CS 3313
Foundations of Computing:

Undecidable Problems and Rice's Theorem

http://gw-cs3313.github.io

1

Summary…..

- Closure properties of Recursive and RE languages
- Concept of Decidability….
  - problem is *decidable* = there is an algorithm (TM that always halt) to answer it
  - Otherwise problem is *undecidable* = no algorithm to solve it
  - Decidable/Solvable and Solvable/Unsolvable mean the same thing
- Example of an Undecidable Problem
- Reducibility – prove other problems are undecidable
- Next: More examples of undecidable problems
  - Read the examples and exercises in the textbook
- and (last result in course)…Rice's Theorem: a powerful result that can be used to show (easily) that many properties of RE languages are undecidable.

2

A key proof technique: Reducability

- Reducibility of a problem A to problem B
- Given two problems A and B,

  problem A is <u>reducible</u> to problem B if an algorithm for solving B can be used to solve problem A

  – Therefore, solving A cannot be harder than solving B

  - *If A is undecidable and A is reducible to B, then B is undecidable*
- Idea: If you had a black box that can solve instances of B, can you solve instances of A using calls to this Black box.
  - The black box is the assumed Algorithm for B.

3

Our current "collection" of undecidable languages

1. We proved that L_d *is not decidable (it is not even r.e.)*
   - $L_d = \{w \mid w = w_i$ and M_i does not accept $w_i\}$.
2. If L_d is not recursive then its complement $\overline{L_d}$ is not recursive, i.e, it is undecidable
   - $\overline{L_d} = \{w \mid w = w_i$ and M_i accepts $w_i\}$.
3. $L_u = \{ <M,w> \mid M$ accepts $w \}$ ....*Halting Problem*
   - We reduced $\overline{L_d}$ to L_u

4

4

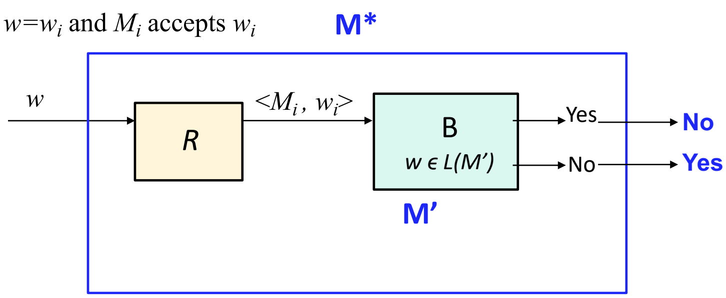
Quick Recap - Proof:
L_u is not Recursive: Proof – construct Algo R

- algorithm R (the reduction): Input is w and output is $<M_i, w_i>$

1. Use the canonical ordering algorithm to find i, where $w = w_i$
2. Generate binary representation of i, this is the code for M_i
3. Concatenate code for M_i and w_i to generate $<M_i, w_i>$

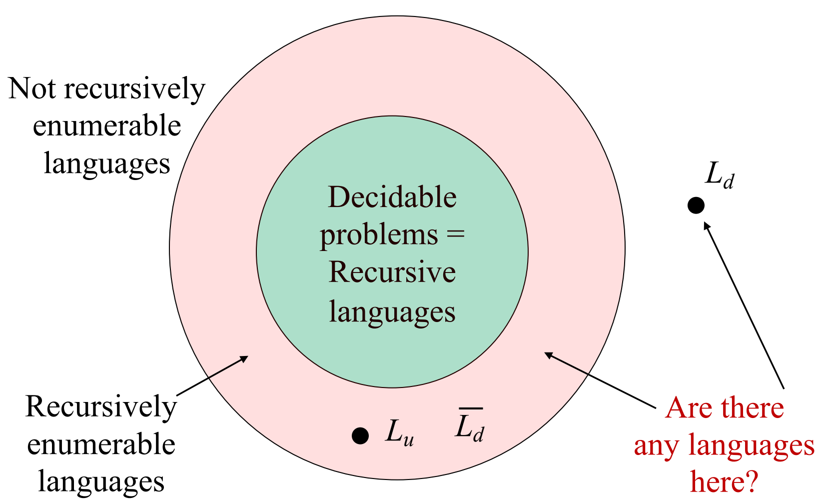
- *Send to hypothetical algorithm B for Halting Problem*
  - *B accepts if and only if w is in* $\overline{L_d}$

$w \in L(M^*)$ iff $w = w_i$ and M_i accepts w_i



5

Bullseye Picture



6

6

Halting Problem ? Other related problems….

- Does M halt on w ? = is L_u decidable ✓
  - Original statement of the halting problem was slightly different but shown to be equivalent.
- Can we check if a program halts on all inputs = Does M halt on all inputs ?
- Is L(M) empty ? – i.e., does the program compute anything?
- Is $L(M_1) = L(M_2)$ – i.e., are two programs equivalent ?
  - *Is this what an autograder program does ?*
- Is $L(M_1) \subseteq L(M_2)$ – i.e., does program M_2 compute everything that M_1 computes?
- Can we check if a program enters a 'checkpoint' = Does M enter a state q ?
  - Variation of homework 8 question.

7

Example 1: Does M halt on all inputs ?

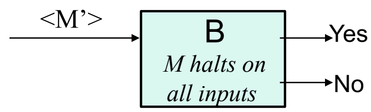
- Input: any program/TM M
- Question: Is there an algorithm that can determine/decide if M halts for all inputs sent to M
  - Note: you cannot test by running all inputs since there are an infinite number of inputs !!
- Prove by reducing Halting problem to this problem
- Starting point – assume this problem is decidable

  implies there is an algorithm B to solve this problem

8

Example 1: Does M halt on all inputs?

- Assume it is decidable => there is an algorithm B to solve it
  - Input to B ?
- Prove Halting problem is reducible to this problem
  - Halting problem: $\{< M, w > \mid M \text{ accepts } w\}$
  - Input to Halting problem ?
- Therefore, what should reducibility algo R do ?



Example 1: Does M halt on all inputs ?

- Algorithm R: Input is $<M,w>$ and output is *M’*

1. *Check length of w. Let length =n*
   - $w = a_1a_2\ldots a_n$ *– note that this info is available from input* $<M,w>$
2. *Create n+2 states* $q_1, q_2, \ldots q_{n+2}$
3. *Add (n+2) to indices of all states in M*
   - *Therefore start state of M now becomes* q_{n+3} *(original* q_1 *with n+2 added)*
4. *Start machine M’ (it first write string w on tape)*
5. *Accept if M accepts – final state of M’ is final state of M*

- To illustrate one complete reducibility proof, let’s examine how to construct a TM for Algorithm R

Example 1: Does M halt on all inputs?

- Algorithm R (generate M'): Input is <M,w> and output is M'
- Details of step 4: $w = a_1 a_2 \ldots a_n$

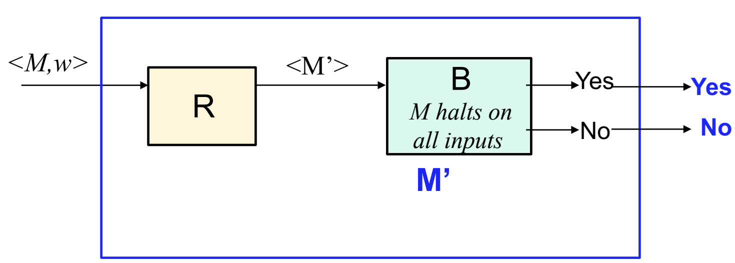
1. $\delta(q_1, B) = (q_2, \$, R)$ for any X in Tape alphabet /* print marker $ at left end */
2. $\delta(q_2, B) = (q_3, a_1, R)$ for any X /* replace first symbol of tape with first symbol of w */
3. …
4. $\delta(q_i, B) = (q_{i+1}, a_{i-1}, R)$ /* write $(i-1)$ symbol of w to tape in state q_i */
5. $\delta(q_{n+1}, X) = (q_{n+2}, a_n, L)$ /* write the last symbol of w */
6. $\delta(q_{n+2}, X) = (q_{n+2}, X, L)$ for any X except $, /* skip/move left to the $ marker */
7. $\delta(q_{n+2}, \$) = (q_{n+3}, B, R)$ /* go to start state of M */
8. Add (n+2) to indices of all states in M and "update" transition function, i.e.,

- Ex: replace $\delta(q_j, X_1) = (q_k, X_2, L)$ with $\delta(q_{j+n+2}, X_1) = (q_{k+n+2}, X_2, L)$

11

Example 1: Does M halt on all inputs is undecidable

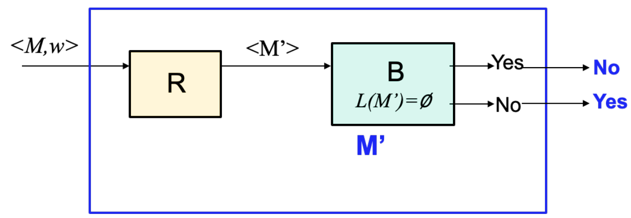
- Input: any program/TM M
- Question: Is there an algorithm that can determine/decide if M halts for all inputs sent to M
- Reducibility: Halting problem (L_u) is reducible to this problem, therefore this problem is undecidable



12

Example 2: L = { <M> | M is a TM and L(M)=Ø }

- Question: Given a Turing machine M, does M accept any input ? (i.e., does M accept the empty set).
- Reducing Halting problem L_u to Emptiness problem:
  - Assume Emptiness problem is decidable – implies there is an algo B that solves it
  - Construct algorithm R, such that testing for emptiness of M' using hypothetical algorithm B will give answer to "*M* accepts *w*".
- Comment: Simply sending <*M*> to algorithm B can tell us if L(M) is empty. But if it is not empty then it does not mean *w* is accepted by M
- Therefore have to send in a modified TM M' to Algo B, and emptiness of M' determines answer to "M accepts w"



13

Example 2: L = { <M> | M is a TM and L(M)=Ø }

- Comment: What if instead of "M is a TM" we replace the problem with "M is a DFA"…..
  - Question: Is this problem decidable ?.......
- Undecidability proof: Reducibility algorithm must generate M' such that M' accepts any string *x* iff M accepts *w*
- Any parallels with the reducibility steps for the problem "Does M halt on all inputs" ?

14

Example 2: L = { <M> | M is a TM and L(M)=Ø }

- Key idea in constructing M': design M' such that

  M' accepts Ø iff M does not accept *w*, and

  M' accepts all strings ({0,1}*) iff M accepts *w*.

  - Design M'so that machine erases its input at the start, then writes w on the tape and starts M
- Modified TM *M'*:

1. For any input on the tape, replace *x* by *w*
2. Go to start state of *M*
3. *M'* accepts any input iff *M* accepts *w* – final state of *M'* is final state of *M*

- *So what should reducibility algo R do:….generate M' !*

15

Example 2: L = { <M> | M is a TM and L(M)=Ø }

- Algorithm R: Input is <*M,w*> and output is *M'*

1. Check length of w. Let length =*n*
   - $w= a_1a_2 \ldots a_n$ – note that this info is available from input <M,w>
2. Create *n+3* states $q_1, q_2, \ldots q_{n+3}$
3. Add *(n+3)* to indices of all states in M
   - Therefore start state of M now becomes q_{n+4} (original q_1 with *n+3* added)
4. Start machine M', and replace any input *x* with string *w*
5. Accept if M accepts – final state of M' is final state of M

- Steps 1,2,3,5 similar to reducibility algorithm for "Does M halt on all inputs"

16

Example 2: L = { <M> | M is a TM and L(M)=Ø }

- TM to implement Algorithm R: Input is <M,w> and output is M'
- Details of step 4: $w = a_1a_2...a_n$

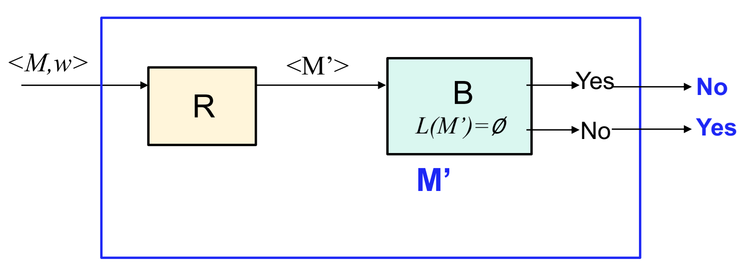
1. $\delta(q_1,X) = (q_2, \$, R)$ for any X in Tape alphabet /* print marker $ at left end */
2. $\delta(q_2,X) = (q_2, a_1, R)$ for any X except B /* replace first symbol of tape with first symbol of w */
3. …
4. $\delta(q_i,X) = (q_{i+1}, a_{i-1}, R)$ for any X except B /*write $(i-1)$ symbol of w to tape in state q_i */
5. $\delta(q_{n+2},X) = (q_{n+2}, B, R)$ /* erase tape to right of w */
6. $\delta(q_{n+2},B) = (q_{n+3}, B, L)$ /* now move left to the $ marker */
7. $\delta(q_{n+3},B) = (q_{n+3}, B, L)$
8. $\delta(q_{n+3},\$) = (q_{n+4}, B, R)$ /* go to start state of M */
9. Add (n+3) to indices of all states in M and "update" transition function, i.e.,

- Ex: replace $\delta(q_j,X_1) = (q_k, X_2, L)$ with $\delta(q_{j+n+3},X_1) = (q_{k+n+3}, X_2, L)$

17

Example 2: L = { <M> | M is a TM and L(M)=Ø } is undecidable

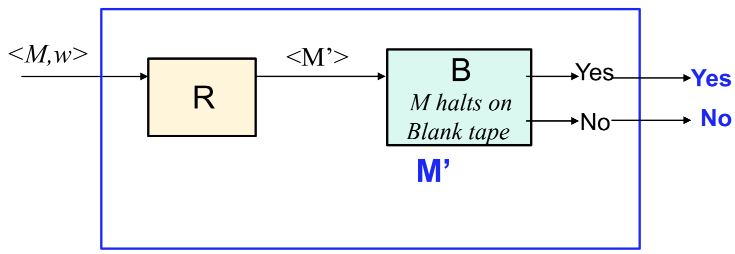
- If *M'* accepts any string *x*, then it erases tape, replaces with *w* and accepts *w* iff *M* accepts *w*.
- If *M'* does not accept any string iff *M* does not accept *w*
- Therefore *M accepts w iff L(M) is not empty*



18

Example 3: Blank tape acceptance {<M>| M halts on blank tape}

- Does M halt when started with the blank tape ?
- Can reduce this problem to the halting problem using a reducibility algorithm similar (identical?) to what we did for the Emptiness problem

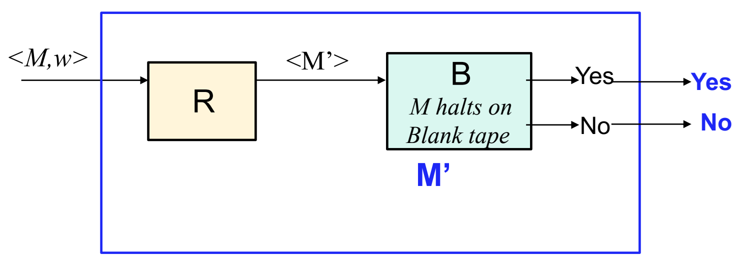


19

Example 3: Blank tape acceptance {<M>| M halts on blank tape} is undecidable

- Algorithm R: Generates modified TM *M' where*

1. M' first writes *w* to the tape.
2. Go to start state of *M*
3. *M'* accepts any input iff *M* accepts *w* – final state of *M'* is final state of *M*

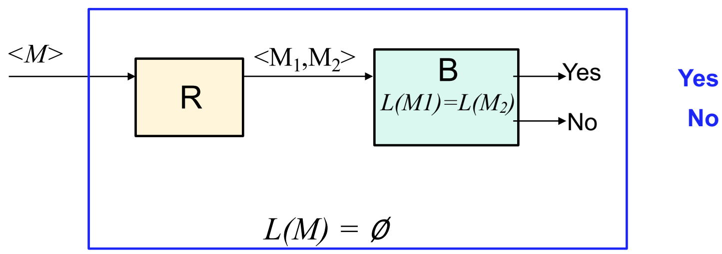


20

Example 4: Equivalence $\{\langle M_1\rangle\langle M_2\rangle \mid L(M_1) = L(M_2)\}$ is undecidable

- Are two programs equivalent ?

  same as asking not equivalent ($L(M_1) \neq L(M_2)$) ?
- Use set theory properties to show:
- Emptiness problem reducible to Equivalence problem
- (or alternatively show Subset testing ($L(M_1) = L(M_2)$?) reduces to Equivalence problem



21

Our current "collection" of undecidable languages

1. $L_d = \{w \mid w = w_i$ and M_i does not accept $w_i\}$.
2. If $\overline{L_d} = \{w \mid w = w_i$ and M_i accepts $w_i\}$.
3. $L_u = \{ \langle M,w \rangle \mid M$ accepts $w \}$ ....*Halting Problem*
4. Does M halt on all inputs ?
5. Is $L(M) = \emptyset$
6. Is $L(M_1) = L(M_2)$
7. Does M accept blank tape ?
8. Is $L(M_1) \subseteq L(M_2)$ *discussed in lab tomorrow*
9. Does M reach a specific state ?
10. Does M reach a specific ID (snapshot) ?
11. Does M print a specific symbol/output ?

22

22

More questions…Undecidable Problems

- Post Correspondence Problem
  - Is a given context-free grammar ambiguous?
  - Do two given CFG's generate the same language?
- Properties of r.e. sets:
  - Is the language accepted by a TM a regular language ?
    - Why bother: if you have a program to solve a problem, then can you implement the program on a Finite State machine ?...without using Pumping lemma!
  - Is the language accepted by a TM a finite language ?
  - Is the language accepted by a TM a context free language ?
  - ……

23

23

The Post Correspondence Problem (PCP)

- Given two sequences A,B of n strings on some alphabet Σ, for instance

$A = w_1, w_2, \ldots, w_n$ and $B = v_1, v_2, \ldots, v_n$

there is a Post correspondence solution (PC solution) for the pair (A, B) if there is a nonempty sequence of integers

$i, j, \ldots, k,$ such that $w_i w_j \ldots w_k = v_i v_j \ldots v_k$

- Example: assume A,B are

$w_1 = 11,$ $w_2, = 10111,$ $w_3 = 0$

$v_1 = 111$, $v_2, = 10,$ $v_3 = 10$

solution for this instance of (A, B) exists: sequence 2113

$w_2 w_1 w_1 w_3 =$ 10111 11 11 0

$v_2 v_1 v_1 v_3 =$ 10 111 111 10

24

The Undecidability of the Post Correspondence Problem

- The Post correspondence problem is to devise an algorithm that determines, for any (A, B) pair, whether or not there exists a PC solution
- For example, there is no PC solution if A and B consist of

$w_1 = 00,\ w_2, = 001,\ w_3 = 1000$ and $v_1 = 0,\ v_2, = 11,\ v_3 = 011$

- **Theorem:** the Post correspondence problem (PCP) is undecidable
  - result is crucial for showing the undecidability of various problems involving context-free languages

Undecidable Problems for Context-Free Languages

- The Post correspondence problem is a convenient tool to study some questions involving context-free languages
- The following questions, among others, can be shown to be undecidable
  - Given an arbitrary context-free grammar G, is G ambiguous?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) \cap L(G_2) = \varnothing$?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) = L(G_2)$?
  - Given arbitrary context-free grammars G_1 and G_2, is $L(G_1) \subseteq L(G_2)$?

Putting it all together: Automata, Grammars, Languages

- Different models of automata: DFA, PDA, TM
  - With increasing "power"
- Grammars to define languages….
  - Regular grammar = DFA
  - Context Free Grammar = PDA
  - Unrestricted grammar = TM
- How do they relate to each other……Chomsky Hierarchy

27

Automata Models (we studied)

- Finite State automaton DFA/NFA = accept Regular Languages
  - No storage
- Pushdown Automata/PDA = accept Context Free languages
  - Storage is a stack
- Turing Machine/TM = accept Recursively Enumerable (*r.e.*) lang.
  - Note: Recursive languages are contained in *r.e.* languages
  - Input place on tape and storage is the tape
  - Reads symbol on tape – changes state, writes to tape and moves tape head left or right

28

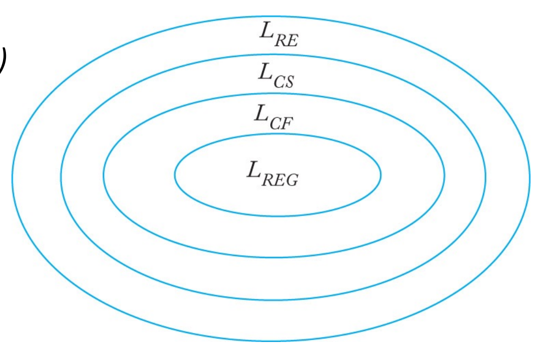
Grammars

- Definition: A grammar G (V,T,P,S) consists of:
  V – variables, T – terminals, S – start variables
  P – set of production rules
- By placing constraints on the type of production rules we get different classes of grammars
- **Unrestricted grammar**: Production rule is of the form $x \rightarrow \ y$ where $x,y \in (V \cup T)^+$
- **Context Sensitive**: $|x| < |y|$, $x,y \in (V \cup T)^+$
- **Context Free**: $x \rightarrow \ y$ *where* $x \in V$ and $y \in (VUT)^*$
- **Regular grammars**: $x \rightarrow \ y$ *where* $x \in V$ and at most one variable in y

29

The Chomsky Hierarchy

- The linguist Noam Chomsky summarized the relationship between language families by classifying them into four language types, type 0 (regular) to type 3 -- the *Chomsky Hierarchy*
- In terms of automata:
- *DFA < PDA < TM*
- *L(DFA)* $\subset$ *L(PDA)* $\subset$ *L(TM)*
- *=> If DFA accepts L*
  then PDA accepts L
  if PDA accepts L
  then TM accepts L



30

Automata Models and The Chomsky Hierarchy

- Theorem: If G is a Regular grammar then L(G) is accepted by a DFA/Reg.Expression.
  - If L is accepted by a DFA then L =L(G) for some regular grammar G.
- Theorem: If G is a context free grammar, then L(G) is accepted by a PDA.
  - If L is accepted by a PDA then L =L(G) for some CFG G
- Theorem: If G is any unrestricted grammar then L(G) is accepted by a Turing machine.
  - All grammars are unrestricted grammars
  - Properties of unrestricted grammars = Properties of languages accepted by Turing machines!
- Theorem: If G is a context sensitive grammar then L(G) is accepted by a linear bounded automaton
  - A linear bounded automaton is a subclass of Turing machines

31

Final Topic: Decision Properties of Recursively Enumerable Languages

- A language is *r.e.* if it is accepted by some TM M
  - A language is recursive if it is accepted by some TM M that always halts
  - A language is regular if it is accepted by some DFA M
  - A language is context free if it is accepted by some PDA M
- Consider statements that discuss properties of *r.e.* languages…

  i.e., the languages are sets of TM codes such that membership of <*M*> in the language depends only on *L(M)* and not on M itself.

- Question: What properties of r.e. languages are decidable
  - What properties can be determined by a program ?

32

Properties of a language

- Let P be a set of *r.e.* languages, each is a subset of $\{0,1\}^*$ (or any alphabet) – P is said to be a property of *r.e.* languages.

- a set L has property P if L is an element of P
  - Ex: If property P is "finiteness", then $\{a^i b^i \mid i < 10\}$ has property P but $\{a^i b^i \mid i > 0\}$ does not have property P
  - Ex: If property P is "regular language", then $\{a^* b^*\}$ has property P but $\{a^i b^i \mid i > 0\}$ does not have property P

- In terms of properties of the language accepted by a turing machine, let $L_P = \{ \langle M \rangle \mid L(M) \textit{ is in } P \textit{ and } M \textit{ is a TM}\}$
  - Note: L(M) means the language is *r.e.* since it is accepted by a TM

33

Trivial and Non-trivial Properties

- Non-trivial property: refers to a property satisfied by some but not all *r.e.* languages
- Trivial property: property satisfied by all or none (of *r.e.* languages)

- More formally:

- P is a *trivial property* if P is empty or P consists of all *r.e.* languages
- P is a *non-trivial property* otherwise.
  - Ex: Finiteness is a non-trivial property

34

Rice's Theorem

- Rice's Theorem: Any non-trivial property P of *r.e.* languages is undecidable.

- *Question asked is:* L_P = { <M> | L(M) is in P and M is a TM}*

- So how does one use this result……
  - Observe that this theorem is about *r.e.* languages -- languages which are accepted by a TM
    - We can give a TM to accept a language in this set of languages

35

Rice's Theorem: Example 1

- Determining if *{<M> | L(M) is finite}* is undecidable.
- What if M is a DFA ?

- Proof – we want to prove by using Rice's theorem.
- All we have to show is that this is a ***non-trivial property***
  - How ?

- Question is posed on the properties of the TM:

  L_P = *{ <M> | L(M) is finite and M is a TM}*

36

Rice’s Theorem: Example 1

- Determining if *{<M> | L(M) is finite}* is undecidable.
- Proof – we want to prove by using Rice’s theorem
- How ?
  - Prove that it is non-trivial property:
  - There is a TM M_1 such that $L(M_1)$ has the property (finiteness)
  - There is a TM M_2 such that $L(M_2)$ does not have the property

  - Design a M_1 to accept the language {aa} – finite
  - Design M_2 to accept the language $\{a^*\}$ – infinite

37

Rice’s Theorem: Example 2

Alternate proof for Emptiness Problem

- *{<M> | L(M) is empty}* is undecidable.
- Proof – we want to prove by using Rice’s theorem.
- To show that this is a non-trivial property
  - Design a TM that accepts empty set
  - Design a TM that accepts non-empty set

38