CS5371
Theory of Computation

Lecture 15: Computability VI
(Post's Problem, Reducibility)

Objectives

- In this lecture, we introduce Post's correspondence problem (playing with a special type of domino)
- We also introduce computable functions, which allows us to look at reduction in a formal way

Post's Correspondence Problem

Let P be a finite set of dominoes $\{d_1, d_2, \ldots, d_k\}$, each piece of domino d_i consists of a top string t_i and a bottom string b_i

[We assume both top and bottom strings are nonempty]

An example set of dominoes:

123	132	32	2	23
2	3	321	321	3

Post's Correspondence Problem

A match in P is a sequence $i_1, i_2, \ldots, i_j$ (allowing repeats) such that $t_{i_1} t_{i_2} \ldots t_{i_j} = b_{i_1} b_{i_2} \ldots b_{i_j}$

- That is, we can find a sequence of dominoes such that the concatenation of top strings equals the concatenation of bottom strings

E.g., a match using the previous set P :

32	123	2	13
321	2	321	3

Post's Correspondence (2)

Let PCP be the language

$\{\langle P \rangle \mid P$ is a set of dominoes with a match$\}$

Theorem: PCP is undecidable

Post's Correspondence (3)

Before we do that, let us study a related problem:

Let MPCP be the language

$\{\langle P \rangle \mid P$ is a set of dominoes with a match starting with the first domino$\}$

Theorem: MPCP is undecidable

Proving MPCP

Proof Idea:

Prove by reducing A_{TM} to MPCP.

I.e., assume MPCP is decidable, show A_{TM} is decidable.

On input M, w, let us design a set P of dominoes, such that

M accepts w $\Leftrightarrow$ there is a match in P

In particular,

M on w has an accepting computation

$\Leftrightarrow$ there is a match in P

Proving MPCP (2)

How can we design a finite set of dominoes in P, so that it is flexible enough to represent M's computation sequence?

Difficulty: We cannot know M's computation sequence other than running M! How do we know what kind of dominoes we need to prepare in advance?

Proving MPCP (3)

Observation:

From a configuration to the next one,

(1) the change is "local", only around the tape head, and

(2) number of possible changes are finite

Proving MPCP (4)

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ and

$w = w_1 w_2 \ldots w_n$

(1) Design first domino:

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

(2) For all a,b in Γ, all q,r in Q ($q \neq q_{rej}$) create:

qa
br

if $\delta(q,a) = (r,b,R)$

Proving MPCP (5)

(3) For all a,b,c in Γ, all q,r in Q ($q \neq q_{rej}$), create:

$$\left[\frac{cqa}{rcb}\right] \quad \left[\frac{\#qa}{\#rb}\right] \quad \left[\frac{\#qa}{\square\#rb}\right]$$

if $\delta(q,a) = (r,b,L)$

The 2nd and 3rd dominoes are used when the tape head is at the leftmost end of tape

(4) For all a in Γ, create:

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

(5) Create:

$$\left[\frac{\#}{\#}\right] \quad \left[\frac{\#}{\square\#}\right]$$

Proving MPCP (6)

Suppose that starting configuration is something like: q_0 0101 and after the first transition, we obtain: $1q_3$ 101

BTW, what do we know about $\delta(q_0,0)$ here??

Then, at the beginning, we have:

#
$\#\#q_0w_1w_2\ldots w_n$

Proving MPCP (7)

Then, with the dominoes created so far, we can obtain a 'partial' match as follows:

#	#	$q_0 0$	1	0	1
##q_0 0101	#	1q_1	1	0	1

In the bottom string, the first configuration #q_0 0101 is matched, while at the same time we have created (uniquely) the next configuration #1q_1 101

Proving MPCP (8)

Applying the above idea repeatedly, we can obtain the 2nd configuration, the 3rd configuration, and so on …

- In particular, if M accepts w, the bottom string will eventually generate an accepting configuration
- On the other hand, if M does not accept w, the bottom string never generates an accepting configuration

Proving MPCP (9)

Now, it remains to create the remaining dominoes so that once we have obtained the accepting configuration (in the bottom string), we can guarantee to obtain a match

- In other words, there will be a match if and only if M accepts w

Proving MPCP (10)

(6) For all a in Γ, create:

aq_{acc}	$q_{acc}a$
q_{acc}	q_{acc}

(7) Create:

$q_{acc}\#\#$
$\#$

- Once we have an accepting configuration, we can apply the dominoes in (6) to 'simplify' the bottom strings, so that one character (adjacent to q_{acc}) is deleted in each subsequent configurations
- Eventually, the bottom string becomes $q_{acc}\#$ which can be matched with the domino in (7)

Proving MPCP (11)

Thus, MPCP has a match if and only if M accepts w.

Consequently, we can conclude that

MPCP is undecidable (Why?)

It remains to show PCP is undecidable …

We prove this by reducing MPCP to PCP

Reducing MPCP to PCP

We use a trick to transform any MPCP problem to a PCP problem, while enforcing the match must start with first domino

Before that, we introduce the following notation: for any string $u = u_1\ u_2 \ldots u_k$

$$*u* = * u_1 * u_2 * \ldots * u_k *$$

$$u* = u_1 * u_2 * \ldots * u_k *$$

$$*u = * u_1 * u_2 * \ldots * u_k$$

Reducing MPCP to PCP (2)

Let P' = $\{d_1, d_2, \ldots, d_k\}$ be the dominoes in MPCP, with d_i consisting a top string t_i and bottom string b_i, denoted by $d_i = t_i \mid b_i$. (d_1 is 1st domino)

We construct P as follows:

1. Add $*t_1 \mid *b_1*$ to P
2. Add $*t_j \mid b_j*$ to P, for every $j = 1,2,\ldots,k$
3. Add $*\blacklozenge \mid \blacklozenge$ to P

Note: Any match in P must start with $*t_1 \mid *b_1*$ and end with $*\blacklozenge \mid \blacklozenge$ to P (why?)

Reducing MPCP to PCP (3)

As an example, if P' is the dominoes in the MPCP for our previous A_{TM} problem, then, the first domino in P will be:

*#
##*q_0* 0*1*0*1*

And the 'partial' match becomes:

*#	*#	*q_0*0	*1	*0	*1
##*q_0* 0*1*0*1*	#*	1*q_1*	1*	0*	1*

Reducing MPCP to PCP (3)

And eventually, if there is a match in P' in our previous example, we have at the end:

* q_{acc} * # * #	*♦
# *	♦

In general, for all instances of MPCP,

P has a match

⇔ P' has a match starting with d_1

Formal Definition of Reduction

We have seen a lot of examples proved by reduction technique. Now, we give one formal definition of reduction

➔ This allows us to understand more about the power of reduction, and allows us to prove more results

The formal definition is based on computable functions (see next slides)

Computable Function

A TM can compute a function as follows: initially, the tape contains the input; once it halts, the tape contains the output

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some TM exists such that for any input w, it **halts** with $f(w)$ on its tape

E.g., all usual arithmetic operations on integers are computable functions

Defining Reduction

Definition: Language A is mapping reducible to language B, written as $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$ such that for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

Then, f is called the reduction of A to B

Some results

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable. (why?)

Corollary: If $A \leq_m B$ and A is undecidable, then B is undecidable.

Some results (2)

Theorem: If $A \leq_m B$ and B is recognizable, then A is recognizable (why?)

Corollary: If $A \leq_m B$ and A is non-recognizable, then B is non-recognizable

Examples ($HALT_{TM}$)

A long time ago, we showed how to reduce A_{TM} to $HALT_{TM}$. To show this by mapping reduction, we want to find a computable function f such that:

If $x = \langle M, w \rangle$,

$f(x)$ will be equal to $\langle M', w' \rangle$ such that

$$x \in A_{TM} \Leftrightarrow f(x) \in HALT_{TM}$$

Else, $f(x) = \varepsilon$ (or, pick any string not in $HALT_{TM}$)

Examples ($HALT_{TM}$)

Then, f can be computed by the TM F below:

F = "On input $\langle M, w\rangle$,

1. Construct the machine M':

   M' = "On input x

   1. Run M on x
   2. If M accepts, accept; Else, enter loop"

2. Output $\langle M', w\rangle$

Thus, f is a computable function, so that

$A_{TM} \leq_m HALT_{TM}$

Examples (PCP)

In PCP problem, we showed how to reduce A_{TM} to MPCP. To show this by mapping reduction, we want to find a computable function f such that:

If $x = \langle M, w \rangle$,

$f(x)$ will be equal to $\langle P \rangle$ such that

$$x \in A_{TM} \Leftrightarrow f(x) \in MPCP$$

Else, $f(x) = \varepsilon$ (or any $\langle P \rangle$ not in MPCP)

As we can find a TM that computes f (how?)

➔ $A_{TM} \leq_m MPCP$

Examples (PCP)

Also, we showed how to reduce MPCP to PCP

To show this by mapping reduction, we want to find a computable function g that:

If $x = \langle P \rangle$,

$g(x)$ will be equal to $\langle P' \rangle$ such that

$$x \in \text{MPCP} \iff g(x) \in \text{PCP}$$

Else, $g(x) = \varepsilon$

We can construct a TM that computes g, so that PCP $\leq_m$ PCP

Examples (PCP)

Combining the two examples, we can argue that the function $h = g \circ f$ is also a computable function (why?), and has the property that:

$$x \in A_{TM} \iff h(x) \in PCP \text{ (why?)}$$

Thus, $A_{TM} \leq_m PCP$, and we conclude that

PCP is undecidable

Examples (E_{TM})

When we show E_{TM} is undecidable, our proof is by reducing A_{TM} to E_{TM}

Let us recall how we do so:

On given any input $\langle M, w \rangle$, we construct a TM M' such that

if M accepts w, then $L(M') = \{w\}$

if M not accept w, then $L(M') = \{\ \}$

This in fact gives us a computable function f reducing A_{TM} to "complement of E_{TM}"

Examples (E_{TM})

I.e., we have a computable function f that:

$$x \in A_{TM} \Leftrightarrow f(x) \notin E_{TM},$$

or equivalently,

$$x \in A_{TM} \Leftrightarrow f(x) \in E'_{TM}$$

where E'_{TM} denotes the complement of E_{TM}

Thus, $A_{TM} \leq_m E'_{TM}$ → E'_{TM} is undecidable →
E_{TM} is undecidable (why?)

Question: Can we find a "direct" mapping reduction of A_{TM} to E_{TM} instead?

Examples (E_{TM})

… the answer is NO (Prob. 5.5), because:

Suppose on the contrary that $A_{TM} \leq_m E_{TM}$.

Then, we have $A'_{TM} \leq_m E'_{TM}$ (why?).

However, E'_{TM} is recognizable (why?)
but A'_{TM} is not recognizable
Thus, contradiction occurs (where?), so that
no reduction of A_{TM} to E_{TM} exists

Examples (EQ_{TM})

We have seen one example of a non-Turing recognizable language: A'_{TM}

Define: A language is co-recognizable if its complement is recognizable.

Then, we have:

Theorem: EQ_{TM} is not recognizable, and not co-recognizable. That is,

EQ_{TM} is not recognizable, and

EQ'_{TM} is not recognizable.

Examples (EQ_{TM})

Proof: We first show that $A_{TM} \leq_m EQ'_{TM}$.

If this can be shown, we equivalently has shown that $A'_{TM} \leq_m EQ_{TM}$ (why?) and proved that EQ_{TM} is not recognizable.

To show $A_{TM} \leq_m EQ'_{TM}$, we construct the TM F giving the desired reduction f as follows (see next slide):

Examples (EQ_{TM})

F = "On input $\langle M, w \rangle$,

1. Construct machines M_1 and M_2:

   M_1 = "On any input,
   1. Reject"

   M_2 = "On any input,
   1. Run M on w. If it accepts, accept"
2. Output $\langle M_1, M_2 \rangle$"

So, on input $x = \langle M, w \rangle$, F computes $\langle M_1, M_2 \rangle$ as $f(x)$. What is the property of $f(x)$?

Examples (EQ_{TM})

Next, we show that $A_{TM} \leq_m EQ_{TM}$.

If this can be shown, we equivalently has shown that $A'_{TM} \leq_m EQ'_{TM}$ and proved that EQ'_{TM} is not recognizable.

To show $A_{TM} \leq_m EQ_{TM}$, we construct the TM G giving the desired reduction g as follows (see next slide):

Examples (EQ_{TM})

G = "On input $\langle M, w \rangle$,

1. Construct machines M_1 and M_2:

   M_1 = "On any input,

   1. Accept"

   M_2 = "On any input,

   1. Run M on w. If it accepts, accept"

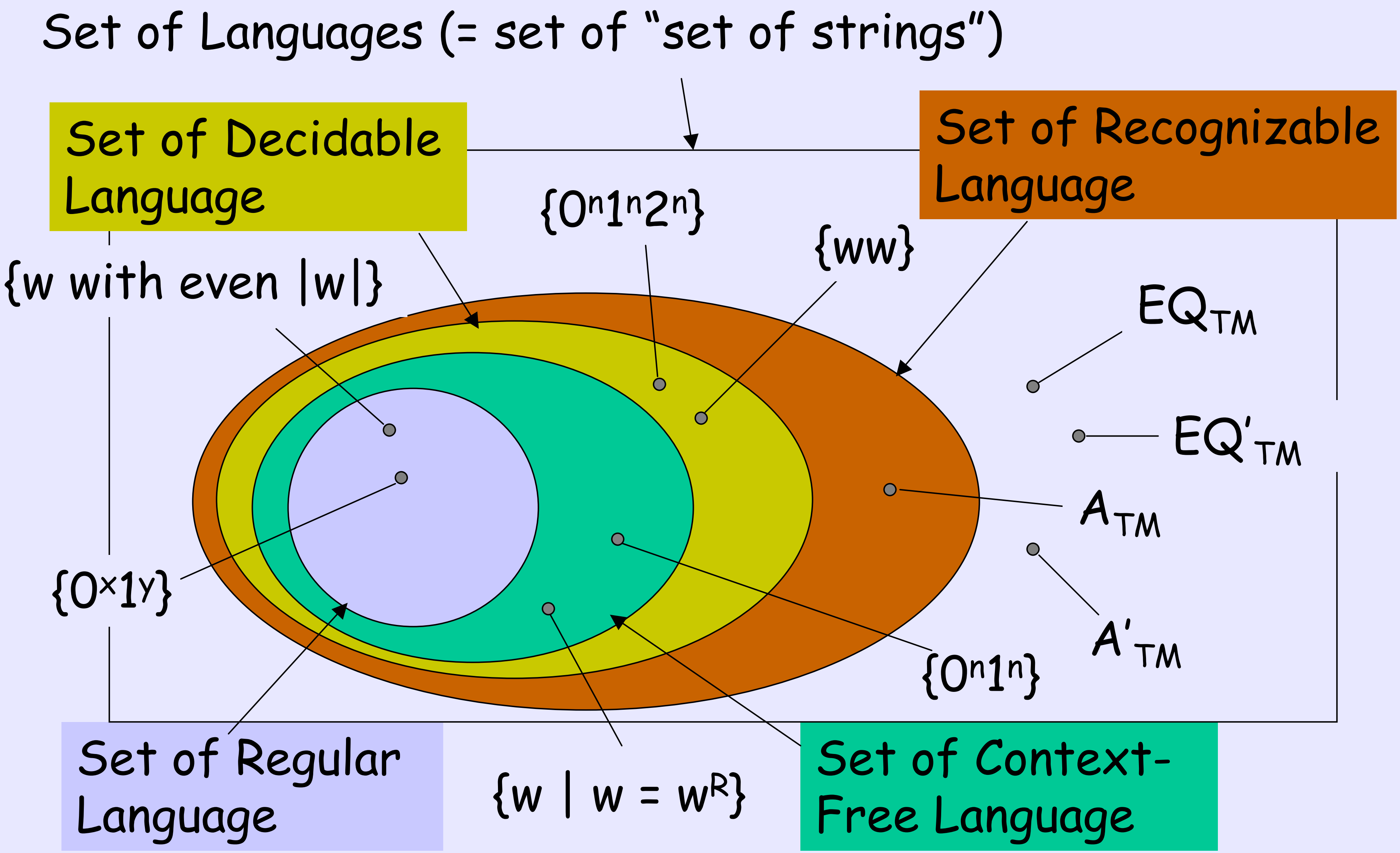
2. Output $\langle M1, M2 \rangle$"

So, on input $x = \langle M, w \rangle$, G computes $\langle M_1, M_2 \rangle$ as $g(x)$. What is the property of $g(x)$?

What we have learnt

- Reduction from A_{TM}:
  - $HALT_{TM}$, E_{TM}, $REGULAR_{TM}$, EQ_{TM}, PCP
- LBA
  - A_{LBA} is decidable (finite test cases)
  - E_{LBA} and ALL_{CFG} are undecidable (reduction from A_{TM} via computation history)
- Computable function, mapping reducibility
  - EQ_{TM} and EQ'_{TM} are non-recognizable (reduction from A'_{TM})

Language Hierarchy (revisited)



Next Time

- Complexity Theory
  - To classify the problems based on the resources (time or memory usage)