The Post-Correspondence Problem: Minions and Matching Strings

Gloria Chun and Alicia Li

PRIMES Circle, Mentor: Alexandra Hoey

MIT

May 22nd, 2021

Table of Contents

An Interesting Puzzle

We have a collection of dominos:

$$\left\{\left[\frac{onsb}{sb}\right], \left[\frac{a}{an}\right], \left[\frac{nas}{as}\right], \left[\frac{i}{ni}\right], \left[\frac{ni}{on}\right], \left[\frac{m}{mi}\right], \left[\frac{n}{\epsilon}\right]\right\}.$$

An Interesting Puzzle

We have a collection of dominos:

$$\{[\frac{onsb}{sb}], [\frac{a}{an}], [\frac{nas}{as}], [\frac{i}{ni}], [\frac{ni}{on}], [\frac{m}{mi}], [\frac{n}{\epsilon}]\}.$$

Goal: to make a *match* (repetition allowed) where the string with all the symbols on the top is the same as the string on the bottom.

An Interesting Puzzle

We have a collection of dominos:

$$\{[\frac{onsb}{sb}], [\frac{a}{an}], [\frac{nas}{as}], [\frac{i}{ni}], [\frac{ni}{on}], [\frac{m}{mi}], [\frac{n}{\epsilon}]\}.$$

Goal: to make a *match* (repetition allowed) where the string with all the symbols on the top is the same as the string on the bottom.

An Interesting Puzzle

We have a collection of dominos:

$$\{[\frac{onsb}{sb}], [\frac{a}{an}], [\frac{nas}{as}], [\frac{i}{ni}], [\frac{ni}{on}], [\frac{m}{mi}], [\frac{n}{\epsilon}]\}.$$

Goal: to make a *match* (repetition allowed) where the string with all the symbols on the top is the same as the string on the bottom.

Solution

$$\{[\frac{m}{mi}], [\frac{i}{ni}], [\frac{ni}{on}], [\frac{onsb}{sb}], [\frac{a}{an}], [\frac{n}{\epsilon}], [\frac{a}{an}], [\frac{nas}{as}]\}$$

Post Correspondence Problem Definition

We have the collection of dominos $\{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], \ldots, [\frac{t_k}{b_k}]\}$.

Definition

A *match* is $i_1, i_2, \ldots, i_\ell$ where $t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$.

Post Correspondence Problem Definition

We have the collection of dominos $\{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], \ldots, [\frac{t_k}{b_k}]\}$.

Definition

A *match* is $i_1, i_2, \ldots, i_\ell$ where $t_{i_1} t_{i_2} \cdots t_{i_\ell} = b_{i_1} b_{i_2} \cdots b_{i_\ell}$.

Definition (Post Correspondence Problem)

PCP is the language of collections of dominos with a match:

$$PCP = \{ \langle P \rangle \mid \text{the set } P \text{ of dominos has a match} \}.$$

Post-Correspondence Problem Cont.

Can you find the match?

Example

$\{[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]\}$

Post-Correspondence Problem Cont.

Can you find the match?

Example

$\{[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]\}$

Solution

$\{[\frac{a}{ab}], [\frac{b}{ca}], [\frac{ca}{a}], [\frac{a}{ab}], [\frac{abc}{c}]\}$

Solving the Post Correspondence Problem?

Can we write a program to determine if a collection of dominos has a match?

Solving the Post Correspondence Problem?

Can we write a program to determine if a collection of dominos has a match?

Unfortunately, this is impossible!

Solving the Post Correspondence Problem?

Can we write a program to determine if a collection of dominos has a match?

Unfortunately, this is impossible!
PCP is *undecidable*.

Solving the Post Correspondence Problem?

Can we write a program to determine if a collection of dominos has a match?

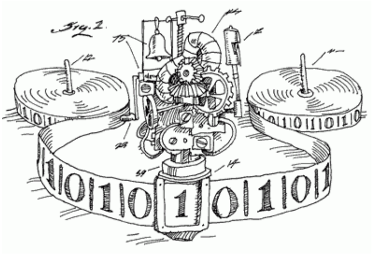
Unfortunately, this is impossible!
PCP is *undecidable*.

Definition

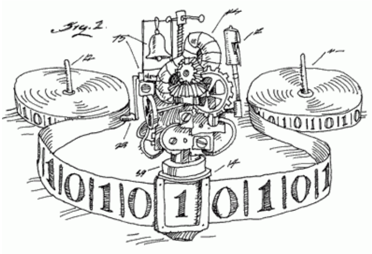
A language is *undecidable* if there is **no** programmable algorithm that

- accepts all inputs in the language, and
- rejects all inputs not in the language.

Turing Machines!

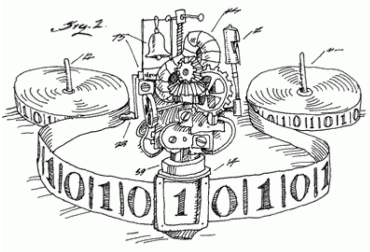


Turing Machines!



A *Turing Machine* consists of

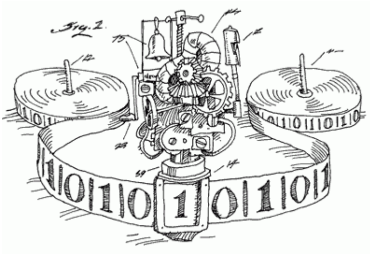
Turing Machines!



A *Turing Machine* consists of

- A finite set of *states* (including a start, accept, and a reject state)

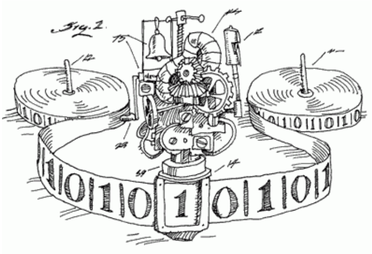
Turing Machines!



A *Turing Machine* consists of

- A finite set of *states* (including a start, accept, and a reject state)
- An *infinite tape*

Turing Machines!



A *Turing Machine* consists of

- A finite set of *states* (including a start, accept, and a reject state)
- An *infinite tape*
- A *tape head* to navigate, read, and edit the elements according to the *transition function*.

Example of a Turing Machine

Example

Below is an example of a Turing machine that recognizes the language $B = \{0^{2^n} | n \in \mathbb{Z}_{\geq 0}\}$. Let's work with this input:

Example of a Turing Machine

Example

Below is an example of a Turing machine that recognizes the language $B = \{0^{2^n} | n \in \mathbb{Z}_{\geq 0}\}$. Let's work with this input:

0	0	0	0	0	0	0	0

Example of a Turing Machine

After receiving the input string,

Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.

Example of a Turing Machine

After receiving the input string,

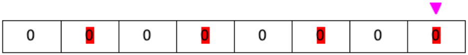
1. Move left to right across the tape, crossing out every second 0.

0	0	0	0	0	0	0	0

Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.

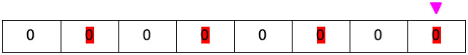


2. Accept the tape if it held a single 0 in step 1.

Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.

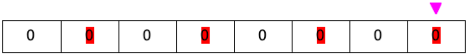


2. Accept the tape if it held a single 0 in step 1.
3. Reject if the tape held more than one 0 in step 1 and the number of 0s was odd.

Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.

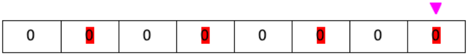


2. Accept the tape if it held a single 0 in step 1.
3. Reject if the tape held more than one 0 in step 1 and the number of 0s was odd.
4. Return the tape head all the way to the tape's left end.

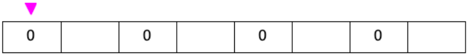
Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.



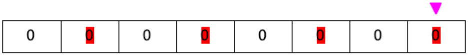
2. Accept the tape if it held a single 0 in step 1.
3. Reject if the tape held more than one 0 in step 1 and the number of 0s was odd.
4. Return the tape head all the way to the tape's left end.



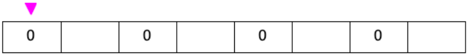
Example of a Turing Machine

After receiving the input string,

1. Move left to right across the tape, crossing out every second 0.

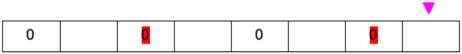


2. Accept the tape if it held a single 0 in step 1.
3. Reject if the tape held more than one 0 in step 1 and the number of 0s was odd.
4. Return the tape head all the way to the tape's left end.

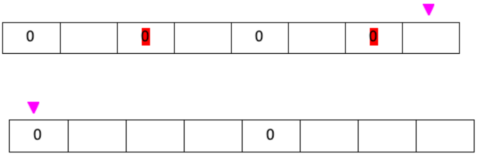


5. Return back to step 1, then repeat.

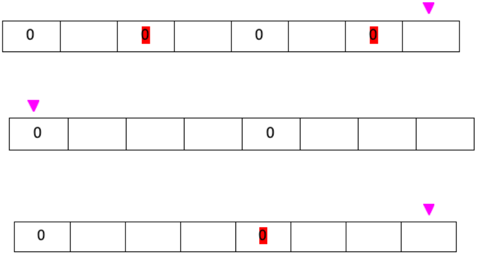
Example of a Turing Machine



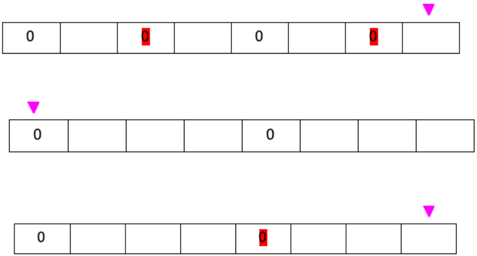
Example of a Turing Machine



Example of a Turing Machine



Example of a Turing Machine



Computation History of a Turing Machine

Definition

A *configuration* of a Turing machine is a setting of three elements:

Computation History of a Turing Machine

Definition

A *configuration* of a Turing machine is a setting of three elements:

1. the current *state*

Computation History of a Turing Machine

Definition

A *configuration* of a Turing machine is a setting of three elements:

1. the current *state*
2. the current *tape contents*

Computation History of a Turing Machine

Definition

A *configuration* of a Turing machine is a setting of three elements:

1. the current *state*
2. the current *tape contents*
3. the current *head location*

Computation History of a Turing Machine

Definition

A *configuration* of a Turing machine is a setting of three elements:

1. the current *state*
2. the current *tape contents*
3. the current *head location*

The history of changes in configurations is called a *computation history*.

How to write a configuration

We represent a configuration as uqv where

- uv is the current tape contents,
- q is the current state, and
- the first symbol of v is the location of the current tape head.

How to write a configuration

We represent a configuration as uqv where

- uv is the current tape contents,
- q is the current state, and
- the first symbol of v is the location of the current tape head.

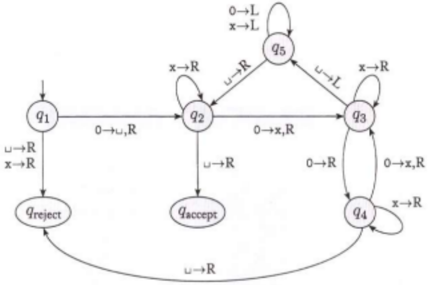
Example

Tape: $\sqcup\, 0\, \mathbf{0}\, 0\, 0$

State: q_2

Configuration: $\sqcup 0\ q_2\ 000$

Computation History Example



Example

On input 00, this Turing machine has computation history

$q_1\ 00$,

The Post-Correspondence Problem: Minions and Matching Strings

PRIMES Circle, Mentor: Alexandra Hoey

Post-Correspondence Problem

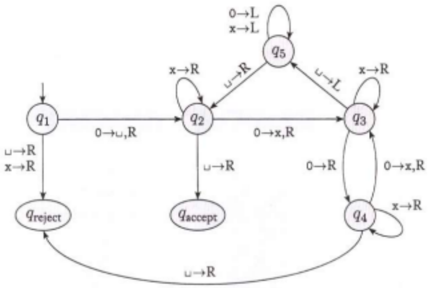
Turing Machines

A_{TM} and Reducibility

Reducing A_{TM} to PCP

Conclusion

Computation History Example

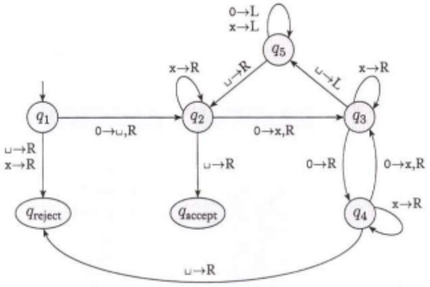


Example

On input 00, this Turing machine has computation history

q_1 00, $\sqcup$ q_2 0,

Computation History Example



Example

On input 00, this Turing machine has computation history

$$q_1\,00,\ \sqcup\, q_2\, 0,\ \sqcup x q_3 \sqcup,\ \sqcup q_5 x \sqcup,\ q_5 \sqcup x \sqcup,\ \sqcup q_2 x \sqcup,\ \sqcup x q_2 \sqcup,\ \sqcup x \sqcup q_{accept}$$

A_{TM}

A_{TM}

The language $A_{TM} = \{\langle M, w\rangle | M \text{ is a TM that accepts } w\}$

IMPORTANT

A_{TM} is undecidable. We can prove this with diagonalization.

Reduction

Definition

A *reduction* converts one problem to another such that a solution to the second problem can be used to solve the first problem.

[Sipser, Corollary 5.23]

If A is undecidable and reducible to B, B must be undecidable as well.

Despicable Me and Reducibility

In *Despicable Me*, the protagonist Gru wants to steal the moon.

Despicable Me and Reducibility

In *Despicable Me*, the protagonist Gru wants to steal the moon.
Reduction: It is enough to steal a shrink ray.

Despicable Me and Reducibility

In *Despicable Me*, the protagonist Gru wants to steal the moon.
Reduction: It is enough to steal a shrink ray.

Then, he can shrink the moon, and steal the new mini moon instead.

Despicable Me and Reducibility

In *Despicable Me*, the protagonist Gru wants to steal the moon.
Reduction: It is enough to steal a shrink ray.

Then, he can shrink the moon, and steal the new mini moon instead.

You might think stealing the moon would be impossible.
If so, then stealing a shrink ray must be impossible too.
(Corollary 5.23)

The Story of Urg

In an alternate universe, let's say Gru has a counterpart Urg. However, Urg is disinterested in stealing the moon. Rather, he wants to steal a shrink ray

The Story of Urg

In an alternate universe, let's say Gru has a counterpart Urg. However, Urg is disinterested in stealing the moon. Rather, he wants to steal a shrink ray

Urg reasons:

1. Stealing a shrink ray is as difficult as stealing the moon (same logic as Gru).

The Story of Urg

In an alternate universe, let's say Gru has a counterpart Urg. However, Urg is disinterested in stealing the moon. Rather, he wants to steal a shrink ray

Urg reasons:

1. Stealing a shrink ray is as difficult as stealing the moon (same logic as Gru).
2. In Urg's world, however, it is proven that stealing the moon is impossible.

The Story of Urg

In an alternate universe, let's say Gru has a counterpart Urg. However, Urg is disinterested in stealing the moon. Rather, he wants to steal a shrink ray

Urg reasons:

1. Stealing a shrink ray is as difficult as stealing the moon (same logic as Gru).
2. In Urg's world, however, it is proven that stealing the moon is impossible.
3. Therefore, stealing a shrink ray must be impossible as well.

Despicable Me and Reducibility

Our Reduction

To show PCP is undecidable,

we simulate an accepting Turing machine configuration through a PCP match

and prove A_{TM} is undecidable.

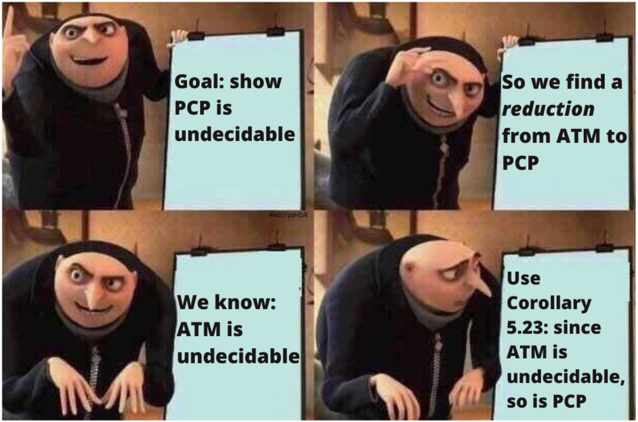
Urg's Reduction

To show he can't steal a shrink ray,

Urg hypothetically shrinks the moon

and proves stealing the moon is impossible.

New Mission: Proving PCP is Undecidable



Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:

Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:
  N: On input $\langle M, w \rangle$,

Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:
  N: On input $\langle M, w \rangle$,
  1. Make a set of dominos that has a match iff M accepts w.

Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:
  N: On input $\langle M, w \rangle$,
  1. Make a set of dominos that has a match iff M accepts w.
  2. Run B on this set of dominos.

Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:
  N: On input $\langle M, w \rangle$,
  1. Make a set of dominos that has a match iff M accepts w.
  2. Run B on this set of dominos.
  3. Accept if B accepts; reject otherwise.

Reducing A_{TM} to PCP

Reduction Strategy:

- Suppose we have a Turing machine B that decides PCP.
- We use B to create a Turing machine N that decides A_{TM}:
  N: On input $\langle M, w \rangle$,
  1. Make a set of dominos that has a match iff M accepts w.
  2. Run B on this set of dominos.
  3. Accept if B accepts; reject otherwise.
- Construct the set of dominos in 7 steps.

MPCP

We must first convert PCP to Modified PCP (MPCP).

Definition

MPCP is the language of instances of *PCP* whose matches start with $[\frac{t_1}{b_1}]$:

$$MPCP = \{ \langle P' \rangle \mid P' \text{ is a collection of dominos with a match starting with the first domino} \}.$$

The Start Domino - Step 1

STEP 1:

Insert the domino $[\frac{\#}{\#q_0 w_1 w_2 \ldots w_n}]$.
This is the start of the match, $[\frac{t_1}{b_1}]$.

The Start Domino - Step 1

STEP 1:

Insert the domino $[\frac{\#}{\#q_0 w_1 w_2 \ldots w_n}]$.
This is the start of the match, $[\frac{t_1}{b_1}]$.

In the next steps (2,3,4) we simulate the computation.

The Transition Function - Steps 2 and 3

Steps 2 and 3 involve the dominos for transition functions (how the current state and current tape head location changes).

The Transition Function - Steps 2 and 3

Steps 2 and 3 involve the dominos for transition functions (how the current state and current tape head location changes).

STEP 2: Tape head moves right.

Read a in state q:

- Turing machine replaces a with b and moves to state q'
- put domino $\left[\frac{q\mathrm{a}}{\mathrm{b}q'}\right]$ into P'.

The Transition Function - Steps 2 and 3

Steps 2 and 3 involve the dominos for transition functions (how the current state and current tape head location changes).

STEP 2: Tape head moves right.

Read $\mathtt{a}$ in state q:

- Turing machine replaces $\mathtt{a}$ with $\mathtt{b}$ and moves to state q'
- put domino $[\frac{q\mathtt{a}}{\mathtt{b}q'}]$ into P'.

STEP 3: Tape head moves left.

- Turing machine replaces $\mathtt{a}$ with $\mathtt{b}$ and moves to state q'
- put domino $[\frac{cq\mathtt{a}}{q'c\mathtt{b}}]$ into P'.

Symbols and Separations - Steps 4 and 5

Now we add the tape alphabet: these dominos are used in the portion of the configurations outside the transition functions.

STEP 4:

For every a in the tape alphabet, insert $\left[\frac{\mathtt{a}}{\mathtt{a}}\right]$ into P'.

Symbols and Separations - Steps 4 and 5

Now we add the tape alphabet: these dominos are used in the portion of the configurations outside the transition functions.

STEP 4:

For every a in the tape alphabet, insert $[\frac{\mathtt{a}}{\mathtt{a}}]$ into P'.

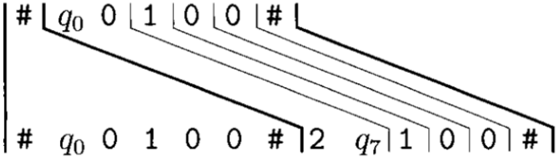
To separate each configuration, we add #s.

STEP 5:

Insert $[\frac{\#}{\#}]$ and $[\frac{\#}{\sqcup\#}]$ into P'.

The second domino allows us to simulate the infinite number of $\sqcup$s.

Example of This Reduction



Accept States - Steps 6 and 7

Add in accept states with symbols to "eat" the remaining characters left on the tape after the machine accepts.

STEP 6:

For every a in the tape alphabet, insert $[\frac{a q_{accept}}{q_{accept}}]$ and $[\frac{q_{accept} a}{q_{accept}}]$ into P'.

Accept States - Steps 6 and 7

Add in accept states with symbols to "eat" the remaining characters left on the tape after the machine accepts.

STEP 6:

For every a in the tape alphabet, insert $[\frac{\mathtt{a}q_{accept}}{q_{accept}}]$ and $[\frac{q_{accept}\mathtt{a}}{q_{accept}}]$ into P'.

STEP 7:

The last domino! $[\frac{q_{accept}\#\#}{\#}]$

Reducing MPCP to PCP

We can force the match to start with the first domino by adding *s:

Reducing MPCP to PCP

We can force the match to start with the first domino by adding *s:

Let

$\star u = *u_1 * u_2 * u_3 \cdots * u_n,$

$u\star = u_1 * u_2 * u_3 \cdots * u_n*,$

$\star u\star = *u_1 * u_2 * u_3 \cdots * u_n*.$

Reducing MPCP to PCP

We can force the match to start with the first domino by adding *s:

Let

$\star u = *u_1 * u_2 * u_3 \cdots * u_n,$

$u\star = u_1 * u_2 * u_3 \cdots * u_n *,$

$\star u \star = *u_1 * u_2 * u_3 \cdots * u_n *.$

Since P' (MPCP) is $\{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], [\frac{t_3}{b_3}], \ldots [\frac{t_k}{b_k}]\}$,

we can let P (PCP) be $\{[\frac{\star t_1}{\star b_1 \star}], [\frac{\star t_2}{b_2 \star}], [\frac{\star t_3}{b_3 \star}], \ldots [\frac{\star t_k}{b_k \star}], [\frac{*\Diamond}{\Diamond}]\}$, where $[\frac{*\Diamond}{\Diamond}]$ is there to add the extra $*$.

Summary

Summary

1. A *match* occurs when the string with all the symbols on the top is the same as the string on the bottom.

Summary

1. A *match* occurs when the string with all the symbols on the top is the same as the string on the bottom.
2. A_{TM} is a set of all Turing machines M and input string w in which the machine M accepts w.

Summary

1. A *match* occurs when the string with all the symbols on the top is the same as the string on the bottom.
2. A_{TM} is a set of all Turing machines M and input string w in which the machine M accepts w.
3. A_{TM} is undecidable! (diagonalization)

Summary

1. A *match* occurs when the string with all the symbols on the top is the same as the string on the bottom.
2. A_{TM} is a set of all Turing machines M and input string w in which the machine M accepts w.
3. A_{TM} is undecidable! (diagonalization)
4. PCP is undecidable because we can reduce A_{TM} to MPCP which we can then reduce back to PCP.

Summary

1. A *match* occurs when the string with all the symbols on the top is the same as the string on the bottom.
2. A_{TM} is a set of all Turing machines M and input string w in which the machine M accepts w.
3. A_{TM} is undecidable! (diagonalization)
4. PCP is undecidable because we can reduce A_{TM} to MPCP which we can then reduce back to PCP.
5. Our reduction is essentially simulating an accepting Turing machine computation through an MPCP match.

Acknowledgements

We would like to thank...

- PRIMES Circle for this wonderful opportunity
- Alexandra for being a great mentor :)
- You for coming!!!

References

- [la] Sipser, M. (2013). *Introduction to the theory of computation* (3rd ed.). Cengage Learning.
- [lb] Coffin, P., Renaud, C. (2010). Despicable Me. Universal Pictures.
- [lc] Turing Machine [Illustration]. (n.d.).

A_{TM} is undecidable

Theorem

A_{TM} is undecidable.

Proof

First, let's assume that A_{TM} is decidable. Then, by the definition of decidable Turing machines, there must exist some decider H, where $L(H) = A_{TM}$, such that for an input string $\langle M, w \rangle$, it

1. accepts if M accepts w, and
2. rejects if M rejects w.

Proof

Now, consider a new Turing machine D, which takes Turing machines $\langle M \rangle$ as inputs. It

1. accepts when H rejects $\langle M, \langle M \rangle \rangle$, and
2. rejects when H accepts $\langle M, \langle M \rangle \rangle$

Proof

Consider a case when D takes in itself, $\langle D \rangle$, as an input. Then by construction, D will

1. reject $\langle D \rangle$ when H accepts $\langle D, \langle D \rangle \rangle$, and
2. accept $\langle D \rangle$ when H rejects $\langle D, \langle D \rangle \rangle$

However, because H accepts $\langle D, \langle D \rangle \rangle$ if and only if D accepts $\langle D \rangle$, we see that there exists a contradiction either way:

1. D rejects $\langle D \rangle$ if and only if D accepts $\langle D \rangle$, or
2. D accepts $\langle D \rangle$ if and only if D rejects $\langle D \rangle$.

Therefore, such a decider H cannot exist, so A_{TM} is undecidable.