So far...

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

Regular languages

Context free languages

RE

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”:
  $\text{HALT} = \{ <M, x> : \text{TM M halts on input x} \}$

- $\text{HALT}$ is recursively enumerable.
  – proof?

- Is $\text{HALT}$ decidable?

Theorem: \text{HALT} is not decidable (undecidable).

Proof:
- Suppose TM H decides $\text{HALT}$
- Define new TM $H'$: on input $<M, x>$
  - 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 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.

$(\Leftarrow)$ 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)

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.

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:
  \( A_{TM} = \{<M, w> : M \text{ accepts input } w\} \)
• We know this language is undecidable:
  \( \text{HALT} = \{<M, w> : M \text{ halts on input } w\} \)
• Idea:
  – suppose \( A_{TM} \) is decidable
  – show that we can use \( A_{TM} \) to decide \( \text{HALT} \)
  – conclude \( \text{HALT} \) is decidable, Contradiction.

Example reduction

• How could we use procedure that decides \( A_{TM} \) to decide \( \text{HALT} \)?
  – given input to \( \text{HALT} \): \(<M, w>\)
• 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 \( \text{HALT} \) using a procedure that decides \( A_{TM} \) ("reducing \( \text{HALT} \) to \( A_{TM} \)).
  – on input \(<M, w>\)
  – check if \(<M, w> \in A_{TM}\)
    • if yes, the \( M \) halts on \( w \); \text{ACCEPT}
    • if no, then \( M \) either rejects \( w \) or it loops on \( w \)
  – construct \( M' \) by swapping \( q_{\text{accept}}/q_{\text{reject}} \) in \( M \)
  – check if \(<M', w> \in A_{TM}\)
    • if yes, then \( M' \) accepts \( w \), so \( M \) rejects \( w \); \text{ACCEPT}
    • if no, then \( M \) neither accepts nor rejects \( w \); \text{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:
  \( E_{TM} = \{<M>: L(M) = \emptyset\} \)
• which problem should we reduce from?
  – HALT = \{<M,w>: M halts on input w\}
  – \( A_{TM} = \{<M,w>: M \text{ accepts input } w\} \)
• Some things we can do:
  – check if \( <M> \in E_{TM} \)
  – construct another TM \( M' \) and check if \( <M'> \in E_{TM} \)

Another example
• We are given input \( <M,w> \)
• We want to use a procedure that decides \( E_{TM} \) to decide if \( <M,w> \in A_{TM} \)

  **Idea:**
  – check if \( <M> \in E_{TM} \)
  – if not?
  – helpful if could make \( M \) reject everything except possibly \( w \).

Another example
• Construct TM \( M' \):
  – on input \( x \), if \( x \neq w \), then reject
  – else simulate \( M \) on \( x \), and accept if \( M \) does.
• on input \( <M,w> \)
  – 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

Example reduction
• We proved
  \( A_{TM} = \{<M,w>: M \text{ accepts input } w\} \)
  undecidable, by reduction from HALT = \{<M,w>: M halts on input w\}

• We proved
  \( E_{TM} = \{<M>: L(M) = \emptyset\} \)
  undecidable by reduction from \( A_{TM} \).