CS1200: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Lecture 22: Unsolvable Problems

Harvard SEAS - Fall 2025 *2025-11-18*

1 Announcements

- Salil's next OH Thu 1-1:45pm SEC 3.327.
- Last SRE this Thursday! Be sure to prepare (both senders and receivers) and arrive on time.

Recommended Reading:

- Hesterberg–Vadhan 24
- MacCormick 6.0–7.6

2 Programs that analyze programs

Q: Are there problems that cannot be solved no matter how much time we allow?

We will find many examples of such impossible-to-solve problems when we turn to questions about analyzing programs. Having programs as input occurs often in the practice of computing:

Input: A RAM program P

Output: A Turing Machine M that is equivalent to P. That is, for every input x, M halts on x iff P halts on x, and if so, then $M(x) = P(x)$.

Computational Problem RAM2TM

Theorem 2.1. *There is an algorithm that solves* RAM2TM*. More generally, for every two of the Turing-equivalent computational models $\mathcal{M}$ and $\mathcal{N}$ that we have seen, there is an algorithm that converts an arbitrary program P in model $\mathcal{M}$ to an equivalent program Q in model $\mathcal{N}$.*

Proof idea.

□

We refer to computational models $\mathcal{M}$ for which programs can be algorithmically converted to and from equivalent programs in the RAM Model as *constructively Turing-equivalent*.

Theorem 2.2 (Universal RAM Program)**.** *There is a RAM program U such that* for every *RAM program P and input x,*

Proof idea.

□

Variants of Universal Programs.

- There are universal programs for every constructively Turing-equivalent model. Runtime statements may vary.
- There is also a universal Word-RAM program, where runtime increases by a constant factor (like in Theorem 2.2) but also word size increases by an additive constant.
- Can also do *clocked* simulations, where $U(P, x, t)$ only simulates P on x for t steps.

See the textbook for precise statements.

Importance of Universal Programs.

- Historically: Universal Turing Machine (Turing, 1936)

- Hardware vs. Software: Can build just one computer (U) and use it to execute any program P we want. Previously: build new hardware for every new type of problem we want to solve. (The Mark I computer near the main entrance of the SEC was one of the first such computers.)
- Inspired the development of modern computers (e.g. the "von Neumann Architecture").
- Programs vs. Data: we can think of programs P as data themselves.

3 The Halting Problem

In the previous section, we were able to specify the computational problems that compilers solve, namely ones like RAM2TM. What problem might we *hope* that the universal program can solve?

Input: A RAM program P and an input x **Output:** The output of P on x (if P ever halts on x)

Computational Problem Determine Output

Q: Why doesn't the Universal RAM Program solve Determine Output?

A:

In fact that limitation is inherent, because the following problem is unsolvable:

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

Computational Problem Halting Problem–RAM

Theorem 3.1. *There is no algorithm that solves* Halting Problem–RAM.

We'll prove this theorem in the next class. For today, we'll just assume it's true.

Definition 3.2. Let $\Pi = (\mathcal{I}, \mathcal{O}, f)$ be a computational problem. We say that Π is *solvable* if there exists RAM program P that solves Π. Otherwise we say that Π is *unsolvable*.

Note that we don't care about runtime of P in this definition; classifying problems by runtime was the subject of Computational Complexity. Halting Problem–RAM is the first unsolvable problem we have seen.

4 Unsolvable problems via reduction

Similarly to what we saw with NP-completeness and SAT, that we have one unsolvable problem (Halting Problem–RAM), we will be able to obtain more via reductions. For this, we recall the following:

Lemma 4.1. *Let Π and Γ be computational problems such that $\Pi \leq \Gamma$. If Π is unsolvable, then Γ is unsolvable.*

We highlight that this lemma applies to *all* reductions, including those whose runtime is more than polynomial.

The following problem is a special case of Halting Problem–RAM.

Input: A RAM program Q

Output: `yes` if _____, `no` otherwise

Computational Problem Halts On Empty–RAM

Here the empty input ε is just an array of length 0.

Theorem 4.2. Halts On Empty–RAM *is unsolvable.*

Proof.

A template for this reduction `Red` is just like the *mapping reductions* between decision problems that we discussed in the context of NP-completeness, except that now we don't care about the runtime of the reduction (since we are studying solvability vs. unsolvability):

```
   Red(P, x):
   Input      : A RAM program P and an input x
   Output     : yes if P halts on x, no otherwise
0  Construct from P and x a RAM program Q_{P,x} such that
     _____;
1  Call _____;
```

Algorithm 4.1: Template for Halting Problem–RAM $\leq$ Halts On Empty–RAM

How can we construct this RAM program $Q_{P,x}$ in Line 0? If P has commands $C_0, \ldots, C_{m-1}$ and x has length n, we construct $Q_{P,x}$ as follows:

The following is the key claim for the correctness of our reduction.

Claim 4.3. *$Q_{P,x}$ halts on ε if and only if P halts on x.*

Proof.

□

Now, we see that plugging the construction of $Q_{P,x}$ from into the reduction template (Algorithm 4.1) gives a correct reduction from the Halting Problem–RAM to Halts On Empty–RAM.

By the unsolvability of Halting Problem–RAM (Theorem 3.1) and Lemma 4.1, we deduce that Halts On Empty–RAM is unsolvable. □

Using compilers, we can deduce that Halts On Empty is unsolvable for other models.

Theorem 4.4. *For every constructively Turing-equivalent computational model $\mathcal{M}$,* Halts On Empty–$\mathcal{M}$ *is unsolvable.*

Proof.

□

The Word-RAM model is also constructively Turing-equivalent, but the formulation is a bit more complex since a Word-RAM's computation (in particular, whether or not it halts) depends not only on its input x but its word size w.

Input: A Word-RAM program R

Output: `yes` if
`no` otherwise

Computational Problem HALTS ON EMPTY–WORD RAM

Theorem 4.5. HALTS ON EMPTY–WORD RAM *is unsolvable.*

Our next example of an unsolvable problem is the following:

Input: A RAM program S

Output: `yes` if _____, `no` otherwise

Computational Problem IS A 3-COLORING SOLVER

Theorem 4.6. IS A 3-COLORING SOLVER *is unsolvable.*

Proof. We give a mapping reduction `Red` from HALTS ON EMPTY–RAM to IS A 3-COLORING SOLVER.

Given a program Q, our reduction `Red` constructs the program S_Q as follows:

To establish the correctness of our mapping reduction, we need to check:

Claim 4.7. *S_Q solves* GRAPH 3-COLORING *if and only if Q halts on ε.*

Proof.

□

□

We note again analogous results hold for all constructively Turing-equivalent models. In particular, the variants of IS A 3-COLORING SOLVER where the inputs are Turing Machines, Word-RAM programs, Python programs, etc. are all unsolvable.

Theorem 4.8 (Rice's Theorem, informally stated)**.** *Every non-trivial semantic property of programs is unsolvable.*

- Semantic property: depends only on the input–output behavior of the program. (As opposed to internals like lines of code, values of variables, runtime, etc.)
- Nontrivial property: no constant answer correctly solves the problem for all inputs.
- Precise statement given in the textbook. If you study it, you may use Rice's Theorem, but be sure to verify both of the required properties when you use it!