Decidable and Undecidable Problems

Beulah A.
AP/CSE

Decidable Problems

- **Decidable problems about regular Languages**
  - Acceptance problem for DFAs
  - Acceptance problem for NFAs
  - Acceptance problem for Regular Expressions
  - Emptiness testing for DFAs
  - 2 DFAs recognizing the same language
- **Decidable problems about Context Free Languages**
  - Does a given CFG generate a given string?
  - Is the language of a given CFG empty?
  - Every CFL is decidable by a Turing machine

Acceptance problem for DFAs

$$A_{DFA} = \{ < B, w > | \; B \text{ is a DFA that accepts a given string } w \}$$

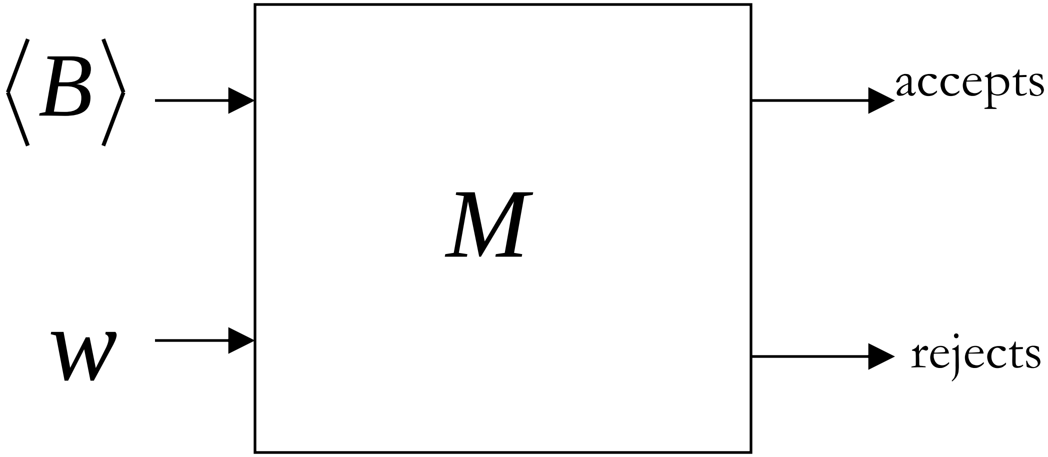
- Language includes encodings of all DFAs and strings they accept.
- Showing language is decidable is same as showing the computational problem is decidable.

- **Theorem 1**: A_{DFA} is a decidable language.
  - **Proof Idea**: Specify a TM M that decides A_{DFA}.
    - M = "On input <B,w>, where B is a DFA and w is a string:
      1. Simulate B on input w.
      2. If simulation ends in accept state, *accept*. If it ends in nonaccepting state, *reject*."

Acceptance problem for DFAs

Input string $\langle B, w \rangle$

Decider for A_{DFA}



M accepts w

M rejects w

Acceptance problem for NFAs

$$A_{NFA} = \{ <B, w> \mid B \text{ is an NFA that accepts a given string } w \}$$

- **Theorem 2**: A_{NFA} is a decidable language.
  - **Proof Idea**: Specify a TM N that decides A_{NFA}.
    - N = "On input <B,w>, where B is an NFA and w is a string:
      1. Convert NFA B to equivalent DFA C.
      2. Run TM M from Theorem 1 on input <C,w>.
      3. If M accepts, *accept*. Otherwise, *reject*."

Acceptance problem for RE

$A_{REX} = \{ <R, w> \mid R \text{ is a regular expression that generates string } w \}$

- **Theorem 3**: A_{REX} is a decidable language.
  - **Proof Idea**: Specify a TM P that decides A_{REX}.
    - P = "On input $<R,w>$, where R is a regular expression and w is a string:
      1. Convert regular expression R to equivalent NFA A
      2. Run TM N from Theorem 2 on input $<A,w>$.
      3. If N accepts, *accept*. If N rejects, *reject*."

Emptiness problem for DFAs

$$\mathrm{E}_{\mathrm{DFA}} = \{ <A> \mid A \text{ is a DFA and } L(A) = \emptyset \}$$

- **Theorem 4**: $\mathrm{E}_{\mathrm{DFA}}$ is a decidable language.
  - **Proof Idea**: Specify a TM T that decides $\mathrm{E}_{\mathrm{DFA}}$.
    - T = "On input $<A>$, where A is a DFA:
      1. Mark start state of A.
      2. Repeat until no new states are marked:

         Mark any state that has a transition coming into it from any state that is already marked.
      3. If no accept state is marked, *accept*; otherwise, *reject*."

2 DFAs recognizing the same language

$$\mathrm{EQ}_{\mathrm{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B) \}$$

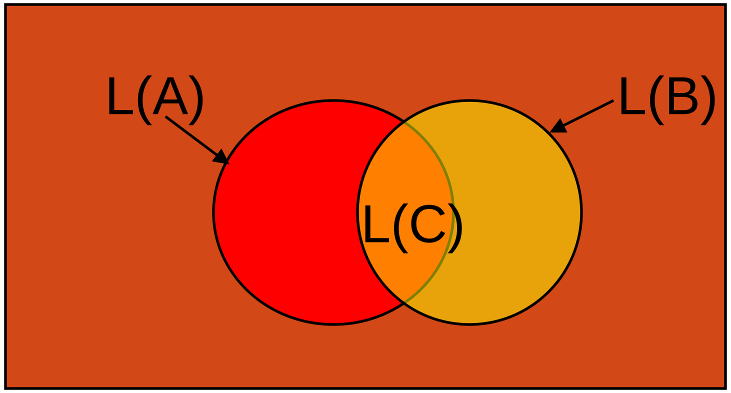
- **Theorem 5**: $\mathrm{EQ}_{\mathrm{DFA}}$ is a decidable language.

symmetric difference:

$$L(C) = \left(L(A) \cap \overline{L(B)} \right) \cup \left(\overline{L(A)} \cap L(B) \right)$$

emptiness:

$$L(C) = \emptyset \Leftrightarrow L(A) = L(B)$$



Does a given CFG generate a given string?

$$A_{\text{CFG}} = \{<G, w> | \, G \text{ is a CFG that generates string } w\}$$

- **Theorem 6** : A_{CFG} is a decidable language.
  - Why is this unproductive: use *G* to go through all derviations to determine if any yields *w*?
  - Better Idea…**Proof Idea**: Specify a TM *S* that decides A_{CFG}.
    - *S* = "On input <*G*,*w*>, where *G* is a CFG and *w* is a string:
      1. Convert *G* to equivalent Chomsky normal form grammar.
      2. List all derivations with 2*n*-1 steps (**why?**), where *n* = length of *w*. (Except if *n*=0, only list derivations with 1 step.)
      3. If any of these derivations yield *w*, *accept*; otherwise, *reject*."

Is the language of a given CFG empty?

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$$

- **Theorem 7**: E_{CFG} is a decidable language.
  - **Proof Idea**: Specify a TM R that decides E_{CFG}.
    - R = "On input $\langle G \rangle$, where G is a CFG:
      1. Mark all terminal symbols in G.
      2. Repeat until no new variables get marked:
         Mark any variable A where G has rule $A \rightarrow U_1, U_2 \ldots U_k$ and each symbol $U_1, U_2 \ldots U_k$ has already been marked.
      1. If start variable is not marked, *accept*; otherwise, *reject*."

Every CFL is decidable by a Turing machine

- **Theorem 8**: Every context-free language is decidable.
  - Let A be a CFL and G be a CFG for A.
  - Design TM M_G that decides A.
  - M_G = "On input w, where w is a string:
    1. Run TM S from Theorem 6 on input $<G,w>$.
    2. If S accepts, *accept*. If S rejects, *reject*."

Undecidable Problems

- Halting Problem
- Post's Correspondence problem
- Busy Beaver problem
- Whether the language accepted by a TM is empty
- Whether the language accepted by a TM is regular language
- Whether the language accepted by a TM is context free language

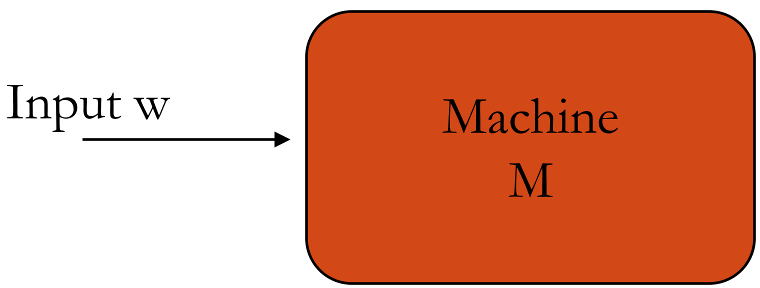
The Halting Problem

An example of a recursive enumerable problem that is also undecidable

What is the Halting Problem?

- *Does a givenTuring Machine M halt on a given input w?*
- Example: Given an arbitrary Turing machine M over alphabet $\Sigma = \{ a , b \}$, and an arbitrary string w over , does M halt when it is given w as an input ?

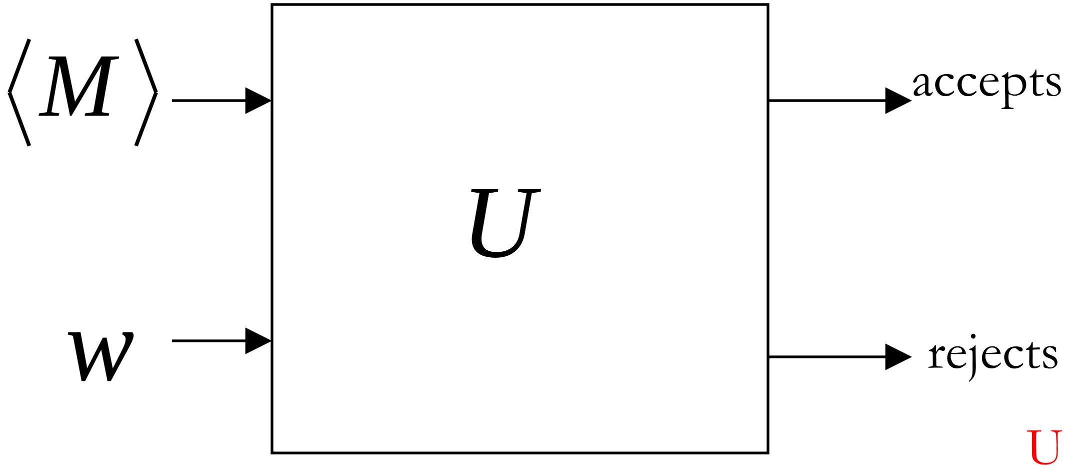
$$\mathrm{HALT}_{\mathrm{TM}} = \{ < M, w > | \text{ M is a TM which halts on } w \}$$



Revisit UTM

Input string

$\langle M, w \rangle$



U accepts, if M accepts w

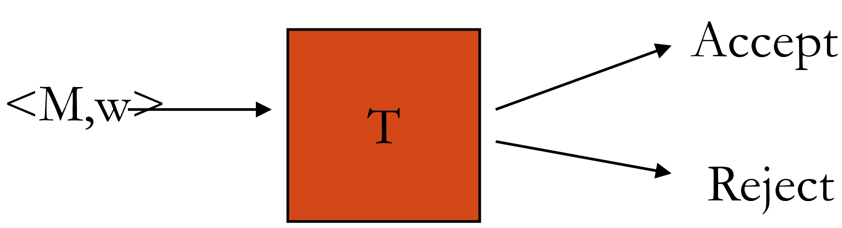
U rejects, if M does not accept w

Theorem: Is $HALT_{TM}$ decidable?

- Halting Problem is undecidable
- ie, there is no Turing Machine that solves Halting Problem
- If there was such a Turing Machine
    - Its input will have two portions, M and w
    - It outputs either a YES or a NO depending on whether M halts on input w

Proof by contradiction

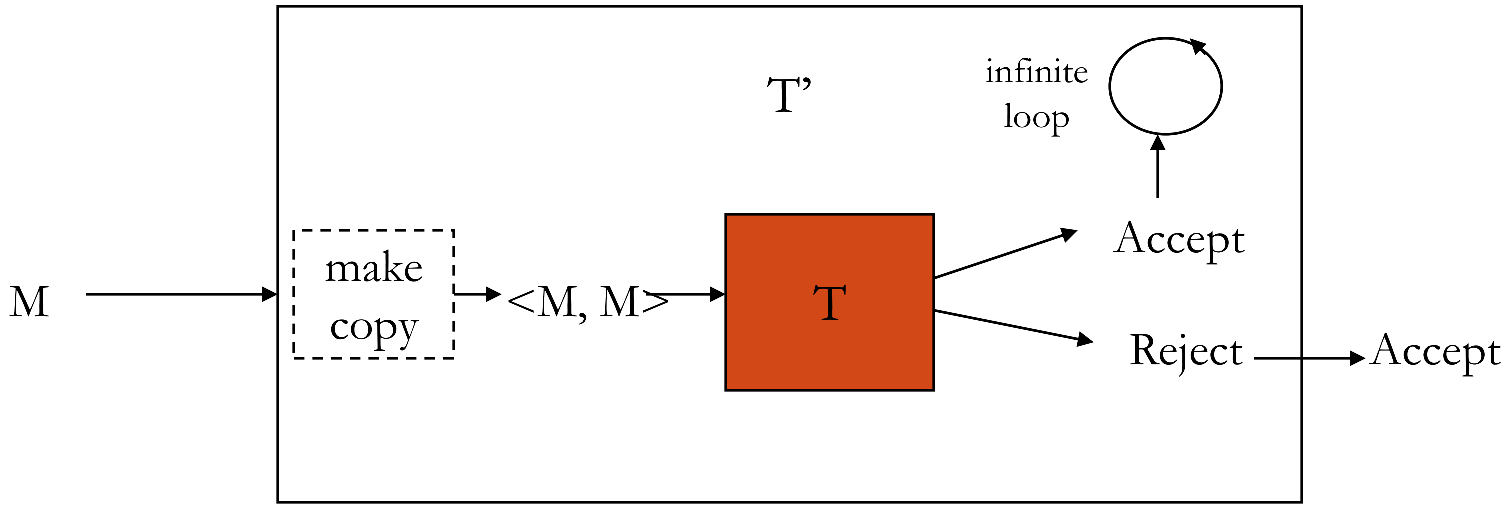
- Suppose Halting Problem is decidable
  - Plan: arrive at a contradiction
- If Halting Problem is decidable, then there exists a TM T that decides Halting Problem



Proof by contradiction

- Create a TM T' based on T as follows:
  - T takes in a TM M
  - In T', M is duplicated so that there are now two portions on the input tape
  - Feed this new input into T
  - When it is about to print reject, print accept instead
  - When it is about to print accept, send the program to an infinite loop

Proof by contradiction

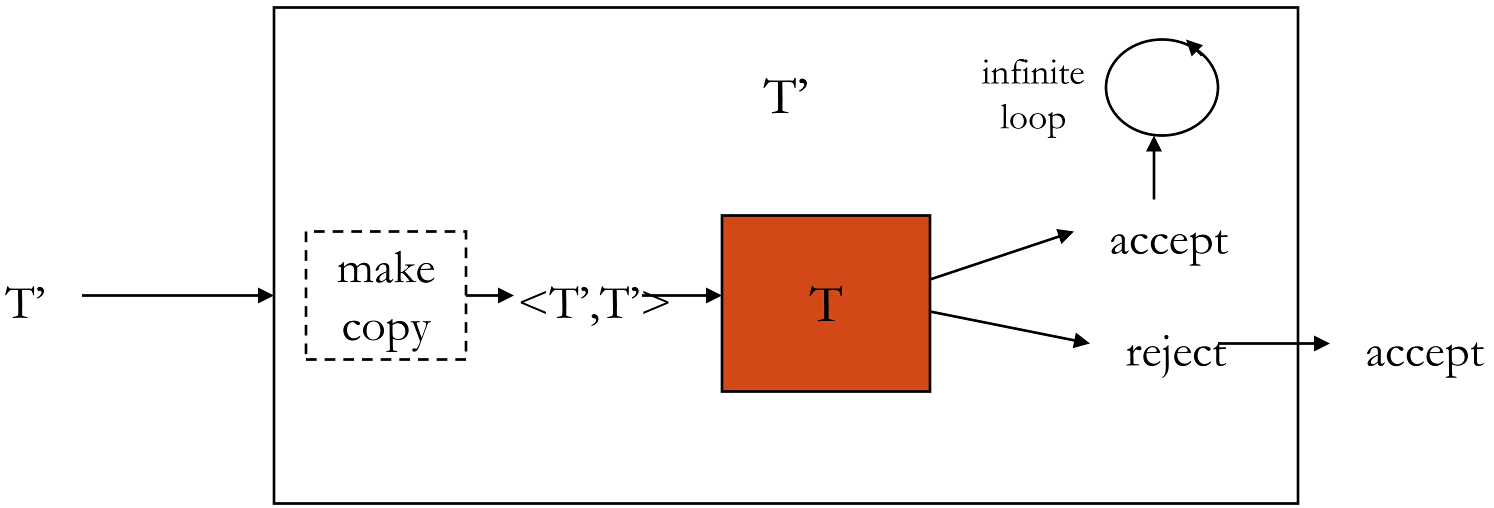


- Program T' takes a M as input, prints accept if M **does not halt** on input M, but goes into an infinite loop if M **halts** on input M

Proof by contradiction

- Consider feeding TM T' to itself
- Consequence (two possibilities)
  - It prints accept
    - T' halts on input T'
      if T' does not halt on input T' → a contradiction
  - It goes to an infinite loop
    - T' does not halt on input T'
      if T' halts on input T' → a contradiction
- Therefore the supposition cannot hold, and Halting Problem is undecidable

Proof by contradiction

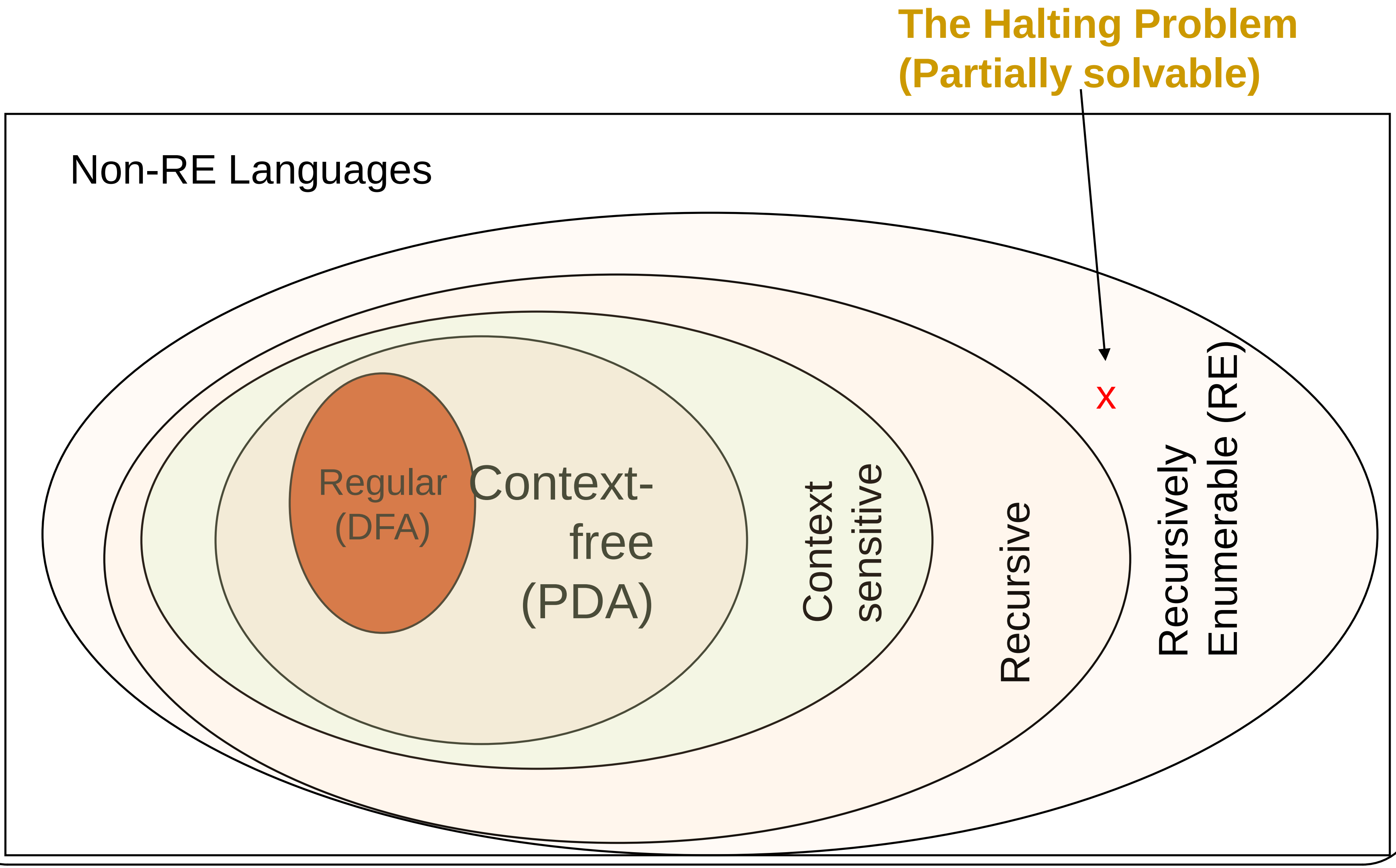


- T' halts on input T' (prints a accept, see outer box) if
- T' does not halt on input T' (T should yield a reject, see inner box)
- T' does not halt on input T' (infinite loop, see outer box) if
- T' halts on input T' (T should yield a accept, see inner box)

HP is semidecidable

- There are problems such as HP that cannot be solved
- Actually, HP is semidecidable, that is if all we need is print accept when M on w halts, but not worry about printing reject if otherwise, a TM machine exists for the halting problem
  - Just simulate M on w, print accept (or go to a final state) when the simulation stops
  - This means that HP is not recursive but it is recursively enumerable

HP is semidecidable

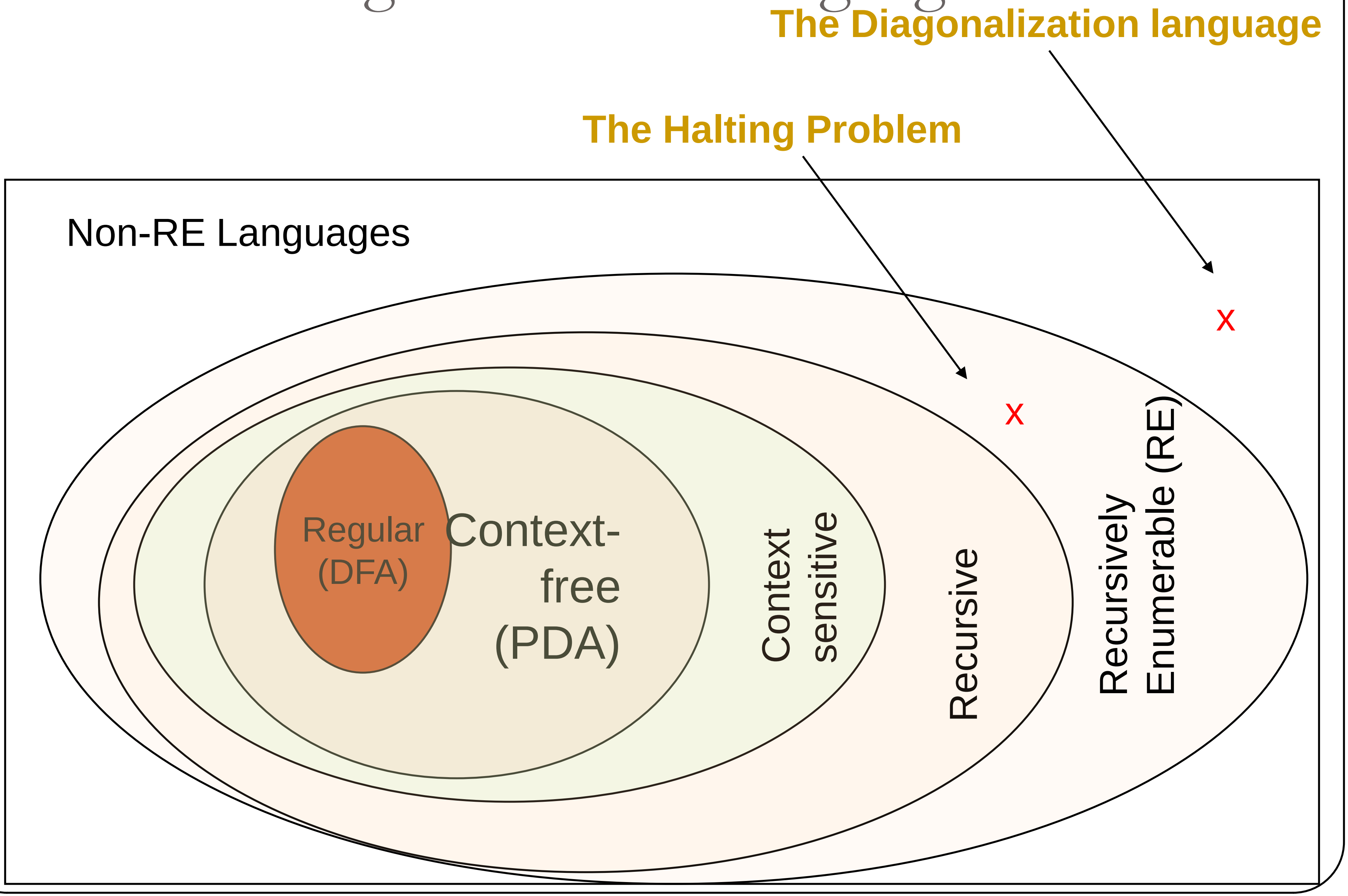


The Diagonalization Language

Example of a language that is
<u>not recursive enumerable</u>

(i.e, no TMs exist)

The Diagonalization Language



A Language about TMs & acceptance

- Let L be the language of all strings <M,w> s.t.:
  1. M is a TM (coded in binary) with input alphabet also binary
  2. w is a binary string
  3. M accepts input w.

Enumerating all binary strings

- Let w be a binary string
- Then $1w \equiv i$, where i is some integer
  - E.g., If $w=\varepsilon$, then $i=1$;
  - If $w=0$, then $i=2$;
  - If $w=1$, then $i=3$; so on…
- If $1w\equiv i$, then call w as the i^{th} word or i^{th} binary string, denoted by w_i.
- ***A canonical ordering of all binary strings:***
  - ***{ε, 0, 1, 00, 01, 10, 11, 000, 100, 101, 110, …..}***
  - $\{w_1, w_2, w_3, w_4, \ldots\, w_i, \ldots\}$

Any TM M can also be binary-coded

- $M = \{ Q, \{0,1\}, \Gamma, \delta, q_0,B,F \}$
  - Map all states, tape symbols and transitions to integers (→ binary strings)
  - $\delta(q_i,X_j) = (q_k,X_l,D_m)$ will be represented as:
    → $0^i1\ 0^j1\ 0^k1\ 0^l1\ 0^m$
- <u>Result:</u> Each TM can be written down as a long binary string
- Canonical ordering of TMs:
  - $\{M_1, M_2, M_3, M_4, \ldots . M_i, \ldots \}$

The Diagonalization Language

- $L_d = \{ w_i \mid w_i \notin L(M_i) \}$
  - The language of all strings whose corresponding machine does *not* accept itself (i.e., its own code)

(input word w)

j →

(TMs) i ↓

	1	2	3	4	…
1	0	1	0	1	…
2	1	1	0	0	…
3	0	1	0	1	…
4	1	0	0	1	…
⋮					

diagonal

- <u>Table:</u> $T[i,j] = 1$, if M_i accepts w_j
  $= 0$, otherwise.

- Make a new language called
  $L_d = \{w_i \mid T[i,i] = 0\}$

L_d is not RE (i.e., has no TM)

<u>Proof (by contradiction):</u>

Let M be the TM for L_d

→ M has to be equal to some M_k s.t.

$$L(M_k) = L_d$$

→ Will w_k belong to $L(M_k)$ or not?

1. If $w_k \in L(M_k)$ ==> $T[k,k]=1$ ==> $w_k \notin L_d$
2. If $w_k \notin L(M_k)$ ==> $T[k,k]=0$ ==> $w_k \in L_d$

A contradiction either way!!

Post's Correspondence Problem

Emil Post

(Post Correspondence Problem)

Definition

Given two lists A and B:

$$A = w_1, w_2, \ldots, w_k \qquad B = x_1, x_2, \ldots, x_k$$

The problem is to determine if there is a sequence of one or more integers $i_1, i_2, \ldots, i_m$ such that:

$$w_{i_1} w_{i_2} \ldots w_{i_m} = x_{i_1} x_{i_2} \ldots x_{i_m}$$

(w_i, x_i) is called a corresponding pair.

Indices may be repeated or omitted

Example

	w_1	w_2	w_3
$A:$	100	11	111
	x_1	x_2	x_3
$B:$	001	111	11

PC-solution: 2,1,3 $w_2 w_1 w_3 = x_2 x_1 x_3$

11100111

Example

$$
\begin{array}{cccc}
 & w_1 & w_2 & w_3 \\
A: & 00 & 001 & 1000 \\
 & x_1 & x_2 & x_3 \\
B: & 0 & 11 & 011
\end{array}
$$

- There is no solution
- Because total length of strings from B is smaller than total length of strings from A

Modified Post Correspondence Problem (MPCP)

Given two lists A and B:

$$A = w_1, w_2, \ldots, w_k \qquad B = x_1, x_2, \ldots, x_k$$

The problem is to determine if there is a sequence of one or more integers $i_1, i_2, \ldots, i_m$ such that:

$$w_1 w_{i_1} w_{i_2} \ldots w_{i_m} = x_1 x_{i_1} x_{i_2} \ldots x_{i_m}$$

(w_i, x_i) is called a corresponding pair.

- Pair (w_1, x_1) is forced to be at the beginning of the two strings.

Example

	A	B
i	w_i	x_i
1	11	1
2	1	111
3	0111	10
4	10	0

This MPCP instance has a solution: 3, 2, 2, 4:

$$w_1w_3w_2w_2w_4 = x_1x_3x_2x_2x_4 = 1101111110$$

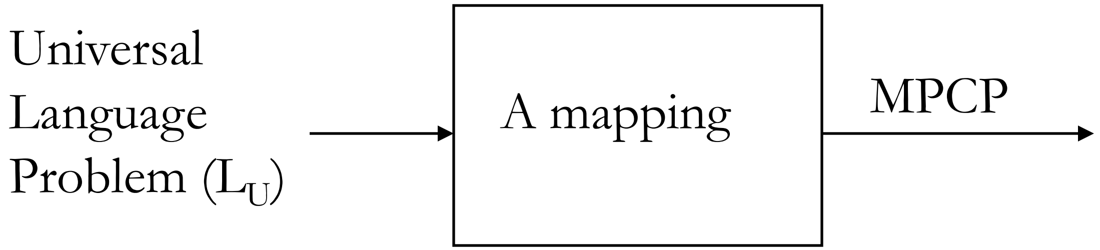
Example

	A	B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Does this MPCP instance have a solution?

Undecidability of PCP

- To show that MPCP is undecidable, we will reduce the universal language problem (L_U) to MPCP:



- If MPCP can be solved, L_U can also be solved. Since we have already shown that L_U is un-decidable, MPCP must also be undecidable.

Mapping L_U to MPCP

- Mapping a universal language problem instance to an MPCP instance is not as easy.
- In a L_U instance, we are given a Turing machine M and an input w, we want to determine if M will accept w.
- To map a L_U instance to an MPCP instance successfully, the mapped MPCP instance should have a solution if and only if M accepts w.

Mapping L_U to MPCP

L_U instance

Given:
(M,w)

Construct an
MPCP instance

MPCP instance

Two lists:
A and B

If M accepts w, the two lists can be matched.
Otherwise, the two lists cannot be matched.

Mapping L_U to MPCP

- We assume that the input Turing machine M:
  - Never prints a blank
  - Never moves left from its initial head position.
- Given M and w, the idea is to map the transition function of M to strings in the two lists in such a way that a matching of the two lists will correspond to a concatenation of the tape contents at each time step.

Mapping L_U to MPCP

- Given M and w, there are five types of strings in list A and B:
- **Starting string (first pair):**

List A	List B
#	$\#q_0w\#$

where q_0 is the starting state of M.

Mapping L_U to MPCP

- **Strings from the transition function δ:**

List A	List B	
qX	Yp	from $\delta(q,X)=(p,Y,R)$
ZqX	pZY	from $\delta(q,X)=(p,Y,L)$
q#	Yp#	from $\delta(q,\#)=(p,Y,R)$
Zq#	pZY#	from $\delta(q,\#)=(p,Y,L)$

where <u>Z is any tape symbol except the blank</u>.

Mapping L_U to MPCP

- **Strings for copying:**

	List A	**List B**
	X	X

where X is any tape symbol (including the blank).

Mapping L_U to MPCP

- **Strings for consuming the tape symbols at the end:**

List A	List B
Xq	q
qY	q
XqY	q

where q is an accepting state, and each X and Y is any tape symbol except the blank.

Mapping L_U to MPCP

- **Ending string:**

List A	**List B**
q##	#

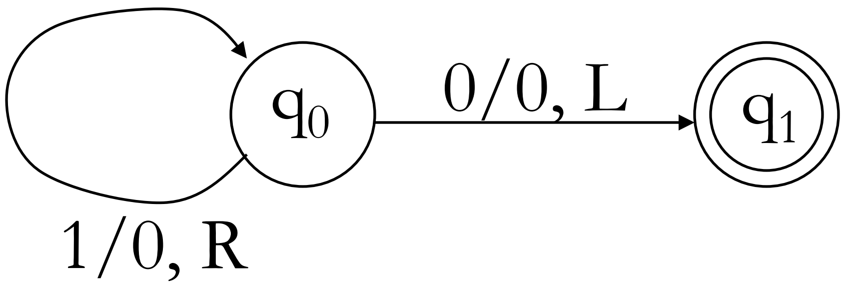
where q is an accepting state.

- Using this mapping, we can prove that the original L_U instance has a solution if and only if the mapped MPCP instance has a solution.

Example

- Consider the following Turing machine:

$$M = (\{q_0, q_1\},\{0,1\},\{0,1,\#\}, \delta, q_0, \#, \{q_1\})$$



$$\delta(q_0,1)=(q_0,0,R) \qquad \delta(q_0,0)=(q_1,0,L)$$

- Consider input w=110.

Example

- Now we will construct an MPCP instance from M and w. There are <u>five</u> types of strings in list A and B:

- **Starting string (<u>first pair</u>):**

List A	**List B**
#	#$q_0$110#

Example

- **Strings from the transition function δ:**

List A	**List B**	
$q_0 1$	$0 q_0$	(from $\delta(q_0,1)=(q_0,0,R)$)
$0 q_0 0$	$q_1 00$	(from $\delta(q_0,0)=(q_1,0,L)$)
$1 q_0 0$	$q_1 10$	(from $\delta(q_0,0)=(q_1,0,L)$)

Example

- **Strings for copying:**

List A	**List B**
#	#
0	0
1	1

Example

- **Strings for consuming the tape symbols at the end:**

List A	**List B**	**List A**	**List B**
$0q_1$	q_1	$0q_11$	q_1
$1q_1$	q_1	$1q_10$	q_1
q_10	q_1	$0q_10$	q_1
q_11	q_1	$1q_10$	q_1

Example

- **Ending string:**

List A	**List B**
q_1##	#

Now, we have constructed an MPCP instance.

Example

List A	List B	List A	List B
1. #	$\#q_0110\#$	9. $0q_1$	q_1
2. q_01	$0q_0$	10. $1q_1$	q_1
3. $0q_00$	q_100	11. q_10	q_1
4. $1q_00$	q_110	12. q_11	q_1
5. #	#	13. $0q_11$	q_1
6. 0	0	14. $1q_10$	q_1
7. 1	1	15. $0q_10$	q_1
8. $q_1\#\#$	#	16. $1q_10$	q_1

Example of ULP to MPCP

- This ULP instance has a solution:

$$q_0 110 \rightarrow 0q_0 10 \rightarrow 00q_0 0 \rightarrow 0q_1 00 \text{ (halt)}$$

- Does this MPCP instance has a solution?

List A:

$\# \; q_0 \; 1 \; 1 \; 0 \; \# \; 0 \; q_0 \; 1 \; 0 \; \# \; 0 \; 0 \; q_0 \; 0 \; \# \; 0 \; q_1 \; 0 \; 0 \; \# \; q_1 \; 0 \; \# \; q_1 \; \# \; \#$

List B:

$\# \; q_0 \; 1 \; 1 \; 0 \; \# \; 0 \; q_0 \; 1 \; 0 \; \# \; 0 \; 0 \; q_0 \; 0 \; \# \; 0 \; q_1 \; 0 \; 0 \; \# \; q_1 \; 0 \; \# \; q_1 \; \# \; \#$

The solution is the sequence of indices:

2, 7, 6, 5, 6, 2, 6, 5, 6, 3, 5, 15, 6, 5, 11, 5, 8

Class Discussion

Consider the input $w = 101$. Construct the corresponding MPCP instance I and show that M will accept w by giving a solution to I.

Class Discussion (cont'd)

	List A	List B
1.	#	$\#q_0101\#$
2.	q_01	$0q_0$
3.	$0q_00$	q_100
4.	$1q_00$	q_110
5.	#	#
6.	0	0
7.	1	1
8.	$q_1\#\#$	#

	List A	List B
9.	$0q_1$	q_1
10.	$1q_1$	q_1
11.	q_10	q_1
12.	q_11	q_1
13.	$0q_11$	q_1
14.	$1q_10$	q_1
15.	$0q_10$	q_1
16.	$1q_10$	q_1