RICE'S THEOREM

Mihai-Valentin DUMITRU
mihai.dumitru2201@upb.ro

November 2022

As you may have noticed from the lab(s) and lecture on reductions, many problems that take as input the description of a Turing Machine are undecidable.

In this lecture, we will formalize this intuition, rigorously defining a category of decision problems and proving their undecidability. This result is known as *"Rice's theorem"*.

1 Intuition

Let's consider an example of a decision problem whose undecidability you've proven at the lab:

$$\texttt{P}(enc(M)) = \begin{cases} TRUE, & \forall w \in \Sigma^*, M[w] \text{ decides if } w \text{ is a palindrome} \\ FALSE, & otherwise \end{cases}$$

Note that it's crucial to distinguish between this problem and the following one:

$$\texttt{PALINDROME}(w) = \begin{cases} TRUE, & w \text{ is a palindrome} \\ FALSE, & otherwise \end{cases}$$

`PALINDROME` tells us if a particular string is a palindrome. It is a decidable problem; at the lab on Turing Machines, you even had to write a machine that decides it.

`P` tells us if a particular machine decides `PALINDROME`. It is *undecidable*; at the first reductions lab, you probably built a reduction to it from the halting problem: $\texttt{HALT} \leq_m \texttt{P}$.

The transformation t takes an input that encodes a (machine, word) tuple (M, w) and produces the encoding of a single machine M'.

The behavior of M' is as follows:

$M'[v]$:
1. memorize v
2. erase v
3. write w
4. simulate $M[w]$
5. erase any output and write v
6. if v is a palindrome:
7. transition to Y
8. else:
9. transition to N

To "memorize v", you can imagine that M' has some separate tape on which it can copy its input; a tape whose contents are not affected by the later simulation of $M[w]$.

After simulating $M[w]$, M' simply acts like some machine that decides whether v is a palindrome. We know that such machine exists, because $\texttt{PALINDROME} \in R$.

M' will correctly decide whether v is a palindrome, but only if it can go past line 4; i.e. if the simulation of $M[w]$ halts. So if we can tell if M' correctly decides whether its input is a palindrome, we can tell if $M[w]$ halts. Knowing that `HALT` is undecidable, leads us to the conclusion that `P` is also undecidable.

But is there anything special about checking if an input is a palindrome? Couldn't we just as well replace the final "if" of M' with something like "if v ends with a 0", or "if v has more than eight symbols" and obtain a similar

result? Namely, that deciding whether a given Turing Machine decides whether its input ends with 0, or if it has more than eight symbols, is impossible?

This is what Rice's theorem tells us.

2 Informal statement

One way to phrase Rice's theorem is the following:

"Any non-trivial property of acceptable problems is undecidable."

This sentence contains some key terms whose meaning is not obvious, so we will unpack them one by one.

2.1 "... property of acceptable problems..."

A *"property"* is something that an object either has, or doesn't. We can model a property using a set: the elements of the set are exactly the objects who have the property. Having the property means being an element of that set, not having the property means not being an element of that set. A property of "problems" is thus a set of problems.

2.2 "... property [...] is undecidable..."

With each set of decision problems, we can associate a "yes" or "no" question regarding membership to that set. I.e. given a decision problem f and a set of decision problems $P \subseteq \mathbb{D}$, we have the questions: *"$f \in P$?"*.

"Yes" or "no" questions remind us of decision problems, which map their inputs to the range $\{FALSE, TRUE\}$. The problem is that we specified the *domain* of decision problems to be the set of all strings over some alphabet: Σ^*. At the lecture on undecidability, we established that the cardinality of all decision problems is strictly greater than the cardinality of all strings, so we cannot have an encoding that maps an arbitrary decision problem to some string. Thus, an arbitrary decision problem cannot be the *"input"* of some other decision problem.

However, there is an interesting subset of decision problems that **can** be represented by strings, namely the *recursively-enumerable problems*. The string representation of such a problem is simply the encoding of a machine that accepts it.

We can have a decision problem which takes as input the encoding of a Turing Machine and maps it to $TRUE$, if the problem accepted by that machine has a certain property, or to $FALSE$ if the problem accepted by that machine doesn't have a certain property.

We abuse a bit the language and say that a "property is undecidable"; but, more clearly, we mean that the decision problem associated with said property (i.e. "does f have property P") is undecidable.

2.3 "... non-trivial property..."

We use "trivial" here in a precise sense: a property is trivial if either:

- all objects have that property
- no objects have that property

In the previous section we established that our "objects" are acceptable problems (represented by the encoding of a machine) and properties are subsets of RE.

Thus, the "trivial properties" are the following two subsets:

- $\emptyset$
- RE

The decision problems associated with them are, respectively:

- $f_\emptyset(enc(M)) = FALSE, \forall M$ (because no acceptable problem is a member of $\emptyset$)
- $f_{RE}(enc(M)) = TRUE, \forall M$ (because every acceptable problem is a member of RE)

These problems are clearly decidable:

- $f_\emptyset$ is decided by the machine which, from the initial state, upon reading any symbol, transitions to N.

- f_{RE} is decided by the machine which, from the initial state, upon reading any symbol, transitions to Y.

Which is why Rice's theorem mentions that only the **non-trivial** ones are undecidable.

We can now rigorously capture the informal description given about into a formal statement of Rice's theorem.

3 Formal statement

Theorem 3.1. Let S be a subset of the acceptable problems, $S \subseteq RE$.
Let f_S be the decision problem:

$$f_S(enc(A_p)) = \begin{cases} TRUE, & p \in S, \ (p \text{ is the problem accepted by } A_p) \\ FALSE & otherwise \end{cases}$$

Then:

$$f_S \in R \Leftrightarrow (S = \emptyset \vee S = RE)$$

$f_S \in R \Rightarrow (S = \emptyset \vee S = RE)$

$(S = \emptyset \vee S = RE) \Rightarrow f_S \in R$

4 Proof

We've already proven the left implication: $(S = \emptyset \vee S = RE) \Rightarrow f_S \in R$. In subsection 2.3, we constructed machines that decide $f_\emptyset$ and f_{RE}.

We now prove the contrapositive of the right implication:

$$(S \neq \emptyset \wedge S \neq RE) \Rightarrow f_S \notin R$$

Any problem in RE is either part of S, or it is not. Because the left-hand-side contains the condition that $S \neq RE$, then there must be at least some problem which is not part of S.

We'll start our proof with the assumption that the trivial problem whose answer is $FALSE$ for any input is not part of our set: $f_{FALSE} \notin S$. Later on, we'll see how we can address that case where $f_{FALSE} \in S$.

Because the left-hand-side also tells us that $S \neq \emptyset$, then there must be some decision problem $g \in S^\dagger$. Remember that $S \subseteq RE$, so any problem in S is acceptable; this means that there exists a Turing Machine M_g that accepts g.

We show that f_S is undecidable, by reducing the halting problem to it. I.e. we prove this statement:

$$\texttt{HALT} \leq_m f_S$$

Our transformation has to turn any (M, w) pair such that $M[w]$ halts into a machine which accepts a problem from S, and any (M, w) pair such that $M[w]$ doesn't halt into a machine which accepts a problem in $RE \setminus S$.

The transformation will take the encoding of any (M, w) and turn it into the encoding of a machine that acts as follows:

$M'[v]$:
1. memorize v
2. erase v
3. write w
4. simulate $M[w]$
5. erase any output and write v
6. simulate $M_g[v]$

If $\texttt{HALT}(enc((M, w))) = TRUE$, then $M[w]$ halts; this means that $M'[v]$ finishes the simulation on line 4 and proceeds to simulate $M_g[v]$ and have the same behavior. Because M_g accepts $g \in S$, M' also accepts g, thus $f_S(enc(M')) = TRUE$.

[†] There might be more, even an infinite number! We are interested only in one of them, chosen arbitrarily.

On the other hand, if $\texttt{HALT}(enc((M,w))) = FALSE$, then $M[w]$ doesn't halt; this means that $M'[v]$ never finishes the simulation on line 4. $M'[v]$ never reaches the final state Y, so it doesn't accept any input. However, each machine accepts *a problem*; the problem accepted by M' is f_{FALSE}. We started our proof with the assumption that $f_{FALSE} \notin S$, so $f_S(enc(M')) = FALSE$.

The point of the construction is that either M' has the same behavior as M_g, accepting g (which is in S), or never halts for any input, accepting f_{FALSE} (which is not in S).

Our demonstration hinges on the fact that $f_{FALSE} \notin S$. But what if $f_{FALSE} \in S$? Well, then we simply consider $RE \setminus S$ as our set and start from there (i.e. we pick a problem $g \in RE \setminus S$). This would give us a proof that $\texttt{HALT} \leq_m f_{RE\setminus S}$, thus $f_{RE\setminus S} \notin R$. But $f_{RE\setminus S}$ is simply the complement of f_S. Remember from the previous lecture that: $f \notin R \Leftrightarrow \overline{f} \notin R$. So $f_S \notin R$.

Note that our proof relies on the fact that our considered subset S contains at least one problem (g) and doesn't contain at least one problem f_{FALSE}. This is true of any subset that is not $\emptyset$ or RE.

5 References and further reading

The theorem presented here was elaborated by Henry Gordon Rice (1920-2003) in his doctoral disertation and in a 1953 article titled "Classes of Recursively Enumerable Sets and Their Decision Problems" [1]. Having been written so early, the paper uses very different notations and conventions than in modern contexts, which makes it quite difficult to read.

The definition and proof presented in this lecture are adapted from subhchapter 9.3.3 of the textbook "'Introduction to Automata Theory, Languages, and Computation" [2].

Bibliography

[1] Henry Gordon Rice. "Classes of recursively enumerable sets and their decision problems". In: *Transactions of the American Mathematical society* 74.2 (1953), pp. 358–366.

[2] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2007. Chap. 9.3.3 Rice's Theorem and Properties of the RE Languages, pp. 397–399.