CS21 Decidability and Tractability

Lecture 12
February 1, 2023

1

Outline

- a non-RE language
- the halting problem
- RE and co-RE
- Reductions
- many-one reductions

2

non-RE languages

Theorem: there exist languages that are not Recursively Enumerable.

Proof outline:

- the set of all TMs is countable
- the set of all languages is uncountable
- the function L:{TMs} →{languages} cannot be onto

3

non-RE languages

- Lemma: the set of all TMs is countable.
- Proof:
  - each TM M can be described by a finite-length string <M>
  - can enumerate these strings, and give the natural bijection with **N**

4

non-RE languages

- Lemma: the set of all languages is uncountable
- Proof:
  - fix an enumeration of all strings $s_1, s_2, s_3, \ldots$ (for example, lexicographic order)
  - a language L is described by its characteristic vector χ_L whose i^{th} element is 0 if s_i is not in L and 1 if s_i is in L

5

non-RE languages

- suppose the set of all languages is countable
- list characteristic vectors of all languages according to the bijection f:

n	f(n)
1	0101010…
2	1010011…
3	1110001…
4	0100011…
…	

6

non-RE languages

– suppose the set of all languages is countable

– list characteristic vectors of all languages according to the bijection f:

n	f(n)
1	0101010…
2	1010011…
3	1110001…
4	0100011…
…	

set x = 1101…

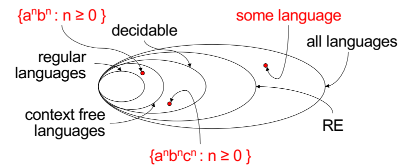
where i^{th} digit ≠ i^{th} digit of f(i)

x cannot be in the list!

therefore, the language with characteristic vector x is not in the list

7

So far…



- This language might be an esoteric, artificially constructed one. Do we care?
- We will show a natural undecidable L next.

8

The Halting Problem

- Definition of the “Halting Problem”:
  HALT = { <M, x> : TM M halts on input x }
- HALT is recursively enumerable.
  – proof?
- Is HALT decidable?

9

The Halting Problem

Theorem: HALT is not decidable (undecidable).

Proof:

– Suppose TM H decides HALT

– Define new TM H’: on input <M>
  - if H accepts <M, <M>> then loop
  - if H rejects <M, <M>> then halt

10

The Halting Problem

Proof:

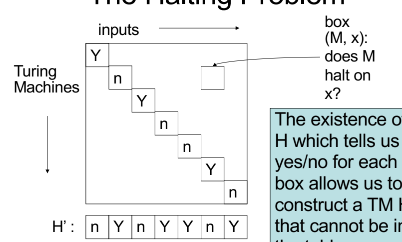
– define new TM H’: on input <M>
  - if H accepts <M, <M>> then loop
  - if H rejects <M, <M>> then halt

– consider H’ on input <H’>:
  - if it halts, then H rejects <H’, <H’>>, which implies it cannot halt
  - if it loops, then H accepts <H’, <H’>> which implies it must halt

– contradiction.

11

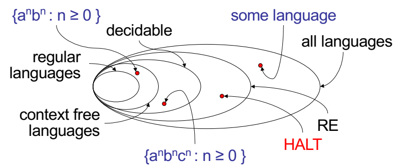
The Halting Problem



The existence of H which tells us yes/no for each box allows us to construct a TM H’ that cannot be in the table.

12

So far…

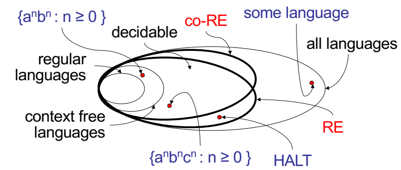


- Can we exhibit a natural language that is non-RE?

13

RE and co-RE

- The complement of a RE language is called a co-RE language



14

RE and co-RE

<u>Theorem</u>: a language L is decidable if and only if L is RE and L is co-RE.

Proof:

(⇒) we already know decidable implies RE
- if L is decidable, then complement of L is decidable by flipping accept/reject.
- so L is in co-RE.

15

RE and co-RE

<u>Theorem</u>: a language L is decidable if and only if L is RE and L is co-RE.

Proof:

(⇐) we have TM M that recognizes L, and TM M' recognizes complement of L.
- on input x, simulate M, M' in parallel
- if M accepts, accept; if M' accepts, reject.

16

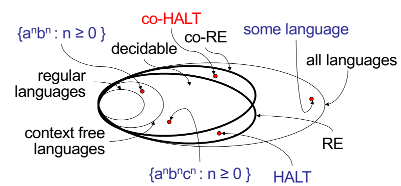
A natural non-RE language

<u>Theorem</u>: the complement of HALT is not recursively enumerable.

Proof:
- we know that HALT is RE
- suppose complement of HALT is RE
- then HALT is co-RE
- implies HALT is decidable. Contradiction.

17

Summary



Main point: some problems have no algorithms, HALT in particular.

18

Reductions

- Given a new problem NEW, want to determine if it is easy or hard
  - right now, easy typically means decidable
  - right now, hard typically means undecidable
- One option:
  - prove from scratch that the problem is decidable, or
  - prove from scratch that the problem is undecidable (dream up a diag. argument)

19

Reductions

- A better option:
  - to prove NEW is decidable, show how to transform it into a known decidable problem OLD so that solution to OLD can be used to solve NEW.
  - to prove NEW is undecidable, show how to transform a known undecidable problem OLD into NEW so that solution to NEW can be used to solve OLD.
- called a **reduction**

20

Reductions

Reductions are one of the most important and widely used techniques in theoretical Computer Science.

- especially for proving problems "hard"
  - often difficult to do "from scratch"
  - sometimes not known how to do from scratch
  - reductions allow proof by giving an algorithm to perform the transformation

21

Example reduction

- Try to prove undecidable:

  A_{TM} = {<M, w> : M accepts input w}
- We know this language is undecidable:

  HALT = {<M, w> : M halts on input w}
- Idea:
  - suppose A_{TM} is decidable
  - show that we can use A_{TM} to decide HALT
  - conclude HALT is decidable. Contradiction.

reduction

22

Example reduction

- How could we use procedure that decides A_{TM} to decide HALT?
  - given input to HALT: <M, w>
- Some things we can do:
  - check if <M, w> ∈ A_{TM}
  - construct another TM M' and check if <M', w> ∈ A_{TM}

23

Example reduction

- Deciding HALT using a procedure that decides A_{TM} ("reducing HALT to A_{TM}").
  - on input <M, w>
  - check if <M, w> ∈ A_{TM}
    - if yes, the M halts on w; **ACCEPT**
    - if no, then M either rejects w or it loops on w
  - construct M' by swapping q_{accept}/q_{reject} in M
  - check if <M', w> ∈ A_{TM}
    - if yes, then M' accepts w, so M rejects w; **ACCEPT**
    - if no, then M neither accepts nor rejects w; **REJECT**

24

Example reduction

- Preceding reduction proved:

Theorem: A_{TM} is undecidable.

Proof (recap):

- suppose A_{TM} is decidable
- we showed how to use A_{TM} to decide HALT
- conclude HALT is decidable. Contradiction.