

CS 483 - Introduction to the Theory of Computation

Lecture 14

Undecidable Languages

We proved **ALL** $\neq$ **R** by showing $|\mathbf{ALL}| > |\mathbf{R}|$.

Undecidable Languages

We proved **ALL** $\neq$ **R** by showing $|\mathbf{ALL}| > |\mathbf{R}|$.

This time let us prove **ALL** $\neq$ **R** directly via diagonalization:

List of all deciders		ε	0	1	00	01	...
	D_0	1	1	0	1	0	
	D_1	0	0	1	0	1	
	D_2	0	1	1	0	0	
	D_3	0	0	0	1	1	
	D_4	0	0	1	1	0	
	$\vdots$						$\ddots$

Undecidable Languages

We proved **ALL** ≠ **R** by showing $|\mathbf{ALL}| > |\mathbf{R}|$.

This time let us prove **ALL** ≠ **R** directly via diagonalization:

List of all
deciders

	ε	0	1	00	01	...
D_0	1	1	0	1	0	
D_1	0	0	1	0	1	
D_2	0	1	1	0	0	
D_3	0	0	0	1	1	
D_4	0	0	1	1	0	
$\vdots$						$\ddots$
L	0	1	0	0	1	...

Undecidable Languages

We proved **ALL** ≠ **R** by showing $|\mathbf{ALL}| > |\mathbf{R}|$.

This time let us prove **ALL** ≠ **R** directly via diagonalization:

Let $D_0, D_1, D_2, \dots$ be all deciders and $w_0, w_1, w_2, \dots$ be all strings in Σ^* .

Define
$$L = \{w_i \mid i \in \mathbb{N}, D_i(w_i) = 0\}.$$

If L is decidable, i.e., $L = L(D_i)$ for some $i \in \mathbb{N}$, then:

- If $w_i \in L$, it means $D_i(w_i) = 0$, so $w_i \notin L(D_i) = L$;
- If $w_i \notin L$, it means $D_i(w_i) = 1$, so $w_i \in L(D_i) = L$.

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{1^i \mid D_i(1^i) = 0\}$$

	ε	1	11	111	1111	...
D_0	1	0	0	1	1	
D_1	0	1	1	0	0	
D_2	0	1	1	0	1	
D_3	0	0	1	0	1	
D_4	0	1	0	0	1	
$\vdots$						$\ddots$
L	0	0	0	1	1	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{1^i \mid D_i(1^i) = 0\} \cup \Sigma^*0\Sigma^*$$

	ε	1	11	111	1111	...
D_0	1	0	0	1	1	
D_1	0	1	1	0	0	
D_2	0	1	1	0	1	
D_3	0	0	1	0	1	
D_4	0	1	0	0	1	
$\vdots$						$\ddots$
L	0	0	0	1	1	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{ D \mid D \text{ is a decider, } D(D) = 0 \}$$

	$\langle D_0 \rangle$	$\langle D_1 \rangle$	$\langle D_2 \rangle$	$\langle D_3 \rangle$	$\langle D_4 \rangle$	...
D_0	0	1	1	1	0	
D_1	0	1	1	0	0	
D_2	0	1	1	1	1	
D_3	0	0	0	1	1	
D_4	1	0	0	1	0	
$\vdots$						$\ddots$
L	1	0	0	0	1	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{ D \mid D \text{ is a decider, } D(D) = 1 \}$$

$$L(D'_i) = \overline{L(D_i)}$$

	$\langle D_0 \rangle$	$\langle D_1 \rangle$	$\langle D_2 \rangle$	$\langle D_3 \rangle$	$\langle D_4 \rangle$	...
D'_0	1	0	0	0	1	
D'_1	1	0	0	1	1	
D'_2	1	0	0	0	0	
D'_3	1	1	1	0	0	
D'_4	0	1	1	0	1	
$\vdots$						$\ddots$
L	0	1	1	1	0	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{ M \mid M \text{ is a Turing machine, } M(M) = 0 \}$$

List of all
Turing machines

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...
M_0	0	1	1	1	0	
M_1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	
M_2	0	$\perp$	1	$\perp$	1	
M_3	0	0	$\perp$	$\perp$	$\perp$	
M_4	$\perp$	0	0	1	0	
$\vdots$						$\ddots$
L	1	0	0	0	1	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{ M \mid M \text{ is a Turing machine, } M(M) \neq 1 \}$$

List of all
Turing machines

	$\langle M_0 \rangle$	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...
M_0	0	1	1	1	0	
M_1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	
M_2	0	$\perp$	1	$\perp$	1	
M_3	0	0	$\perp$	$\perp$	$\perp$	
M_4	$\perp$	0	0	1	0	
$\vdots$						$\ddots$
L	1	1	0	1	1	...

Undecidable Languages

There are many other ways of diagonalization:

$$L = \{ M \mid M \text{ is a Turing machine, } M(M) \neq 1 \}$$

Proof by word: Assume L is decided by M . Then:

- If $M \in L$, it means $M(M) \neq 1$, so $M \notin L(M) = L$.
- If $M \notin L$, it means $M(M) = 1$, so $M \in L(M) = L$.

Undecidable Languages

Undecidability can also be proved using closure properties:

$$L' = \{ M \mid M \text{ is a Turing machine, } M(M) = 1 \}$$

Undecidable Languages

Undecidability can also be proved using closure properties:

$$L' = \{ M \mid M \text{ is a Turing machine, } M(M) = 1 \}$$

$$L = \{ M \mid M \text{ is a Turing machine, } M(M) \neq 1 \}$$

$$= \{ \text{Turing machine } M \} \cap \overline{L'}$$

If L' is decidable then L is also decidable, a contradiction.

Undecidable Languages

Undecidability can also be proved using **reductions**:

$$A_{\text{TM}} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) = 1 \}$$

Undecidable Languages

Undecidability can also be proved using **reductions**:

$$A_{\text{TM}} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) = 1 \}$$

Assume A_{TM} is decided by a Turing machine H .

Then we construct another Turing machine $H'(M)$:

- Run $H(M, M)$;
- Accept if it accepted, and reject if it rejected.

Then H' decides L' , a contradiction.

Decidability in Weaker Models

Similar languages for weaker models are indeed [decidable](#):

- $A_{\text{DFA}} = \{ (M, w) \mid \text{DFA } M \text{ accepts } w \}$

By direct simulation of a DFA with a Turing machine;

- $A_{\text{NFA}} = \{ (M, w) \mid \text{NFA } M \text{ accepts } w \}$

By converting the NFA into a DFA;

- $A_{\text{CFG}} = \{ (G, w) \mid \text{CFG } G \text{ generates } w \}$

By converting the CFG into its Chomsky normal form, and dynamic programming;

- $A_{\text{PDA}} = \{ (M, w) \mid \text{PDA } M \text{ accepts } w \}$

By converting the PDA into a CFG.

Halting Problem

$$\text{HALT} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) \neq \perp \}$$

Halting Problem

$$\text{HALT} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) \neq \perp \}$$

Proof that HALT is undecidable:

Assume HALT can be decided by a Turing machine H .

Construct another Turing machine $H'(M, w)$:

- Run $H(M, w)$;
 - Reject if it rejected.
 - If it accepted, run $U(M, w)$ and accept or reject accordingly.

Halting Problem

$$\text{HALT} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) \neq \perp \}$$

Proof that HALT is undecidable:

Assume HALT can be decided by a Turing machine H .

Construct another Turing machine $H'(M, w)$:

- Run $H(M, w)$;
 - Reject if it rejected.
 - If it accepted, run $U(M, w)$ and accept or reject accordingly.

Then H' decides A_{TM} , a contradiction.

Halting Problem

Notice that HALT is recognizable: run $U(M, w)$, accept when it halts.

Halting Problem

Notice that HALT is recognizable: run $U(M, w)$, accept when it halts.

Since L is decidable if and only if both L and $\bar{L}$ are recognizable, we have

$$\overline{\text{HALT}} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) = \perp \}$$

is unrecognizable. Similarly,

$$\overline{A_{\text{TM}}} = \{ (M, w) \mid M \text{ is a Turing machine, } M(w) \neq 1 \}$$

is also unrecognizable.

Witness Formulation for **RE**

Theorem. A language L is in **RE** if and only if it can be written as

$$L = \{ x \mid \exists y, (x, y) \in L' \}$$

for some $L' \in \mathbf{R}$.

Witness Formulation for **RE**

Theorem. A language L is in **RE** if and only if it can be written as

$$L = \{ x \mid \exists y, (x, y) \in L' \}$$

for some $L' \in \mathbf{R}$.

“only if” direction: assume M recognize L . Then L' is decidable for:

$$L' = \{ (x, t) \mid M \text{ accepts } x \text{ in } t \text{ steps} \}$$

Witness Formulation for **RE**

Theorem. A language L is in **RE** if and only if it can be written as

$$L = \{ x \mid \exists y, (x, y) \in L' \}$$

for some $L' \in \mathbf{R}$.

“only if” direction: assume M recognize L . Then L' is decidable for:

$$L' = \{ (x, t) \mid M \text{ accepts } x \text{ in } t \text{ steps} \}$$

“if” direction: assume M' decides L' . Then $M(x)$ recognizes L :

- For each $y \in \Sigma^*$:
 - Run $M'(x, y)$, and accept if it accepted.

Magic Button

What if you have a magic program that decides halting problem?



Magic Button

What if you have a magic program that decides halting problem?

You can solve e.g. Goldbach's conjecture:

Every even $n \geq 4$ is a sum of two primes.



Magic Button

What if you have a magic program that decides halting problem?

You can solve e.g. Goldbach's conjecture:

Every even $n \geq 4$ is a sum of two primes.

Construct Turing machine $M(\varepsilon)$:

- For every $n = 4, 6, 8, \dots$
 - Check if $n = p + q$ for some prime $p, q < n$
 - If not then halt.

And then run $\text{HALT}(M, \varepsilon)$.



Magic Button

In fact, you can essentially resolve every mathematical conjecture!

A mathematical proof is a finite list of statements, where each statement is either an axiom, or is derived from previous statements.

Construct a Turing machine $P(x)$ where x is a statement:

- For each $y \in \Sigma^*$:
 - Check if y encodes a valid proof for x ; if so then accept.

Run $\text{HALT}(P, x)$; it accepts if and only if x is provable.

Gödel's Incompleteness Theorem

How about we run $P(x)$ and $P(\neg x)$ in parallel? Are we guaranteed to know whether x is true or false?

Gödel's Incompleteness Theorem

How about we run $P(x)$ and $P(\neg x)$ in parallel? Are we guaranteed to know whether x is true or false?

Theorem (Gödel). In every **consistent** axiom system capable of modeling arithmetic, there are statements that can neither be proved or disproved.

Gödel's Incompleteness Theorem

How about we run $P(x)$ and $P(\neg x)$ in parallel? Are we guaranteed to know whether x is true or false?

Theorem (Gödel). In every **consistent** axiom system capable of modeling arithmetic, there are statements that can neither be proved or disproved.

Proof sketch. The statement " $M(w)$ halts" can be expressed with arithmetic (by encoding each configuration uqv as a number).

If every such statement can be proved or disproved, then running $P("M(w) \text{ halts}")$ and $P("M(w) \text{ does not halt}")$ in parallel decides HALT.