Theory of Computation: The Recursion Theorem

Arjun Chandrasekhar

This is my favorite lecture

This is my favorite lecture

- I love this topic

This is my favorite lecture

- I love this topic
- This is the first lecture I ever made from scratch

This is my favorite lecture

- I love this topic
- This is the first lecture I ever made from scratch
- For once, Turing machines will be *more* convenient than modern programming languages

This is my favorite lecture

- I love this topic
- This is the first lecture I ever made from scratch
- For once, Turing machines will be *more* convenient than modern programming languages
- This lecture got me my first full-time teaching job

This is my favorite lecture

- I love this topic
- This is the first lecture I ever made from scratch
- For once, Turing machines will be *more* convenient than modern programming languages
- This lecture got me my first full-time teaching job
- This lecture lead to one of my favorite teaching stories

Pop quiz!

For extra credit: write a non-empty program that prints out its own source code (without using any I/O operations). You have 20 minutes.

Recall

Recall

- Certain problems are not decidable, or even recognizable

Recall

- Certain problems are not decidable, or even recognizable
- Initially we proved this with Diagonalization

Recall

- Certain problems are not decidable, or even recognizable
- Initially we proved this with Diagonalization
- Usually we prove this using reduction

Recall

- Certain problems are not decidable, or even recognizable
- Initially we proved this with Diagonalization
- Usually we prove this using reduction
  - If A is undecidable/unrecognizable, and $A \leq_M B$, then B is undecidable/unrecognizable

Recall

- Certain problems are not decidable, or even recognizable
- Initially we proved this with Diagonalization
- Usually we prove this using reduction
  - If A is undecidable/unrecognizable, and $A \leq_M B$, then B is undecidable/unrecognizable
- Is there an alternative (perhaps easier) way to prove these results?

Today's goals

Today's goals

- Understand how to construct a self-reproducing Turing machine

Today's goals

- Understand how to construct a self-reproducing Turing machine
- Describe Turing machines that analyze (and contradict) their own behavior

Today's goals

- Understand how to construct a self-reproducing Turing machine
- Describe Turing machines that analyze (and contradict) their own behavior
- Use the recursion theorem for concise impossiblity proofs

Self-reproducing robots

Self-reproducing robots

- Can we build a robot that builds another robot?

Self-reproducing robots

- Can we build a robot that builds another robot?
- Can we build a robot that builds another robot-building robot?

Self-reproducing robots

- Can we build a robot that builds another robot?
- Can we build a robot that builds another robot-building robot?
- Can we build a robot-building robot that builds an <u>identical</u> robot-building robot?

Self-reproducing programs

Self-reproducing programs

- Can we ~~build~~ write a ~~robot~~ program that builds another ~~robot~~ program?

Self-reproducing programs

- Can we ~~build~~ write a ~~robot~~ program that builds another ~~robot~~ program?
- Can we ~~build~~ write a ~~robot~~ program that builds another ~~robot-building robot~~ program-building program?

Self-reproducing programs

- Can we ~~build~~ write a ~~robot~~ program that builds another ~~robot~~ program?
- Can we ~~build~~ write a ~~robot~~ program that builds another ~~robot-building robot~~ program-building program?
- Can we build a ~~robot-building robot~~ program-building program that builds an identical ~~robot-building robot~~ program-building program?

Statement of the recursion theorem

Statement of the recursion theorem

The recursion theorem:

Statement of the recursion theorem

The recursion theorem:

1. There exists a Turing machine that prints out its own description

Statement of the recursion theorem

The recursion theorem:

1. There exists a Turing machine that prints out its own description
2. A Turing machine can be programmed to obtain and then analyze its own description.

Statement of the recursion theorem

The recursion theorem:

1. There exists a Turing machine that prints out its own description
2. A Turing machine can be programmed to obtain and then analyze its own description.

We will prove both parts by construction.

Statement of the recursion theorem

The recursion theorem:

1. There exists a Turing machine that prints out its own description
2. A Turing machine can be programmed to obtain and then analyze its own description.

We will prove both parts by construction.

- In doing so, we will see why the simplicity of the TM model can be convenient!

Idea behind a self-reproducing programs

Print the following sentence twice, the second time in quotes
"Print the following sentence twice, the second time in quotes"

Review of notation

Review of notation

- If M is a Turing machine, then $\langle M \rangle$ refers to the string description of M

Review of notation

- If M is a Turing machine, then $\langle M \rangle$ refers to the string description of M
- Can think of $\langle M \rangle$ as the source code, and M is the executable file

Review of notation

- If M is a Turing machine, then $\langle M \rangle$ refers to the string description of M
- Can think of $\langle M \rangle$ as the source code, and M is the executable file
- Think back to the first two programming assignments

Review of notation

- If M is a Turing machine, then $\langle M \rangle$ refers to the string description of M
- Can think of $\langle M \rangle$ as the source code, and M is the executable file
- Think back to the first two programming assignments
  - On the first assignment, you read a string description $\langle D \rangle$ of a DFA and parsed it into an actual DFA object D

Review of notation

- If M is a Turing machine, then $\langle M \rangle$ refers to the string description of M
- Can think of $\langle M \rangle$ as the source code, and M is the executable file
- Think back to the first two programming assignments
  - On the first assignment, you read a string description $\langle D \rangle$ of a DFA and parsed it into an actual DFA object D
  - On the second assignment, you read a string description $\langle N \rangle$ of an NFA, converted it into an equivalent DFA object D, and output a string description $\langle D \rangle$

Building blocks of our self-reproducing program

Building blocks of our self-reproducing program

To construct a Turing machine that prints out its own description, we will split the program up into two sub-machines Q and $P_{\langle Q \rangle}$

Building blocks of our self-reproducing program

To construct a Turing machine that prints out its own description, we will split the program up into two sub-machines Q and $P_{\langle Q \rangle}$

- Machine Q takes an input string w and creates a machine P_w that always prints out the same string w

Building blocks of our self-reproducing program

To construct a Turing machine that prints out its own description, we will split the program up into two sub-machines Q and $P_{\langle Q \rangle}$

- Machine Q takes an input string w and creates a machine P_w that always prints out the same string w
- Machine $P_{\langle Q \rangle}$ is a Turing machine that ignores its input and always prints out $\langle Q \rangle$, i.e. a description of the machine Q

Machines that only print one string

Machines that only print one string

Machines that only print one string

Let P_w be a Turing machine that erases whatever is on the tape and prints out string w.

Machines that only print one string

Let P_w be a Turing machine that erases whatever is on the tape and prints out string w.

- w is a constant that we decide on ahead of time

Machines that only print one string

Let P_w be a Turing machine that erases whatever is on the tape and prints out string w.

- w is a constant that we decide on ahead of time
- Can a human construct this machine?

Machines that only print one string

Let P_w be a Turing machine that erases whatever is on the tape and prints out string w.

- w is a constant that we decide on ahead of time
- Can a human construct this machine?
- Can we write a computer program to construct this machine?

Machines that create single-string machines

Machines that create single-string machines

Let $q(w) = \langle P_w \rangle$. That is, the function q takes as input a string w. It outputs the description of a Turing machine P_w that always prints out the string w.

Machines that create single-string machines

Let $q(w) = \langle P_w \rangle$. That is, the function q takes as input a string w. It outputs the description of a Turing machine P_w that always prints out the string w.

- Is q a Turing-computable function?

Machines that create single-string machines

Let $q(w) = \langle P_w \rangle$. That is, the function q takes as input a string w. It outputs the description of a Turing machine P_w that always prints out the string w.

- Is q a Turing-computable function?
- If you understand what q is doing and convince yourself that q is computable, then the proof of the recursion theorem will be straightforward

Machines that create single-string machines

Machines that create single-string machines

- Consider the Turing machine $P_{\text{computability}}$ ignores its input and always prints out the same string "computability"

Machines that create single-string machines

- Consider the Turing machine $P_{\text{computability}}$ ignores its input and always prints out the same string "computability"
- Can we construct a machine $Q_{\text{computability}}$ that prints out $\langle P_{\text{computability}} \rangle$?

Machines that create single-string machines

- Consider the Turing machine $P_{\text{computability}}$ ignores its input and always prints out the same string "computability"
- Can we construct a machine $Q_{\text{computability}}$ that prints out $\langle P_{\text{computability}} \rangle$?
  - Yes!

Machines that create single-string machines

- Consider the Turing machine $P_{\text{computability}}$ ignores its input and always prints out the same string "computability"
- Can we construct a machine $Q_{\text{computability}}$ that prints out $\langle P_{\text{computability}} \rangle$?
  - Yes!
  - We can treat $\langle P_{\text{computability}} \rangle$ like any other pre-determined string w

Machines that create single-string machines

- Consider the Turing machine $P_{\text{recursion}}$ ignores its input and always prints out the same string "recursion"

Machines that create single-string machines

- Consider the Turing machine $P_{\text{recursion}}$ ignores its input and always prints out the same string "recursion"
- Can we construct a machine $Q_{\text{recursion}}$ that prints out $\langle P_{\text{recursion}} \rangle$?

Machines that create single-string machines

- Consider the Turing machine $P_{\text{recursion}}$ ignores its input and always prints out the same string "recursion"
- Can we construct a machine $Q_{\text{recursion}}$ that prints out $\langle P_{\text{recursion}} \rangle$?
  - Yes!

Machines that create single-string machines

- Consider the Turing machine $P_{\text{recursion}}$ ignores its input and always prints out the same string "recursion"
- Can we construct a machine $Q_{\text{recursion}}$ that prints out $\langle P_{\text{recursion}} \rangle$?
  - Yes!
  - It's just like the last slide. We use the string "recursion" instead of the string "computability"

Machines that create single-string machines

- Can we construct a machine Q_w that prints out $\langle P_w \rangle$ for *any* arbitrary string w?

Machines that create single-string machines

- Can we construct a machine Q_{w} that prints out $\langle P_{\text{w}} \rangle$ for *any* arbitrary string w?
  - Yes!

Machines that create single-string machines

- Can we construct a machine Q_{w} that prints out $\langle P_{\mathrm{w}} \rangle$ for *any* arbitrary string w?
  - Yes!
  - It's just like the last two slides. Instead of the string "recursion" or "computability", we substitute our string of choice w

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?
- Yes!

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?
- Yes!
- We know what 99% of the program P_w will look like

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?
- Yes!
- We know what 99% of the program P_w will look like
- If we are given w, we know how to finish writing the program

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?
- Yes!
- We know what 99% of the program P_w will look like
- If we are given w, we know how to finish writing the program
  - Substitute in w for the part where P prints out its output string

Machines that create single-string machines

- Can we construct a machine Q that takes as input w and outputs $\langle P_w \rangle$, the description of the machine P_w?
- Yes!
- We know what 99% of the program P_w will look like
- If we are given w, we know how to finish writing the program
  - Substitute in w for the part where P prints out its output string
  - We basically convert w from a constant to a parameter of the method Q

A self-reproducing program

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:
  1. Compute $\langle P_{\langle M \rangle} \rangle$, i.e. a description of a machine that ignores its input and prints out the string $\langle M \rangle$ (where $\langle M \rangle$ is also a Turing machine description)

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:
  1. Compute $\langle P_{\langle M \rangle} \rangle$, i.e. a description of a machine that ignores its input and prints out the string $\langle M \rangle$ (where $\langle M \rangle$ is also a Turing machine description)
  2. Combine $\langle P_{\langle M \rangle} \rangle$ and $\langle M \rangle$ into a single machine M^*

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:
  1. Compute $\langle P_{\langle M \rangle} \rangle$, i.e. a description of a machine that ignores its input and prints out the string $\langle M \rangle$ (where $\langle M \rangle$ is also a Turing machine description)
  2. Combine $\langle P_{\langle M \rangle} \rangle$ and $\langle M \rangle$ into a single machine M^*
  3. Print out $\langle M^* \rangle$ and halt

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:
  1. Compute $\langle P_{\langle M \rangle} \rangle$, i.e. a description of a machine that ignores its input and prints out the string $\langle M \rangle$ (where $\langle M \rangle$ is also a Turing machine description)
  2. Combine $\langle P_{\langle M \rangle} \rangle$ and $\langle M \rangle$ into a single machine M^*
  3. Print out $\langle M^* \rangle$ and halt
- $P_{\langle Q \rangle}$ ignores whatever input is on the tape and prints out the description of machine Q.

A self-reproducing program

Now we are ready to describe the machine SELF, a machine which prints out its own description. SELF consists of two sub-machines, $P_{\langle Q \rangle}$ and Q.

- Q takes as input $\langle M \rangle$, a Turing machine description, and does the following:
  1. Compute $\langle P_{\langle M \rangle} \rangle$, i.e. a description of a machine that ignores its input and prints out the string $\langle M \rangle$ (where $\langle M \rangle$ is also a Turing machine description)
  2. Combine $\langle P_{\langle M \rangle} \rangle$ and $\langle M \rangle$ into a single machine M^*
  3. Print out $\langle M^* \rangle$ and halt
- $P_{\langle Q \rangle}$ ignores whatever input is on the tape and prints out the description of machine Q.

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

A self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

1. Machine $P_{\langle Q \rangle}$ prints $\langle Q \rangle$ on the tape, and passes control to Q.

A self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

1. Machine $P_{\langle Q \rangle}$ prints $\langle Q \rangle$ on the tape, and passes control to Q.
2. Machine Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description, which was produced by $P_{\langle Q \rangle}$

A self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

1. Machine $P_{\langle Q \rangle}$ prints $\langle Q \rangle$ on the tape, and passes control to Q.
2. Machine Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description, which was produced by $P_{\langle Q \rangle}$
3. Machine Q constructs the machine P_w, which in this case is $P_{\langle Q \rangle}$.

A self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

1. Machine $P_{\langle Q \rangle}$ prints $\langle Q \rangle$ on the tape, and passes control to Q.
2. Machine Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description, which was produced by $P_{\langle Q \rangle}$
3. Machine Q constructs the machine P_w, which in this case is $P_{\langle Q \rangle}$.
4. Machine Q then combines the machines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into one machine M^*

A self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

1. Machine $P_{\langle Q \rangle}$ prints $\langle Q \rangle$ on the tape, and passes control to Q.
2. Machine Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description, which was produced by $P_{\langle Q \rangle}$
3. Machine Q constructs the machine P_w, which in this case is $P_{\langle Q \rangle}$.
4. Machine Q then combines the machines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into one machine M^*

Does that last step loop familiar?

A Self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

- SELF produces a machine $\langle M^* \rangle$ which is a combination of $P_{\langle Q \rangle}$ and Q

A Self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

- SELF produces a machine $\langle M^* \rangle$ which is a combination of $P_{\langle Q \rangle}$ and Q
- But SELF also of a combination of $P_{\langle Q \rangle}$ and Q!

A Self-reproducing program

SELF first runs $P_{\langle Q \rangle}$, and then runs Q. What happens?

- SELF produces a machine $\langle M^* \rangle$ which is a combination of $P_{\langle Q \rangle}$ and Q
- But SELF also of a combination of $P_{\langle Q \rangle}$ and Q!
- SELF has reproduced its own description!

A Self-reproducing program

SELF = ⟨$P_{\langle Q\rangle}$Q⟩

Initially $P_{\langle Q\rangle}$ starts with control of the tape, with an input string w

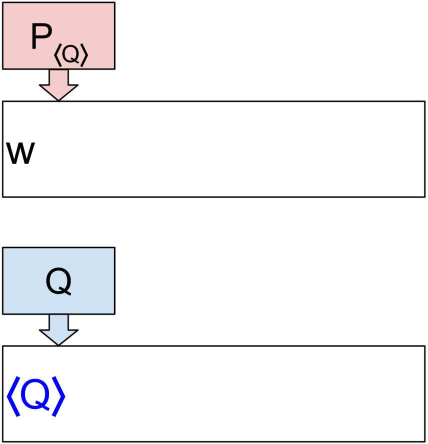
1. $P_{\langle Q\rangle}$ erases w and prints ⟨Q⟩ on the tape, then passes control to Q
2. Q reads ⟨Q⟩ on the tape. That is, it encounters its own description which was produced by $P_{\langle Q\rangle}$
3. Q reads ⟨Q⟩ and uses this to construct $P_{\langle Q\rangle}$
4. Q combines ⟨$P_{\langle Q\rangle}$⟩ and ⟨Q⟩ into a single machine ⟨M^*⟩

A Self-reproducing program

SELF = $\langle P_{\langle Q \rangle} Q \rangle$

Initially $P_{\langle Q \rangle}$ starts with control of the tape, with an input string w

1. **$P_{\langle Q \rangle}$ erases w and prints $\langle Q \rangle$ on the tape, then passes control to Q**
2. Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description which was produced by $P_{\langle Q \rangle}$
3. Q reads $\langle Q \rangle$ and uses this to construct $P_{\langle Q \rangle}$
4. Q combines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into a single machine $\langle M^* \rangle$



A Self-reproducing program

SELF = $\langle P_{\langle Q \rangle} Q \rangle$

Initially $P_{\langle Q \rangle}$ starts with control of the tape, with an input string w

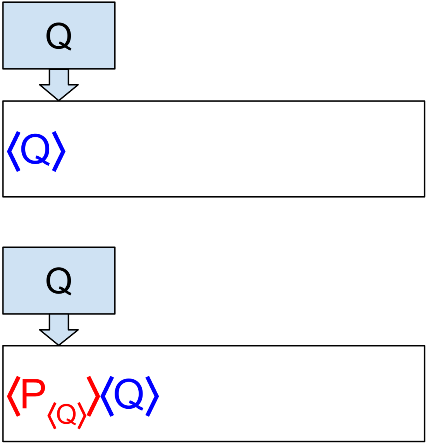
1. $P_{\langle Q \rangle}$ erases w and prints $\langle Q \rangle$ on the tape, then passes control to Q
2. **Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description which was produced by $P_{\langle Q \rangle}$**
3. Q reads $\langle Q \rangle$ and uses this to construct $P_{\langle Q \rangle}$
4. Q combines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into a single machine $\langle M^* \rangle$

A Self-reproducing program

$$SELF = \langle P_{\langle Q \rangle} Q \rangle$$

Initially $P_{\langle Q \rangle}$ starts with control of the tape, with an input string w

1. $P_{\langle Q \rangle}$ erases w and prints $\langle Q \rangle$ on the tape, then passes control to Q
2. Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description which was produced by $P_{\langle Q \rangle}$
3. **Q reads $\langle Q \rangle$ and uses this to construct $P_{\langle Q \rangle}$**
4. Q combines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into a single machine $\langle M^* \rangle$

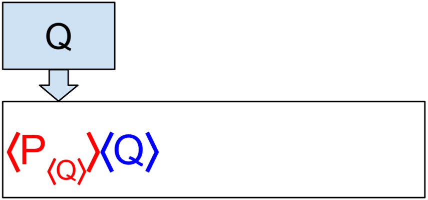


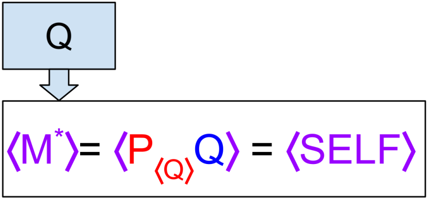
A Self-reproducing program

$\text{SELF} = \langle P_{\langle Q \rangle} Q \rangle$

Initially $P_{\langle Q \rangle}$ starts with control of the tape, with an input string w

1. $P_{\langle Q \rangle}$ erases w and prints $\langle Q \rangle$ on the tape, then passes control to Q
2. Q reads $\langle Q \rangle$ on the tape. That is, it encounters its own description which was produced by $P_{\langle Q \rangle}$
3. Q reads $\langle Q \rangle$ and uses this to construct $P_{\langle Q \rangle}$
4. **Q combines $\langle P_{\langle Q \rangle} \rangle$ and $\langle Q \rangle$ into a single machine $\langle M^{*} \rangle$**





Self-analyzing programs

Self-analyzing programs

- Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$.

Self-analyzing programs

- Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.
  - T represents a machine that receives a machine description as one of its inputs and analyzes that machine

Self-analyzing programs

- Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.
  - T represents a machine that receives a machine description as one of its inputs and analyzes that machine
- There is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every w

$$r(w) = t(\langle R \rangle, w)$$

Self-analyzing programs

- Let T be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$.
  - T represents a machine that receives a machine description as one of its inputs and analyzes that machine
- There is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$, where for every w
$$r(w) = t(\langle R \rangle, w)$$
  - R performs the same analysis as T does, but it analyzes its own description rather than taking a machine description as one of its inputs

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.
   - Instead of erasing whatever input it gets on the tape, it writes $\langle QT \rangle$ following its input w

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.
   - Instead of erasing whatever input it gets on the tape, it writes $\langle QT \rangle$ following its input w
2. Q reads $\langle QT \rangle$ on the tape and uses this to construct $P_{\langle QT \rangle}$.

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.
   - Instead of erasing whatever input it gets on the tape, it writes $\langle QT \rangle$ following its input w
2. Q reads $\langle QT \rangle$ on the tape and uses this to construct $P_{\langle QT \rangle}$.
3. Q then combines $\langle P_{\langle QT \rangle} \rangle$, $\langle Q \rangle$, and $\langle T \rangle$ into one machine $\langle M^* \rangle = \langle P_{\langle QT \rangle} QT \rangle = \langle R \rangle$

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.
   - Instead of erasing whatever input it gets on the tape, it writes $\langle QT \rangle$ following its input w
2. Q reads $\langle QT \rangle$ on the tape and uses this to construct $P_{\langle QT \rangle}$.
3. Q then combines $\langle P_{\langle QT \rangle} \rangle$, $\langle Q \rangle$, and $\langle T \rangle$ into one machine $\langle M^* \rangle = \langle P_{\langle QT \rangle} QT \rangle = \langle R \rangle$
4. Q writes $\langle R \rangle$ onto the tape (again, following w) and passes control to T

Self-analyzing programs

We construct R in three parts: $P_{\langle QT \rangle}, Q, T$

1. $P_{\langle QT \rangle}$ prints out the description $\langle QT \rangle$, a combination of machines Q and T.
   - Instead of erasing whatever input it gets on the tape, it writes $\langle QT \rangle$ following its input w
2. Q reads $\langle QT \rangle$ on the tape and uses this to construct $P_{\langle QT \rangle}$.
3. Q then combines $\langle P_{\langle QT \rangle} \rangle$, $\langle Q \rangle$, and $\langle T \rangle$ into one machine $\langle M^* \rangle = \langle P_{\langle QT \rangle} QT \rangle = \langle R \rangle$
4. Q writes $\langle R \rangle$ onto the tape (again, following w) and passes control to T
5. T reads w and $\langle R \rangle$ on. the tape and computes $t(\langle R \rangle, w)$

Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w

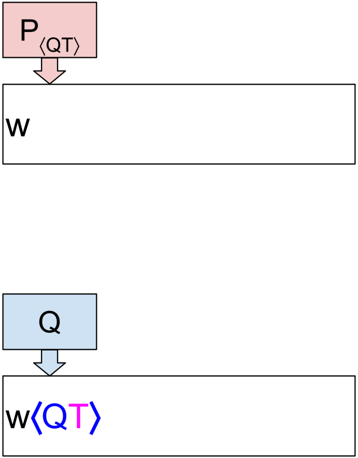
1. $P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q
2. Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$
3. Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$
4. Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T
5. T reads w and $\langle R \rangle$ on the tape and computes $t(w, \langle R \rangle)$

Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w

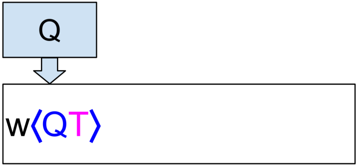
1. **$P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q**
2. Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$
3. Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$
4. Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T
5. T reads w and $\langle R \rangle$ on the tape and computes t(w, $\langle R \rangle$)



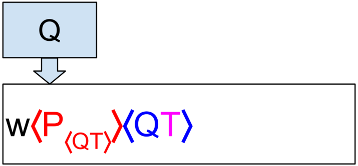
Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w



1. $P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q
2. **Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$**
3. Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$
4. Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T
5. T reads w and $\langle R \rangle$ on the tape and computes t(w, $\langle R \rangle$)

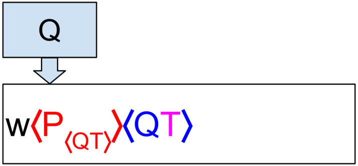


Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w

1. $P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q
2. Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$
3. **Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$**
4. Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T
5. T reads w and $\langle R \rangle$ on the tape and computes $t(w, \langle R \rangle)$



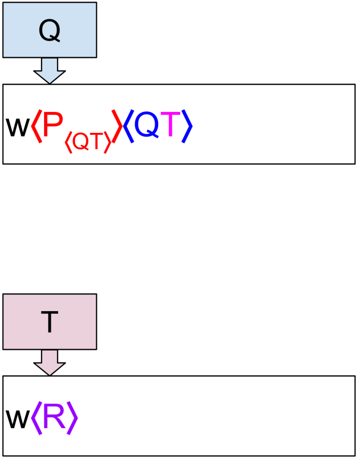
$\langle M^* \rangle = \langle P_{\langle Q \rangle} QT \rangle = \langle R \rangle$

Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w

1. $P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q
2. Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$
3. Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$
4. **Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T**
5. T reads w and $\langle R \rangle$ on the tape and computes $t(w, \langle R \rangle)$

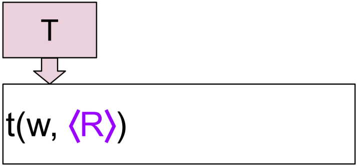


Self-analyzing programs

$R = \langle P_{\langle QT \rangle} QT \rangle$

Initially $P_{\langle QT \rangle}$ starts with control of the tape, with an input string w

1. $P_{\langle QT \rangle}$ prints $\langle QT \rangle$ on the tape (following the original input w), and then passes control to Q
2. Q reads $\langle QT \rangle$ and uses it to construct $P_{\langle QT \rangle}$
3. Q combines $P_{\langle QT \rangle}$, $\langle Q \rangle$, and $\langle T \rangle$ into a single machine $\langle M^* \rangle$
4. Q writes $\langle R \rangle$ onto the tape (following w) and passes control to T
5. **T reads w and $\langle R \rangle$ on the tape and computes t(w, $\langle R \rangle$)**



Self-analyzing programs

Self-analyzing programs

- When constructing a Turing machine M, you can include the command "Obtain M's description $\langle M \rangle$" as part of the pseudocode

Self-analyzing programs

- When constructing a Turing machine M, you can include the command "Obtain M's description $\langle M \rangle$" as part of the pseudocode
- Whatever you were going to do with $\langle M \rangle$, the recursion theorem tells us that there exists a machine that finds a way to put its own description on the tape before processing that description

HALT is undecidable: another proof

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run H on $\langle M, w\rangle$

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run H on $\langle M, w\rangle$
4. "Do the opposite"

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run H on $\langle M, w\rangle$
4. "Do the opposite"
    - If H accepts $\langle M, w\rangle$, go into a loop

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run H on $\langle M, w\rangle$
4. "Do the opposite"
    - If H accepts $\langle M, w\rangle$, go into a loop
    - If H rejects $\langle M, w\rangle$, immediately halt

HALT is undecidable: another proof

Theorem: $\mathrm{HALT} = \{\langle M, w\rangle | M \text{ halts on } w\}$ is undecidable

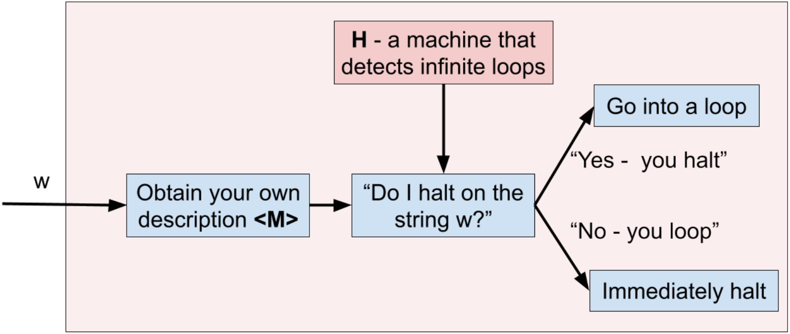
Proof: AFSOC machine H decides HALT. Create the following machine M:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run H on $\langle M, w\rangle$
4. "Do the opposite"
   - If H accepts $\langle M, w\rangle$, go into a loop
   - If H rejects $\langle M, w\rangle$, immediately halt

M halts on input w if H says it should loop, and loops if H says it should halt.

HALT is undecidable: another proof

HALT = {<M, w> | M halts on w}
A paradoxical machine **M**



If we can decide HALT, we create a paradoxical machine

Undecidability proofs using the recursion theorem

Undecidability proofs using the recursion theorem

We can use the following recipe to prove that a language L is undecidable

Undecidability proofs using the recursion theorem

We can use the following recipe to prove that a language L is undecidable

1. AFSOC a machine D can decide L

Undecidability proofs using the recursion theorem

We can use the following recipe to prove that a language L is undecidable

1. AFSOC a machine D can decide L
2. Create a machine M that obtains its own description, uses D to analyze itself, and "does the opposite" of what it should

Undecidability proofs using the recursion theorem

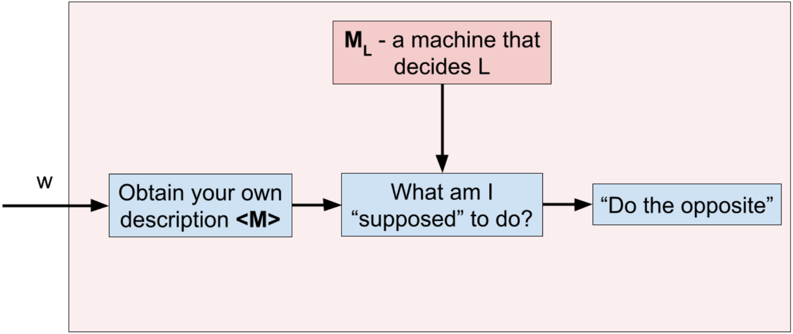
We can use the following recipe to prove that a language L is undecidable

1. AFSOC a machine D can decide L
2. Create a machine M that obtains its own description, uses D to analyze itself, and "does the opposite" of what it should
3. Conclude that D is not deciding L correctly

Recursion theorem recipe

L = an undecidable language

A paradoxical machine **M**



If we can decide L, we create a paradoxical machine

A_{TM} is undecidable: an alternate proof

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

A_{TM} is undecidable: an alternate proof

Theorem: $\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Let's try proving this using the recursion theorem

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$

A_{TM} is undecidable: an alternate proof

Theorem: $\mathrm{A}_{\mathrm{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run A on $\langle M, w\rangle$

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run A on $\langle M, w\rangle$
4. "Do the opposite"

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run A on $\langle M, w\rangle$
4. "Do the opposite"
   - If A accepts $\langle M, w\rangle$, reject

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run A on $\langle M, w\rangle$
4. "Do the opposite"
   - If A accepts $\langle M, w\rangle$, reject
   - If A rejects $\langle M, w\rangle$, accept

$\mathrm{A_{TM}}$ is undecidable: an alternate proof

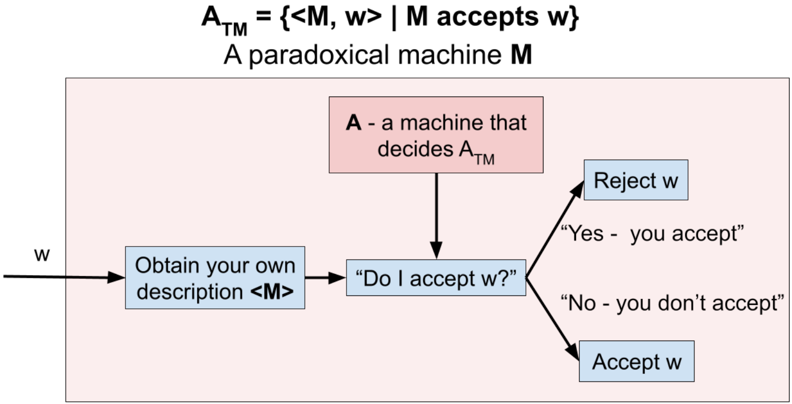
Theorem: $\mathrm{A_{TM}} = \{\langle M, w\rangle | w \in L(M)\}$ is undecidable

Proof: AFSOC machine A decides ATM. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run A on $\langle M, w\rangle$
4. "Do the opposite"
   - If A accepts $\langle M, w\rangle$, reject
   - If A rejects $\langle M, w\rangle$, accept

M accepts w if A says it should reject, and rejects if A says it should accept.

A_{TM} is undecidable: an alternate proof



If we can decide A_{TM}, we create a paradoxical machine

The language $\mathrm{REG}_{\mathrm{TM}}$

The language $\mathrm{REG}_{\mathrm{TM}}$

Consider the following language:

$$\mathrm{REG}_{\mathrm{TM}} = \{\langle M\rangle | L(M) \text{ is regular}\}$$

The language $\mathrm{REG}_{\mathrm{TM}}$

Consider the following language:

$$\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$$

- We receive a TM description $\langle M \rangle$ as input

The language $\mathrm{REG}_{\mathrm{TM}}$

Consider the following language:

$$\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$$

- We receive a TM description $\langle M \rangle$ as input
- We seek to determine if M recognizes a regular language

The language $\mathrm{REG}_{\mathrm{TM}}$

Consider the following language:

$$\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$$

- We receive a TM description $\langle M \rangle$ as input
- We seek to determine if M recognizes a regular language
  - "Can this TM be converted to a DFA?"

$\mathrm{REG_{TM}}$ is undecidable

Theorem: $\mathrm{REG_{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

$\mathrm{REG_{TM}}$ is undecidable

Theorem: $\mathrm{REG_{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

- Hint 1: Use the recursion theorem

$\mathrm{REG_{TM}}$ is undecidable

Theorem: $\mathrm{REG_{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

- Hint 1: Use the recursion theorem
- Hint 2: Make a machine that is regular when it shouldn't be, and vice-versa

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG}_{\mathrm{TM}}$.
Construct a machine M that does the following:

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG}_{\mathrm{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input

$\mathrm{REG_{TM}}$ is undecidable

Theorem: $\mathrm{REG_{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG_{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG}_{\mathrm{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$
3. Run R on $\langle M \rangle$

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG}_{\mathrm{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$
3. Run R on $\langle M \rangle$
4. "Do the opposite"

$\mathrm{REG_{TM}}$ is undecidable

Theorem: $\mathrm{REG_{TM}} = \{\langle M \rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG_{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$
3. Run R on $\langle M \rangle$
4. "Do the opposite"
   - If R accepts $\langle M \rangle$, simulate a machine that recognizes $0^n 1^n$

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

Theorem: $\mathrm{REG}_{\mathrm{TM}} = \{\langle M\rangle | L(M) \text{ is regular}\}$ is undecidable

Proof: AFSOC machine R decides $\mathrm{REG}_{\mathrm{TM}}$.
Construct a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M\rangle$
3. Run R on $\langle M\rangle$
4. "Do the opposite"
   - If R accepts $\langle M\rangle$, simulate a machine that recognizes 0^n1^n
   - If R accepts $\langle M\rangle$, simulate a machine that recognizes 0^*1^*

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

- If R says M is regular, M recognizes 0^n1^n - a non-regular language.

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

- If R says M is regular, M recognizes 0^n1^n - a non-regular language.
- If M says A is not regular, it recognizes 0^*1^* - a regular language.

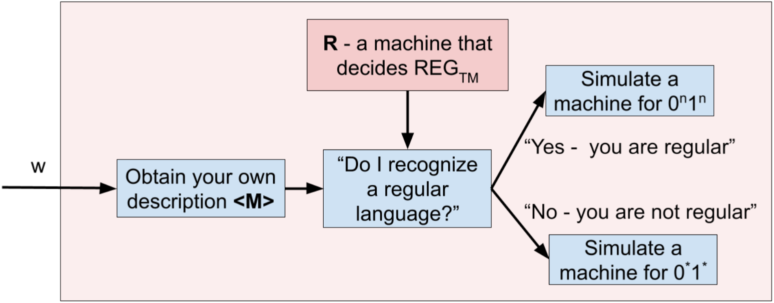
$\mathrm{REG_{TM}}$ is undecidable

- If R says M is regular, M recognizes 0^n1^n - a non-regular language.
- If M says A is not regular, it recognizes 0^*1^* - a regular language.
- Thus, R is not deciding $\mathrm{REG_{TM}}$ correctly

$\mathrm{REG}_{\mathrm{TM}}$ is undecidable

$\mathrm{REG}_{\mathrm{TM}}$ = {<M> | L(M) is regular}

A paradoxical machine **M**



If we can decide $\mathrm{REG}_{\mathrm{TM}}$, we create a paradoxical machine

Minimal Turing machines

Minimal Turing machines

- Let M be a Turing machine, with string description $\langle M \rangle$.

Minimal Turing machines

- Let M be a Turing machine, with string description $\langle M \rangle$.
- We say M is **minimal** if there is no Turing machine M_2 such that:

Minimal Turing machines

- Let M be a Turing machine, with string description $\langle M \rangle$.
- We say M is **minimal** if there is no Turing machine M_2 such that:
  1. $L(M) = L(M_2)$

Minimal Turing machines

- Let M be a Turing machine, with string description $\langle M \rangle$.
- We say M is **minimal** if there is no Turing machine M_2 such that:
  1. $L(M) = L(M_2)$
  2. $|\langle M_2 \rangle| < |\langle M \rangle|$

Minimal Turing machines

- Let M be a Turing machine, with string description $\langle M \rangle$.
- We say M is **minimal** if there is no Turing machine M_2 such that:
  1. $L(M) = L(M_2)$
  2. $|\langle M_2 \rangle| < |\langle M \rangle|$

i.e., there is no machine with a shorter description than M that recognizes the same language.

The language $\mathrm{MIN}_{\mathrm{TM}}$

The language $\mathrm{MIN}_{\mathrm{TM}}$

$$\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$$

The language $\mathrm{MIN}_{\mathrm{TM}}$

$$\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$$

- We receive a TM description/source code $\langle M \rangle$

The language $\mathrm{MIN}_{\mathrm{TM}}$

$$\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$$

- We receive a TM description/source code $\langle M \rangle$
- We want to know if M is minimal

The language $\mathrm{MIN}_{\mathrm{TM}}$

$$\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$$

- We receive a TM description/source code $\langle M \rangle$
- We want to know if M is minimal
  - "Can this source code be rewritten more concisely?"

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M$ is a minimal TM$\}$ is not RE

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$ is not RE

Proof: AFSOC some enumerator E enumerates $\mathrm{MIN}_{\mathrm{TM}}$. Create a machine M that does the following:

$\mathrm{MIN_{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN_{TM}} = \{\langle M\rangle | M \text{ is a minimal TM}\}$ is not RE

Proof: AFSOC some enumerator E enumerates $\mathrm{MIN_{TM}}$. Create a machine M that does the following:

1. M takes a string w as input

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$ is not RE

Proof: AFSOC some enumerator E enumerates $\mathrm{MIN}_{\mathrm{TM}}$. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$ is not RE

Proof: AFSOC some enumerator E enumerates $\mathrm{MIN}_{\mathrm{TM}}$. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$
3. Run E until it lists a machine $\langle M_2 \rangle$ such that $|\langle M \rangle| < |\langle M_2 \rangle|$ (i.e. a machine with a longer description)

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

Theorem: $\mathrm{MIN}_{\mathrm{TM}} = \{\langle M \rangle | M \text{ is a minimal TM}\}$ is not RE

Proof: AFSOC some enumerator E enumerates $\mathrm{MIN}_{\mathrm{TM}}$. Create a machine M that does the following:

1. M takes a string w as input
2. Obtain its own description $\langle M \rangle$
3. Run E until it lists a machine $\langle M_2 \rangle$ such that $|\langle M \rangle| < |\langle M_2 \rangle|$ (i.e. a machine with a longer description)
4. Simulate M_2 on w

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

- Note that $L(M) = L(M_2)$, and $|\langle M \rangle| < |\langle M_2 \rangle|$

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

- Note that $L(M) = L(M_2)$, and $|\langle M \rangle| < |\langle M_2 \rangle|$
- But this is not supposed to be possible since M_2 is supposed to be a minimal TM.

$\mathrm{MIN}_{\mathrm{TM}}$ is not recursively enumerable

- Note that $L(M) = L(M_2)$, and $|\langle M \rangle| < |\langle M_2 \rangle|$
- But this is not supposed to be possible since M_2 is supposed to be a minimal TM.
- Thus, our enumerator did not enumerate $\mathrm{MIN}_{\mathrm{TM}}$ correctly.