

# Computation History

The computation history for a TM on an input is the sequence of configurations that the machine goes through as it processes the input

**Note:** the computation history of TM  $M$  is a complete record of the computation performed by  $M$ .

# 22c:131 Limits of Computation

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c131>

The University of Iowa  
Department of Computer Science

Computation Histories

## Note

- Computation histories are finite sequences.
- If  $M$  does not halt on  $w$ , no accepting or rejecting computation history exists for  $M$  on  $w$ .
- Deterministic machines have at most one computation history on a given input.
- Nondeterministic machines may have many computation histories on a single input, corresponding to various computation branches.

## Computation histories

Let  $M$  be a TM and  $w$  an input string. An *computation history* for  $M$  on  $w$  is a sequence of configurations  $C_1, C_2, \dots, C_k$  where:

1.  $C_1$  is the start configuration of  $M$  on  $w$
2.  $C_{i+1}$  legally follows from  $C_i$  according to the transition function of  $M$

We distinguish two kind of histories:

- Accepting computation history:  $C_k$  is an accepting configuration of  $M$
- Rejection computation history:  $C_k$  is an rejecting configuration of  $M$

## Schematically, Figure 1

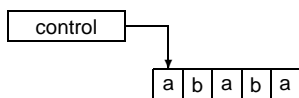


Figure 1: Schematic of a LBA

## Linear Bounded Automata

A LBA is a restricted type of TM wherein the tape head isn't permitted to move off the portion of the tape containing the input.

If the machine tries to move its head off either end of the input, the head stays where it is, in the same way that the head will not move off the left-hand end of the ordinary TM's tape.

## Observations

- Despite their memory constraint, LBA are quite powerful.
- Deciders for  $A_{DFA}$ ,  $E_{DFA}$ ,  $A_{CFG}$ ,  $E_{CFG}$ , all are LBAs.
- Most practical computing problems can be solved by an LBA.

## Note

- A LBA is a TM with a limited memory, as shown in Figure 1
- A LBA can only solve problems requiring memory that can fit within the tape used as input.
- Using a tape alphabet larger than input alphabet allows the available memory to be increased up to a constant factor.
- For an input of length  $n$ , the memory amount available is linear in  $n$  (hence, the name of the model).

## Lemma 5.8

Let  $M$  be an LBA with  $q$  states and  $g$  symbols in its tape alphabet. There are exactly  $qng^n$  distinct configurations of  $M$  for a tape of length  $n$ .

**Proof:**

- A configuration of  $M$  is a tuple  $(state, headPosition, tapeContents)$ .
- $M$  has  $q$  states
- Length of input is  $n$ , so the head can be in  $n$  positions
- Only  $g^n$  possible strings can appear on the tape

Hence,  $qng^n$  is the number of different configurations of  $M$

## A solvable problem

**Problem:** testing whether an LBA accepts an input is a solvable problem

**Language:** the language  $A_{LBA}$

$A_{LBA} = \{\langle M, w \rangle \mid M \text{ is an LBA that accepts string } w\}$

is decidable

## Loop Detection

- As it computes,  $M$  goes from configuration to configuration
- If  $M$  ever repeats a configuration, it would go on to repeat this configuration over and over again, and thus looping
- Since  $M$  is an LBA, it has only  $qng^n$  different configurations. Hence, if it performs a number of steps larger than  $qng^n$  and it did not halt, then  $M$  repeats a configuration and thus it loops

## Theorem 5.9

$A_{LBA}$  is decidable

**Proof idea:** to decide whether LBA  $M$  accepts  $w$

- Simulate  $M$  on  $w$ .
- During the simulation, if  $M$  halts and accepts or reject, accepts or rejects accordingly
- If simulation does not halt, detect the loop so that one can halt and reject

## Observations

- Theorem 5.9 shows that LBA and TM differ in one essential way: for LBAs acceptance problem is decidable while for TMs it is not.
- Certain other problems involving LBAs remain undecidable. Example: emptiness problem  
 $E_{LBA} = \{\langle M \rangle \mid M \in LBA \wedge L(M) = \emptyset\}$  is undecidable

## Proof

The algorithm that decides  $A_{LBA}$  is:

$L =$  "On input  $\langle M, w \rangle$ ,  $M$  and LBA,  $w$  a string:

1. Simulate  $M$  on  $w$  for  $qng^n$  steps or until it halts
2. If  $M$  halted, *accept* if  $M$  has accepted, and *reject* if  $M$  rejected.  
If  $M$  has not halted, *reject*

## Using $E_{LBA}$ decidability

- For a TM  $M$  and input  $w$  determine whether  $M$  accept  $w$  by constructing an LBA  $B$  and testing if  $L(B)$  is empty.
- Language recognized by  $B$  consists of all accepting computation histories of  $M$  on  $w$ . If  $M$  accepts,  $L(B)$  contains one string and so  $L(B) \neq \emptyset$ ; if  $M$  does not accept,  $L(B) = \emptyset$

Hence, if we can detect whether  $L(B)$  is empty we can determine whether  $M$  accept  $w$

## Theorem 5.10

$E_{LBA}$  is undecidable.

**Proof idea:**

- By reduction from  $A_{TM}$ , show that if  $E_{LBA}$  is decidable then so is  $A_{TM}$ .

## LBA $B$ works as follows:

- Breaks up  $x$  determining configurations  $C_1, C_2, \dots, C_k$
- Check the conditions:
  1.  $C_1$  is the start configuration for  $M$  on  $w$ , i.e.  $C_1 = q_0 w_1 w_2 \dots w_n$ ,  $q_0$  the start state of  $M$
  2. Each  $C_{i+1}$  legally follows from  $C_i$ , i.e., verify that  $C_i$  and  $C_{i+1}$  are identical except for the positions under and adjacent to the head in  $C_i$ . These positions must be updated according to the transition function of  $M$
  3.  $C_k$  is an accepting configuration for  $M$  on  $w$ , i.e.,  $C_k$  contains  $q_{accept}$  of  $M$

## Constructing $B$ from $M$ and $w$

- We need to show how a TM can obtain a description of  $B$  from  $M$  and  $w$
- Construct  $B$  to accept its input  $x$ , if  $x$  is an accepting computation history for  $M$  on  $w$ .

**Note:** the accepting configurations of  $B$  are represented as single strings with configurations separated by #, i.e.:

$$x = \#C_1\#C_2\#\dots\#C_k\#$$

## Proof

Suppose that TN  $R$  decides  $E_{LBA}$ . Construct TM  $S$  that decides  $A_{TM}$  as follows:

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

1. Construct LBA  $B$  from  $M$  and  $w$  as described above
2. Run  $R$  on input  $\langle B \rangle$
3. If  $R$  rejects, *accept*; if  $R$  accepts, *reject*."

## Note

- Item (2) above can be verified directly from the transition function  $\delta$ . This is shown in Figure 2 for  $\delta(q_3, a) = (q_5, x, R)$ .
- Item (2) above can be performed by zig-zagging between corresponding positions of  $C_i, C_{i+1}$ .

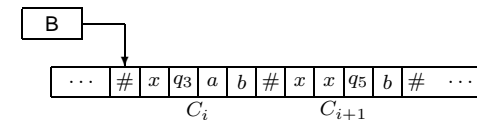


Figure 2: Checking computation history

## Theorem 5.13

**Problem:** Test whether a CFG generates all possible strings over its terminal alphabet  $\Sigma$ .

**Language:** the language

$$ALL_{CFG} = \{\langle G \rangle \mid G \text{ CFG} \wedge L(G) = \Sigma^*\}$$

**Theorem**  $ALL_{CFG}$  is undecidable.

**Proof idea:** by contradiction. Assume that  $ALL_{CFG}$  is decidable and show that then  $A_{TM}$  is then decidable.

## Note

- If  $R$  accepts  $\langle B \rangle$  then  $L(B) = \emptyset$ , thus  $M$  has no accepting computation on  $w$ , and  $M$  does not accept  $w$ . Consequently  $S$  rejects  $\langle M, w \rangle$
- If  $R$  rejects  $\langle B \rangle$ ,  $L(B) \neq \emptyset$ . Since the only string  $B$  can accept is an accepting computation history for  $M$  on  $w$  it means that  $M$  accepts  $w$ . Consequently  $S$  accepts  $\langle M, w \rangle$

This is a contradiction, so  $R$  cannot exist.

## Strategy

- An accepting computation history for  $M$  on  $w$  has the form  $\#C_1\#C_2\#\dots\#C_k\#$ .
- Hence,  $G$  generates all strings that:
  1. do not start with  $C_1$ ,
  2. do not end with an accepting configuration
  3. for some  $i$ ,  $C_i$  does not properly yield  $C_{i+1}$  under the rules of  $M$

## Proof idea, continuation

Using the decidability of  $ALL_{CFG}$ , one can devise the following decision procedure for  $A_{TM}$ :

- For a TM  $M$  and input  $w$  construct CFG  $G$  that generates all strings over the extended alphabet of  $M$  iff  $M$  does not accept  $w$ .
- If  $M$  does accept  $w$ ,  $G$  does not generate a particular string which is the accepting computation history for  $M$  on  $w$ .

**Note:**  $G$  is designed such that it generates all strings that are not accepting computation histories for  $M$  on  $w$ .

## Proof idea, continuation

Using the decidability of  $2E_{CFG}$ , one can devise the following decision procedure for  $A_{TM}$ :

- For a TM  $M$  and input  $w$  construct CFGs  $G_1$  and  $G_2$  such that  $L(G_1) \cap L(G_2)$  contains a particular string which is the accepting computation history for  $M$  on  $w$ , if  $M$  accepts  $w$ .
- If  $M$  does not accept  $w$ ,  $L(G_1) \cap L(G_2)$  does not contain anything.

## Theorem 5.14

**Problem:** Test whether the intersection of two CFLs is empty.

**Language:** The language

$$2E_{CFG} = \{\langle G_1, G_2 \rangle \mid G_1, G_2 \text{ CFG} \wedge L(G_1) \cap L(G_2) = \emptyset\}$$

**Theorem:**  $2E_{CFG}$  is undecidable.

**Proof idea:** By contradiction. Assume that  $2E_{CFG}$  is decidable and show that then  $A_{TM}$  is then decidable.

## Constructing $G_2$

$G_2$ : We generate a pair of configurations  $C_i \# C_{i+1}^R$ , if  $C_{i+1}$  legally follows from  $C_i$  according to the transition function of  $M$ .

- $S \rightarrow \#P \mid \#$
- $P \rightarrow A\#P \mid \epsilon$
- $A \rightarrow aAa$  for any  $a \in \Gamma$
- $A \rightarrow qaBpb$  if  $\delta(q, a) = (p, b, R)$
- $B \rightarrow aBa \mid \#$  for any  $a \in \Gamma$ .

## Constructing $G_1$

$G_1$ : At first, we start with the start configuration and then generate a pair of configurations  $C_i^R \# C_{i+1}$ , if  $C_{i+1}$  legally follows from  $C_i$  according to the transition function of  $M$ .

- $S \rightarrow \#q_0w\#P$
- $P \rightarrow A\#P \mid C\#$
- $A \rightarrow aAa$  for any  $a \in \Gamma$
- $A \rightarrow aqBbp$  if  $\delta(q, a) = (p, b, R)$
- $B \rightarrow aBa \mid \#$  for any  $a \in \Gamma$ .
- $C \rightarrow \dots$  an accepting configuration

## Application

Use the theorem 5.13 to show that

$EQ_{CFG} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFG and } L(G) = L(H)\}$   
is undecidable

**Solution:** Assume  $EQ_{CFG}$  were decidable. Construct a decider  $M$  for

$ALL_{CFG} = \{\langle G \rangle \mid G \text{ CFG, } L(G) = \Sigma^*\}$  by:

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

1. Construct CFG  $H$  such that  $L(H) = \Sigma^*$
2. Run the decider for  $EQ_{CFG}$  on  $\langle G, H \rangle$
3. If it accepts, *accept*; if it rejects, *reject*."

## Property of $G_1$ and $G_2$

- $G_1$  generates a sequence of configurations  
 $\#C_1\#C_2^R\#\dots\#C_{2k-1}\#C_{2k}^R\#\dots$  such that  $C_{2i+1}$  follows  $C_{2i}$  legally.
- $G_2$  generates a sequence of configurations  
 $\#C_1\#C_2^R\#\dots\#C_{2k-1}\#C_{2k}^R\#\dots$  such that  $C_{2i}$  follows  $C_{2i-1}$  legally.
- $L(G_1) \cap L(G_2)$  contains only an accepting computation history.

## A Puzzle Kind Description

**Puzzle:** is a collection of dominos like:

$\{[\frac{b}{ca}], [\frac{a}{ab}], [\frac{ca}{a}], [\frac{abc}{c}]\}$

**A match** is a list of these dominos (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the string of symbols on the bottom

**Example:**  $\{[\frac{a}{ab}], [\frac{b}{ca}], [\frac{ca}{a}], [\frac{a}{ab}], [\frac{abc}{c}]\}$

Reading off the top string we get: abcaaabc

Reading off the bottom string we get: abcaaabc

which is the same.

## Note

$M$  decides  $ALL_{CFG}$  assuming that a decider for  $EQ_{CFG}$  exists. Since we know that  $ALL_{CFG}$  is undecidable, a contradiction result

## Observation

For some puzzle finding a match may not be possible.

For example, the puzzle  $\{[\frac{abc}{ab}], [\frac{ca}{a}], [\frac{acc}{ba}]\}$  cannot contain a match

**Reason 1:** top string is longer than the corresponding bottom string

## Note

- We can also depict a match by deforming the dominos so that the corresponding symbols from top and bottom line up, Figure 3

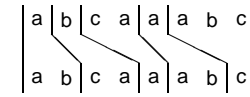


Figure 3: Deforming dominos

## Mathematical formulation

An instance of PCP is a collection of dominos:

$$P = \{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], \dots, [\frac{t_k}{b_k}]\}$$

A match is a sequence  $i_1, i_2, \dots, i_l$  of  $P$  components where

$$t_{i_1}t_{i_2}\dots t_{i_l} = b_{i_1}b_{i_2}\dots b_{i_l}$$

**PCP:** determine whether  $P$  has a match.

**Notation:**

$$PCP = \{\langle P \rangle \mid P \text{ instance of PCP with a match}\}$$

## Post correspondence problem:

Determine whether a collection of dominos has a match.

**Note:** this problem is unsolvable by algorithms

## Theorem 5.11

PCP is undecidable

**Proof idea:** by reduction from  $A_{TM}$  via accepting computation histories

- Show that from a TM  $M$  and input  $w$  we can construct an instance  $P$  of PCP where a match is an accepting computation history for  $M$  on  $w$
- If we could determine whether this instance of PCP has a match, we would be able to determine whether  $M$  accepts  $w$
- Since  $A_{TM}$  is undecidable we cannot determine whether  $P$  has a match

## Other formulations

An instance of PCP over  $\Sigma$  consists of two:

- Consider two lists:  $A = x_1, \dots, x_n$  and  $B = y_1, \dots, y_n$ ,  $x_i, y_i \in \Sigma^*$ ,  $1 \leq i \leq n$
- Do there exist a sequence  $i_1, \dots, i_m$  of integers such that  $x_{i_1} \dots x_{i_m} = y_{i_1} \dots y_{i_m}$ ?

**Note:** To show the correspondence, the lists  $A$  and  $B$  can be written:

$$\left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \dots, \begin{bmatrix} x_n \\ y_n \end{bmatrix} \right\}, x_i, y_i \in \Sigma^*.$$

## Technical details

- For convenience in the construction of  $P$  we assume that  $M$  on  $w$  never attempts to move its head off the left-hand end of the tape (we need to alter  $M$  to prevent such behavior)
- Alter PCP to require that a match starts with the first domino,  $\begin{bmatrix} t_1 \\ b_1 \end{bmatrix}$ . This is called modified PCP,  $MPCP$ :  
 $MPCP = \{ \langle P \rangle \mid P \text{ instance of PCP with a match that starts with first domino} \}$

**Note:** we will show later how to remove this requirement

## Constructing $P$

- Choose dominos in  $P$  so that making a match forces a simulation of  $M$  to occur
- In the match, each domino links a position or positions in one configuration with the corresponding one(s) in the next configuration

# Mapping $P'$ into $P$

- Let  $u = u_1u_2 \dots u_n$  a string of length  $n$ . Define the following three strings:

$$\star u = \star u_1 \star u_2 \star u_3 \star \dots \star u_n$$

$$u \star = u_1 \star u_2 \star u_3 \star \dots \star u_n \star$$

$$\star u \star = \star u_1 \star u_2 \star u_3 \star \dots \star u_n \star$$

- If  $P'$  were  $\{[\frac{t_1}{b_1}], [\frac{t_2}{b_2}], [\frac{t_3}{b_3}], \dots, [\frac{t_k}{b_k}]\}$  we let  $P$  be the collection  $\{[\frac{\star t_1}{\star b_1 \star}], [\frac{\star t_1}{b_1 \star}], [\frac{\star t_2}{b_2 \star}], [\frac{\star t_2}{\star b_2 \star}], \dots, [\frac{\star t_k}{b_k \star}], [\frac{\star t_k}{\star b_k \star}]\}$

# Proof

Let TM  $R$  decide the PCP and construct  $S$  deciding  $A_{TM}$ .

- Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$  and  $w \in \Sigma^*$
- The TM  $S$  that decides  $A_{TM}$  constructs an instance of the PCP that has a match iff  $M$  accepts  $w$ .
- $S$  constructs first an instance  $P'$  of MPCP. This construction has seven parts to be carried out shortly
- To convert  $P'$  to  $P$ , an instance of PCP, build the requirement of starting with first domino directly in the problem, so no need to state it explicitly

# Construction of $S$ : Part 1

Put  $[\frac{\#}{\#q_0w_1w_2 \dots w_n\#}]$  into  $P'$  as its first domino  $[\frac{t_1}{b_1}]$

**Note:** because  $P'$  is an instance of MPCP the match must begin with this domino. The bottom string begins correctly with  $C_1 = q_0w_1w_2 \dots w_n$ , the first configuration of the accepting computation history for  $M$  on  $w$ , as seen in Figure 4

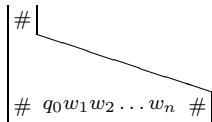


Figure 4: Beginning of the MPCP match

# Note

- Considering  $P$  an instance of the PCP, we see that only domino that could possibly start a match is the first one,  $[\frac{\star t_1}{\star b_1 \star}]$ . This is because it is the only domino where both the top and the bottom start with the same symbol, namely  $\star$ .
- Beside forcing the match to start with first domino, the presence of  $\star$ -s doesn't affect possible matches because they simply interleave with the original symbols, which occur in the even positions of the match
- The domino  $[\frac{\star \diamond}{\diamond}]$  is there to allow the top to add the extra  $\star$  at the end of the match

## Construction of $S$ : Part 2

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$  and  $w \in \Sigma^*$ . For every  $a, b \in \Gamma$  and every  $q, r \in Q, q \neq q_r$ , if  $\delta(q, a) = (r, b, R)$  put  $\begin{bmatrix} qa \\ br \end{bmatrix}$  into  $P'$

## Note

- In part 2,3,4 of the construction we add to  $P'$  dominos that perform the main part of the simulation
- Part 2 handle 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

## Construction of $S$ : Part 4

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$  and  $w \in \Sigma^*$ . For every  $a \in \Gamma$ , put  $\begin{bmatrix} a \\ a \end{bmatrix}$  into  $P'$

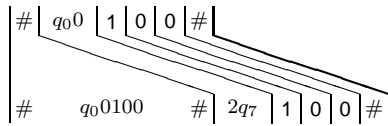
**Illustration:** to construct a hypothetical example showing what we did so far we choose:

$$\Sigma = \{0, 1, 2\}, \Gamma = \{0, 1, 2, \sqcup\}, w = 0100, \delta(q_0, 0) = (q_7, 2, R)$$

## Construction of $S$ : Part 3

Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$  and  $w \in \Sigma^*$ . For every  $a, b, c \in \Gamma$  and every  $q, r \in Q, q \neq q_r$ , if  $\delta(q, a) = (r, b, L)$  put  $\begin{bmatrix} cqa \\ rcb \end{bmatrix}$  into  $P'$

## The match constructed so far



Limits of Computation – p.50/58

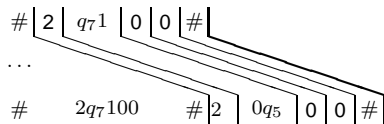
## Illustration, continuation

- Part 1 places  $[\frac{\#}{\#q_00100\#}] = [\frac{t_1}{b_1}]$  in  $P'$
- Part 2 places  $[\frac{q_00}{2q_7}]$  in  $P'$  as  $\delta(q_0, 0) = (q_7, 2, R)$
- Part 4 places the dominos  $[\frac{0}{0}], [\frac{1}{1}], [\frac{2}{2}], [\frac{\#}{\#}]$  in  $P'$ .

Limits of Computation – p.49/58

## Continue the example

Assume that in state  $q_7$ , upon reading 1,  $M$  goes into state  $q_5$ , writes 0, and moves head to the right, i.e.,  $\delta(q_7, 1) = (q_5, 0, R)$ , i.e.,  $[\frac{q_71}{0q_5}] \in P'$ . Hence, latest partial match extends to:



Limits of Computation – p.52/58

## Construction of $S$ : Part 5

Put  $[\frac{\#}{\#}]$  and  $[\frac{\#}{\square\#}]$  into  $P'$

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

Limits of Computation – p.51/58

## Observations

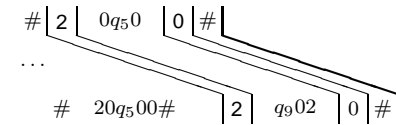
- As we construct the match we are forced to simulate  $M$  on  $w$ .
- This process continues until  $M$  reaches a halting state.
- If an accept state occurs, we want to let the top of partial match catch up with the bottom. This is done by Part 6.

## Note

If  $\delta(q_5, 0) = (q_9, 2, L)$  then we have dominos

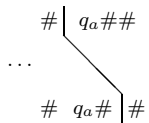
$$\left[ \frac{0q_5 0}{q_9 0 2} \right], \left[ \frac{1q_5 0}{q_9 1 2} \right], \left[ \frac{2q_5 0}{q_9 2 2} \right], \left[ \frac{\sqcup q_5 0}{q_9 \sqcup 2} \right]$$

**Note:** the first domino is relevant because symbol to the left of the head is 0 and the preceding partial match extends to:



## Construction of $S$ : Part 7

Finally, add the domino  $\left[ \frac{q_a \# \#}{\#} \right]$  and complete the match:



## Construction of $S$ : Part 6

For every  $a \in \Gamma$ , put  $\left[ \frac{a q_a}{q_a} \right], \left[ \frac{q_a a}{q_a} \right]$  into  $P'$ .

Part 6 has the effect of adding “pseudo-steps” to the TM  $M$  after it halted, where the head “eats” adjacent symbols until non are left.

## PCP over $\Sigma = \{0, 1\}$ is undecidable

**Proof idea:** by reduction from PCP.

**Reduction:**

- Construct a computable function, *Encode*, that takes a PCP instance of any finite alphabet and produces a binary alphabet PCP, called BPCP.
- Since PCP is undecidable so is BPCP

## A decidable version of PCP

**Problem:** PCP over the alphabet  $\Sigma = \{1\}$  is decidable

**Proof:** we describe a TM  $M$  that decide the unary PCP

- Consider the unary instance of PCP  $P = \{[\frac{1^{a_1}}{1^{b_1}}], \dots, [\frac{1^{a_n}}{1^{b_n}}]\}$ .
- The machine  $M$  performs as follows:  
 $M =$  "On input  $\langle a_1, b_1, \dots, a_n, b_n \rangle$ :
  1. Check if  $a_i = b_i$  for some  $i$ ; if so, *accept*
  2. Check if there exist  $i, j$  such that  $a_i > b_i$  and  $a_j < b_j$ . If so, *accept*; otherwise *reject*."