Halting, Undecidability, and Maybe Some Complexity

Lecture 9

Tuesday, September 22, 2020

LaTeXed: September 1, 2020 21:23

Quote

"Young man, in mathematics you don't understand things. You just get used to them."
– John von Neumann.

9.1
Cantor's diagonalization argument

You can not count the real numbers

$I = (0, 1)$.
$\mathbb{N} = \{1, 2, 3, \ldots\}$ the integer numbers

Claim (Cantor)

$|\mathbb{N}| \neq |I|$

Claim (Warm-up)

$|\mathbb{N}| \leq |I|$

Proof.

$|\mathbb{N}| \leq |I|$ exists a one-to-one mapping from $\mathbb{N}$ to I. One such mapping is $f(i) = 1/i$, which readily implies the claim. □

You can not count the real numbers

$I = (0, 1)$.
$\mathbb{N} = \{1, 2, 3, \ldots\}$ the integer numbers

Claim (Cantor)

$|\mathbb{N}| \neq |I|$

Claim (Warm-up)

$|\mathbb{N}| \leq |I|$

Proof.

$|\mathbb{N}| \leq |I|$ exists a one-to-one mapping from $\mathbb{N}$ to I. One such mapping is $f(i) = 1/i$, which readily implies the claim. □

You can not count the real numbers II

$I = (0, 1)$, $\mathbb{N} = \{1, 2, 3, \ldots\}$.

Claim (Cantor)

$|\mathbb{N}| \neq |I|$, where $I = (0, 1)$.

Proof.

Write every number in $(0, 1)$ in its decimal expansion. E.g., $1/3 = 0.33333333333333333333\ldots$.

Assume that $|\mathbb{N}| = |I|$. Then there exists a one-to-one mapping $f : \mathbb{N} \rightarrow I$. Let β_i be the ith digit of $f(i) \in (0, 1)$.

$d_i =$ any number in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$

$D = 0.d_1 d_2 d_3 \ldots \in (0, 1)$.

D is a well defined unique number in $(0, 1)$,

But there is no j such that $f(j) = D$. A contradiction. □

You can not count the real numbers II

$I = (0,1)$, $\mathbb{N} = \{1,2,3,\ldots\}$.

Claim (Cantor)

$|\mathbb{N}| \neq |I|$, where $I = (0,1)$.

Proof.

Write every number in $(0,1)$ in its decimal expansion. E.g.,
$1/3 = 0.33333333333333333333\ldots$.
Assume that $|\mathbb{N}| = |I|$. Then there exists a one-to-one mapping $f : \mathbb{N} \to I$. Let β_i be the ith digit of $f(i) \in (0,1)$.
$d_i =$ any number in $\{0,1,2,3,4,5,6,7,8,9\} \setminus \{d_{i-1}, \beta_i\}$
$D = 0.d_1 d_2 d_3 \ldots \in (0,1)$.
D is a well defined unique number in $(0,1)$,
But there is no j such that $f(j) = D$. A contradiction. □

The matrix...

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	...
1	1	1	0	0	...
2	0	**1**	0	1	...
3	1	0	**1**	1	...
4	0	1	0	**0**	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

The matrix...

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$\dots$
1	$\beta_1 = 1$	1	0	0	$\dots$
2	0	$\beta_2 = \mathbf{1}$	0	1	$\dots$
3	1	0	$\beta_3 = \mathbf{1}$	1	$\dots$
4	0	1	0	$\beta_4 = \mathbf{0}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$d_i =$ any number in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$

The matrix...

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$\dots$
1	**1**	1	0	0	$\dots$
2	0	**1**	0	1	$\dots$
3	1	0	**1**	1	$\dots$
4	0	1	0	**0**	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$d_i =$ any number in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$

$\implies \forall i \; \beta_i \neq d_i$.

The matrix...

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$\dots$
1	1	1	0	0	$\dots$
2	0	**1**	0	1	$\dots$
3	1	0	**1**	1	$\dots$
4	0	1	0	**0**	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$d_i =$ any number in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$

$\implies \forall i \ \beta_i \neq d_i$.

$D = 0.23232323\dots$.

D can not be the i column, because $\beta_i \neq d_i$.

The matrix...

	$f(1)$	$f(2)$	$f(3)$	$f(4)$	$\dots$
1	1	1	0	0	$\dots$
2	0	**1**	0	1	$\dots$
3	1	0	**1**	1	$\dots$
4	0	1	0	**0**	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$d_i =$ any number in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \setminus \{d_{i-1}, \beta_i\}$
$\implies \forall i \ \beta_i \neq d_i$.
$D = 0.23232323\ldots$.
D can not be the i column, because $\beta_i \neq d_i$.
But D can not be in the matrix...

The liar paradox

When one day an expedition was sent to the spatial coordinates that Voojagig had claimed for this planet they discovered only a small asteroid inhabited by a solitary old man who claimed repeatedly that nothing was true, though he was later discovered to be lying.
– The Hitchhiker Guide to the Galaxy

1. The liar's paradox: This sentence is false.
2. Related to Russell's paradox.
3. Omnipotence paradox: Can [an omnipotent being] create a stone so heavy that it cannot lift it?

The liar paradox

When one day an expedition was sent to the spatial coordinates that Voojagig had claimed for this planet they discovered only a small asteroid inhabited by a solitary old man who claimed repeatedly that nothing was true, though he was later discovered to be lying.
– The Hitchhiker Guide to the Galaxy

1. The liar's paradox: This sentence is false.
2. Related to Russell's paradox.
3. Omnipotence paradox: Can [an omnipotent being] create a stone so heavy that it cannot lift it?

The liar paradox

When one day an expedition was sent to the spatial coordinates that Voojagig had claimed for this planet they discovered only a small asteroid inhabited by a solitary old man who claimed repeatedly that nothing was true, though he was later discovered to be lying.
– The Hitchhiker Guide to the Galaxy

1. The liar's paradox: This sentence is false.
2. Related to Russell's paradox.
3. Omnipotence paradox: Can [an omnipotent being] create a stone so heavy that it cannot lift it?

The liar paradox

When one day an expedition was sent to the spatial coordinates that Voojagig had claimed for this planet they discovered only a small asteroid inhabited by a solitary old man who claimed repeatedly that nothing was true, though he was later discovered to be lying.
– The Hitchhiker Guide to the Galaxy

1. The liar's paradox: This sentence is false.
2. Related to Russell's paradox.
3. Omnipotence paradox: Can [an omnipotent being] create a stone so heavy that it cannot lift it?

THE END

...

(for now)

9.2
Introduction to the halting theorem

The halting problem

Halting problem: Given a program Q, if we run it would it stop?

Q: Can one build a program P, that always stops, and solves the halting problem.

Theorem ("Halting theorem")

There is no program that always stops and solves the halting problem.

The halting problem

Halting problem: Given a program Q, if we run it would it stop?
Q: Can one build a program P, that always stops, and solves the halting problem.

Theorem ("Halting theorem")

There is no program that always stops and solves the halting problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a weird number *if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$.
No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a weird number *if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$. No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a weird number if

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$. No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

Intuition, why solving the Halting problem is really hard

Definition

An integer number n is a weird number *if*

- *the sum of the proper divisors (including 1 but not itself) of n the number is $> n$,*
- *no subset of those divisors sums to the number itself.*

70 is weird. Its divisors are $1, 2, 5, 7, 10, 14, 35$. $1 + 2 + 5 + 7 + 10 + 14 + 35 = 74$.
No subset of them adds up to 70.

Open question: Are there are any odd weird numbers?

Write a program P that tries all odd numbers in order, and check if they are weird. The programs stops if it found such number.

If can solve halting problem $\implies$ can resolve this open problem.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

If you can halt, you can prove or disprove anything...

1. Consider any math claim C.
2. Prover algorithm P_C:
   (A) Generate sequence of all possible proofs (sequence of strings) into a pipe/queue.
   (B) $\langle p \rangle \leftarrow$ pop top of queue.
   (C) Feed $\langle p \rangle$ and $\langle C \rangle$, into a proof verifier ("easy").
   (D) If $\langle p \rangle$ valid proof of $\langle C \rangle$, then stop and accept.
   (E) Go to (B).
3. P_C halts $\iff$ C is true and has a proof.
4. If halting is decidable, then can decide if any claim in math is true.

THE END

...

(for now)

9.3
The halting theorem

Encodings

M: Turing machine
$\langle M \rangle$: a binary string uniquely describing M (i.e., it is a number.
w: An input string.
$\langle M, w \rangle$: A unique binary string encoding both M and input w.

$$A_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Encodings

M: Turing machine
$\langle M \rangle$: a binary string uniquely describing M (i.e., it is a number.
w: An input string.
$\langle M, w \rangle$: A unique binary string encoding both M and input w.

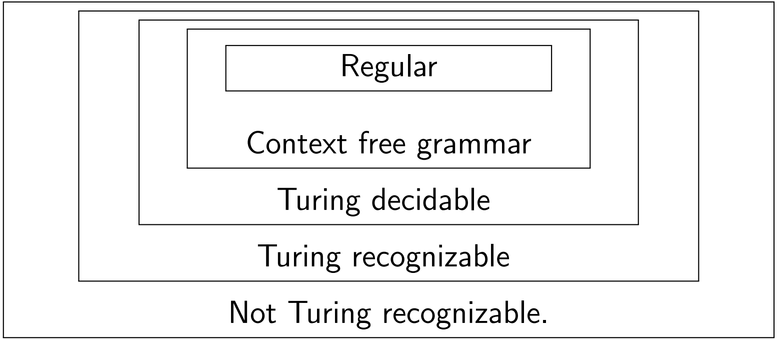
$$\mathrm{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Encodings

M: Turing machine
$\langle M \rangle$: a binary string uniquely describing M (i.e., it is a number.
w: An input string.
$\langle M, w \rangle$: A unique binary string encoding both M and input w.

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Complexity classes



$\mathbf{A}_{\mathrm{TM}}$ is TM recognizable...

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Lemma

$\mathbf{A}_{\mathrm{TM}}$ is Turing recognizable.

Proof.

Input: $\langle M, w \rangle$.
Using UTM simulate running M on w. If M accepts w then accept, if M rejects then reject. Otherwise, the simulation runs forever. □

$\mathbf{A}_{\mathrm{TM}}$ is TM recognizable...

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Lemma

$\mathbf{A}_{\mathrm{TM}}$ *is Turing recognizable.*

Proof.

Input: $\langle M, w \rangle$.

Using UTM simulate running M on w. If M accepts w then accept, if M rejects then reject. Otherwise, the simulation runs forever. □

$\mathbf{A}_{\mathrm{TM}}$ is not TM decidable!

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)

$\mathbf{A}_{\mathrm{TM}}$ *is not Turing decidable.*

Proof: Assume $\mathbf{A}_{\mathrm{TM}}$ is TM decidable...

Halt: TM deciding $\mathbf{A}_{\mathrm{TM}}$. **Halt** always halts, and works as follows:

$$\mathbf{Halt}\Big(\langle M, w \rangle\Big) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$

$\mathbf{A}_{\mathrm{TM}}$ is not TM decidable!

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)

$\mathbf{A}_{\mathrm{TM}}$ is not Turing decidable.

Proof: Assume $\mathbf{A}_{\mathrm{TM}}$ is TM decidable...

Halt: TM deciding $\mathbf{A}_{\mathrm{TM}}$. **Halt** always halts, and works as follows:

$$\textbf{Halt}\Big(\langle M, w \rangle\Big) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$

$\mathbf{A}_{\mathrm{TM}}$ is not TM decidable!

$$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

Theorem (The halting theorem.)

$\mathbf{A}_{\mathrm{TM}}$ *is not Turing decidable.*

Proof: Assume $\mathbf{A}_{\mathrm{TM}}$ is TM decidable...

Halt: TM deciding $\mathbf{A}_{\mathrm{TM}}$. **Halt** always halts, and works as follows:

$$\textbf{Halt}\Big(\langle M, w \rangle\Big) = \begin{cases} \text{accept} & M \text{ accepts } w \\ \text{reject} & M \text{ does not accept } w. \end{cases}$$

Halting theorem proof continued 1

We build the following new function:

```
Flipper(⟨M⟩)
    res ← Halt(⟨M, M⟩)
    if res is accept then
            reject
    else
            accept
```

Flipper always stops:

$$\textbf{Flipper}\left(\langle M\rangle\right) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle . \end{cases}$$

Halting theorem proof continued 1

We build the following new function:

```
Flipper(⟨M⟩)
   res ← Halt(⟨M, M⟩)
   if res is accept then
            reject
   else
            accept
```

Flipper always stops:

$$\textbf{Flipper}\Big(\langle M\rangle\Big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle\,. \end{cases}$$

Halting theorem proof continued 2

$$\textbf{Flipper}\Big(\langle M\rangle\Big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle \,. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding $\langle$**Flipper**$\rangle$. Run **Flipper** on itself:

$$\textbf{Flipper}\Big(\langle\textbf{Flipper}\rangle\Big) = \begin{cases} \text{reject} & \textbf{Flipper} \text{ accepts } \langle\textbf{Flipper}\rangle \\ \text{accept} & \textbf{Flipper} \text{ does not accept } \langle\textbf{Flipper}\rangle \,. \end{cases}$$

This is absurd. Ridiculous even!

Assumption that **Halt** exists is false. $\implies$ A_{TM} is not TM decidable. □

Halting theorem proof continued 2

$$\textbf{Flipper}\Big(\langle M\rangle\Big) = \begin{cases} \text{reject} & M \text{ accepts } \langle M\rangle \\ \text{accept} & M \text{ does not accept } \langle M\rangle \,. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding $\langle$**Flipper**$\rangle$. Run **Flipper** on itself:

$$\textbf{Flipper}\Big(\langle \textbf{Flipper}\rangle\Big) = \begin{cases} \text{reject} & \textbf{Flipper} \text{ accepts } \langle \textbf{Flipper}\rangle \\ \text{accept} & \textbf{Flipper} \text{ does not accept } \langle \textbf{Flipper}\rangle \,. \end{cases}$$

This is absurd. Ridiculous even!

Assumption that **Halt** exists is false. $\implies$ A_{TM} is not TM decidable. □

Halting theorem proof continued 2

$$\textbf{Flipper}\left(\langle M \rangle \right) = \begin{cases} \text{reject} & M \text{ accepts } \langle M \rangle \\ \text{accept} & M \text{ does not accept } \langle M \rangle \,. \end{cases}$$

Flipper is a TM (duh!), and as such it has an encoding ⟨**Flipper**⟩. Run **Flipper** on itself:

$$\textbf{Flipper}\left(\langle \textbf{Flipper} \rangle \right) = \begin{cases} \text{reject} & \textbf{Flipper} \text{ accepts } \langle \textbf{Flipper} \rangle \\ \text{accept} & \textbf{Flipper} \text{ does not accept } \langle \textbf{Flipper} \rangle \,. \end{cases}$$

This is absurd. Ridiculous even!
Assumption that **Halt** exists is false. $\implies$ $\mathbf{A}_{\mathrm{TM}}$ is not TM decidable. □

But where is the diagonalization argument????

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\dots$
M_1	**rej**	acc	rej	rej	$\dots$
M_2	rej	**acc**	rej	acc	$\dots$
M_3	acc	acc	**acc**	rej	$\dots$
M_4	rej	acc	acc	**rej**	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

THE END

...

(for now)

9.4
Unrecognizable

TM recognizable

Definition

Language L *is* TM decidable *if there exists* M *that always stops, such that* $L(M) = L$.

Definition

Language L *is* TM recognizable *if there exists* M *that stops on some inputs, such that* $L(M) = L$.

Theorem (Halting)

$A_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ accepts } w \right\}$. *is* TM *recognizable, but not decidable.*

TM recognizable

Definition

Language L *is* TM decidable *if there exists* M *that always stops, such that* $L(M) = L$.

Definition

Language L *is* TM recognizable *if there exists* M *that stops on some inputs, such that* $L(M) = L$.

Theorem (Halting)

$A_{TM} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ accepts } w \right\}$. *is* TM *recognizable, but not decidable.*

TM recognizable

Definition

Language L *is* TM decidable *if there exists* M *that always stops, such that* $L(M) = L$.

Definition

Language L *is* TM recognizable *if there exists* M *that stops on some inputs, such that* $L(M) = L$.

Theorem (Halting)

$\mathbf{A}_{\mathrm{TM}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ accepts } w \right\}$. *is* TM *recognizable, but not decidable.*

TM recognizable

Lemma

If L and $\overline{L} = \Sigma^* \setminus L$ are both TM recognizable, then L and $\overline{L}$ are decidable.

Proof.

M: TM recognizing L.

M_c: TM recognizing $\overline{L}$.

Given input x, using UTM simulating running M and M_c on x in parallel. One of them must stop and accept. Return result.

$\implies$ L is decidable. □

TM recognizable

Lemma

If L and $\overline{L} = \Sigma^* \setminus L$ are both TM recognizable, then L and $\overline{L}$ are decidable.

Proof.

M: TM recognizing L.
M_c: TM recognizing $\overline{L}$.
Given input x, using UTM simulating running M and M_c on x in parallel. One of them must stop and accept. Return result.
$\implies$ L is decidable. □

Complement language for $\mathbf{A}_{\mathrm{TM}}$

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \Sigma^* \setminus \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ accepts } w \right\}.$$

But don't really care about invalid inputs. So, really:

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a TM and } M \text{ does } \textbf{not} \text{ accept } w \right\}.$$

Complement language for $\mathbf{A}_{\mathrm{TM}}$

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \Sigma^* \setminus \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ accepts } w \right\}.$$

But don't really care about invalid inputs. So, really:

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ does } \textbf{not} \text{ accept } w \right\}.$$

Complement language for $\mathbf{A}_{\mathrm{TM}}$ is not TM-recognizable

Theorem

The language

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ does not accept } w \right\}.$$

is not TM *recognizable.*

Proof.

$\mathbf{A}_{\mathrm{TM}}$ is TM-recognizable.
If $\overline{\mathbf{A}_{\mathrm{TM}}}$ is TM-recognizable
$\implies$ (by Lemma)
$\mathbf{A}_{\mathrm{TM}}$ is decidable. A contradiction. □

Complement language for $\mathbf{A}_{\mathrm{TM}}$ is not TM-recognizable

Theorem

The language

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \left\{ \langle M, w \rangle \;\middle|\; M \text{ is a } \mathrm{TM} \text{ and } M \text{ does not accept } w \right\}.$$

is not TM *recognizable.*

Proof.

$\mathbf{A}_{\mathrm{TM}}$ is TM-recognizable.
If $\overline{\mathbf{A}_{\mathrm{TM}}}$ is TM-recognizable
$\implies$ (by Lemma)
$\mathbf{A}_{\mathrm{TM}}$ is decidable. A contradiction. □

Complement language for $\mathbf{A}_{\mathrm{TM}}$ is not TM-recognizable

Theorem

The language

$$\overline{\mathbf{A}_{\mathrm{TM}}} = \left\{ \langle M, w \rangle \;\middle|\; M \textit{ is a } \mathrm{TM} \textit{ and } M \textit{ does } \text{not} \textit{ accept } w \right\}.$$

is not TM *recognizable.*

Proof.

$\mathbf{A}_{\mathrm{TM}}$ is TM-recognizable.
If $\overline{\mathbf{A}_{\mathrm{TM}}}$ is TM-recognizable
$\implies$ (by Lemma)
$\mathbf{A}_{\mathrm{TM}}$ is decidable. A contradiction. □

THE END

...

(for now)

9.5
Turing complete

Equivalent to a program

Definition

A system is Turing complete *if one can simulate a Turing machine using it.*

1. Programming languages (yey!).
2. C++ templates system (boo).
3. John Conway's game of life.
4. Many games (Minesweeper).
5. Post's correspondence problem.

Equivalent to a program

Definition

A system is Turing complete *if one can simulate a Turing machine using it.*

1. Programming languages (yey!).
2. C++ templates system (boo).
3. John Conway's game of life.
4. Many games (Minesweeper).
5. Post's correspondence problem.

Post's correspondence problem

S: set of **domino tiles**.

$\begin{array}{|c|}\hline abb \\ \hline bc \\ \hline\end{array}$: domino piece a string at the top and a string at the bottom.

Example:

$$S = \left\{ \begin{array}{|c|}\hline b \\ \hline ca \\ \hline\end{array}, \begin{array}{|c|}\hline a \\ \hline ab \\ \hline\end{array}, \begin{array}{|c|}\hline ca \\ \hline a \\ \hline\end{array}, \begin{array}{|c|}\hline abc \\ \hline c \\ \hline\end{array} \right\}.$$

$$S = \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

match for S: ordered list of dominos from S, such that top strings make same string as bottom strings. Example:

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right].$$

(1) Can use same domino more than once.
(2) Do not have to use all pieces of S.

Matching dominos

$$S = \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}.$$

match for S: ordered list of dominos from S, such that top strings make same string as bottom strings. Example:

$$\left[\frac{a}{ab}\right]\left[\frac{b}{ca}\right]\left[\frac{ca}{a}\right]\left[\frac{a}{ab}\right]\left[\frac{abc}{c}\right].$$

(1) Can use same domino more than once.
(2) Do not have to use all pieces of S.

Matching dominos

$$S = \left\{ \begin{array}{|c|}\hline b \\ \hline ca \\ \hline \end{array}, \begin{array}{|c|}\hline a \\ \hline ab \\ \hline \end{array}, \begin{array}{|c|}\hline ca \\ \hline a \\ \hline \end{array}, \begin{array}{|c|}\hline abc \\ \hline c \\ \hline \end{array} \right\}.$$

match for S: ordered list of dominos from S, such that top strings make same string as bottom strings. Example:

$$\begin{array}{|c|}\hline a \\ \hline ab \\ \hline \end{array} \begin{array}{|c|}\hline b \\ \hline ca \\ \hline \end{array} \begin{array}{|c|}\hline ca \\ \hline a \\ \hline \end{array} \begin{array}{|c|}\hline a \\ \hline ab \\ \hline \end{array} \begin{array}{|c|}\hline abc \\ \hline c \\ \hline \end{array}.$$

(1) Can use same domino more than once.
(2) Do not have to use all pieces of S.

Post's Correspondence Problem

Post's Correspondence Problem (PCP) is deciding whether a set of dominos has a match or not.

modified Post's Correspondence Problem (MPCP): PCP + a special tile.

Matches for MPCP have to start with the special tile.

Theorem

The MPCP *problem is undecidable.*

THE END

...

(for now)