THEORY OF COMPUTATION
CODE:303105306

UNIT 5:UNDECIDABILITY

CHAPTER-5

Undecidability

Problems and Languages

- Decision problems can be captured as languages
- Example:
  - The problem of determining whether a number is divisible by 3 corresponds to the language of all strings that form numbers divisible by 3
  - The finite automaton (or turing machine) for divisibility by 3 **solves** the problem and represents the language

Using a TM to solve a problem

- ❖ Recall the two possible interpretations of “solving” a problem through Turing Machines
- ❖ Device a TM so that acceptable (input) strings in the corresponding language are those that end up in a final state
    - ❖ When not acceptable, we don’t need to care if the TM halts
- ❖ Device a machine so that it leaves YES on the tape when the string is acceptable, NO if not
    - ❖ Note: steeper requirement than the first option

Undecidable problems

- A decision problem (or language) is decidable if there is a Turing Machine that solves that problem
- Leaves a YES on the tape when the string is acceptable, leaves a NO when it is not
- A problem (or language) is undecidable if there is no Turing Machine that solves (or decides) that problem
- The terms solvable/unsolvable are sometimes used instead of decidable/undecidable

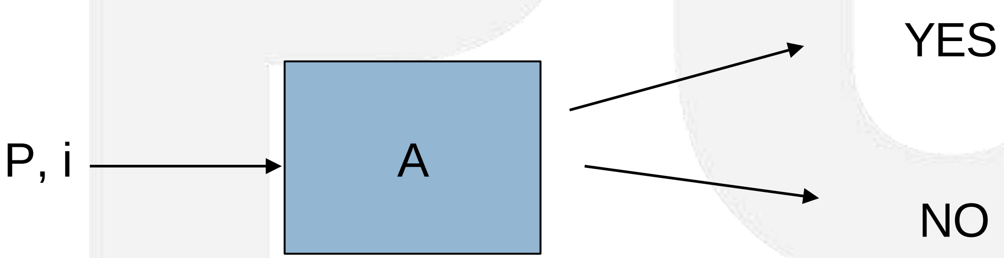
The Halting Problem

- The Halting Problem (HP):

  Given a TM P and input i, does P halt on i?
- HP is undecidable
- That is, there is no program (turing machine) that solves HP
- If there was such a program
- Its input will have two portions, P and i
- It outputs either a YES or a NO depending on whether P halts on input i

HP is Undecidable

- ❖ Proof by contradiction: suppose HP is decidable
  - ❖ Plan: arrive at a contradiction
- ❖ If HP is decidable, then there exists a program A that solves HP.

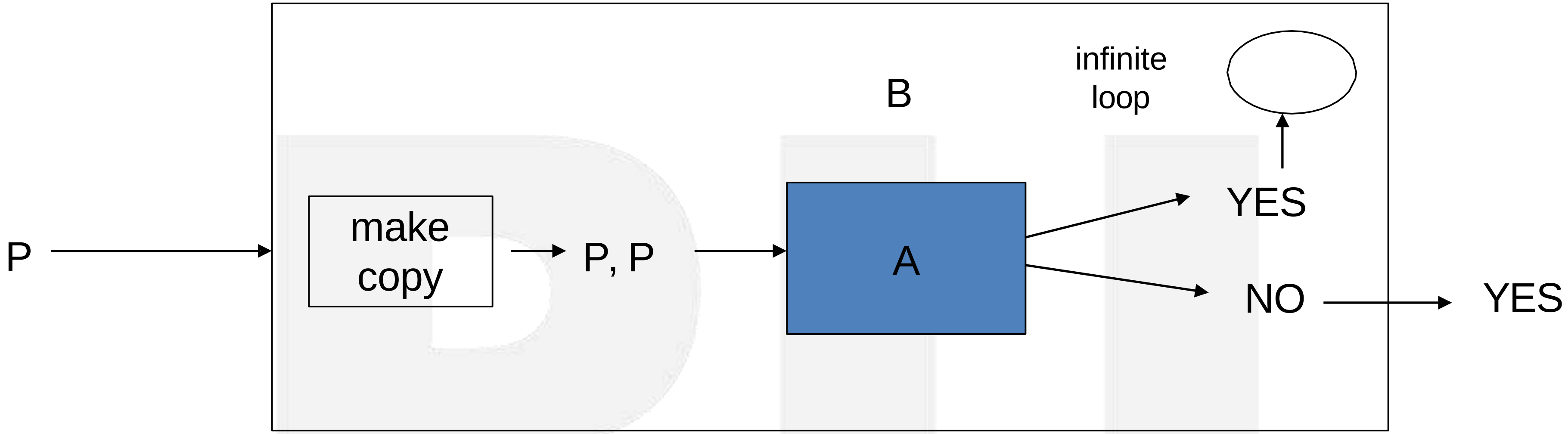


HP is Undecidable

Create a program B based on A as follows:

- B takes in a program P
- In B, P is duplicated so that there are now two portions on the input tape
- Feed this new input into A
- When it is about to print NO, print YES instead
- When it is about to print YES, send the program to an infinite loop

HP is Undecidable



- Program B takes a program P as input,
- prints a YES if P does not halt on input P,
- but goes into an infinite loop if P halts on input P

Parul® University

DIGITAL LEARNING CONTENT

HP is Undecidable

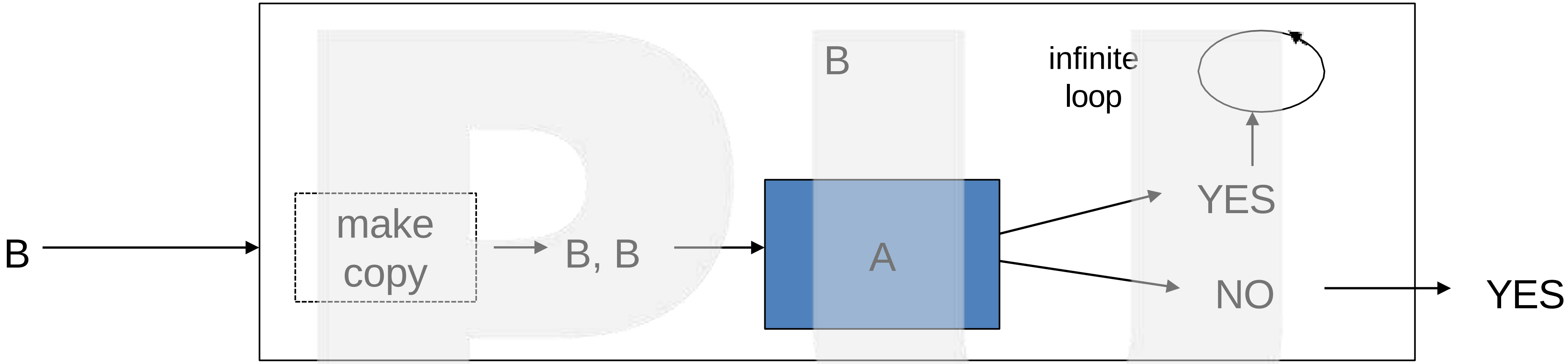
- Consider feeding program B to itself
- Consequence (two possibilities)
  - It prints a YES
    - B halts on input B

    if B does not halt on input B → a contradiction
  - It goes to an infinite loop
    - B does not halt on input B

    if B halts on input B → a contradiction
- Therefore the supposition cannot hold, and HP is undecidable

HP is Undecidable

❖Feeding program B to itself



❖B halts on input B (prints a YES, see outer box) if B does not halt on input B (A should yield a NO, see inner box)

❖B does not halt on input B (infinite loop, see outer box) if B halts on input B (A should yield a YES, see inner box)

Notes and Conclusions(Undecidable Problems)

- There are problems such as HP that cannot be solved
- Actually, HP is **semidecidable**, that is if all we need is print YES when P on i halts, but not worry about printing NO if otherwise, a TM machine exists for the halting problem
  - Just simulate P on i, print YES (or go to a final state) when the simulation stops
  - This means that HP is not recursive but it is recursively enumerable

Universal Turing Machine

- **Limitation Of Turing Machine**

Turing Machines are "hardwired"

they execute only one program

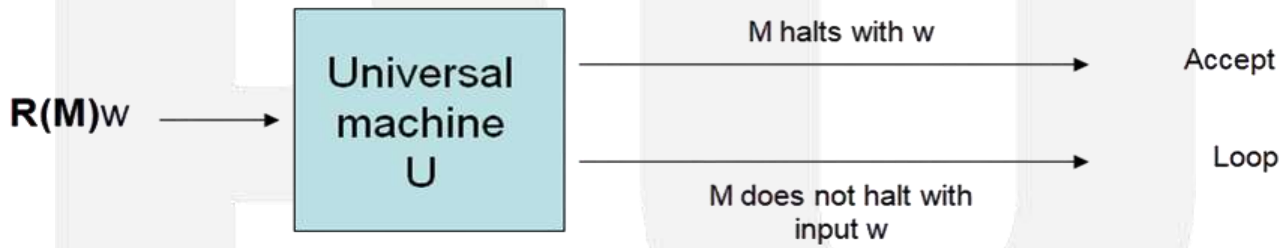
- Real Computers are re-programmable
- **Solution: Universal Turing Machine**

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine

Universal Turing Machine

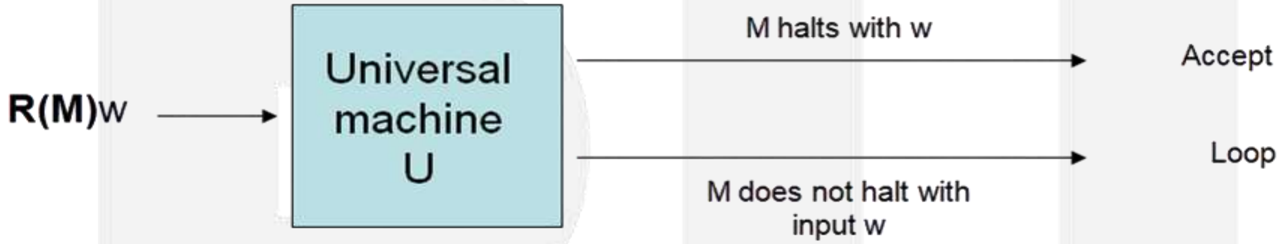
- A **universal Turing machine** is designed to simulate the computations of an arbitrary Turing machine M.



- Here, **R(M)** represents a Turing machine M that accepts by halting.

  W represents the input string.

Universal Turing Machine



- If **M** halts and accepts input w ===> U halts and accepts input W too
- If **M** does not halt with W ===> Neither does **U**.

Alphabet Encoding

Symbols:	a	b	c	d
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
Encoding:	1	11	111	1111

Alphabet Encoding

- State Encoding

States:	q_1	q_2	q_3	q_4
Encoding:	1	11	111	1111

- Head Move Encoding

Move:	L	R
	1	11

Symbol - Encoding

Input alphabet
{0,1}

tape alphabet
{0, 1, B}

states q0, q1 ….

0	1
1	11
B	111
q_0	1
q_1	11
…	
q_n	1^{n+1}
L	1
R	11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,11\,0\,11\,0\,1$

separator

Turing Machine Encoding

Transitions:

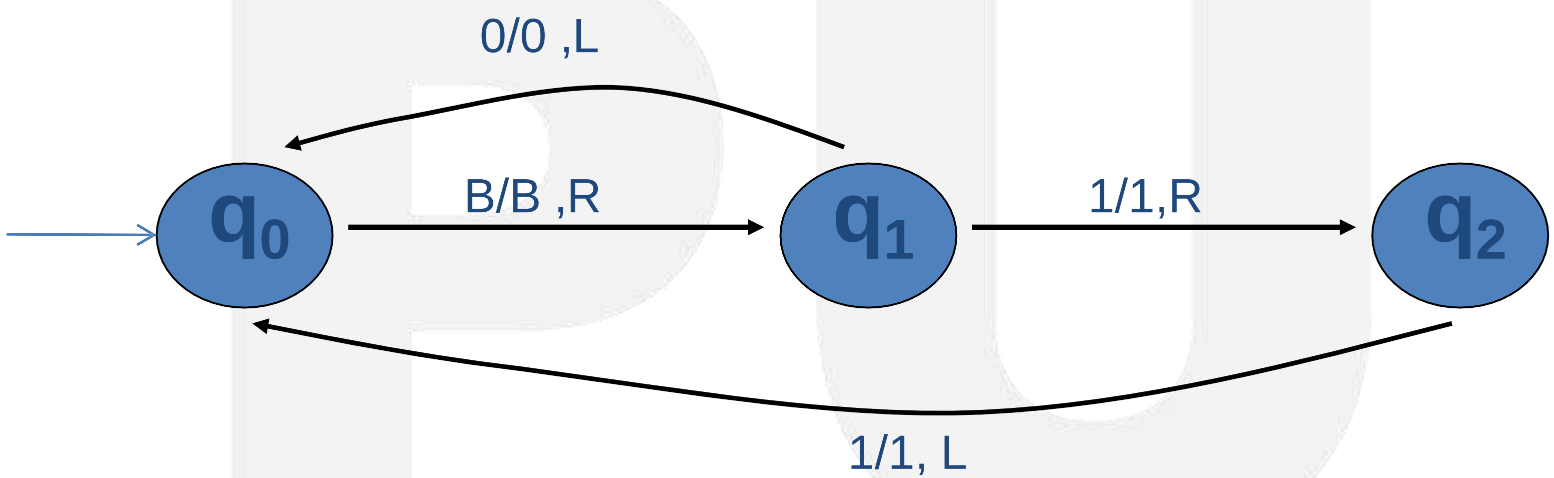
$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

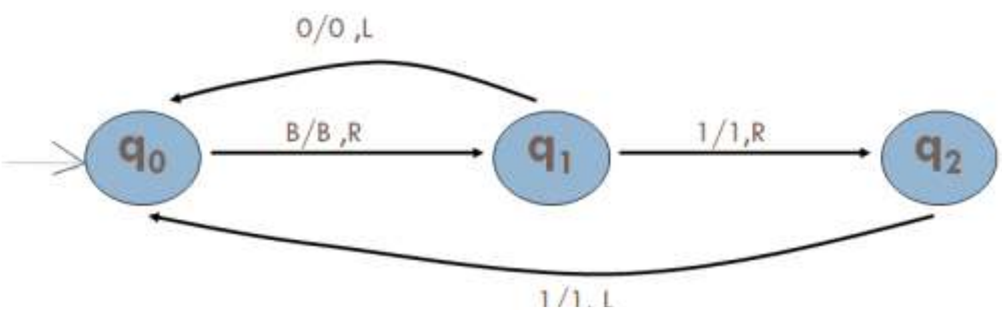
1 0 1 0 1 1 0 1 1 0 1 00 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1

separator

Turing Machine Example with Halts



Universal Turing Machine Example



$\delta(q_0, B) = [q_1, B, R]$ 1011101101111011

$\delta(q_1, 0) = [q_0, 0, L]$ 1101010101

$\delta(q_1, 1) = [q_2, 1, R]$ ....

$\delta(q_2, 1) = [q_0, 1, L]$ ....

000*1011101101111011*00*1101010101*00....00....000

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*110101010101*00*….*00*….*000

tape 1:

0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	.	.	.

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*1101010101*00*….*00*….*000

tape 1:

0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	.	.	.

tape 2:

1	B				

current / updated state

Put q_0 as the start state

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*1101010101*00....00....000

tape 1:

0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	.	.	.

tape 2:

1	B				

current / updated state

tape 3:

B	1	0	1	1	B

string *w*

Three-tape Deterministic Universal Turing Machine

000*1011101101110111011*00*1101010101*00*….*00*….*000

tape 1:

0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	.	.	.

tape 2:

1	B				

current / updated state

↑

tape 3:

B	1	0	1	1	B

string *w*

↑

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*1101010101*00....00....000

tape 1:

0	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	1	0	0

tape 2:

1	1	B			

current / updated state

tape 3:

B	1	0	1	1	B

string *w*

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*1101010101*00*….*00*….*000

tape 1:

.	.	.	1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	.	.

tape 2:

1	1	1	B		

current / updated state

↑

tape 3:

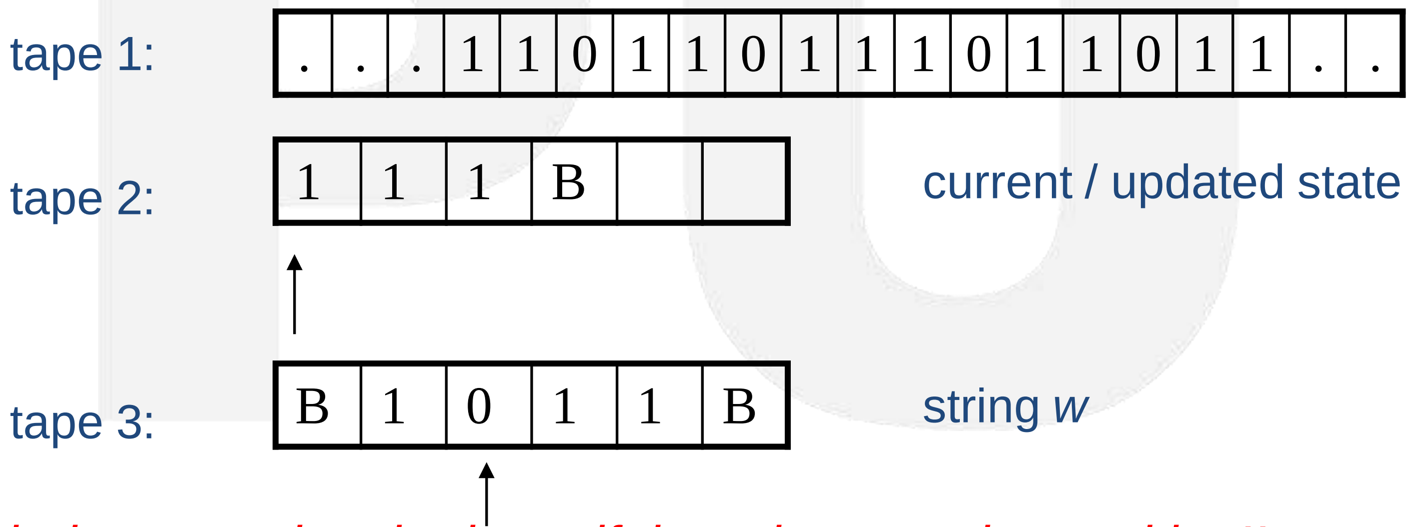
B	1	0	1	1	B

↑

string *w*

Three-tape Deterministic Universal Turing Machine

000*101110110111011*00*1101010101*00*....*00*....*000



U halts accepting the input if there is no such transition!!

Church Turing Thesis

- What does computable mean?
- Meaning of term Computable was given by:
  - Alonzo church (Lambda calculus)
  - Allen Turing (Turing Machine)
- Turing Machine carried out any *algorithmic procedure* that can be carried out at all, by a human computer or a team of humans and an electronic computer. This statement usually referred to as Church’s thesis, or the Church-Turing thesis.

Church Turing Thesis

- Here is an informal summary of some of the evidence.
  - Humans normally work with a two-dimensional sheet of paper. A TM tape could be organized so as to simulate two dimensions; one likely consequence would be that the TM would require more moves to do what a human could do in one.
  - Various enhancements of the TM model have been suggested to make the operation more like that of a human computer, or more convenient & efficient. The multitape TM is an example.

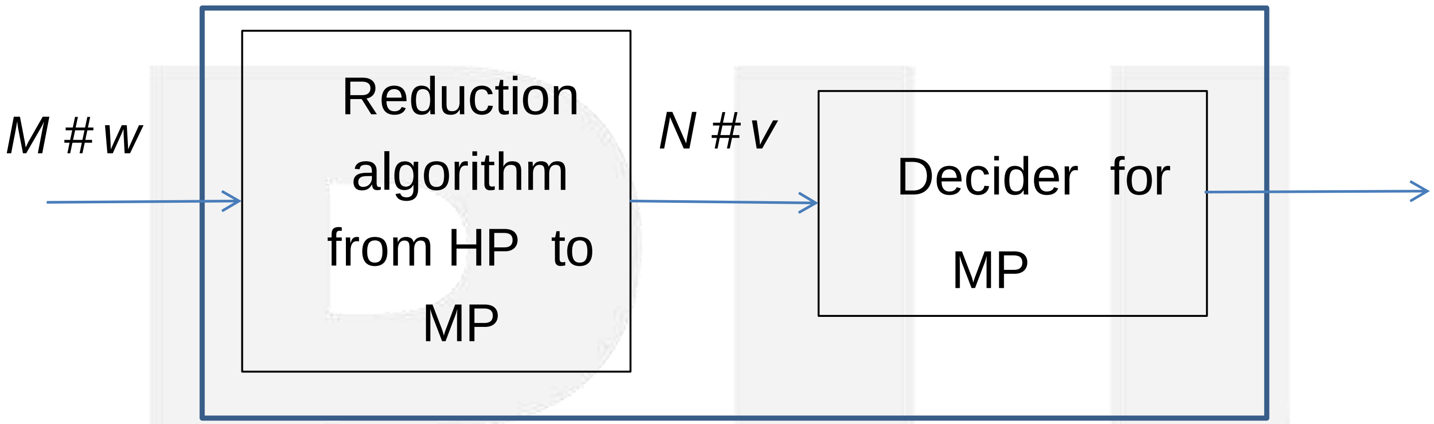
Church Turing Thesis

- ❖ Other theoretical models includes abstract machines with two stacks or with a queue.
- ❖ Since the introduction of the Turing machine, no one has suggested any type of computation that ought to be included in the category of "*algorithmic procedure*" and cannot be implemented on a TM.

Diagonalization

- Any Turing machine M can be encoded as a string over {0, 1}.
- Any input w for M can also be encoded as a binary string.
- Two important problems (languages)
  - MP = $\{M \# w \mid M$ accepts input $w\}$.
  - HP = $\{M \# w \mid M$ halts on input $w\}$.
- A total TM (or decider) halts on all inputs.
- Both these problems are Turing-recognizable (r.e.).
- By a diagonalization argument, we have proved HP to be non-recursive.
- No decider can exist for HP, no matter how intelligent Turing machines are.
- A similar diagonalization argument can be made for MP.

Reduction



- We want to prove the undecidability of the MP.
- A reduction algorithm converts an input $M \# w$ for HP to an input $N \# v$ for MP.

Reduction

- The reduction algorithm is a total Turing machine (halts after each conversion).
- *N* accepts *v* if and only if *M* halts on *w*.
- If MP has a decider *D*, then the reduction algorithm followed by *D* decides HP.
- Contradiction. So a decider of MP cannot exist.

The Reduction Algorithm

- Input : M and Output: M and v.
- Steps:
  - Add a new accept state t' and a new reject state r' to M.
  - Mark the old accept and reject states t and r of M as non-halting.
  - Add transitions $\delta(t, *) = (t', *, R)$ and $\delta(r, *) = (t', *, R)$.
  - Take $v = w$.
  - Convince yourself that a total TM can transform (M, w) to (N, v).
  - N always rejects by looping (no transition to r' added).
  - If M halts after accepting (in state t) or rejecting (in state r), N runs one more step to jump to t' and accepts.
  - If M loops on w, N also loops.
  - M halts on $w \Longleftrightarrow N$ accepts v

Direction of Reduction

❖From a problem already known to be undecidable to a problem which we want to prove to be undecidable.

❖A valid reduction from MP to HP

Input: $M \# w$ for the membership problem
Output: $N \# v$ for the halting problem

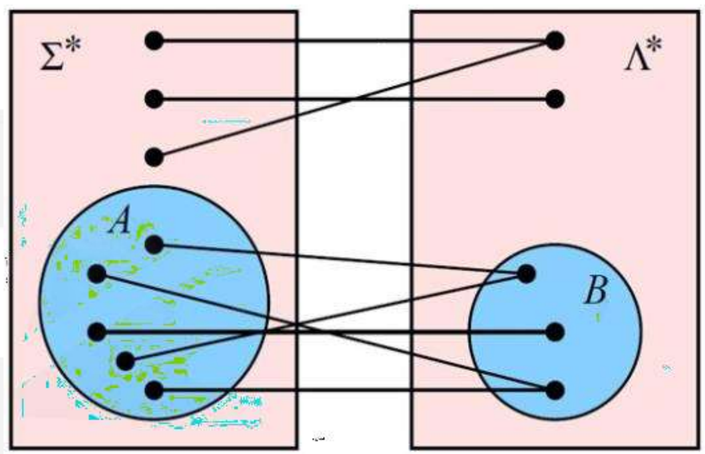
1. Keep the accept state t of M the same in N.
2. Create a new reject state r' for N, and transitions $\delta(r, *) = (r, *, R)$ (loop in state r).
3. Take $v = w$.

Direction of Reduction

❖ This is not an undecidability proof for MP. A decider for MP may not be forced to use a (hypothetical) decider for HP.

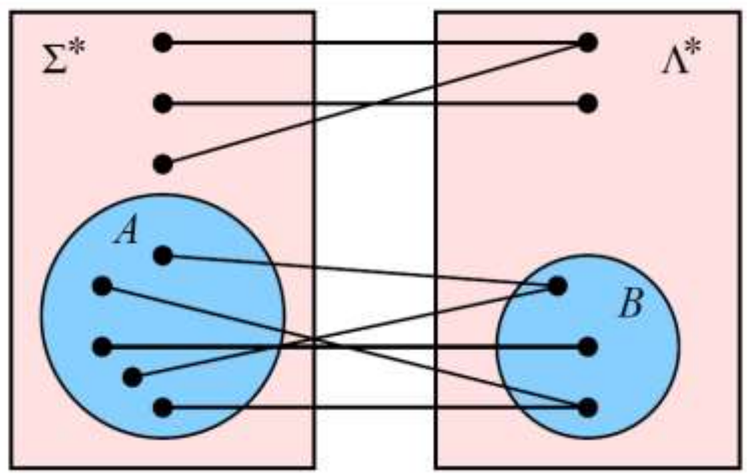
❖ If MP was proved to be undecidable, this reduction proves the undecidability of HP.

Formal Definition of Reduction



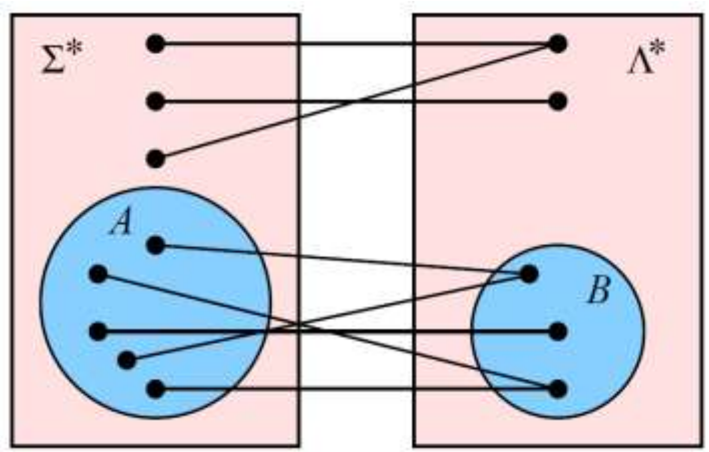
- Let $A \subseteq \Sigma^*$ and $B \subseteq \Lambda^*$ be languages.
- Consider a map $\sigma : \Sigma^* \to \Lambda^*$.
- If $w \in A$, then $\sigma(w) \in B$.
- If $w \in \Sigma^* / A$, then $\sigma(w) \in \Lambda^* / B$.

Formal Definition of Reduction



- σ need not be injective.
- A Turing machine *R* implements σ.
- On every input *w*, the TM *R* halts after correctly computing σ(*w*).
- We call *R* a reduction algorithm

Formal Definition of Reduction



- σ is a reduction from *A* to *B*.
- Notation: $A \leq B$ (many-to-one reduction) or $A \leq B$ (Turing reduction).
- The membership problem for *A* is no more difficult than the membership problem for *B*.
- Example: $HP \leq MP$ and $MP \leq HP$.

Notes on Reduction

- A language L can be rephrased as the membership problem:
- Given $w \in \Sigma^*$, is $w \in L$?
- We talk about reduction of one problem to another.
- For problems P, Q, we can write $P \leq Q$.
- A reduction algorithm is supposed to convert an instance of P to an instance of Q.
- A reduction algorithm makes no effort to solve either P or Q.

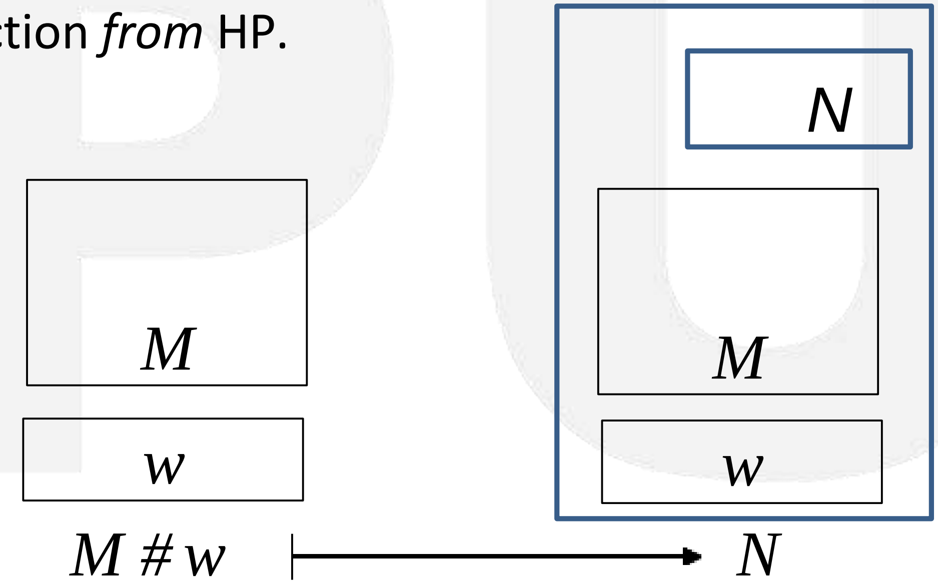
Notes on Reduction

- Two uses of reduction $P \le Q$:
- Given a solver for Q, use this solver as a subroutine to solve P. This is one way of solving P, not the only or the most efficient way.
- If no solver for P exists, then no solver for Q can exist.

Reduction Example 1

❖ **Proposition:** The problem whether a given Turing machine *M* accepts the null string ε is undecidable.

❖ *Proof* Use reduction *from* HP.



Reduction Example 1

- **Input:** *M* and *w* (an instance of HP).
- **Output:** A Turing machine *N* that accepts ε if and only if *M* halts on *w*.
- *N* can use *M* and *w* in any manner it likes. These are part of its finite control.
- Behaviour of *N* on input *v*:
  - Erase input *v*.
  - Write the string *w* on the tape.
  - Simulate *M* on *w*.
  - If the simulation halts, accept *v*.

Reduction Example 1

- N accepts its input $v \Leftrightarrow M$ halts on w.

$$L(N) = \begin{cases} \Sigma^* & \text{if } M \text{ halts on } w, \\ \Phi & \text{if } M \text{ does not halt on } w. \end{cases}$$

- In particular, N accepts ε ⇔ M halts on w.

Reduction Example 1

The same proof can be used to prove that the following problems are also undecidable.

❖**Proposition:** Let *w* be a fixed string over Σ. The problem whether a given Turing machine *M* accepts *w* is undecidable.

❖**Proposition:** The problem whether a given Turing machine *M* accepts any string at all is undecidable.

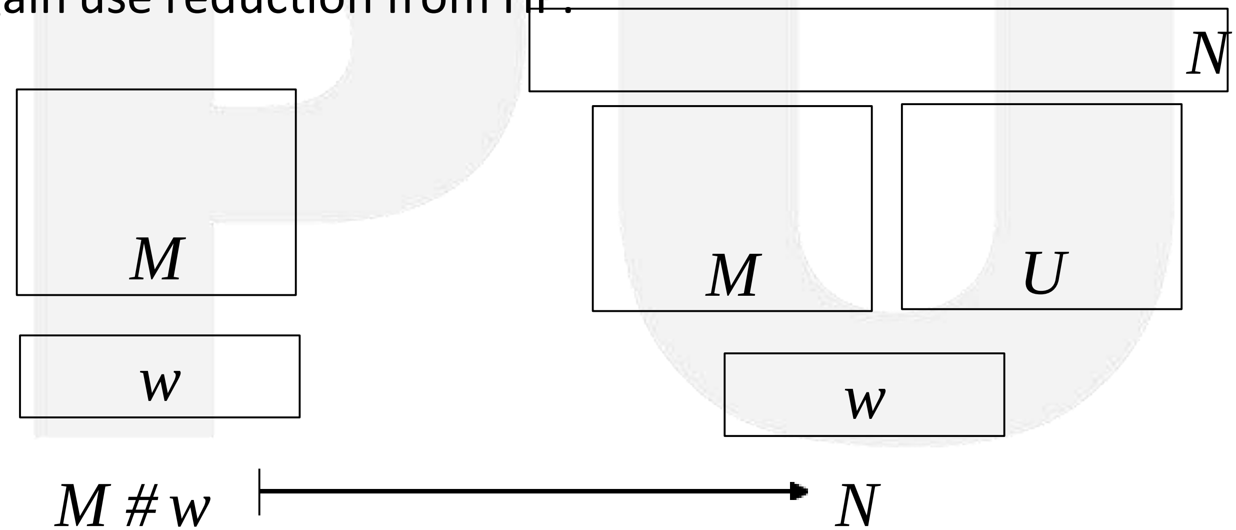
❖**Proposition:** The problem whether a given Turing machine *M* accepts all the strings over Σ is undecidable.

❖**Proposition:** The problem whether a given Turing machine *M* accepts only finitely many strings is undecidable.

Reduction Example 2

❖ **Proposition:** The problem whether the language of a given Turing machine *M* is regular is undecidable.

❖ *Proof* Again use reduction from HP.



Reduction Example 2

- **Input:** An instance for HP (*M* and *w*)
- **Output:** A Turing machine *N* whose language is regular if and only if *M* halts on *w*.
- *N* has the information of *M* and *w* embedded in its finite control.
- *N* embeds the information of another fixed Turing machine *U* in its finite control.
- Take any language *L* that is recursively enumerable but not recursive.
- Take any TM *U* whose language is *L*.
- For example, if *L* = MP, then *U* is the Universal Turing Machine.

Reduction Example 2

- *N* upon the input of *v*, does the following.
  - Store *v* on a separate tape/track.
  - Write *w* on the tape, and simulate *M* on *w*.
  - If the simulation halts, do:
    - Simulate *U* on *v*.
    - If *U* accepts *v*, accept *v*.
  - *N* accepts *v* if and only if both the following conditions hold.
    - *M* halts on *w*.
    - *U* accepts (and halts) on *v*.

Reduction Example 2

- $L(N) = \begin{cases} L & \text{if } M \text{ halts on } w \\ \Phi & \text{if } M \text{ does not halt on } w \end{cases}$

❖ Φ is regular, but A is not regular.

Reduction Example 2

- Let $L_2 = \{N \mid \mathrm{L}(N) \text{ is regular}\}$.
- We have a reduction from HP to the complement $\overline{L}_2$.
- This proves that $\overline{L}_2$ is not recursive.
- But recursive languages are closed under complementation, so L_2 is not recursive too.
- Alternative argument:
  - Let $\overline{L}_2$ have a decider $\bar{D}$.
  - Then L_2 has a decider D that simulates $\bar{D}$ and flips the decision of $\bar{D}$.
  - The above reduction followed by D decides HP.

Reduction Example 2

❖The same reduction can be used to prove the following undecidability results.

❖**Proposition:** The problem whether the language of a given Turing machine M is finite is undecidable.

❖**Proposition:** The problem whether the language of a given Turing machine M is context-free is undecidable.

❖**Proposition**: The problem whether the language of a given Turing machine M is context- sensitive is undecidable.

❖**Proposition:** The problem whether the language of a given Turing machine M is recursive is undecidable.

Reduction in Example 2

❖**Note:** The problem whether the language of a given Turing machine *M* is recursively enumerable is trivially decidable.

A Theorem about Reduction

- **Theorem:** Let A, B be languages along with a reduction $A \leq B$. If B is r.e., then A is also r.e.
- Contrapositively, if A is not r.e., then B is also not r.e.

Proof

- Let σ be the reduction map from A to B.
- Let B = L (N) for a Turing machine N.
- A recognizer M for A can be designed as follows.
- On an input w, M does the following:
    - Compute σ(w) from w.
    - Run N on σ(w).
    - Accept if and only if N accepts σ(w).

Another Theorem about Reduction

❖ **Theorem:** Let A, B be languages along with a reduction $A \leq B$. If B is recursive, then A is also recursive.Contrapositively, if A is not recursive, then B is also not recursive.

Proof

❖ Let B be recursive.

❖ Let σ be the reduction map $A \leq B$.

❖ Since B is r.e., A is r.e. too (by the previous theorem).

❖ σ is also a reduction map for $\bar{A} \leq \bar{B}$

❖ $\bar{B}$ is recursive and so r.e.

❖ By the previous theorem, $\bar{A}$ is r.e. too.

❖ Since A and $\bar{A}$ are both r.e., A is recursive.

Rice's Theorem

Properties of Regular Expression Languages

- Let RE ={L(*M*) | *M* is a Turing machine}
- RE is the class of all RE languages.
- A property of RE sets is a map (Either T or F)

$$P : \text{RE} \rightarrow \{T, F\}.$$

- Example 1: Emptiness is a property defined as

$$P_{EMP}(L) = \begin{cases} T & \text{if } L = \emptyset \\ F & \text{if } L \neq \emptyset \end{cases}$$

Rice's Theroem

- R.E. languages are specified by Turing machines.
- Properties is also specified by Turing machines.
- Example 2: The emptiness property is specified by any member of

$P_{EMP} = \{M \mid L(M) = \Phi\}$ where M is a Turing Machine

Rice's Theorem

Examples of Various Properties

- Finiteness property: Any member of $\{M \mid L(M) \text{ is finite}\}$
- Regularity property: Any member of $\{M \mid L(M) \text{ is regular}\}$
- Context-free property: Any member of $\{M \mid L(M) \text{ is context free}\}$
- Acceptance of a string: Any member of $\{M \mid 01011000 \in L(M)\}$
- Full-ness property: Any member of $\{M \mid L(M) = \Sigma^*\}$

Rice's Theorem

- We specify a property for a single Turing machine, the language of Turing Machine has that property
- Properties of Turing Machines are Properties of Regular Expression Sets, not of Turing Machines.
- A property must be independent of the representative machine.

Types Of Properties

- **Types of Properties**
- Trivial Properties

  The constant map RE → {*T*, *F*} taking all *L* ∈ RE to *T*

  The constant map RE → {*T*, *F*} taking all *L* ∈ RE to *F*

- *Non Trivial Properties*

  Any other Properties it is called NonTrivial Properties it means

  The constant map RE → {*T*, *F*} taking all *L* ∉ RE to *T or F*

Types Of Properties

❖ *Monotone Properties*

- ❖ Assume that $F \leq T$
- ❖ Whenever $A \subseteq B$, we have $P(A) \leq P(B)$.
- ❖ Examples of monotone properties: L (M) is infinite, L (M) = Σ^*.
- ❖ Examples of non-monotone properties: L (M) is finite, L (M) = Φ

Rice's Theorem Part-1

Theorem

Any non-trivial property P of r.e. languages is undecidable. In other words, the set $\Pi = \{N \mid P(L(N)) = T\}$ is not recursive.

Proof

- Let *P* be a non-trivial property of r.e. languages.
- Suppose $P(\Phi) = F$ (the other case can be analogously handled).
- Since *P* is non-trivial, there exist $L \in$ R.E. , L ƒ $\neq \Phi$, such that $P(L) = T$.
- Let *K* be a Turing machine with $L(K) = L$.
- We make a reduction from HP to Π.

Rice's Theorem: The Reduction HP≤m Π

- **Input:** *M* # *w* (an instance of HP)
- **Output:** A Turing machine *N* such that *P*(L(*N*)) = *T* if and only if *M* halts on *w*.
- Behaviour of *N* on input *v*:
  - Copy *v* to a separate tape.
  - Write *w* to the first tape, and simulate *M* on *w*.
  - If the simulation halts:
    - Simulate *K* on *v*.
    - Accept if and only if *K* accepts *v*.

Rice's Theorem: The Reduction HP≤m Π

- If M halts on w, L(N) = L(K) = L.
- If M does not halt on w, L(N) = Φ
- $P(L) = T$ and $P(\Phi) = F$.

Rice's Theorem: Part 2

Theorem

No non-monotone property P of R.E. languages is semidecidable. In other words, the set Π = {N | P(L (N)) = T} is not recursively enumerable.

- *Proof*
- *Here, P* is non-monotone properties. So there exist R.E. languages L_1 and L_2 such that $L_1 \subseteq L_2$,

  $P(L_1) = T$,

  $P(L_2) = F$.
- Take Turing machines M_1, M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$.
- We supply a reduction from HP to Π.
- The reduction algorithms embeds the information of
  - M, w, M_1, and M_2 in the finite control of N.

Rice's Theorem: Part 2: The Reduction HP≤Π

- **Input:** $M \# w$.
- **Output**: A Turing machine N such that $P(L(N)) = T$ if and only if M does not halt on w.
- Behavior of N on input v:
  - Copy v from the first tape to the second tape, and w from the finite control to the third tape.
  - Run three simulations in parallel (one step of each in round-robin fashion)
- M_1 on v on the first tape, M_2 on v on the second tape, M on w on the third tape.

Rice's Theorem: Part 2: The Reduction HP≤Π

- Accept if and only if one of the following conditions hold:
  - M_1 accepts v,
  - M halts on w, and M_2 accepts v.

- M does not halt on $w \Rightarrow N$ accepts by (1) $\Rightarrow$ L(N) = L (M_1) = L_1.
- If M halts on w, N accepts if either M_1 or M_2 accepts v. In this case,
- L (N) = L(M_1) ∪ L(M_2) = $L_1 \cup L_2 = L_2$ (since $L_1 \subseteq L_2$).

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in