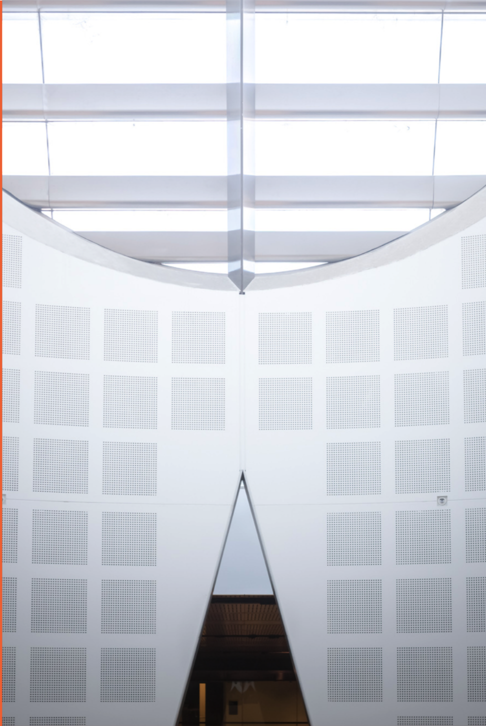
COMP2022
Models of Computation

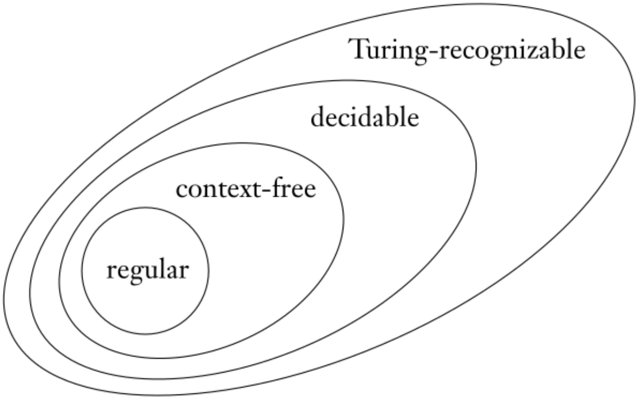
Undecidability
Sipser Chapter 4

Sri AravindaKrishnan Thyagarajan (Aravind)

September 19, 2024

THE UNIVERSITY OF SYDNEY





Today we show:

1. There is a non-recognisable language.
2. There is a recognisable language that is not decidable.

Which of the following approaches can be used to show that a language L is not Turing-recognisable?

1. Show that there is some TM B such that $L = L(B)$
2. Show that for every TM B there is some string x such that either ($x \in L(B)$ and $x \notin L$) or ($x \notin L(B)$ and $x \in L$).

What does the following do?

```python
def M1(x):
  return not("hello world" in x)

s = """
def M1(x):
  return not("hello world" in x)
"""

print(M1(s))
```

1. It prints `True`
2. It prints `False`
3. There is a syntax error, you can't input a function to itself

Kleene: Here is a slightly crazy language...

$$L_{\mathsf{Diag}} = \{\mathrm{Source}_M : M \text{ is a TM that does not accept } \mathrm{Source}_M\}$$

Student: Not so crazy. For instance, $\mathrm{Source}_{M1} \in L_{\mathsf{Diag}}$

K: Right. The big question... Is L_{Diag} recognisable?

S: I guess "No"... because I don't see how a TM can check non-acceptance...

K: Ok, but how could we go about **proving** there isn't a TM for L_{Diag}?

S: Well, for every TM B we would have to find an input x that shows that $L(B)$ and L_{Diag} disagree, i.e., either

1. x is accepted by B but x is not in L_{Diag}, or
2. x is not accepted by B but x is in L_{Diag}.

K: Right! What could such a disagreeing input be...?

A non-recognisable language

Theorem

$$L_{\mathsf{Diag}} = \{\mathrm{Source}_M : M \text{ is a TM that does not accept } \mathrm{Source}_M\}$$

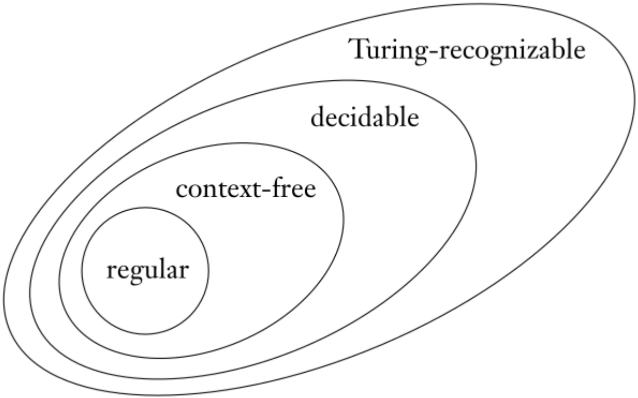
is not recognisable.

Proof. Let B be any TM. We will show $L(B) \neq L_{\mathsf{Diag}}$ by showing that they disagree on the input $x = \mathrm{Source}_B$.

There are two cases: either $\mathrm{Source}_B \in L_{\mathsf{Diag}}$ or not.

1. if $\mathrm{Source}_B \in L_{\mathsf{Diag}}$ then B does not accept Source_B, so $\mathrm{Source}_B \notin L(B)$.
2. if $\mathrm{Source}_B \notin L_{\mathsf{Diag}}$ then B does accept Source_B, so $\mathrm{Source}_B \in L(B)$. $\square$

Illustrating the proof: Diagonalisation



Today we show:

1. There is a non-recognisable language. ✓
2. There is a recognisable language that is not decidable.

K: Now we want to find a recognisable language that is not decidable. Can you think of a candidate language?

S: Well, I remember that

$$L_{\textsf{TM-acceptance}} = \{\mathrm{Source}_M, w \mid M \text{ is a TM that accepts } w\}$$

is recognisable. And, I don't think it is decidable... but I don't see how to prove this.

K: To prove it, we will show that if it were possible to decide $L_{\textsf{TM-acceptance}}$ then it would be possible to recognise $L_{\textsf{Diag}}$, which we know is impossible.

S: How would this work?

K: We suppose there is a decider A for $L_{\textsf{TM-acceptance}}$ and we use it to build a TM B that recognises $L_{\textsf{Diag}}$, which we know is impossible. So we conclude that our original assumption that there is a decider for $L_{\textsf{TM-acceptance}}$ must be false. So the language is not decidable.

Theorem (Turing)

There is no TM A that decides $L_{\textit{TM-acceptance}}$

Proof. Assume A exists. We use A to define another TM B:

```python
def B(x):
  return not A(x,x)
```

So, for every TM M, the following are equivalent statements:

- B accepts Source_M
- A does not accept $\mathrm{Source}_M, \mathrm{Source}_M$
- M does not accept Source_M
- $\mathrm{Source}_M \in L_{\mathsf{Diag}}$

In other words, B decides L_{Diag}, which we already saw is impossible. So A cannot exist. $\square$

Summary

- $L_{\text{TM-acceptance}}$ is Turing-recognisable but not Turing-decidable.
- By the Church-Turing Thesis, this means that no program or computer can decide $L_{\text{TM-acceptance}}$.
- The same argument shows that $L_{\text{Python-acceptance}}$ is Python-recognisable but not Python-decidable.

There are two ways to show that a language L is undecidable:

1. Directly, using a diagonalisation argument.
2. Indirectly by assuming that there is a decider A for L, building an algorithm that using A decides a known undecidable problem.

You will take this approach in the tutorial where you will show that the following problems are undecidable:

– "Does M halt on input w?" – this is called the halting problem
– "Is $L(M)$ regular?"

Good to know

Actually, there are lots of undecidable languages/problems.

- Rule of thumb: any problem that talks about the **languages of TMs** M is probably undecidable!
- Also some problems about simpler models than TMs are undecidable

  (The equivalence problem for CFGs is undecidable!)
- Also some problems about logic

  (We will mention one when we cover predicate logic)
- In fact, there are even degrees of uncomputability,

  which are studied in an area called, strangely enough, computability theory.