Universal Turing Machines and Undecidability

Nabil Mustafa

Computational Complexity

Turing machine recap

A Turing machine **TM** is a tuple $M = (\Gamma, Q, \delta)$ where

- Γ: set of symbols that **TM** 's tapes can contain.

Turing machine recap

A Turing machine **TM** is a tuple $M = (\Gamma, Q, \delta)$ where

- Γ: set of symbols that **TM** 's tapes can contain.
- Q: possible states **TM** can be in.
  - q_{start}: the **TM** starts in this state
  - q_{halt}: the **TM** halts when this state is reached

Turing machine recap

A Turing machine **TM** is a tuple $M = (\Gamma, Q, \delta)$ where

- Γ: set of symbols that **TM** 's tapes can contain.
- Q: possible states **TM** can be in.
  - q_{start}: the **TM** starts in this state
  - q_{halt}: the **TM** halts when this state is reached

Storage for **TM** :

- A special register stores the current state.

Turing machine recap

A Turing machine **TM** is a tuple $M = (\Gamma, Q, \delta)$ where

- Γ: set of symbols that **TM** 's tapes can contain.
- Q: possible states **TM** can be in.
  - q_{start}: the **TM** starts in this state
  - q_{halt}: the **TM** halts when this state is reached

Storage for **TM** :

- A special register stores the current state.
- 1 input tape, 1 output tape and 1 work tape

Robustness of **TM**

- Many details of our **TM** quite arbitrary

Robustness of TM

- Many details of our **TM** quite arbitrary
- Does restricting the alphabet Γ to $\{0, 1, \Box, \triangleright\}$ matter?

Robustness of **TM**

- Many details of our **TM** quite arbitrary
- Does restricting the alphabet Γ to $\{0, 1, \square, \triangleright\}$ matter?

No!

If function f is computable by a **TM** using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** using the alphabet $\{0, 1, \square, \triangleright\}$.

Robustness of **TM**

- Many details of our **TM** quite arbitrary
- Does restricting the alphabet Γ to $\{0, 1, \Box, \triangleright\}$ matter?

No!

If function f is computable by a **TM** using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** using the alphabet $\{0, 1, \Box, \triangleright\}$.

- Does using more work tapes make everything much faster?

Robustness of **TM**

- Many details of our **TM** quite arbitrary
- Does restricting the alphabet Γ to $\{0, 1, \square, \triangleright\}$ matter?

No!

If function f is computable by a **TM** using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** using the alphabet $\{0, 1, \square, \triangleright\}$.

- Does using more work tapes make everything much faster?

No!

If function f is computable by a **TM** using k tapes, then it is computable in time $5kT(n)^2$ by a **TM** using a single work tape.

Robustness of **TM**

- Many details of our **TM** quite arbitrary
- Does restricting the alphabet Γ to $\{0, 1, \Box, \rhd\}$ matter?

No!

If function f is computable by a **TM** using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** using the alphabet $\{0, 1, \Box, \rhd\}$.

- Does using more work tapes make everything much faster?

No!

If function f is computable by a **TM** using k tapes, then it is computable in time $5kT(n)^2$ by a **TM** using a single work tape.

Church-Turing Hypothesis

Every physically realizable computation device can be simulated by a **TM** .

Alphabet size doesn't matter

Claim

*If function f is computable by a **TM** M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** M' using the alphabet $\{0, 1, \Box, \rhd\}$.*

Alphabet size doesn't matter

Claim

If function f is computable by a ***TM*** *M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a* ***TM*** *M' using the alphabet $\{0, 1, \square, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \square, \triangleright\}$.

Alphabet size doesn't matter

Claim

If function f is computable by a ***TM*** *M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a* ***TM*** *M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.

- How to simulate each step of M.

Alphabet size doesn't matter

Claim

*If function f is computable by a **TM** M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by bits.
- How to simulate each step of M.

Alphabet size doesn't matter

Claim

If function f is computable by a ***TM*** *M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a* ***TM*** *M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by $\log |\Gamma|$ bits.
- How to simulate each step of M.

Alphabet size doesn't matter

Claim

*If function f is computable by a **TM** M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** M' using the alphabet $\{0, 1, \square, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \square, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by $\log |\Gamma|$ bits.
- How to simulate each step of M.
  - Read the encoded symbol via $\log |\Gamma|$ bits.

Alphabet size doesn't matter

Claim

If function f is computable by a ***TM*** *M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a* ***TM*** *M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by $\log |\Gamma|$ bits.
- How to simulate each step of M.
  - Read the encoded symbol via $\log |\Gamma|$ bits. Problem: remember the bits read.

Alphabet size doesn't matter

Claim

*If function f is computable by a **TM** M using alphabet Γ, then it is computable in time $4\log|\Gamma| \cdot T(n)$ by a **TM** M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by $\log|\Gamma|$ bits.
- How to simulate each step of M.
  - Read the encoded symbol via $\log|\Gamma|$ bits. Problem: remember the bits read.
  - Modify the transition table appropriately

Alphabet size doesn't matter

Claim

*If function f is computable by a **TM** M using alphabet Γ, then it is computable in time $4 \log |\Gamma| \cdot T(n)$ by a **TM** M' using the alphabet $\{0, 1, \Box, \triangleright\}$.*

Have to decide:

- How to represent each symbol of Γ using $\{0, 1, \Box, \triangleright\}$.
  - Use binary encoding. Each symbol encoded by $\log |\Gamma|$ bits.
- How to simulate each step of M.
  - Read the encoded symbol via $\log |\Gamma|$ bits. Problem: remember the bits read.
  - Modify the transition table appropriately
  - Lookup the (remembered) read bits in the (modified) table

Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

n	x	y	i	b	j	3

Remembering bits read

Problem: How to remember the bits read?

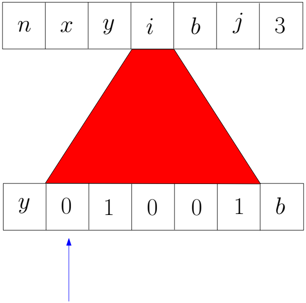
$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

n	x	y	j	b	j	3

Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

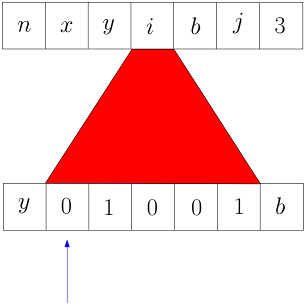


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	01001	q'	01010	R

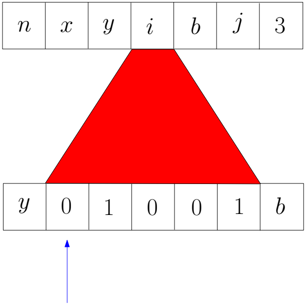


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	0	q_0	0	R

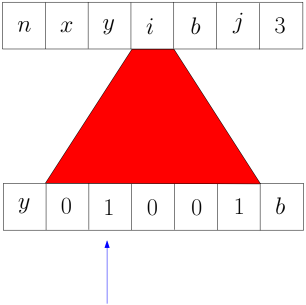


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	0	q_0	0	R
q_0	1	q_{01}	1	R

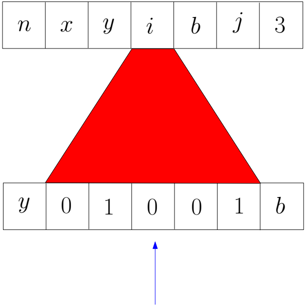


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R

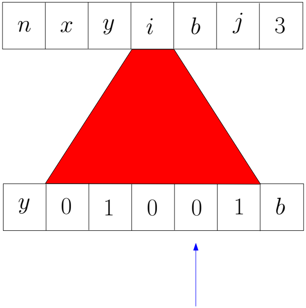


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R

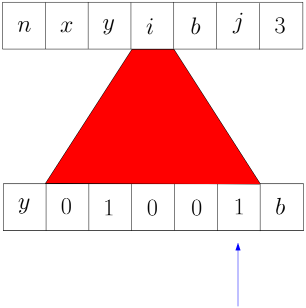


Remembering bits read

Problem: How to remember the bits read?

$\mathbf{Q}$	Γ	$\mathbf{Q}$	Γ	{L,R,S}
q	i	q'	j	R

q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	q'	1	R



Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L

Lets say that $i \to 01001$ and $j \to 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L

Lets say that $i \to 01001$ and $j \to 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L
$qb_{4.01001}$	$-$	$qw_{1.01001}$	$-$	S

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L
$qb_{4.01001}$	$-$	$qw_{1.01001}$	$-$	S
$qw_{1.01001}$	$-$	$qw_{2.01001}$	0	R

Lets say that $i \rightarrow 01001$ and $j \rightarrow 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L
$qb_{4.01001}$	$-$	$qw_{1.01001}$	$-$	S
$qw_{1.01001}$	$-$	$qw_{2.01001}$	0	R
$qw_{2.01001}$	$-$	$qw_{3.01001}$	1	R

Lets say that $i \to 01001$ and $j \to 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	–	$qb_{2.01001}$	–	L
$qb_{2.01001}$	–	$qb_{3.01001}$	–	L
$qb_{3.01001}$	–	$qb_{4.01001}$	–	L
$qb_{4.01001}$	–	$qw_{1.01001}$	–	S
$qw_{1.01001}$	–	$qw_{2.01001}$	0	R
$qw_{2.01001}$	–	$qw_{3.01001}$	1	R
$qw_{3.01001}$	–	$qw_{4.01001}$	0	R

Lets say that $i \to 01001$ and $j \to 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L
$qb_{4.01001}$	$-$	$qw_{1.01001}$	$-$	S
$qw_{1.01001}$	$-$	$qw_{2.01001}$	0	R
$qw_{2.01001}$	$-$	$qw_{3.01001}$	1	R
$qw_{3.01001}$	$-$	$qw_{4.01001}$	0	R
$qw_{4.01001}$	$-$	$qw_{5.01001}$	1	R

Lets say that $i \to 01001$ and $j \to 01010$. Then,

Q	Γ	Q	Γ	$\{L, S, R\}$
q	0	q_0	0	R
q_0	1	q_{01}	1	R
q_{01}	0	q_{010}	0	R
q_{010}	0	q_{0100}	0	R
q_{0100}	1	$qb_{1.01001}$	1	L
$qb_{1.01001}$	$-$	$qb_{2.01001}$	$-$	L
$qb_{2.01001}$	$-$	$qb_{3.01001}$	$-$	L
$qb_{3.01001}$	$-$	$qb_{4.01001}$	$-$	L
$qb_{4.01001}$	$-$	$qw_{1.01001}$	$-$	S
$qw_{1.01001}$	$-$	$qw_{2.01001}$	0	R
$qw_{2.01001}$	$-$	$qw_{3.01001}$	1	R
$qw_{3.01001}$	$-$	$qw_{4.01001}$	0	R
$qw_{4.01001}$	$-$	$qw_{5.01001}$	1	R
$qw_{5.01001}$	$-$	q'	0	R

Note: need to move back $2 \log |\Gamma|$ steps as well!

Total time required

What is the number of states in M'?

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.
- What is the size of the old transition table?

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.

- What is the size of the old transition table?
  - $|Q| \cdot |\Gamma|$

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.
- What is the size of the old transition table?
  - $|Q| \cdot |\Gamma|$
- Total number of new states required: $|Q||\Gamma| 4 \log |\Gamma|$.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.
- What is the size of the old transition table?
  - $|Q| \cdot |\Gamma|$
- Total number of new states required: $|Q||\Gamma| 4 \log |\Gamma|$.
- Each step of M requires $5 \log |\Gamma|$ steps of M'.

Total time required

What is the number of states in M'?

- Each line in the transition table of M creates $4 \log |\Gamma|$ states.
  - $\log |\Gamma|$ steps to read the symbol encoding.
  - $\log |\Gamma|$ steps to go back to the start of the symbol encoding.
  - $\log |\Gamma|$ steps to write the new symbol encoding.
  - $2 \log |\Gamma|$ in case the tape-head moves to the left.

- What is the size of the old transition table?
  - $|Q| \cdot |\Gamma|$

- Total number of new states required: $|Q||\Gamma| 4 \log |\Gamma|$.

- Each step of M requires $5 \log |\Gamma|$ steps of M'.
- Total time taken: $5 \log |\Gamma| T(n)$ steps.

More work tapes do not make much difference

Claim

If a function f is computable by a ***TM*** *M using k tapes, then its computable in time $5kT(n)^2$ by a* ***TM*** *using a single work tape.*

More work tapes do not make much difference

Claim

If a function f is computable by a ***TM*** *M using k tapes, then its computable in time $5kT(n)^2$ by a* ***TM*** *using a single work tape.*

Proof

- Lets consider that a **TM** M computes the function f and has k tapes (plus additional input and output tapes)
- Next we consider a single work tape Turing machine $\hat{M}$
- $\hat{M}$ encodes the k tapes of M on a single tape by using locations $1, k+1, 2k+1, \ldots$ to encode the first tape, locations $2, k+2, 2k+2, \ldots$ to encode the second tape etc.
- For every symbol a in M's alphabet, $\hat{M}$ will contain both the symbol a and the symbol $\hat{a}$. In the encoding of each tape, exactly one symbol will be of the '^ type', indicating that the corresponding head of M is positioned in that location.

More work tapes do not make much difference

Proof Cont'd

- To simulate one step of M, the machine makes two passes of its work tape: first it traverses the tape in the left-to-right direction and records (via additional states) the k symbols of the form $\hat{a}$
- Then $\hat{M}$ uses M's transition function to determine the new state, symbols, and head movements and sweeps the tape back in the right-to-left direction to update the encoding accordingly.
- $\hat{M}$ will never reach more than location $kT(n)$ of its work tape, meaning that for each of the at most $T(n)$ steps of M, $\hat{M}$ performs at most $5kT(n)$ work (sweeping back and forth requires about $2T(n)$ steps, and some additional steps needed for updating head movement and book keeping).

Universal Turing Machines

- So far, constructed Turing machines for a specific task.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?
  - Given a 'program' **P** and an input x for **P**, run **P** on x.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?
  - Given a 'program' **P** and an input x for **P**, run **P** on x.
  - Note that a program is just an algorithm for doing something, i.e., its a Turing machine.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?
  - Given a 'program' **P** and an input x for **P**, run **P** on x.
  - Note that a program is just an algorithm for doing something, i.e., its a Turing machine.
- So . . . given *any* **TM** M and x, does there exist a **TM** UTM that will run M on x

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?
  - Given a 'program' **P** and an input x for **P**, run **P** on x.
  - Note that a program is just an algorithm for doing something, i.e., its a Turing machine.
- So ... given *any* **TM** M and x, does there exist a **TM** UTM that will run M on x
  - How to give M to UTM? Since M is a machine.

Universal Turing Machines

- So far, constructed Turing machines for a specific task.
  - a primitive computer that only performs one task.
- Not very useful, since there are many things we want to do.
- Question: does there exist a **TM** that does many things?
  - Given a 'program' **P** and an input x for **P**, run **P** on x.
  - Note that a program is just an algorithm for doing something, i.e., its a Turing machine.
- So . . . given *any* **TM** M and x, does there exist a **TM** UTM that will run M on x
  - How to give M to UTM? Since M is a machine.
  - How can there be a fixed machine UTM that will run *any* other **TM** ?

What is a Turing machine really

- Observe that the **TM** s are not all that different

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \Box, \triangleright\}$

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \Box, \triangleright\}$
- The only thing different between different Turing machines is the transition table

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \Box, \triangleright\}$
- The only thing different between different Turing machines is the transition table
- But the transition table can be written up as a string of bits!

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \Box, \triangleright\}$
- The only thing different between different Turing machines is the transition table
- But the transition table can be written up as a string of bits!
- M_α: **TM** represented by string of bits α

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \Box, \triangleright\}$
- The only thing different between different Turing machines is the transition table
- But the transition table can be written up as a string of bits!
- M_α: **TM** represented by string of bits α
- $\lfloor M \rfloor \in \{0, 1\}^*$: string of bits representing M

What is a Turing machine really

- Observe that the **TM** s are not all that different
  - Can assume all have 1 input tape, 1 output tape, 1 work tape.
  - Can assume $\Gamma = \{0, 1, \square, \triangleright\}$
- The only thing different between different Turing machines is the transition table
- But the transition table can be written up as a string of bits!
- M_α: **TM** represented by string of bits α
- $\lfloor M \rfloor \in \{0, 1\}^*$: string of bits representing M
- So UTM takes α and x, and simulates M_α on x.

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:
  - Read the symbol of the input x at the tape-head.

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:
    - Read the symbol of the input x at the tape-head.
    - Scan the transition table to find a rule which applies.

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:
  - Read the symbol of the input x at the tape-head.
  - Scan the transition table to find a rule which applies.
  - Apply this rule to write new symbol and move the tape head.

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:
  - Read the symbol of the input x at the tape-head.
  - Scan the transition table to find a rule which applies.
  - Apply this rule to write new symbol and move the tape head.
  - Store the new state of M

The Algorithm for **TM**

How can a fixed Turing machine *UTM* run *any* other **TM** ?

- Basic idea: Running a **TM** is also just an algorithm!
- For example, we ran the **TM** for $PAL(x)$ by hand.
- We just have to follow some rules, i.e., there is a simple algorithm for running a **TM** M:
  - Read the symbol of the input x at the tape-head.
  - Scan the transition table to find a rule which applies.
  - Apply this rule to write new symbol and move the tape head.
  - Store the new state of M
- So we just have to design a **TM** that can run the above algorithm.

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Transition function of *UTM*:

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Transition function of *UTM*:

- Scan T_5 to find a match of the state with T_3

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Transition function of *UTM*:

- Scan T_5 to find a match of the state with T_3
- Scan T_5 to find a match of symbol with T_1 and T_4.

Universal Turing Machine

Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Transition function of *UTM*:

- Scan T_5 to find a match of the state with T_3
- Scan T_5 to find a match of symbol with T_1 and T_4.
- Copy new state to T_3

Universal Turing Machine

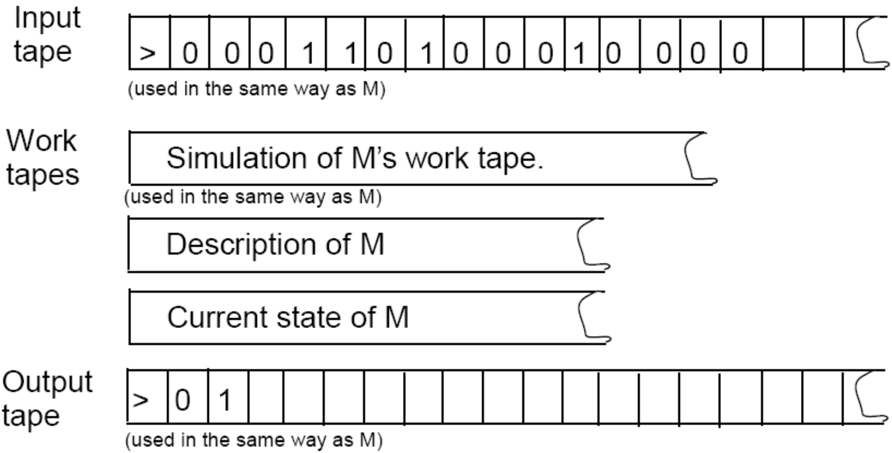
Storage:

- T_1: input tape containing α, x.
- T_2: output tape
- Three work tapes:
  - T_3 contains the current state of M_α
  - T_4 is used the same way as M_α's work-tape.
  - T_5 contains the transition function.

Transition function of *UTM*:

- Scan T_5 to find a match of the state with T_3
- Scan T_5 to find a match of symbol with T_1 and T_4.
- Copy new state to T_3
- Copy new symbol to T_4 and move heads of T_1 and T_3.

Simulating M_α



A Quote

"If we were so clever that we could understand our brain, our brain would be so complex that we couldn't understand it."

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \ \ \forall x$

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \ \ \forall x$
- It would seem that any function $f : \{0,1\}^* \to \{0,1\}$ can be computed by a Turing machine.

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \ \ \forall x$
- It would seem that any function $f : \{0,1\}^* \rightarrow \{0,1\}$ can be computed by a Turing machine.
  - We don't care *how* much time is taken

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \;\; \forall x$
- It would seem that any function $f : \{0,1\}^* \rightarrow \{0,1\}$ can be computed by a Turing machine.
  - We don't care *how* much time is taken
  - So just spend as much time till the computation of the function finishes.

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \ \ \forall x$
- It would seem that any function $f : \{0,1\}^* \to \{0,1\}$ can be computed by a Turing machine.
  - We don't care *how* much time is taken
  - So just spend as much time till the computation of the function finishes.

Uncomputable functions exist

There exists $UC : \{0,1\}^* \to \{0,1\}$ not computable by any **TM** .

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \ \ \forall x$
- It would seem that any function $f : \{0,1\}^* \rightarrow \{0,1\}$ can be computed by a Turing machine.
  - We don't care *how* much time is taken
  - So just spend as much time till the computation of the function finishes.

Uncomputable functions exist

There exists $UC : \{0,1\}^* \rightarrow \{0,1\}$ not computable by any **TM** .

- **HALT**(P, x): Given program P and x, does P halt on x?

Uncomputable functions

- f is computable by a **TM** M if $M(x) = f(x) \;\; \forall x$
- It would seem that any function $f : \{0,1\}^* \rightarrow \{0,1\}$ can be computed by a Turing machine.
  - We don't care *how* much time is taken
  - So just spend as much time till the computation of the function finishes.

Uncomputable functions exist

There exists $UC : \{0,1\}^* \rightarrow \{0,1\}$ not computable by any **TM** .

- **HALT**(P, x): Given program P and x, does P halt on x?

Halting function

HALT is not computable by any **TM** .

A Detour

- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets

A Detour

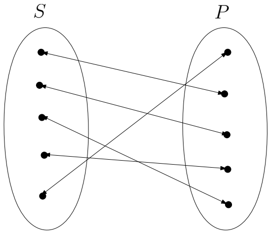
- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?

A Detour

- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \rightarrow P$ **iff** $|S| = |P|$

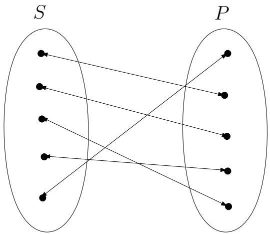
A Detour

- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \rightarrow P$ **iff** $|S| = |P|$



A Detour

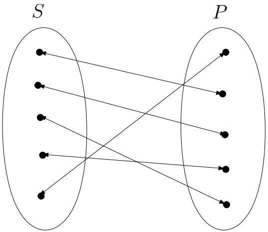
- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \to P$ **iff** $|S| = |P|$



- What if S and P are infinite sets?

A Detour

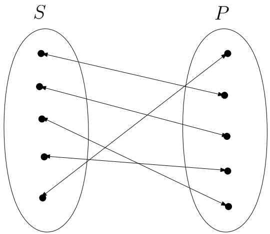
- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \to P$ **iff** $|S| = |P|$



- What if S and P are infinite sets?
- Any bijection between $S = \{1, 2, \ldots\}$ and $P = \{2, 3, \ldots\}$?

A Detour

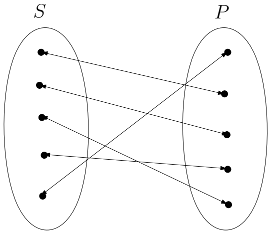
- Let $S = \{s_1, \dots, s_n\}$ and $P = \{p_1, \dots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \to P$ **iff** $|S| = |P|$



- What if S and P are infinite sets?
- Any bijection between $S = \{1, 2, \dots\}$ and $P = \{2, 3, \dots\}$?
  - $f : S \to P$ defined as: $f(a) = a + 1$.

A Detour

- Let $S = \{s_1, \ldots, s_n\}$ and $P = \{p_1, \ldots, p_m\}$ be two finite sets
- Question: When does a bijection between S and P exist?
  - Answer: $\exists$ a bijection $f : S \rightarrow P$ **iff** $|S| = |P|$



- What if S and P are infinite sets?
- Any bijection between $S = \{1, 2, \ldots\}$ and $P = \{2, 3, \ldots\}$?
  - $f : S \rightarrow P$ defined as: $f(a) = a + 1$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 3$, $3 \leftrightarrow 4$ and so on.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?

A Detour

- Bijection between $S = \{1, 2, 3, \dots\}$ and $P = \{2, 4, 6, \dots\}$?
  - $f : S \longrightarrow P$ defined as: $f(a) = 2a$.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?
  - $f : S \longrightarrow P$ defined as: $f(a) = 2a$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$ and so on.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?
  - $f : S \rightarrow P$ defined as: $f(a) = 2a$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$ and so on.
  - Looks weird, but make sure you completely understand this.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?
  - $f : S \rightarrow P$ defined as: $f(a) = 2a$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$ and so on.
  - Looks weird, but make sure you completely understand this.

Theorem

A bijection exists between natural numbers $\mathbb{N}$ and rationals $\mathbb{Q}$.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?
  - $f : S \rightarrow P$ defined as: $f(a) = 2a$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$ and so on.
  - Looks weird, but make sure you completely understand this.

Theorem

A bijection exists between natural numbers $\mathbb{N}$ and rationals $\mathbb{Q}$.

Theorem

No bijection exists between natural numbers $\mathbb{N}$ and reals $\mathbb{R}$.

A Detour

- Bijection between $S = \{1, 2, 3, \ldots\}$ and $P = \{2, 4, 6, \ldots\}$?
  - $f : S \to P$ defined as: $f(a) = 2a$.
  - $1 \leftrightarrow 2$, $2 \leftrightarrow 4$, $3 \leftrightarrow 6$ and so on.
  - Looks weird, but make sure you completely understand this.

Theorem

A bijection exists between natural numbers $\mathbb{N}$ and rationals $\mathbb{Q}$.

Theorem

No bijection exists between natural numbers $\mathbb{N}$ and reals $\mathbb{R}$.

Question: What about an infinite set, and its power set?

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Proof

- Consider an infinite set S and its corresponding power set P

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Proof

- Consider an infinite set S and its corresponding power set P
- We show a contradiction if $\exists$ bijection $f : S \rightarrow P$

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Proof

- Consider an infinite set S and its corresponding power set P
- We show a contradiction if $\exists$ bijection $f : S \rightarrow P$
- Let i be any element of S and A be any element of P

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Proof

- Consider an infinite set S and its corresponding power set P
- We show a contradiction if $\exists$ bijection $f : S \rightarrow P$
- Let i be any element of S and A be any element of P
- If bijection exists, every $A \in P$ is mapped to by some $i \in S$

Bijection between a set and its powerset

Claim

There does not exist bijection between infinite set and its power set

Proof

- Consider an infinite set S and its corresponding power set P
- We show a contradiction if $\exists$ bijection $f : S \rightarrow P$
- Let i be any element of S and A be any element of P
- If bijection exists, every $A \in P$ is mapped to by some $i \in S$
- Consider the following set $A' \in P$:

$$A' = \{i \in S : i \notin f(i)\}$$

- A': All i which are not present in the set $f(i)$

Proof Cont'd

- Now lets look at A' more closely

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists$ j such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?
- If the answer is **'Yes'**:

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?
- If the answer is **'Yes'**:
  - If $j \in A'$, then by definition of A', j maps to a set which does not contain j
  - Contradiction!

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?
- If the answer is **'Yes'**:
  - If $j \in A'$, then by definition of A', j maps to a set which does not contain j
  - Contradiction!
- If the answer is **'No'**:

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?
- If the answer is **'Yes'**:
  - If $j \in A'$, then by definition of A', j maps to a set which does not contain j
  - Contradiction!
- If the answer is **'No'**:
  - If $j \notin A'$, then j is mapping to a set (A') which doesn't contain j. So by definition of A', j should be in A'.
  - Contradiction!

Proof Cont'd

- Now lets look at A' more closely
- Since f is a bijection, $\exists\ j$ such that $f(j) = A'$
- Here is the fatal question: Is $j \in A'$?
- If the answer is **'Yes'**:
  - If $j \in A'$, then by definition of A', j maps to a set which does not contain j
  - Contradiction!
- If the answer is **'No'**:
  - If $j \notin A'$, then j is mapping to a set (A') which doesn't contain j. So by definition of A', j should be in A'.
  - Contradiction!

Important Corollary

Given *any* function $f : S \longrightarrow P$, there always exists a $p_j \in P$ such that **no** $s_i \in S$ maps to p_j.

A Quote

"It seemed unworthy of a grown man to spend his time on such trivialities, but what was I to do?" –Bertrand Russell.

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings
  - Interpret each p_j to represent a language, say, L_j

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings
  - Interpret each p_j to represent a language, say, L_j
- Now, each **TM** M_{s_i} decides some language L_j

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings
  - Interpret each p_j to represent a language, say, L_j
- Now, each **TM** M_{s_i} decides some language L_j
  - This defines $f(\cdot)$: $f(s_i) = p_j$, where M_{s_i} decides L_j

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings
  - Interpret each p_j to represent a language, say, L_j
- Now, each **TM** M_{s_i} decides some language L_j
  - This defines $f(\cdot)$: $f(s_i) = p_j$, where M_{s_i} decides L_j
- By previous corollary, $\exists\ p_k \in P$ s.t. no s_i maps to p_k

Uncountable Sets and Uncomputable functions

Claim: Uncomputable functions exist

S: Infinite set of all binary strings, P: Power-set of S

- Each $s_i \in S$ is a binary string
  - A Turing Machine can be represented by a string
  - Interpret each s_i to represent some **TM** M_{s_i}
- Each $p_j \in P$ is a *set* of binary strings
  - A language is just a set of strings
  - Interpret each p_j to represent a language, say, L_j
- Now, each **TM** M_{s_i} decides some language L_j
  - This defines $f(\cdot)$: $f(s_i) = p_j$, where M_{s_i} decides L_j
- By previous corollary, $\exists\ p_k \in P$ s.t. no s_i maps to p_k
  - Then there is no **TM** that decides the language L_k!

HALT

Halting function

HALT is not computable by any **TM**

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts
- Consider the case when $x = \lfloor M \rfloor$: $M_H(\lfloor M \rfloor; \lfloor M \rfloor)$

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts
- Consider the case when $x = \lfloor M \rfloor$: $M_H(\lfloor M \rfloor; \lfloor M \rfloor)$
- If M_H exists, then there also exists a **TM** D s.t.

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts
- Consider the case when $x = \lfloor M \rfloor$: $M_H(\lfloor M \rfloor; \lfloor M \rfloor)$
- If M_H exists, then there also exists a **TM** D s.t.
  - D takes as input a **TM** M

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts
- Consider the case when $x = \lfloor M \rfloor$: $M_H(\lfloor M \rfloor; \lfloor M \rfloor)$
- If M_H exists, then there also exists a **TM** D s.t.
  - D takes as input a **TM** M
  - D works the same way as M_H. However, when M_H is about to halt with a **'yes'**, D goes into an infinite loop.

HALT

Halting function

HALT is not computable by any **TM**

Proof.

- Assume a **TM** M_H computes **HALT**
  - M_H takes as input a **TM** M (as a string) and M's input x
  - $M_H(\lfloor M \rfloor; x) = 1$ if M halts on input x, 0 otherwise
  - Note that M_H *always* halts
- Consider the case when $x = \lfloor M \rfloor$: $M_H(\lfloor M \rfloor; \lfloor M \rfloor)$
- If M_H exists, then there also exists a **TM** D s.t.
  - D takes as input a **TM** M
  - D works the same way as M_H. However, when M_H is about to halt with a **'yes'**, D goes into an infinite loop.
  - When M_H would halt with a **'No'**, D also halts.

□

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$
  - Contradiction!

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$
  - Contradiction!
- If $D(\lfloor D \rfloor)$ does not halt:

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$
  - Contradiction!
- If $D(\lfloor D \rfloor)$ does not halt:
  - $D(\lfloor D \rfloor)$ does not halt $\implies$ D halts on input $\lfloor D \rfloor$

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$
  - Contradiction!
- If $D(\lfloor D \rfloor)$ does not halt:
  - $D(\lfloor D \rfloor)$ does not halt $\implies$ D halts on input $\lfloor D \rfloor$
  - Contradiction!

HALT

Proof Cont'd

- So far: if M_H exists, then there also exists D:
  - $D(\lfloor M \rfloor)$ does not halt if M halts on input $\lfloor M \rfloor$
  - $D(\lfloor M \rfloor)$ halts if M does not halt on input $\lfloor M \rfloor$
- *Fatal question*: Does $D(\lfloor D \rfloor)$ halt?
- If $D(\lfloor D \rfloor)$ halts:
  - $D(\lfloor D \rfloor)$ halts $\implies$ D does not halt on input $\lfloor D \rfloor$
  - Contradiction!
- If $D(\lfloor D \rfloor)$ does not halt:
  - $D(\lfloor D \rfloor)$ does not halt $\implies$ D halts on input $\lfloor D \rfloor$
  - Contradiction!
- This type of argument is called a *diagonalization* argument.

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**
- **HALT** doesn't have an algorithm, hence so doesn't A

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**
- **HALT** doesn't have an algorithm, hence so doesn't A

Claim: This is undecidable: $\{M\colon M \text{ halts on all inputs}\}$

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**
- **HALT** doesn't have an algorithm, hence so doesn't A

Claim: This is undecidable: $\{M\colon M \text{ halts on all inputs}\}$

- Reduce **HALT** to this problem

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**
- **HALT** doesn't have an algorithm, hence so doesn't A

Claim: This is undecidable: $\{M\colon M \text{ halts on all inputs}\}$

- Reduce **HALT** to this problem
- Given $M; x$ construct the following machine M':
  $M'(y) = M(x)$ if $y = x$, $M'(y) = 0$ otherwise

The Halting problem spawns more problems

Using reductions, can show many problems to be undecidable

- Goal: show that the problem A is undecidable
- Establish that, if there were an algorithm for problem A, then there would be an algorithm for **HALT**
- **HALT** doesn't have an algorithm, hence so doesn't A

Claim: This is undecidable: $\{M\colon M \text{ halts on all inputs}\}$

- Reduce **HALT** to this problem
- Given $M; x$ construct the following machine M':
  $M'(y) = M(x)$ if $y = x$, $M'(y) = 0$ otherwise
- M' halts on all inputs if and only if M halts on x