

CHAPTER 5: REDUCIBILITY.

- HOW CAN WE PROVE THAT SOME PROBLEMS ARE UNDECIDABLE?

NEW TECHNIQUE: "REDUCE" ONE PROBLEM TO ANOTHER.

- THE EQUALITY OF TWO TURING MACHINES IS UNDECIDABLE.
- LINEAR-BOUNDED AUTOMATA (LBA's)
- THE "POST CORRESPONDENCE PROBLEM" (PCP)

An engineer and a mathematician were hiking when they were suddenly attacked by a bear.

The engineer grabbed a stick and, yelling and stabbing wildly with the stick, managed to fight off the bear.

The next day, when they were hiking, they were attacked again by a bear.

The mathematician picked up a nearby stick and handed it to the engineer, thereby "REDUCING" the problem to a previously solved problem.

REDUCIBILITY

We want to "REDUCE" a hard problem into an easier problem.

A solution to the easier problem can then be used to solve the harder problem.

HARD PROBLEM: FLY FROM PORTLAND TO CAIRO.

BUT: WE KNOW THERE IS A FLIGHT FROM N.Y. TO CAIRO.

EASY PROBLEM: FLY FROM PORTLAND TO N.Y.

Now: If we can find a solution to the easier problem, we have solved the harder problem.

REVERSING THE LOGIC

REDUCE A HARD PROBLEM INTO
AN EASIER PROBLEM.

BUT WHAT IF THE HARD PROBLEM
IS KNOWN TO BE UNSOLVABLE?
THEN THE "EASIER" PROBLEM MUST
ALSO BE UNSOLVABLE!

HARD PROBLEM: LIVE FOREVER

↻ KNOWN TO BE ~~IMPOSSIBLE~~ IMPOSSIBLE!

"EASIER" PROBLEM: STOP AGING.

If we can find a solution to the
"STOP AGING" problem, then we could
solve the "LIVE FOREVER" problem.

BUT: WE KNOW "LIVING FOREVER" IS
IMPOSSIBLE. SO WE CAN CONCLUDE
THAT IT IS IMPOSSIBLE TO "STOP AGING." 4

LOGIC

KNOWN FACT: A_{TM} IS UNDECIDABLE.

What about problem X ?

Is X undecidable?

THEOREM

X is undecidable.

PROOF APPROACH.

Assume X is decidable.

Reduce A_{TM} (a "hard" problem)
into X (the "easier" problem)

Use the solution to X to
solve A_{TM} .

Use the decidability of X
to find an algorithm to
decide A_{TM} .

Build ~~a~~ a TM to decide A_{TM} ,
~~using~~ using the TM to decide
 X as a subroutine.

But we know that there cannot
exist a decider for A_{TM} !

THE "HALTING" PROBLEM

"Does a program halt when given a specific input?"

THEOREM THE LANGUAGE

$$\text{HALT}_{\text{TM}} = \left\{ \langle M, w \rangle \mid \begin{array}{l} M \text{ is a Turing Machine} \\ \text{and } M \text{ halts on} \\ \text{input } w. \end{array} \right\}$$

IS UNDECIDABLE.

PROOF

- Assume it is decidable.

THERE IS A TURING MACHINE R
THAT DECIDES HALT_{TM} .

- Use R to build another Turing Machine, S , that decides A_{TM} .

REDUCE A_{TM} TO HALT_{TM}

- But A_{TM} is undecidable.
- CONTRADICTION!

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine that } \underline{\text{accepts}} w \}$

TO DECIDE A_{TM} ...

"S" { GIVEN M AND SOME INPUT w ,
IF M ACCEPTS w
THEN \Rightarrow ACCEPT.
IF M REJECTS w OR LOOPS
THEN \Rightarrow REJECT.

DECIDERS CAN NEVER LOOP!

(NOTE: "S" IS PROVEN NOT TO EXIST!)

GIVEN

$R =$ A TURING MACHINE THAT DECIDES:

$HALT_{TM} = \{ \langle M, w \rangle \mid M \text{ is a Turing Machine that } \underline{\text{halts}} \text{ on } w \}$

TO DECIDE $HALT_{TM}$

"R" { GIVEN M AND SOME INPUT w ,
IF M ACCEPTS OR REJECTS w ...
THEN \Rightarrow ACCEPT
IF M LOOPS ...
THEN \Rightarrow REJECT

7

GOAL: CONSTRUCT S , A DECIDER FOR ATM

R = Algorithm to DECIDE $HALT_{TM}$

S = Algorithm to DECIDE ATM.

HERE IS ALGORITHM S :

INPUT: $\langle M, w \rangle$

Run R on $\langle M, w \rangle$ to see if M halts or loops.

IF R rejects, it means M loops.
THEN REJECT

IF R accepts, it means M halts.
(And R will never loop.)

Simulate/Run M on input w .
When M halts...

IF M accepts, Then \Rightarrow ACCEPT
IF M rejects, Then \Rightarrow REJECT

So S is a DECIDER for ATM.

CONTRADICTION: ATM is UNDECIDABLE.

THEOREM

DOES A T.M. MACHINE ACCEPT
ANY STRING? — UNDECIDABLE.

THE LANGUAGE

$$E_{TM} = \left\{ \langle M \rangle \mid M \text{ is a Turing Machine} \right. \\ \left. \text{and } L(M) = \emptyset \right\}$$

IS UNDECIDABLE.

PROOF:

- Assume that R decides E_{TM} .
- Use it to construct S ,
a decider for A_{TM} .
- Contradiction.

IN MORE DETAIL ...

APPROACH:

We will
REDUCE
 A_{TM} to E_{TM}

Goal:

- Construct an algorithm(S) which is given $\langle M, w \rangle$ and decides whether M accepts w .

Step 1:

- Modify M a little bit.
- Call the new machine M'

M'

(ASSUME: w is a constant
 x is the input.)

- Will reject all strings that do not match w .
- May or may not accept w .

$$L(M') = \{w\} \text{ OR } L(M') = \emptyset$$

ALGORITHM FOR M'

INPUT: x

IF $x \neq w$ THEN REJECT.

OTHERWISE

SIMULATE M ON x .

IF M ACCEPTS, THEN ACCEPT.

IF M REJECTS, THEN REJECT.

ALGORITHM FOR S : (TO DECIDE ATM)

INPUT: M, w

STEP 1: Construct M' .

Take M .

Add a test ($x=w$) in front
of the initial state

Then pass control to M .

NOTE:

$L(M') = \{w\}$ if M accepts w

$L(M') = \emptyset$ otherwise.

STEP 2:

Use R to decide whether
 $L(M')$ is empty or not.

R accepts $\Rightarrow L(M')$ is empty $\Rightarrow M$ does
not accept w

R rejects $\Rightarrow L(M') = \{w\} \Rightarrow M$ accepts w .

STEP 3: We now have decided ATM .

CONTRADICTION! (E_{TM} is UNDECIDABLE) ||

DEFINITION

A "COMPUTABLE" FUNCTION:

THERE IS A TURING MACHINE
WHICH COMPUTES IT.

$$f: \Sigma^* \rightarrow \Sigma^*$$

- THE INPUT x IS ON THE TAPE.
- THE T.M. RUNS AND ALWAYS
HALTS!
- THE RESULT $f(x)$ IS LEFT
ON THE TAPE.

THEOREM

THE LANGUAGE

$$EQ_{TM} = \left\{ \langle M_1, M_2 \rangle \mid \begin{array}{l} M_1 \text{ and } M_2 \text{ are Turing} \\ \text{Machines and} \\ L(M_1) = L(M_2) \end{array} \right\}$$

IS UNDECIDABLE.

IN PLAIN ENGLISH?

"Given two programs, you can not compare them to see whether they do the same thing."

"Any correct algorithm to compare the functionality of two programs will sometimes fail to halt."

PROVING PROGRAM CORRECTNESS

"INFORMAL IDEA OF WHAT
YOU WANT THE PROGRAM
TO DO."



Subject to human
error; artform?

"FORMAL SPECIFICATION"

$\forall x. (x \text{ is a mouse click}) \Rightarrow \dots$

"SPECIFICATION LANGUAGE"



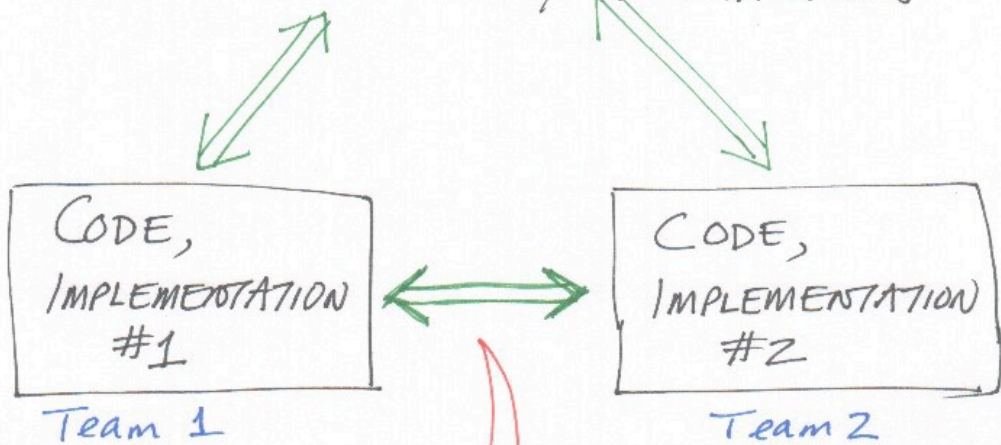
UNDECIDABLE IN GENERAL,
But that does not mean
you can't find a proof
in some cases.

"CODE"

PROGRAM/ALGORITHM EXPRESSED
IN SOME VERIFIABLE
PROGRAMMING LANGUAGE.

ANOTHER APPROACH

"INFORMAL IDEA OF WHAT YOU WANT THE PROGRAM TO DO." / "SPECIFICATION"



IDEA:

Try to prove these algorithms are equivalent.

(Must use human guidance to direct search.)

Problems in finding a proof?

May indicate ambiguities, or lack of detail in the specification.

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid L(M_1) = L(M_2) \}$$

PROOF APPROACH

REDUCE E_{TM} TO EQ_{TM} .

WE ALREADY KNOW E_{TM} IS UNDECIDABLE.

$$A_{TM} \leq_m E_{TM} \leq_m EQ_{TM}$$

↙ Symbol for "Mapping Reducible"

It means we can perform the transformation of one problem into another

With ~~an algorithm~~^a COMPUTABLE ALGORITHM.

(A Turing Machine that always halts)

GIVEN AN ALGORITHM TO SOLVE EQTM,
HOW DO WE PROCEED?

R = DECIDER FOR EQTM

S = DECIDER FOR ETM

INPUT TO S: M

S will decide
whether $L(M) = \emptyset$

• LET M_\emptyset BE A TURING
MACHINE THAT ALWAYS
REJECTS

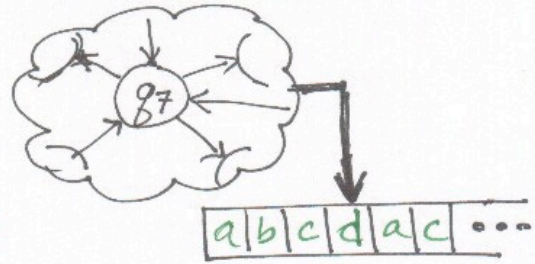
REJECT

• Write M_\emptyset on the tape
after M.

• CALL **R** to decide whether
 $L(M) = L(M_\emptyset)$

"CONFIGURATION"

C: abcq₇dac



A SEQUENCE OF CONFIGURATIONS:

... $\Rightarrow C_4 \Rightarrow C_5 \Rightarrow C_6 \Rightarrow C_7 \Rightarrow \dots$

└─ step in the computation.

"ACCEPTING COMPUTATION HISTORY"

$C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow C_4 \Rightarrow \dots \Rightarrow C_L$

Initial Starting Configuration

Legal steps for this machine

An Accepting State

"REJECTING COMPUTATION HISTORY"

$C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow \dots \Rightarrow C_{\#L}$

Same, except the last configuration is a Rejecting State

A COMPUTATION HISTORY

IS A FINITE SEQUENCE.

IF MACHINE DOES NOT HALT?

⇒ NO HISTORY.

DETERMINISTIC MACHINES:

AT MOST, ONE HISTORY
(on a given input)

NON-DETERMINIST MACHINES:

MAY HAVE MANY HISTORIES.

FROM NOW ON...

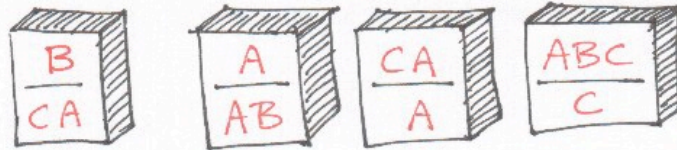
ASSUME OUR TM IS DETERMINISTIC.

THERE IS EITHER

- AN ACCEPTING HISTORY
- A REJECTING HISTORY
- NO HISTORY / MACHINE LOOPS.

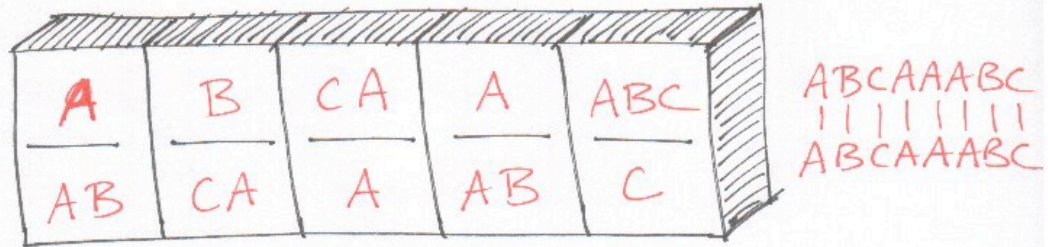
PCP: POST CORRESPONDENCE PROBLEM

DOMINOS:



WE GET AS MANY OF EACH TYPE AS WE NEED.

GOAL: FIND A ^{finite} SEQUENCE OF DOMINOS SUCH THAT THE TOP AND BOTTOM STRINGS ARE THE SAME.



Does a solution exist?
This problem is unsolvable!
(i.e., UNDECIDABLE)

ANOTHER PCP INSTANCE.

	A	B	
①	1	111	
②	10111	10	② ① ① ③
③	10	0	10111.1.1.1.0 10.111.111.0

A NON-SOLUTION: ① ① ②

A: 1.1.10111.

B: 111.111.10.

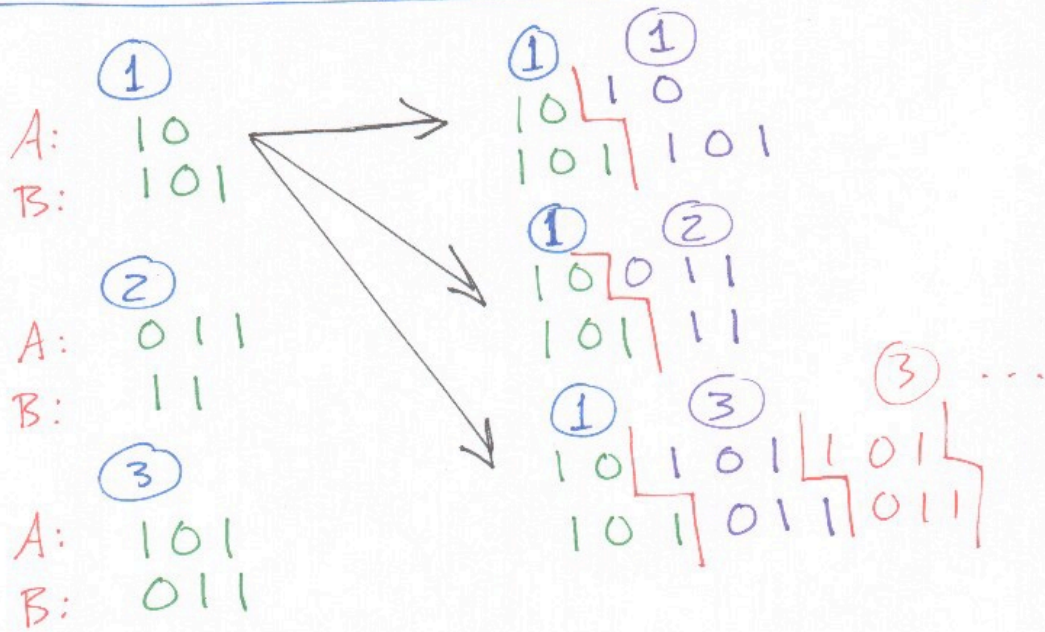
A SOLUTION: ② ① ① ③

A: 10111.1.1.1.0

B: 10.111.111.0

ANOTHER PCP INSTANCE

	A	B
①	10	101
②	011	11
③	101	011



THEOREM

The POST CORRESPONDENCE PROBLEM
IS UNDECIDABLE

PROOF (OVERVIEW)

Reduce A_{TM} to a PCP.

If A_{TM} accepts, then

M computes on w and accepts.

Then, there is a finite

COMPUTATION HISTORY which
describes the computation
of M on w .

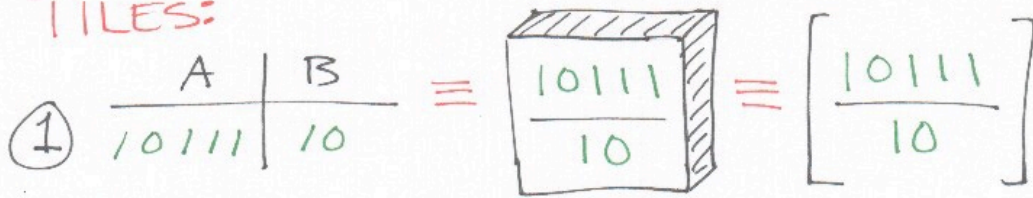
ENCODE $\langle M, w \rangle$ INTO A PCP INSTANCE.

THERE IS A SOLUTION TO THE PCP
IFF THERE IS AN ACCEPTING
COMPUTATION HISTORY.

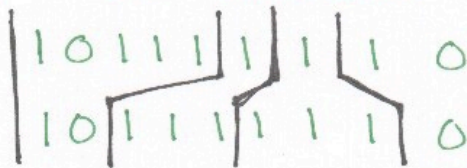
IF WE COULD DECIDE THIS INSTANCE
OF THE PCP, THEN WE
COULD DECIDE A_{TM} .

BUT WE CAN'T DECIDE A_{TM} !

TILES:



PCP SOLUTIONS:



TURING MACHINE CONFIGURATIONS:

101q₄011

COMPUTATION HISTORY:

q₀101# 1q₄01# ... # 011q_A101#

THIS IS AN ACCEPTING HISTORY

INITIAL STATE
AT LEFT END
OF TAPE

q_A accept

GIVEN INPUT $\langle M, w \rangle$ TO A_{TM} ,

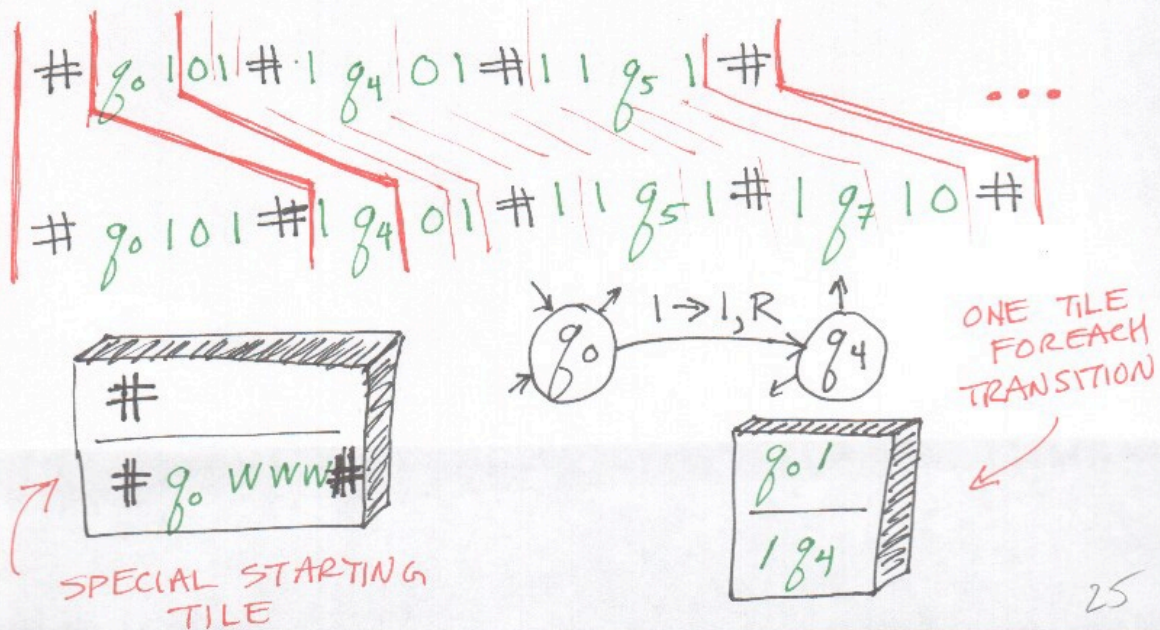
CONSTRUCT AN INSTANCE OF
THE PCP...

(CREATE A COLLECTION OF TILES)

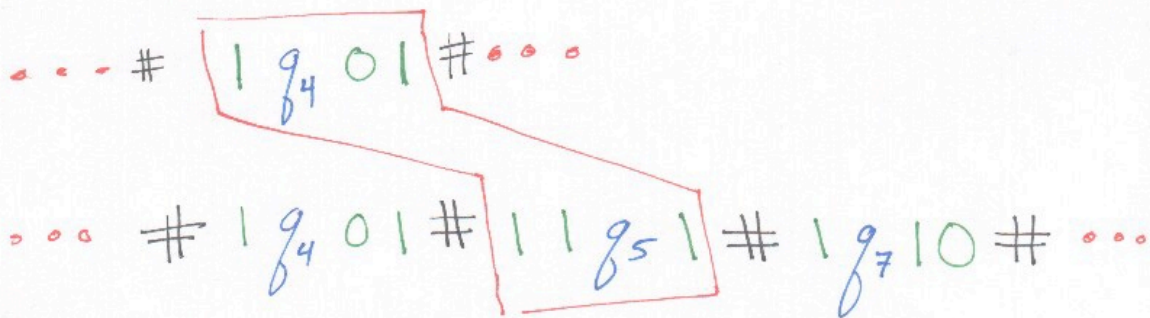
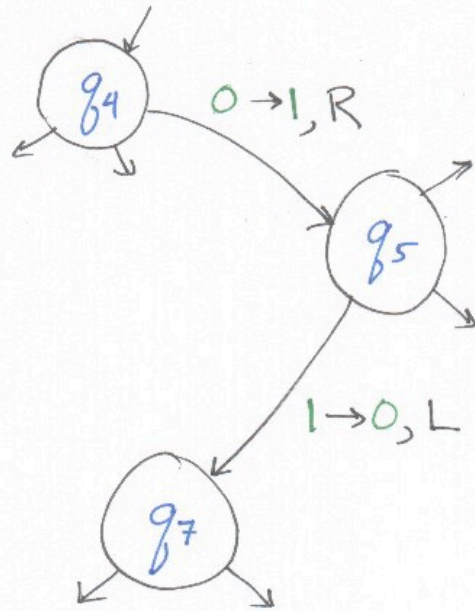
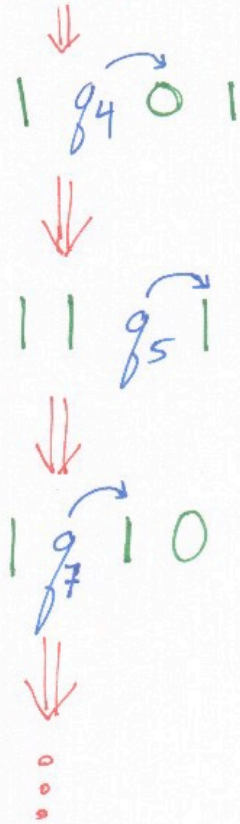
SUCH THAT, IF YOU CAN FIND

A SOLUTION TO THIS PCP INSTANCE,
THEN YOU'VE FOUND AN ACCEPTING
COMPUTATION HISTORY.

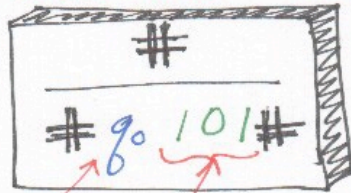
A SOLUTION WILL LOOK LIKE THIS:



COMPUTATION HISTORY

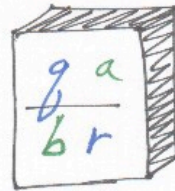
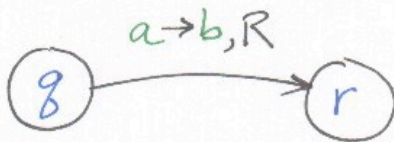


SPECIAL STARTING TILE:

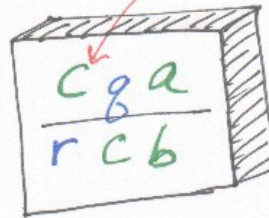
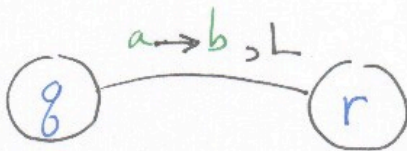


INITIAL STATE

RIGHT MOVES:

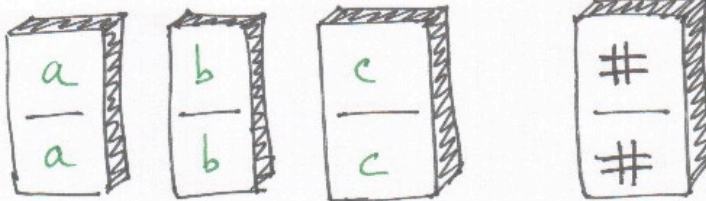


LEFT MOVES

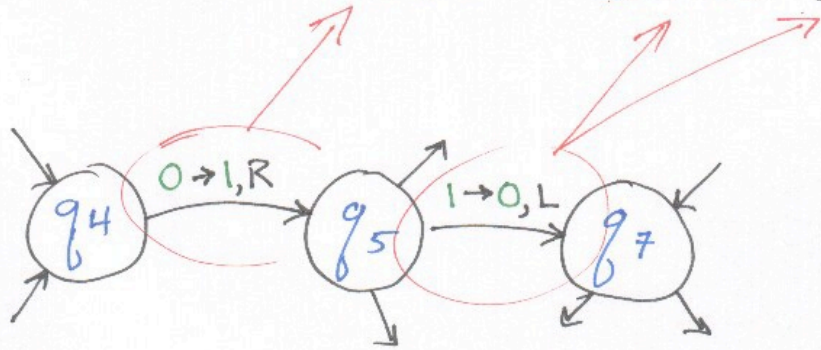
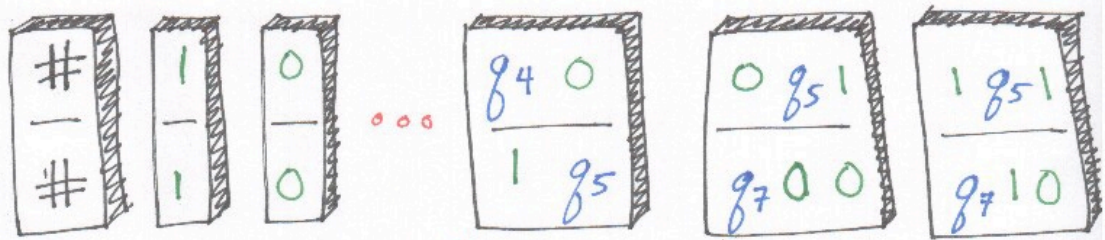
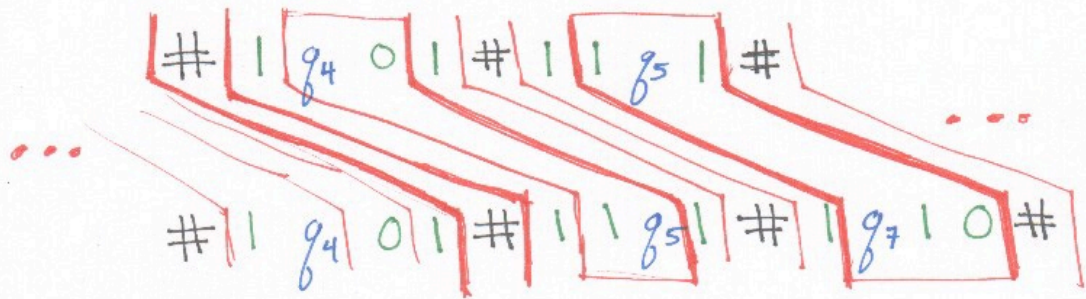


Add one tile like this for every c in I

TILES TO "COPY" THE TAPE:



For every symbol in I



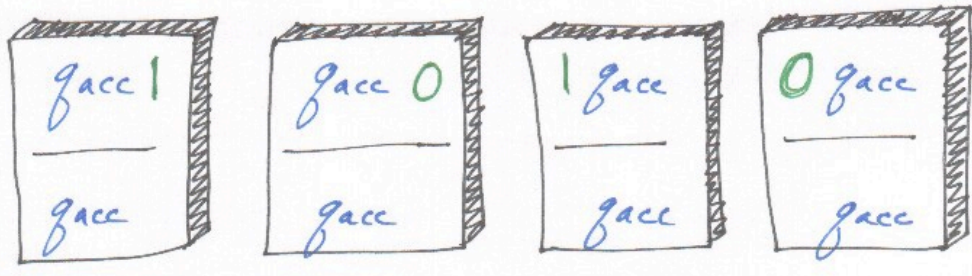
Q: HOW DO WE ACCEPT?

A: COMPLETE THE MATCH!

ADD SPECIAL TILES TO ALLOW q_{accept}
TO "EAT" THE SYMBOLS ON
THE TAPE:

1 0 | q_{accept} | |

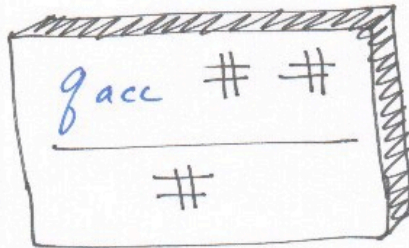
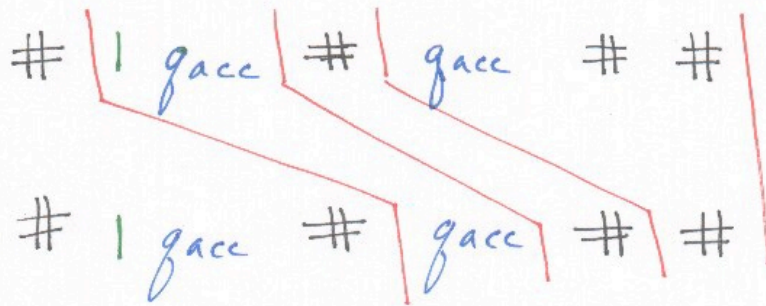
1 0 q_{accept} | | # 1 0 | q_{acc} |



↖ For every symbol in Γ

EVENTUALLY, NOTHING REMAINS
EXCEPT *gacc*.

ADD A TILE TO FINISH THE MATCH.

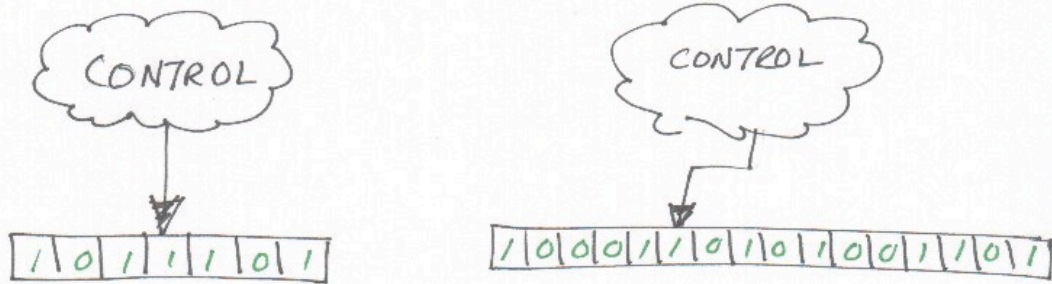


REVIEW OF THE PROOF.

- IF YOU CAN FIND A SOLUTION TO THIS INSTANCE OF THE PCP, THEN YOU HAVE FOUND A LEGAL ACCEPTING COMPUTATION HISTORY, IN WHICH MACHINE M ACCEPTS STRING w .
- DOES A SOLUTION EXIST?
- IF YOU CAN DECIDE THE ANSWER, THEN YOU CAN DECIDE WHETHER M ACCEPTS w .
- WE KNOW THAT A_{TM} IS UNDECIDABLE.
- THEREFORE, WE HAVE PROVEN THAT THE PROBLEM OF FINDING A SOLUTION TO THE PCP IS UNDECIDABLE!
(IN GENERAL).

LINEAR BOUNDED AUTOMATON (LBA)

A RESTRICTED FORM OF TURING MACHINE.



THE TAPE IS LIMITED IN SIZE TO THE SIZE OF THE INPUT.

(i.e., HEAD NOT ALLOWED TO MOVE BEYOND THE INPUT.)

LBA'S ARE NOT AS POWERFUL AS FULL TURING MACHINES.

BUT ARE REALLY QUITE POWERFUL.

ASSUME OUR TAPE ALPHABET CAN BE LARGER THAN INPUT ALPHABET.

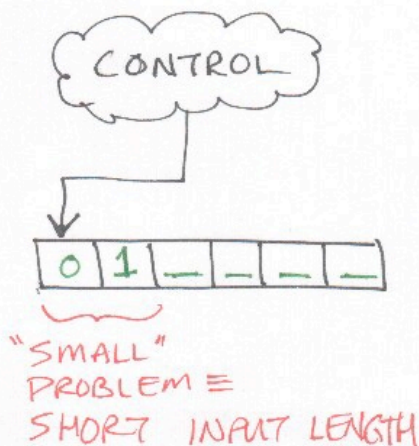
WE CAN USE A LARGER TAPE ALPHABET TO ~~TO~~ STORE MORE INFORMATION IN OUR LIMITED TAPE.

OR EQUIVALENTLY...

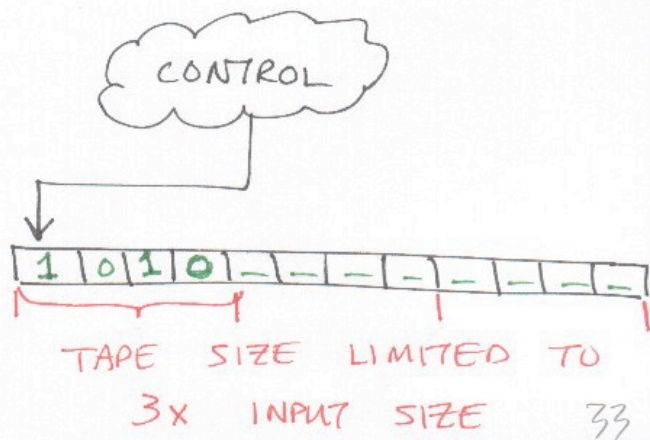
WE CAN RESTRICT OUR MACHINE TO USING ONLY A ~~SMALL~~ SMALL PORTION OF THE TAPE.

"A LINEAR FUNCTION OF THE INPUT SIZE"

"SMALL" PROBLEM



"LARGE" PROBLEM



THEOREM

THE LANGUAGE

$$A_{\text{LBA}} = \left\{ \langle M, w \rangle \mid \begin{array}{l} M \text{ is an LBA} \\ \text{and } M \text{ accepts } w \end{array} \right\}$$

IS DECIDABLE !!

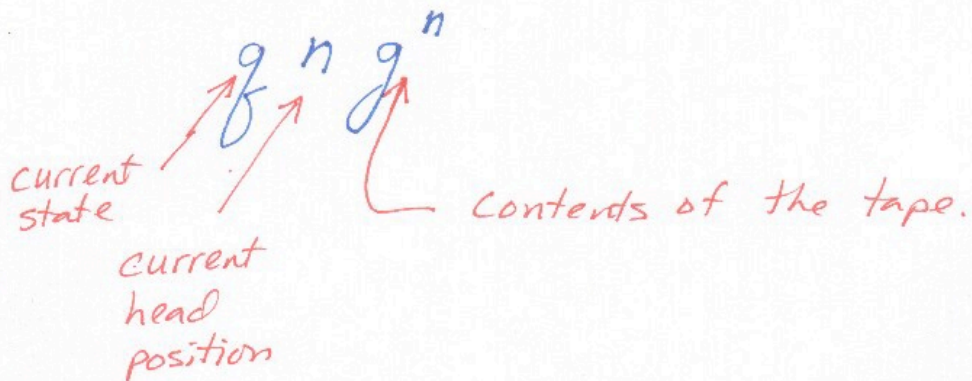
CONSIDER SOME LBA...

q = Number of states

σ = Size of tape alphabet

n = Length of tape.

How many distinct configurations are there?



This may be a big number
but it is finite.

A_{LBA} IS DECIDABLE.

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

Just simulate M on w .

M could accept, reject, or loop.

How can we detect looping?

If ~~the~~ M ever enters a configuration it has already been in, then it must loop forever.

There are only finitely many possible configurations.

If ~~the~~ simulation goes that long, it must be looping.

So just run the simulation g^{n^2} steps.

If ~~it~~ it has not halted by then, THEN reject.

LINEAR BOUNDED AUTOMATA (LBAs)
ARE VERY POWERFUL.

NOT FULL TURING MACHINE POWER, BUT...

WHAT PROBLEMS CAN THEY SOLVE?

A_{DFA}

A_{CFG}

E_{DFA}

E_{CFG}

DECIDABLE BY
AN LBA.

REDUCING ONE PROBLEM INTO ANOTHER

DEFN

REDUCING LANGUAGES.

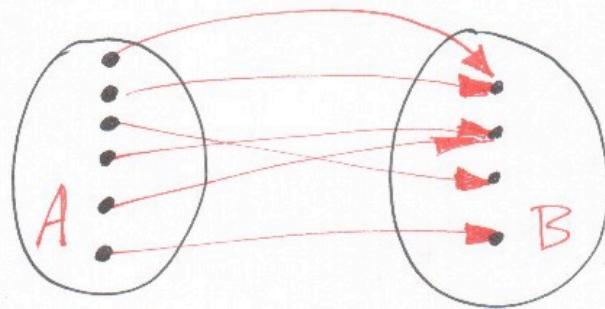
Language A "REDUCES" to language B...

$$A \leq_m B$$

if there exists a computable function $f: \Sigma^* \rightarrow \Sigma^*$

such that, for every x

$$x \in A \text{ iff } f(x) \in B$$



MORE ON REDUCIBILITY

If A reduces to B and
 B is decidable,
THEN A is decidable.

If A reduces to B and
 A is undecidable,
THEN B is undecidable.

If A reduces to B and
 B is Turing Recognizable,
THEN A is Turing Recognizable.

If A reduces to B and
 A is not Turing Recognizable,
THEN B is not Turing Recognizable.
