Turing machines: Undecidable problems

Peter Hajnal

Bolyai Institute, SzTE, Szeged

2023 fall

Technical problems

Technical problems

To talk about a computational problem and an algorithm, that soves it first we need a finite alphabet Σ (and much more).

Technical problems

To talk about a computational problem and an algorithm, that soves it first we need a finite alphabet Σ (and much more).

SET THEORY: The finite sets don"t form a set.

Technical problems

To talk about a computational problem and an algorithm, that soves it first we need a finite alphabet Σ (and much more).

SET THEORY: The finite sets don"t form a set.

We can agree on that for coding we use one of the sets $\{1, 2, , \ldots, n\}$ $(n \in \mathbb{N}_+)$.

Technical problems

To talk about a computational problem and an algorithm, that soves it first we need a finite alphabet Σ (and much more).

SET THEORY: The finite sets don"t form a set.

We can agree on that for coding we use one of the sets $\{1, 2, , \ldots, n\}$ $(n \in \mathbb{N}_+)$.

Also we can assume that for a description of an algorithm we use

(1) $k \in N$, the number of tapes,
(2) $m \in \mathbb{N}_+$, the set of states $\{1, 2, \ldots, m\}$,
(3) $\ell \in \mathbb{N}_+$, to describe Γ, the alphabet for the work tapes,
(4) δ, a transition function with finite domain and finite range depending on our previous choices.

Technical problems

To talk about a computational problem and an algorithm, that soves it first we need a finite alphabet Σ (and much more).

SET THEORY: The finite sets don"t form a set.

We can agree on that for coding we use one of the sets $\{1, 2, , \ldots, n\}$ $(n \in \mathbb{N}_+)$.

Also we can assume that for a description of an algorithm we use

(1) $k \in N$, the number of tapes,
(2) $m \in \mathbb{N}_+$, the set of states $\{1, 2, \ldots, m\}$,
(3) $\ell \in \mathbb{N}_+$, to describe Γ, the alphabet for the work tapes,
(4) δ, a transition function with finite domain and finite range depending on our previous choices.

Observation

The number of TM's/algorithms is countably infinite, $\aleph_0$.

Corollary

Theorem

Fix a finite Σ

(i) the set of decidable languages ($L \subset \Sigma^*$) is a countably infinite set,

(ii) the set of enumerable languages ($L \subset \Sigma^*$) is a countably infinite set,

(iii) the set of all languages ($\mathcal{P}(\Sigma)^*$) is set of cardinality continuum.

Corollary

Theorem

Fix a finite Σ

(i) the set of decidable languages ($L \subset \Sigma^*$) is a countably infinite set,

(ii) the set of enumerable languages ($L \subset \Sigma^*$) is a countably infinite set,

(iii) the set of all languages ($\mathcal{P}(\Sigma)^*$) is set of cardinality continuum.

Corollary

There exists non-enumerable, and hence undecidable language.

Coding TM's

Coding TM's

The set of TM's is countably infinite. It gives a possibility to code them.

Coding TM's

The set of TM's is countably infinite. It gives a possibility to code them.

It is easy to agree on a coding system.

Coding TM's

The set of TM's is countably infinite. It gives a possibility to code them.

It is easy to agree on a coding system.

Coding Turing Machines

One can code TM's:

$$T \mapsto \lceil T \rceil.$$

Coding TM's

The set of TM's is countably infinite. It gives a possibility to code them.

It is easy to agree on a coding system.

Coding Turing Machines

One can code TM's:

$$T \mapsto \lceil T \rceil.$$

We can use to $\Sigma = \{0, 1\}$ to code the inputs, outputs (computational problem) and the Turing machines too.

Turing's Theorem (universal TM)

Turing's Theorem (universal TM)

Theorem (Alan Turing)

Assume the we have a coding of inputs and algorithms using the alphabet Σ. There is an algorithm that simulates any algorithm.

Turing's Theorem (universal TM)

Theorem (Alan Turing)

Assume the we have a coding of inputs and algorithms using the alphabet Σ. There is an algorithm that simulates any algorithm. There is a Turing machine that gets $(\lceil\omega\rceil, \lceil T\rceil)$ as input. It computes $T(\omega)$, if T stops on input ω. Furthermore it loops infinitely if T doesn't stop on ω

Turing's Theorem (universal TM)

Theorem (Alan Turing)

Assume the we have a coding of inputs and algorithms using the alphabet Σ. There is an algorithm that simulates any algorithm. There is a Turing machine that gets $(\lceil\omega\rceil, \lceil T\rceil)$ as input. It computes $T(\omega)$, if T stops on input ω. Furthermore it loops infinitely if T doesn't stop on ω

Definition: Universal Turing machine

A machine that is described in the previous theorem is called UNIVERSAL TURING MACHINE.

A dictionary

Complexity theory	Everyday life
Turing machine	Algorithm
An agreement how to code a TM/algorithm	A programming language
$\lceil T \rceil$	A program/code
Universal TM	A computer that is capable to run programs coded in certain language.

Halting problem

Halting problem

Now we are ready to describe a natural language, that is undecidable.

Halting problem

Now we are ready to describe a natural language, that is undecidable.

Definition: Halting problem

$$HALTING = \{(\lceil T \rceil, \lceil \omega \rceil) : \ T \text{ halts on } \omega\}.$$

Turing's theorem on the Halting problem

Turing's theorem on the Halting problem

Turing's theorem

(i) HALTING $\in \mathcal{S}$,

(ii) HALTING $\notin \mathcal{D}$.

Turing's theorem on the Halting problem

Turing's theorem

(i) HALTING $\in \mathcal{S}$,

(ii) HALTING $\notin \mathcal{D}$.

The first part of the theorem is proven by the existence of the universal TM.

Proof

Proof

Proof by contradiction. Assume the I is a TM that solve HALTING.

Proof

Proof by contradiction. Assume the I is a TM that solve HALTING.

We have a few technical assumptions.

> We code (in a bijective manner) the possible inputs by positive integers.

Proof

Proof by contradiction. Assume the I is a TM that solve HALTING.

We have a few technical assumptions.

- We code (in a bijective manner) the possible inputs by positive integers. Furthermore given $i \in \mathbb{N}_+$ we can compute the input ω_i, that is coded by i and vice versa.
- We code (in a bijective manner) the TM's.

Proof

Proof by contradiction. Assume the I is a TM that solve HALTING.

We have a few technical assumptions.

- We code (in a bijective manner) the possible inputs by positive integers. Furthermore given $i \in \mathbb{N}_+$ we can compute the input ω_i, that is coded by i and vice versa.
- We code (in a bijective manner) the TM's. Furthermore given $i \in \mathbb{N}_+$ we can compute the algorithm T_i, that is coded by i and vice versa.

Proof II

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Based on I we can compute the table.

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Based on I we can compute the table. We are hunting for a contradiction.

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Based on I we can compute the table. We are hunting for a contradiction.

The Turing machine E

(Átló): It gets $\omega = \omega_i$, and coputes the diagonal element of the previous table.

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Based on I we can compute the table. We are hunting for a contradiction.

The Turing machine E

(Átló): It gets $\omega = \omega_i$, and coputes the diagonal element of the previous table.

(Switch): If the diagonal element is ∞ the E halts immediately.

Proof II

Imagine that we have a table of type $\mathbb{N}_+ \times \mathbb{N}_+$: In the position (i, j) we have 1 iff T_i stops on ω_j, otherwise we have ∞ (i.e. T_i loops infinitely on ω_j.

Based on I we can compute the table. We are hunting for a contradiction.

The Turing machine E

(Átló): It gets $\omega = \omega_i$, and coputes the diagonal element of the previous table.

(Switch): If the diagonal element is ∞ the E halts immediately.

If the diagonal element is 1 the E runs an infinite loop.

Proof III

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table.

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table. That is 1 iff E halts on ω_i, ∞ otherwise.

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table. That is 1 iff E halts on ω_i, ∞ otherwise.

1st option: E halts on ω_i. By the definition of E it ruins forever:

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table. That is 1 iff E halts on ω_i, ∞ otherwise.

1st option: E halts on ω_i. By the definition of E it ruins forever: a contradiction.

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table. That is 1 iff E halts on ω_i, ∞ otherwise.

1st option: E halts on ω_i. By the definition of E it ruins forever: a contradiction.

2nd option: E runs forever on ω_i. By the definition of E it halts:

Proof III

E is a TM, i.e. $E = T_i$ for certain i.

What happens if we run E on ω_i?

It decodes i, T_i and computes the corresponding diagonal element of our table. That is 1 iff E halts on ω_i, ∞ otherwise.

1st option: E halts on ω_i. By the definition of E it ruins forever: a contradiction.

2nd option: E runs forever on ω_i. By the definition of E it halts: a contradiction.

Break

Hilbert's X. Problem

Hilbert's X. Problem

Hilbert's X. Problem

Let

$$DIOPHANTOS = \{\lceil p(x) \rceil : p \in \mathbb{Z}[x_1, x_2, \ldots, x_n], \\ p \text{ has an integer root}\}.$$

Hilbert's X. Problem

Hilbert's X. Problem

Let

$$DIOPHANTOS = \{\lceil p(x) \rceil : p \in \mathbb{Z}[x_1, x_2, \ldots, x_n], \\ p \text{ has an integer root}\}.$$

Matijaszevics (1970)

$$DIOPHANTOS \notin \mathcal{D}.$$

Hilbert's X. Problem: The history

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

1936 Turing introduces the notion of TM, Church thesis is accepted,

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

1936 Turing introduces the notion of TM, Church thesis is accepted,

1950- Davies and Robinson introduce diophantine sets, and starts to investigate them,

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

1936 Turing introduces the notion of TM, Church thesis is accepted,

1950- Davies and Robinson introduce diophantine sets, and starts to investigate them,

1970 Matijaszevics solves the last (hardest) step, he proves the above theorem.

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

1936 Turing introduces the notion of TM, Church thesis is accepted,

1950- Davies and Robinson introduce diophantine sets, and starts to investigate them,

1970 Matijaszevics solves the last (hardest) step, he proves the above theorem.

It is easy to see that DIOPHANTOSZ$\in \mathcal{S}$ (why?).

Hilbert's X. Problem: The history

1900 Hilbert presents the problem,

1935 Church announces the "Church' thesis",

1936 Turing introduces the notion of TM, Church thesis is accepted,

1950- Davies and Robinson introduce diophantine sets, and starts to investigate them,

1970 Matijaszevics solves the last (hardest) step, he proves the above theorem.

It is easy to see that DIOPHANTOSZ$\in \mathcal{S}$ (why?).

There are special cases when the decision problem is solvable.

Word problem of groups

Word problem of groups

The input of the word problem is a (multiplicative) group.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm?

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.
- Expressions can be constructed from the elements of B, each describing further elements of the group.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.
- Expressions can be constructed from the elements of B, each describing further elements of the group. For example: If $B = \{a, b, c\}$, then $abbaca^{-1}ba^{-1}$ is such an expression.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.
- Expressions can be constructed from the elements of B, each describing further elements of the group. For example: If $B = \{a, b, c\}$, then $abbaca^{-1}ba^{-1}$ is such an expression. 1 is an expression, the empty product, describing the identity element of the group.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.
- Expressions can be constructed from the elements of B, each describing further elements of the group. For example: If $B = \{a, b, c\}$, then $abbaca^{-1}ba^{-1}$ is such an expression. 1 is an expression, the empty product, describing the identity element of the group. So, our expressions, in technical terms, are our "words", constructed by multiplication/concatenation of elements from B and B^{-1}, the inverses of elements from B.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

- We start with a finite set of group elements: B.
- Expressions can be constructed from the elements of B, each describing further elements of the group. For example: If $B = \{a, b, c\}$, then $abbaca^{-1}ba^{-1}$ is such an expression. 1 is an expression, the empty product, describing the identity element of the group. So, our expressions, in technical terms, are our "words", constructed by multiplication/concatenation of elements from B and B^{-1}, the inverses of elements from B.
- Of course, different words can describe the same element.

Word problem of groups

The input of the word problem is a (multiplicative) group.

How to input a group into an algorithm? One solution: Combinatorial group theory.

• We start with a finite set of group elements: B.

• Expressions can be constructed from the elements of B, each describing further elements of the group. For example: If $B = \{a, b, c\}$, then $abbaca^{-1}ba^{-1}$ is such an expression. 1 is an expression, the empty product, describing the identity element of the group. So, our expressions, in technical terms, are our "words", constructed by multiplication/concatenation of elements from B and B^{-1}, the inverses of elements from B.

• Of course, different words can describe the same element. Group theory guarantees that $aa^{-1}b$ and b describe the same element.

Word Problem for Groups: Freely Generated Groups

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.
- If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element.

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

- If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.
- If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.
- This defines an equivalence relation on the set of group expressions that can be written from B.

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

- If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.

- This defines an equivalence relation on the set of group expressions that can be written from B. It is easy to define multiplication, inversion, and the identity class among equivalence classes. Thus, we obtain a group.

Word Problem for Groups: Freely Generated Groups

• An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

• If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.

• This defines an equivalence relation on the set of group expressions that can be written from B. It is easy to define multiplication, inversion, and the identity class among equivalence classes. Thus, we obtain a group.

• This is the *freely generated* group associated with the generator set B.

Word Problem for Groups: Freely Generated Groups

• An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

• If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.

• This defines an equivalence relation on the set of group expressions that can be written from B. It is easy to define multiplication, inversion, and the identity class among equivalence classes. Thus, we obtain a group.

• This is the *freely generated* group associated with the generator set B. It is the *largest* group generated by B.

Word Problem for Groups: Freely Generated Groups

- An elementary simplification of a word is the removal of consecutive pairs xx^{-1} or $x^{-1}x$.

- If in a word sequence $w_1, w_2, w_3, \ldots, w_n$ any two consecutive words are each other's elementary simplification, then any two elements of the sequence describe the same group element. We say that w_1 and w_n are equivalent.

- This defines an equivalence relation on the set of group expressions that can be written from B. It is easy to define multiplication, inversion, and the identity class among equivalence classes. Thus, we obtain a group.

- This is the *freely generated* group associated with the generator set B. It is the *largest* group generated by B.

- For the freely generated group with respect to B, it is easy to design an algorithm that determines whether two given words describe the same group element.

Word Problem for Groups: Finitely Presented Groups

Word Problem for Groups: Finitely Presented Groups

More general groups can be described by generalizing the above method.

Word Problem for Groups: Finitely Presented Groups

More general groups can be described by generalizing the above method.

• Specify word equalities that cannot be derived through elementary simplifications (and, of course, inverses of elementary simplifications).

Word Problem for Groups: Finitely Presented Groups

More general groups can be described by generalizing the above method.

• Specify word equalities that cannot be derived through elementary simplifications (and, of course, inverses of elementary simplifications). If we provide a set of such relations, there corresponds a group to it: the concept of elementary simplification/complication can be extended by rewriting the expression on one side of the equality to the expression on the other side.

Word Problem for Groups: Finitely Presented Groups

More general groups can be described by generalizing the above method.

• Specify word equalities that cannot be derived through elementary simplifications (and, of course, inverses of elementary simplifications). If we provide a set of such relations, there corresponds a group to it: the concept of elementary simplification/complication can be extended by rewriting the expression on one side of the equality to the expression on the other side.

• So, if we have a set B and a set of equalities T (with a word on each side), we have described a group $G = \langle B; T \rangle$.

Word Problem for Groups: Finitely Presented Groups

More general groups can be described by generalizing the above method.

• Specify word equalities that cannot be derived through elementary simplifications (and, of course, inverses of elementary simplifications). If we provide a set of such relations, there corresponds a group to it: the concept of elementary simplification/complication can be extended by rewriting the expression on one side of the equality to the expression on the other side.

• So, if we have a set B and a set of equalities T (with a word on each side), we have described a group $G = \langle B; T \rangle$.

• If B and T are finite, then the groups described in this way are finitely presented groups.

Word Problem for Groups: Examples of Finitely Presented Groups

Word Problem for Groups: Examples of Finitely Presented Groups

Example

$\langle a, b; ab = ba \rangle$ is a group.

Word Problem for Groups: Examples of Finitely Presented Groups

Example

$\langle a, b; ab = ba \rangle$ is a group.

It can be easily verified that this is $(\mathbb{Z}, +) \times (\mathbb{Z}, +)$.

Word Problem for Groups: Examples of Finitely Presented Groups

Example

$\langle a, b; ab = ba \rangle$ is a group.

It can be easily verified that this is $(\mathbb{Z}, +) \times (\mathbb{Z}, +)$.

Example

$\langle a, b; a^n = b^2 = abab = 1 \rangle$ is a group.

Word Problem for Groups: Examples of Finitely Presented Groups

Example

$\langle a, b; ab = ba \rangle$ is a group.

It can be easily verified that this is $(\mathbb{Z}, +) \times (\mathbb{Z}, +)$.

Example

$\langle a, b; a^n = b^2 = abab = 1 \rangle$ is a group.

It can be easily verified that this is D_n.

Word Problem for Groups: The Theorem

Word Problem for Groups: The Theorem

The problem informally: Given a finite generating set B, a finite set of relations T (thus given is a finitely presented group $G = G(B; T)$). Also given are two words built from B. Decide whether they describe the same group element.

Word Problem for Groups: The Theorem

The problem informally: Given a finite generating set B, a finite set of relations T (thus given is a finitely presented group $G = G(B; T)$). Also given are two words built from B. Decide whether they describe the same group element.

Definition

$$\text{WORD PROBLEM} = \{\lceil B, T; w_1 = w_2 \rceil : \text{ in the } \langle B; T \rangle \text{ group, the group elements represented by } w_1 \text{ and } w_2 \text{ are the same}\}$$

Word Problem for Groups: The Theorem

The problem informally: Given a finite generating set B, a finite set of relations T (thus given is a finitely presented group $G = G(B; T)$). Also given are two words built from B. Decide whether they describe the same group element.

Definition

$$\text{WORD PROBLEM} = \{\lceil B, T; w_1 = w_2 \rceil : \text{ in the } \langle B; T \rangle \text{ group, the group elements represented by } w_1 \text{ and } w_2 \text{ are the same}\}$$

Theorem (Novikov (1955), Boone (1958))

The problem is undecidable,

$$\text{WORD PROBLEM} \notin \mathcal{D}.$$

Homeomorphism

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces?

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion:

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion: Starting from simple, well-known topological spaces, we *build* further, more complex ones with simple operations.

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion: Starting from simple, well-known topological spaces, we *build* further, more complex ones with simple operations.

Perhaps the most combinatorial option is to start with simplices.

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion: Starting from simple, well-known topological spaces, we *build* further, more complex ones with simple operations.

Perhaps the most combinatorial option is to start with simplices. Simplices are points, line segments, triangles, tetrahedra. These are precisely the simplices up to three dimensions.

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion: Starting from simple, well-known topological spaces, we *build* further, more complex ones with simple operations.

Perhaps the most combinatorial option is to start with simplices. Simplices are points, line segments, triangles, tetrahedra. These are precisely the simplices up to three dimensions. For every natural number d, a d-dimensional simplex can be defined, for example, as the convex hull of the origin and the standard basis elements e_i in $\mathbb{R}^d$.

Homeomorphism

HOMEOMORPHISM takes as input two topological spaces. We need to determine whether they are homeomorphic.

Again, the essential question: How do we encode topological spaces? The simplest solution for describing a broad class of topological spaces is recursion: Starting from simple, well-known topological spaces, we *build* further, more complex ones with simple operations.

Perhaps the most combinatorial option is to start with simplices. Simplices are points, line segments, triangles, tetrahedra. These are precisely the simplices up to three dimensions. For every natural number d, a d-dimensional simplex can be defined, for example, as the convex hull of the origin and the standard basis elements e_i in $\mathbb{R}^d$.

An operation that can be used for construction is gluing along faces of simplices.

Homeomorphism (continued)

Homeomorphism (continued)

It is easy to prove that the dimension of the initial simplices and the knowledge of the faces used in gluing are sufficient to determine the homomorphism type of the described topological space.

Homeomorphism (continued)

It is easy to prove that the dimension of the initial simplices and the knowledge of the faces used in gluing are sufficient to determine the homomorphism type of the described topological space.

To describe this, we identify simplices and their faces with the set of their vertices.

Homeomorphism (continued)

It is easy to prove that the dimension of the initial simplices and the knowledge of the faces used in gluing are sufficient to determine the homomorphism type of the described topological space.

To describe this, we identify simplices and their faces with the set of their vertices. The simplicial complex becomes a set system over a finite set V.

Homeomorphism (continued)

It is easy to prove that the dimension of the initial simplices and the knowledge of the faces used in gluing are sufficient to determine the homomorphism type of the described topological space.

To describe this, we identify simplices and their faces with the set of their vertices. The simplicial complex becomes a set system over a finite set V. The simplicial complex can be characterized by a single property: every subset belonging to it also belongs to the set (the vertex set of any subset of a simplex is the vertex set of a well-defined face, which is also a simplex).

Homeomorphism: The Theorem

Homeomorphism: The Theorem

Theorem

The SIMPLICIAL-COMPLEXES-HOMEOMORPHISM problem is undecidable. In other words,

$$\text{HOMEOMORPHISM} \notin \mathcal{D}.$$

Post's Domino Problem

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*. For each type, we have infinitely many dominoes at our disposal.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*. For each type, we have infinitely many dominoes at our disposal. The question is whether we can arrange our dominoes into a (finite) row in such a way that when the bottom and top patterns are read together (concatenated), they form the same word.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*. For each type, we have infinitely many dominoes at our disposal. The question is whether we can arrange our dominoes into a (finite) row in such a way that when the bottom and top patterns are read together (concatenated), they form the same word.

Our description was elementary. Instead, the problem can be formulated in the language of semigroups, often referred to in the literature as a problem on semigroups.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*. For each type, we have infinitely many dominoes at our disposal. The question is whether we can arrange our dominoes into a (finite) row in such a way that when the bottom and top patterns are read together (concatenated), they form the same word.

Our description was elementary. Instead, the problem can be formulated in the language of semigroups, often referred to in the literature as a problem on semigroups.

The problem is undecidable.

Post's Domino Problem

In the POST problem, we are given a finite alphabet Σ. The input is a set of dominoes: Finite types of dominoes, where each type has a bottom and a top pattern, each being a word in Σ^*. For each type, we have infinitely many dominoes at our disposal. The question is whether we can arrange our dominoes into a (finite) row in such a way that when the bottom and top patterns are read together (concatenated), they form the same word.

Our description was elementary. Instead, the problem can be formulated in the language of semigroups, often referred to in the literature as a problem on semigroups.

The problem is undecidable.

Theorem (Post)

$$\text{POST} \notin \mathcal{D}.$$

Tiling Problems

Tiling Problems

Divide a square into four quarter-squares with two diagonals. Color each of the resulting squares with a color.

Tiling Problems

Divide a square into four quarter-squares with two diagonals. Color each of the resulting squares with a color. This square is called a tile type.

Tiling Problems

Divide a square into four quarter-squares with two diagonals. Color each of the resulting squares with a color. This square is called a tile type.

Divide the plane with parallel horizontal and vertical lines into square-sized tiles.

Tiling Problems

Divide a square into four quarter-squares with two diagonals. Color each of the resulting squares with a color. This square is called a tile type.

Divide the plane with parallel horizontal and vertical lines into square-sized tiles.

Tiling Problem

Given finitely many tile types. For each type, we have infinitely many tiles.

Tiling Problems

Divide a square into four quarter-squares with two diagonals. Color each of the resulting squares with a color. This square is called a tile type.

Divide the plane with parallel horizontal and vertical lines into square-sized tiles.

Tiling Problem

Given finitely many tile types. For each type, we have infinitely many tiles. Can we tile the plane (can we place a tile in each square of the above partition) in such a way that at the edges meeting an edge, the corresponding two quarters of the tiles have the same color?

Tiling Problems: Wang Tiles

Tiling Problems: Wang Tiles

Wang Tiling Problem

Given finitely many tile types.

Tiling Problems: Wang Tiles

Wang Tiling Problem

Given finitely many tile types. Place one tile of each type side by side on the plane.

Tiling Problems: Wang Tiles

Wang Tiling Problem

Given finitely many tile types. Place one tile of each type side by side on the plane. For each type, we have infinitely many tiles.

Tiling Problems: Wang Tiles

Wang Tiling Problem

Given finitely many tile types. Place one tile of each type side by side on the plane. For each type, we have infinitely many tiles.

Can we tile the plane (can we place a tile in each square of the above partition) in such a way that the placement of individual tiles can be obtained from the placements of the types by shifting, and at the edges meeting an edge, the corresponding two quarters of the tiles have the same color?

Tiling Problems: The Theorems

Tiling Problems: The Theorems

Theorem

$$TILING \notin \mathcal{D}.$$

Tiling Problems: The Theorems

Theorem

$$\mathit{TILING} \notin \mathcal{D}.$$

Theorem

$$\mathit{WANG\text{-}TILING} \notin \mathcal{D}.$$

This is the end!

Thank you for your attention!