COL 352 Introduction to Automata and Theory of Computation

Nikhil Balaji

Bharti 420
Indian Institute of Technology, Delhi
nbalaji@cse.iitd.ac.in

April 9, 2023

Lecture 28: Reductions 3

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ is a PDA and $L(M) = \Sigma^*\}$ is undecidable.

Proof Strategy

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M\rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*)

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M\rangle \mid M$ is a PDA and $L(M) = \Sigma^*\}$ is undecidable.

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ is a PDA and $L(M) = \Sigma^*\}$ is undecidable.

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Formally,

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M\rangle \mid M$ is a PDA and $L(M) = \Sigma^*\}$ is undecidable.

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Formally,

Input $(M, w) \quad \longrightarrow \quad N_{M,w}$

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Formally,

Input $(M, w) \quad \longrightarrow \quad N_{M,w}$

if $w \in L(M) \quad \longrightarrow \quad \exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M \rangle \mid M$ is a PDA and $L(M) = \Sigma^*\}$ is undecidable.

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Formally,

$$\begin{array}{lcl} \text{Input } (M, w) & \longrightarrow & N_{M,w} \\ \text{if } w \in L(M) & \longrightarrow & \exists x \in \Sigma^*, \text{ s.t. } x \notin L(N_{M,w}) \\ \text{if } w \notin L(M) & \longrightarrow & L(N_{M,w}) = \Sigma^* \end{array}$$

Universality of CFG

Lemma

$ALL_{CFG} = \{\langle M\rangle \mid M$ *is a PDA and* $L(M) = \Sigma^*\}$ *is undecidable.*

Proof Strategy

For a TM M and input w we create a PDA $N_{M,w}$ such that

$N_{M,w}$ accepts all strings (i.e. accepts Σ^*) if M accepts w, and

$N_{M,w}$ rejects at least one string if M does not accept w.

Formally,

$$\text{Input } (M, w) \quad \longrightarrow \quad N_{M,w}$$

$$\text{if } w \in L(M) \quad \longrightarrow \quad \exists x \in \Sigma^*, \text{ s.t. } x \notin L(N_{M,w})$$

$$\text{if } w \notin L(M) \quad \longrightarrow \quad L(N_{M,w}) = \Sigma^*$$

Filling in the details

The following two details need to be addressed.

Q_1 How should we design $N_{M,w}$?

Filling in the details

The following two details need to be addressed.

Q_1 How should we design $N_{M,w}$?

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input $(M, w) \longrightarrow N_{M,w}$

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

$$\text{Input } (M, w) \quad \longrightarrow \quad N_{M,w}$$

$$\text{if } w \in L(M) \quad \longrightarrow \quad \exists x \in \Sigma^*, \text{ s.t. } x \notin L(N_{M,w})$$

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input (M, w) $\longrightarrow$ $N_{M,w}$

if $w \in L(M)$ $\longrightarrow$ $\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$

if $w \notin L(M)$ $\longrightarrow$ $L(N_{M,w}) = \Sigma^*$

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

$$\begin{array}{lcl} \text{Input } (M, w) & \longrightarrow & N_{M,w} \\ \text{if } w \in L(M) & \longrightarrow & \exists x \in \Sigma^*, \text{ s.t. } x \notin L(N_{M,w}) \\ \text{if } w \notin L(M) & \longrightarrow & L(N_{M,w}) = \Sigma^* \end{array}$$

Assume that ALL_{CFL} is decidable.

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

$$\begin{array}{lcl}
\text{Input } (M, w) & \longrightarrow & N_{M,w} \\
\text{if } w \in L(M) & \longrightarrow & \exists x \in \Sigma^*, \text{ s.t. } x \notin L(N_{M,w}) \\
\text{if } w \notin L(M) & \longrightarrow & L(N_{M,w}) = \Sigma^*
\end{array}$$

Assume that ALL_{CFL} is decidable.
C be the TM deciding it.

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Design A as follows:

For an M, w pair, create $N_{M,w}$.

Input (M, w)	$\longrightarrow$	$N_{M,w}$
if $w \in L(M)$	$\longrightarrow$	$\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$
if $w \notin L(M)$	$\longrightarrow$	$L(N_{M,w}) = \Sigma^*$

Assume that ALL_{CFL} is decidable.
C be the TM deciding it.

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input (M, w)	$\longrightarrow$	$N_{M,w}$
if $w \in L(M)$	$\longrightarrow$	$\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$
if $w \notin L(M)$	$\longrightarrow$	$L(N_{M,w}) = \Sigma^*$

Assume that ALL_{CFL} is decidable.
C be the TM deciding it.

Design A as follows:

For an M, w pair, create $N_{M,w}$.

Feed $\langle N_{M,w} \rangle$ to C.

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input (M, w) $\longrightarrow$ $N_{M,w}$

if $w \in L(M)$ $\longrightarrow$ $\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$

if $w \notin L(M)$ $\longrightarrow$ $L(N_{M,w}) = \Sigma^*$

Assume that ALL_{CFL} is decidable. C be the TM deciding it.

Design A as follows:

For an M, w pair, create $N_{M,w}$.

Feed $\langle N_{M,w} \rangle$ to C.

If C accepts then reject;

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input (M, w)	$\longrightarrow$	$N_{M,w}$
if $w \in L(M)$	$\longrightarrow$	$\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$
if $w \notin L(M)$	$\longrightarrow$	$L(N_{M,w}) = \Sigma^*$

Assume that ALL_{CFL} is decidable.
C be the TM deciding it.

Design A as follows:

For an M, w pair, create $N_{M,w}$.

Feed $\langle N_{M,w} \rangle$ to C.

If C accepts then reject;

else accept.

Details for Q_2

Q_2 If such an $N_{M,w}$ is designed then why have we proved that ALL_{CFL} is undecidable?

Input (M, w)	$\longrightarrow$	$N_{M,w}$
if $w \in L(M)$	$\longrightarrow$	$\exists x \in \Sigma^*,$ s.t. $x \notin L(N_{M,w})$
if $w \notin L(M)$	$\longrightarrow$	$L(N_{M,w}) = \Sigma^*$

Assume that ALL_{CFL} is decidable.
C be the TM deciding it.

Design A as follows:

For an M, w pair, create $N_{M,w}$.

Feed $\langle N_{M,w} \rangle$ to C.

If C accepts then reject;

else accept.

Details for Q_1: reduction via computation history

Q_1 How should we design $N_{M,w}$?

Details for Q_1: reduction via computation history

Q_1 How should we design $N_{M,w}$?

Main idea

Use computational history of M on w.

Details for Q_1: reduction via computation history

Q_1 How should we design $N_{M,w}$?

Main idea

Use computational history of M on w.

Accepting computation history is a sequece of configutations: $C_1, C_2, \ldots, C_\ell$ such that

Details for Q_1: reduction via computation history

Q_1 How should we design $N_{M,w}$?

Main idea

Use computational history of M on w.

Accepting computation history is a sequece of configutations: $C_1, C_2, \ldots, C_\ell$ such that

- C_1 is a start configutation.
- C_ℓ is an accepting configuration.
- for each $1 \leq i \leq \ell$, C_i yields C_{i+1}.

Rejecting computation history is a sequece of configutations: $C_1, C_2, \ldots, C_\ell$ such that

Details for Q_1: reduction via computation history

Q_1 How should we design $N_{M,w}$?

Main idea

Use computational history of M on w.

Accepting computation history is a sequece of configutations: $C_1, C_2, \ldots, C_\ell$ such that

- C_1 is a start configutation.
- C_ℓ is an accepting configuration.
- for each $1 \leq i \leq \ell$, C_i yields C_{i+1}.

Rejecting computation history is a sequece of configutations: $C_1, C_2, \ldots, C_\ell$ such that

- C_1 is a start configutation.
- C_ℓ is a rejecting configuration.
- for each $1 \leq i \leq \ell$, C_i yields C_{i+1}.

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation OR
  - x is a computational history, C_1 is a start configuration, C_ℓ is an accepting configutation, but there exists an i s.t. $1 \le i \le \ell - 1$ and C_i does not yield C_{i+1}.

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation OR
  - x is a computational history, C_1 is a start configuration, C_ℓ is an accepting configutation, but there exists an i s.t. $1 \leq i \leq \ell - 1$ and C_i does not yield C_{i+1}.
- If M accepts w, let $\tilde{x}$ be a accepting computation history of M on w.

  $N_{M,w}$ will reject $\tilde{x}$

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation OR
  - x is a computational history, C_1 is a start configuration, C_ℓ is an accepting configutation, but there exists an i s.t. $1 \leq i \leq \ell - 1$ and C_i does not yield C_{i+1}.
- If M accepts w, let $\tilde{x}$ be a accepting computation history of M on w.

  $N_{M,w}$ will reject $\tilde{x}$, i.e. $\tilde{x} \notin L(N_{M,w})$.

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation OR
  - x is a computational history, C_1 is a start configuration, C_ℓ is an accepting configutation, but there exists an i s.t. $1 \leq i \leq \ell - 1$ and C_i does not yield C_{i+1}.
- If M accepts w, let $\tilde{x}$ be a accepting computation history of M on w.

  $N_{M,w}$ will reject $\tilde{x}$, i.e. $\tilde{x} \notin L(N_{M,w})$.
- If M does not accept w, then

  no matter what x is, $N_{M,w}$ will accept x

Universality of CFLs is undecidable!

- Interpret input x to $N_{M,w}$ as a computational history of M on w.
- Design $N_{M,w}$ s.t. it accepts x if any of the following conditions holds:
  - x does not have the pattern of a computational history of x OR
  - x is a computational history, but C_1 is not a start configuration OR
  - x is a computational history, C_1 is a start configuration, but C_ℓ is not an accepting configutation OR
  - x is a computational history, C_1 is a start configuration, C_ℓ is an accepting configutation, but there exists an i s.t. $1 \le i \le \ell - 1$ and C_i does not yield C_{i+1}.
- If M accepts w, let $\tilde{x}$ be a accepting computation history of M on w.

  $N_{M,w}$ will reject $\tilde{x}$, i.e. $\tilde{x} \notin L(N_{M,w})$.
- If M does not accept w, then

  no matter what x is, $N_{M,w}$ will accept x, i.e. $L(N_{M,w}) = \Sigma^*$.

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists.

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Example: Consider the lists $A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$.

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Example: Consider the lists $A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$. **Solution:** There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$,

Witness: $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Example: Consider the lists $A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$. **Solution:** There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$,

Witness: $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$

- What about $A = \{0011, 11, 1101\}$ and $B = \{101, 1, 110\}$?

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \le i_1, i_2, \ldots, i_m \le n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Example: Consider the lists $A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$. **Solution:** There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$,

Witness: $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$

- What about $A = \{0011, 11, 1101\}$ and $B = \{101, 1, 110\}$?
- What about $A = \{100, 0, 1\}$ and $B = \{1, 100, 0\}$?

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Example: Consider the lists $A = \langle 110, 0011, 0110 \rangle$ and $B = \langle 110110, 00, 110 \rangle$. **Solution:** There is a sequence $i = 2, 3, 1$ such that $s_2 s_3 s_1 = t_2 t_3 t_1$,

Witness: $s_2 s_3 s_1 = 00110110110$ and $t_2 t_3 t_1 = 00110110110$

- What about $A = \{0011, 11, 1101\}$ and $B = \{101, 1, 110\}$?
- What about $A = \{100, 0, 1\}$ and $B = \{1, 100, 0\}$?(Shortest solution lenth $= 75$)

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists.

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Can you design an algorithm to solve this problem? A semi-algorithm?

A String Matching Problem

String List Matching Problem

Given two lists $A = \langle s_1, s_2, \ldots, s_n \rangle$ and $B = \langle t_1, t_2, \ldots, t_n \rangle$ of strings of equal length, decide whether there is a sequence of combining elements that produces same string for both lists. Formally, does there exist a finite sequence $1 \leq i_1, i_2, \ldots, i_m \leq n$ (no limit on length) such that

$$s_{i_1} s_{i_2} \ldots s_{i_n} = t_{i_1} t_{i_2} \ldots t_{i_n}$$

Can you design an algorithm to solve this problem? A semi-algorithm?

Theorem

There is no algorithm for the string-list matching problem. In other words, this problem is undecidable.

Post's Correspondence Problem

A Domino game

Given a collection of dominos $\begin{bmatrix} b \\ ca \end{bmatrix}$ $\begin{bmatrix} a \\ ab \end{bmatrix}$ $\begin{bmatrix} ca \\ a \end{bmatrix}$ $\begin{bmatrix} abc \\ c \end{bmatrix}$

- A match is a list of these dominos (with possible repetitions) such that the string formed in top is same as string formed by bottom row.

Post's Correspondence Problem

A Domino game

Given a collection of dominos $\begin{bmatrix} b \\ ca \end{bmatrix}$ $\begin{bmatrix} a \\ ab \end{bmatrix}$ $\begin{bmatrix} ca \\ a \end{bmatrix}$ $\begin{bmatrix} abc \\ c \end{bmatrix}$

- A match is a list of these dominos (with possible repetitions) such that the string formed in top is same as string formed by bottom row.
- Does this collection of dominos have a match?

Post's Correspondence Problem

A Domino game

Given a collection of dominos $\begin{bmatrix} b \\ ca \end{bmatrix}$ $\begin{bmatrix} a \\ ab \end{bmatrix}$ $\begin{bmatrix} ca \\ a \end{bmatrix}$ $\begin{bmatrix} abc \\ c \end{bmatrix}$

- A match is a list of these dominos (with possible repetitions) such that the string formed in top is same as string formed by bottom row.
- Does this collection of dominos have a match?
- Same as the string matching problem

Post's Correspondence Problem

A Domino game

Given a collection of dominos $\begin{bmatrix} b \\ ca \end{bmatrix}$ $\begin{bmatrix} a \\ ab \end{bmatrix}$ $\begin{bmatrix} ca \\ a \end{bmatrix}$ $\begin{bmatrix} abc \\ c \end{bmatrix}$

- A match is a list of these dominos (with possible repetitions) such that the string formed in top is same as string formed by bottom row.
- Does this collection of dominos have a match?
- Same as the string matching problem
- Called Post's Correspondance Problem or PCP.

Theorem

PCP is undecidable.

- Encode TM computation histories!
- Each transition as a domino!
- Simulate the run using the dominos.

Proof of Undecidability of PCP

Simplifying assumptions

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.
- If $w = \varepsilon$, then use $\sqcup$ instead of w.

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.
- If $w = \varepsilon$, then use $\sqcup$ instead of w.
- $P = \left\{ \begin{bmatrix} t_1 \\ b_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ b_2 \end{bmatrix}, \ldots, \begin{bmatrix} t_k \\ b_k \end{bmatrix} \right\}$
- Modify PCP so that match must start with a given domino, say the first one.

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.
- If $w = \varepsilon$, then use $\sqcup$ instead of w.
- $P = \left\{ \begin{bmatrix} t_1 \\ b_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ b_2 \end{bmatrix}, \ldots, \begin{bmatrix} t_k \\ b_k \end{bmatrix} \right\}$
- Modify PCP so that match must start with a given domino, say the first one.

We define a reduction from A_{TM} to (M)PCP.

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.
- If $w = \varepsilon$, then use $\sqcup$ instead of w.
- $P = \left\{ \begin{bmatrix} t_1 \\ b_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ b_2 \end{bmatrix}, \ldots, \begin{bmatrix} t_k \\ b_k \end{bmatrix} \right\}$
- Modify PCP so that match must start with a given domino, say the first one.

We define a reduction from A_{TM} to (M)PCP. Let an instance of A_{TM} be

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, q_r)$
- $w = w_1 \ldots w_n$

Proof of Undecidability of PCP

Simplifying assumptions

- Assume that the tape of TM is one-way infinite and never attempts to move left of its left-end.
- If $w = \varepsilon$, then use $\sqcup$ instead of w.
- $P = \left\{ \begin{bmatrix} t_1 \\ b_1 \end{bmatrix}, \begin{bmatrix} t_2 \\ b_2 \end{bmatrix}, \ldots, \begin{bmatrix} t_k \\ b_k \end{bmatrix} \right\}$
- Modify PCP so that match must start with a given domino, say the first one.

We define a reduction from A_{TM} to (M)PCP. Let an instance of A_{TM} be

- $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, q_r)$
- $w = w_1 \ldots w_n$

We build instance P' of MPCP in several steps.

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \# q_0 w_1 w_2 \ldots w_n \# \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \#q_0 w_1 w_2 \ldots w_n \# \end{bmatrix}$$

- Step 2: Encode transitions of TM into dominos.

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \# q_0 w_1 w_2 \dots w_n \# \end{bmatrix}$$

- Step 2: Encode transitions of TM into dominos.
- For every $a, b, c \in \Gamma$ and every $q, q' \in Q$, $q \neq q_r$,
  - If $\delta(q, a) = (q', b, R)$, then add domino to P'

Proof of Undecidability of PCP

- Step 1: fix first domino in P'
$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \#q_0 w_1 w_2 \dots w_n \# \end{bmatrix}$$
- Step 2: Encode transitions of TM into dominos.
- For every $a, b, c \in \Gamma$ and every $q, q' \in Q$, $q \neq q_r$,
  - If $\delta(q, a) = (q', b, R)$, then add domino to P'
$$\begin{bmatrix} qa \\ bq' \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \# q_0 w_1 w_2 \dots w_n \# \end{bmatrix}$$

- Step 2: Encode transitions of TM into dominos.

- For every $a, b, c \in \Gamma$ and every $q, q' \in Q$, $q \neq q_r$,
  - If $\delta(q, a) = (q', b, R)$, then add domino to P'

$$\begin{bmatrix} qa \\ bq' \end{bmatrix}$$

  - If $\delta(q, a) = (q', b, L)$, then add domino to P'

Proof of Undecidability of PCP

- Step 1: fix first domino in P'
$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \#q_0 w_1 w_2 \dots w_n \# \end{bmatrix}$$
- Step 2: Encode transitions of TM into dominos.
- For every $a, b, c \in \Gamma$ and every $q, q' \in Q$, $q \neq q_r$,
  - If $\delta(q, a) = (q', b, R)$, then add domino to P'
$$\begin{bmatrix} qa \\ bq' \end{bmatrix}$$
  - If $\delta(q, a) = (q', b, L)$, then add domino to P'
$$\begin{bmatrix} cqa \\ q'cb \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 1: fix first domino in P'

$$\begin{bmatrix} t_1 \\ b_1 \end{bmatrix} = \begin{bmatrix} \# \\ \# q_0 w_1 w_2 \dots w_n \# \end{bmatrix}$$

- Step 2: Encode transitions of TM into dominos.

- For every $a, b, c \in \Gamma$ and every $q, q' \in Q$, $q \neq q_r$,
  - If $\delta(q,a) = (q', b, R)$, then add domino to P'
    $$\begin{bmatrix} qa \\ bq' \end{bmatrix}$$
  - If $\delta(q,a) = (q', b, L)$, then add domino to P'
    $$\begin{bmatrix} cqa \\ q'cb \end{bmatrix}$$
  - add all dominos (i.e, for all $a \in \Gamma \cup \{\#\}$ to P'.
    $$\begin{bmatrix} a \\ a \end{bmatrix} \quad \begin{bmatrix} \# \\ \sqcup\# \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos
  - For every $a \in \Gamma$, we add all dominos to P'

$$\begin{bmatrix} q_a a \\ q_a \end{bmatrix} \quad \begin{bmatrix} a q_a \\ q_a \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos
  - For every $a \in \Gamma$, we add all dominos to P'

$$\begin{bmatrix} q_a a \\ q_a \end{bmatrix} \quad \begin{bmatrix} a q_a \\ q_a \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos
  - For every $a \in \Gamma$, we add all dominos to P'

$$\begin{bmatrix} q_a a \\ q_a \end{bmatrix} \quad \begin{bmatrix} a q_a \\ q_a \end{bmatrix}$$

- Step 4: Complete the match

$$\begin{bmatrix} q_a\#\# \\ \# \end{bmatrix}$$

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos
  - For every $a \in \Gamma$, we add all dominos to P'

$$\begin{bmatrix} q_a a \\ q_a \end{bmatrix} \quad \begin{bmatrix} a q_a \\ q_a \end{bmatrix}$$

- Step 4: Complete the match

$$\begin{bmatrix} q_a \# \# \\ \# \end{bmatrix}$$

- This completes the reduction, i.e., map from instance of A_{TM} to PCP.
- M accepts w if and only if P' gives a solution to PCP.

Proof of Undecidability of PCP

- Step 3: Acceptance in to eating dominos
  - For every $a \in \Gamma$, we add all dominos to P'

$$\begin{bmatrix} q_a a \\ q_a \end{bmatrix} \quad \begin{bmatrix} a q_a \\ q_a \end{bmatrix}$$

- Step 4: Complete the match

$$\begin{bmatrix} q_a \# \# \\ \# \end{bmatrix}$$

- This completes the reduction, i.e., map from instance of A_{TM} to PCP.
- M accepts w if and only if P' gives a solution to PCP.
- MPCP to PCP?