

Lecture 10

- Reduction Involving Controlled Execution

Lecture 10

- Reduction Involving Controlled Execution
- Rice Theorem

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages
- Turing Reductions

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages
- Turing Reductions
- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages
- Turing Reductions
- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- Undecidability of a **Tiling** Problem

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages
- Turing Reductions
- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- Undecidability of a **Tiling** Problem

Lecture 10

- Reduction Involving **Controlled Execution**
- Rice Theorem
- Description Length/Kolmogorov Complexity
- Undecidability of Kolmogorov Complexity
- \mathcal{RE} -Complete Languages
- Turing Reductions
- Relations Between Hilbert's 10th Problem and Gödel's Incompleteness Theorem
- Undecidability of a **Tiling** Problem

- Sipser, Sections 6.2, 6.3, 6.4

The Language L_{Σ^*}

$$L_{\Sigma^*} \triangleq \{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \Sigma^* \}.$$

Recall similarity (and difference) to All_{CFG} .

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

We saw a similar theorem regarding EQ_{TM} .

While proof for EQ_{TM} was quite simple, proof here will be a bit more involved, making use of a new technique: **Controlled execution**.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

- First we'll show $H_{\text{TM}} \leq_m L_{\Sigma^*}$.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

- First we'll show $H_{\text{TM}} \leq_m L_{\Sigma^*}$.
- This implies that $\overline{H_{\text{TM}}}$, the complement of the halting problem, is reducible to $\overline{L_{\Sigma^*}}$. Since $\overline{H_{\text{TM}}} \notin \mathcal{RE}$, this will imply that $\overline{L_{\Sigma^*}}$ is not enumerable, so $L_{\Sigma^*} \notin \text{co}\mathcal{RE}$.

$L_{\Sigma^*} \notin \text{coRE}$

Claim: $H_{\text{TM}} \leq_m L_{\Sigma^*}$.

The reduction $f : H_{\text{TM}} \longrightarrow L_{\Sigma^*}$ works as follows:

F : On input $\langle M, w \rangle$. Construct machine M_1 :

- On any input x , run M on w .
If it halts, M_1 accepts x .

$L_{\Sigma^*} \notin \text{coRE}$

Claim: $H_{\text{TM}} \leq_m L_{\Sigma^*}$.

The reduction $f : H_{\text{TM}} \longrightarrow L_{\Sigma^*}$ works as follows:

F : On input $\langle M, w \rangle$. Construct machine M_1 :

- On any input x , run M on w .
If it halts, M_1 accepts x .
- Output $\langle M_1 \rangle$.

$L_{\Sigma^*} \notin \text{coRE}$

Claim: $H_{\text{TM}} \leq_m L_{\Sigma^*}$.

The reduction $f : H_{\text{TM}} \longrightarrow L_{\Sigma^*}$ works as follows:

F : On input $\langle M, w \rangle$. Construct machine M_1 :

- On any input x , run M on w .
If it halts, M_1 accepts x .
- Output $\langle M_1 \rangle$.

The function f is clearly computable.

$L_{\Sigma^*} \notin \text{coRE}$

Claim: $H_{\text{T M}} \leq_m L_{\Sigma^*}$.

The reduction $f : H_{\text{T M}} \longrightarrow L_{\Sigma^*}$ works as follows:

F : On input $\langle M, w \rangle$. Construct machine M_1 :

- On any input x , run M on w .
If it halts, M_1 accepts x .
- Output $\langle M_1 \rangle$.

The function f is clearly computable.

If $\langle M, w \rangle \in H_{\text{T M}}$ then $L(M_1) = \Sigma^*$, while
if $\langle M, w \rangle \notin H_{\text{T M}}$ then $L(M_1) = \emptyset^*$.

$L_{\Sigma^*} \notin \text{coRE}$

Claim: $H_{\text{T M}} \leq_m L_{\Sigma^*}$.

The reduction $f : H_{\text{T M}} \longrightarrow L_{\Sigma^*}$ works as follows:

F : On input $\langle M, w \rangle$. Construct machine M_1 :

- On any input x , run M on w .
If it halts, M_1 accepts x .
- Output $\langle M_1 \rangle$.

The function f is clearly computable.

If $\langle M, w \rangle \in H_{\text{T M}}$ then $L(M_1) = \Sigma^*$, while
if $\langle M, w \rangle \notin H_{\text{T M}}$ then $L(M_1) = \emptyset^*$.

Thus we have $H_{\text{T M}} \leq_m L_{\Sigma^*}$.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

- Now we'll show $\overline{H_{\text{TM}}} \leq_m L_{\Sigma^*}$.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

- Now we'll show $\overline{H_{\text{TM}}} \leq_m L_{\Sigma^*}$.
- Since $\overline{H_{\text{TM}}} \notin \mathcal{RE}$, this will imply $L_{\Sigma^*} \notin \mathcal{RE}$.

$$L_{\Sigma^*} \notin \mathcal{RE} \cup \text{co}\mathcal{RE}$$

Theorem: Both L_{Σ^*} and its complement, $\overline{L_{\Sigma^*}}$, are not in \mathcal{RE} , namely are not enumerable.

- Now we'll show $\overline{H_{\text{TM}}} \leq_m L_{\Sigma^*}$.
- Since $\overline{H_{\text{TM}}} \notin \mathcal{RE}$, this will imply $L_{\Sigma^*} \notin \mathcal{RE}$.

The proof will be a bit more tricky.

$L_{\Sigma^*} \notin \mathcal{RE}$ (using controlled execution)

The reduction $g : \overline{H_{\text{TM}}} \longrightarrow L_{\Sigma^*}$ works as follows:

$L_{\Sigma^*} \notin \mathcal{RE}$ (using controlled execution)

The reduction $g : \overline{H_{\text{TM}}} \longrightarrow L_{\Sigma^*}$ works as follows:

G : On input $\langle M, w \rangle$. Construct machine M_2 :

- On any input y , run M on w .
If it halts in at most $|y|$ steps,
 M_2 rejects y , else M_2 accepts y .
- Output $\langle M_2 \rangle$.

$L_{\Sigma^*} \notin \mathcal{RE}$ (using controlled execution)

The reduction $g : \overline{H_{\text{TM}}} \longrightarrow L_{\Sigma^*}$ works as follows:

G : On input $\langle M, w \rangle$. Construct machine M_2 :

- On any input y , run M on w .
If it halts in at most $|y|$ steps,
 M_2 rejects y , else M_2 accepts y .
- Output $\langle M_2 \rangle$.

The function g is clearly computable.

$L_{\Sigma^*} \notin \mathcal{RE}$ (using controlled execution)

The reduction $g : \overline{H_{\text{TM}}} \longrightarrow L_{\Sigma^*}$ works as follows:

G : On input $\langle M, w \rangle$. Construct machine M_2 :

- On any input y , run M on w .
If it halts in at most $|y|$ steps,
 M_2 rejects y , else M_2 accepts y .
- Output $\langle M_2 \rangle$.

The function g is clearly computable.

If $\langle M, w \rangle \notin H_{\text{TM}}$ then $L(M_2) = \Sigma^*$, while

If $\langle M, w \rangle \in H_{\text{TM}}$ then $L(M_2) \neq \Sigma^*$

(if M halts on w in k steps, then $L(M_2) = \Sigma^{\leq k}$.)

$L_{\Sigma^*} \notin \mathcal{RE}$ (using controlled execution)

The reduction $g : \overline{H_{\text{TM}}} \longrightarrow L_{\Sigma^*}$ works as follows:

G : On input $\langle M, w \rangle$. Construct machine M_2 :

- On any input y , run M on w .
If it halts in at most $|y|$ steps,
 M_2 rejects y , else M_2 accepts y .
- Output $\langle M_2 \rangle$.

The function g is clearly computable.

If $\langle M, w \rangle \notin H_{\text{TM}}$ then $L(M_2) = \Sigma^*$, while

If $\langle M, w \rangle \in H_{\text{TM}}$ then $L(M_2) \neq \Sigma^*$

(if M halts on w in k steps, then $L(M_2) = \Sigma^{\leq k}$.)

Thus we have $\overline{H_{\text{TM}}} \leq_m L_{\Sigma^*}$.



Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM accept the **empty** language?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM accept the **empty** language?
- Does a TM accept a language that contains **all prime numbers**?

Bucket of Undecidable Problems

Similar reduction techniques prove **undecidability** of

- Does a TM accept a **decidable** language?
- Does a TM accept a **regular** language?
- Does a TM accept a **context-free** language?
- Does a TM accept a **finite** language?
- Does a TM accept the **empty** language?
- Does a TM accept a language that contains **all prime numbers**?
- Does a TM accept Σ^* ?

Remark: Beyond undecidability, techniques (and results) are different.

Rice's Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.

Rice's Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.
- Looks like **any** non-trivial property of TMs is undecidable.

Rice's Theorem

By now, some of you may have become cynical and embittered.

- Like, been there, done that, bought the T-shirt.
- Looks like **any** non-trivial property of TMs is undecidable.

That is correct.

Rice's Theorem – Restatement

Theorem: If \mathcal{C} is a proper, non-empty subset of the set of enumerable languages, then it is undecidable to test whether for a given encoding of a TM, $\langle M \rangle$, $L(M)$ is in \mathcal{C} .

(See problem 5.22 in Sipser's book)

Rice's Theorem

Theorem: Let \mathcal{C} be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in \mathcal{C} . Then $L_{\mathcal{C}}$ is undecidable.

Rice's Theorem

Theorem: Let \mathcal{C} be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in \mathcal{C} . Then $L_{\mathcal{C}}$ is undecidable.

Proof by reduction from A_{TM} .

Rice's Theorem

Theorem: Let \mathcal{C} be a proper non-empty subset of the set of enumerable languages. Denote by $L_{\mathcal{C}}$ the set of all TMs encodings, $\langle M \rangle$, such that $L(M)$ is in \mathcal{C} . Then $L_{\mathcal{C}}$ is undecidable.

Proof by reduction from A_{TM} .

Given M and w , we will construct M_0 such that:

- If M accepts w , then $\langle M_0 \rangle \in L_{\mathcal{C}}$.
- If M does not accept w , then $\langle M_0 \rangle \notin L_{\mathcal{C}}$.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- (Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.)

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- (Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.)
- Since \mathcal{C} is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall \mathcal{C} contains only **recursively enumerable** languages).

Rice's Theorem

- Without loss of generality, $\emptyset \notin \mathcal{C}$.
- (Otherwise, look at $\overline{\mathcal{C}}$, also proper and non-empty.)
- Since \mathcal{C} is not empty, there exists some language $L \in \mathcal{C}$. Let M_L be a TM accepting this language (recall \mathcal{C} contains only **recursively enumerable** languages).
- continued ...

Rice's Theorem

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w , then $L(M_0) = L \in \mathcal{C}$.

Rice's Theorem

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w , then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept w , then $L(M_0) = \emptyset \notin \mathcal{C}$.

Rice's Theorem

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w , then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept w , then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y :

1. Run M on w .
2. If M accepts w , run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.

Rice's Theorem

Given $\langle M, w \rangle$, construct M_0 such that:

- If M accepts w , then $L(M_0) = L \in \mathcal{C}$.
- If M does not accept w , then $L(M_0) = \emptyset \notin \mathcal{C}$.

M_0 on input y :

1. Run M on w .
2. If M accepts w , run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.

Claim: The transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a mapping reduction from A_{TM} to $L_{\mathcal{C}}$.

Rice's Theorem

Proof: M_0 on input y :

1. Run M on w .
2. If M accepts, run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.

Rice's Theorem

Proof: M_0 on input y :

1. Run M on w .
 2. If M accepts, run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.
- The machine M_0 is simply a concatenation of two known TMs – the **universal machine**, and M_L .

Rice's Theorem

Proof: M_0 on input y :

1. Run M on w .
 2. If M accepts, run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.
- The machine M_0 is simply a concatenation of two known TMs – the **universal machine**, and M_L .
 - Therefore the transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a computable function, defined for all strings in Σ^* .

Rice's Theorem

Proof: M_0 on input y :

1. Run M on w .
2. If M accepts, run M_L on y .
 - a. if M_L accepts, **accept**, and
 - b. if M_L rejects, **reject**.

- The machine M_0 is simply a concatenation of two known TMs – the **universal machine**, and M_L .
- Therefore the transformation $\langle M, w \rangle \rightarrow \langle M_0 \rangle$ is a computable function, defined for all strings in Σ^* .
- (But what do we actually do with strings not of the form $\langle M, w \rangle$?)

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, w \rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, w \rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.

Rice's Proof (Concluded)

- If $\langle M, w \rangle \in A_{\text{TM}}$ then M_0 gets to step 2, and runs M_L on y .
- In this case, $L(M_0) = L$, so $L(M_0) \in \mathcal{C}$.
- On the other hand, if $\langle M, w \rangle \notin A_{\text{TM}}$ then M_0 never gets to step 2.
- In this case, $L(M_0) = \emptyset$, so $L(M_0) \notin \mathcal{C}$.
- This establishes the fact that $\langle M, w \rangle \in A_{\text{TM}}$ iff $\langle M_0 \rangle \in L_{\mathcal{C}}$. So we have $A_{\text{TM}} \leq_m L_{\mathcal{C}}$, thus $L_{\mathcal{C}}$ is undecidable. ♣

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
 - “does $L(M)$ contain infinitely many primes”

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
 - “does $L(M)$ contain infinitely many primes”
 - “does $L(M)$ contain an arithmetic progression of length 15”

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
 - “does $L(M)$ contain infinitely many primes”
 - “does $L(M)$ contain an arithmetic progression of length 15”
 - “is $L(M)$ empty”

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
 - “does $L(M)$ contain infinitely many primes”
 - “does $L(M)$ contain an arithmetic progression of length 15”
 - “is $L(M)$ empty”
- Decidability of properties related to the encoding itself cannot be inferred from Rice. For example “does $\langle M \rangle$ has an even number of states” is decidable.

Rice's Theorem (Reflections)

- Rice's theorem can be used to show undecidability of properties like
 - “does $L(M)$ contain infinitely many primes”
 - “does $L(M)$ contain an arithmetic progression of length 15”
 - “is $L(M)$ empty”
- Decidability of properties related to the encoding itself cannot be inferred from Rice. For example “does $\langle M \rangle$ has an even number of states” is decidable.
- Properties like “does M reaches state q_6 on the empty input string” are undecidable, but this **does not** follow from Rice's theorem.

\mathcal{RE} -Completeness

Question: Is there a language L that is **hardest** in the class \mathcal{RE} of enumerable languages (languages accepted by some TM)?

\mathcal{RE} -Completeness

Question: Is there a language L that is **hardest** in the class \mathcal{RE} of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to **define** what you mean by “hardest language”.

\mathcal{RE} -Completeness

Question: Is there a language L that is **hardest** in the class \mathcal{RE} of enumerable languages (languages accepted by some TM)?

Answer: Well, you have to **define** what you mean by “hardest language”.

Definition: A language $L_0 \subseteq \Sigma^*$ is called \mathcal{RE} -complete if the following holds

- $L_0 \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

\mathcal{RE} -Completeness

Definition: A language $L_0 \subseteq \Sigma^*$ is called \mathcal{RE} -complete if the following holds

- $L \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

\mathcal{RE} -Completeness

Definition: A language $L_0 \subseteq \Sigma^*$ is called \mathcal{RE} -complete if the following holds

- $L \in \mathcal{RE}$ (membership).
- For every $L \in \mathcal{RE}$, $L \leq_m L_0$ (hardness).

The second item means that for every enumerable L there is a mapping reduction f_L from L to L_0 . The reduction f_L depends on L and will typically differ from one language to another.

\mathcal{RE} -Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

\mathcal{RE} -Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem: The language A_{TM} is \mathcal{RE} -Complete.

\mathcal{RE} -Completeness

Question: Having defined a reasonable notion, we should make sure it is not vacuous, namely verify there is at least one language satisfying it.

Theorem: The language A_{TM} is \mathcal{RE} -Complete.

Proof:

- The universal machine U accepts the language A_{TM} , so $A_{\text{TM}} \in \mathcal{RE}$.
- Suppose L is in \mathcal{RE} , and let M_L be a TM accepting it. Then $f_L(x) = \langle M_L, x \rangle$ is a mapping reduction from L to A_{TM} (why?). ♣

Some Points For Thought

- How would you define that a certain language, L_1 , is **strictly harder** than a different language, L_2 ?
- Is there a pair of such languages **within \mathcal{RE}** ?
- How would you define that a certain pair of language, L_1 and L_2 , are **computationally incompatible**?

Description Length and Information

Consider the two (equal length – 28 bits each) strings

01010101010101010101010101010101

0010110011101010100110001111

Which of these two strings has **more information**?

Description Length and Information

Consider the two (equal length – 28 bits each) strings

01010101010101010101010101010101

0010110011101010100110001111

Which of these two strings has **more information**?

This raises the difficult question of what information means, and how can it be measured.

Description Length and Information

Consider the two (equal length – 28 bits each) strings

01010101010101010101010101010101

0010110011101010100110001111

Which of these two strings has **more information**?

This raises the difficult question of what information means, and how can it be measured.

Following Andrei Nikolaevich Kolmogorov (April 25, 1903 - October 20, 1987), we will measure the information of a string by means of its **description length**.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with **shorter explanations** are typically **less complex** than phenomena with longer explanations.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with **shorter explanations** are typically **less complex** than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are **more informative** than those with shorter description.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with **shorter explanations** are typically **less complex** than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are **more informative** than those with shorter description.

Of course, we should still **define** what **description length** means.

Information and Description Length

The motivation for Kolmogorov complexity is that phenomena with **shorter explanations** are typically **less complex** than phenomena with longer explanations.

Consequently, we will say that strings with longer description length are **more informative** than those with shorter description.

Of course, we should still **define** what **description length** means.

An alternative route (not taken here) is to consider how much a string can be **compressed**.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as **computing functions**.

In particular, we can talk about the **Universal TM** for computing functions.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as **computing functions**.

In particular, we can talk about the **Universal TM** for computing functions.

Definition: Let M be a TM, and f_M be the function it computes.

The **Kolmogorov Complexity** of a string x with respect to M , $K_M(x)$, is defined as the **length** of the shortest string y satisfying $f_M(y) = x$.

Kolmogorov Complexity

In this part of the lecture, we view all TMs as **computing functions**.

In particular, we can talk about the **Universal TM** for computing functions.

Definition: Let M be a TM, and f_M be the function it computes.

The **Kolmogorov Complexity** of a string x with respect to M , $K_M(x)$, is defined as the **length** of the shortest string y satisfying $f_M(y) = x$.

If there is no such y , we define $K_M(x) = \infty$.

Kolmogorov Complexity

Hey, this definition is **no good**. It is totally arbitrary and depends on the particular choice of machine M . Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Kolmogorov Complexity

Hey, this definition is **no good**. It is totally arbitrary and depends on the particular choice of machine M . Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Well **giddy, mates**, and **no worries** . We will immediately show how this can be fixed.

Kolmogorov Complexity

Hey, this definition is **no good**. It is totally arbitrary and depends on the particular choice of machine M . Moreover, some strings may have $K_M(x) = \infty$, which is counter intuitive.

Well **giddy, mates**, and **no worries** . We will immediately show how this can be fixed.

Theorem: Let U be a **universal Turing machine**. For every Turing machine, M , there is a constant c_M (depending on M alone) such that for **every string** $x \in \Sigma^*$, $K_U(x) \leq K_M(x) + c_M$.

Kolmogorov Complexity

Theorem: Let U be a universal Turing machine. For every Turing machine, M , there is a constant c_M (depending on M alone) such that for every $x \in \Sigma^*$,
$$K_U(x) \leq K_M(x) + c_M.$$

Kolmogorov Complexity

Theorem: Let U be a **universal Turing machine**. For every Turing machine, M , there is a constant c_M (depending on M alone) such that for every $x \in \Sigma^*$,
 $K_U(x) \leq K_M(x) + c_M$.

Proof: Let y be a shortest string such that $f_M(y) = x$. Then for the **universal TM**, U ,
 $f_U(\langle M, y \rangle) = f_M(y) = x$.

Using prefix-free encodings for TMs, $\langle M, y \rangle$ is simply the concatenation of $\langle M \rangle$, followed by the string y . So we get

$$K_U(x) \leq |y| + |\langle M \rangle| = K_M(x) + |\langle M \rangle|.$$

So the theorem holds where $c_M = |\langle M \rangle|$. ♣.

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x ,

$$|K_{U_1}(x) - K_{U_2}(x)| < c .$$

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x ,

$$|K_{U_1}(x) - K_{U_2}(x)| < c .$$

So we can take any universal TM, U , define

$$K(x) = K_U(x) ,$$

and refer to this measure as “Kolmogorov complexity of the string x .”

Kolmogorov Complexity

Corollary: If both U_1 and U_2 are universal Turing machines, then there is a constant c such that for every string x ,

$$|K_{U_1}(x) - K_{U_2}(x)| < c .$$

So we can take any universal TM, U , define

$$K(x) = K_U(x) ,$$

and refer to this measure as “Kolmogorov complexity of the string x .”

We now show that for every string x , $K(x)$ equals at most x 's length plus a constant.

Kolmogorov Complexity

Theorem: There is a constant c such that for every string x , $K(x) \leq |x| + c$.

Pay attention to the **order of quantifiers** in the statement.

Kolmogorov Complexity

Theorem: There is a constant c such that for every string x , $K(x) \leq |x| + c$.

Pay attention to the **order of quantifiers** in the statement.

Proof: Let M_{ID} be a TM computing the identity function $f(x) = x$ (e.g. a TM that halts immediately). Obviously for any string x , $K_{M_{ID}}(x) = |x|$. By previous theorem, there is a constant c such that for any string x ,

$$K(x) = K_U(x) \leq K_{M_{ID}}(x) + c = |x| + c \quad \clubsuit$$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$
- $K(1^{2^n}) \leq \log(n) + c$

Kolmogorov Complexity

Are there strings whose Kolmogorov complexity is substantially smaller than their own length?

- $K(xx) \leq K(x) + c$
- $K(1^n) \leq \log(n) + c$
- $K(1^{2^n}) \leq \log(n) + c$

But these strings with very concise description are **rare**.

Kolmogorov Complexity

A simple **counting argument** gives

Theorem: For every integer $c \geq 1$, the number of strings in $\{0, 1\}^n$ for which $K(x) \leq n - c$ is at most $2^n / 2^{c-1}$.

Kolmogorov Complexity

A simple **counting argument** gives

Theorem: For every integer $c \geq 1$, the number of strings in $\{0, 1\}^n$ for which $K(x) \leq n - c$ is at most $2^n / 2^{c-1}$.

Proof: In $\{0, 1\}^*$ there is 1 string of length 0, 2 string of length 1, ..., 2^{n-c} string of length $n - c$. The total number of strings up to length $n - c$ is $2^{n+1-c} - 1 < 2^n / 2^{c-1}$. So the number of possible descriptions y of length $\leq n - c$ is no more than $2^n / 2^{c-1}$. This implies that the number of length n strings whose description length is c shorter than their own length is at most $2^n / 2^{c-1}$. ♣

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But **is it computable?**

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But **is it computable?**

Theorem: The function $K(\cdot)$ is **not computable**.

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But **is it computable?**

Theorem: The function $K(\cdot)$ is **not computable**.

Proof: By contradiction. For every n , let y_n be the lexicographically first string y satisfying $n < K(y)$. Then the sequence $\{y_n\}_{n=1}^{\infty}$ is well defined.

Kolmogorov Complexity Uncomputable

The function $K(\cdot)$ is total (defined for every string x) and unbounded. But **is it computable?**

Theorem: The function $K(\cdot)$ is **not computable**.

Proof: By contradiction. For every n , let y_n be the lexicographically first string y satisfying $n < K(y)$. Then the sequence $\{y_n\}_{n=1}^{\infty}$ is well defined.

Assume K is computable. We'll show this implies the existence of a constant c such that for every n ,
 $K(y_n) < \log(n) + c$.

Kolmogorov Complexity Uncomputable

Consider the following TM, M :

On input n (in binary), M generates, one by one, all binary strings x_0, x_1, x_2, \dots in lexicographic order.

For each x_i it produces, M computes $K(x_i)$.

Kolmogorov Complexity Uncomputable

Consider the following TM, M :

On input n (in binary), M generates, one by one, all binary strings x_0, x_1, x_2, \dots in lexicographic order.

For each x_i it produces, M computes $K(x_i)$.

If $K(x_i) > n$, the TM, M , outputs x_i and halts.

Otherwise, the TM, M , continues to examine the lexicographically next string, x_{i+1} .

Kolmogorov Complexity Uncomputable

Consider the following TM, M :

On input n (in binary), M generates, one by one, all binary strings x_0, x_1, x_2, \dots in lexicographic order.

For each x_i it produces, M computes $K(x_i)$.

If $K(x_i) > n$, the TM, M , outputs x_i and halts.

Otherwise, the TM, M , continues to examine the lexicographically next string, x_{i+1} .

Since the function K is unbounded, it is guaranteed that M will eventually reach a string x satisfying

$K(x) > n$.

Kolmogorov Complexity Uncomputable

Q.: What will the TM, M , output on input n (in binary)?

Kolmogorov Complexity Uncomputable

Q.: What will the TM, M , output on input n (in binary)?

Claim: By its definition, on input n (in binary), the TM, M , output y_n (the lexicographically first string whose Kolmogorov complexity exceeds n , $K(x) > n$).

Kolmogorov Complexity Uncomputable

Q.: What will the TM, M , output on input n (in binary)?

Claim: By its definition, on input n (in binary), the TM, M , output y_n (the lexicographically first string whose Kolmogorov complexity exceeds n , $K(x) > n$).

But length of n is $\log_2(n)$. So $K_M(y_n) \leq \log_2(n)$.

Kolmogorov Complexity Uncomputable

Q.: What will the TM, M , output on input n (in binary)?

Claim: By its definition, on input n (in binary), the TM, M , output y_n (the lexicographically first string whose Kolmogorov complexity exceeds n , $K(x) > n$).

But length of n is $\log_2(n)$. So $K_M(y_n) \leq \log_2(n)$.

We saw that there is a constant c_M such that for every y , $K(y) \leq K_M(y) + c_M$, so for every n , $K(y_n) \leq \log_2(n) + c_M$.

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that for **every** y , $K(y) \leq K_M(y) + c_M$,
so for **every** n , $K(y_n) \leq \log_2(n) + c_M$.

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that for every y , $K(y) \leq K_M(y) + c_M$,
so for every n , $K(y_n) \leq \log_2(n) + c_M$.

By definition, of y_n for every n , $n < K(y_n)$.
Combining the last two inequalities, we get, for every n ,

$$n < \log_2(n) + c_M .$$

Kolmogorov Complexity Uncomputable

We know that there is a constant c_M such that for **every** y , $K(y) \leq K_M(y) + c_M$,
so for **every** n , $K(y_n) \leq \log_2(n) + c_M$.

By definition, of y_n for every n , $n < K(y_n)$.
Combining the last two inequalities, we get, for every n ,

$$n < \log_2(n) + c_M .$$

But asymptotically n grows faster than $\log_2(n) + c_M$.
Contradiction (to computability of $K(\cdot)$). ♣

Reducibilities

Notion of **reducibility** was important for producing a solution to A if we got a solution to B . Inversly, if reducibility from A to B establishes that if A has no solution, neither does B .

- Central working horse was **mapping reducibility**,
 $A \leq_m B$.

Reducibilities

Notion of **reducibility** was important for producing a solution to A if we got a solution to B . Inversly, if reducibility from A to B establishes that if A has no solution, neither does B .

- Central working horse was **mapping reducibility**, $A \leq_m B$.
- Is **mapping** reducibility general enough notion to capture above intuition?

Reducibilities

- Is **mapping** reducibility general enough notion to capture above intuition?

Reducibilities

- Is **mapping** reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, \overline{L} .

Reducibilities

- Is **mapping** reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, \overline{L} .
- An answer to “**is** $x \in L$ ” is obtained from an answer to “**is** $x \in \overline{L}$ ” by simply reversing the original answer.

Reducibilities

- Is **mapping** reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, \overline{L} .
- An answer to “**is** $x \in L$ ” is obtained from an answer to “**is** $x \in \overline{L}$ ” by simply reversing the original answer.
- So, in particular, $\overline{A_{TM}}$ should be reducible to A_{TM} . However certainly $\overline{A_{TM}} \not\leq_m A_{TM}$ (**why?**).

Reducibilities

- Is **mapping** reducibility general enough notion to capture above intuition?
- Not really. For example, any language L is intuitively reducible to its complement, \overline{L} .
- An answer to “**is** $x \in L$ ” is obtained from an answer to “**is** $x \in \overline{L}$ ” by simply reversing the original answer.
- So, in particular, $\overline{A_{TM}}$ should be reducible to A_{TM} . However certainly $\overline{A_{TM}} \not\leq_m A_{TM}$ (**why?**).
- We now seek a **more general notion** of reducibilities than \leq_m .

Oracles

Definition: An **oracle** for a language B is a auxiliary device with two tapes, one called the **query tape**, the other called the **response tape**.

- When a string $x \in \Sigma^*$ is written on the query tape, the oracle writes a “yes/no” answer on the response tape.
- If $x \in B$ the oracle writes “yes”, while if $x \notin B$ the oracle writes “no”.

Oracles

Definition: An **oracle** for a language B is a auxiliary device with two tapes, one called the **query tape**, the other called the **response tape**.

- When a string $x \in \Sigma^*$ is written on the query tape, the oracle writes a “yes/no” answer on the response tape.
- If $x \in B$ the oracle writes “yes”, while if $x \notin B$ the oracle writes “no”.

Remarks

- The oracle always **answers correctly**.
- Oracles are **not realistic** computing devices.

Oracle Turing Machines

Definition: An **oracle Turing machine** is a TM with access to an oracle.

- The TM can query the oracle, and base its future steps upon the oracle's responses.
- We write M^B to denote a TM with an access to an oracle for the language B .

Oracle Turing Machines

Definition: An **oracle Turing machine** is a TM with access to an oracle.

- The TM can query the oracle, and base its future steps upon the oracle's responses.
- We write M^B to denote a TM with an access to an oracle for the language B .

Remarks:

- At any step in its computation, M^B can query the oracle on **just one string**.
- So in a terminating computation only **finitely many queries** can be made.

Turing Reducibility

Definition: Let A and B be two languages. We say that A is **Turing reducible** to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that **decides** A .

Turing Reducibility

Definition: Let A and B be two languages. We say that A is **Turing reducible** to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that **decides** A .

Theorem: If $A \leq_T B$ and B is decidable, then A is decidable.

Turing Reducibility

Definition: Let A and B be two languages. We say that A is **Turing reducible** to B and denote $A \leq_T B$, if there is an oracle Turing machine M^B that **decides** A .

Theorem: If $A \leq_T B$ and B is decidable, then A is decidable.

Simple Observation: If $A \leq_m B$ then $A \leq_T B$. The opposite does not hold.

Hilbert's 10th Problem

For every TM, M , it is possible to construct a polynomial in $n + m$ variables,

$$f_M(y_1, y_2, \dots, y_m, x_1, x_2, \dots, x_n),$$

satisfying: for every $w \in \Sigma^*$ there are integers a_1, a_2, \dots, a_m (depending on w such that M accepts w iff

$$f_M(a_1, a_2, \dots, a_m, x_1, x_2, \dots, x_n)$$

has integer roots x_1, x_2, \dots, x_n .

Hilbert's 10th Problem

Remark: The transformation

$\langle M, w \rangle \rightarrow \langle f_M, a_1, a_2, \dots, a_m \rangle$ is *computable*.

Hilbert's 10th Problem

Remark: The transformation
 $\langle M, w \rangle \rightarrow \langle f_M, a_1, a_2, \dots, a_m \rangle$ is *computable*.

After this reduction is established (don't forget it took 70 year to do), it is obvious Hilbert's 10th problem is undecidable.

Number Theory

Number theory can be viewed as the collection of **true statements** over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y [y = x + 1]$ (existence of successor)

Number Theory

Number theory can be viewed as the collection of **true statements** over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
(Fermat' last theorem for exponent $n = 3$)

Number Theory

Number theory can be viewed as the collection of **true statements** over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
(Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z [p > x \wedge p \neq (y + 1) \cdot (z + 1)]$
(existence of infinitely many primes)

Number Theory

Number theory can be viewed as the collection of **true statements** over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
(Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z [p > x \wedge p \neq (y + 1) \cdot (z + 1)]$
(existence of infinitely many primes)
- $\forall x \exists p \forall y \forall z [p > x \wedge p \neq (y + 1) \cdot (z + 1) \wedge (p + 2) \neq (y + 1) \cdot (z + 1)]$
(the twin prime **conjecture**)

Number Theory

Number theory can be viewed as the collection of **true statements** over the model of natural numbers with addition and multiplication, $(\mathcal{N}, +, \cdot)$. For example

- $\forall x \exists y [y = x + 1]$ (existence of successor)
- $\forall x \forall y \forall z [x \cdot x \cdot x + y \cdot y \cdot y \neq z \cdot z \cdot z]$
(Fermat' last theorem for exponent $n = 3$)
- $\forall x \exists p \forall y \forall z [p > x \wedge p \neq (y + 1) \cdot (z + 1)]$
(existence of infinitely many primes)
- $\forall x \exists p \forall y \forall z [p > x \wedge p \neq (y + 1) \cdot (z + 1) \wedge (p + 2) \neq (y + 1) \cdot (z + 1)]$
(the twin prime **conjecture**)
- **Goldbach conjecture**: Every even integer is the sum of two primes. Express it in the language.

Peano Arithmetic

- The theory of numbers is usually denoted $\text{Th}(\mathcal{N}, +, \cdot)$.

Peano Arithmetic

- The theory of numbers is usually denoted $\text{Th}(\mathcal{N}, +, \cdot)$.
- The “usual” system of axioms of number theory is called **first-order Peano arithmetic**, and denoted by *PA*.

Peano Arithmetic

- The theory of numbers is usually denoted $\text{Th}(\mathcal{N}, +, \cdot)$.
- The “usual” system of axioms of number theory is called **first-order Peano arithmetic**, and denoted by *PA*.
- *PA* includes axioms about the successor operation, well ordering, commutativity and associativity of $+$ and \cdot , distributive law, \dots , and the **induction axiom**.

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Question: Is the theory of integers with just addition, $\text{Th}(\mathcal{N}, +)$, **complete**?

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Question: Is the theory of integers with just addition, $\text{Th}(\mathcal{N}, +)$, **complete**?

For example,

- $\forall x \exists y [y = x + x]$ is true, while
- $\exists y \forall x [y = x + x]$ is false.

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Question: Is the theory of integers with just addition, $\text{Th}(\mathcal{N}, +)$, **complete**?

For example,

- $\forall x \exists y [y = x + x]$ is true, while
- $\exists y \forall x [y = x + x]$ is false.

Using techniques from **finite automata**, turns out this theory is **complete**.

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Completeness of Logical Theories

A logical theory, Th , with an associated axiom system and a model is called **complete** if every **correct statement** is also **provable** (from the axioms).

Question: Is the theory of integers with addition and multiplication $\text{Th}(\mathcal{N}, +, \cdot)$ **complete**?

Gödel's Incompleteness Theorem (1931)

Theorem: $\text{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Gödel's Incompleteness Theorem (1931)

Theorem: $\text{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof: ● By contradiction, using undecidability of Hilbert's 10th.

● Recall M accepts w iff

$$f_M(a_1, a_2, \dots, a_m, x_1, x_2, \dots, x_n)$$

has integer roots x_1, x_2, \dots, x_n .

Gödel's Incompleteness Theorem (1931)

Theorem: $\text{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof: ● By contradiction, using undecidability of Hilbert's 10th.

● Recall M accepts w iff

$$f_M(a_1, a_2, \dots, a_m, x_1, x_2, \dots, x_n)$$

has integer roots x_1, x_2, \dots, x_n .

● Notice that

$$\phi = \exists x_1 \dots \exists x_n f_M(a_1, \dots, a_m, x_1, \dots, x_n) = 0$$

is a statement in our language.

Gödel's Incompleteness Theorem (1931)

Theorem: $\text{Th}(\mathcal{N}, +, \cdot)$ is incomplete.

Proof: ● By contradiction, using undecidability of Hilbert's 10th.

● Recall M accepts w iff

$$f_M(a_1, a_2, \dots, a_m, x_1, x_2, \dots, x_n)$$

has integer roots x_1, x_2, \dots, x_n .

● Notice that

$$\phi = \exists x_1 \dots \exists x_n f_M(a_1, \dots, a_m, x_1, \dots, x_n) = 0$$

is a statement in our language.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was complete, then either ϕ or $\bar{\phi}$ were provable.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was **complete**, then either ϕ or $\bar{\phi}$ were provable.
- By zig-zaging over all possible proofs, TM can always find a proof for any **correct theorem**.

Gödel's Incompleteness Theorem (1931)


- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was **complete**, then either ϕ or $\bar{\phi}$ were provable.
- By zig-zaging over all possible proofs, TM can always find a proof for any **correct theorem**.
- Can assign one TM to try proving ϕ , another to try prove $\bar{\phi}$.

Gödel's Incompleteness Theorem (1931)


- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was **complete**, then either ϕ or $\bar{\phi}$ were provable.
- By zig-zaging over all possible proofs, TM can always find a proof for any **correct theorem**.
- Can assign one TM to try proving ϕ , another to try prove $\bar{\phi}$.
- Exactly one will succeed, determining if M accepts w . Contradiction.

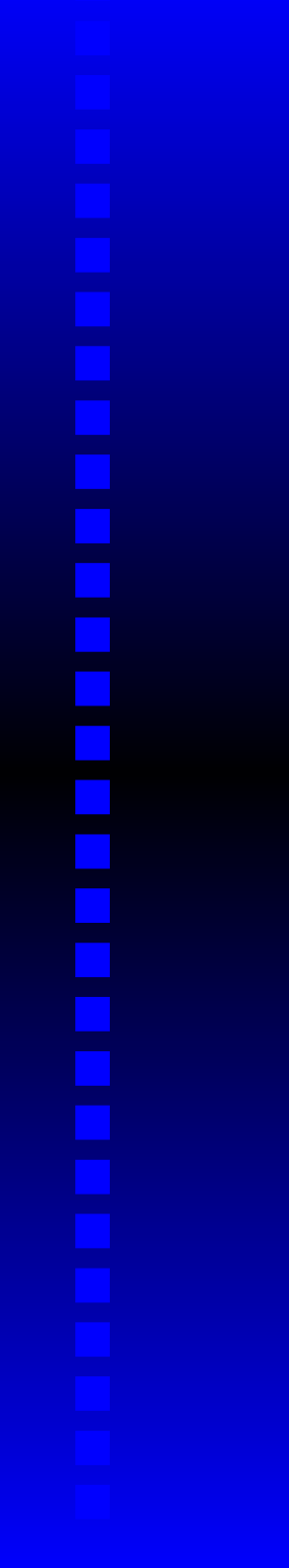


Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was **complete**, then either ϕ or $\bar{\phi}$ were provable.
- By zig-zaging over all possible proofs, TM can always find a proof for any **correct theorem**.
- Can assign one TM to try proving ϕ , another to try prove $\bar{\phi}$.
- Exactly one will succeed, determining if M accepts w . Contradiction. 
- **Important comment:** This conceptually simple proof uses the undecidability of Hilbert's 10th, established in 1970. It was not available to Gödel in 1931, when he proved the theorem.

Gödel's Incompleteness Theorem (1931)

- Either ϕ or $\bar{\phi}$ are correct. If $\text{Th}(\mathcal{N}, +, \cdot)$ was **complete**, then either ϕ or $\bar{\phi}$ were provable.
- By zig-zaging over all possible proofs, TM can always find a proof for any **correct theorem**.
- Can assign one TM to try proving ϕ , another to try prove $\bar{\phi}$.
- Exactly one will succeed, determining if M accepts w . Contradiction. 
- **Important comment:** This conceptually simple proof uses the undecidability of Hilbert's 10th, established in 1970. It was not available to Gödel in 1931, when he proved the theorem.





Undecidability of Tiling Problems

An old times board-and-chalk presentation.