CS21
Decidability and Tractability

Lecture 14

February 7, 2014

Outline

- Post Correspondence problem
- a non-RE and non-co-RE language
- the Recursion Theorem
- Gödel Incompleteness Theorem

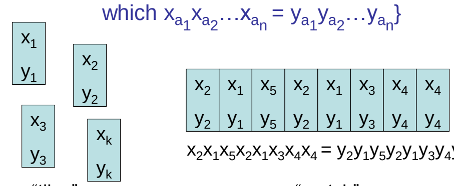
Post Correspondence Problem

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

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

Post Correspondence Problem

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



$x_2x_1x_5x_2x_1x_3x_4x_4 = y_2y_1y_5y_2y_1y_3y_4y_4$

"tiles" "match"

Post Correspondence Problem

<u>**Theorem**</u>: PCP is undecidable.

Proof:

- reduce from A_{TM} (i.e. show $A_{TM} \leq_m$ PCP)
- two step reduction makes it easier
- first, show $A_{TM} \leq_m$ MPCP

  (MPCP = "modified PCP")
- next, show MPCP $\leq_m$ PCP

Post Correspondence Problem

MPCP = {<$(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$> : $x_i, y_i \in \Sigma^*$ and there exists $(a_1, a_2, \ldots, a_n)$ for which $x_1x_{a_1}x_{a_2}\ldots x_{a_n} = y_1y_{a_1}y_{a_2}\ldots y_{a_n}$}

Proof of MPCP $\leq_m$ PCP:

- notation: for a string $u = u_1u_2u_3\ldots u_m$
  - $*u$ means the string $*u_1*u_2*u_3*u_4\ldots*u_m$
  - $u*$ means the string $u_1*u_2*u_3*u_4\ldots*u_m*$
  - $*u*$ means the string $*u_1*u_2*u_3*u_4\ldots*u_m*$

Post Correspondence Problem

Proof of MPCP $\leq_m$ PCP:

- given an instance $(x_1, y_1), \ldots, (x_k, y_k)$ of MPCP
- produce an instance of PCP:
  $(*x_1, *y_1*)$, $(*x_1, y_1*)$, $(*x_2, y_2*)$, …, $(*x_k, y_k*)$, $(*\blacklozenge, \blacklozenge)$
- YES maps to YES?
  - given a match in original MPCP instance, can produce a match in the new PCP instance
- NO maps to NO?
  - given a match in the new PCP instance, can produce a match in the original MPCP instance

Post Correspondence Problem

- YES maps to YES?
  - given a match in original MPCP instance, can produce a match in the new PCP instance

x_1	x_4	x_5	x_2	x_1	x_3	x_4	x_4
y_1	y_4	y_5	y_2	y_1	y_3	y_4	y_4

$*x_1$	$*x_4$	$*x_5$	$*x_2$	$*x_1$	$*x_3$	$*x_4$	$*x_4$	$*\blacklozenge$
$*y_1*$	y_4*	y_5*	y_2*	y_1*	y_3*	y_4*	y_4*	$\blacklozenge$

Post Correspondence Problem

can't match unless start with this tile

- NO maps to NO?
  - given a match in the new PCP instance, can produce a match in the original MPCP instance

$*x_1$	$*x_4$	$*x_5$	$*x_2$	$*x_1$	$*x_3$	$*x_4$	$*x_4$	$*\blacklozenge$
$*y_1*$	y_4*	y_5*	y_2*	y_1*	y_3*	y_4*	y_4*	$\blacklozenge$

x_1	x_4	x_5	x_2	x_1	x_3	x_4	x_4
y_1	y_4	y_5	y_2	y_1	y_3	y_4	y_4

"*" symbols must align

can only appear at the end

Post Correspondence Problem

Theorem: PCP is undecidable.

Proof:

- show $A_{TM} \leq_m$ MPCP

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

- show MPCP $\leq_m$ PCP

Post Correspondence Problem

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

- given instance of A_{TM}: <M, w>
- idea: a match will record an accepting computation history for M on input w
- start tile records starting configuration:
  - add tile $(\#, \#q_0 w_1 w_2 w_3 \ldots w_n\#)$

#		=	#
$\#q_0 w_1 w_2 \ldots w_n\#$			$\#C_1\#$

Post Correspondence Problem

#	?	?	…	?		=	$\#C_1\#$
$\# q_0 w_1 w_2 \ldots w_n\#$	?	?		?			$\#C_1\#C_2\#$

- 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)

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)

cqa
rcb

Post Correspondence Problem

$\left[\frac{\#}{\#q_0w_1w_2\ldots w_n\#}\right]\left[\frac{?}{?}\right]\left[\frac{?}{?}\right]\ldots\left[\frac{?}{?}\right] = \left[\frac{\#C_1\#}{\#C_1\#C_2\#}\right]$

- 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 (#, _#)

$\left[\frac{a}{a}\right]$ $\left[\frac{\#}{\#}\right]$ $\left[\frac{\#}{\_\#}\right]$

Post Correspondence Problem

$\left[\frac{\#}{\#uaq_{accept}v\#}\right]\left[\frac{?}{?}\right]\ldots\left[\frac{?}{?}\right] = \left[\frac{\#uaq_{accept}v\#}{\#uaq_{accept}v\#uq_{accept}v\#}\right]$

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

$\left[\frac{aq_{accept}}{q_{accept}}\right]$

Post Correspondence Problem

$\left[\frac{\#}{\#q_{accept}av\#}\right]\left[\frac{?}{?}\right]\ldots\left[\frac{?}{?}\right] = \left[\frac{\#q_{accept}av\#}{\#q_{accept}av\#q_{accept}v\#}\right]$

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

$\left[\frac{q_{accept}a}{q_{accept}}\right]$

Post Correspondence Problem

$\left[\frac{\#}{\#q_{accept}\#}\right]\left[\frac{?}{?}\right]\ldots\left[\frac{?}{?}\right] = \left[\frac{\#q_{accept}\#\#}{\#q_{accept}\#\#}\right]$

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

$\left[\frac{q_{accept}\#\#}{\#}\right]$

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:

$\left[\frac{\#C_1\#C_2\#C_3\#\ldots\#C_m\#}{\#C_1\#C_2\#C_3\#\ldots\#C_m\#}\right]$ where $\#C_1\#C_2\#C_3\#\ldots\#C_m\#$ 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)

Post Correspondence Problem

We have proved:

Theorem: PCP is undecidable.

by showing:

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

Beyond RE and co-RE

- We saw (by a counting argument) that there is *som* Therefore, not in co-RE that is Therefore, not in RE or co-RE.
- We will prove this for a natural language:
  $EQ_{TM} = \{<M_1, M_2> : L(M_1) = L(M_2)\}$
- Recall:
  - A_{TM} is undecidable, but RE
  - co-A_{TM} is undecidable, but coRE

Beyond RE and co-RE

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

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

Beyond RE and co-RE

Proof ($A_{TM} \leq_m EQ_{TM}$)
- f(<M, w>) = $<M_1, M_2>$ 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?
  - $<M, w> \in A_{TM} \Rightarrow$ $L(M_1) = \Sigma^*$, $L(M_2) = \Sigma^*$, $\Rightarrow$ $f(<M, w>) \in EQ_{TM}$
- NO maps to NO?
  - $<M, w> \notin A_{TM} \Rightarrow$ $L(M_1) = \Sigma^*$, $L(M_2) = \emptyset \Rightarrow$ $f(<M, w>) \notin EQ_{TM}$

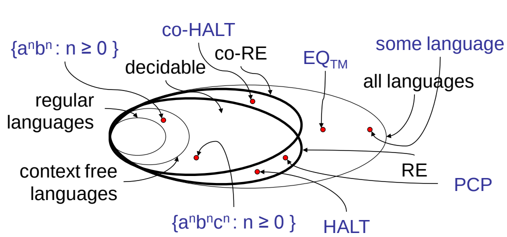
Beyond RE and co-RE

Proof (co-$A_{TM} \leq_m EQ_{TM}$)
- f(<M, w>) = $<M_1, M_2>$ 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?
  - $<M, w> \in$ co-$A_{TM} \Rightarrow$ $L(M_1) = \emptyset$, $L(M_2) = \emptyset \Rightarrow$ $f(<M, w>) \in EQ_{TM}$
- NO maps to NO?
  - $<M, w> \notin$ co-$A_{TM} \Rightarrow$ $L(M_1) = \emptyset$, $L(M_2) = \Sigma^*$, $\Rightarrow$ $f(<M, w>) \notin EQ_{TM}$

Summary



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:"

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,

- obtain own description <M'>
- run M on input <M', x>
- if M rejects, accept; if M accepts, reject.

if M' on input x:

- accepts, then M rejects <M', x>, but then M' does not accept!
- rejects, then M accepts <M', x>, but then M' accepts!

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 <P_w>

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

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: <A> = q(<B>)

A

- output <B>

B

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

*combine with description on tape to produce a complete TM

The Recursion Theorem

Note: <A> = q(<B>)

A

- output <B>

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: <B>
- B runs. Tape contents: q(<B>)<B> $\Rightarrow$ <AB>
- AB is our desired machine SELF.

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, <R>).

- 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

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, <R>).

Proof outline: TM R has 3 parts

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

The Recursion Theorem

Proof details: TM R has 3 parts

- Part A: output description of BT
  - <A> = q(<BT>)
- Part B: prepend description of A
  - read contents of tape <BT>
  - apply q to it q(<BT>) = <A>
  - prepend to tape <ABT>
- Part “T”: run TM T
  - 2nd argument on tape is description of R