REDUCIBILITY:
A TECHNIQUE FOR PROVING UNDECIDABILITY

REDUCIBILITY

How can we prove that some problems are undecidable?

NEW TECHNIQUE:

"REDUCE" ONE PROBLEM TO ANOTHER PROBLEM.

RESULTS:

THE EQUIVALENCE OF TWO TURING MACHINE? ... UNDECIDABLE!

IS A GIVEN PROGRAM/ALGORITHM GUARANTEED TO HALT? ... UNDECIDABLE!

WILL A GIVEN TURING MACHINE ACCEPT ANY STRING? ... UNDECIDABLE!

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.

<u>HARD PROBLEM:</u>

Fly from Portland to Cairo.

We know there are direct flights from Portland to New York.

So we have "REDUCED" the problem to an easier problem.

<u>EASIER PROBLEM:</u>

Fly from New York to Cairo.

<u>So...</u> If we can find a solution to the easier problem, we can use it to solve the harder problem!

REVERSING THE LOGIC

REDUCE A <u>HARD</u> PROBLEM INTO AN <u>EASIER</u> PROBLEM.

BUT WHAT IF THE HARD PROBLEM IS KNOWN TO BE UNSOLVABLE?

THEN THE "EASIER" PROBLEM MUST ALSO BE UNSOLVABLE!

<u>HARD PROBLEM:</u> LIVE FOREVER

KNOWN TO BE IMPOSSIBLE!

<u>"EASIER" PROBLEM:</u> STOP AGING.

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

<u>BUT</u>: WE KNOW "LIVING FOREVER" IS IMPOSSIBLE. <u>SO</u> WE CAN CONCLUDE THAT IT IS IMPOSSIBLE TO "STOP AGING."

LOGIC

KNOWN FACT: A_{TM} IS UNDECIDABLE.

WHAT ABOUT SOME OTHER PROBLEM **P**...
IS **P** UNDECIDABLE?

THEOREM

P IS UNDECIDABLE.

PROOF APPROACH

- Assume **P** IS DECIDABLE.
- Reduce A_{TM} (a "HARD" problem) into **P** (the "EASIER" problem)
- Use the solution of **P** to solve A_{TM}.

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

  Build a TM to decide A_{TM} using the TM to decide **P** as a subroutine.
- But we know that a decider for A_{TM} cannot exist.

∴ Contradiction: **P** is not decidable!

THE HALTING PROBLEM

PROOF OF ITS UNDECIDABILITY

THE "HALTING" PROBLEM

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

THEOREM THE LANGUAGE

$$HALT_{TM} = \left\{ \langle M, w \rangle \;\middle|\; \begin{array}{l} M \text{ is a Turing Machine} \\ \text{and } M \text{ halts on} \\ \text{input } w. \end{array} \right\}$$

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!

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

To DECIDE A_{TM} ...

"S"
- GIVEN M AND SOME INPUT w,
- IF M ACCEPTS w
  - THEN ⇒ ACCEPT.
- IF M REJECTS w OR LOOPS
  - THEN ⇒ REJECT.

DECIDERS CAN NEVER LOOP!

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

GIVEN

R = A TURING MACHINE THAT DECIDES:

$HALT_{TM} = \{\langle M, w\rangle \mid M$ is a TURING MACHINE that **halts** on $w\}$

To DECIDE $HALT_{TM}$

"R"
- GIVEN M AND SOME INPUT w,
- IF M ACCEPTS OR REJECTS w...
  - THEN ⇒ ACCEPT
- IF M LOOPS...
  - THEN ⇒ REJECT

GOAL: CONSTRUCT S, A DECIDER FOR A_{TM}

R = Algorithm to DECIDE $HALT_{TM}$

S = Algorithm to DECIDE A_{TM}.

HERE IS ALGORITHM **S**:

<u>INPUT</u>: $\langle M, w \rangle$

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

<u>If</u> **R** rejects, it means M loops.

<u>THEN</u> REJECT

<u>If</u> **R** accepts, it means M halts.

(And **R** will never loop.)

Simulate/Run M on input w.

When M halts...

<u>If</u> M accepts, <u>Then</u> ⟹ ACCEPT

<u>If</u> M rejects, <u>Then</u> ⟹ REJECT

So S is a DECIDER for A_{TM}.

CONTRADICTION: A_{TM} IS UNDECIDABLE.

DOES A TURING MACHINE ACCEPT ANY STRINGS?

THEOREM

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

THE LANGUAGE

$$E_{TM} = \left\{ \langle M \rangle \;\middle|\; \begin{array}{l} M \text{ is a Turing Machine} \\ \text{and } L(M) = \emptyset \end{array} \right\}$$

IS UNDECIDABLE.

PROOF:

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

APPROACH: We will **REDUCE** A_{TM} to E_{TM}

IN MORE DETAIL...

Goal:

- Construct an algorithm (S) which is given $\langle M, w \rangle$ an 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 A_{TM})

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

COMPUTABLE FUNCTIONS

DEFINITION

A "**COMPUTABLE**" FUNCTION:

A FUNCTION

$$f: \Sigma^* \to \Sigma^*$$

THAT CAN BE COMPUTED BY A TURING MACHINE.

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

EQUIVALENCE OF TWO TURING MACHINES

THEOREM

THE LANGUAGE

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

IS UNDECIDABLE.

IN PLAIN ENGLISH?

"Given two programs, you can <u>not</u> 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."

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

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 ~~[illegible]~~ a COMPUTABLE ALGORITHM.

(A Turing Machine that <u>always</u> halts)

GIVEN AN ALGORITHM TO SOLVE EQ_{TM},

HOW DO WE PROCEED?

R = DECIDER FOR EQ_{TM}

S = DECIDER FOR E_{TM}

<u>INPUT TO S:</u> M ← S will decide whether $L(M) = \emptyset$

- LET $M_{\emptyset}$ BE A TURING MACHINE THAT ALWAYS REJECTS

- Write $M_{\emptyset}$ on the tape after M.
- CALL **R** to decide whether

$$L(M) = L(M_{\emptyset})$$

PROVING PROGRAM CORRECTNESS

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

Subject to human error; artform?

"FORMAL SPECIFICATION"

$\forall x.$ (x is a mouse click) $\Rightarrow \ldots$

"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"

CODE, IMPLEMENTATION #1

Team 1

CODE, IMPLEMENTATION #2

Team 2

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.

REDUCING ONE LANGUAGE TO ANOTHER.

REDUCING ONE PROBLEM INTO ANOTHER

DEFN

REDUCING LANGUAGES.

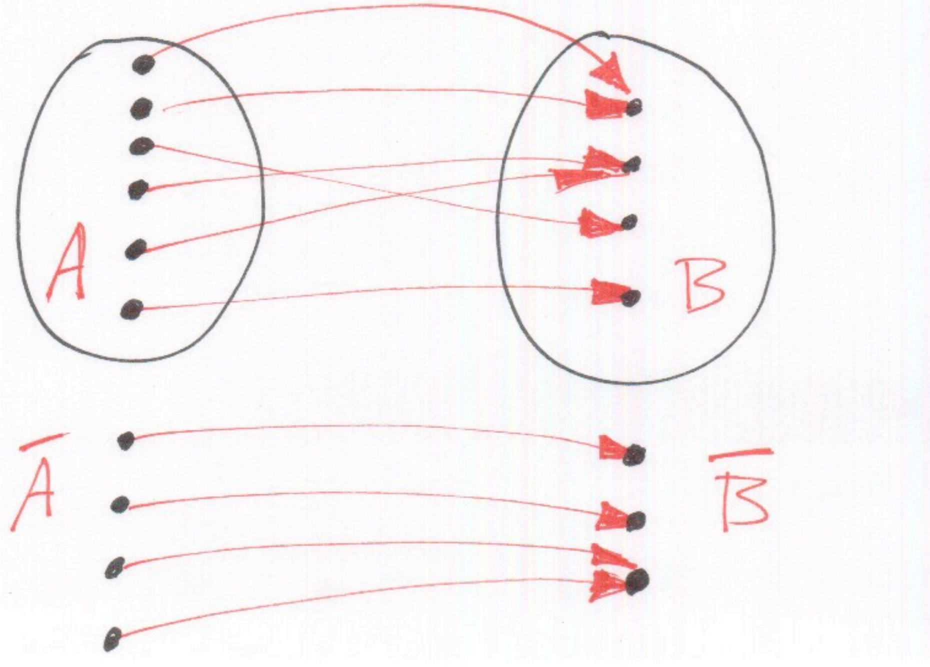
Language A "REDUCES" to language B ...

$$A \leq_m B$$

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

such that, for every x

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



MORE ON REDUCIBILITY

<u>If</u> A reduces to B <u>and</u>
B is decidlable,
<u>THEN</u> A is decidable.

<u>If</u> A reduces to B <u>and</u>
A is undecidable,
<u>THEN</u> B is undecidable.

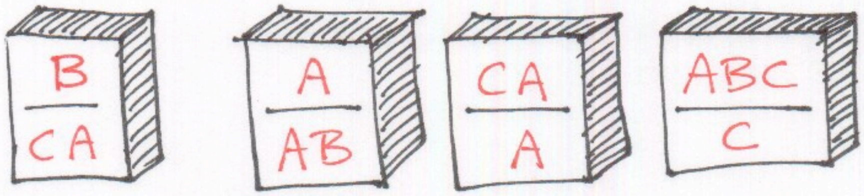
<u>If</u> A reduces to B <u>and</u>
B is Turing Recognizable,
<u>THEN</u> A is Turing Recognizable.

<u>If</u> A reduces to B <u>and</u>
A is not Turing Recognizable,
<u>THEN</u> B is not Turing Recognizable.

THE POST CORRESPONDENCE PROBLEM

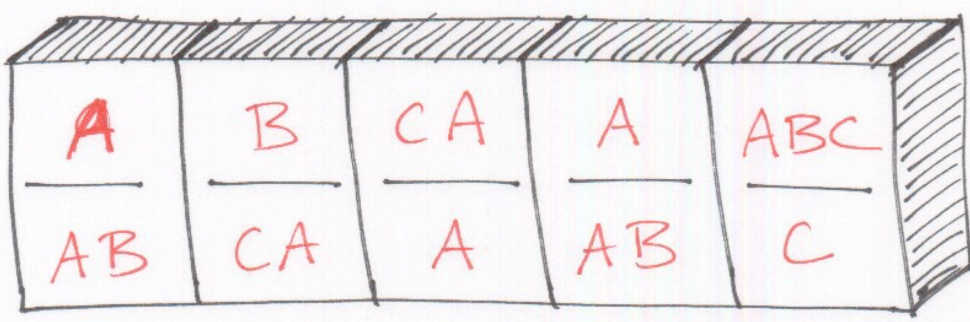
PCP: POST CORRESPONDENCE PROBLEM

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.



ABCAAABC

ABCAAABC

Does a solution exist?

This problem is unsolvable!

(i.e., UNDECIDABLE)

AN "INSTANCE" OF THE PCP PROBLEM

	A	B
1	1	111
2	10111	10
3	10	0

A NON-SOLUTION: 1 1 2

A: 1 1 1 0 1 1 1

B: 1 1 1 1 1 1 1 0

A SOLUTION: 2 1 1 3

A: 1 0 1 1 1 . 1 . 1 . 1 0

B: 1 0 . 1 1 1 . 1 1 1 . 0

Another PCP Instance

	A	B
1	10	101
2	011	11
3	101	011

1

A: 10

B: 101

2

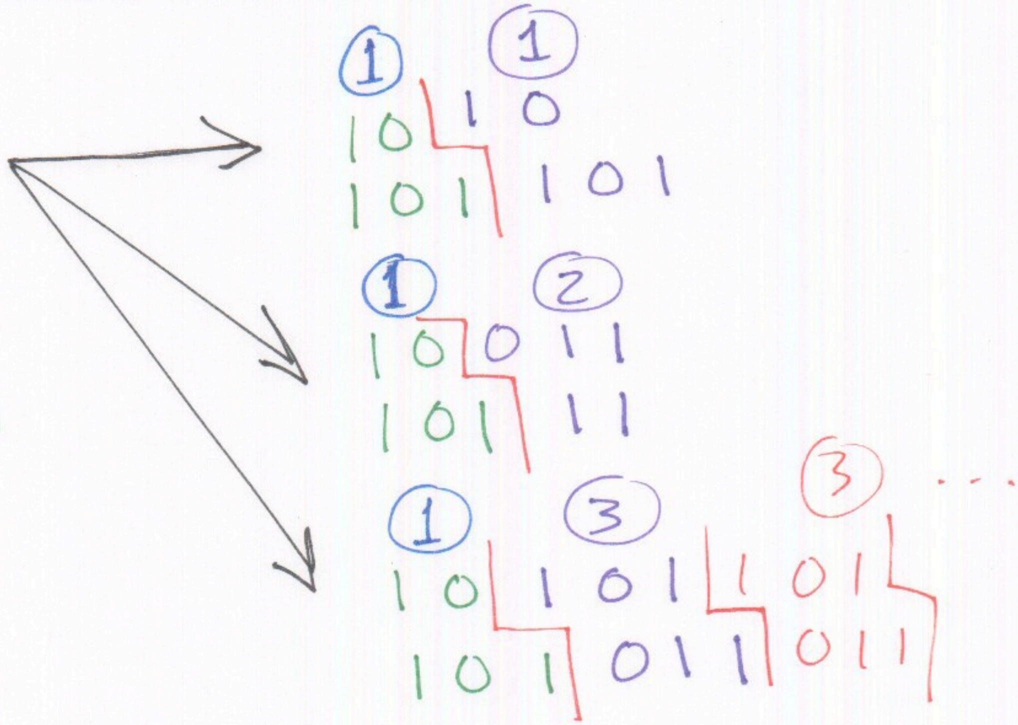
A: 011

B: 11

3

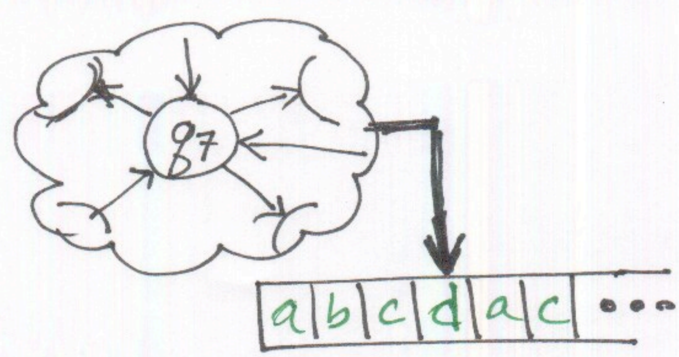
A: 101

B: 011



PROOF THAT THE POST CORRESPONDENCE PROBLEM IS UNDECIDABLE

"CONFIGURATION"



C: $abc\,q_7\,dac$

A SEQUENCE OF CONFIGURATIONS:

$\cdots \Rightarrow C_4 \Rightarrow \underbrace{C_5 \Rightarrow C_6}_{\text{step in the computation.}} \Rightarrow C_7 \Rightarrow \cdots$

AN "ACCEPTING COMPUTATION HISTORY"

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

Initial Starting Configuration

Legal steps for this machine

An Accepting State

A "REJECTING COMPUTATION HISTORY"

$C_1 \Rightarrow C_2 \Rightarrow C_3 \Rightarrow \cdots \Rightarrow C_L$

Same, except the last configuration is a Rejecting State

A COMPUTATION HISTORY

IS A <u>FINITE</u> 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.

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:

$$① \quad \begin{array}{c|c} A & B \\ \hline 10111 & 10 \end{array} \equiv \boxed{\frac{10111}{10}} \equiv \left[\frac{10111}{10}\right]$$

PCP SOLUTIONS:

1 0 1 1 1 1 1 1 0

1 0 1 1 1 1 1 1 1 0

TURING MACHINE CONFIGURATIONS:

$101q_4011$

COMPUTATION HISTORY:

$\# q_0 101 \# 1q_4 01 \# \cdots \# 011 q_A 101 \#$

THIS IS AN ACCEPTING HISTORY

INITIAL STATE AT LEFT END OF TAPE

$q_A =$

$q_{accept} =$

q_{acc}

eg. $w = 101$

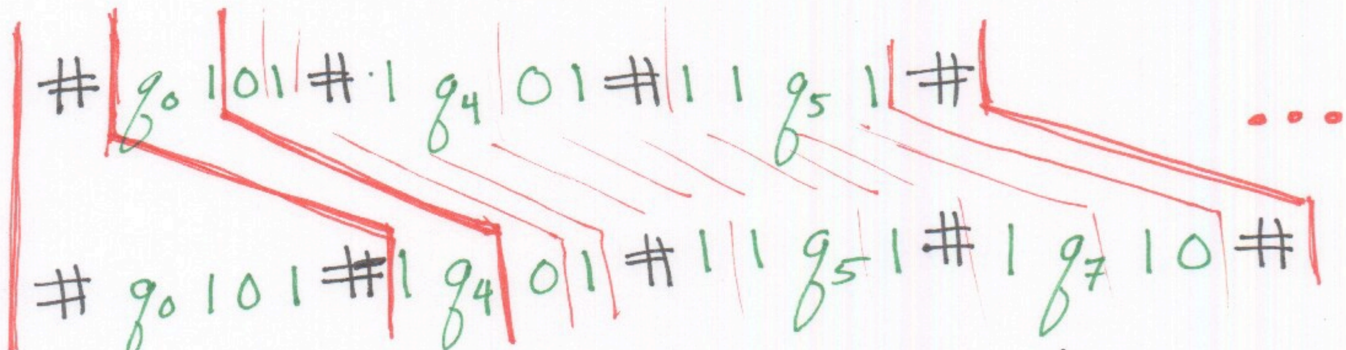
GIVEN INPUT $\langle M, w \rangle$ TO A_{TM},

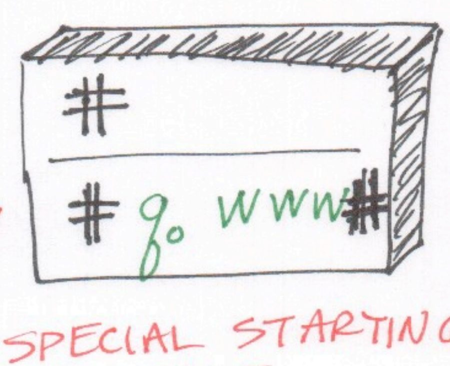
CONSTRUCT AN INSTANCE OF THE PCP...

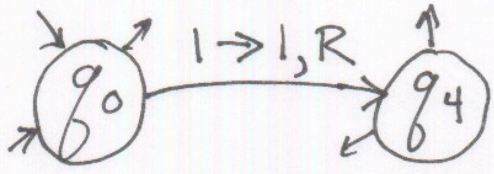
(CREATE A COLLECTION OF TILES)

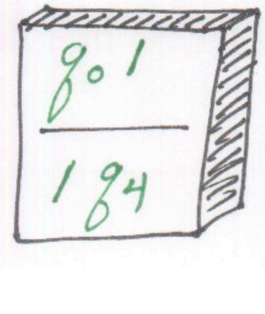
SUCH THAT, IF YOU CAN FIND A SOLUTION TO THIS PCP INSTANCE, THEN YOU'VE FOUND AN ACCEPTING COMPUTATION HISTORY.

A SOLUTION WILL LOOK LIKE THIS:









ONE TILE FOREACH TRANSITION

SPECIAL STARTING TILE

COMPUTATION HISTORY

$\Downarrow$

$1 \; q_4 \; 0 \; 1$

$\Downarrow$

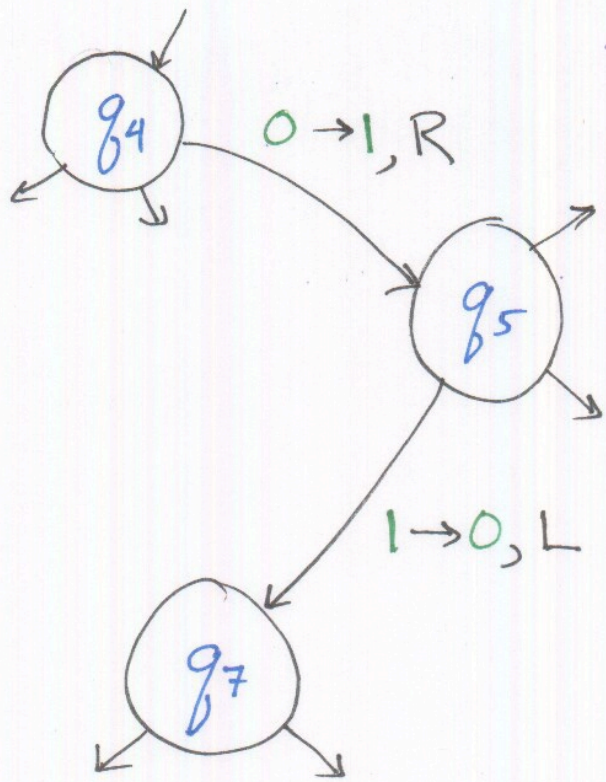
$1 \; 1 \; q_5 \; 1$

$\Downarrow$

$1 \; q_7 \; 1 \; 0$

$\Downarrow$

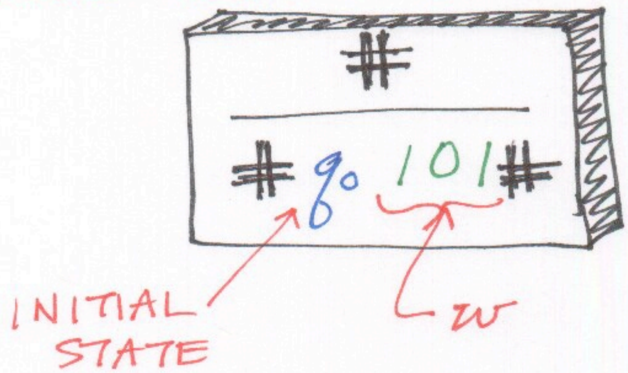
$\vdots$



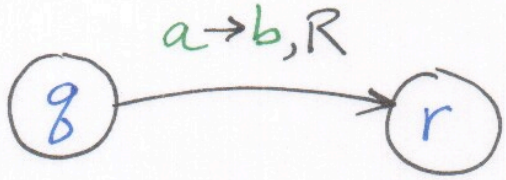
$\cdots \; \# \; 1 \; q_4 \; 0 \; 1 \; \# \; \cdots$

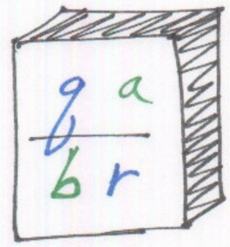
$\cdots \; \# \; 1 \; q_4 \; 0 \; 1 \; \# \; 1 \; 1 \; q_5 \; 1 \; \# \; 1 \; q_7 \; 1 \; 0 \; \# \; \cdots$

SPECIAL STARTING TILE:

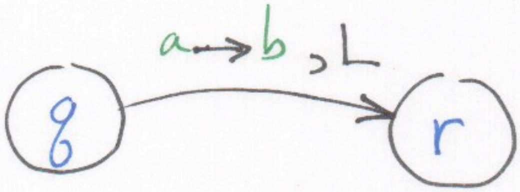


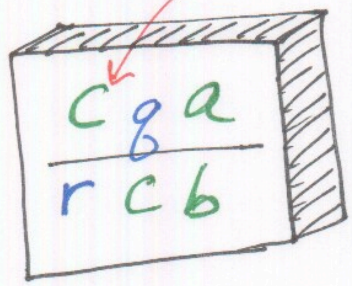
RIGHT MOVES:





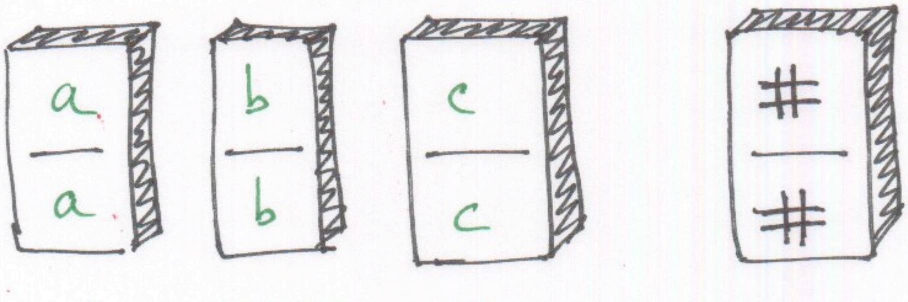
LEFT MOVES



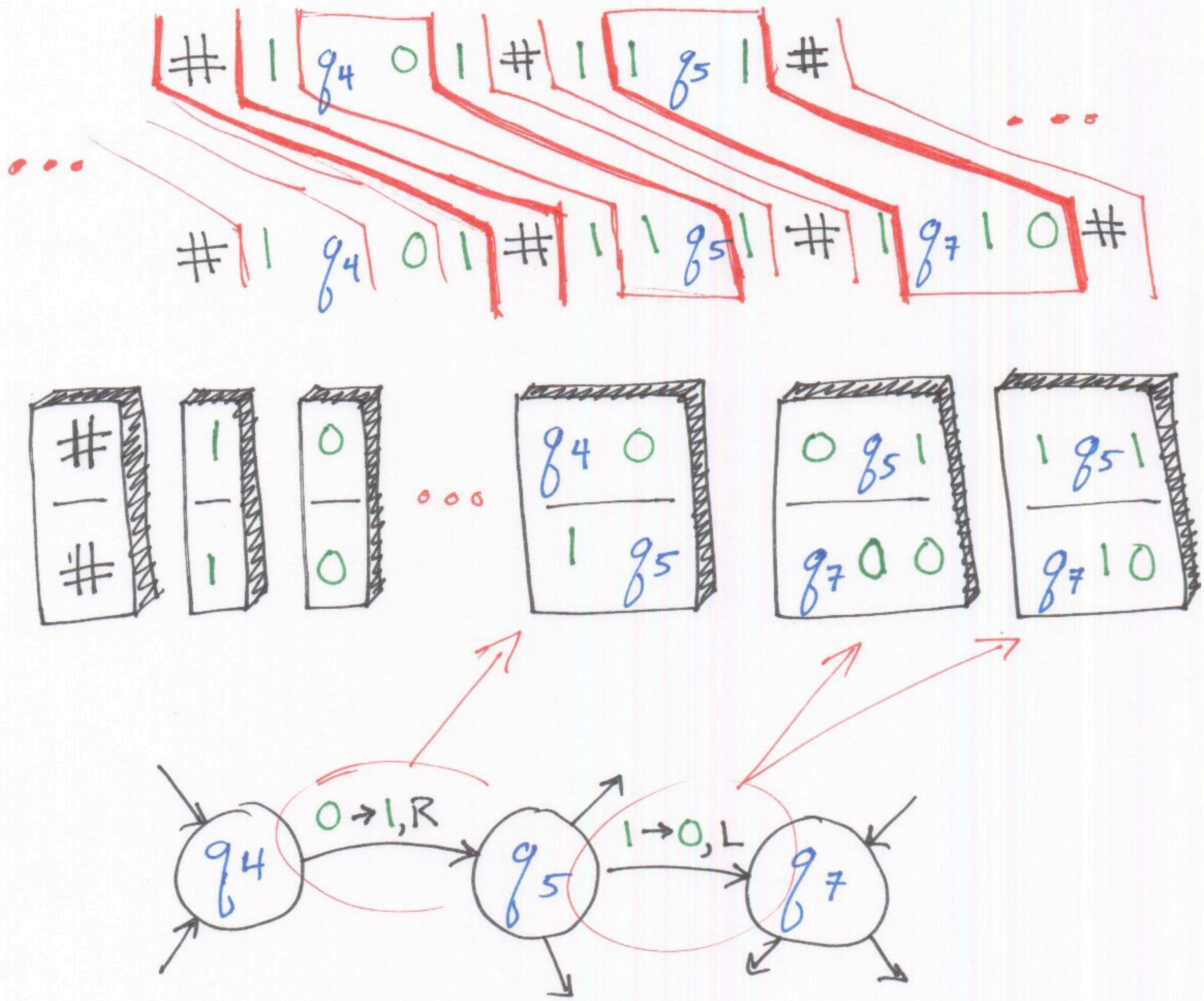


Add one tile like this for <u>every</u> $c \in \Gamma$

TILES TO "COPY" THE TAPE:



For every symbol in Γ


1 q4 0 1 # 1 1 q5 1
1 q4 0 1 # 1 1 q5 1 # 1 q7 1 0
/
1 / 1
0 / 0
q4 0 / 1 q5
0 q5 1 / q7 0 0
1 q5 1 / q7 1 0
0→1,R
1→0,L
q4
q5
q7

Q: HOW DO WE ACCEPT?

A: COMPLETE THE MATCH!

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

$\# \; 1 \; 0 \; q_{accept} \; 1 \; 1 \; \#$

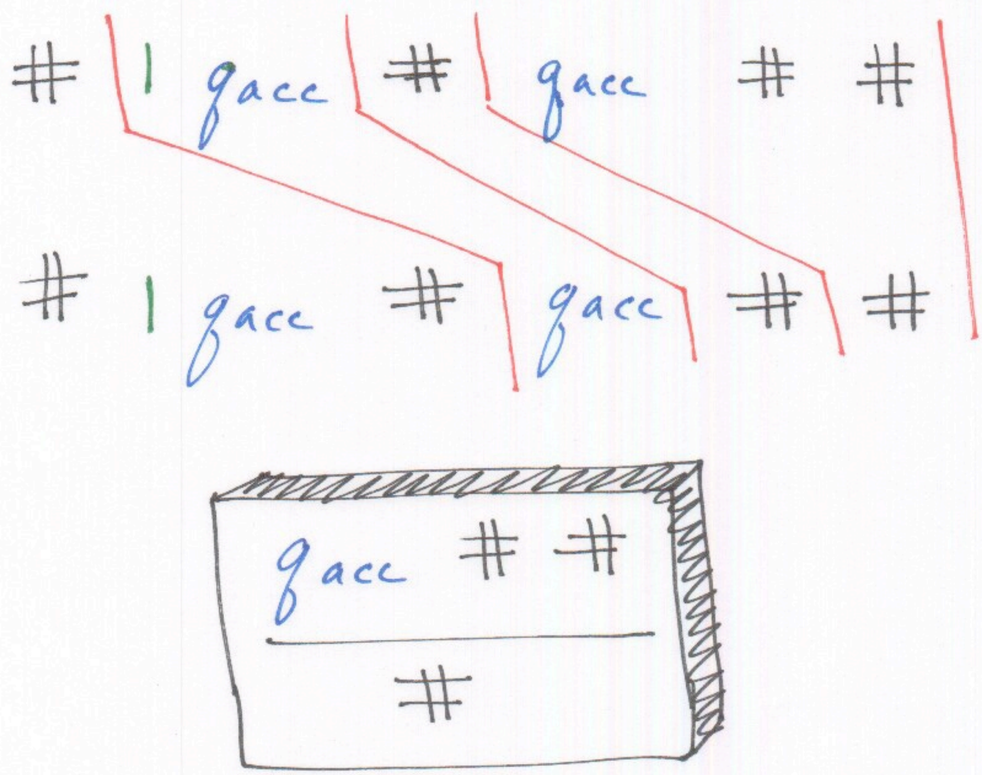
$\# \; 1 \; 0 \; q_{accept} \; 1 \; 1 \; \# \; 1 \; 0 \; q_{acc} \; 1 \; \#$

$\left[\dfrac{q_{acc}\,1}{q_{acc}}\right] \quad \left[\dfrac{q_{acc}\,0}{q_{acc}}\right] \quad \left[\dfrac{1\,q_{acc}}{q_{acc}}\right] \quad \left[\dfrac{0\,q_{acc}}{q_{acc}}\right]$

For every symbol in Γ

EVENTUALLY, NOTHING REMAINS EXCEPT gacc.

ADD A TILE TO FINISH THE MATCH.



REVIEW OF THE PROOF.

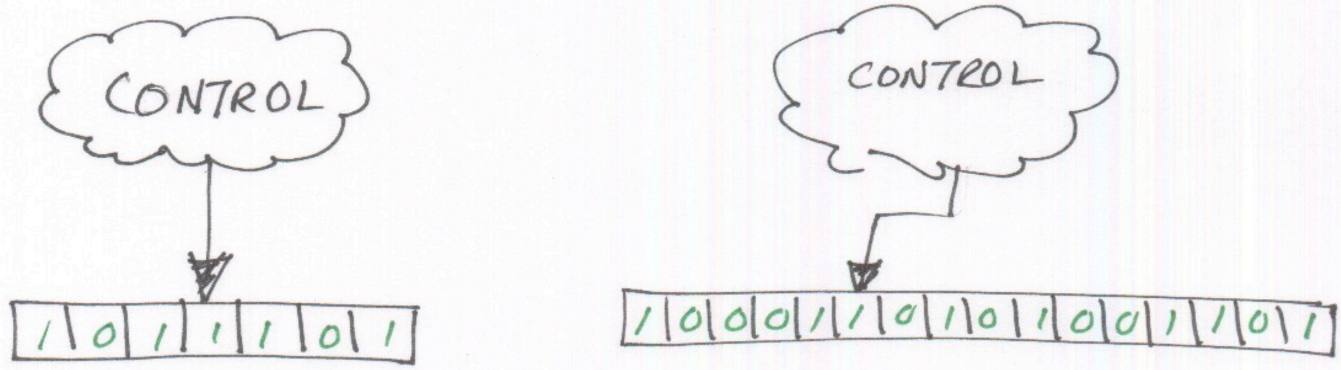
- IF YOU CAN FIND A SOLUTION TO THIS INSTANCE OF THE PCP, THEN YOU HAVE FOUND A LEGAL ACCEPTING COMPUTATION HISTORY, IN WHICH MACHINE M ACCEPTS STRING w.
- DOES A SOLUTION EXIST?
- IF YOU CAN DECIDE THE ANSWER, THEN YOU CAN DECIDE WHETHER M ACCEPTS w.
- WE KNOW THAT A_{TM} IS UNDECIDABLE.
- THEREFORE,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

(IN GENERAL).

LINEAR BOUNDED AUTOMATA

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.)

LBAs ARE <u>NOT</u> AS POWERFUL AS FULL TURING MACHINES.

BUT ARE REALLY QUITE POWERFUL.

ASSUME OUR TAPE ALPHABET CAN BE LARGER THAN INPUT ALPHABET.

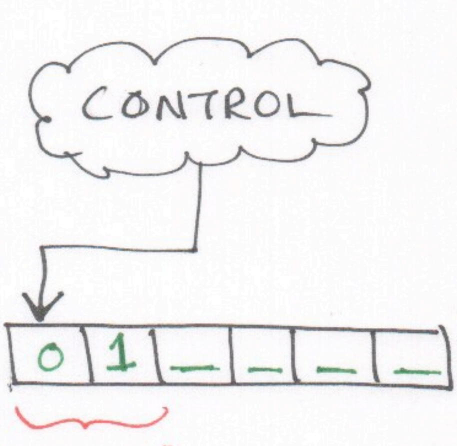
WE CAN USE A LARGER TAPE ALPHABET TO STORE MORE INFORMATION IN OUR LIMITED TAPE.

OR EQUIVALENTLY...

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

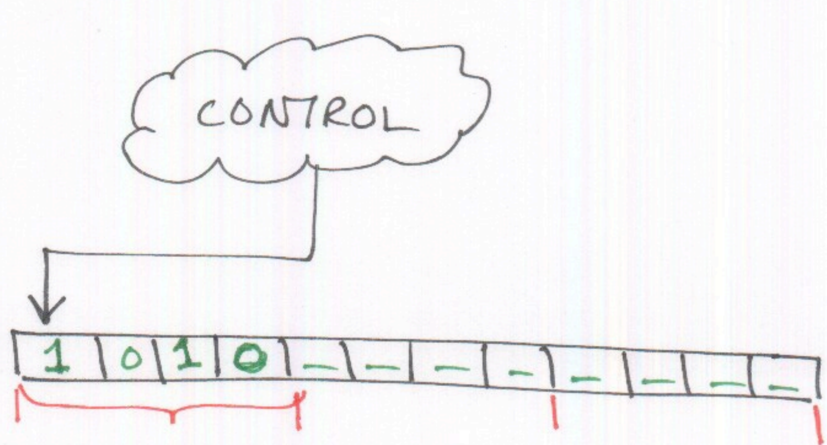
"A LINEAR FUNCTION OF THE INPUT SIZE"

"SMALL" PROBLEM



"SMALL" PROBLEM ≡ SHORT INPUT LENGTH

"LARGE" PROBLEM



TAPE SIZE LIMITED TO 3x INPUT SIZE

THEOREM

THE LANGUAGE

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

IS DECIDABLE!!

CONSIDER SOME LBA...

q = Number of states

g = Size of tape alphabet

n = Length of tape.

How many distinct configurations are there?

$$q \; n \; g^n$$

current state

current head position

contents of the tape.

This may be a big number but it is finite.

A_{LBA} IS DECIDABLE.

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

PROOF

Just simulate M on w.

M could accept, reject, or loop.

How can we detect looping?

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

There are only finitely many possible configurations.

If ~~it~~ the simulation goes qng^n that long, it <u>must</u> be looping.

So just run the simulation qng^n steps.

If ~~it~~ qng^n 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?

$$\left.\begin{array}{l} A_{DFA} \\ A_{CFG} \\ E_{DFA} \\ E_{CFG} \end{array}\right\} \text{DECIDABLE BY AN LBA.}$$