Great Ideas in Computing

University of Toronto CSC196
Fall 2020

Week 4: October 18-October 22 (2021)

Week 6 slides

Announcements:

- We have an additional TA: Kian Kianpisheh
- Assignment A2 has now been posted. It is due Friday, October 29, 8AM. There is a 5% penalty for each 24 hours the assignment is late with a maximum of 96 hours to submit a late assignment.
- This assignment introduces some new concepts and other facts that are being discussed today and in last weeks tutorial.
- Tutorials are an important part of this course and you are expected to attend. **You are responsible for material discussed in the tutorials.**
- The first term quiz will be held in person this Friday, October 22. We will start the quiz promptly at 9:00 so that you have a full hour. You can bring in one 8 by 11 sheet of handwritten notes to use in the quiz but mainly your notes are just a good way to study.

This weeks agenda

The agenda for this week

- More on the Church-Turing hypothesis.
- Turing reduction (denoted $A \leq_T B$). Solving problem A using a solution for problem B and its application to proving undecidability results.
- Diagonalization
- The Entscheidungsproblem
- A pictorial and formal definition of Turing machines.
- The extended Church-Turing hypothesis. Sometimes called the Cook-Karp hypothesis.
- If time permits we begin a new topic, namely search engines.

Quick followup on the quantum computing discussion

Any comments on quantum computing?

Quick followup on the quantum computing discussion

Any comments on quantum computing?

Professor Wiebe touched on a number of concepts relating to complexity theory and cryptography in his discussion of quantum computing. In particular, he mentioned the *extended Church-Turing hypothesis* which I will discuss today.

I will be discussing the class *NP* when we discuss complexity theory. And I will be discussing one time pads and RSA encryption when we discuss cryptography.

What was your main "take-away" from the quantum discussion?

The Church-Turing hypothesis

As already discussed, there is a wide consensus on the acceptance of the Church-Turing thesis. That is, the equating of *computable* with Turing computable.

I want to emphasize however, here we are talking about discrete computation. There are some different formulations of what we mean by say computing a function $f : \mathbb{R} \to \mathbb{R}$.

For computability and discrete computation, it doesn't matter if we consider functions $f : \mathbb{N} \to \mathbb{N}$ or $f : \Sigma^* \to \Sigma^*$ for any for alphabet Σ with $|\Sigma| \geq 2$. Why?
When $|\Sigma| = 1$, there are some issues regarding how to encode things and when we consider complexity, it is best to assume $|\Sigma| \geq 2$ when representing integers.

The Church-Turing hypothesis continued

For a number of proposed alternative computational models either it has been shown that the model is either equivalent in representational power or weaker.

Moreover, let $\mathcal{M}$ an alternative model, then one can exhibit a mapping of any $\mathcal{M}$ computation into a Turing computation. This implies that the Turing model is at least as powerful as the $\mathcal{M}$ model. To prove equivalence of the models, one needs to show the converse is also true.

We can extend the defiintion of the Turing model (and other models) to the computation of the computation of higher order *operators*. For example let F be the set of one variable differentiable functions. Then we would want an operator that takes a function to its derivative. Similarly we would want an operator for integrating a function. And, of course, we would have to say how we are representing the function being differentiated or integrated.

Turing reducibility

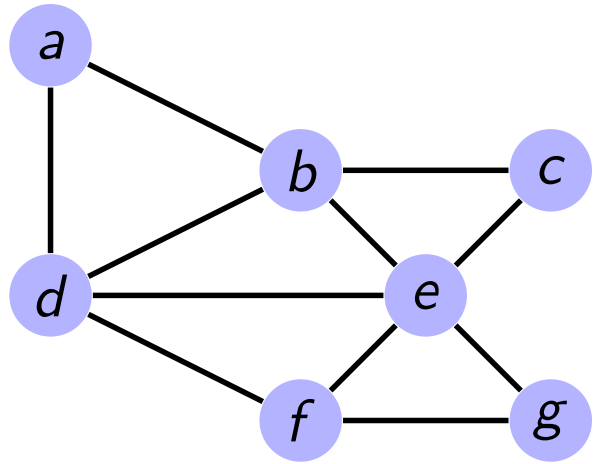
Alan Turing also introduced the idea of an *oracle Turing machine*. While one can formalize this concept, we will just use some examples. In today's terminology it is easy to understand the concept if one is familiar with subroutines.

Intuitively, when we say that solving a problem A (say computing a function) by reducing the problem to being able to solve a problem B, we mean that in a program P_A for A, we can ask ask (perhaps many times) for a different program P_B to solve B given some input z. That is, P_A temporarily turns control over to P_B (specifying some input z and P_B returns with the solution $B(z)$ and P_A continues its computation.

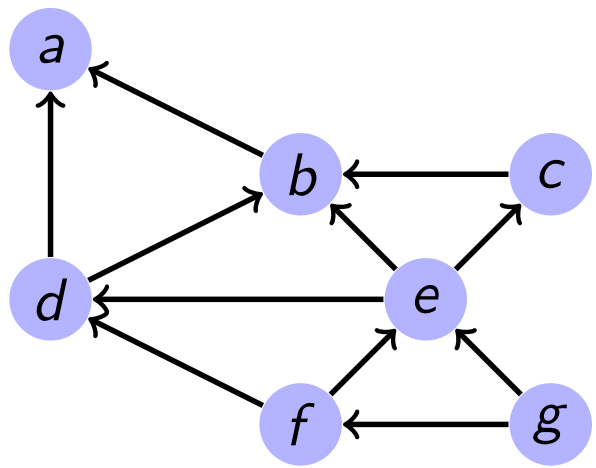
Let's just consider this concept in terms of $\{$YES,No$\}$ decision problems. Here is an example in terms of graphs. So first I will define the concept of a graph. Graphs come in two varieties, undirected graphs

Graphs: come in two varieties

1. undirected graphs ("graph" usually means an undirected graph.)



2. directed graphs (often called di-graphs).



Reducing strong connectivity to s-t connectivity

In a directed graph, $G = (V, E)$, a directed path from a node (also called a vertex) $u \in V$ to a node $v \in V$ is a sequence $u = u_0, u_1, u_2, \ldots, u_r = v$ such that $(u_i, u_{i+1}) \in E$. E is the set of edges in the directed graph.

The strong connectivity decision problem is : "Given a directed graph G, output YES if for every $u, v \in V$, there is a directed path from u to v, output NO otherwise.

For a directed graph the s-t connectivity problem is : Given a directed graph G and two vertices $s, t \in V$, is there a path from s to t.

How would you reduce the strong connectivity problem to the s-t connectivity problem?

Reducing strong connectivity to s-t connectivity

In a directed graph, $G = (V, E)$, a directed path from a node (also called a vertex) $u \in V$ to a node $v \in V$ is a sequence $u = u_0, u_1, u_2, \ldots, u_r = v$ such that $(u_i, u_{i+1}) \in E$. E is the set of edges in the directed graph.

The strong connectivity decision problem is : "Given a directed graph G, output YES if for every $u, v \in V$, there is a directed path from u to v, output NO otherwise.

For a directed graph the s-t connectivity problem is : Given a directed graph G and two vertices $s, t \in V$, is there a path from s to t.

How would you reduce the strong connectivity problem to the s-t connectivity problem?
We could simply ask "the oracle" for s-t connectivity if the answer is YES for ever pair of nodes $s, t \in V$.

Two consequences of a reduction

We denote a reduction of problem A to problem B by the notation $A \leq_T B$ where the T stands for Turing.

Suppose we can reduce a decision problem A to a decision problem B. Then if we can decide problem B, we can decide problem A.

This is how we normally think of reductions if we are an algorithm designer.

If $A \leq_T B$, there is another consequence, namely the *contrapositive*: if problem A is undecidable, then problem B is undecidable.

In propositional logic we say $B \implies A$ is equivalent to $\neg A \implies \neg B$. That is, $\neg A \implies \neg B$ is the contrapositive of $B \implies A$. (Do not confuse with the *converse* where the converse of $B \implies A$ is $A \implies B$.)

End of Monday, October 18 class; new announcements

We ended at slide 10. We will continue with a reduction that yields an undecidability result and then try to finish up our discussion of computability. We may or may not get to search engines this week.

I fixed the typos regarding s-t connectivity. Please do point out any errors in the slides during or after class.

New Announcements

- There is no class next Monday, October 25 but I am hoping to schedule a zoom TA office hour for some time during the week and before Friday, the 29th. I can also schedule a zoom meetinfg with anyone at some mutually convenient time. However, I strongly advise using Piazza for questions about the quiz or the second assignment.
- On Wednesday, October 27 we will have a guest presentation and discussion by Professor Nisarg Shah on the topic of social choice.
- There will be a zoom tutorial on Friday, October 29.

A few words about social choice

The field of social choice concerns how we make collective decisions based on indiviual preferences. The two main aspects are "fairness" (e.g., how do we divide divisible or indivisible objects in a fair way) and voting (i.e., how do we choose a winner or group of winners).

Beyond Nisarg Shah's contributions to social choice theory, he is a prinicpal designer of two plaforms in public use: Spliddit (for fair allocation) and Robovote (for group decision making).

Social choice (and other topics that used to be the demain of the social sciences) have now become important areas in CS. These topics have changed the way we think about algorithms, and CS already has had and will continue to have significant impact in the social sciences (e.g., economics, political science, sociology). Like other "killer appllications", these applications fit into the theme of great ideas.

Decidability and undecidability

There are a number of simple reductions proving undecidability. You can look at http://www.cs.virginia.edu/ evans/cs302/classes/class17.pdf for such examples.

When we discuss decidability and undecidability, we usually formulate this in tems of language recognition. Let's consder Turing machines with input strings $w \in \Sigma^*$ for some finite alphabet Σ.

Notation: If M is a TM, we let $< M >$ be an encoding of M and for a string $w \in \Sigma^*$ we let $|w|$ be the length of the string. We also use "iff" for "if and only if".

A language is a subset of strings; that is, $\mathcal{L} \subseteq \Sigma^*$ satisfying some property. We say that $\mathcal{L}$ is decidable (also called recognizable) if there is a TM M that will halt and accept a string w if $w \in \mathcal{L}$ and it will halt and reject w if $w \notin \mathcal{L}$. Otherwise we say that $\mathcal{L}$ is undecidable.

A simple example of using a reduction to prove undecidability

For example consider the following languages:
$\mathcal{L}_1 = \{< M > | M \text{ is a TM that accepts an input } w \text{ iff } |w| \geq 15\}$.
$\mathcal{L}_2 = \{< M > | M \text{ is a TM that accepts an input } w \text{ iff } |w| \geq 16\}$.
Suppose Turing showed that $\mathcal{L}_1$ is undecidable (which it is). We want to show that $\mathcal{L}_2$ is undecidable.

It suffices to show that $\mathcal{L}_1 \leq_T \mathcal{L}_2$

A simple example of using a reduction to prove undecidability

For example consider the following languages:
$\mathcal{L}_1 = \{< M > | M \text{ is a TM that accepts an input } w \text{ iff } |w| \geq 15\}$.
$\mathcal{L}_2 = \{< M > | M \text{ is a TM that accepts an input } w \text{ iff } |w| \geq 16\}$.
Suppose Turing showed that $\mathcal{L}_1$ is undecidable (which it is). We want to show that $\mathcal{L}_2$ is undecidable.

It suffices to show that $\mathcal{L}_1 \leq_T \mathcal{L}_2$
In fact, we will use a very simple type of reduction which I will call a *transformation* and which we can denote by $\leq_\tau$. We say that $A \leq_\tau B$ if there is a computable function f such that $w \in A$ iff $f(w) \in B$. It should be easy to see that $A \leq_\tau B$ is a special case of $\leq_T$.

Showing the desired transformtion for the languages on the last slide

Here is a description of the transformation of $< M >$ to $f(< M >)$ such that $< M_1 > \in \mathcal{L}_1$ iff $f(< M_2 >) \in \mathcal{L}_2$.

Given the description of a TM M_1, we are going to construct a TM M_2. That is, $< M_2 > = f(< M_1 >$. Given an input w, M_2 operates as follows: If $|w| < 16, M_2$ rejects w. Othwerwise, suppose $w = \sigma w'$ for some $\sigma \in \Sigma$ and $w' \in \Sigma^*$. M_2 will simulate M_1 on w'.

Claim: M_2 accepts w iff M_1 accepts w'. That is, $< M_1 > \in \{calL_1$ iff $< M_2 > \in \{calL_2$

Given that the halting problem is undecidable, many other problems can be proved to be undecidable using reductions. Some of these problems do not mention Turing machines but undecidability comes from the problem "being able to encode a Turing computation".

The Entscheidungsproblem

In his seminal paper "On Computable Numbers With an Application to the Entscheidungsproblem (i.e. decision problem), Turing uses his model and the undecidability of the halting problem, to prove the undecidabiliy of the "Entscheidungsproblem" posed by Hilbert in 1928. (Church provided an independent proof within his formalism.)

Sometimes this is informally stated as "can mathematics be decided ?"

The Entscheidungsproblem question refers to the decidability of predicate logic which Church and Turing independently resolved in 1936-1937. It would take a little while to formally define this "Entscheidungsproblem" but here is an example of the kind of question that one wants to answer:
Given a formula such as $\forall x \exists y : y < x$
can we determine if such a formula is always true no matter what what ordered domain x, y and $<$ refer to?
For example, $x < y$ and $y < z$ implies $x \neq z$ is always true.
But $x < y$ implies $\exists z : x < z < y$ is not true of all ordered domains (e.g., consider the integers)

A pictorial representation of a Turing machine

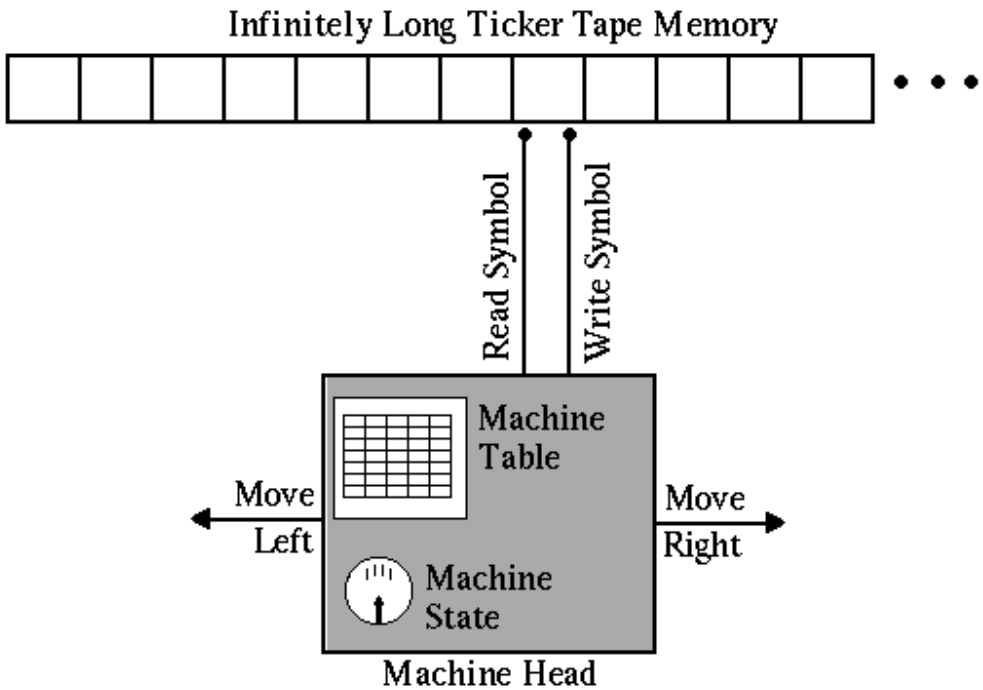


Figure: Figure taken from Michael Dawson "Understanding Cognitive Science"

Comments on Turing's model

- Formally, a Turing machine algorithm is described by the following function $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
  Q is a *finite* set of *states*. Γ is a finite set of symbols
  (e.g., $\Gamma = \{\#, 0, 1, a, b, \ldots\}$ and perhaps $\Sigma = \{0, 1\}$)
- Note: Each δ function is the definition of a single Turing machine; that is, each δ function is the statement of an algorithm.
- We can assume there is a halting state q_{halt} such that the machine halts if it enters state q_{halt}. There is also an initial state q_0.
- We view a Turing machine P as computing a function $f_P : \Sigma^* \rightarrow \Sigma^*$ where $\Sigma \subseteq \Gamma$ where $y = f(x)$ is the string that remains if (and when) the machine halts. There can be other conventions as to interpreting the resulting output y.
- Note that the model is precisely defined as is the concept of a computation step. A *configuration* of a TM is specified by the contents of the tape, the state, and the position of the tape head. A computation of a TM is a sequence of configurations, starting with an initial configuration.
- For decision problems, we can have YES and NO halting states.

A more general Turing machine model

The Turing machine model has been extended to allow separate read (for the input) and write (for the output when computing a function) tapes and any finite number of work tapes. Here is a figure of a multi-tape TM (but without separate input and output tapes).

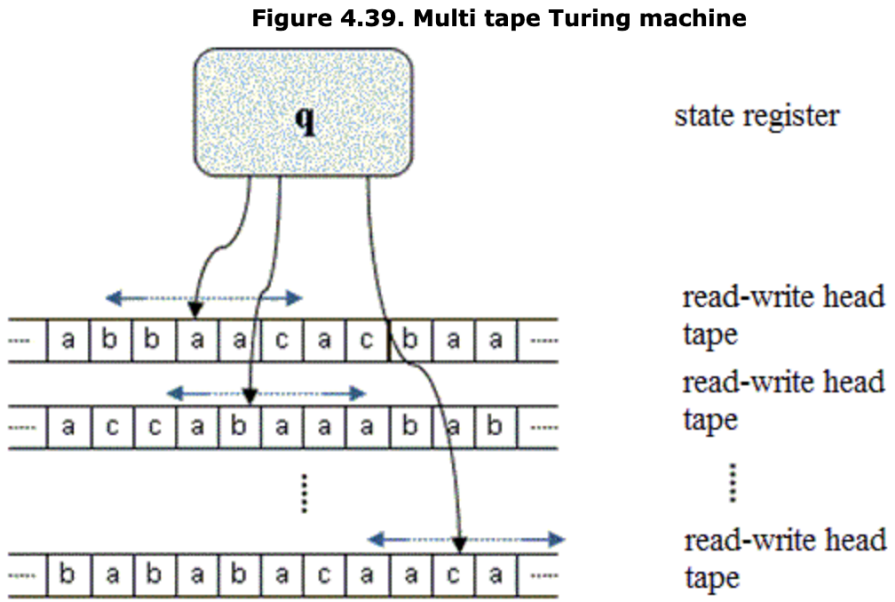


Figure: Figure from the Bela Gyires Informatics Curriculum Repository

The extended Church-Turing Thesis

Church-Turing computability provides a formal definition of *computable* (in principle). But it allows for computable functions of arbitrary complexity.

Indeed it is not difficult to show (again using diagonalization) that for any time bound $T(n)$ there are computable decision problems that require more time (or more memory) than $T(n)$ for sufficiently large values of n where n is the length of the input and output strings. Why do we usually say that operations in the von Neumann model take unit time?.

The extended Church-Turing Thesis

Church-Turing computability provides a formal definition of *computable* (in principle). But it allows for computable functions of arbitrary complexity.

Indeed it is not difficult to show (again using diagonalization) that for any time bound $T(n)$ there are computable decision problems that require more time (or more memory) than $T(n)$ for sufficiently large values of n where n is the length of the input and output strings. Why do we usually say that operations in the von Neumann model take unit time?.

The extended Church-Turing thesis states that any function that is "**feasibly computable**" must be computable by a Turing machine within time $p(n)$ where $p()$ is a polynomial. This extended thesis is stated with regard to classical computers and not necessarily quantum computers.

Note: Currently we (in theoretical CS) believe that there are functions (e.g., factoring large integers) "efficiently computable" by quantum computers but not efficiently computable by classical computers. . But can quantum computers of sufficient size be built?

Extended Church-Turing thesis continued

Informal theorem: Any "**reasonable"** classical computation model $\mathcal{M}$ can be simulated by a one tape or multi-tape Turing machine so that if say f is computable in time $T(n)$ on $\mathcal{M}$ then f is computable on a Turing machine in time $p_{\mathcal{M}}(T(n))$ for some fixed polynomial $p_{\mathcal{M}}$.

Using a mutli-tape TM, for most models, $p_{\mathcal{M}}(m)$ is $O(m^2)$ or $O(m^3)$. For example, if $T(n)$ is n^2 and $p_{\mathcal{M}}(m)$ is m^3 then $p_{\mathcal{M}}(T(n)$ is n^6.

What is **"not reasonable"**?

Extended Church-Turing thesis continued

Informal theorem: Any "**reasonable**" classical computation model $\mathcal{M}$ can be simulated by a one tape or multi-tape Turing machine so that if say f is computable in time $T(n)$ on $\mathcal{M}$ then f is computable on a Turing machine in time $p_{\mathcal{M}}(T(n))$ for some fixed polynomial $p_{\mathcal{M}}$.

Using a mutli-tape TM, for most models, $p_{\mathcal{M}}(m)$ is $O(m^2)$ or $O(m^3)$. For example, if $T(n)$ is n^2 and $p_{\mathcal{M}}(m)$ is m^3 then $p_{\mathcal{M}}(T(n)$ is n^6.

What is **"not reasonable"**?

Consider having unit time operations $+, -, *, \div$ where $\div$ means integer division; that is, dividing a by b results in the remainder. Now consider repeated squaring of $2, 2^2 = 4, 4^2 = 16, \ldots, 2^{(2^n)}$. That is, in n multiplications, we can construct an integer whose binary representation has 2^n bits.

It turns out that with such a model we can factor integers in polynomial time. **BUT** this is not reasonable as we are doing classical operations on exponentially long strings in unit time which is not reasonable.

Diagonalization

You may have learned in high school why using the diagonalization method it can be proved that the set of real numbers say in $(0, 1)$ is *uncountable* while the set of rationals in $(0, 1)$ is *countable*.

The idea is that since the rationals are countable we can list them and lets say that $r_i = \text{ the } i^{th}$ rational number in $(0, 1)$ (in some agreed uopon listing of these rationals). Suppose as a binary fraction $r_i = .r_i(1)r_i(2)r_I(3)\ldots r_i(m_i)$ for some m_i and $r_i(j) = 0$ for all $j > m_i$.

We can create a non-rational number $x = .s_1\ s_2\ s_3\ldots$ where $s_i = 1 - r_i(i)$.

Diagonalization and the halting problem

We will just sketch why the halting problem is undecidable.

Using diagonalization, we can show that the following halting problem is undecidable. Namely, the set of Turing machines is a countable set and let $\mathcal{M}_i$ denote the i^{th} TM. Consider the following function $f(i) = YES$ if $\mathcal{M}_i(i) = \text{NO}$ or $M_i(i)$ does not halt, and $f(i) = \text{NO}$ otherwise.

If the halting problem were decideable, then using a UTM and the claimed decidability of the halthing problem, f would be a computable function but that would be a contradiction since f is defined to be different than every TM $\mathcal{M}_j$.