CSC 320 - Lecture 15

#church-turing-thesis #undecidable #undecidability #languages #co-turing-recognizable #turing-machines #computable-functions #reduction #undecidable-languages #mapping-reducibility #time-complexity

---

Church-Turing Thesis

Algorithms = Deciders

Algorithms. Finite number of unambiguous instructions (each instruction is of finite length). Produces the desired result in a finite number of steps.

Deciders. Turing machine that halts on any input (and accepts or rejects).

A problem can be solved following an algorithm (while ignoring resource limitations) if and only if it is computable by a Turing machine.

Another Undecidable Language

$$E_{TM} = \{\langle M \rangle \mid M \text{ is a } TM \text{ and } L(M) = \emptyset\}$$

Proof via reduction from A_{TM}. Prove that E_{TM} is undecidable.

Proof. Assume E_{TM} is decidable; let R be a decider for E_{TM}.

To achieve a contradiction, design decider S for A_{TM} that uses R as a subroutine.

Recall. $A_{TM} = \{\langle M \rangle \mid M \text{ is a } TM \text{ and accepts input string } w\}$. Therefore, S takes as input $\langle M, w \rangle$.

Question. How can R help to decide if $\langle M, v \rangle \in A_{TM}$?

Proof. Decider S for A_{TM} takes as input $\langle M, w \rangle$.

For each $\langle M, w \rangle$ we design an input $\langle M'_w \rangle$ for subroutine R that accepts only w if M accepts w and accepts the empty language otherwise.

Note. We are doing some pre-processing. This is why we have the extra step.

- $M'_w =$ "On input x

- If $x \neq w$ then rejects
- If $x = w$ then run M on input w and accept if M does"

We are now ready to design S.

- $S =$ "On input $\langle M, w \rangle$
  - Construct description of M'_w and run R on input $\langle M'_w \rangle$.
    - If R accepts then reject
    - Id R rejects then accept"

Question. Is S a decider for A_{TM}?

Note. Missing class notes from drawing board.

Show. If $\langle M, w \rangle \in A_{TM}$ then S accepts, else S rejects.

1. Let $\langle M, w \rangle \in A_{TM}$. Then R, on $\langle M'_w \rangle$, rejects since $L(M'_w) \neq \emptyset$. Therefore S accepts.
2. Let $\langle M, w \rangle \notin A_{TM}$. Then R, on $\langle M'_w \rangle$, accepts since $L(M'_w) = \emptyset$. Therefore S rejects.

We Know

- A_{TM} and $HALT_{TM}$ are both undecidable.
- A_{TM} and $HALT_{TM}$ are both Turing-Recognizable

What about a concrete language that is not Turing-Recognizable?

Are There Languages That Are Not Turing-Recognizable?

Definition. A language is **co-Turing-Recognizable** if it is the complement of a Turing-Recognizable language.

Theorem. A language A is decidable if and only if A is Turing-Recognizable and co-Turing-Recognizable.

Proof. We show..

1. A decidable $\Rightarrow$ A's complement $\bar{A}$ is decidable $\Rightarrow$ A and $\bar{A}$ are both Turing-Recognizable.
2. A and $\bar{A}$ are both Turing-Recognizable $\Rightarrow$ A decidable.

---

1. A decidable

Show. A's complement $\bar{A}$ is decidable.

Note. $w \in A \Leftrightarrow w \notin \bar{A}$. And M is a decider.

Let D_A be a decider for language A.

- We design TM $M =$ "On input w simulate D_A on input w.
  - If D_A accepts then reject
  - If D_A rejects then accept".

2. A and $\bar{A}$ are both Turing-Recognizable

Show. A is decidable.

Let M_A be recognizer for A and C be a recognizer for $\bar{A}$. The following Turing Machine M is a decider for A. And M is a decider since either M_A ir C accepts w.

- $M =$ "On input w run M_A and C simultaneously.
  - If M_A accepts then accept
  - If C accepts then reject".

A Language That Is Not Turing-Recognizable

Theorem. The complement of A_{TM}, $\bar{A_{TM}}$, is not Turing-Recognizable.

Proof. We know: A_{TM} is Turing-Recognizable. Assume $\bar{A_{TM}}$ is Turing-Recognizable. Then A_{TM} is decidable.

CONTRADICTION! A_{TM} is undecidable.

Note. Missing class notes from drawing board

Computable Functions

We know what decidable languages are. Often, we talk about computing functions. What is a computable function?

Definition. Let $f : \Sigma^* \longrightarrow \Sigma^*$. f is called **computable function** if some TM M exists, with for input w, M halts with just $f(w)$ on its tape.

Mapping Reducibility

Definition. Language A is mapping reducible to language B, or $A \leq_m B$, if there is a computable function $f : \Sigma^* \longrightarrow \Sigma^*$, where for every $w : w \in A \Leftrightarrow f(w) \in B$.

Function f is called reduction from A to B.

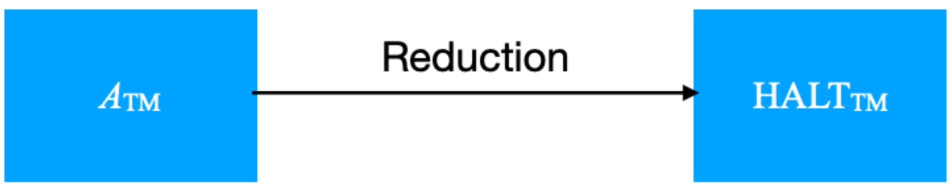
Theorem. If $A \leq_m B$ and B is decidable then A is decidable.

Proof. Let M be a decider for B and let f be a reduction from A to B.

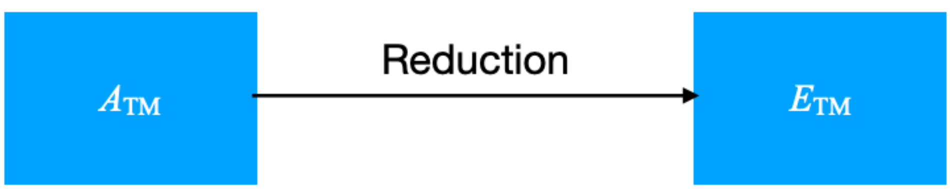
We build decider N for A.

- $N =$ "On input w:
  - Compute $f(w)$
  - Run M on input $f(w)$
  - Output whatever M outputs".

Revisiting The Reduction Proofs...



Can we prove mapping reducibility?



$HALT_{TM}$ Is Mapping Reducibility To A_{TM}

Show. There is a function $f : \Sigma^* \longrightarrow \Sigma^*$ with $\langle M, w \rangle \in A_{TM} \Leftrightarrow f(\langle M, w \rangle) \in HALT_{TM}$.

We design a TM F that computes f.

- $F =$ "On input $\langle M, w \rangle$ construct description of TM M'.
  - $M' =$ "On Input x run M on x
    - If M accepts then accept
    - If M rejects then enter a loop"

- Output $\langle M', w \rangle$".

Time Complexity

- From now on: decidable problems.
- Running time/time complexity of Turing Machine
- Asymptotic notation (Big Oh, etc.) applies
- $TIME(t(n))$

Note. If you program something that has an infinite loop then it is not an algorithm. It is something else.

Running Time / Time Complexity

Definition. Let M be a (deterministic) decider. The **running time** or **time complexity** of M is age function $f : \mathbb{N} \longrightarrow \mathbb{N}$ where $f(n)$ is the maximum number of steps that M uses on any input of length n.

If $f(n)$ is the running time of M then we say: M **runs in time** $f(n)$ **and** M **is an** $f(n)$**-time TM**.

Note. $f(n)$ doesn't have to be exact, it can be an upper bound.

Time Complexity Class

Let $t : \mathbb{N} \longrightarrow \mathbb{R}^+$ be a function. **Time Complexity Class** $TIME(t(n))$ is the collection of all languages decidable by an $O(t(n))$-time TM.

Up Next

- What do we know about time complexity when...
  - Comparing multi-tape TMs to single-tape TMs.
  - Comparing nondeterministic TMs to deterministic TMs.

Previous Lecture

Lecture14

Next Lecture

Lecture16