Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

Modeling Computation: The Essentials

Algorithms and Turing Machines

- Let f be a function that takes a string of bits, i.e., a member of the set $\{0,1\}^*$, and outputs either 0 or 1.
- An **algorithm**, or a **Turing machine**, for computing f is a set of mechanical rules, such that, by following them, we can compute $f(x)$, given any input $x \in \{0,1\}^*$.
- The set of rules being followed is fixed, i.e., the same rules must work for all possible inputs, though each rule may be applied arbitrarily many times.

Rules

- Each rule involves one or more of the following "elementary" operations:
  1. Read a bit of the input.
  2. Read a bit, or possibly a symbol from a slightly larger alphabet, say a digit in $\{0, \ldots, 9\}$, from the scratch pad, or working space, that the algorithm is allowed to use.

  Based on the values read:
  1. Write a bit/symbol to the scratch pad.
  2. Either stop and output 0 or 1, or choose a new rule from the set that will be applied next.

Running Time, Robustness and Encoding of Machines

- The **running time** of the algorithm is the number of these basic operations performed.
- A machine **runs in time** $T(n)$ if it performs at most $T(n)$ basic operations on inputs of length n.

Robustness

- The model is robust to almost any tweak in the definition, such as:
  - Changing the alphabet from $\{0, 1, \ldots, 9\}$ to $\{0, 1\}$;
  - Allowing multiple scratchpads;

    $\vdots$
- The most basic version of the model can simulate the most complicated version with at most polynomial (actually quadratic) slowdown.
- Thus, t steps on the complicated model can be simulated in $O(t^c)$ steps on the weaker model, where c is a constant depending only on the two models.

Encoding of Machines

- An algorithm (i.e., a machine) can be represented as a bit string once we decide on some canonical encoding.
- Thus an algorithm/machine can be viewed as a possible input to another algorithm.
- This blurs the boundary between input, software and hardware.
- We denote by M_α the machine whose representation as a bit string is α.

Universal Machines and Uncomputability

- There is a **universal Turing machine** U that can simulate any other Turing machine given its bit representation.
  - Given a pair of bit strings (x, α) as input, machine U simulates the behavior of M_α on input x.
  - This simulation is very efficient, in the sense that, if the running time of M_α is $T(|x|)$, then the running time of U is $\mathrm{O}(T(|x|) \log T(|x|))$.
- The existence of a universal machine U, together with the possibility of encoding Turing machines, can be used to show the existence of functions that are not computable by any Turing machine.

Subsection 2

The Turing Machine

Scratch Pad

- The k-**tape Turing machine** (**TM**) is a concrete realization of the informal model:
  - The scratch pad consists of k tapes.
    - A **tape** is an infinite one-directional line of cells, each of which can hold a symbol from a finite set Γ, called the **alphabet** of the machine.
    - Each tape is equipped with a **tape head** that can potentially read or write symbols to the tape one cell at a time.
    - The machine's computation is divided into discrete time steps, and the head can move left or right one cell in each step.
    - The first tape of the machine is designated as the **input tape**. The machine's head can only read symbols from that tape, but not write them, i.e., it is a read-only head.
    - The $k-1$ read-write tapes are called **work tapes**. The last work tape is designated as the **output tape** of the machine, on which it writes its final answer before halting its computation.
- There are variants of Turing machines with **random access memory**.
- Their computational powers are equivalent to the standard model.

Finite Set of Operations/Rules

- The machine has a finite set of **states**, denoted Q.
- The machine contains a "register" that can hold an element of Q.
- This is the "current state" of the machine.
- It determines its action at the next computational step:
  (1) Read the symbols in the cells directly under the k heads;
  (2) For the $k-1$ read-write tapes, replace each symbol with a new symbol (or do not change it by writing down the old symbol again);
  (3) Change the register to contain another state from the finite set Q (or do not change the state by choosing the old state again);
  (4) Move each head one cell to the left or to the right (or stay in place).
- Turing machines can be thought of as simplified versions of modern computers.
  - The machine's tapes correspond to a computer's memory;
  - The transition function and register correspond to a computer's CPU.
- However, it is best to think of Turing machines as simply a formal way to describe algorithms.

The Formal Definition of a Turing Machine

Definition (Turing Machine)

A **Turing Machine (TM)** M is described by a tuple (Γ, Q, δ) consisting of:

- A finite set Γ of the symbols that M's tapes can contain, including:
  - A designated "blank" symbol, denoted $\sqcup$;
  - A designated "start" symbol, denoted $\triangleright$;
  - The numbers 0 and 1.

  We call Γ the **alphabet** of M.
- A finite set Q of possible **states** M's register can be in, including:
  - A designated start state q_{start};
  - A designated halting state q_{halt}.
- A function $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$, where $k \geq 2$, describing the rules M uses in performing each step.

  This function is called the **transition function** of M.

Operation of a Turing Machine

- Suppose the following hold:
  - The machine is in state $q \in Q$;
  - The symbols currently being read in the k tapes are
    $$(\sigma_1, \sigma_2, \ldots, \sigma_k);$$
  - The transition function gives
    $$\delta(q, (\sigma_1, \ldots, \sigma_k)) = (q', (\sigma'_2, \ldots, \sigma'_k), z),$$
    where $z \in \{\mathtt{L}, \mathtt{S}, \mathtt{R}\}^k$.
- Then at the next step:
  - The σ symbols in the last $k-1$ tapes will be replaced by the σ' symbols;
  - The machine will be in state q',
  - The k heads will move Left, Right or Stay in place, as given by z. If the machine tries to move left from a leftmost position it stays put.

Start Configuration of a Turing Machine

- The following is the **start configuration of** M **on input** x:
  - All tapes except for the input are initialized:
    - In their first location to the start symbol $\triangleright$;
    - In all other locations to the blank symbol $\sqcup$.
  - The input tape contains:
    - The start symbol $\triangleright$;
    - The nonblank string x;
    - The blank symbol $\sqcup$ on the rest of its cells.
  - All heads start at the left ends of the tapes.
  - The machine is in the special starting state q_{start}.
- Once the machine is in q_{halt}, the transition function δ does not allow it to further modify the tape or change states.

Simulating a Programming Language Using TMs

- Any program written in any of the familiar programming languages, such as C or Java, has an equivalent Turing machine:
- First, programs in these programming languages can be translated (compiled) into an equivalent machine language program.

  It consists of a sequence of instructions of a few simple types, e.g.:

  (a) Read from memory into one of a finite number of registers;
  (b) Write a register's contents to memory;
  (c) Add the contents of two registers and store the result in a third;
  (d) Perform (c) but with other operations, such as multiplication instead of addition.

Simulating a Programming Language (Cont'd)

- All these operations can be easily simulated by a Turing machine.
  - The memory and registers can be implemented using the machine's tapes;
  - The instructions can be encoded by the machine's transition function.
- To simulate the computer's memory, a two-tape TM can use:
  - One tape for the simulated memory;
  - The other tape to do binary-to-unary conversion that allows it, for a number i in binary, to read or modify the ith location of its first tape.

Subsection 3

Efficiency and Running Time

Computing and Running Time

- Every nontrivial computational task requires at least reading the entire input.
- So we count the number of basic steps as a function of the input length.

Definition (Computing a Function and Running Time)

Let $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$ be some functions, and let M be a Turing machine.

- We say that M **computes** f if, for every $x \in \{0,1\}^*$, if M is initialized to the start configuration on input x, then it halts with $f(x)$ written on its output tape.
- We say that M **computes** f **in** $T(n)$**-time** if its computation on every input x requires at most $T(|x|)$ steps.

Time-Constructible Functions

- A function $T : \mathbb{N} \to \mathbb{N}$ is **time constructible** if $T(n) \geq n$ and there is a TM M that computes the function $x \mapsto \llcorner T(|x|) \lrcorner$ in time $T(n)$, where, as usual, $\llcorner T(|x|) \lrcorner$ denotes the binary representation of the number $T(|x|)$.

  Examples: The functions $n, n \log n, n^2, 2^n$ are time-constructible.
- Almost all functions encountered will be time constructible.
- We restrict attention to time bounds of this form.
- Allowing time bounds that are not time constructible can lead to anomalous results.
- The condition $T(n) \geq n$ allows the algorithm time to read its input.

Variations of the Turing Machine Model

- Most changes in the details to the definition of the Turing Machine model do not yield a substantially different model, in the sense that the model introduced can simulate any of these new models.
- In the context of computational complexity, we have to verify, not only that one model can simulate another, but also that it can do so efficiently.
- We state a few results of this type.
- The derived conclusion is that the exact model is unimportant if we are willing to ignore polynomial factors in the running time.
- Variations on the model include:
  - Restricting the alphabet Γ to be $\{0, 1, \sqcup, \triangleright\}$;
  - Restricting the machine to have a single work tape;
  - Allowing the tapes to be infinite in both directions.

Restricting the Alphabet Γ

Claim

Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be time-constructible. Suppose 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 $\widetilde{M}$ using alphabet $\{0, 1, \sqcup, \rhd\}$.

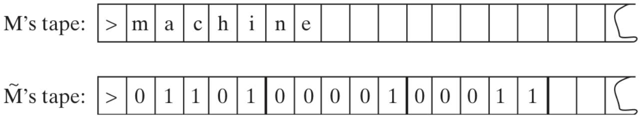
- Let M be a TM, with alphabet Γ, k tapes and state set Q, that computes the function f in $T(n)$ time.

  We describe an equivalent TM $\widetilde{M}$ computing f, with alphabet $\{0, 1, \sqcup, \rhd\}$, k tapes and a set Q' of states.

  The idea is that any member of Γ can be encoded using $\log |\Gamma|$ bits.

Restricting the Alphabet Γ (Cont'd)

- Each of $\widetilde{M}$'s work tapes will simply encode one of M's tapes. For every cell in M's tape we will have $\log|\Gamma|$ cells in the corresponding tape of $\widetilde{M}$.



We can simulate a machine M using the alphabet $\{\triangleright, \Box, \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$ by a machine M' using $\{\triangleright, \Box, 0, 1\}$ via encoding every tape cell of M using five cells of M'.

Restricting the Alphabet Γ (Cont'd)

- To simulate one step of M, the machine $\widetilde{M}$ will:
  (1) Use $\log|\Gamma|$ steps to read from each tape the $\log|\Gamma|$ bits encoding a symbol of Γ;
  (2) Use its state register to store the symbols read;
  (3) Use M's transition function to compute the symbols M writes and M's new state given the information gathered;
  (4) Store this information in its state register;
  (5) Use $\log|\Gamma|$ steps to write the encodings of these symbols on its tapes.

Restricting the Alphabet Γ (Cont'd)

- One can verify that the simulation can be carried out if $\widetilde{M}$ has access to registers that can store:
  - M's state;
  - k symbols in Γ;
  - A counter from 1 to $\log|\Gamma|$.

  Thus, there is such a machine $\widetilde{M}$ utilizing no more than $c|Q||\Gamma|^{k+1}$ states for some absolute constant c.

  We can show that, for every input $x \in \{0,1\}^n$, if on input x the TM M outputs $f(x)$ within $T(n)$ steps, then $\widetilde{M}$ will output the same value within less than $4\log|\Gamma|T(n)$ steps.

Restricting to a Single Work Tape

- Define a **single tape Turing machine** to be a TM that has only one read-write tape, that is used as input, work and output tape.
- We show that going from multiple tapes to a single tape can at most square the running time.

Claim

Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be time-constructible. Suppose f is computable in time $T(n)$ by a TM M using k tapes. Then it is computable in time

$$5kT(n)^2$$

by a single-tape TM $\widetilde{M}$.

Restricting to a Single Work Tape (Cont'd)

- The idea is for $\widetilde{M}$ to encode the k tapes of M on a single tape.

  $\widetilde{M}$ uses:
  - Locations $1, k+1, 2k+1, \ldots$ to encode the first tape;
  - Locations $2, k+2, 2k+2, \ldots$ to encode the second tape;

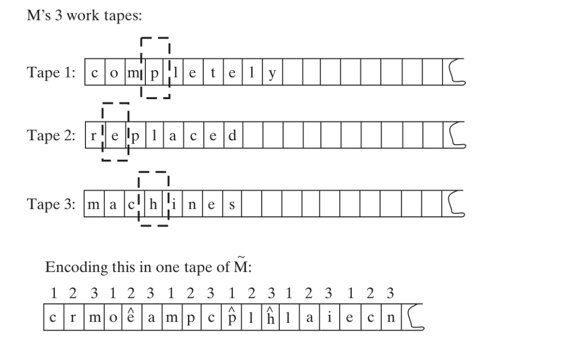
  $\vdots$

  The encoding is as follows.
  - For every symbol a in M's alphabet, $\widetilde{M}$ will contain both the symbol a and the symbol $\hat{a}$.
  - In the encoding of each tape, exactly one symbol will be of the ˆ type. This symbol indicates the position of the corresponding head of M.

Restricting to a Single Work Tape (Cont'd)

- $\widetilde{M}$ will not touch the first $n+1$ locations of its tape, where the input is located.

  It will, rather, start by taking $O\left(n^2\right)$ steps to copy the input bit by bit into the rest of the tape, while encoding it in the described way.



Simulating a machine M with three tapes using a machine $\tilde{M}$ with a single tape.

Restricting to a Single Work Tape (Cont'd)

- To simulate one step of M, $\widetilde{M}$ makes two sweeps of its work tape.
  - First, it sweeps the tape in the left-to-right direction and records to its register the k symbols that are marked by $\hat{}$;
  - Then $\widetilde{M}$ uses M's transition function to determine the new state, symbols and head movements;
  - Finally, it sweeps the tape back in the right-to-left direction to update the encoding accordingly.

Clearly, $\widetilde{M}$ will have the same output as M.

By hypothesis, on n-length inputs, M never reaches more than location $T(n)$ of any of its tapes.

So $\widetilde{M}$ will never need to reach more than location $2n + kT(n) \leq (k+2)T(n)$ of its work tape.

Thus, for each of the at most $T(n)$ steps of M, $\widetilde{M}$ performs at most $5 \cdot k \cdot T(n)$ work.

Oblivious Turing machines

- With a bit of care, one can ensure that the proof of the preceding claim yields a TM $\widetilde{M}$ with the following property.
  - Its head movements do not depend on the input but only depend on the input length: For every input $x \in \{0,1\}^*$ and $i \in \mathbb{N}$, the location of each of M's heads at the ith step of execution on input x is only a function of $|x|$ and i.
- A machine with this property is called **oblivious**.
- The fact that every TM can be simulated by an oblivious TM can be used to simplify some proofs in complexity.

Bidirectional Turing Machines

- Define a **bidirectional TM** to be a TM whose tapes are infinite in both directions.

Claim

Let $f : \{0,1\}^* \to \{0,1\}^*$ and let $T : \mathbb{N} \to \mathbb{N}$ be time-constructible. Suppose f is computable in time $T(n)$ by a bidirectional TM M. Then it is computable in time

$$4T(n)$$

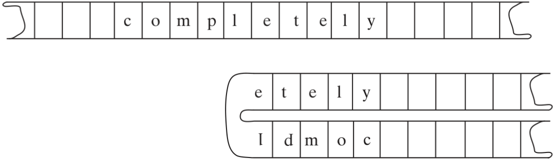
by a standard (unidirectional) TM $\widetilde{M}$.

- The idea is for $\widetilde{M}$ to use alphabet Γ^2, if M uses uses alphabet Γ.
  So $\widetilde{M}$'s alphabet symbols correspond to a pairs of symbols in M's alphabet.

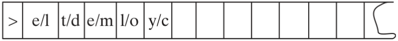
Bidirectional Turing Machines (Cont'd)

- A tape of M is "folded" in an arbitrary location.
  Each location of $\widetilde{M}$'s tape encodes two locations of M's tape.

M's tape is infinite in both directions:



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



To simulate a machine M with alphabet Γ that has tapes infinite in both directions, we use a machine $\tilde{M}$ with alphabet Γ^2 whose tapes encode the "folded" version of M's tapes.

Bidirectional Turing Machines (Cont'd)

- At first, $\widetilde{M}$ will ignore the second symbol in the cell it reads and act according to M's transition function.

  If this transition function instructs $\widetilde{M}$ to go "over the edge" of its tape, then it will start:
  - Ignoring the first symbol in each cell and use only the second symbol;
  - Interchanging left and right movements.

  If it needs to go over the edge again, then it will go back to:
  - Reading the first symbol of each cell;
  - Translating movements normally.

Subsection 4

Machines as Strings and the Universal Turing Machine

Representing a Turing Machine as a String

- We can represent a Turing machine as a string:
  - Write the description of the TM on paper;
  - Encode this description as a sequence of zeros and ones.
- This string can be given as input to another TM.
- The behavior of a TM is determined by its transition function.
- So we use the list of all inputs and outputs of this function as the encoding of the Turing machine.

Properties of the Representation Scheme

- We adopt a representation scheme that satisfies the following:
  1. Every string in $\{0,1\}^*$ represents some Turing machine.
     Strings that are not valid encodings represent some fixed trivial TM.
  2. Every TM is represented by infinitely many strings.
     The representation can end with an arbitrary number of 1s, that are ignored.
- We denote by $\llcorner M \lrcorner$ the TM M's representation as a binary string.
- If α is a string then M_α denotes the TM that α represents.
- But we also use M to denote both the TM and its representation.

Universal Turing Machine: Ability to Simulate

- There exists a universal Turing machine that can simulate the execution of every other TM M, given M's description as input.
- The parameters of the universal TM are fixed:
  - Alphabet size;
  - Number of states;
  - Number of tapes.
- The corresponding parameters for the machine being simulated could be much larger.
- This is not a hurdle because of the ability to use encodings.
- Even if the universal TM has a very simple alphabet, this suffices to:
  - Represent the simulated machine's state and transition table on its tapes;
  - Follow along the machine's computation step by step.

Efficiency of Universal Turing Machines

Theorem (Efficient Universal Turing Machine)

There exists a TM $\mathcal{U}$ such that, for all $x, \alpha \in \{0,1\}^*$,

$$\mathcal{U}(x, \alpha) = M_\alpha(x),$$

where M_α denotes the TM represented by α. Moreover, if M_α halts on input x within T steps then $\mathcal{U}(x, \alpha)$ 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.

- We only exhibit a $\mathcal{U}$ accomplishing the task in CT^2 time.
- $\mathcal{U}$ on input x, α, where α represents a TM M, needs to output $M(x)$. We may assume that M:
  (1) Has a single work tape (in addition to the input and output tape);
  (2) Uses the alphabet $\{0, 1, \sqcup, \triangleright\}$.

Proof of the Efficiency Theorem

- $\mathcal{U}$ can transform an encoding of a TM M into one of an equivalent TM $\widetilde{M}$ that satisfies these properties.

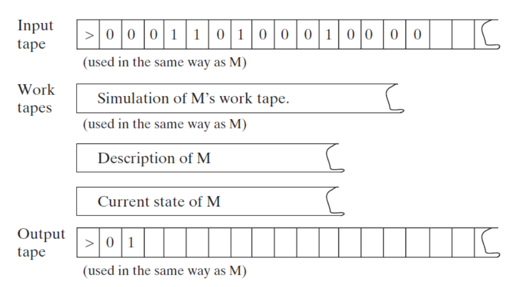
  A quadratic slowdown may be introduced.

  I.e., we may transform M, running in T time, to running in $C'T^2$ time, where C' depends on M's alphabet size and number of tapes.

  The TM $\mathcal{U}$ uses the alphabet $\{0, 1, \sqcup, \triangleright\}$ and three work tapes in addition to its input and output tape.
  - $\mathcal{U}$ uses its input tape, output tape and one of the work tapes in the same way M uses its three tapes;
  - $\mathcal{U}$ will use its first extra work tape to store the table of values of M's transition function;
  - $\mathcal{U}$ will use its second extra work tape to store the current state of M.

Proof of the Efficiency Theorem (Cont'd)

To simulate one computational step of M:



- $\mathcal{U}$ scans the table of M's transition function and the current state to find the new state, the symbols to be written and the head movements.
- Then $\mathcal{U}$ executes the transition, as specified.

We see that each computational step of M is simulated using C steps of $\mathcal{U}$, where C is some number depending on the size of the transition function's table.

Universal TM with Time Bound

- It is sometimes useful to consider a variant of the universal TM $\mathcal{U}$.
- It gets a number T as an extra input, in addition to x and α.
- It outputs

$$\mathcal{U}(x,\alpha,T) = \begin{cases} M_\alpha(x), & \text{if } M_\alpha \text{ halts on } x \text{ within } T \text{ steps,} \\ \text{✗}, & \text{otherwise.} \end{cases}$$

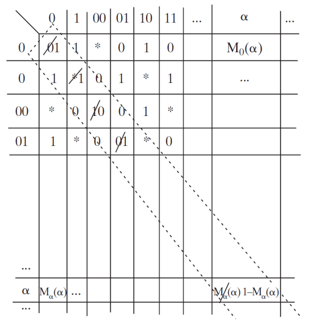
- By adding a time counter to $\mathcal{U}$, the proof of the theorem can be easily modified to give such a universal TM.
- The time counter is used to keep track of the number of steps that the computation has taken so far.

Subsection 5

Uncomputability: An Introduction

Uncomputable Functions and Diagonalization

- It may be surprising that there exist functions that cannot be computed within any finite number of steps!
- The next theorem shows the existence of uncomputable functions.
- In fact, the uncomputable function that it exhibits has range $\{0,1\}$, i.e., is a language.
- Uncomputable functions with range $\{0,1\}$ are also known as **undecidable languages**.
- The proof uses a technique called **diagonalization**, which is useful in complexity theory as well.



The Undecidable Language UC

Theorem

There exists a function $\mathrm{UC} : \{0,1\}^* \to \{0,1\}$ that is not computable by any TM.

- We define he function UC by setting, for every $\alpha \in \{0,1\}^*$,

$$\mathrm{UC}(\alpha) = \begin{cases} 0, & \text{if } M_\alpha(\alpha) = 1 \\ 1, & \text{otherwise} \end{cases}.$$

Suppose, for the sake of contradiction, that UC is computable.
Hence, there exists a TM M, such that

$$M(\alpha) = \mathrm{UC}(\alpha), \quad \text{for every } \alpha \in \{0,1\}^*.$$

Then, in particular, $M(\llcorner M \lrcorner) = \mathrm{UC}(\llcorner M \lrcorner)$.
This is impossible, since, by the definition of UC,

$$\mathrm{UC}(\llcorner M \lrcorner) = 1 \quad \text{iff} \quad M(\llcorner M \lrcorner) \neq 1.$$

The Halting Problem

- The function $\textsc{Halt}$, on input $\langle \alpha, x \rangle$, outputs 1 if and only if the TM M_α halts on input x within a finite number of steps.
- If computers could compute $\textsc{Halt}$, the task of designing bug-free software and hardware would become much easier.

Theorem

$\textsc{Halt}$ is not computable by any TM.

- Suppose there was a TM $M_{\textsc{Halt}}$ computing $\textsc{Halt}$.

  We will use $M_{\textsc{Halt}}$ to construct a TM M_{UC} computing UC.

  This would contradict the preceding theorem.

The Halting Problem (Cont'd)

- The machine M_{UC} operates as follows:

  Suppose it receives input α.

  It runs $M_{\mathrm{HALT}}(\alpha, \alpha)$.

  - If the result is 0, meaning that M_α does not halt on α, then M_{UC} outputs 1.
  - Otherwise, M_{UC} uses the universal TM $\mathcal{U}$ to compute $b = M_\alpha(\alpha)$.
    - If $b = 1$, then M_{UC} outputs 0.
    - Otherwise, it outputs 1.

  We assumed that $M_{\mathrm{HALT}}(\alpha, \alpha)$ outputs $\mathrm{HALT}(\alpha, \alpha)$ within a finite number of steps.

  Then the TM $M_{\mathrm{UC}}(\alpha)$ will output $\mathrm{UC}(\alpha)$.

The Idea of a Reduction

- The proof technique employed to show undecidability of $\textsc{Halt}$ is called a **reduction**.
- We showed that computing UC is reducible to computing $\textsc{Halt}$,
- I.e., that if there were a hypothetical algorithm for $\textsc{Halt}$, then there would be one for UC.
- Reductions are often used to show that a problem B is at least as hard as a problem A.
- This involves devising an algorithm that could solve A, given a procedure that solves B.
- There are many other examples of interesting uncomputable (also known as **undecidable**) functions.

Subsection 6

The Class P

Decidable Languages

- A **complexity class** is a set of functions that can be computed within given resource bounds.
- For technical convenience, we will pay special attention to Boolean functions, which define **decision problems** or **languages**.
- We say that a machine **decides** a language $L \subseteq \{0,1\}^*$ if it computes the function $f_L : \{0,1\}^* \to \{0,1\}$, where

$$f_L(x) = 1 \quad \text{iff} \quad x \in L.$$

Deterministic Time Bounds

Definition (The class DTIME)

Let $T : \mathbb{N} \to \mathbb{N}$ be some function. A language L is in

$$\mathrm{DTIME}(T(n))$$

iff there is a Turing machine that:

- Decides L;
- Runs in time $c \cdot T(n)$, for some constant $c > 0$.

- The D in the notation DTIME refers to "deterministic".
- The Turing machine introduced in this chapter is more precisely called the **deterministic Turing machine**, since, for any given input x, the machine's computation can proceed in exactly one way.

The Class P

- To make the notion of "efficient computation" precise, we equate it with polynomial running time, i.e., with time at most n^c, for some constant $c > 0$.

Definition (The class P)

We define

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

- The question as to whether IndSet (independent set) has an efficient algorithm can be expressed as "Is IndSet in P?"

Example: Graph Connectivity

- In the **graph connectivity** problem, we are given:
  - A graph G;
  - Two vertices s, t in G.
- We have to decide if s is connected to t in G.
- The problem is in P.
- The algorithm that shows this uses **depth-first search**.
  - It explores the graph edge-by-edge starting from s.
  - It marks visited edges.
  - In subsequent edges, it also tries to explore all unvisited edges that are adjacent to previously visited edges.
- After at most $\binom{n}{2}$ steps, all edges are either visited or will never be visited.

Class P Consists of Decision Problems

- The class P contains only decision problems.
- Thus, we cannot say, e.g., that "integer multiplication is in P".
- Instead, we may say that its decision version is in P:

$$\{\langle x, i\rangle : \text{The } i\text{th bit of } xy \text{ is } 1\}.$$

Running Time is a Function of the Number of Input Bits

- The running time is a function of the number of bits in the input.
- Consider the problem of solving a system of linear equations over the rational numbers.
- Given is a pair
$$\langle A, \boldsymbol{b} \rangle,$$
where:
  - A is an $m \times n$ rational matrix;
  - $\boldsymbol{b}$ is an m-dimensional rational vector.
- The problem is to find out if there exists an n-dimensional vector $\boldsymbol{x}$, such that
$$A\boldsymbol{x} = \boldsymbol{b}.$$

Running Time and the Number of Input Bits (Cont'd)

- The standard Gaussian elimination algorithm solves this problem in $O(n^3)$ arithmetic operations.
- But on a Turing machine, each arithmetic operation has to be executed bit by bit.
- To prove that this decision problem is in P, we have to verify that Gaussian elimination (or some other algorithm) runs on a Turing machine in time polynomial in the number of bits required to represent the input.

Decidability, P and Model of Computation

- We defined the classes of “computable” languages and P using Turing machines.
- Would these classes be different if we had used a different computational model?
- We saw that each of the variants of the Turing machine model we encountered can simulate any other with at most quadratic slowdown.
- So, for all these variants, polynomial time is the same, as is the set of computable problems.

The Church-Turing Thesis

The Church-Turing (CT) Thesis: Every physically realizable computation device, whether it is based on silicon, DNA, neurons or some other alien technology, can be simulated by a Turing machine.

- The thesis implies that the set of computable problems would be no larger on any other computational model than on the Turing machine.

The Strong Form of the CT Thesis: Every physically realizable computation model can be simulated by a TM with polynomial overhead.

- If true, it implies that the class P defined by any other physically realizable model will be the same as ours.

Criticisms of P Addressed by Other Classes

- Worst-case exact computation is too strict. The definition of P only considers algorithms that compute the function on every possible input, whereas not all possible inputs arise in practice.

  Possible remedies:
  - Average-case complexity;
  - Approximation Algorithms.
- Other physically realizable models. Subtleties in the strong form of the Church-Turing Thesis:
  (a) Precision when dealing with real numbers.
  (b) Use of randomness; the class BPP.
  (c) Use of quantum mechanics; the class BQP.
  (d) Use of other exotic physics, such as string theory; also BQP?
- Decision problems are too limited. Several classes intend to capture tasks such as computing non-Boolean functions, solving search problems, approximating optimization problems, interaction, etc.