Great Ideas in Computing

University of Toronto CSC196
Fall 2022

Week 4: October 3-7 (2022)

Week 4 slides

Announcements:

- This week the tutorial is on Wednesday (October 5). Inn addition to todays class, the Friday class will be a guest presentation by Rahul Krishnan.
- Assignment A1 is due Friday, October 7 at 8AM. There was a typo on the assignement saying that it was due Friday, October 8 but on the syllabus it does say Oct 7.
- The first quiz is Friday, October 21
- There may be one more additional guest by Nisarg Shah on Monday, October 24. He will discuss fairness and fair allocation.

Todays agenda

- The Church-Turing thesis continued
- Reductions
- Decidability and Undecidability
- Formalizing a Turing machine
- The extended (i.e., polynomial time) Church-Turing thesis.
- Diagonalization.

Note: For those who may be feeling that the seminar is getting too math intensive, our next topic will be more familiar. Namely, we will be discussing search engines.

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.

A pictorial representation of a Turing machine

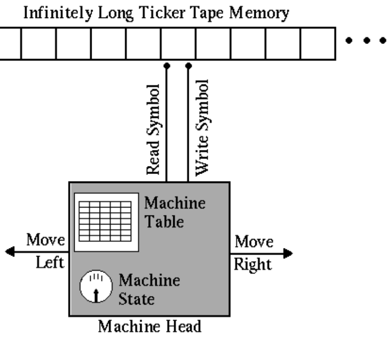

Figure: Figure taken from Michael Dawson "Understanding Cognitive Science"

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.

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 $A \implies B$.)

Decidability and undecidability

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 $w \notin \mathcal{L}$. Note, in particular, M halts (i.e. stops) on every input. Otherwise we say that $\mathcal{L}$ is undecidable.

A simple example of using a reduction to prove undecidability

For example consider the following languages:
$\mathcal{L}_1 = \{< M, w > | M \text{ is a TM that halts on input string } w\}$.
Note: The question as to whether or not $\mathcal{L}_1$ is decidable is what is usually called the halting problem.
$\mathcal{L}_2 = \{< M > | M \text{ is a TM that halts on all inputs } w\}$.
In fact, we can let w be a fixed input (e,g, $w = 010$) or we can even let w be $w =< M >$. That is, we can consider the languages
$\mathcal{L}_3 = \{< M > | M \text{ is a TM that halts on the input string } w = 010\}$ or
$\mathcal{L}_4 = \{< M, < M >> | M \text{ is a TM that halts on the input string } w =< M >\}$

Suppose $\mathcal{L}_1$ (or $\mathcal{L}_3$, or $\mathcal{L}_4$ is undecidable (which they are). We want to show that $\mathcal{L}_2$ is undecidable.

Let's show that $\mathcal{L}_3 \leq_T \mathcal{L}_2$

End of Monday, Oct 3 class

We ended on the previous slide claiming that we can show that $\mathcal{L}_3 \leq_T \mathcal{L}_2$. Thus if $\mathcal{L}_3$ is undecidable then $\mathcal{L}_2$ mjust also be unddecidable.

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 $\leq_T$. Is this obvious?

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 $< M_3 > \in \mathcal{L}_3$ iff $f(< M_2 >) \in \mathcal{L}_2$.

Given an encoding of a Turing machine TM M_3, we are going to construct a TM M_2. That is, $< M_2 >= f(< M_1 >$. M_2 operates on input string w as follows:
If $w \neq 010$, M_2 halts. (It isn't important if M_2 accepts or rejects.)

Claim: M_2 accepts w iff M_3 accepts w. That is, $< M_3 > \in \{calL_3$ iff $< M_2 > \in \{calL_2$

Given that the halting problem is undecidable, many other problems can be proved to be undecidable using reductions. Most 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) but is true of the rationals.

Repeating the pictorial representation of a Turing machine

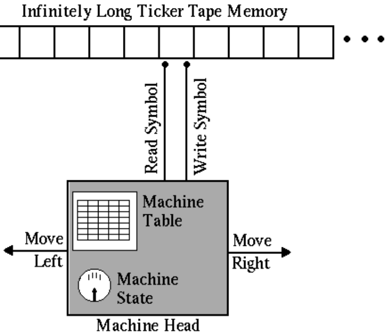

Figure: Figure taken from Michael Dawson, "Understanding Cognitive Science"

Formalization of a Turing machine

- Formally, a Turing machine algorithm is described by the following function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
  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 $f_P : \Sigma^* \to \Sigma^*$ 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).

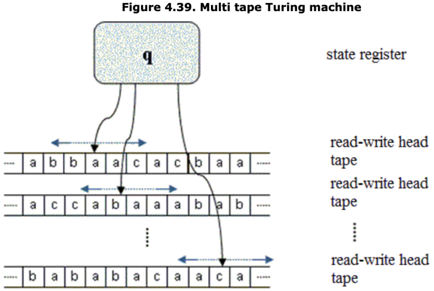

Figure: Figure from the Bela Gyires Informatics Curriculum Repository

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 $f(i) = \mathit{YES}$ if $\mathcal{M}_i(i) = \text{ NO or }\ M_i(i)$ does not halt, and $f(i) =$ 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 $\mathcal{M}_j$.
What Turing showed is that we can encode whether or not a Turing machine M accepts input w (i.e. the halting problem) by a statement in predicate logic.