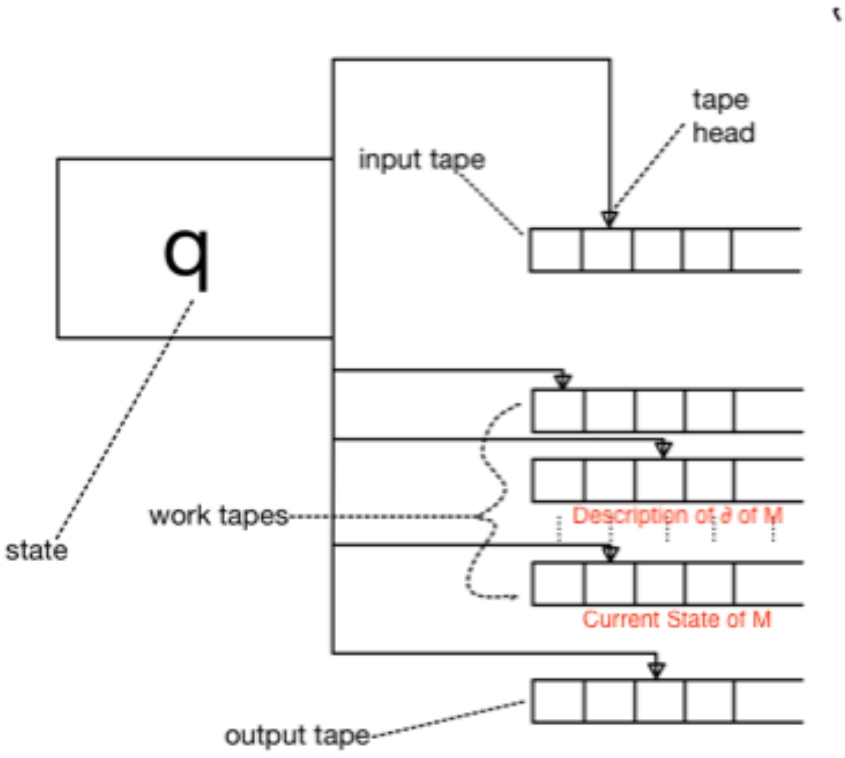
A Universal Turing machine (UTM) is a Turing machine that simulates an arbitrary Turing machine on a given input. The key observation is that one can encode the description of a Turing Machine M by a finite length binary string $\lfloor M \rfloor \in \{0,1\}^*$ that represent the encodes information about the state space, alphabet, and transition function of the TM. A UTM $\mathcal{U}$ essentially acts as an interpreter for these "Turing machine programs", and is given as input $< \lfloor M \rfloor, x >$, with the intention that $\mathcal{U}$ writes $M(x)$ on its output tape (or loops forever is M does not halt x).

For more about some useful properties of the encoding of TMs as finite bit strings, see Chapter 1.3 in the Arora-Barak book.

2.1 A sketch of the UTM construction

We will use $\lfloor M \rfloor$ to denote the binary representation of M's description.
Input: $(\lfloor M \rfloor, x)$
Output: $M(x)$
High level description of UTM: The main idea is the following: the UTM $\mathcal{U}$ uses one if it's work tapes to maintain the current state of the TN M, and stores the transition function of M on a different worktape. Now, we can always assume that M has one work tape (as we say in previous class a k-work tape machine can be simulated by a 1-work tape TM without much loss in efficiency). The UTM $\mathcal{U}$ uses its final work tape as the worktape of M. (Here we have to be careful, and first modify M so that its tape alphabet is modified to $\{0,1\}$ along with a few other standard symbols.)

If M runs in $T(n)$ time, one can implement the above sketched construction of UTM such that it runs in $O(T(n)\log(T(n)))$. Note that the construction of $\mathcal{U}$ takes care so that the number of states or tape alphabet of $\mathcal{U}$ does not depend on the input TM description to $\mathcal{U}$.



* Base Image of TM from Atul Ganju

3 Church-Turing Thesis

In computability theory, the Church–Turing thesis is a widely believed hypothesis about the nature of computable functions. It essentially states that any function (of the form $f : \{0,1\}^* \to \{0,1\}^*$) can be computed by a general model of computation if and only if it is computable by a Turing machine.

For example, this is witnessed by the fact that various design choices in the definition of a multi-tape TM (e.g., number of tapes, alphabet size, sequential vs random access to tape cells) do not change the set of computable functions.

4 Non-Deterministic Turing Machines (NDTM)

Informally, the biggest difference between a NDTM and a multi-tape Turing Machine is that an NDTM has 2 transition functions: δ_0 and δ_1. An NDTM can explore multiple computation paths simultaneously (i.e., at each computation step, the NDTM has the option to choose from any of it's transition functions to transition to the next computation step) and it will accept if and only if at least one of the computational paths accepts. It is formally defined as follows:

Definition 4.1 (NDTM). *An NDTM (similar to a k-tape Turing Machine) for $k \geq 3$ is defined as follows.*

- *The first tape is the input tape.*

- *The second tape is the output tape.*
- *Any other remaining tapes ($k-2$) are work tapes.*
- Γ *is the tape alphabet.*
- Σ *is the input alphabet.*
- Q *is the finite set of internal states.*
- $\forall i$, $\delta_i : Q \times \Gamma^k \times \Sigma \to Q \times \Gamma^k \times \Sigma \times \{L, S, R\}^k$ *represents the state change, tape write, and tape movement (Left, Stay, Right).*
- *The TM starts at q_{start} and terminates at q_{halt}.*

Definition 4.2. *An NDTM N accepts input x if: $\exists$ a sequence of choices of δ_0 & δ_1, s.t. N can go from $q_{start} \to q_{halt}$, halt with 1 written on the output tape. Further, we say N runs in $T(n)$ time if: for all $n \in \mathbb{N}$, any input $x \in 0, 1^n$, and for all sequences of choices of the transition functions (δ_0, δ_1), N reaches q_{halt} after at most $T(n)$ steps.*

4.1 Time Complexities of TMs

Recall the definitions of the time complexity of deterministic and non-deterministic TMs. $DTIME(T(n))$ = languages computable by a TM in O(T(n)) time.

$$P := \bigcup_{c \geq 1} DTIME(n^c)$$

$NTIME(T(n))$ = languages computable by a NDTM in O(T(n)) time.

$$NP := \bigcup_{c \geq 1} NTIME(n^c)$$

A widely believed conjecture (and a major open question) in complexity theory is that $P \neq NP$.

5 Alternate Definition of NP

Definition 5.1 (NP Definition)**.** *$L \in NP$ if $\exists$ a poly-time TM (i.e. Verifier) V s.t. $x \in L$ iff $y \in \{0,1\}^{poly(|x|)}$ (i.e. Witness/Certificate) s.t. $V(x, y) = 1$.*

Definition 5.2 (More Concise NP Definition)**.** *$L \in NP$ if $\exists$ a poly-time TM (i.e. Verifier) V and a constant c s.t. $x \in L$ iff $y \in \{0,1\}^{c*|x|^c}$ (i.e. Witness/Certificate) s.t. $V(x, y) = 1$.*

Claim 5.3. *NP defined in 4.1.2 (NP) is equivalent to NP defined in 6.2 (NP*). (ie. NP = NP*)*

Proof. We want to show that $L \in$ NP $\Rightarrow L$ is decided by an NDTM running in polynomial time. In other words, we want to show that NP $\subseteq$ NP*.

- By definition of NP, $\exists$ a verifier V for L in running time of $O(n^c)$ for some fixed constant c.
- We then define an NDTM N with the following:
  1. Nondeterministically guess a certificate/string $\{0,1\}^*$ s that has a length $\leq O(n^c)$.
  2. Run V on input (x, s). Accept if V accepts, else reject.

Now we want to show the other direction: If L is decided by an NDTM running in polynomial time, $L \in$ NP. In other words, we want to show that NP$^* \subseteq$ NP.

- By definition of NTIME, $\exists$ an NDTM N that decides L running in $O(n^c)$ for some fixed constant c.
- Define a verifier V that takes (x, s) where s is the witness and describes a sequence of computational path choices made by N.
- V simulates N on input (x, s) and checks whether each transition made is valid.
- If N accepts in the end, V accepts, else V rejects.

6 Reductions

The notion of reductions between computational problems is extremely useful to deduce relative hardness results. While we have been not very successful at understanding the absolute complexities of various problems, the use of reductions have led to a very successful theory of understanding relative hardness of such problems.

We now formally define a type of reductions that we will use.

Definition 6.1. *A language $A \subset \{0,1\}^*$ (polytime Karp or many-one) reduces to a language $B \subset \{0,1\}^*$ if $\exists$ a polytime computable function f, s.t. $x \in A \iff f(x) \in B$. We will denote this by $A \leq_p B$.*

Note that if $A \leq_p B$, and $B \in P$, then we have $A \in P$.