Undecidable Problems in Algebra

Turing Machines and Computability

Matthew Moore

The University of Colorado at Boulder

March 1, 2011

This is the first talk in a series. The goal of the series is to address certain undecidable problems in algebra by associating to each Turing machine an algebraic structure. This talk will cover the requisite background information on Turing machines, computability, and the halting problem. No prior knowledge of computability theory will be assumed.

The Goal

- Associate to each Turing machine, $\mathcal{T}$, an algebraic object, $\mathbb{A}(\mathcal{T})$ such that...

The Goal

- Associate to each Turing machine, $\mathcal{T}$, an algebraic object, $\mathbb{A}(\mathcal{T})$ such that...

- $\mathbb{A}(\mathcal{T})$ has property **P** if and only if $\mathcal{T}$ halts.

The Goal

- Associate to each Turing machine, $\mathcal{T}$, an algebraic object, $\mathbb{A}(\mathcal{T})$ such that...
- $\mathbb{A}(\mathcal{T})$ has property **P** if and only if $\mathcal{T}$ halts.
- Since determining if $\mathcal{T}$ halts is an undecidable problem, we will have shown that determining if an algebra has property **P** is undecidable.

Turing Machines

Definition

A Turing machine consists of

Turing Machines

Definition

A Turing machine consists of

1. an infinite tape

Turing Machines

Definition

A Turing machine consists of

1. an infinite tape divided into cells,

Turing Machines

Definition

A Turing machine consists of

1. an infinite tape divided into cells, each containing either a 0 or a 1;

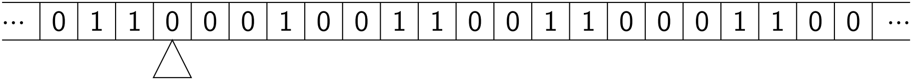
$\cdots$	0	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1	0	0	$\cdots$

Turing Machines

Definition

A Turing machine consists of

1. an infinite tape divided into cells, each containing either a 0 or a 1;
2. a movable "head" that can read the contents of whatever cell it is currently placed over and write a 0 or 1 to that cell;

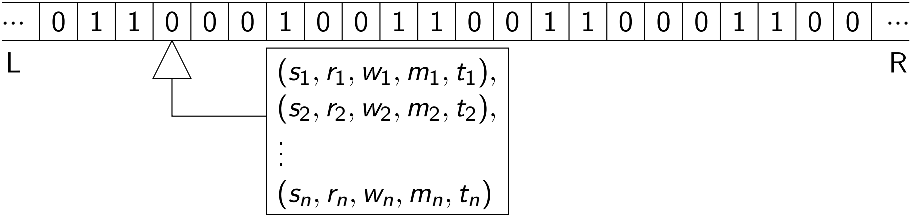


Turing Machines

Definition

A Turing machine consists of

1. an infinite tape divided into cells, each containing either a 0 or a 1;
2. a movable "head" that can read the contents of whatever cell it is currently placed over and write a 0 or 1 to that cell; and
3. a program, consisting of a (finite) sequence 5-tuples, (s, r, w, m, t), meant to be interpreted as "if in state s and reading r, then write w, move m, and enter state t.



More definitions...

Definition

- The contents of the tape that a Turing machine is given is called the *input*. All inputs are assumed to be finite (only finitely many 1's).

More definitions...

Definition

- The contents of the tape that a Turing machine is given is called the *input*. All inputs are assumed to be finite (only finitely many 1's). Since the set of all inputs is countable, we fix some enumeration of them and refer to the input by a natural number, $\varphi(n)$

More definitions...

Definition

- The contents of the tape that a Turing machine is given is called the *input*. All inputs are assumed to be finite (only finitely many 1's). Since the set of all inputs is countable, we fix some enumeration of them and refer to the input by a natural number, $\varphi(n)$
- If the Turing machine eventually stops computing, it is said to have *halted*.

More definitions...

Definition

- The contents of the tape that a Turing machine is given is called the *input*. All inputs are assumed to be finite (only finitely many 1's). Since the set of all inputs is countable, we fix some enumeration of them and refer to the input by a natural number, $\varphi(n)$
- If the Turing machine eventually stops computing, it is said to have *halted*.
- If a Turing machine has halted on some input, the contents of the tape are called the *output*.

More definitions...

Definition

- The contents of the tape that a Turing machine is given is called the *input*. All inputs are assumed to be finite (only finitely many 1's). Since the set of all inputs is countable, we fix some enumeration of them and refer to the input by a natural number, $\varphi(n)$
- If the Turing machine eventually stops computing, it is said to have *halted*.
- If a Turing machine has halted on some input, the contents of the tape are called the *output*. We enumerate the output. $(\varphi(n) = m)$

Examples

1. Turing's first machine: take any tape as input and print the sequence $1, 0, 1, 0, \ldots$.

Examples

1. Turing's first machine: take any tape as input and print the sequence $1, 0, 1, 0, \ldots$.

$$\begin{array}{lllll}
(\mathrm{START}, & 0, & 1 & \mathrm{R}, & s_1) \\
(\mathrm{START}, & 1, & 1 & \mathrm{R}, & s_1) \\
(s_1, & 0, & 0 & \mathrm{R}, & \mathrm{START}) \\
(s_1, & 1, & 0 & \mathrm{R}, & \mathrm{START})
\end{array}$$

Examples

1. Turing's first machine: take any tape as input and print the sequence $1, 0, 1, 0, \ldots$.

$$\begin{array}{lllll}
(\text{START}, & 0, & 1 & \text{R}, & s_1) \\
(\text{START}, & 1, & 1 & \text{R}, & s_1) \\
(s_1, & 0, & 0 & \text{R}, & \text{START}) \\
(s_1, & 1, & 0 & \text{R}, & \text{START})
\end{array}$$

2. Addition: take two numbers (in unary) and output their sum.

Examples

1. Turing's first machine: take any tape as input and print the sequence $1, 0, 1, 0, \ldots$.

$$\begin{array}{lllll} (\text{START}, & 0, & 1 & \text{R}, & s_1) \\ (\text{START}, & 1, & 1 & \text{R}, & s_1) \\ (s_1, & 0, & 0 & \text{R}, & \text{START}) \\ (s_1, & 1, & 0 & \text{R}, & \text{START}) \end{array}$$

2. Addition: take two numbers (in unary) and output their sum.

$$\begin{array}{lllll} (\text{START}, & 1, & 0, & \text{R}, & s_1) \\ (s_1, & 1, & 1, & \text{R}, & s_1) \\ (s_1, & 0, & 1, & \text{R}, & \text{HALT}) \end{array}$$

More Examples

- Multiplication by 2: take a single number (in unary) and output 2 times that number.

More Examples

- Multiplication by 2: take a single number (in unary) and output 2 times that number.

$$
\begin{array}{lllll}
(\mathrm{START}, & 1, & 0, & \mathrm{R}, & s_1) \\
(s_1, & 1, & 1, & \mathrm{R}, & s_1) \\
(s_1, & 0, & 0, & \mathrm{R}, & s_2) \\
(s_2, & 0, & 1, & \mathrm{R}, & s_3) \\
(s_2, & 1, & 1, & \mathrm{R}, & s_2) \\
(s_3, & 0, & 1, & \mathrm{L}, & s_4) \\
(s_4, & 1, & 1, & \mathrm{L}, & s_4) \\
(s_4, & 0, & 0, & \mathrm{L}, & s_5) \\
(s_5, & 1, & 1, & \mathrm{L}, & s_5) \\
(s_5, & 0, & 0, & \mathrm{R}, & \mathrm{START})
\end{array}
$$

More Examples

- Multiplication by 2: take a single number (in unary) and output 2 times that number.

$$
\begin{array}{lllll}
(\text{START}, & 1, & 0, & \text{R}, & s_1) \\
(s_1, & 1, & 1, & \text{R}, & s_1) \\
(s_1, & 0, & 0, & \text{R}, & s_2) \\
(s_2, & 0, & 1, & \text{R}, & s_3) \\
(s_2, & 1, & 1, & \text{R}, & s_2) \\
(s_3, & 0, & 1, & \text{L}, & s_4) \\
(s_4, & 1, & 1, & \text{L}, & s_4) \\
(s_4, & 0, & 0, & \text{L}, & s_5) \\
(s_5, & 1, & 1, & \text{L}, & s_5) \\
(s_5, & 0, & 0, & \text{R}, & \text{START})
\end{array}
$$

Church-Turing Thesis

Any effectively calculable function is a computable function.

The Halting Problem

Since a Turing machine consists of a finite list of instructions, the set of all Turing machines is countable. We enumerate them as φ_i, $i \in \mathbb{N}$.

The Halting Problem

Since a Turing machine consists of a finite list of instructions, the set of all Turing machines is countable. We enumerate them as φ_i, $i \in \mathbb{N}$. Let

$$K = \{(i, x) \mid \varphi_i(x) \text{ halts}\}.$$

The Halting Problem

Since a Turing machine consists of a finite list of instructions, the set of all Turing machines is countable. We enumerate them as φ_i, $i \in \mathbb{N}$. Let

$$K = \{(i, x) \mid \varphi_i(x) \text{ halts}\}.$$

Question: Given a pair, $(i, x) \in \mathbb{N}^2$, can we *computably* decide if $(i, x) \in K$?

The Halting Problem

Since a Turing machine consists of a finite list of instructions, the set of all Turing machines is countable. We enumerate them as φ_i, $i \in \mathbb{N}$. Let

$$K = \{(i, x) \mid \varphi_i(x) \text{ halts}\}.$$

Question: Given a pair, $(i, x) \in \mathbb{N}^2$, can we *computably* decide if $(i, x) \in K$?
That is... Is there a Turing machine, η, such that $\eta = \chi_K$?

The Halting Problem

Since a Turing machine consists of a finite list of instructions, the set of all Turing machines is countable. We enumerate them as φ_i, $i \in \mathbb{N}$. Let

$$K = \{(i, x) \mid \varphi_i(x) \text{ halts}\}.$$

Question: Given a pair, $(i, x) \in \mathbb{N}^2$, can we *computably* decide if $(i, x) \in K$?
That is... Is there a Turing machine, η, such that $\eta = \chi_K$?
Answer: No!

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists.

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this).

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this). Let $\psi(x) = \Psi(x, x)$.

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this). Let $\psi(x) = \Psi(x, x)$. Since ψ is computable, there is some y such that $\psi = \varphi_y$.

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this). Let $\psi(x) = \Psi(x, x)$. Since ψ is computable, there is some y such that $\psi = \varphi_y$. Note that ψ is total, so $(y, y) \in K$.

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this). Let $\psi(x) = \Psi(x, x)$. Since ψ is computable, there is some y such that $\psi = \varphi_y$. Note that ψ is total, so $(y, y) \in K$. Therefore

$$\varphi_y(y) = \psi(y)$$

The Halting Problem

Theorem

There is no Turing machine, η, such that $\eta = \chi_K$.

Proof. Suppose that such an η exists. Then

$$\Psi(x, i) = \begin{cases} \varphi_i(x) + 1 & \text{if } \eta(i, x) = 1 \\ 1 & \text{otherwise} \end{cases}$$

is computable (need a universal Turing machine for this). Let $\psi(x) = \Psi(x, x)$. Since ψ is computable, there is some y such that $\psi = \varphi_y$. Note that ψ is total, so $(y, y) \in K$. Therefore

$$\varphi_y(y) = \psi(y) = \varphi_y(y) + 1,$$

a contradiction. □

Summary (and more!)

- A Turing machine is theoretic construction to model a kind of computer.

Summary (and more!)

- A Turing machine is theoretic construction to model a kind of computer.
- The Church-Turing Thesis:

  Every effectively calculable function is computable.

Summary (and more!)

- A Turing machine is theoretic construction to model a kind of computer.
- The Church-Turing Thesis:

  Every effectively calculable function is computable.
- There is no algorithm to determine whether a general Turing machine halts (K is undecidable).

Summary (and more!)

- A Turing machine is theoretic construction to model a kind of computer.
- The Church-Turing Thesis:

  Every effectively calculable function is computable.
- There is no algorithm to determine whether a general Turing machine halts (K is undecidable).
- Given a property $\mathbf{P}$, if we show

$$\forall \mathcal{T}\ [(\mathcal{T} \text{ has } \mathbf{P}) \Leftrightarrow (\mathcal{T} \text{ halts})],$$

Summary (and more!)

- A Turing machine is theoretic construction to model a kind of computer.
- The Church-Turing Thesis:

  Every effectively calculable function is computable.
- There is no algorithm to determine whether a general Turing machine halts (K is undecidable).
- Given a property $\mathbf{P}$, if we show

$$\forall \mathcal{T}\ [(\mathcal{T} \text{ has } \mathbf{P}) \Leftrightarrow (\mathcal{T} \text{ halts})],$$

  then we have shown that there is no algorithm to decide if $\mathcal{T}$ has $\mathbf{P}$.

Thank you.