Lecture 13

Ben Rosenberg

June 28, 2021

Proof that the decidable version of the Universal Turing Machine cannot exist

Recall:

Definition 1: Language L is recursively enumerable if there exists a Turing machine that accepts L.

Definition 2: Language L is decidable if there exists a Turing machine that accepts L and halts on every input.

Definition 3: $L_H = \{(M, w) | M \text{ is a Turing machine that halts on input } w\}$

Theorem 1: If L_H is recursively enumerable, then the Universal Turing Machine M_U accepts it.

$$M_U(M, w) \to \begin{cases} \text{halt if} & M(w) \searrow \\ \text{diverge if} & M(w) \nearrow \end{cases}$$

Let M_H be the Turing Machine that *decides* L_H:

$$M_H(M, w) \to \begin{cases} \text{halt and accept if} & M(w) \searrow \\ \text{halt and reject if} & M(w) \nearrow \end{cases}$$

Theorem 2: M_H does not exist.

Proof: Assume M_H exists. We will construct a new machine M_1 from M_H as follows:

$$M_1(M, w) \to \begin{cases} \text{diverge if} & M(w) \searrow \\ \text{halt if} & M(w) \nearrow \end{cases}$$

Recall that the machine M_H accepts if it finds itself in some state-symbol pair for which there is no transition in δ, and the state is an accepting state in F. In M_1, we simply need to "upgrade" these pairs to escape from the acceptance of M_H.

We need not touch the rejecting states, as they will all halt and therefore be rejected.

Now, we construct Turing Machine D from M_1 as follows:

$$D(X) = M_1(X, X)$$

We see that:

$$D(X) \to \begin{cases} \text{diverge if} & X(X) \searrow \\ \text{halt if} & X(X) \nearrow \end{cases}$$

What will happen on $D(D)$?

We will have the following:

$$D(D) \to \begin{cases} \text{diverge if} & D(D) \searrow \\ \text{halt if} & D(D) \nearrow \end{cases}$$

This is evidently a contradiction. We cannot halt if it diverges or diverge if it halts, and yet the above statement implies that this must be the case.

As such, M_H cannot exist.

□

Thus, we cannot always have the operating system that we would like to have; namely, M_H.

More consequences:

- L_H is recursively enumerable, but ***not*** decidable, and
- M_H that decides L_H does not exist.

~~Church-Turing thesis~~

How does a person apply an algorithm?

Consider a gargantuan notebook. In the beginning of the book, within finitely many pages, we have the algorithm written out, like a recipe. Thereafter, we have empty pages *forever*. The person operates by opening the algorithm pages, and seeing how to go about it. They alternate between reading some pages and writing some pages in the blank space.

A page is a finite set of pixels, and thus there are only finitely many possible contents of each page. So, every possible page in this book, whether it's an algorithm page, or input page, or intermediate result page, is merely one of these finitely many possible pages.

This is exactly what a Turing Machine does.

And so, a person applying an algorithm does not do anything other than what a Turing Machine can do. As such, a Turing Machine is as good as anything else that is yet conceivable.

Aside: L_H is also called the Halting Problem.

Coming up with other languages that cannot be decided but are recursively enumerable

We cannot reach M_H. We do not know whether we can reach some collection of problems M_β, but if we can build M_β we can build M_H. As such, we will not be able to reach M_β. (This is a reduction.)

Definition 4: A property of recursively enumerable languages is that the language is a *predicate* whose domain is the class of recursively enumerable languages.

Examples $P_1(L) \iff L =$ is empty

- $P_1(\emptyset) = 1$
- $P_1(\text{anything else}) = 0$

$P_2(L) \iff L$ contains the empty string

- $P_2(\emptyset) = 0$
- $P_2(a^*) = 1$
- ...

$P_3(L) \iff L$ is infinite

- $P_3(\emptyset) = 0$

- $P_3(a^*) = 1$
- $P_3(a) = 0$
- $P_3((aa)^*) = 1$

$P_4(L) \iff L$ consists of an even number of elements

- $P_5(\emptyset) = 0$
- $P_5(a^*) = 1$
- $P_5(a) = 0$
- $P_5(a \cup b) = 0$
- $P_5(\lambda) = 0$

$P_6(L) \iff L$ contains a string of even length

- $P_6(\emptyset) = 1$ (vacuous truth)
- $P_6(a^*) = 0$
- $P_6(a) = 0$
- $P_6(a \cup b) = 0$
- $P_6((aa)^*) = 1$
- $P_6((a \cup b)^*) = 0$
- $P_6(\lambda) = 1$

Rice's Theorem

Definition 5: A property, say, α, of recursively enumerable languages, is **nontrivial** if there exists at least one recursively enumerable languages for which α is true and at least one recursively enumerable language for which α is false.

Otherwise, if α sends all recursively enumerable languages to the same value (whether it be true or false) α is trivial.

Example: all of the above properties in the examples are nontrivial.

Other examples:

- regular languages
- context-free languages
- decidable languages (ex. L_H is not decidable but is recursively enumerable)
- is a specific regular language (ex. complement of that language)

Trivial properties:

- is recursively enumerable (true)
- is uncountably infinite (false)
- anything that is a negation of a tautology (always false) or a tautology (true)

Evidently, nontrivial properties are significantly more interesting.

Rice's Theorem (definition)

Theorem 3: Let β be a nontrivial property of recursively enumerable languages such that $\beta(\emptyset) = 0$. Define a Turing Machine M_β as follows:

$$M_\beta(M) = \begin{cases} \text{halt and accept if} & \beta(L(M)) = 1 \\ \text{halt and reject if} & \beta(L(M)) = 0 \end{cases}$$

In brief: $M_\beta(M) = \beta(L(M))$

$M\beta$ **does not exist.**

Note: we can always just rewrite β to be equal to $\neg\beta$, so that $\beta(\emptyset)$ is always 0.

So we have:

- M is the argument Turing Machine
- $L(M)$ is the language accepted by M
- $\beta(L(M))$ is the value of β for $L(M)$
- $M_\beta(M) = \beta(L(M))$

Proof that M_β **does not exist**: (proves that we cannot convert between a regular expression and a Turing Machine's language)

We will use M_β to build M_H. Once we use M_β to build M_H, we will have proven that M_β cannot exist.

Let L_1 be a recursively enumerable language such that $\beta(L_1) = 1$ (is true). Observe that $L_1 \neq \emptyset$ by the above definition of β.

Let M_1 accept L_1 by halting. (M_1 accepts L_1 because there has to be one that accepts L_1 because it is recursively enumerable.)

Our M_H operates as follows:

$M_H(M, w)$:

- construct Turing Machine D
  - $D(x)$:
    - run $M(w)$ until x; then,
    - run $M_1(x)$
- return $M_\beta(D)$

Indeed, M_H gives the correct answer.

Case 1: $M(w) \nearrow$

- $M(w) \nearrow \implies D(x) \nearrow$ because D will never reach x as it will be busy with $M(w)$.
  - This implies that $L(D) = \emptyset$. But then $\beta(L(D)) = \beta(\emptyset) = 0$. And then, $M_\beta(D) = \beta(L(D)) = \beta(\emptyset) = 0$
  - And so, $M_H(M, w) = M_\beta(D) = 0$.

Case 2: $M(w) \searrow$

- $M(w) \searrow \implies D(x) = M_1(x) \implies L(D) = L(M_1) = L_1 \implies \beta(L(D)) = \beta(L_1) = 1 \implies M_\beta(D) = M_\beta(L(D)) = \beta(L_1) = 1$.

And so, our machine gives the correct answer of 0 or 1 depending on whether M halts on w or not.