Undecidability

Lecture in INF4130

Department of Informatics

October 18th, 2018

Background from Lecture 1

- Formal Languages
- Turing Machines
  - General purpose computational models
  - Infinite tape
  - Accepting, Rejecting and Looping
  - Turing machines can simulate other Turing machines (Exercise-set-1, Universal Turing machine)
- Church Turing Thesis

Terminology

Example (The language PRIMES)

$PRIMES = \{n \mid n \text{ is a prime number}\}$.

Example (The decision problem PRIMALITY)

INSTANCE: A natural number, n.
QUESTION: Is n a prime number?

Example (Checking membership for PRIMES)

M = "On input n:

(1) if $n < 2$, *reject*.
(2) for all $2 \leq i \leq \sqrt{n}$:
(3) if i divides n, *reject*.
(4) *accept*."

Definition (Turing-recognizable languages)

The set of stings A, that a Turing machine M accept, is called *the language of* M, or *the language recognized by* M. We write $A = L(M)$. A language is called *Turing-recognizable* if some Turing machine recognizes it.

Definition (Deciders)

A Turing machine that halts on all inputs, it is called a *decider*. If M is a decider it will either accept or reject its input. The language A is said to be *decided* by M.

Definition (Turing-recognizable languages)

The set of stings A, that a Turing machine M accept, is called *the language of* M, or *the language recognized by* M. We write $A = L(M)$. A language is called *Turing-recognizable* if some Turing machine recognizes it.

Definition (Deciders)

A Turing machine that halts on all inputs, it is called a *decider*. If M is a decider it will either accept or reject its input. The language A is said to be *decided* by M.

Definition (Decidable and undecidable languages)

A language is *(Turing) decidable* if there exists a Turing machine that decides it. If a language is not decidable, we call it *undecidable*.

Example (Life on Mars, [1])

Let A be the language $\{s\}$ where

$$s = \begin{cases} 0, & \text{if life never will be found on Mars.} \\ 1, & \text{if life will be found on Mars someday.} \end{cases}$$

Is A a decidable language or not?

Theorem

Any finite language is decidable.

Theorem

Any finite language is decidable.

Proof

If A is a finite language, a decider M_D for A can be constructed by hard-coding all yes-instances of A into M_D. On input w, M_D accepts if w is one of the yes-instances of A, and rejects otherwise. □

- Are any languages undecidable, and if so, how do we prove it?

- Are any languages undecidable, and if so, how do we prove it?
- We will first prove that a particular problem is undecidable.
- After finding such a problem, we can show undecidability of other problems using a technique called *reductions*.

- Are any languages undecidable, and if so, how do we prove it?
- We will first prove that a particular problem is undecidable.
- After finding such a problem, we can show undecidability of other problems using a technique called *reductions*.
- We will meet a very similar situation in later lectures, when we define *NP*-completeness.

The Halting problem

The Halting problem

Definition (The Halting Problem)

INSTANCE: A Turing machine M with an input w.
QUESTION: Does M halt when run on w?

Definition (HALT)

$HALT = \{\langle M, w\rangle | TM\ M \text{ halts on input } w\}$

The Halting problem

Definition (The Halting Problem)

INSTANCE: A Turing machine M with an input w.
QUESTION: Does M halt when run on w?

Definition (HALT)

$HALT = \{\langle M, w\rangle | TM\ M \text{ halts on input } w\}$

Theorem

$HALT$ is an undecidable language.

Intuition and proof overview

Why can we not just simulate M on w?

Intuition and proof overview

Why can we not just simulate M on w?
U = "On input $\langle M, w \rangle$:

(1) Simulate M on input w.

(2) If M accepts, *accept*.

(3) If M rejects, *accept*."

The Turing machine U actually recognizes $HALT$, but it does not decide it. We need to show that no Turing machine decides $HALT$.

Intuition and proof overview

Why can we not just simulate M on w?
$U =$ "On input $\langle M, w \rangle$:

(1) Simulate M on input w.

(2) If M accepts, *accept*.

(3) If M rejects, *accept*."

The Turing machine U actually recognizes $HALT$, but it does not decide it. We need to show that no Turing machine decides $HALT$.
We will assume that Turing machine H decides $HALT$ and from that derive a contradiction.

Proof of undecidability of HALT

Assume that there exists a TM H that decides $HALT$. H takes $\langle M, w \rangle$ as input and *accepts* if M halts on w. If M loops forever on input w, H rejects.
We now construct the following TM called D:

Proof of undecidability of HALT

Assume that there exists a TM H that decides $HALT$. H takes $\langle M, w \rangle$ as input and *accepts* if M halts on w. If M loops forever on input w, H rejects.
We now construct the following TM called D:
$D =$ "On input $\langle M_1 \rangle$:

(1) Simulate H on $\langle M_1, \langle M_1 \rangle \rangle$.

(2) If H accepts, loop forever.

(3) If H rejects, *accept*."

Proof of undecidability of HALT

Assume that there exists a TM H that decides $HALT$. H takes $\langle M, w \rangle$ as input and *accepts* if M halts on w. If M loops forever on input w, H rejects.
We now construct the following TM called D:
$D =$ "On input $\langle M_1 \rangle$:

(1) Simulate H on $\langle M_1, \langle M_1 \rangle \rangle$.

(2) If H accepts, loop forever.

(3) If H rejects, *accept*."

What happens if we now run D on input $\langle D \rangle$? Well, (1) D will send $\langle D, \langle D \rangle \rangle$ to H which will check if D halts on input D. If (2) H accepts then D will enter a loop and never halt, but if (3) H rejects, then D will halt! Either way H will answer the question wrong. Thus we have a contradiction, so our assumption that there existed a decider for $HALT$ was false. □

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

*Actually the exact same proof as last slide, but from a different perspective

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

Again, we assume that H exist and create D as before. Remember that D checks if its input, M_1, halts on itself by using H as a subroutine. Then D behaves the opposite way from how M_1 behaves on itself. Now we create the following table where the entry i,j is the result of running H on $\langle M_i, \langle M_j \rangle \rangle$:

*Actually the exact same proof as last slide, but from a different perspective

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

Again, we assume that H exist and create D as before. Remember that D checks if its input, M_1, halts on itself by using H as a subroutine. Then D behaves the opposite way from how M_1 behaves on itself. Now we create the following table where the entry i,j is the result of running H on $\langle M_i, \langle M_j \rangle\rangle$:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...	$\langle D \rangle$	...
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*		*accept*	
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*	...	*reject*	...
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D							

*Actually the exact same proof as last slide, but from a different perspective

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

Again, we assume that H exist and create D as before. Remember that D checks if its input, M_1, halts on itself by using H as a subroutine. Then D behaves the opposite way from how M_1 behaves on itself. Now we create the following table where the entry i,j is the result of running H on $\langle M_i, \langle M_j \rangle\rangle$:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...	$\langle D \rangle$	...
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*		*accept*	
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*	...	*reject*	...
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
⋮			⋮		⋱		
D	*reject*	*reject*	*accept*	*accept*			

*Actually the exact same proof as last slide, but from a different perspective

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

Again, we assume that H exist and create D as before. Remember that D checks if its input, M_1, halts on itself by using H as a subroutine. Then D behaves the opposite way from how M_1 behaves on itself. Now we create the following table where the entry i,j is the result of running H on $\langle M_i, \langle M_j \rangle\rangle$:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$	$\langle D \rangle$	$\dots$
M_1	<u>*accept*</u>	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	<u>*accept*</u>	*accept*	*accept*		*accept*	
M_3	*reject*	*reject*	<u>*reject*</u>	*reject*	$\dots$	*reject*	$\dots$
M_4	*accept*	*accept*	*reject*	<u>*reject*</u>		*accept*	
$\vdots$			$\vdots$		$\ddots$		
D	*reject*	*reject*	*accept*	*accept*		<u>?</u>	
$\vdots$			$\vdots$				$\ddots$

*Actually the exact same proof as last slide, but from a different perspective

We will now look at an alternative proof*.

Diagonalization proof of undecidability of HALT

Again, we assume that H exist and create D as before. Remember that D checks if its input, M_1, halts on itself by using H as a subroutine. Then D behaves the opposite way from how M_1 behaves on itself. Now we create the following table where the entry i,j is the result of running H on $\langle M_i, \langle M_j \rangle \rangle$:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	...	$\langle D \rangle$	...
M_1	*accept*	*reject*	*accept*	*reject*		*accept*	
M_2	*accept*	*accept*	*accept*	*accept*		*accept*	
M_3	*reject*	*reject*	*reject*	*reject*	...	*reject*	...
M_4	*accept*	*accept*	*reject*	*reject*		*accept*	
⋮			⋮		⋱		
D	*reject*	*reject*	*accept*	*accept*		?	
⋮			⋮				⋱

Since D will accept the opposite of the diagonal, we have our contradiction. □

*Actually the exact same proof as last slide, but from a different perspective

Reductions

Reductions

- translating one problem into another
- the typical way of showing undecidability, is via reductions
- more on reductions when we come to P and NP

Reductions

- translating one problem into another
- the typical way of showing undecidability, is via reductions
- more on reductions when we come to P and NP

Definition (Turing Reducibility)

Language A is *(Turing) reducible* to language B, written $A \leq_T B$, if A is decidable given a decider to B as a subroutine*.

*Such a decider for B is often called an *oracle*.

A typical reduction

Example (Dollar-language)

Let $L_\$ = \{\langle M\rangle |$ TM M eventually writes a \$ when started on a blank tape$\}$

A typical reduction

Example (Dollar-language)

Let $L_\$ = \{\langle M \rangle |$ TM M eventually writes a \$ when started on a blank tape$\}$

We will show how to reduce *HALT* to $L_\$$. Since *HALT* is undecidable, we will know that $L_\$$ is undecidable.

A typical reduction

Proof: $L_\$$ is undecidable

First we assume (for contradiction) that $L_\$$ is decidable, that is, $M_\$$ exists and decides $L_\$$. We now want to use $M_\$$ to create a decider for *HALT* (which we know cannot exist) to get our contradiction.

$H =$ "On input $\langle M, w\rangle$:

(1) Create TM M' from $\langle M, w\rangle$ such that:
$M' =$ "Ignore the input:

  (1) Simulate M on w. // Note: important that this step doesn't write \$
  (2) Write \$ on the tape.
  (3) *Accept*."

(2) Simulate $M_\$$ on $\langle M'\rangle$.

(3) If $M_\$$ accepts, *accept*. If $M_\$$ rejects, *reject*."

□

Comments to the proof that $L_\$$ is undecidable

- The reduction is very typical and actually straight forward
- The action "write \$" seems very arbitrary

Comments to the proof that $L_\$$ is undecidable

- The reduction is very typical and actually straight forward
- The action "write \$" seems very arbitrary

Theorem (Rice's theorem)

Let R be a language consisting of Turing machine descriptions, such that R contains some, but not all Turing machine descriptions. Furthermore, let the membership in R for any Turing machine M_1, depend solely on the language of M_1, that is: $L(M_1) = L(M_2) \implies (\langle M_1 \rangle \in R \leftrightarrow \langle M_2 \rangle \in R)$. *Then R is an undecidable language.*

Theorem (Rice's theorem)

Let R be a language consisting of Turing machine descriptions, such that R contains some, but not all Turing machine descriptions. Furthermore, let the membership in R for any Turing machine M_1, depend solely on the language of M_1, that is:
$L(M_1) = L(M_2) \implies (\langle M_1 \rangle \in R \leftrightarrow \langle M_2 \rangle \in R)$. *Then R is an undecidable language.*

Proof

Weekly exercise. □

Theorem (Rice's theorem)

Let R be a language consisting of Turing machine descriptions, such that R contains some, but not all Turing machine descriptions. Furthermore, let the membership in R for any Turing machine M_1, depend solely on the language of M_1, that is:
$L(M_1) = L(M_2) \implies (\langle M_1 \rangle \in R \leftrightarrow \langle M_2 \rangle \in R)$. *Then R is an undecidable language.*

Proof

Weekly exercise. □

Note that the "Dollar-language" is not captured by Rice's theorem. Writing a \$ on the tape is not a property concerning the language of the Turing machine.

Example (ACCEPT)

Let $ACCEPT = \{\langle M, w\rangle \mid \text{TM } M \text{ accepts } w\}$. Show that $ACCEPT$ is an undecidable language by giving a reduction from $HALT$.

Example (ACCEPT)

Let $ACCEPT = \{\langle M, w\rangle |$ TM M accepts $w\}$. Show that $ACCEPT$ is an undecidable language by giving a reduction from $HALT$.

Proof

We want to show $HALT \leq_T ACCEPT$. Idea: construct $\langle M', w'\rangle$ from $\langle M, w\rangle$ such that M' accepts w' iff M halts on w.

$M' =$ "Ignore the input:

(1) Simulate M on w.

(2) *Accept*."

Now we may send M' together with some input to M_{ACCEPT} (the assumed decider for $ACCEPT$). If M_{ACCEPT} says that M' accepted its input, then we know that the simulation of M on w must have halted. If M_{ACCEPT} rejects, then we know that M was looping on w. □

Example (EMPTY)

Let $EMPTY = \{\langle M \rangle | L(M) = \emptyset\}$. Show that $EMPTY$ is an undecidable language by giving a reduction from $HALT$.

Proof

We want to show $HALT \leq_T EMPTY$. Idea: construct $\langle M' \rangle$ from $\langle M, w \rangle$ such that $L(M') \neq \emptyset$ iff M halts on w.

$M' =$ "On input x:

(1) if $(x \neq w)$, *reject*.

(2) Simulate M on w.

(3) *Accept* ."

Now we may send M' to M_{EMPTY} (the assumed decider for $EMPTY$). M' was constructed to reject all inputs except w, and to only accept w if M halts on w. If $L(M') \neq \emptyset$ then M' must have accepted w, so M must have halted on w. So if M_{EMPTY} "says yes" to M', we must "say no" to $\langle M, w \rangle$, and vice versa. □

Mapping reductions

Definition (Computable functions)

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Mapping reductions

Definition (Computable functions)

A function $f : \Sigma^* \to \Sigma^*$ is a *computable function* if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Definition (Mapping reductions)

Language A is *mapping reducible* to language B, written $A \leq_m B$, if there exists a computable function $f : \Sigma^* \to \Sigma^*$, where for every w:
$w \in A \leftrightarrow f(w) \in B$. The function f is called a reduction from A to B.

Theorem

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

Theorem

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

Theorem

If $A \leq_m B$, and A is undecidable, then B is undecidable.

References

Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.