Undecidability

Algorithms and Complexity Theory

Matei Popovici[1]

[1]POLITEHNICA University of Bucharest
Computer Science and Engineering Department, Bucharest, Romania

January 10, 2013

What we need to remember from previous courses

Recall:

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine
- The meaning of $M(x) = 1$, where M is a (Deterministic) Turing Machine

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine
- The meaning of $M(x) = 1$, where M is a (Deterministic) Turing Machine
- For each NTM M, there exists an **equivalent** DTM M'

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine
- The meaning of $M(x) = 1$, where M is a (Deterministic) Turing Machine
- For each NTM M, there exists an **equivalent** DTM M'
- What does it mean for two TMs to be **equivalent** ?

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine
- The meaning of $M(x) = 1$, where M is a (Deterministic) Turing Machine
- For each NTM M, there exists an **equivalent** DTM M'
- What does it mean for two TMs to be **equivalent** ?
- The **Universal Turing Machine**

What we need to remember from previous courses

Recall:

- The interpretation of a Turing Machine
- The meaning of $M(x) = 1$, where M is a (Deterministic) Turing Machine
- For each NTM M, there exists an **equivalent** DTM M'
- What does it mean for two TMs to be **equivalent** ?
- The **Universal Turing Machine**

Question: *What are the **limits** of the Turing Machine ?*

Solving problems vs. computing functions

$$M(x) = 1$$

Solving problems vs. computing functions

$$M(x) = 1$$

Proposition

Let Σ be an alphabet. Σ^* is ***countably infinite.***

Solving problems vs. computing functions

$$M(x) = 1$$

Proposition

*Let Σ be an alphabet. Σ^* is **countably infinite.***

Remark

*Any **problem instance** can be interpreted as a natural number.*
*Any **problem** can be interpreted as a function $f : \mathbb{N} \to \{0, 1\}$*

Solving problems vs. computing functions

$$M(x) = 1$$

Proposition

*Let Σ be an alphabet. Σ^* is **countably infinite.***

Remark

*Any **problem instance** can be interpreted as a natural number.*
*Any **problem** can be interpreted as a function $f : \mathbb{N} \to \{0, 1\}$*

Let M be a TM and $f : \mathbb{N} \to \{0, 1\}$ be a function.

Solving problems vs. computing functions

$$M(x) = 1$$

Proposition

*Let Σ be an alphabet. Σ^* is **countably infinite.***

Remark

*Any **problem instance** can be interpreted as a natural number.*
*Any **problem** can be interpreted as a function $f : \mathbb{N} \to \{0, 1\}$*

Let M be a TM and $f : \mathbb{N} \to \{0, 1\}$ be a function.

Definition

We say M **decides** f iff $M(\mathbf{enc(n)}) = f(n)$ for all $n \in \mathbb{N}$.

Solving problems vs. computing functions

$$M(x) = 1$$

Proposition

*Let Σ be an alphabet. Σ^* is **countably infinite.***

Remark

*Any **problem instance** can be interpreted as a natural number.*
*Any **problem** can be interpreted as a function $f : \mathbb{N} \to \{0, 1\}$*

Let M be a TM and $f : \mathbb{N} \to \{0, 1\}$ be a function.

Definition

We say M **decides** f iff $M(\mathbf{enc(n)}) = f(n)$ for all $n \in \mathbb{N}$.

Definition

We say M **accepts** f iff $M(enc(n)) = 1$ whenever $f(n) = 1$, and $M(\mathbf{enc(n)})$ does not halt whenever $f(n) = 0$

Turing Machines vs functions

How many TMs, and how many functions $f : \mathbb{N} \to \{0, 1\}$?

Turing Machines vs functions

How many TMs, and how many functions $f : \mathbb{N} \to \{0, 1\}$?

Proposition

The set $\mathbb{H}\mathrm{om}(\mathbb{N}, \{0, 1\}) = \{f \mid f : \mathbb{N} \to \{0, 1\}\}$ *is* ***uncountably infinite***

Turing Machines vs functions

How many TMs, and how many functions $f : \mathbb{N} \to \{0, 1\}$?

Proposition

The set $\mathbb{H}\mathrm{om}(\mathbb{N}, \{0, 1\}) = \{f \mid f : \mathbb{N} \to \{0, 1\}\}$ *is* ***uncountably infinite***

Proof.

Blackboard □

Turing Machines vs functions

How many TMs, and how many functions $f : \mathbb{N} \to \{0, 1\}$?

Proposition

The set $\mathbb{H}\text{om}(\mathbb{N}, \{0, 1\}) = \{f \mid f : \mathbb{N} \to \{0, 1\}\}$ *is* ***uncountably infinite***

Proof.

Blackboard □

Proposition

The set of ***Turing Machines*** *is* ***countably infinite***

Turing Machines vs functions

How many TMs, and how many functions $f : \mathbb{N} \to \{0, 1\}$?

Proposition

The set $\mathbb{H}\text{om}(\mathbb{N}, \{0, 1\}) = \{f \mid f : \mathbb{N} \to \{0, 1\}\}$ *is* ***uncountably infinite***

Proof.

Blackboard □

Proposition

The set of ***Turing Machines*** *is* ***countably infinite***

Proof.

Blackboard □

Recursive and recursive-enumerable sets

Let $A \subseteq \mathbb{N}$, $f : \mathbb{N} \to \{0, 1\}$, and $f_A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{otherwise} \end{cases}$

Recursive and recursive-enumerable sets

Let $A \subseteq \mathbb{N}$, $f : \mathbb{N} \to \{0, 1\}$, and $f_A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{otherwise} \end{cases}$

Definition (Recursive set, function)

f is recursive if there exists a TM which **decides** f. A is recursive if there exists a TM which decides the f_A.

Recursive and recursive-enumerable sets

Let $A \subseteq \mathbb{N}$, $f : \mathbb{N} \to \{0, 1\}$, and $f_A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{otherwise} \end{cases}$

Definition (Recursive set, function)

f is recursive if there exists a TM which **decides** f. A is recursive if there exists a TM which decides the f_A.

Definition (Recursively enumerable set, function)

f is recursively enumerable if there exists a TM which **accepts** f. A is recursively enumerable if there exists a TM which **accepts** f_A.

Recursive and recursive-enumerable sets

Let $A \subseteq \mathbb{N}$, $f : \mathbb{N} \to \{0, 1\}$, and $f_A(n) = \begin{cases} 1 & \text{if } n \in A \\ 0 & \text{otherwise} \end{cases}$

Definition (Recursive set, function)

f is recursive if there exists a TM which **decides** f. A is recursive if there exists a TM which decides the f_A.

Definition (Recursively enumerable set, function)

f is recursively enumerable if there exists a TM which **accepts** f. A is recursively enumerable if there exists a TM which **accepts** f_A.

Definition

Let R be the set of **recursive** functions and RE be the set of **recursively-enumerable** functions.

R and RE

What is the relationship between R and RE ?

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

Proof.

blackboard □

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

Proof.

blackboard □

Proposition

$R \neq RE$

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

Proof.

blackboard □

Proposition

$R \neq RE$

Proof.

blackboard □

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

Proof.

blackboard □

Proposition

$R \neq RE$

Proof.

blackboard □

Proposition

The halting problem is in RE but not in R

R and RE

What is the relationship between R and RE ?

Proposition

R is contained in RE

Proof.

blackboard □

Proposition

$R \neq RE$

Proof.

blackboard □

Proposition

The halting problem is in RE but not in R

Proof.

blackboard □

Insights on R and RE

Insights on R and RE:

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**
- Problems which are important for the world, are **undecidable** (i.e. not in R, i.e. not solvable by computers)

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**
- Problems which are important for the world, are **undecidable** (i.e. not in R, i.e. not solvable by computers)
- *Who/What* exactly are those problems ?

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**
- Problems which are important for the world, are **undecidable** (i.e. not in R, i.e. not solvable by computers)
- *Who/What* exactly are those problems ?

Theorem (Rice [1])

Let $\mathcal{C} \subseteq RE$, $\mathcal{C} \neq \emptyset$ and M be a TM. Establishing whether the function accepted by M is a member of $\mathcal{C}$ is undecidable

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**
- Problems which are important for the world, are **undecidable** (i.e. not in R, i.e. not solvable by computers)
- *Who/What* exactly are those problems ?

Theorem (Rice [1])

Let $\mathcal{C} \subseteq RE$, $\mathcal{C} \neq \emptyset$ and M be a TM. Establishing whether the function accepted by M is a member of $\mathcal{C}$ is undecidable

The set $\mathcal{C}$ can be interpreted as a **property** of *algorithms* (that is TM's that **accept** functions $f : \mathbb{N} \to \{0, 1\}$).

Insights on R and RE

Insights on R and RE:

- Turing Machine (and other models of computation) are **limited**
- Problems which are important for the world, are **undecidable** (i.e. not in R, i.e. not solvable by computers)
- *Who/What* exactly are those problems ?

Theorem (Rice [1])

Let $\mathcal{C} \subseteq RE$, $\mathcal{C} \neq \emptyset$ and M be a TM. Establishing whether the function accepted by M is a member of $\mathcal{C}$ is undecidable

The set $\mathcal{C}$ can be interpreted as a **property** of *algorithms* (that is TM's that **accept** functions $f : \mathbb{N} \to \{0, 1\}$). Verifying whether an *algorithm* has an **extensional** and **non-trivial** property, is undecidable.

Undecidability in practice

Some important undecidable problems:

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*
- *The satisfiability problem for First-Order Logic*

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*
- *The satisfiability problem for First-Order Logic*
- *The domino problem (see introductory slides)*

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*
- *The satisfiability problem for First-Order Logic*
- *The domino problem (see introductory slides)*
- *Solving Diophantine equations*

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*
- *The satisfiability problem for First-Order Logic*
- *The domino problem (see introductory slides)*
- *Solving Diophantine equations*
- *Type inference in System F (polymorphic lambda calculus)*

Undecidability in practice

Some important undecidable problems:

- *Verifying **total** program correctness*
- *The satisfiability problem for First-Order Logic*
- *The domino problem (see introductory slides)*
- *Solving Diophantine equations*
- *Type inference in System F (polymorphic lambda calculus)*

Beyond R and RE. Is there something more ?

Bibliography I


C.H. Papadimitriou.
Computational complexity.
Addison-Wesley, 1994.