CS21
Decidability and Tractability

Lecture 11

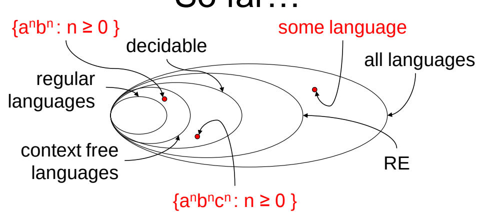
January 31, 2014

Outline

- the Halting Problem
- reductions
- many-one reductions

- undecidable problems
  - computation histories
  - surprising contrasts between decidable/undecidable

So far…



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

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?

The Halting Problem

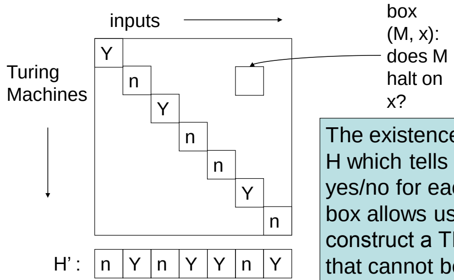
Theorem: HALT is not decidable (undecidable).

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

The Halting Problem

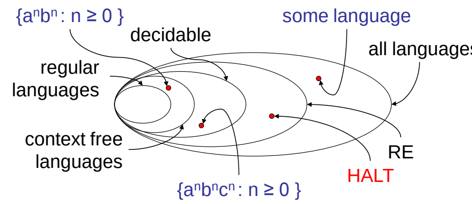
Proof:
- define new TM H': on input <M>
  - if H accepts <M, <M>> then loop
  - if H rejects <M, <M>> then halt
- consider H' on input <H'>:
  - if it halts, then H rejects <H', <H'>>, which implies it cannot halt
  - if it loops, then H accepts <H', <H'>> which implies it must halt
- contradiction.

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.

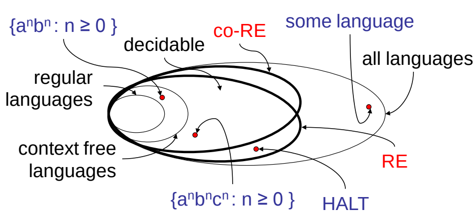
So far…



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

RE and co-RE

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



RE and co-RE

Theorem: a language L is decidable if and only if L is RE and L is co-RE.

Proof:

($\Rightarrow$) 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.

RE and co-RE

Theorem: a language L is decidable if and only if L is RE and L is co-RE.

Proof:

($\Leftarrow$) 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.

A natural non-RE language

Theorem: the complement of HALT is not recursively enumerable.

Proof:

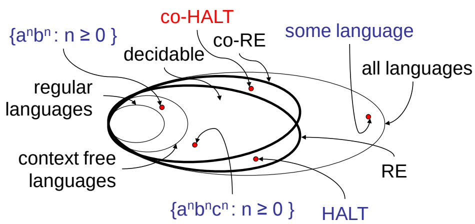
– we know that HALT is RE

– suppose complement of HALT is RE

– then HALT is co-RE

– implies HALT is decidable. Contradiction.

Summary



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

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)

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**

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

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

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> $\in A_{TM}$
  - construct another TM M' and check if <M', w> $\in A_{TM}$

Example reduction

- Deciding HALT using a procedure that decides A_{TM} ("reducing HALT to A_{TM}").
  - on input $\langle M, w\rangle$
  - check if $\langle M, w\rangle \in A_{TM}$
    - 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 $\langle M', w\rangle \in A_{TM}$
    - if yes, then M' accepts w, so M rejects w; **ACCEPT**
    - if no, then M neither accepts nor rejects w; **REJECT**

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.

Another example

- Try to prove undecidable:

  $E_{TM} = \{\langle M\rangle : L(M) = \emptyset\}$
- which problem should we **reduce from**?
  - HALT = $\{\langle M, w\rangle$: M halts on input w$\}$
  - $A_{TM} = \{\langle M, w\rangle$: M accepts input w$\}$
- Some things we can do:
  - check if $\langle M\rangle \in E_{TM}$
  - construct another TM M' and check if $\langle M'\rangle \in E_{TM}$

Another example

- We are given input $\langle M, w\rangle$
- We want to use a procedure that decides E_{TM} to decide if $\langle M, w\rangle \in A_{TM}$
- Idea:
  - check if $\langle M\rangle \in E_{TM}$
  - if not?
  - helpful if could make M reject everything except possibly w.

Another example

Is this OK? finite # of states?

- Construct TM M':
  - on input x, if $x \neq w$, then reject
  - else simulate M on x, and accept if M does.
- on input $\langle M, w\rangle$
  - construct M' from description of M
  - check if $M' \in E_{TM}$
    - if no, M must accept w; **ACCEPT**
    - if yes, M cannot accept w; **REJECT**

Another example

- Preceding reduction proved:

Theorem: E_{TM} is undecidable.

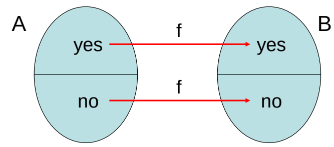
Proof (recap):
- suppose E_{TM} is decidable
- we showed how to use E_{TM} to decide A_{TM}
- conclude A_{TM} is decidable. Contradiction.

Definition of reduction

- Can you reduce co-HALT to HALT?
- We know that HALT is RE
- Does this show that co-HALT is RE?
  - recall, we showed co-HALT is not RE
- our current notion of reduction cannot distinguish complements

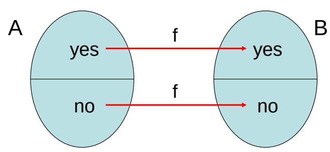
Definition of reduction

- More refined notion of reduction:
  - "many-one" reduction (commonly)
  - "mapping" reduction (book)



reduction from language A to language B

Definition of reduction



- function f should be computable

Definition: f : $\Sigma^* \rightarrow \Sigma^*$ is computable if there exists a TM M_f such that on every $w \in \Sigma^*$ M_f halts on w with f(w) written on its tape.