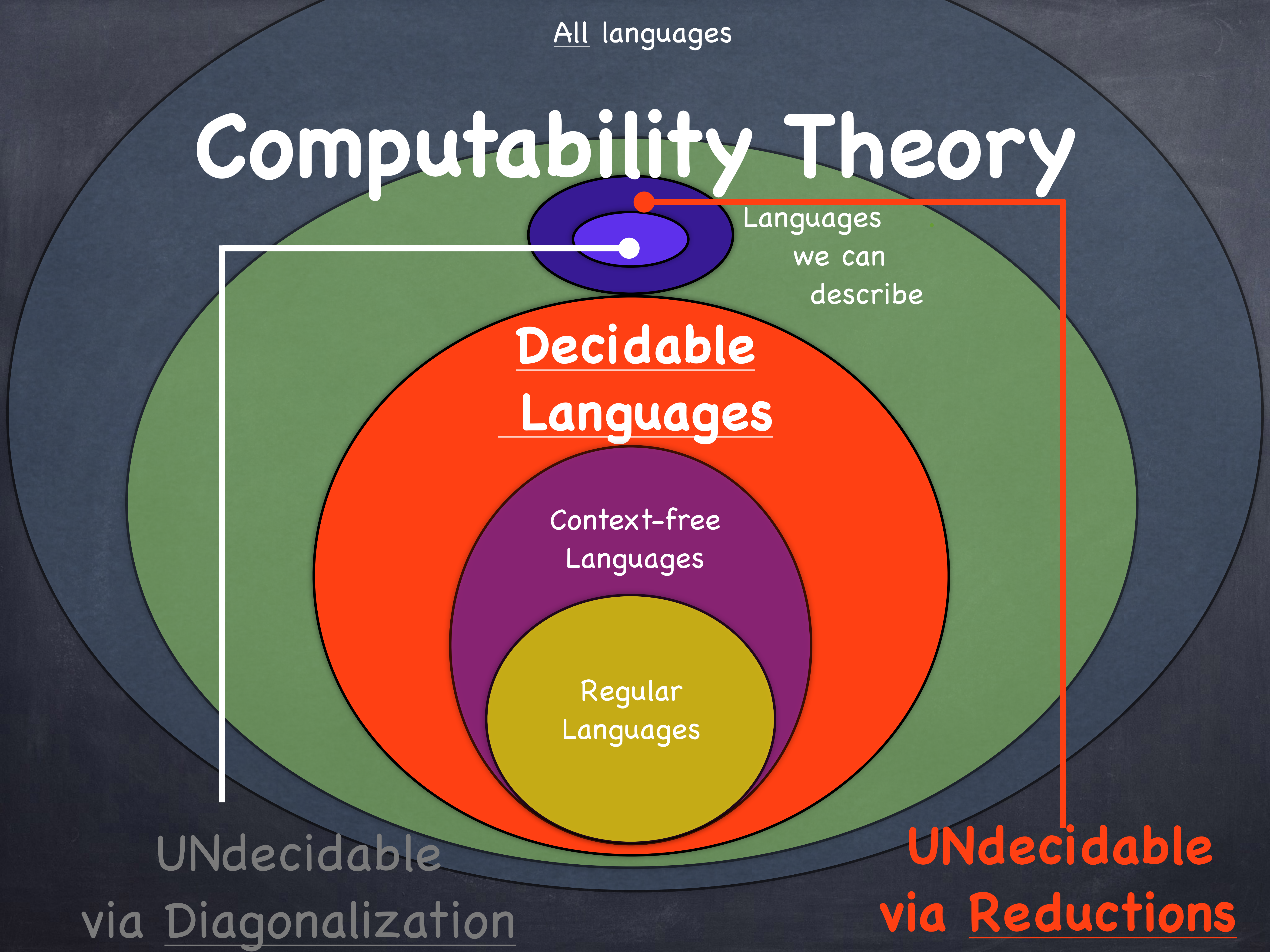
COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 20-21: Reducibility


All languages
Computability Theory
Languages
we can
describe
Decidable
Languages
Context-free
Languages
Regular
Languages
UNdecidable
via Diagonalization
UNdecidable
via Reductions

Reducibility

Decidable	Undecidable
A_{DFA}	EQ_{CFG}
A_{NFA}	A_{TM}
A_{REX}	$HALT_{TM}$
E_{DFA}	E_{TM}
EQ_{DFA}	$REGULAR_{TM}$
A_{CFG}	EQ_{TM}
E_{CFG}	PCP

Reducibility

Reducibility always involves two problems, which we call A and B. If A reduces to B, we can use a solution to B to solve A. So in our example, A is the problem of finding your way around the city and B is the problem of obtaining a map. Note that reducibility says nothing about solving A or B alone, but only about the solvability of A in the presence of a solution to B.

Reducibility

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}.$$

THEOREM 5.1

$HALT_{\mathsf{TM}}$ is undecidable.

Reducibility

PROOF Let's assume for the purposes of obtaining a contradiction that TM R decides $HALT_{\mathsf{TM}}$. We construct TM S to decide A_{TM}, with S operating as follows.

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Run TM R on input $\langle M, w \rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, *accept*; if M has rejected, *reject*."

Clearly, if R decides $HALT_{\mathsf{TM}}$, then S decides A_{TM}. Because A_{TM} is undecidable, $HALT_{\mathsf{TM}}$ also must be undecidable.

Reducibility

$$E_{\mathsf{TM}} = \{\langle M\rangle |\ M \text{ is a TM and } L(M) = \emptyset\}.$$

THEOREM 5.2

E_{TM} is undecidable.

Reducibility

PROOF Let's write the modified machine described in the proof idea using our standard notation. We call it M_1.

$M_1 =$ "On input x:

1. If $x \neq w$, *reject*.
2. If $x = w$, run M on input w and *accept* if M does."

This machine has the string w as part of its description. It conducts the test of whether $x = w$ in the obvious way, by scanning the input and comparing it character by character with w to determine whether they are the same.

Reducibility

Putting all this together, we assume that TM R decides E_{TM} and construct TM S that decides A_{TM} as follows.

$S =$ "On input $\langle M, w \rangle$, an encoding of a TM M and a string w:

1. Use the description of M and w to construct the TM M_1 just described.
2. Run R on input $\langle M_1 \rangle$.
3. If R accepts, *reject*; if R rejects, *accept*."

Note that S must actually be able to compute a description of M_1 from a description of M and w. It is able to do so because it needs only add extra states to M that perform the $x = w$ test.

If R were a decider for E_{TM}, S would be a decider for A_{TM}. A decider for A_{TM} cannot exist, so we know that E_{TM} must be undecidable.

Reducibility

$$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}.$$

THEOREM 5.3

$REGULAR_{\mathsf{TM}}$ is undecidable.

Reducibility

PROOF We let R be a TM that decides $REGULAR_{\mathsf{TM}}$ and construct TM S to decide A_{TM}. Then S works in the following manner.

$S =$ "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Construct the following TM M_2.

   $M_2 =$ "On input x:

   1. If x has the form $0^n 1^n$, *accept*.
   2. If x does not have this form, run M on input w and *accept* if M accepts w."

2. Run R on input $\langle M_2 \rangle$.
3. If R accepts, *accept*; if R rejects, *reject*."

$$\mathbf{L(M_2)} = \begin{cases} \{0^n 1^n \mid n \geq 0\} & \text{if } \mathbf{M} \text{ rejects } \mathbf{w} \\ \Sigma^* & \text{if } \mathbf{M} \text{ accepts } \mathbf{w} \end{cases}$$

Reducibility

$$EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}.$$

THEOREM 5.4

EQ_{TM} is undecidable.

Reducibility

PROOF We let TM R decide EQ_{TM} and construct TM S to decide E_{TM} as follows.

S = "On input $\langle M \rangle$, where M is a TM:

1. Run R on input $\langle M, M_1 \rangle$, where M_1 is a TM that rejects all inputs.
2. If R accepts, *accept*; if R rejects, *reject*."

If R decides EQ_{TM}, S decides E_{TM}. But E_{TM} is undecidable by Theorem 5.2, so EQ_{TM} also must be undecidable.

Reducibility

Decidable	Undecidable
A_{DFA}	EQ_{CFG}
A_{NFA}	A_{TM}
A_{REX}	$HALT_{TM}$
E_{DFA}	E_{TM}
EQ_{DFA}	$REGULAR_{TM}$
A_{CFG}	EQ_{TM}
E_{CFG}	PCP

Reducibility

ALL_{CFG}

Decidable	Undecidable
A_{DFA}	EQ_{CFG}
A_{NFA}	A_{TM}
A_{REX}	$HALT_{TM}$
E_{DFA}	E_{TM}
EQ_{DFA}	$REGULAR_{TM}$
A_{CFG}	EQ_{TM}
E_{CFG}	PCP

$MPCP$

Reducibility

5.28 Rice's theorem. Let P be any nontrivial property of the language of a Turing machine. Prove that the problem of determining whether a given Turing machine's language has property P is undecidable.

In more formal terms, let P be a language consisting of Turing machine descriptions where P fulfills two conditions. First, P is nontrivial—it contains some, but not all, TM descriptions. Second, P is a property of the TM's language—whenever $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$. Here, M_1 and M_2 are any TMs. Prove that P is an undecidable language.

Post Correspondence Problem

Post Correspondence Problem

Post Correspondence Problem

- In 1946, Emil Post gave a very natural example of an **undecidable** language…

Post Correspondence Problem

- In 1946, Emil Post gave a very natural example of an **undecidable** language...
- It is the "Post Correspondence Problem".

Post Correspondence Problem

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.
- A solution to **PCP**

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.
- A solution to **PCP**

aa
a

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.

- A solution to **PCP**

aa	bbb
a	a

Post Correspondence Problem

$$\left[\frac{aaa}{bb}\right] \left[\frac{a}{bb}\right] \left[\frac{bbb}{a}\right] \left[\frac{aa}{a}\right] \left[\frac{\quad}{bb}\right] \left[\frac{b}{\quad}\right]$$

- An instance of **PCP** with 6 dominos.

- A solution to **PCP**

$$\left[\frac{aa}{a}\right] \left[\frac{bbb}{a}\right] \left[\frac{b}{\quad}\right]$$

Post Correspondence Problem

$$\left[\frac{aaa}{bb}\right] \quad \left[\frac{a}{bb}\right] \quad \left[\frac{bbb}{a}\right] \quad \left[\frac{aa}{a}\right] \quad \left[\frac{\;}{bb}\right] \quad \left[\frac{b}{\;}\right]$$

- An instance of **PCP** with 6 dominos.
- A solution to **PCP**

$$\left[\frac{aa}{a}\right] \quad \left[\frac{bbb}{a}\right] \quad \left[\frac{b}{\;}\right] \quad \left[\frac{\;}{bb}\right]$$

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.

- A solution to **PCP**

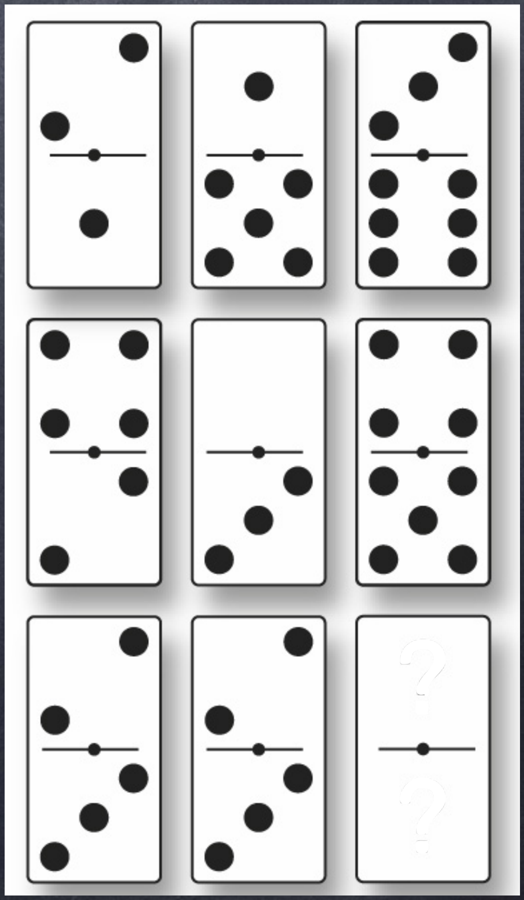
aa	bbb	b		
a	a		bb	bb

Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.
- A solution to **PCP**

aa	bbb	b		
a	a		bb	bb



Post Correspondence Problem

aaa	a	bbb	aa		b
bb	bb	a	a	bb	

- An instance of **PCP** with 6 dominos.

- A solution to **PCP**

aa	bbb	b		
a	a		bb	bb

Post Correspondence Problem

Post Correspondence Problem

- Given n dominos, $[u_1/v_1]$ … $[u_n/v_n]$ where each u_i or v_i is a string of symbols.

Post Correspondence Problem

- Given n dominos, $[u_1/v_1]$ … $[u_n/v_n]$ where each u_i or v_i is a string of symbols.
- Is there an integer k and a sequence $\langle i_1, i_2, i_3, \ldots, i_k \rangle$ (with each $1 \leq i_j \leq n$) s.t.

$$u_{i_1} \circ u_{i_2} \circ u_{i_3} \circ \ldots \circ u_{i_k} = v_{i_1} \circ v_{i_2} \circ v_{i_3} \circ \ldots \circ v_{i_k} ?$$

Post Correspondence Problem

$$\frac{u_1}{v_1} \quad \frac{u_2}{v_2} \quad \frac{u_3}{v_3} \quad \cdots \quad \frac{u_n}{v_n}$$

- Given n dominos, $[u_1/v_1]$ … $[u_n/v_n]$ where each u_i or v_i is a string of symbols.
- Is there an integer k and a sequence $\langle i_1, i_2, i_3, \ldots, i_k \rangle$ (with each $1 \le i_j \le n$) s.t.

$$u_{i_1} \circ u_{i_2} \circ u_{i_3} \circ \ \ldots \ \circ u_{i_k} = v_{i_1} \circ v_{i_2} \circ v_{i_3} \circ \ \ldots \ \circ v_{i_k} \ ?$$

A Solution to PCP

A Solution to PCP

$$\boxed{\frac{u_1}{v_1}} \quad \boxed{\frac{u_2}{v_2}} \quad \boxed{\frac{u_3}{v_3}} \quad \cdots \quad \boxed{\frac{u_n}{v_n}}$$

A Solution to PCP

$$\boxed{\frac{u_1}{v_1}} \quad \boxed{\frac{u_2}{v_2}} \quad \boxed{\frac{u_3}{v_3}} \quad \cdots \quad \boxed{\frac{u_n}{v_n}}$$

- A solution is of this form:

A Solution to PCP

$$\boxed{\frac{u_1}{v_1}} \quad \boxed{\frac{u_2}{v_2}} \quad \boxed{\frac{u_3}{v_3}} \quad \cdots \quad \boxed{\frac{u_n}{v_n}}$$

- A solution is of this form:

$$\boxed{\frac{u_{i_1}}{v_{i_1}}} \quad \boxed{\frac{u_{i_2}}{v_{i_2}}} \quad \boxed{\frac{u_{i_3}}{v_{i_3}}} \quad \boxed{\frac{u_{i_4}}{v_{i_4}}} \quad \boxed{\frac{u_{i_5}}{v_{i_5}}} \quad \cdots \quad \boxed{\frac{u_{i_k}}{v_{i_k}}}$$

A Solution to PCP

$$\frac{u_1}{v_1} \quad \frac{u_2}{v_2} \quad \frac{u_3}{v_3} \quad \cdots \quad \frac{u_n}{v_n}$$

- A solution is of this form:

$$\frac{u_{i_1}}{v_{i_1}} \quad \frac{u_{i_2}}{v_{i_2}} \quad \frac{u_{i_3}}{v_{i_3}} \quad \frac{u_{i_4}}{v_{i_4}} \quad \frac{u_{i_5}}{v_{i_5}} \quad \cdots \quad \frac{u_{i_k}}{v_{i_k}}$$

s.t.

$$u_{i_1} \circ u_{i_2} \circ u_{i_3} \circ \ \ldots \ \circ u_{i_k} = v_{i_1} \circ v_{i_2} \circ v_{i_3} \circ \ \ldots \ \circ v_{i_k} \ ?$$

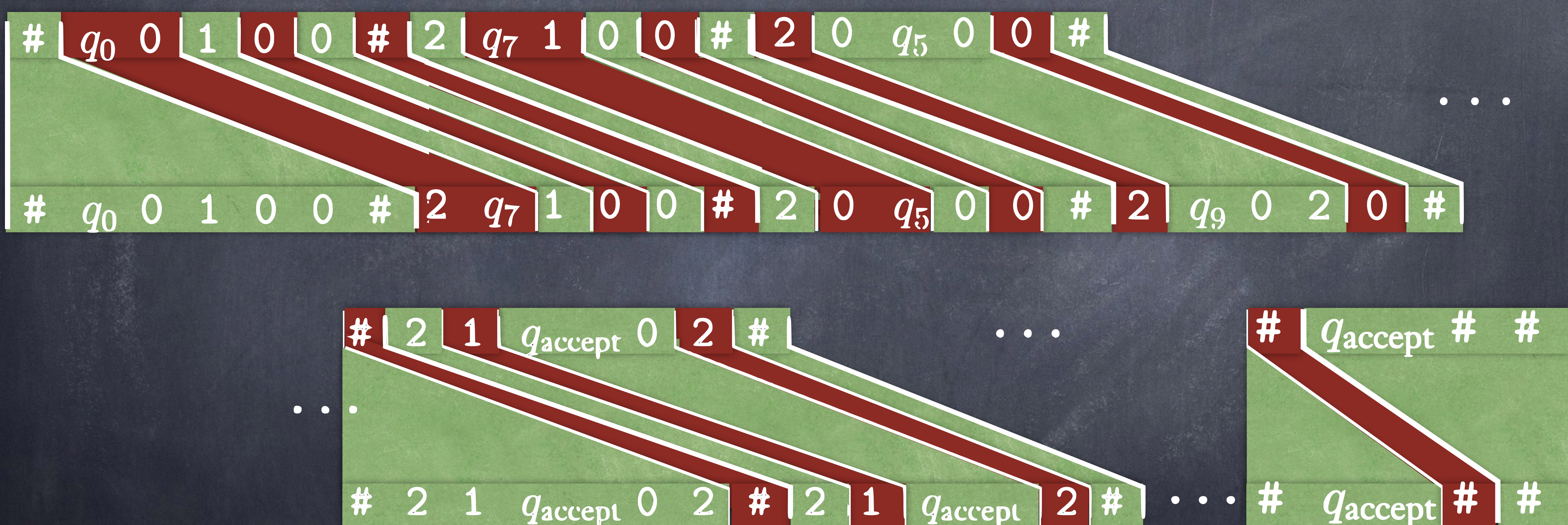
Post Correspondence Problem

Post Correspondence Problem

- **Theorem:**

  The Post Correspondence Problem cannot be **decided** by any algorithm (or computer program). In particular, no algorithm can identify in a finite amount of time some instances that have a **No** outcome. However, if a solution exists, we can find it. **PCP** is Turing-recognizable.

Reducing A_{TM} to MPCP a (mostly) complete example



Post Correspondence Problem

Post Correspondence Problem

- **Proof Idea**:

  Reduction - if **PCP** was **decidable** then the **ACCEPTANCE** problem would be **decidable** as well.

Computation History

DEFINITION 5.5

Let M be a Turing machine and w an input string. An ***accepting computation history*** for M on w is a sequence of configurations, $C_1, C_2, \ldots, C_l$, where C_1 is the start configuration of M on w, C_l is an accepting configuration of M, and each C_i legally follows from C_{i-1} according to the rules of M. A ***rejecting computation history*** for M on w is defined similarly, except that C_l is a rejecting configuration.

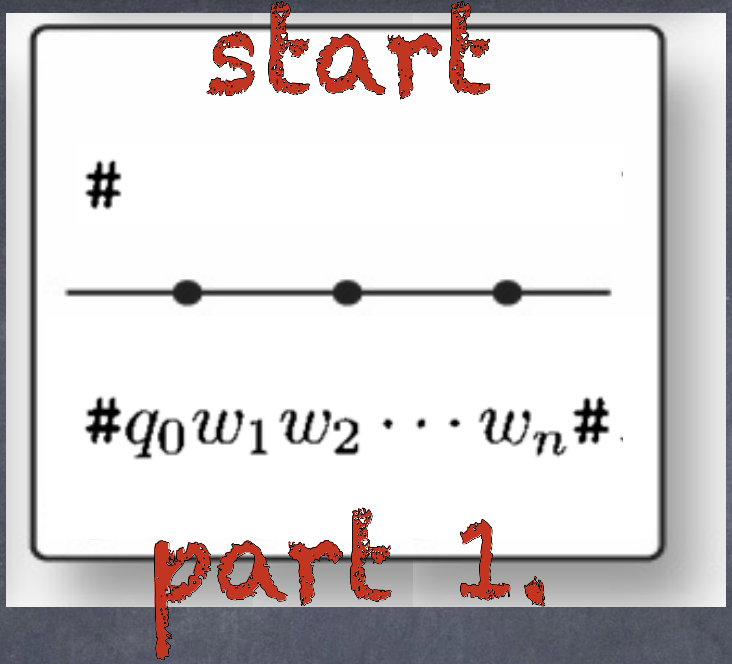
ATM :

A_{TM} : a Reduction to MPCP

A_{TM} : a Reduction to MPCP

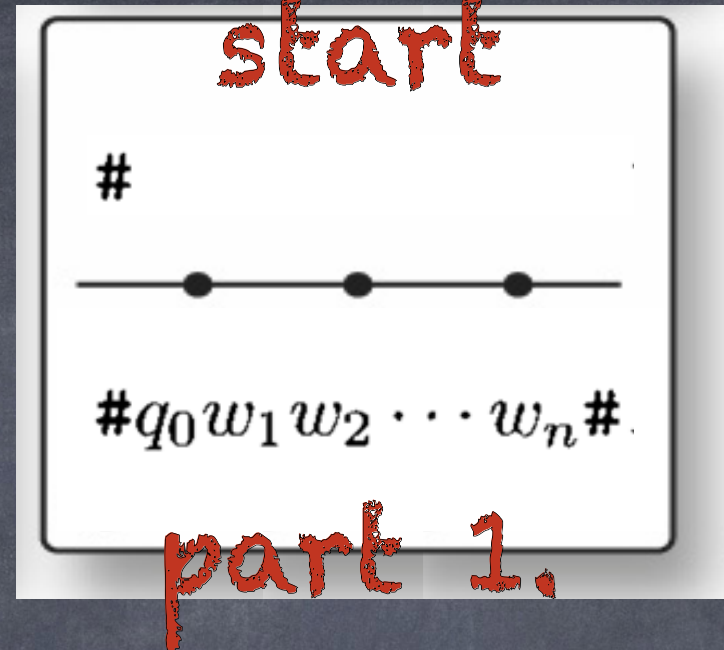
A story in seven parts

A_{TM} :
a Reduction
to MPCP

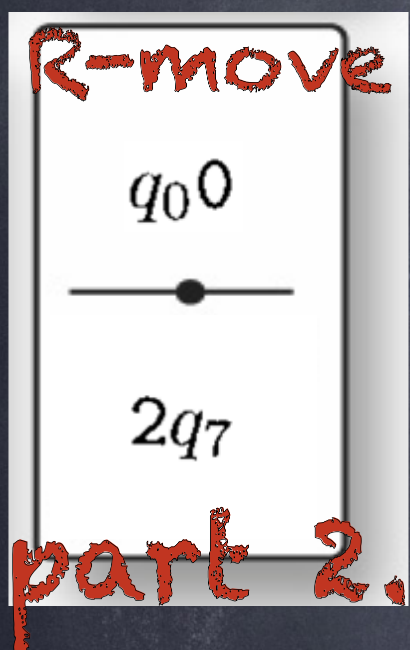


A story
in seven
parts

A_{TM} : a Reduction to MPCP



A story in seven parts



A_{TM} : a Reduction to MPCP

A story in seven parts

start

$\left[\dfrac{\#}{\#q_0w_1w_2\cdots w_n\#}\right]$

part 1.

R-move

$\left[\dfrac{q_00}{2q_7}\right]$

part 2.

L-move

$\left[\dfrac{0q_50}{q_902}\right]$

part 3.

...

$\left[\dfrac{2q_50}{q_922}\right]$

ATM : a Reduction to MPCP

A story in seven parts

start

$$\left[\frac{\#}{\#q_0w_1w_2\cdots w_n\#}\right]$$

part 1.

R-move

$$\left[\frac{q_00}{2q_7}\right]$$

part 2.

L-move

$$\left[\frac{0q_50}{q_902}\right]$$

part 3.

...

$$\left[\frac{2q_50}{q_922}\right]$$

copy

$$\left[\frac{0}{0}\right]$$

part 4.

...

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

A_{TM} : a Reduction to MPCP

A story in seven parts

start

$$\left[\frac{\#}{\#q_0w_1w_2\cdots w_n\#}\right]$$

part 1.

R-move

$$\left[\frac{q_0 0}{2q_7}\right]$$

part 2.

L-move

$$\left[\frac{0q_5 0}{q_9 02}\right]$$

part 3.

...

$$\left[\frac{2q_5 0}{q_9 22}\right]$$

copy

$$\left[\frac{0}{0}\right]$$

part 4.

...

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

blank

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

part 5.

$$\left[\frac{\#}{\sqcup\#}\right]$$

A_{TM} : a Reduction to MPCP

A story in seven parts

start

$\left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n \#}\right]$

part 1.

R-move

$\left[\frac{q_0 0}{2 q_7}\right]$

part 2.

L-move

$\left[\frac{0 q_5 0}{q_9 0 2}\right]$

part 3.

...

$\left[\frac{2 q_5 0}{q_9 2 2}\right]$

copy

$\left[\frac{0}{0}\right]$

part 4.

...

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

blank

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

part 5.

$\left[\frac{\#}{\sqcup\#}\right]$

clean

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

part 6.

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

A_{TM} : a Reduction to MPCP

A story in seven parts

start

$\left[\dfrac{\#}{\#q_0w_1w_2\cdots w_n\#}\right]$

part 1.

R-move

$\left[\dfrac{q_00}{2q_7}\right]$

part 2.

L-move

$\left[\dfrac{0q_50}{q_902}\right]$ … $\left[\dfrac{2q_50}{q_922}\right]$

part 3.

copy

$\left[\dfrac{0}{0}\right]$ … $\left[\dfrac{\sqcup}{\sqcup}\right]$

part 4.

blank

$\left[\dfrac{\#}{\#}\right]$ $\left[\dfrac{\#}{\sqcup\#}\right]$

part 5.

clean

$\left[\dfrac{a\,q_{accept}}{q_{accept}}\right]$ $\left[\dfrac{q_{accept}\,a}{q_{accept}}\right]$

part 6.

the end

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

part 7.

Reducing A_{TM} to MPCP

PROOF We let TM R decide the PCP and construct S deciding A_{TM}. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q, Σ, Γ, and δ, are the state set, input alphabet, tape alphabet, and transition function of M, respectively.

Reducing A_{TM} to MPCP

PROOF We let TM R decide the PCP and construct S deciding A_{TM}. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q, Σ, Γ, and δ, are the state set, input alphabet, tape alphabet, and transition function of M, respectively.

$$MPCP = \{\langle P\rangle \mid P \text{ is an instance of the Post correspondence problem with a match that starts with the first domino}\}.$$

Reducing A_{TM} to MPCP

PROOF We let TM R decide the PCP and construct S deciding A_{TM}. Let

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

where Q, Σ, Γ, and δ, are the state set, input alphabet, tape alphabet, and transition function of M, respectively.

$$MPCP = \{\langle P\rangle \mid P \text{ is an instance of the Post correspondence problem with a match that starts with the first domino}\}.$$

In this case S constructs an instance of the PCP P that has a match iff M accepts w. To do that S first constructs an instance P' of the MPCP. We describe the construction in seven parts, each of which accomplishes a particular aspect of simulating M on w. To explain what we are doing we interleave the construction with an example of the construction in action.

Part 1. The construction begins in the following manner.

$$\text{Put } \left[\frac{\#}{\#q_0w_1w_2\cdots w_n\#}\right] \text{ into } P' \text{ as the first domino } \left[\frac{t_1}{b_1}\right].$$

Because P' is an instance of the MPCP, the match must begin with this domino. Thus the bottom string begins correctly with $C_1 = q_0w_1w_2\cdots w_n$, the first configuration in the accepting computation history for M on w, as shown in the following figure.

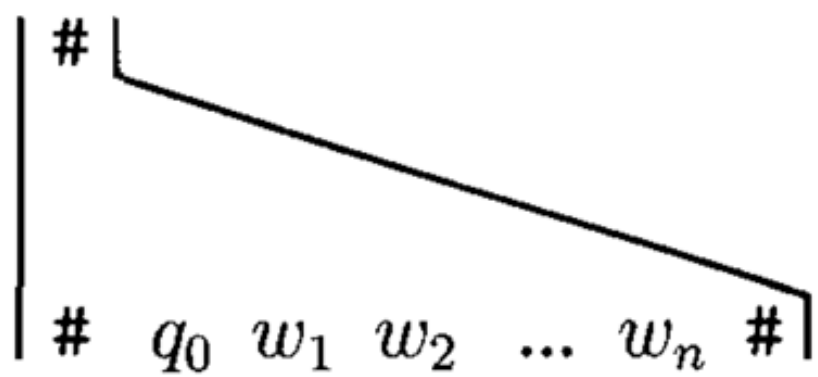


FIGURE 5.16
Beginning of the MPCP match

Reducing A_{TM} to MPCP

In this depiction of the partial match achieved so far, the bottom string consists of $\#q_0w_1w_2\cdots w_n\#$ and the top string consists only of #. To get a match we need to extend the top string to match the bottom string. We provide additional dominos to allow this extension. The additional dominos cause M's next configuration to appear at the extension of the bottom string by forcing a single-step simulation of M.

In parts 2, 3, and 4, we add to P' dominos that perform the main part of the simulation. Part 2 handles head motions to the right, part 3 handles head motions to the left, and part 4 handles the tape cells not adjacent to the head.

Reducing A_{TM} to MPCP

Part 2. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q,a) = (r, b, \mathrm{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P'.$$

Reducing A_{TM} to MPCP

Part 2. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \mathrm{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P'.$$

Part 3. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \mathrm{L}), \text{ put } \left[\frac{cqa}{rcb}\right] \text{ into } P'.$$

Reducing A_{TM} to MPCP

Part 2. For every $a, b \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \mathrm{R}), \text{ put } \left[\frac{qa}{br}\right] \text{ into } P'.$$

Part 3. For every $a, b, c \in \Gamma$ and every $q, r \in Q$ where $q \neq q_{\text{reject}}$,

$$\text{if } \delta(q, a) = (r, b, \mathrm{L}), \text{ put } \left[\frac{cqa}{rcb}\right] \text{ into } P'.$$

Part 4. For every $a \in \Gamma$,

$$\text{put } \left[\frac{a}{a}\right] \text{ into } P'.$$

Now we make up a hypothetical example to illustrate what we have built so far. Let $\Gamma = \{0, 1, 2, \sqcup\}$. Say that w is the string 0100 and that the start state of M is q_0. In state q_0, upon reading a 0, let's say that the transition function dictates that M enters state q_7, writes a 2 on the tape, and moves its head to the right. In other words, $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$.

Part 1 places the domino

$$\left[\frac{\#}{\#q_0 0100\#}\right] = \left[\frac{t_1}{b_1}\right]$$

in P', and the match begins:

In addition, part 2 places the domino

$$\left[\frac{q_0 0}{2q_7}\right]$$

as $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$ and part 4 places the dominos

$$\left[\frac{0}{0}\right], \left[\frac{1}{1}\right], \left[\frac{2}{2}\right], \text{ and } \left[\frac{\sqcup}{\sqcup}\right]$$

in P', as 0, 1, 2, and $\sqcup$ are the members of Γ. That, together with part 5, allows us to extend the match to

Reducing A_{TM} to MPCP

Thus the dominos of parts 2, 3, and 4 let us extend the match by adding the second configuration after the first one. We want this process to continue, adding the third configuration, then the fourth, and so on. For it to happen we need to add one more domino for copying the # symbol.

Reducing A_{TM} to MPCP

Thus the dominos of parts 2, 3, and 4 let us extend the match by adding the second configuration after the first one. We want this process to continue, adding the third configuration, then the fourth, and so on. For it to happen we need to add one more domino for copying the # symbol.

Part 5.

$$\text{Put } \left[\frac{\#}{\#}\right] \text{ and } \left[\frac{\#}{\sqcup\#}\right] \text{ into } P'.$$

The first of these dominos allows us to copy the # symbol that marks the separation of the configurations. In addition to that, the second domino allows us to add a blank symbol $\sqcup$ at the end of the configuration to simulate the infinitely many blanks to the right that are suppressed when we write the configuration.

Reducing A_{TM} to MPCP

Continuing with the example, let's say that in state q_7, upon reading a 1, M goes to state q_5, writes a 0, and moves the head to the right. That is, $\delta(q_7, 1) = (q_5, 0, \mathrm{R})$. Then we have the domino

$$\left[\frac{q_7 1}{0 q_5}\right] \text{ in } P'.$$

So the latest partial match extends to

$$\# \; 2 \; q_7 \; 1 \; 0 \; 0 \; \#$$

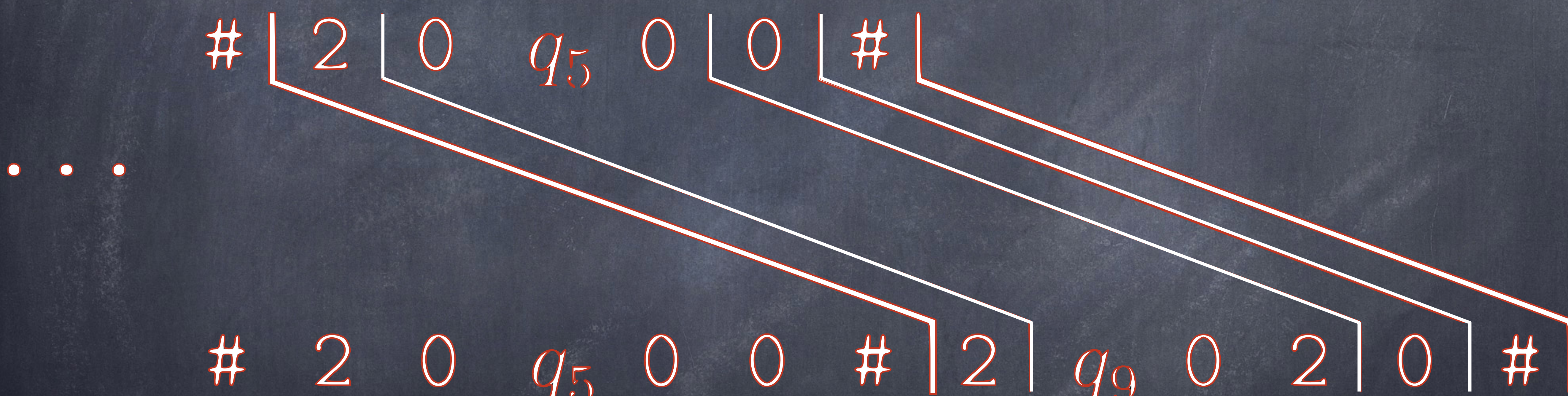
. . .

$$\# \; 2 \; q_7 \; 1 \; 0 \; 0 \; \# \; 2 \; 0 \; q_5 \; 0 \; 0 \; \#$$

Then, suppose that in state q_5, upon reading a 0, M goes to state q_9, writes a 2, and moves its head to the left. So $\delta(q_5, 0) = (q_9, 2, \mathrm{L})$. Then we have the dominos

$$\left[\frac{0q_50}{q_902}\right], \left[\frac{1q_50}{q_912}\right], \left[\frac{2q_50}{q_922}\right], \text{ and } \left[\frac{\sqcup q_50}{q_9\sqcup 2}\right].$$

The first one is relevant because the symbol to the left of the head is a 0. The preceding partial match extends to



Note that, as we construct a match, we are forced to simulate M on input w. This process continues until M reaches a halting state. If an accept state occurs, we want to let the top of the partial match "catch up" with the bottom so that the match is complete. We can arrange for that to happen by adding additional dominos.

Part 6. For every $a \in \Gamma$,

$$\text{put } \left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right] \text{ and } \left[\frac{q_{\text{accept}}\, a}{q_{\text{accept}}}\right] \text{ into } P'.$$

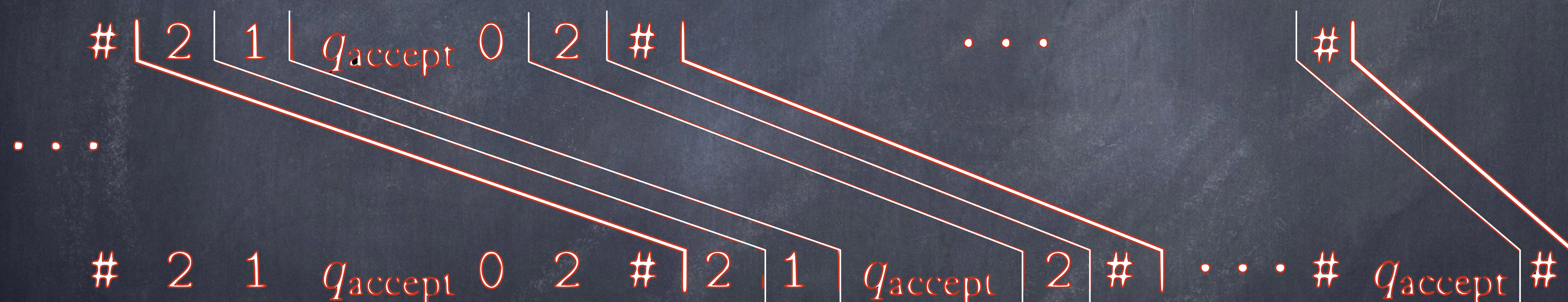
This step has the effect of adding "pseudo-steps" of the Turing machine after it has halted, where the head "eats" adjacent symbols until none are left. Continuing with the example, if the partial match up to the point when the machine halts in an accept state is

. . .

2 1 q_{accept} 0 2

Reducing A_{TM} to MPCP

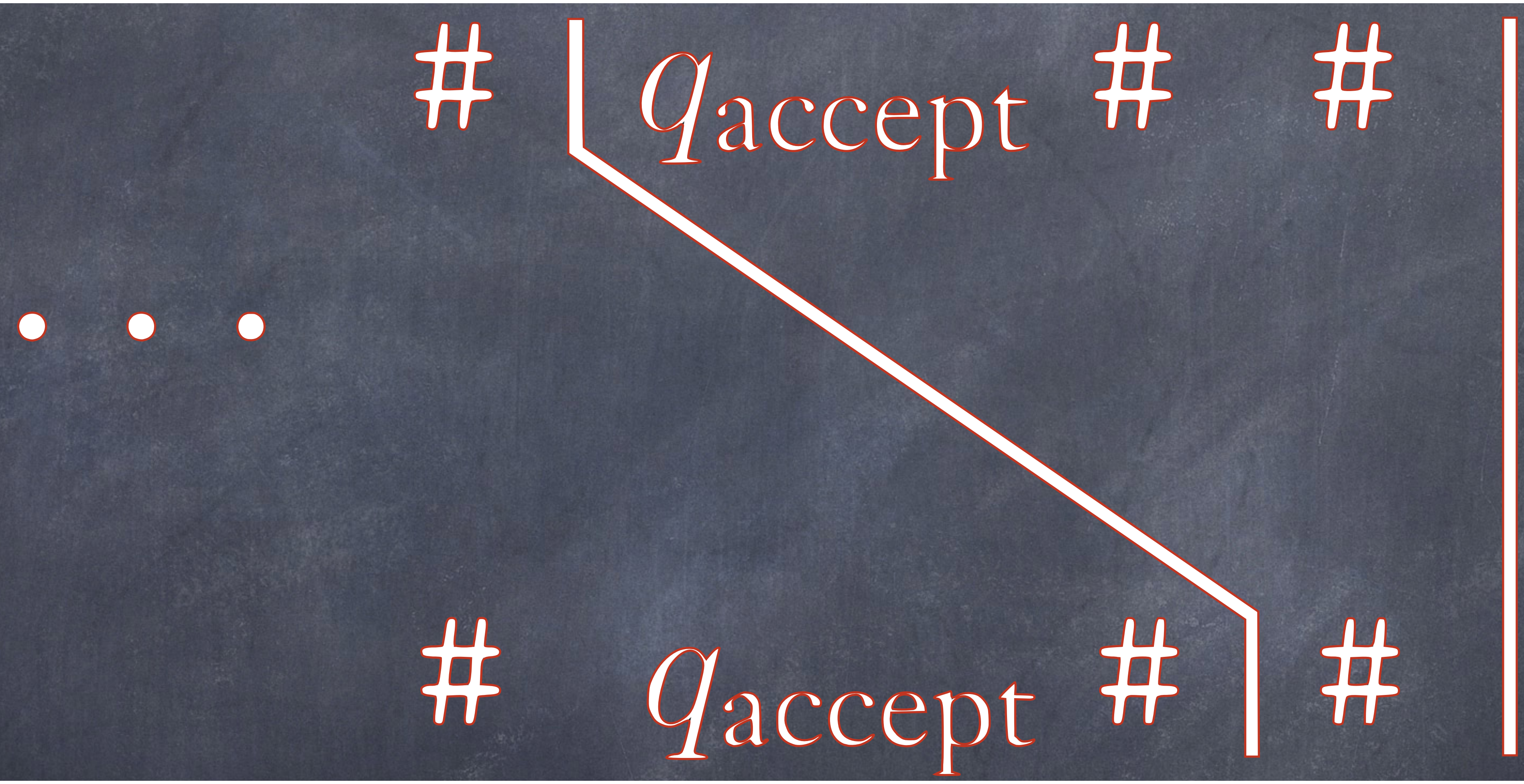
The dominos we have just added allow the match to continue:



Part 7. Finally we add the domino

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

and complete the match:



That concludes the construction of P'. Recall that P' is an instance of the MPCP whereby the match simulates the computation of M on w. To finish the proof, we recall that the MPCP differs from the PCP in that the match is required to start with the first domino in the list. If we view P' as an instance of the PCP instead of the MPCP, it obviously has a match, regardless of whether M halts on w. Can you find it? (Hint: It is very short.)

A_{TM} : a Reduction to MPCP

A story in seven parts

start

$$\left[\frac{\#}{\#q_0 w_1 w_2 \cdots w_n\#}\right]$$

part 1.

R-move

$$\left[\frac{q_0 0}{2 q_7}\right]$$

part 2.

L-move

$$\left[\frac{0 q_5 0}{q_9 0 2}\right] \quad \ldots \quad \left[\frac{2 q_5 0}{q_9 2 2}\right]$$

part 3.

copy

$$\left[\frac{0}{0}\right] \quad \ldots \quad \left[\frac{\sqcup}{\sqcup}\right]$$

part 4.

blank

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

part 5.

clean

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

part 6.

the end

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

part 7.

Reducing A_{TM} to MPCP
a (mostly) complete example

Reducing MPCP to PCP

We now show how to convert P' to P, an instance of the PCP that still simulates M on w. We do so with a somewhat technical trick. The idea is to build the requirement of starting with the first domino directly into the problem so that stating the explicit requirement becomes unnecessary. We need to introduce some notation for this purpose.

Let $u = u_1 u_2 \cdots u_n$ be any string of length n. Define $\star u$, $u\star$, and $\star u\star$ to be the three strings

$$\begin{aligned}
\star u &= * u_1 * u_2 * u_3 * \quad \cdots \quad * u_n \\
u\star &= u_1 * u_2 * u_3 * \quad \cdots \quad * u_n * \\
\star u \star &= * u_1 * u_2 * u_3 * \quad \cdots \quad * u_n * .
\end{aligned}$$

Here, $\star u$ adds the symbol $*$ before every character in u, $u\star$ adds one after each character in u, and $\star u\star$ adds one both before and after each character in u.

Reducing MPCP to PCP

To convert P' to P, an instance of the PCP, we do the following. If P' were the collection

$$\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \left[\frac{t_3}{b_3}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\},$$

we let P be the collection

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star}\right], \left[\frac{\star t_1}{b_1 \star}\right], \left[\frac{\star t_2}{b_2 \star}\right], \left[\frac{\star t_3}{b_3 \star}\right], \ldots, \left[\frac{\star t_k}{b_k \star}\right], \left[\frac{* \Diamond}{\Diamond}\right] \right\}.$$

Reducing MPCP to PCP

Considering P as an instance of the PCP, we see that the only domino that could possibly start a match is the first one,

$$\left[\frac{\star t_1}{\star b_1 \star}\right],$$

because it is the only one where both the top and the bottom start with the same symbol—namely, $*$. Besides forcing the match to start with the first domino, the presence of the $*$s doesn't affect possible matches because they simply interleave with the original symbols. The original symbols now occur in the even positions of the match. The domino

$$\left[\frac{*\Diamond}{\Diamond}\right]$$

is there to allow the top to add the extra $*$ at the end of the match.

Reducibility

ALL_{CFG}

Decidable	Undecidable
A_{DFA}	EQ_{CFG}
A_{NFA}	A_{TM}
A_{REX}	$HALT_{TM}$
E_{DFA}	E_{TM}
EQ_{DFA}	$REGULAR_{TM}$
A_{CFG}	EQ_{TM}
E_{CFG}	PCP

MPCP

Reducibility

$$ALL_{\mathsf{CFG}} = \{\langle G\rangle |\ G \text{ is a CFG and } L(G) = \Sigma^*\}.$$

THEOREM 5.13

ALL_{CFG} is undecidable.

$\mathbf{EQ_{CFG}}$ decidable ⇒ $\mathbf{ALL_{CFG}}$ decidable

$\mathbf{EQ_{CFG}} = \{ \langle G_1,G_2\rangle \mid G_1,G_2 \text{ are CFGs and } L(G_1)=L(G_2)\}$

Let $\langle G_2\rangle$ be such that $L(G_2)=\Sigma^*$. (G_2: $R \rightarrow \varepsilon \mid 0R \mid 1R$)

$\langle G\rangle \in \mathbf{ALL_{CFG}} \Leftrightarrow \langle G,G_2\rangle \in \mathbf{EQ_{CFG}}$

$\mathbf{ALL_{CFG}}$ decidable ⇒ $\mathbf{A_{TM}}$ decidable

We now describe how to use a decision procedure for ALL_{CFG} to decide A_{TM}. For a TM M and an input w, we construct a CFG G that generates all strings if and only if M does not accept w. So if M does accept w, G does *not* generate some particular string. This string is—guess what—the accepting computation history for M on w. That is, G is designed to generate all strings that are *not* accepting computation histories for M on w.

To make the CFG G generate all strings that fail to be an accepting computation history for M on w, we utilize the following strategy. A string may fail to be an accepting computation history for several reasons. An accepting computation history for M on w appears as $\#C_1\#C_2\#\cdots\#C_l\#$, where C_i is the configuration of M on the ith step of the computation on w. Then, G generates all strings

1. that *do not* start with C_1,
2. that *do not* end with an accepting configuration, or
3. in which some C_i *does not* properly yield C_{i+1} under the rules of M.

If M does not accept w, no accepting computation history exists, so *all* strings fail in one way or another. Therefore, G would generate all strings, as desired.

PDA D(↔G) for M does not accept w

$$\# \underbrace{\quad\longrightarrow\quad}_{C_1} \# \underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}} \# \underbrace{\quad\longrightarrow\quad}_{C_3} \# \underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

FIGURE 5.14
Every other configuration written in reverse order

$$\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longleftarrow}_{C_2^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longleftarrow}_{C_4^{\mathcal{R}}}\#\quad\cdots\quad\#\underbrace{\qquad}_{C_l}\#$$

$$\# \underbrace{\longrightarrow}_{C_1} \# \underbrace{\longleftarrow}_{C_2^{\mathcal{R}}} \# \underbrace{\longrightarrow}_{C_3} \# \underbrace{\longleftarrow}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.

$$\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longleftarrow}_{C_2^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longleftarrow}_{C_4^{\mathcal{R}}}\#\quad\cdots\quad\#\underbrace{\qquad\qquad}_{C_l}\#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.

$$\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longleftarrow}_{C_2^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longleftarrow}_{C_4^{\mathcal{R}}}\#\quad\cdots\quad\#\underbrace{\qquad\qquad}_{C_l}\#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:

$$\# \underbrace{\quad\longrightarrow\quad}_{C_1} \# \underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}} \# \underbrace{\quad\longrightarrow\quad}_{C_3} \# \underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:
  - It works by scanning the input until it nondeterministically decides that it has come to C_i.

$$\#\underbrace{\quad\longrightarrow\quad}_{C_1}\#\underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}}\#\underbrace{\quad\longrightarrow\quad}_{C_3}\#\underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}}\#\quad\cdots\quad\#\underbrace{\qquad\qquad}_{C_l}\#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:
  - It works by scanning the input until it nondeterministically decides that it has come to C_i.
  - Next, it pushes C_i onto the stack until it comes to the end as marked by the # symbol.

$$\# \underbrace{\longrightarrow}_{C_1} \# \underbrace{\longleftarrow}_{C_2^{\mathcal{R}}} \# \underbrace{\longrightarrow}_{C_3} \# \underbrace{\longleftarrow}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:
  - It works by scanning the input until it nondeterministically decides that it has come to C_i.
  - Next, it pushes C_i onto the stack until it comes to the end as marked by the # symbol.
  - Then D pops the stack to compare with C_{i+1}.

$$\# \underbrace{\quad\longrightarrow\quad}_{C_1} \# \underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}} \# \underbrace{\quad\longrightarrow\quad}_{C_3} \# \underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad}_{C_l} \#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:
  - It works by scanning the input until it nondeterministically decides that it has come to C_i.
  - Next, it pushes C_i onto the stack until it comes to the end as marked by the # symbol.
  - Then D pops the stack to compare with C_{i+1}.
  - They are supposed to match except around the head position, where the difference is dictated by the transition function of M.

$$\# \underbrace{\quad\longrightarrow\quad}_{C_1} \# \underbrace{\quad\longleftarrow\quad}_{C_2^{\mathcal{R}}} \# \underbrace{\quad\longrightarrow\quad}_{C_3} \# \underbrace{\quad\longleftarrow\quad}_{C_4^{\mathcal{R}}} \# \quad \cdots \quad \# \underbrace{\qquad\qquad\qquad}_{C_l} \#$$

- One branch checks on whether the beginning of the input string is C_1 and accepts if it isn't.
- Another branch checks on whether the input string ends with a configuration containing the accept state, q_{accept}, and accepts if it isn't.
- The third branch is supposed to accept if some C_i does not properly yield C_{i+1}:
  - It works by scanning the input until it nondeterministically decides that it has come to C_i.
  - Next, it pushes C_i onto the stack until it comes to the end as marked by the # symbol.
  - Then D pops the stack to compare with C_{i+1}.
  - They are supposed to match except around the head position, where the difference is dictated by the transition function of M.
  - Finally, D accepts if it discovers a mismatch or an improper update.

PDA D($\leftrightarrow$G) for $\langle M \rangle$ does not accept w

$$\mathbf{L(D)} = \begin{cases} \Sigma^* \setminus \{\textbf{accepting computation history}\} & \text{if } \mathbf{M} \text{ accepts } \mathbf{w} \\ \Sigma^* & \text{if } \mathbf{M} \text{ rejects } \mathbf{w} \end{cases}$$

- On input $\langle M,w \rangle$ generate $\langle G \rangle$ s.t.
  $L(G)=\Sigma^* \leftrightarrow$ M rejects w

- If All_{CFG} is decidable, then so is A_{TM}.

Mapping Reducibility

Computable Functions

A Turing machine computes a function by starting with the input to the function on the tape and halting with the output of the function on the tape.

DEFINITION 5.17

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Computable Functions

A Turing machine computes a function by starting with the input to the function on the tape and halting with the output of the function on the tape.

DEFINITION 5.17

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

EXAMPLE 5.18

All usual arithmetic operations on integers are computable functions. For example, we can make a machine that takes input $\langle m, n \rangle$ and returns $m + n$, the sum of m and n. We don't give any details here, leaving them as exercises.

Mapping Reducibility

FORMAL DEFINITION OF MAPPING REDUCIBILITY

Now we define mapping reducibility. As usual we represent computational problems by languages.

DEFINITION 5.20

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***reduction*** of A to B.

Mapping Reducibility

The following figure illustrates mapping reducibility.

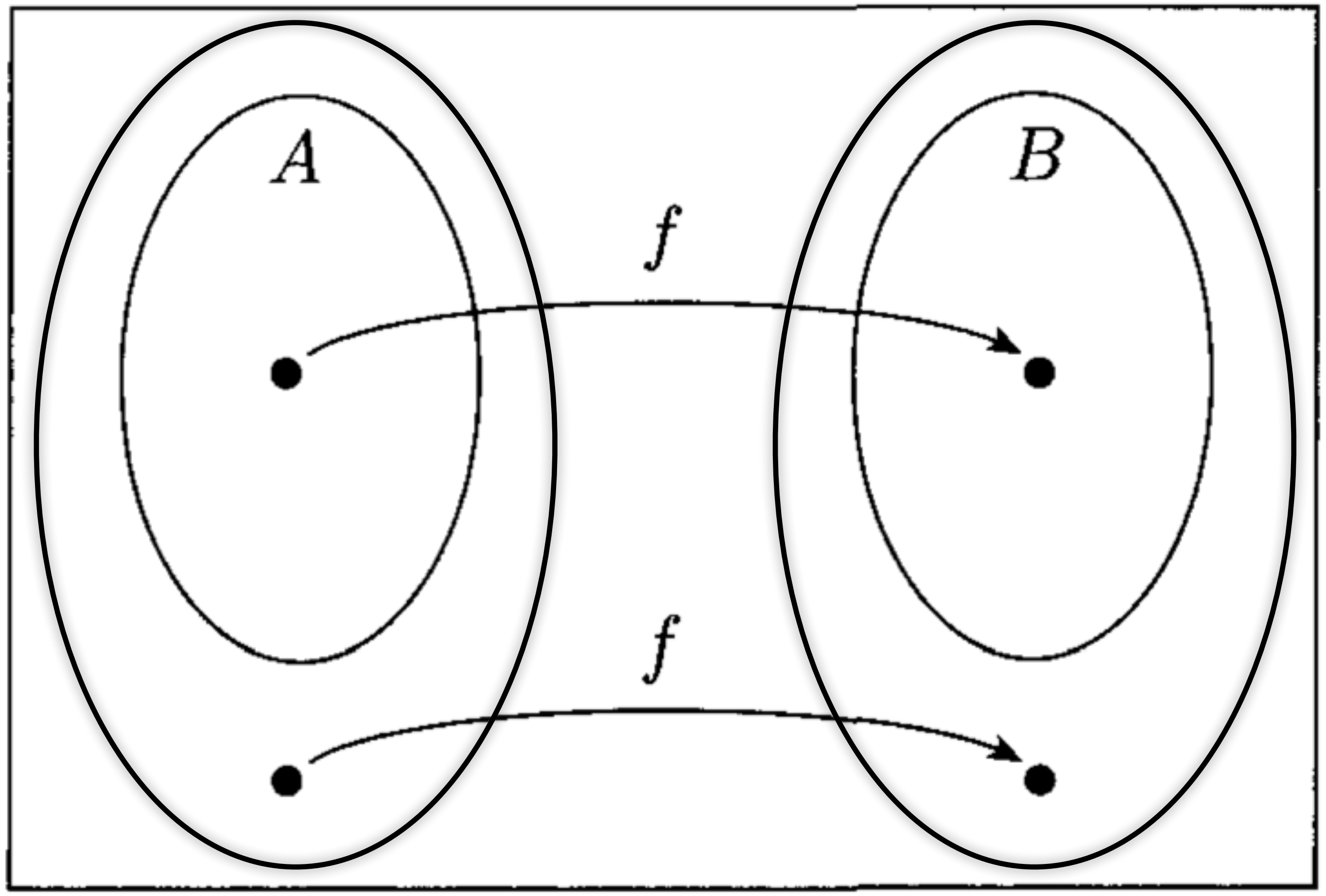


FIGURE 5.21

THEOREM 5.22

If $A \leq_m B$ and B is decidable, then A is decidable.

PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B. Thus M accepts $f(w)$ whenever $w \in A$. Therefore N works as desired.

THEOREM 5.22

If $A \leq_m B$ and B is decidable, then A is decidable.

PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because f is a reduction from A to B. Thus M accepts $f(w)$ whenever $w \in A$. Therefore N works as desired.

COROLLARY 5.23

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

EXAMPLE 5.24

In Theorem 5.1 we used a reduction from A_{TM} to prove that $HALT_{\mathsf{TM}}$ is undecidable. This reduction showed how a decider for $HALT_{\mathsf{TM}}$ could be used to give a decider for A_{TM}. We can demonstrate a mapping reducibility from A_{TM} to $HALT_{\mathsf{TM}}$ as follows. To do so we must present a computable function f that takes input of the form $\langle M, w\rangle$ and returns output of the form $\langle M', w'\rangle$, where

$$\langle M, w\rangle \in A_{\mathsf{TM}} \text{ if and only if } \langle M', w'\rangle \in HALT_{\mathsf{TM}}.$$

The following machine F computes a reduction f.

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

1. Construct the following machine M'.
   $M' =$ "On input x:
   1. Run M on x.
   2. If M accepts, *accept*.
   3. If M rejects, enter a loop."
2. Output $\langle M', w\rangle$."

Mapping Reducibility

EXAMPLE 5.25

The proof of the undecidability of the Post correspondence problem in Theorem 5.15 contains two mapping reductions. First, it shows that $A_{\mathsf{TM}} \leq_{\mathrm{m}} MPCP$ and then it shows that $MPCP \leq_{\mathrm{m}} PCP$. In both cases we can easily obtain the actual reduction function and show that it is a mapping reduction. As Exercise 5.6 shows, mapping reducibility is transitive, so these two reductions together imply that $A_{\mathsf{TM}} \leq_{\mathrm{m}} PCP$.

Mapping Reducibility

THEOREM 5.28

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

The proof is the same as that of Theorem 5.22, except that M and N are recognizers instead of deciders.

Mapping Reducibility

THEOREM 5.28 ....

If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

The proof is the same as that of Theorem 5.22, except that M and N are recognizers instead of deciders.

COROLLARY 5.29 ....

If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Mapping Reducibility

In a typical application of this corollary, we let A be $\overline{A_{\mathsf{TM}}}$, the complement of A_{TM}. We know that $\overline{A_{\mathsf{TM}}}$ is not Turing-recognizable from Corollary 4.23. The definition of mapping reducibility implies that $A \leq_{\mathrm{m}} B$ means the same as $\overline{A} \leq_{\mathrm{m}} \overline{B}$. To prove that B isn't recognizable we may show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} \overline{B}$. We can also use mapping reducibility to show that certain problems are neither Turing-recognizable nor co-Turing-recognizable, as in the following theorem.

Mapping Reducibility

THEOREM 5.30 ....

EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable.

PROOF First we show that EQ_{TM} is not Turing-recognizable. We do so by showing that A_{TM} is reducible to $\overline{EQ_{\mathsf{TM}}}$. The reducing function f works as follows.

$F =$ "On input $\langle M, w\rangle$ where M is a TM and w a string:

1. Construct the following two 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 $\langle M_1, M_2\rangle$."

Here, M_1 accepts nothing. If M accepts w, M_2 accepts everything, and so the two machines are not equivalent. Conversely, if M doesn't accept w, M_2 accepts nothing, and they are equivalent. Thus f reduces A_{TM} to $\overline{EQ_{\mathsf{TM}}}$, as desired.

To show that $\overline{EQ_{\mathsf{TM}}}$ is not Turing-recognizable we give a reduction from A_{TM} to the complement of $\overline{EQ_{\mathsf{TM}}}$—namely, EQ_{TM}. Hence we show that $A_{\mathsf{TM}} \leq_{\mathrm{m}} EQ_{\mathsf{TM}}$. The following TM G computes the reducing function g.

$G =$ "The input is $\langle M, w \rangle$ where M is a TM and w a string:

1. Construct the following two machines M_1 and M_2.

$M_1 =$ "On any input:

1. *Accept*."

$M_2 =$ "On any input:

1. Run M on w.

2. If it accepts, *accept*."

2. Output $\langle M_1, M_2 \rangle$."

The only difference between f and g is in machine M_1. In f, machine M_1 always rejects, whereas in g it always accepts. In both f and g, M accepts w iff M_2 always accepts. In g, M accepts w iff M_1 and M_2 are equivalent. That is why g is a reduction from A_{TM} to EQ_{TM}.

Turing Reducibility

Turing Reducibility

DEFINITION 6.18

An ***oracle*** for a language B is an external device that is capable of reporting whether any string w is a member of B. An ***oracle Turing machine*** is a modified Turing machine that has the additional capability of querying an oracle. We write M^B to describe an oracle Turing machine that has an oracle for language B.

Turing Reducibility

EXAMPLE 6.19

Consider an oracle for A_{TM}. An oracle Turing machine with an oracle for A_{TM} can decide more languages than an ordinary Turing machine can. Such a machine can (obviously) decide A_{TM} itself, by querying the oracle about the input. It can also decide E_{TM}, the emptiness testing problem for TMs with the following procedure called $T^{A_{\mathsf{TM}}}$.

$T^{A_{\mathsf{TM}}}$ = "On input $\langle M \rangle$, where M is a TM:

1. Construct the following TM N.
   N = "On any input:
   1. Run M in parallel on all strings in Σ^*.
   2. If M accepts any of these strings, *accept*."
2. Query the oracle to determine whether $\langle N, 0 \rangle \in A_{\mathsf{TM}}$.
3. If the oracle answers NO, *accept*; if YES, *reject*."

Turing Reducibility

DEFINITION 6.20

Language A is ***Turing reducible*** to language B, written $A \leq_{\mathrm{T}} B$, if A is decidable relative to B.

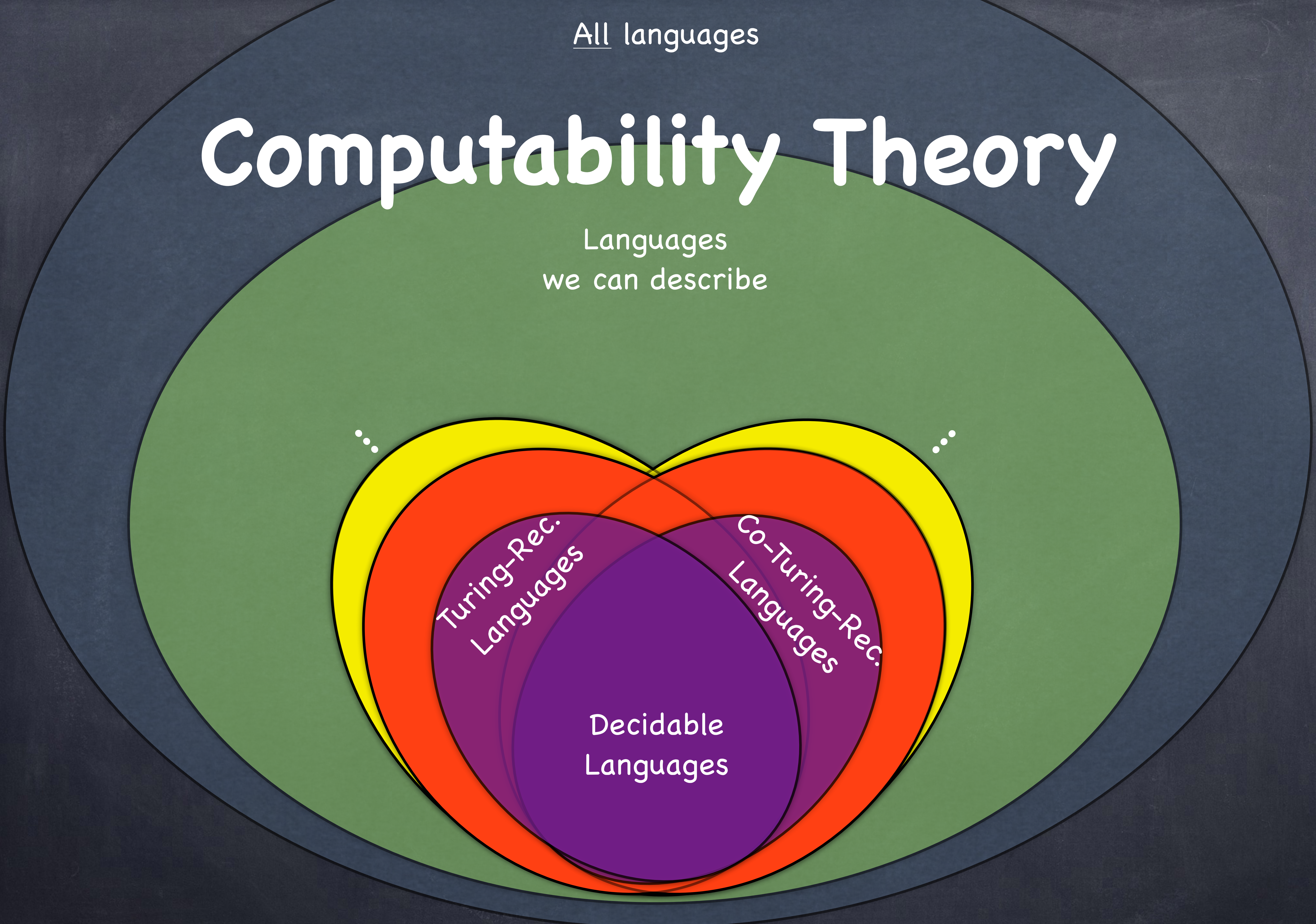
Turing Reducibility

THEOREM 6.21

If $A \leq_T B$ and B is decidable, then A is decidable.

PROOF If B is decidable, then we may replace the oracle for B by an actual procedure that decides B. Thus we may replace the oracle Turing machine that decides A by an ordinary Turing machine that decides A.

Computability Theory



COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 20-21: Reducibility