CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Lecture 23: Uncomputable problems

Harvard SEAS - Fall 2024 *2024-11-21*

1 Announcements

Announcements:

- Salil OH 11-12 pm in SEC 3.327, Anurag OH on zoom on Friday 1:30-2:30 pm
- Pset 9 out
- No sections next week due to thanksgiving.

Recommended Reading:

- MacCormick Ch. 3 and §7.7–7.9

2 One-sided solutions to decision problems

In Lecture 2, we defined what it means for an algorithm to solve a computational problem. Let us restate this definition, focusing on decision problems and using the RAM model of computation. Note that RAM programs are not crucial here; we could have used Word-RAM programs, Python, C, etc. to model our algorithms (recall the Church-Turing thesis).

Definition 2.1. Let $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ be a decision problem and let Q be a RAM program. We say that Q *solves* Π if

- For every $x \in \mathcal{I}$ such that $f(x) = \{\texttt{yes}\}$, $Q(x)$ halts and returns `yes`.
- For every $x \in \mathcal{I}$ such that $f(x) = \{\texttt{no}\}$, $Q(x)$ halts and returns `no`.

Last time we stated our first example of an *unsolvable problem*, which cannot be solved by any algorithm whatsoever.

Input	**:** A RAM program P and an input x
Output	**:** `yes` if P halts on input x, `no` otherwise

Computational Problem HaltingProblem-RAM

Today, we will see why this problem can't be solved by any algorithm. We will also see other decision problems with same property. Before delving into the proof, it is very useful to look at a weaker notion of solving a problem.

Definition 2.2. Let $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ be a decision problem and let Q be a RAM program. We say that Q *one-sided solves* Π if

- For every $x \in \mathcal{I}$ such that $f(x) = \texttt{yes}$,

- For every $x \in \mathcal{I}$ such that $f(x) = \texttt{no}$,

In other words, for a RAM program Q to one-sided solve Π, it is only required that Q halts and gives correct answer in the `yes` case. In the `no` case, Q is allowed to not halt, but if it halts then it must output correctly. This is not really a satisfactory notion of solving a computational problem, because if Q has not halted after we run it for a long time, we don't know whether it eventually will halt and give us an answer or not.

Q: If a RAM program Q always halts and one-sided solves Π, then does it solve Π?

A:

Examples:

- Here is a one-sided algorithm for the HaltingProblem-RAM. It makes use of the universal RAM program U (Theorem 2.1 in Lecture 22) that can simulate any RAM program on any input.

- Another example is the following weird decision problem, which asks what a program does if we run it on itself.

Input	**:** A RAM program P
Output	**:** `yes` if $P(P) = \texttt{no}$;
	`no` if $P(P) = \texttt{yes}$;
	`no` if $P(P)$ does not halt.

Computational Problem RejectSelf-RAM

The RAM program to one-sided solve this is as follows.

- RejectSelf-RAM is a somewhat contrived problems about programs. Lets look at more interesting examples. The first one is a problem about polynomial equations.

Input	**:** A multivariate polynomial $p(x_0, x_1, \ldots, x_{n-1})$ with integer coefficients
Output	**:** `yes` if there are natural numbers $\alpha_0, \alpha_1, \ldots, \alpha_{n-1} \in \mathbb{N}$ such that $p(\alpha_0, \alpha_1, \ldots, \alpha_{n-1}) = 0$, `no` otherwise

Computational Problem Diophantine Equations

Fermat's Last Theorem (a conjecture of Pierre de Fermat from 1637) is an instance of this problem, with $n = 3$ and $p(x_0, x_1, x_2) = x_0^k + x_1^k - x_2^k$ (Andrew Wiles in 1995 famously proved that there are no solutions to this instance when $k \geq 3$). For univariate polynomials $p(x) = ax^2+bx+c$ of degree 2, there *is* an algorithm to solve this problem; namely by checking whether either of the roots $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ lie in $\mathbb{N}$. Here is a RAM program to one-sided solve Diophantine Equations.

It searches over all tuples with the hope of setting the polynomial to 0. If polynomial is never 0, the algorithm will never halt.

- Another example is that of tiling a plane with squares.

Input	**:** A finite set T of square tiles with each of the four edges colored (using an arbitrarily large palette of colors)
Output	**:** `no` if the entire 2-d plane can be tiled using tiles from T so that colors of abutting edges match, `yes` otherwise

Computational Problem Tiling

There is a RAM program that one-sided solves Tiling. It proceeds by searching over larger and larger square regions of 2-d plane and trying all tilings within these regions to see if tiling fails. If the tiling is not possible (the `yes` case), then this will be seen in some large enough square region and the program will stop. If the tiling is possible (the `no` case), the algorithm will run forever.

3 The Halting Problem is unsolvable

We now formally show the unsolvability of the Halting Problem. Importantly, the theorem is about solving the Halting Problem in the usual sense, rather than one-sided solving. The latter is doable - as we saw above.

Theorem 3.1. *(Turing) There is no algorithm that solves HaltingProblem-RAM.*

Note that Turing proved this theorem for HaltingProblem-TM; the statement above for the RAM version is equivalent to it (as discussed in Lecture 22). We will soon see that RejectSelf-RAM is also unsolvable. Further ahead, we will mention that Diophantine Equations and Tiling are all unsolvable. But the HaltingProblem-RAM has a special place among these unsolvable problems, due to the following lemma.

Lemma 3.2. *Let* $\Pi = (\mathcal{I}, \{\texttt{yes}, \texttt{no}\}, f)$ *be a decision problem that is one-sided solvable. It holds that* Π *reduces to HaltingProblem-RAM.*

Proof. Let Π be one-sided solvable by a RAM program Q. The reduction algorithm A is as follows

```
1 A:
2
```

The program uses the HaltingProblem-RAM oracle to check if Q halts on x. By the one-sided property, if Q does not halt on x, then $f(x)$ must be `no`. If Q halts on x, then we can simply run Q on x – using the universal RAM program U – and output the result. □

Remark: (Class RE)

Now we are in a position to prove Theorem 3.1.

Proof. Our strategy is as follows. We will show that RejectSelf-RAM is unsolvable. However, combined with Lemma 4.2 from Lecture 22, Lemma 3.2 shows that if the HaltingProblem-RAM were solvable, then RejectSelf would be solvable. This leads to contradiction.

Formally,

□

The self-contradictory nature of R is very related to the paradoxical sentence "This sentence is false" and also the paradoxical set of all sets that don't contain themselves $\{S : S \notin S\}$. It comes out of "Cantor's diagonalization" argument very similar as the one used to prove that the real numbers are uncountable.

4 Natural unsolvable problems

The phenomenon of unsolvability is not limited to problems about programs. Indeed, there's a sense in which *almost all* computational problems are unsolvable: it can be shown that there are "uncountably many" decision problems over any infinite set $\mathcal{I}$ of inputs, but there are only "countably many" RAM programs P, so most decision problems cannot be solved by any RAM program.

But it's more interesting and useful to identify natural examples of unsolvable problems. Diophantine Equations introduced earlier is a striking example. This problem was posed by Hilbert in a famous address to the International Congress of Mathematicians in 1900, as part of a list of 23 problems that Hilbert laid out as challenges for mathematicians to tackle in the 20th century. Several of Hilbert's problems were part of a general project to fully formalize and mechanize mathematics. Hilbert's 2nd problem was to prove consistency of the axioms of mathematics, which was shown to be impossible by Gődel's Incompleteness Theorem in 1931. Turing's work on the undecidability of the Halting Problem was also motivated by Hilbert's program, and was used by Turing to show that there is no algorithm to decide to truth of well-defined mathematical statements (since whether or not a Turing machine halts on a given input is a well-defined mathematical statement), resolving Hilbert's Entscheidungsproblem ("Decision Problem," posed by Hilbert and Ackermann in 1928). Hilbert's 10th problem asks about a very special case of the Entscheidungsproblem, namely Diophantine Equations, and was finally resolved in 1970 through the work of several mathematicians:

Theorem 4.1 (Matiyasevich, Robinson, Davis, Putnam)**.** *Diophantine Equations is unsolvable.*

Tiling is yet another example of a simple unsolvable problem.

Theorem 4.2 (Wang, Berger)**.** *Tiling is unsolvable.*

The tiling problem has a beautiful theory that dates back to Persian architectures, with more recent surprises being Wang/Berger/Penrose's aperiodic tiling. See here and here for nice optional reads.

5 Program Verification and Analysis

A dream (especially for TFs in CS courses!) is that we could just write a mathematical specification of what a program P should do, such as the following , and then automatically verify that a program meets the specification:

$$\text{Spec} : \forall A, \ell, u, k \;\; (0 \le \ell \le u, \forall i \;\; A[i-1] \le A[i]) \to (P(A, \ell, u, k) = \texttt{yes} \leftrightarrow \exists i \in [\ell, u] \;\; A[i] = k)\,.$$

Dream: an algorithm V that given a specification Spec and a program P, $V(\text{Spec}, P)$ will always tell us whether or not P satisfies Spec.
Q: Why is the dream not achievable?

Thus all program verification tools have one or more of the following limitations:

Nevertheless, the tools for program verification have grown in power and sophistication over the years, and we will study one of the most powerful approaches in next lecture.