

CS21 Decidability and Tractability

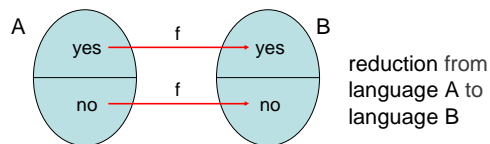
Lecture 13
February 6, 2008

Outline

- Post Correspondence Problem
- a natural non-RE and non-coRE language
- The Recursion Theorem
- Gödel Incompleteness Theorem

Definition of reduction

- More refined notion of reduction:
 - “many-one” reduction (commonly)
 - “mapping” reduction (book)



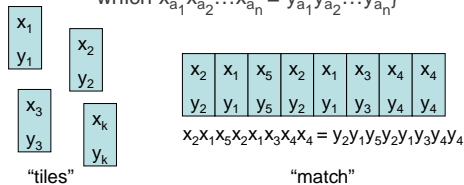
Post Correspondence Problem

- many undecidable problems unrelated to TMs and automata
- classic example: Post Correspondence Problem

PCP = $\{ \langle (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \rangle : x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \dots, a_n) \text{ for which } x_{a_1}x_{a_2}\dots x_{a_n} = y_{a_1}y_{a_2}\dots y_{a_n} \}$

Post Correspondence Problem

PCP = $\{ \langle (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \rangle : x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \dots, a_n) \text{ for which } x_{a_1}x_{a_2}\dots x_{a_n} = y_{a_1}y_{a_2}\dots y_{a_n} \}$



Post Correspondence Problem

Theorem: PCP is undecidable.

Proof:

– show $A_{TM} \leq_m \text{MPCP}$

MPCP = $\{ \langle (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \rangle : x_i, y_i \in \Sigma^* \text{ and there exists } (a_1, a_2, \dots, a_n) \text{ for which } x_{a_1}x_{a_2}\dots x_{a_n} = y_{a_1}y_{a_2}\dots y_{a_n} \}$

– show $\text{MPCP} \leq_m \text{PCP}$

Post Correspondence Problem

Proof of $A_{TM} \leq_m MPCP$:

- given instance of A_{TM} : $\langle M, w \rangle$
- idea: a match will record an accepting computation history for M on input w
- start tile records starting configuration:
 - add tile $(\#, \#q_0w_1w_2\dots w_n\#)$

$$\begin{array}{|c|} \hline \# \\ \hline \#q_0w_1w_2\dots w_n\# \\ \hline \end{array} = \begin{array}{|c|} \hline \# \\ \hline \#C_1\# \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

7

Post Correspondence Problem

$$\begin{array}{|c|} \hline \# \\ \hline \#q_0w_1w_2\dots w_n\# \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \dots \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} = \begin{array}{|c|} \hline \#C_1\# \\ \hline \#C_1\#C_2\# \\ \hline \end{array}$$

- tiles for head motions to the right:
 - for all $a, b \in \Gamma$ and all $q, r \in Q$ with $q \neq q_{reject}$, if $\delta(q, a) = (r, b, R)$, add tile (qa, br)
- tiles for head motions to the left:
 - for all $a, b, c \in \Gamma$ and all $q, r \in Q$ with $q \neq q_{reject}$, if $\delta(q, a) = (r, b, L)$, add tile (cqa, rcb)

$$\begin{array}{|c|} \hline qa \\ \hline br \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline cqa \\ \hline rcb \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

8

Post Correspondence Problem

$$\begin{array}{|c|} \hline \# \\ \hline \#q_0w_1w_2\dots w_n\# \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \dots \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} = \begin{array}{|c|} \hline \#C_1\# \\ \hline \#C_1\#C_2\# \\ \hline \end{array}$$

- tiles for copying (not near head)
 - for all $a \in \Gamma$, add tile (a, a)
- tiles for copying # marker
 - add tile $(\#, \#)$
- tiles for copying # marker and adding $_$ to end of tape
 - add tile $(\#, _ \#)$

$$\begin{array}{|c|} \hline a \\ \hline a \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline \# \\ \hline \# \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline \# \\ \hline _ \# \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

9

Post Correspondence Problem

$$\begin{array}{|c|} \hline \# \\ \hline \#uq_{accept}v\# \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \dots \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} = \begin{array}{|c|} \hline \#uq_{accept}v\# \\ \hline \#uq_{accept}v\#uq_{accept}v\# \\ \hline \end{array}$$

- tiles for deleting symbols to left of q_{accept}
 - for all $a \in \Gamma$, add tile $(aq_{accept}, q_{accept})$

$$\begin{array}{|c|} \hline aq_{accept} \\ \hline q_{accept} \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

10

Post Correspondence Problem

$$\begin{array}{|c|} \hline \# \\ \hline \#uq_{accept}\# \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \dots \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} = \begin{array}{|c|} \hline \#uq_{accept}av\# \\ \hline \#uq_{accept}av\#uq_{accept}v\# \\ \hline \end{array}$$

- tiles for deleting symbols to right of q_{accept}
 - for all $a \in \Gamma$, add tile $(q_{accept}a, q_{accept})$

$$\begin{array}{|c|} \hline q_{accept}a \\ \hline q_{accept} \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

11

Post Correspondence Problem

$$\begin{array}{|c|} \hline \# \\ \hline \#q_{accept}\# \\ \hline \end{array} \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} \dots \begin{array}{|c|} \hline ? \\ \hline ? \\ \hline \end{array} = \begin{array}{|c|} \hline \#q_{accept}\#\# \\ \hline \#q_{accept}\#\# \\ \hline \end{array}$$

- tiles for completing the match
 - for all $a \in \Gamma$, add tile $(q_{accept}\#\#, \#)$

$$\begin{array}{|c|} \hline q_{accept}\#\# \\ \hline \# \\ \hline \end{array}$$

February 6, 2008

CS21 Lecture 13

12

Post Correspondence Problem

– YES maps to YES?

- by construction, if M accepts w , there is a way to assemble the tiles to achieve this match:

#C₁#C₂#C₃#...#C_m#
#C₁#C₂#C₃#...#C_m#

where #C₁#C₂#C₃#...#C_m# is an accepting computation history

– NO maps to NO?

- sketch: at any step if the “intended” next tile is not used, then it is impossible to recover and produce a match in the end (case analysis)

February 6, 2008

CS21 Lecture 13

13

Post Correspondence Problem

We have proved:

Theorem: PCP is undecidable.

by showing:

- $A_{TM} \leq_m \text{MPCP}$
- $\text{MPCP} \leq_m \text{PCP}$
- conclude $A_{TM} \leq_m \text{PCP}$

February 6, 2008

CS21 Lecture 13

14

Beyond RE and co-RE

- We saw (by a counting argument) that there is/ isn't some L that is RE or co-RE .
Therefore, not in co-RE
Therefore, not in RE
- We will prove this for a natural language:
 $EQ_{TM} = \{ \langle M_1, M_2 \rangle : L(M_1) = L(M_2) \}$
- Recall:
 - A_{TM} is undecidable, but RE
 - $\text{co-}A_{TM}$ is undecidable, but coRE

February 6, 2008

CS21 Lecture 13

15

Beyond RE and co-RE

Theorem: EQ_{TM} is neither RE nor coRE.

Proof:

- not RE:
 - reduce from $\text{co-}A_{TM}$ (i.e. show $\text{co-}A_{TM} \leq_m EQ_{TM}$)
 - what should $f(\langle M, w \rangle)$ produce?
- not co-RE:
 - reduce from A_{TM} (i.e. show $A_{TM} \leq_m EQ_{TM}$)
 - what should $f(\langle M, w \rangle)$ produce?

February 6, 2008

CS21 Lecture 13

16

Beyond RE and co-RE

Proof ($A_{TM} \leq_m EQ_{TM}$)

– $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$ described below:

TM M_1 : on input x ,
• accept
TM M_2 : on input x ,
• simulate M on input w
• accept if M accepts w

• YES maps to YES?

$\langle M, w \rangle \in A_{TM} \Rightarrow$
 $L(M_1) = \Sigma^*, L(M_2) = \Sigma^* \Rightarrow$
 $f(\langle M, w \rangle) \in EQ_{TM}$

• NO maps to NO?

$\langle M, w \rangle \notin A_{TM} \Rightarrow$
 $L(M_1) = \Sigma^*, L(M_2) = \emptyset \Rightarrow$
 $f(\langle M, w \rangle) \notin EQ_{TM}$

February 6, 2008

CS21 Lecture 13

17

Beyond RE and co-RE

Proof ($\text{co-}A_{TM} \leq_m EQ_{TM}$)

– $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$ described below:

TM M_1 : on input x ,
• reject
TM M_2 : on input x ,
• simulate M on input w
• accept if M accepts w

• YES maps to YES?

$\langle M, w \rangle \in \text{co-}A_{TM} \Rightarrow$
 $L(M_1) = \emptyset, L(M_2) = \emptyset \Rightarrow$
 $f(\langle M, w \rangle) \in EQ_{TM}$

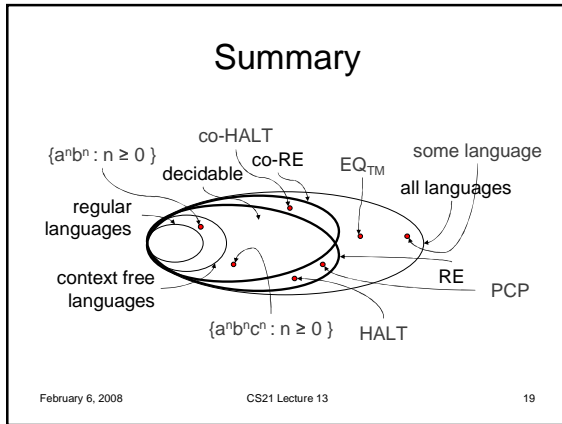
• NO maps to NO?

$\langle M, w \rangle \notin \text{co-}A_{TM} \Rightarrow$
 $L(M_1) = \emptyset, L(M_2) = \Sigma^* \Rightarrow$
 $f(\langle M, w \rangle) \notin EQ_{TM}$

February 6, 2008

CS21 Lecture 13

18



The Recursion Theorem

- A very useful, and non-obvious, capability of Turing Machines:
 - in the course of computation, can print out a description of itself!
- how is this possible?
 - an example of a program that prints out self:
 Print two copies of the following, the 2nd one in quotes:
 "Print two copies of the following, the 2nd one in quotes:"

February 6, 2008 CS21 Lecture 13 20

The Recursion Theorem

- Why is this useful?
- Example: slick proof that A_{TM} undecidable
 - assume TM M decides A_{TM}
 - construct machine M' as follows:

on input x , <ul style="list-style-type: none"> • obtain own description $\langle M' \rangle$ • run M on input $\langle M', x \rangle$ • if M rejects, accept; if M accepts, reject. 	if M' on input x : <ul style="list-style-type: none"> • accepts, then M rejects $\langle M', x \rangle$, but then M' does not accept! • rejects, then M accepts $\langle M', x \rangle$, but then M' accepts!
--	---

February 6, 2008 CS21 Lecture 13 21

The Recursion Theorem

- Lemma: there is a computable function $q: \Sigma^* \rightarrow \Sigma^*$ such that $q(w)$ is a description of a TM P_w that prints out w and then halts.
- Proof:
 - on input w , construct TM P_w that has w hard-coded into it; output $\langle P_w \rangle$

February 6, 2008 CS21 Lecture 13 22

The Recursion Theorem

- Warm-up: produce a TM SELF that prints out its own description.
- Two parts:
 - Part A:
 - output a description of B
 - pass control to B .
 - Part B:
 - prepend a description of A
 - done

February 6, 2008 CS21 Lecture 13 23

The Recursion Theorem

- Part A:
 - output a description of B
 - pass control to B .
- Part B:
 - prepend a description of A
 - done

Recall: $q(w)$ is a description of a TM P_w that prints out w and then halts.

Note: $\langle A \rangle = q(\langle B \rangle)$ <div style="border: 1px solid black; padding: 5px; margin: 5px;"> A • output $\langle B \rangle$ </div>	<div style="border: 1px solid black; padding: 5px; margin: 5px;"> B • read contents of tape • apply q to it • prepend result to tape </div>
--	--

February 6, 2008 CS21 Lecture 13 24

The Recursion Theorem

Note: $\langle A \rangle = q(\langle B \rangle)$

A

- output $\langle B \rangle$

Recall: $q(w)$ is a description of a TM P_w that prints out w and then halts.

B

- read contents of tape
- apply q to it
- prepend result to tape

- watch closely as TM AB runs:
- A runs. Tape contents: $\langle B \rangle$
- B runs. Tape contents: $q(\langle B \rangle)\langle B \rangle = \langle AB \rangle$
- AB is our desired machine SELF.

February 6, 2008

CS21 Lecture 13

25

The Recursion Theorem

Theorem: Let T be a TM that computes fn:

$$t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

There is a TM R that computes the fn:

$$r: \Sigma^* \rightarrow \Sigma^*$$

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

- This allows “obtain own description” as valid step in TM program
 - first modify TM so that it takes an additional input (that is own description); use at will

February 6, 2008

CS21 Lecture 13

26

The Recursion Theorem

Theorem: Let T be a TM that computes fn:

$$t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

There is a TM R that computes the fn:

$$r: \Sigma^* \rightarrow \Sigma^*$$

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

Proof outline: TM R has 3 parts

- Part A: output description of BT
- Part B: prepend description of A
- Part “T”: run TM T

February 6, 2008

CS21 Lecture 13

27

The Recursion Theorem

Proof details: TM R has 3 parts

Part A: output description of BT

- $\langle A \rangle = q(\langle BT \rangle)$

Part B: prepend description of A

- read contents of tape $\langle BT \rangle$
- apply q to it $q(\langle BT \rangle) = \langle A \rangle$
- prepend to tape $\langle ABT \rangle$

Part “T”: run TM T

- 2nd argument on tape is description of R

February 6, 2008

CS21 Lecture 13

28