COMP2022|2922
Models of Computation

Undecidability
Sipser Chapter 4

Sasha Rubin

September 8, 2022

THE UNIVERSITY OF SYDNEY

Recap

- A language L is Turing-recognisable if $L = L(M)$ for some TM.
- A language L is Turing-decidable if $L = L(M)$ for some TM that halts on all inputs (aka M is a decider).
- If L is Turing-decidable then it is also Turing-Recognisable.
- Is the converse true?
  - We saw that the acceptance problem for TMs is recognisable.
  - We now show that it is not decidable.

Recall that if M is a TM then $\langle < M \rangle >$ is a string representing that machine (e.g., in the text format used in this course).

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ is a TM that accepts $w\}$ is not decidable.

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ *is a TM that accepts* $w\}$ *is not decidable.*

- Suppose there is a decider for this language, call it A.

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ *is a TM that accepts* $w\}$ *is not decidable.*

- Suppose there is a decider for this language, call it A.
- Use TM A to build a new TM B whose inputs are TMs:

B does the following on input $\langle M\rangle$:

1. Run A on input $\langle M, w\rangle$ where $w = \langle M\rangle$.
2. Wait for A to accept or reject.
3. Output the opposite of A's result.

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ *is a TM that accepts* $w\}$ *is not decidable.*

- Suppose there is a decider for this language, call it A.
- Use TM A to build a new TM B whose inputs are TMs:

> B does the following on input $\langle M\rangle$:
>
> 1. Run A on input $\langle M, w\rangle$ where $w = \langle M\rangle$.
> 2. Wait for A to accept or reject.
> 3. Output the opposite of A's result.

- TM B has the following property:

$$B(\langle M\rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M\rangle \\ reject & \text{if } M \text{ does accept } \langle M\rangle \end{cases}$$

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ *is a TM that accepts* $w\}$ *is not decidable.*

- Suppose there is a decider for this language, call it A.
- Use TM A to build a new TM B whose inputs are TMs:

> B does the following on input $\langle M\rangle$:
>
> 1. Run A on input $\langle M, w\rangle$ where $w = \langle M\rangle$.
> 2. Wait for A to accept or reject.
> 3. Output the opposite of A's result.

- TM B has the following property:

$$B(\langle M\rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M\rangle \\ reject & \text{if } M \text{ does accept } \langle M\rangle \end{cases}$$

- Crazy question: what happens when we run B on $\langle B\rangle$?

Theorem (Turing)

$\{\langle M, w\rangle \mid M$ *is a TM that accepts* $w\}$ *is not decidable.*

- Suppose there is a decider for this language, call it A.
- Use TM A to build a new TM B whose inputs are TMs:

> B does the following on input $\langle M\rangle$:
>
> 1. Run A on input $\langle M, w\rangle$ where $w = \langle M\rangle$.
> 2. Wait for A to accept or reject.
> 3. Output the opposite of A's result.

- TM B has the following property:

$$B(\langle M\rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M\rangle \\ reject & \text{if } M \text{ does accept } \langle M\rangle \end{cases}$$

- Crazy question: what happens when we run B on $\langle B\rangle$?
- We get a contradiction! Conclude A does not exist!... Let's see this contradiction...

What is the contradiction?

We saw B has the following property:

$$B(\langle M \rangle) = \begin{cases} accept & \text{if } M \text{ does not accept } \langle M \rangle \\ reject & \text{if } M \text{ does accept } \langle M \rangle \end{cases}$$

So, in particular, taking M to be the TM B itself, we have that:

$$B(\langle B \rangle) = \begin{cases} accept & \text{if } B \text{ does not accept } \langle B \rangle \\ reject & \text{if } B \text{ does accept } \langle B \rangle \end{cases}$$

This is obviously impossible!

Summary

- There is a Turing-recognisable language that is not Turing-decidable.
- The proof is subtle, and uses self-reference.
- The same proof idea can be applied to any programming language to conclude that there is a language that can't be decided by any program in that language (tutorial).

  Halting problem in Python is undecidable

Good to know

- Actually, there are lots of undecidable languages/problems.
- Rule of thumb: any problem that talks about the *languages of TMs* is probably undecidable!
  - "Is $w \in L(M)$?"
  - "Is $L(M)$ regular?"
  - "Is $L(M_1) = L(M_2)$?"
- Also some problems about simpler models than TMs are undecidable

  (we will mention one when we cover Grammars)
- Also some problems about logic

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

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

Closure properties

- We know the regular languages are closed under Boolean operations and the regular operations.
- What about the decidable languages?
- What about the recognisable languages?

You will look at closure properties of recognisable languages in the tutorial. Let's get started...

Closure properties

Theorem
The decidable languages are closed under union.

Proof Let L_1 and L_2 be languages that are decided by TMs M_1 and M_2 respectively. We construct a TM that decides $L_1 \cup L_2$ as follows.

On input w:

1. Run M_1 and wait for it to halt.
2. If M_1 it accepts then accept. Otherwise continue.
3. Run M_2 and wait for it to halt.
4. If M_2 accepts then accept, otherwise reject.

Closure properties

Theorem

The decidable languages are closed under complement.

Proof.

- Suppose L is decided by a deterministic TM M.
- Define M' to be the same as M with q_{accept} and q_{reject} swapped.
- M' decides $\Sigma^* \setminus L$.

Corollary

The decidable languages are closed under intersection.

Closure properties

Theorem

A language is decidable iff it and its complement are recognisable.

Proof idea.

$\Rightarrow$: follows from the closure properties of decidable languages.

$\Leftarrow$: We construct a decider for L from a TM M_1 that recognises L and a TM M_2 that recognises $\Sigma^* \setminus L$ as follows. On input w:

- Run M_1 and M_2 in parallel on w.
  - use one tape for each machine.
  - at each step, simulate M_1 on tape 1 and M_2 on tape 2.
- If M_1 ever accepts, then accept; and if M_2 ever accepts then reject.

The point is that exactly one of M_1, M_2 must eventually accept w, and so we have built a decider for L.

So what?

Corollary

The language

$$L_{\textit{TM-non-acceptance}} = \{\langle M, w\rangle \mid M \textit{ is a TM that does not accept } w\}$$

is not recognisable.

Proof Idea

- Suppose it were.
- Then both $L_{\text{TM-acceptance}}$ and its complement are recognisable.[1]
- Hence $L_{\text{TM-acceptance}}$ is decidable.
- But we know this is not the case (Turing).

[1] There is an issue here that I am hiding. See Tutorial.

The halting problem is undecidable

Theorem
The language

$$L_{TM\text{-}Halting} := \{\langle M, w\rangle \mid M \text{ is a TM that halts on input } w\}$$

is recognisable, but not decidable.

See tutorial.

COMP2022|2922
Models of Computation

Intro to Computational Complexity
Sipser Chapter 7

Sasha Rubin

September 8, 2022

THE UNIVERSITY OF SYDNEY

Agenda

Investigate the time (= number of steps) taken by TMs to solve problems.

- What is time complexity?
- What is the **P** vs **NP** problem?

Recall that a polynomial in the variable n is an expression of the form

$$\sum_{i=0}^{k} a_i n^i = a_k n^k + \cdots + a_1 n^1 + a_0 n^0$$

where each a_i is a real number and $k \in \mathbb{N}$.

e.g., $2n^3 - 3n + 1$ is a polynomial.

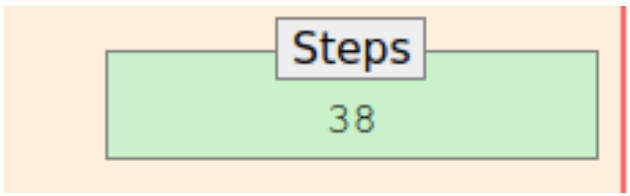
How many steps does an algorithm take?

What counts as a step for pseudocode?[2]

- assignments $a = 42$
- comparisons $i < j$
- Boolean formulas $(p \& q) \& q$
- mathematical operations $a = 42x + y$

What counts as a step for Turing machines?

- A single transition counts as a step.
- The number of steps of a run on a given input is computed by the simulator http://morphett.info/turing/



[2] This is how we do it in the DS+A course.

What is time complexity?

- How many steps does an algorithm/TM take? It depends on the input.
- We consider the number of steps as a function of the length (size) of the input.
- We consider the largest number of steps for all inputs of a particular length.
- This is captured by a function $f : \mathbb{N} \to \mathbb{N}$ called the time complexity of the algorithm.

  $f(n) =$ the largest number of steps taken by the algorithm on any input of length n.
- We say that the algorithm/TM runs in time $f(n)$

What is time complexity?

```
1  def(array A of integers):
2    for i = 1 to size(A)-1:
3      for j = i+1 to size(A):
4        if A[i] > A[j] then swap A[i] with A[j]
5    return A
```

- On strings of size n, line 4 is executed $(n-1)+(n-2)+\cdots+2+1 = n(n-1)/2$ steps.
- So, the time complexity of this algorithm is $\Theta(n^2)$, i.e., quadratic.[3]
- In particular, it is a quadratic-time algorithm.

BTW, what does this algorithm do?

[3] Recall that a function f is $\Theta(g)$, read "Big-Theta g", if there are constants c, d, e such that $cg(n) \leq f(n) \leq dg(n)$ for all $n > e$.

What is time complexity?

Recall:

- TM recognising the language $\{0^n 1^n : n \geq 1\}$
- Idea: match leftmost 0 with leftmost 1, replacing them by X and Y respectively, and repeat.
- link to TM

The time complexity of this TM is $\Theta(n^2)$. We say:

- This is a quadratic-time TM, or
- The time-complexity of this TM is quadratic, or
- This TM runs in quadratic time, or
- This language can be decided in quadratic time.

The most important class of languages

Definition

The collection of all languages that are decided by TMs that run in polynomial time is denoted **P** (read "P" or "P Time" or "Poly Time" or "Polynomial Time")

- **P** includes all the languages decided by linear-time algorithms, and quadratic-time algorithms, and cubic-time algoriths, etc.
- **P** is robust to certain changes in the model, notably multiple tapes.
- **P** roughly corresponds to the problems that can be realistically solved on a computer.

Examples of languages in P

- Every regular language.
- $\{0^n 1^n : n \geq 1\}$
- The language of encodings $\langle a_1, a_2, \cdots, a_k \rangle$ of sorted sequences of integers (aka "sorting").
- Some of the problems studied in this course, *e.g.*, DFA membership, RE membership, DFA equivalence.
- Most of the problems studied in
  - `COMP2123:Data Structures and Algorithms`
  - `COMP3027:Algorithm Design`

Self-test

The halting-problem is not in **P** because...

Vote now! (on mentimeter)

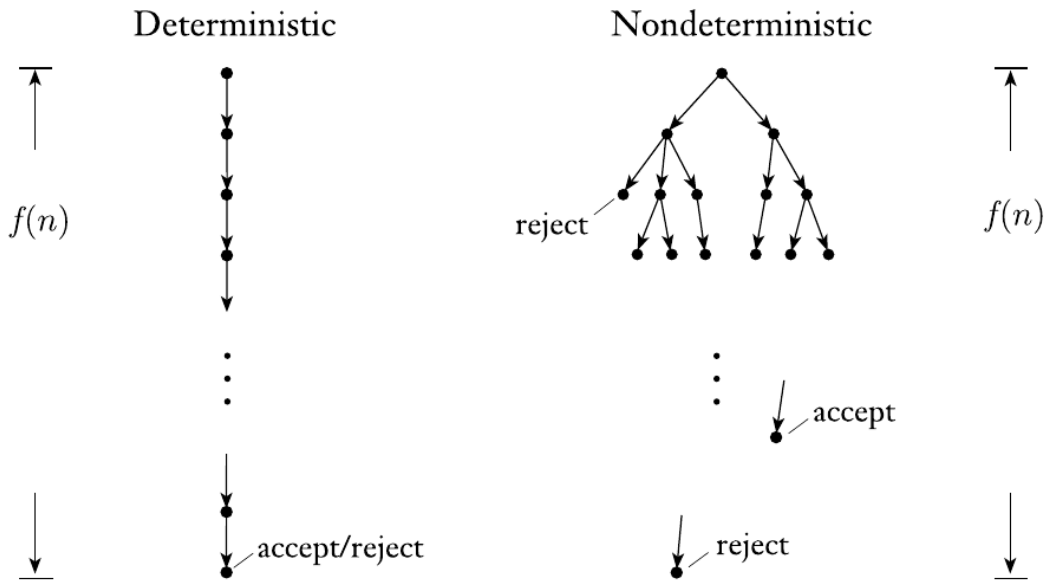
1. It is recognisable, and some recognisable languages are in **P**.
2. It is not decidable, but every language in **P** is decidable.
3. Actually, it is in **P**!

What is time complexity for NTMs?

Definition

An NTM N is a decider if on every input, every branch of its computation tree halts.

The time-complexity of N is the the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that N uses on any branch of its computation on any input of length n.



The second most important class of languages

Definition
The collection of all languages that are decided by nondeterministic TMs that run in polynomial time is denoted NP (read "NP" or "Nondeterministic Polynomial").[4]

Many many natural languages/problems are in **NP**, including:

- all languages that are in **P**. Why?
- the clique problem (CLIQUE)
- (lots others, we will see some later)

[4] It does not stand for "Not Polynomial"

Graphs (AK)

A graph G is a pair (V, E) where

- V is a set of vertices
- E is a set of pairs of vertices, called edges.

  we say the vertices are connected by an edge.
- We consider undirected edges

  each edge is an unordered pair of vertices $\{u, v\}$

Example:

- A vertex represents a city
- An edge represents a two-way road between two cities

A non-empty graph is called a clique (aka "completely connected") if every pair of different nodes is connected by an edge.

CLIQUE is in NP

The CLIQUE problem: given (V, E) and $K \in \mathbb{N}$, decide if the graph contains a clique on K vertices.

To show that CLIQUE is in **NP**, we only need to give a polynomial time NTM that decides it.

- Here is a high-level description of such a TM:

On input $\langle (V, E), K \rangle$:

1. Nondeterministically select a subset $C \subseteq V$ of K vertices.
2. Test whether every pair of different nodes in C is connected by an edge.
3. If yes, *accept*; else *reject*

The most(?) important unsolved problem in computer science

Is **P** equal to **NP**?

The most(?) important unsolved problem in computer science

Is **P** equal to **NP**?

What can we conclude about this problem?

- Every language in **P** is in **NP** (obvious)
- Every language in **NP** can be decided in deterministic exponential time (tutorial)

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP}$$

- It is known that $\mathbf{P} \neq \mathbf{EXP}$.

Good to know (i)

- There are "hardest" problems in **NP**, so called **NP**-complete problems, and showing that any of these is in **P** would show $\mathbf{P} = \mathbf{NP}$
- You can read more about the problem and the prize for solving it at the Clay Math Institute.

Good to know (ii)

- You will study **NP** and **NP**-complete problems in `COMP3027:Algorithm Design`.
- There you will study an equivalent definition of **NP** in terms of TMs called "verifiers" ...