Turing Machines TODO: ADD TURING MACHINE CODE

Your Name

April 20, 2025

Contents

1 Introduction

This report explores the fundamental limits of computation by first illustrating, in Section 8.1, a concrete "hello–world" problem that no program can decide, motivating the need for a simpler, formal model. Section 8.2 then introduces the Turing machine—a finite control with an infinite read/write tape—and shows how its precise notation enables rigorous proofs of undecidability and intractability across domains. Building on this, Section 9.1 defines the diagonalization language L_d, consisting of those machine–input pairs where the machine does not accept its own code, and proves L_d is not even recursively enumerable. Finally, Section 9.2 examines the universal language L_u, which is semi-decidable by simulating any encoded Turing machine on its input, yet cannot be decided by any halting algorithm, thereby delineating the distinction between recursive (decidable) and merely RE (semi-decidable) problems.

2 Summary of Reading

8.1 Problems That Computers Cannot Solve

The purpose of this section is to provide an informal, C-programming-based introduction to the proof of a specific problem that computers cannot solve.

The particular problem we discuss is whether the first thing a C program prints is *hello, world*. Although we might imagine that simulation of the program would allow us to tell what the program does, we must in reality contend with programs that take an unimaginably long time before making any output at all. This problem— not knowing when, if ever, something will occur— is the ultimate cause of our inability to tell what a program does. However, proving formally that there is no program to do a stated task is quite tricky, and we need to develop some formal mechanics. In this section, we give the intuition behind the formal proofs.

8.2 The Turing Machine

The theory of undecidable and intractable problems not only shows that certain questions (such as whether a C program ever prints "hello, world") have no algorithmic solution, but also highlights that many decidable problems require impractical amounts of time. To address everyday questions in domains beyond direct program analysis—grammar ambiguity, Boolean-formula satisfiability, etc.—we introduce the Turing machine: a mathematically precise model consisting of a finite control and a single infinite tape on which symbols can be read, written, and the head moved left or right. Because its configurations can be described succinctly, the Turing machine enables formal proofs of undecidability and intractability where real-world program states are too complex to handle.

Using Turing machines, we can show that problems like Post's Correspondence are undecidable and that many natural decision problems are intractable despite being decidable. The fact that Turing machines compute exactly the same class of functions as other models (partial-recursive functions, lambda calculus, and modern computers) underpins the Church–Turing thesis, giving us a unified framework for understanding what can—and cannot—be computed in practice.

9.1 A Language That Is Not Recursively Enumerable

We begin by coding every binary string as both a potential Turing machine and its own input, so that the "ith" string w_i corresponds to the "ith" machine M_i. Define the diagonalization language

$$L_d \;=\; \{\, w_i \mid w_i \notin L(M_i) \,\},$$

i.e. those strings that their corresponding machine does *not* accept. By ordering strings first by length and then lexicographically, and by using a simple delimiter-based encoding for states, tape symbols, and transitions, we ensure every Turing machine with alphabet $\{0,1\}$ has at least one binary code, and every binary string can be interpreted as some machine (possibly trivial).

To show L_d is not even recursively enumerable, assume for contradiction that some machine M accepts exactly L_d. Let w_k be a code for M itself. Then M accepts w_k if and only if $w_k \notin L(M_k)$, but since $M = M_k$, this says "$w_k \in L(M_k)$ if and only if $w_k \notin L(M_k)$," a contradiction. Hence no Turing machine can enumerate or accept L_d, proving it is not RE.

9.2 An Undecidable Problem That Is RE

The class of recursively enumerable (RE) languages consists of those accepted by some Turing machine (TM), but some of these machines may never halt on inputs not in the language. We refine RE languages into two subclasses: recursive (decidable) languages, for which a TM always halts and decides membership, and the remainder of RE, where a machine halts only on strings in the language. Recursive languages are closed under complement, so if an RE language's complement fails to be RE, the language cannot be recursive. Our next focus is the universal language

$$L_u = \{\langle M, w\rangle \mid M \text{ accepts } w\},$$

which is RE because a universal TM can simulate M on w and halt if M does.

Although L_u is RE, it is not recursive. If it were, then its complement would also be RE, and one could reduce the non-RE diagonalization language L_d to it, contradicting the non-enumerability of L_d. Thus L_u exemplifies a natural decision problem that is semi-decidable but inherently undecidable.

3 Homework

Exercise A

Problem summary: Construct three Turing machines over the alphabet $\{0,1\}$:

1. M_1 accepts the language $\{1\,0^n \mid n \geq 0\}$ and on input $1\,0^n$ outputs $1\,0^{n+1}$.
2. M_2 accepts $\{1\,0^n \mid n \geq 0\}$ and on input $1\,0^n$ outputs the single symbol 1.
3. M_3 accepts all binary strings and on any input replaces each 0 by 1 and each 1 by 0.

Answer: The Turing Machine Simulator is currently down so I have not verified this turing machine. I will udate this reort with the simulator script as soon as the service is back online

(1) M_1 $M_1 = (\{q_0, q_1, q_2, q_a\}, \{0,1\}, \{0,1,B\}, \delta_1, q_0, B, \{q_a\})$ with

$$\begin{aligned}
\delta_1(q_0, 1) &= (q_1, 1, R),\\
\delta_1(q_1, 0) &= (q_1, 0, R),\\
\delta_1(q_1, B) &= (q_2, 0, L),\\
\delta_1(q_2, 0) &= (q_2, 0, L),\\
\delta_1(q_2, 1) &= (q_a, 1, R).
\end{aligned}$$

(2) M_2 $M_2 = (\{q_0, q_1, q_2, q_3, q_a\}, \{0,1\}, \{0,1,B\}, \delta_2, q_0, B, \{q_a\})$ with

$$\begin{aligned}
\delta_2(q_0, 1) &= (q_1, 1, R),\\
\delta_2(q_1, 0) &= (q_1, 0, R),\\
\delta_2(q_1, B) &= (q_2, B, L),\\
\delta_2(q_2, 0) &= (q_2, B, L),\\
\delta_2(q_2, 1) &= (q_3, 1, R),\\
\delta_2(q_3, 0) &= (q_3, B, R), \quad \delta_2(q_3, 1) = (q_3, B, R),\\
\delta_2(q_3, B) &= (q_a, B, R).
\end{aligned}$$

(3) M_3 $M_3 = (\{q_0, q_a\}, \{0,1\}, \{0,1,B\}, \delta_3, q_0, B, \{q_a\})$ with

$$\delta_3(q_0, 0) = (q_0, 1, R), \quad \delta_3(q_0, 1) = (q_0, 0, R), \quad \delta_3(q_0, B) = (q_a, B, R).$$

Exercise 1

Problem summary: Classify each of the following languages over the encoding of Turing machines (and inputs) as decidable, recursively enumerable (r.e.), or co-r.e. (has an r.e. complement):

$$\begin{aligned}
L_1 &= \{\, M \mid M \text{ halts on its own description}\},\\
L_2 &= \{(M, w) \mid M \text{ halts on input } w\},\\
L_3 &= \{(M, w, k) \mid M \text{ halts on } w \text{ within } k \text{ steps}\}.
\end{aligned}$$

Answer:

1. L_1 is r.e. but not co-r.e., and therefore undecidable.
2. L_2 is r.e. but not co-r.e., and therefore undecidable.
3. L_3 is decidable (by simulating M on w for k steps), hence r.e. and co-r.e.

Exercise 2

Problem summary: Decide for each of the following whether it holds in general. If yes, give a brief proof; if no, give a counterexample.

1. If L_1 and L_2 are decidable, then $L_1 \cup L_2$ is decidable.
2. If L is decidable, then its complement $\overline{L}$ is decidable.
3. If L is decidable, then L^* is decidable.
4. If L_1 and L_2 are r.e., then $L_1 \cup L_2$ is r.e.
5. If L is r.e., then $\overline{L}$ is r.e.
6. If L is r.e., then L^* is r.e.

Answer: The Turing Machine Simulator is currently down so I have not verified this turing machine. I will udate this reort with the simulator script as soon as the service is back online

1. **True.** Given deciders for L_1 and L_2, on input w run the first; if it accepts, accept; otherwise run the second and accept or reject accordingly.
2. **True.** If M decides L, build a decider for $\overline{L}$ that runs M and flips its answer ("accept" $\leftrightarrow$ "reject").

3. **True.** To decide $w \in L^*$, use dynamic programming: for each $0 \le i \le |w|$, test whether there is a split $w = uv$ with $u \in L$ (by the decider) and $v \in L^*$ (by recursion or table lookup). Since $|w|$ is finite, this always halts.

4. **True.** If M_1 and M_2 semi-decide L_1 and L_2, then on input w dovetail (interleave) simulations of $M_1(w)$ and $M_2(w)$; accept as soon as either halts accepting.

5. **False.** Counterexample: the universal language $L_u = \{\langle M, w\rangle \mid M \text{ halts on } w\}$ is r.e. but its complement is not r.e. (otherwise the halting problem would be decidable).

6. **True.** Recursively enumerable languages are closed under concatenation and union; hence $L^* = \bigcup_{k\ge 0} L^k$ is r.e. by enumerating all k-fold concatenations.

4 Conclusion

Having explored the fundamental limits of algorithmic computation, the readings introduced a concrete demonstration of an undecidable problem via the "hello–world" detection within C programs, showing how self-referential constructions lead to no effective testing procedure. Section 8.2 formalized computation with the Turing machine model, enabling precise proofs of undecidability and delineating decidable, recursively enumerable, and non-enumerable languages. In Section 9.1 we defined the diagonalization language $L_d = \{w_i \mid M_i \text{ does not accept } w_i\}$ and proved it is not even recursively enumerable by exploiting a contradiction on the machine's own code. Section 9.2 contrasted L_d with the universal language L_u, which is recursively enumerable yet undecidable, illustrating the distinction between semi-decidable and decidable problems and leveraging closure properties of recursive languages under complement.

Through exercises we applied these concepts concretely: constructing Turing machines for simple language transformations; classifying halting-related languages (L_1, L_2, L_3) as r.e., decidable, or co-r.e.; and investigating closure properties of decidable and r.e. classes under union, complement, and Kleene star. These tasks reinforced the interplay between formal definitions, encoding techniques, and simulated execution in establishing decidability and complexity boundaries. The readings and exercises thus illuminate the power and limits of algorithmic methods and refine our understanding of computability.

References

[HMU] J. E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*, Archive.org Link.