15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

THE CHURCH-TURING THESIS

Intuitive Notion of Algorithms

EQUALS

Turing Machines

UNDECIDABILITY II:
REDUCTIONS

TUESDAY Feb 18

A_{TM} = { (M,w) | M is a TM that accepts string w }

A_{TM} is undecidable: (constructive proof & subtle)

Assume machine H semi-decides A_{TM} (such exist, why?)

$$H(\ (M,w)\ ) = \begin{cases} \text{Accept if M accepts w} \\ \text{Rejects or loops otherwise} \end{cases}$$

Construct a new TM D_H as follows: on input M, run H on (M,M) and output the "opposite" of H whenever possible.

$$D_H(D_H) = \begin{cases} \text{Reject if } D_H \text{ accepts } D_H \\ \text{(i.e. if } H(D_H, D_H) = \text{Accept)} \\ \\ \text{Accept if } D_H \text{ rejects } D_H \\ \text{(i.e. if } H(D_H, D_H) = \text{Reject)} \\ \\ \text{loops if } D_H \text{ loops on } D_H \\ \text{(i.e. if } H(D_H, D_H) \text{ loops)} \end{cases}$$

Note: It must be the case that D_H loops on D_H

There is **no** contradiction here!

Thus we have **effectively** constructed an instance which does not belong to A_{TM} (namely, (D_H, D_H)) but H fails to tell us that.

That is:

Given any semi-decision machine H for A_{TM}

(and thus a potential decision machine for A_{TM}),

we can effectively construct an instance which does not belong to A_{TM} (namely, (D_H, D_H))

but H fails to tell us that.

So H cannot be a decision machine for A_{TM}

In most cases, we will show that a language L is undecidable by showing that if it is decidable, then so is A_{TM}

We reduce deciding A_{TM} to deciding the language in question

A_{TM} “<“ L

THE HALTING PROBLEM

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

Theorem: $HALT_{TM}$ is undecidable

Proof: Assume, for a contradiction, that TM H decides $HALT_{TM}$

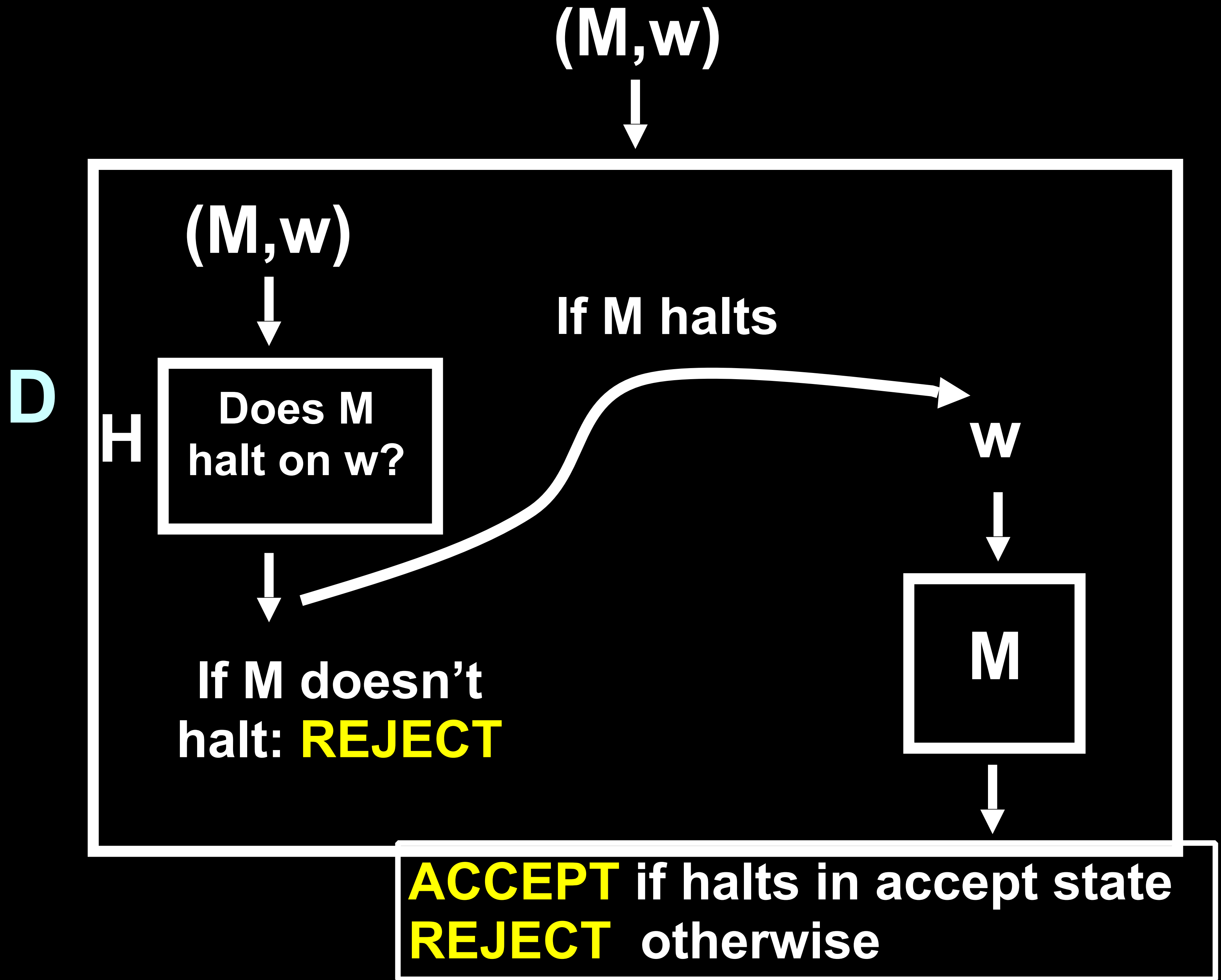
We use H to construct a TM D that decides A_{TM}

On input (M,w), D runs H on (M,w)

If H rejects then reject

If H accepts, run M on w until it halts:

Accept if M accepts and
Reject if M rejects


(M,w)
D
(M,w)
H
Does M halt on w?
If M halts
w
If M doesn't halt: REJECT
M
ACCEPT if halts in accept state
REJECT otherwise

In most cases, we will show that a language L is undecidable by showing that if it is decidable, then so is A_{TM}

We **reduce** deciding A_{TM} to deciding the language in question

A_{TM} “<“ L

So, A_{TM} “<“ $Halt_{TM}$

Is $Halt_{TM}$ “<“ A_{TM} ?

A_{TM} = { (M,w) | M is a TM that accepts string w }

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w } (*)

E_{TM} = { M | M is a TM and L(M) = ∅ } (*)

REG_{TM} = { M | M is a TM and L(M) is regular} (*)

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)} (*)

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* } (*)

ALL UNDECIDABLE

(*) Use Reductions to Prove

Which are SEMI-DECIDABLE?

What about complements?

E_{TM} = { M | M is a TM and L(M) = ∅ }

Theorem: E_{TM} is undecidable

Proof: Assume, for a contradiction, that TM Z decides E_{TM}. Use Z as a subroutine to decide A_{TM}

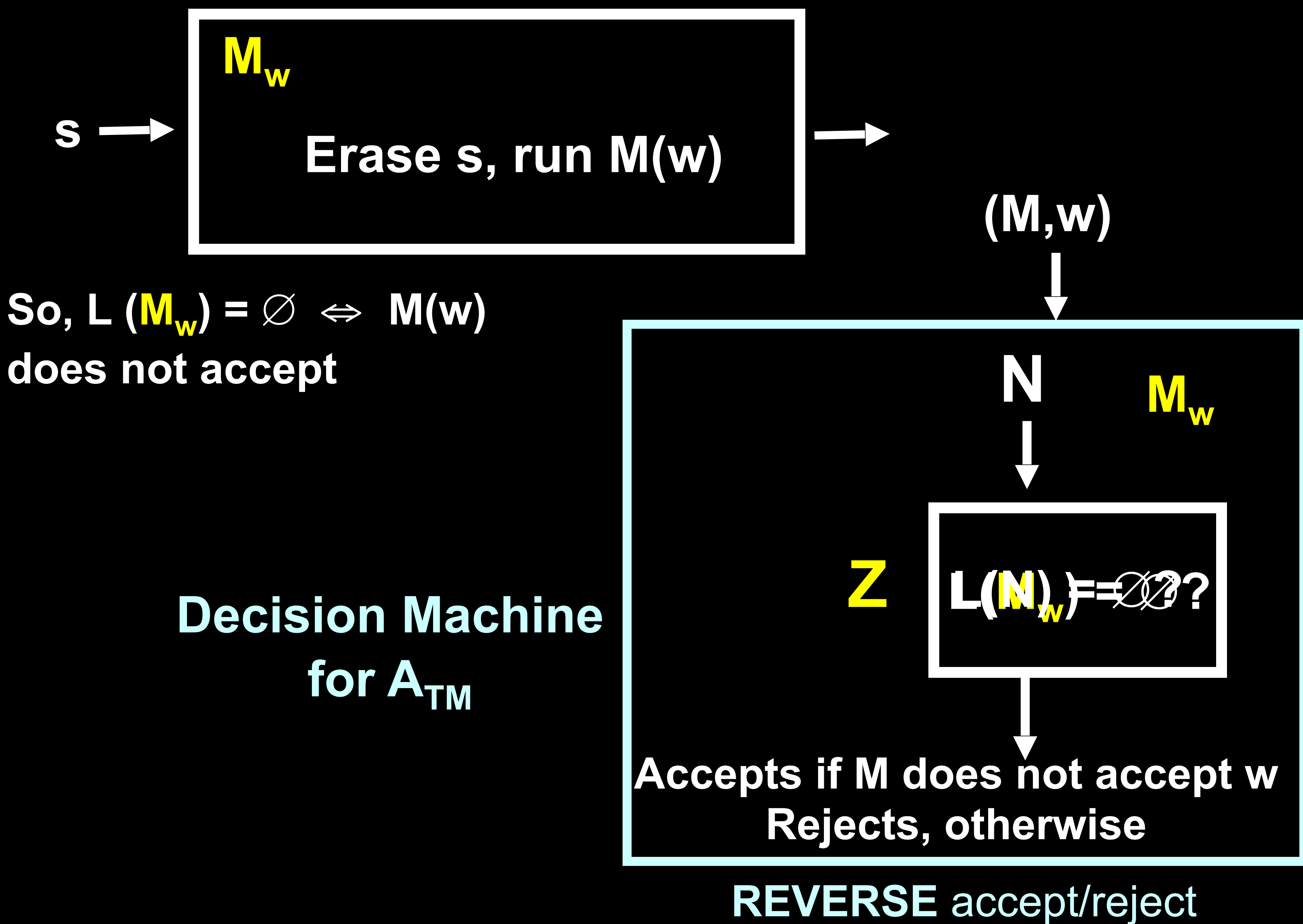
Algorithm for deciding A_{TM}: On input (M,w):

1. Create M_w

So, L (M_w) = ∅ ⇔ M(w) does not accept

L (M_w) ≠ ∅ ⇔ M(w) accepts

2. Run Z on M_w



M_w
s
Erase s, run M(w)
(M,w)
So, L (M_w) = ∅ ⇔ M(w) does not accept
N
M_w
Z
L(N) = ∅?
L(M_w) = ∅?
Decision Machine for A_TM
Accepts if M does not accept w
Rejects, otherwise
REVERSE accept/reject

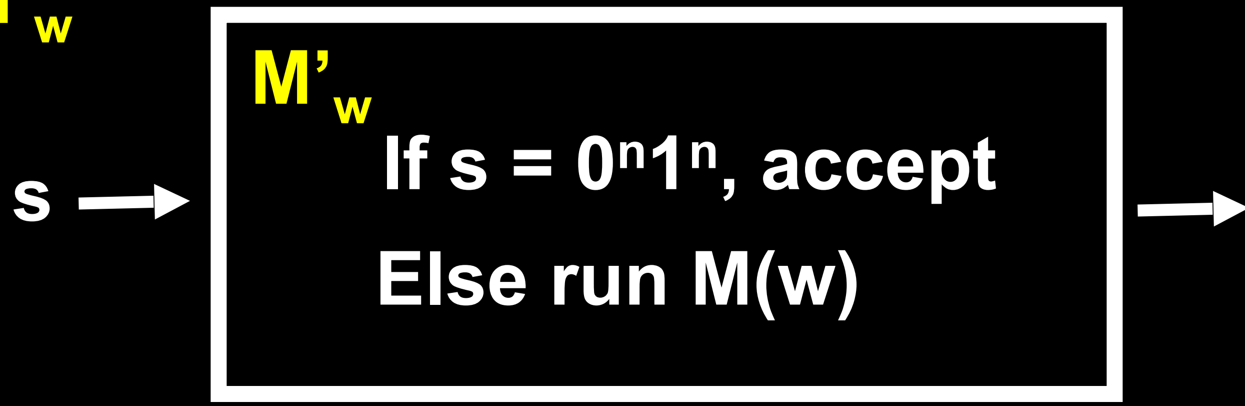
$REGULAR_{TM}$ = { M | M is a TM and L(M) is regular}

Theorem: $REGULAR_{TM}$ is undecidable

Proof: Assume, for a contradiction, that TM R decides $REGULAR_{TM}$

Use R as a subroutine to decide A_{TM}

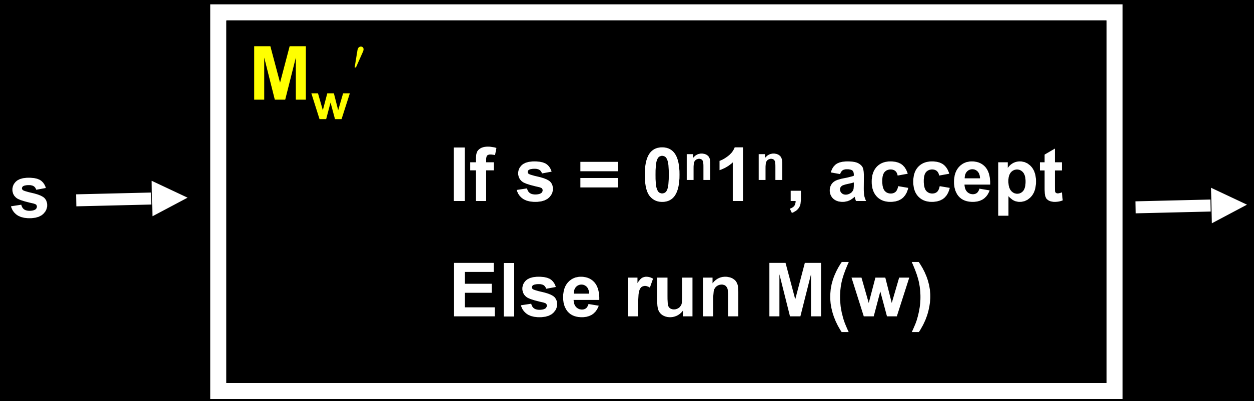
1. Create M'_w



So, L (M'_w) = Σ^* $\Leftrightarrow$ M(w) accepts

L (M'_w) = $\{0^n1^n\}$ $\Leftrightarrow$ M(w) does not accept

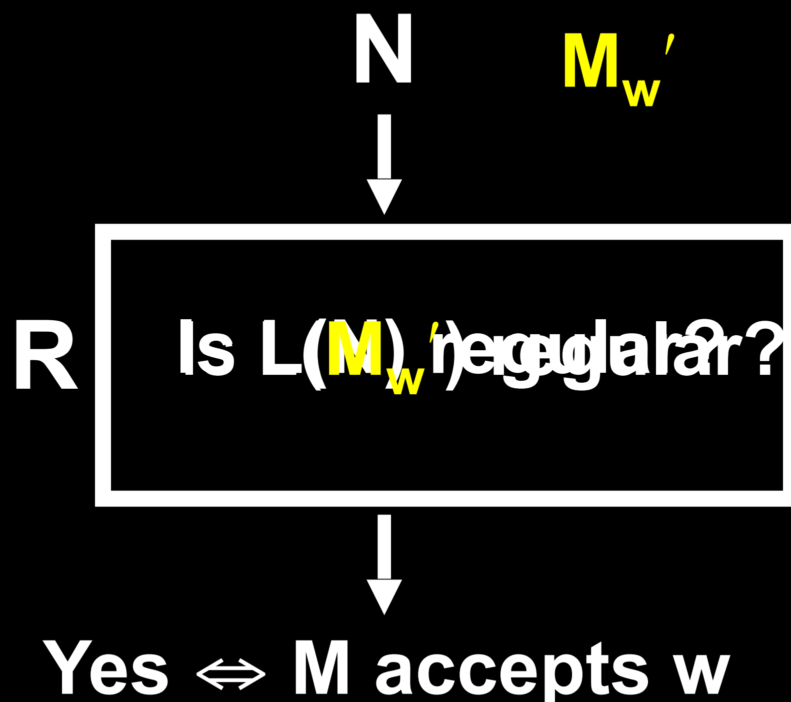
2. Run R on M'_w



$L(M_w') = \Sigma^*$ if M(w) accepts

$\{0^n1^n\}$ otherwise

$L(M_w')$ is regular $\Leftrightarrow$ M(w) accepts



MAPPING REDUCIBILITY

$f : \Sigma^* \rightarrow \Sigma^*$ is a computable function if some Turing machine M, on every input w, halts with just f(w) on its tape

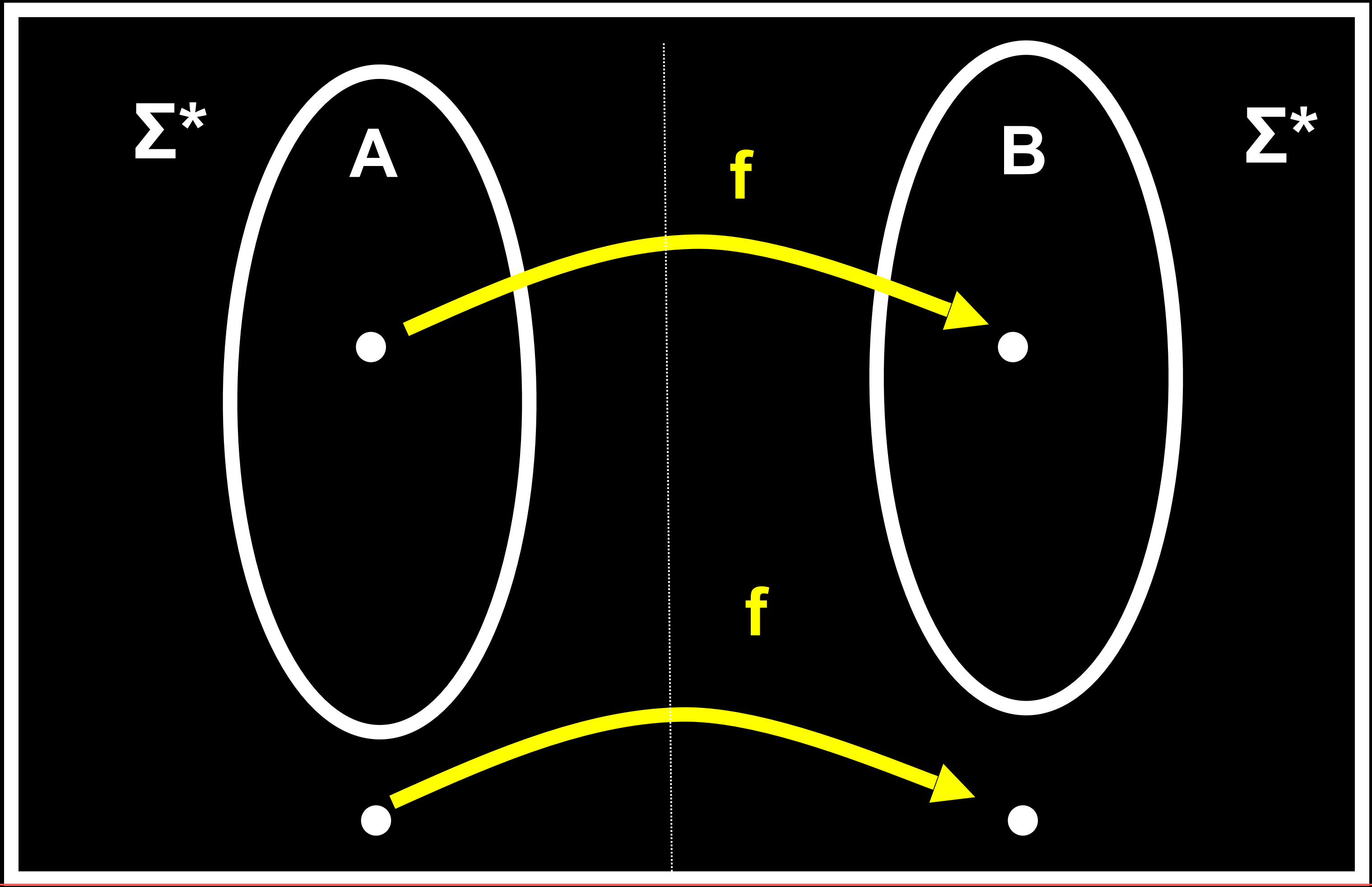
A language A is *mapping reducible* to language B, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a *reduction* from A to B

Think of f as a "computable coding"

A is **mapping reducible** to **B**, $A \leq_m B$,
if there is a computable $f : \Sigma^* \rightarrow \Sigma^*$
such that $w \in A \Leftrightarrow f(w) \in B$



Also, $\neg A \leq_m \neg B$, why?

Theorem: If $A \leq_m B$ and B is decidable, then A is decidable

Proof: Let M decide B and let f be a reduction from A to B

We build a machine N that decides A as follows:

On input w:

1. Compute $f(w)$
2. Run M on $f(w)$

Theorem: If $A \leq_m B$ and B is (semi) decidable, then A is (semi) decidable

Proof: Let M (semi) decide B and let f be a reduction from A to B

We build a machine N that (semi) decides A as follows:

On input w:

1. Compute f(w)
2. Run M on f(w)

All undecidability proofs from today can be seen as constructing an f that reduces A_{TM} to the proper language

(Sometimes you have to consider the complement of the language.)

All undecidability proofs from today can be seen as constructing an f that reduces A_{TM} to the proper language

$A_{TM} \leq_m HALT_{TM}$ (So also, $\neg A_{TM} \leq_m \neg HALT_{TM}$):

Map $(M, w) \rightarrow (M', w)$
where $M'(w) = M(w)$ if $M(w)$ accepts
loops otherwise

So $(M, w) \in A_{TM} \Leftrightarrow (M', w) \in HALT_{TM}$

A_{TM} = { (M,w) | M is a TM that accepts string w }

E_{TM} = { M | M is a TM and L(M) = $\varnothing$ }

CLAIM: $A_{TM} \leq_m \neg E_{TM}$

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

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

f: (M,w) → M_w where M_w (s) = M(w)

So, M(w) accepts ⇔ L (M_w) ≠ $\varnothing$

So, (M, w) $\in A_{TM}$ ⇔ $M_w \in \neg E_{TM}$

So $\neg E_{TM}$ is NOT DECIDABLE, but it is SEMI-DECIDABLE (why?) Is E_{TM} SEMI-DECIDABLE?

A_{TM} = { (M,w) | M is a TM that accepts string w }

REG_{TM} = { M | M is a TM and L(M) is regular}

CLAIM: $A_{TM} \leq_m REG_{TM}$ So REG_{TM} is UNDECIDABLE

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

f: (M,w) → M'_w where	M'_w (s) = accept if s = 0^n1^n
	M(w) otherwise

So, L (M'_w) = Σ^* if M(w) accepts

$\{0^n1^n\}$ if not

So, (M, w) $\in A_{TM} \Leftrightarrow M'_w \in REG_{TM}$

Is REG SEMI-DECIDABLE? (¬ REG is not. Why?)

A_{TM} = { (M,w) | M is a TM that accepts string w }

REG_{TM} = { M | M is a TM and L(M) is regular}

CLAIM: $\neg A_{TM} \leq_m REG_{TM}$ So REG_{TM} is NOT SEMI-DECIDABLE

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

f: (M,w) → M''$_w$ where	M''$_w$ (s) = accept if $s = 0^n1^n$ *and* M(w) accepts Loop otherwise

So, L (M'$_w$) = $\{0^n1^n\}$ if M(w) accepts
$\varnothing$ if not

So, (M, w) $\notin A_{TM} \Leftrightarrow$ M''$_w$ $\in REG_{TM}$

So, REG NOT SEMI-DECIDABLE

A_{TM} = { (M,w) | M is a TM that accepts string w }

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

E_{TM} = { M | M is a TM and L(M) = $\emptyset$ }

REG_{TM} = { M | M is a TM and L(M) is regular}

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)}

ALL_{PDA} = { P | P is a PDA and L(P) = Σ^* }

ALL UNDECIDABLE

Which are SEMI-DECIDABLE?

What about complements?

$E_{TM} = \{ M \mid M \text{ is a TM and } L(M) = \varnothing \}$

$EQ_{TM} = \{(M, N) \mid M, N \text{ are TMs and } L(M) = L(N)\}$

CLAIM: $E_{TM} \leq_m EQ_{TM}$ So EQ_{TM} is UNDECIDABLE

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

$f : M \to (M, M_{\varnothing})$ where $M_{\varnothing}(s)$ = Loops

So, $M \in E_{TM} \Leftrightarrow (M, M_{\varnothing}) \in EQ_{TM}$

Is EQ_{TM} SEMI-DECIDABLE? NO, since,

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

What about $\neg EQ_{TM}$?

A_{TM} = { (M,w) | M is a TM that accepts string w }

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)}

CLAIM: $A_{TM} \leq_m EQ_{TM}$

So $\neg EQ_{TM}$ is not semi-decidable

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

$f : (M,w) \rightarrow (M_w, M_A)$

Where for each s in Σ^*,

M_w (s) = M(w) and M_A(s) always accepts

So, $(M,w) \in A_{TM} \Leftrightarrow (M_w, M_A) \in EQ_{TM}$

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

Undecidable given a TM to tell if the language it recognizes is empty. It's not even semi-decidable, altho it is semi-decidable to tell if the language is non-empty.

$A_{TM} \leq_m REG_{TM}$

$A_{TM} \leq_m \neg REG_{TM}$

Undecidable given a TM to tell if it is equivalent to a FSM. It's not even semi-decidable, nor is it semi-decidable to tell if it is not equivalent to a FSM.

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

So, $\neg A_{TM} \leq_m EQ_{TM}$

Also, $A_{TM} \leq_m EQ_{TM}$

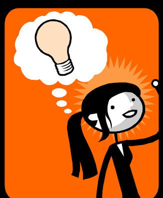
Undecidable given 2 TMs to tell if they are equivalent. It's not even semi-decidable, nor is it semi-decidable to tell If they are not

A_{TM} = { (M,w) | M is a TM that accepts string w }

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* }

CLAIM: $A_{TM} \leq_m \neg ALL_{PDA}$

$\neg A_{TM} \leq_m ALL_{PDA}$



CONSTRUCT f : Σ* → Σ*

Idea! More subtle construction

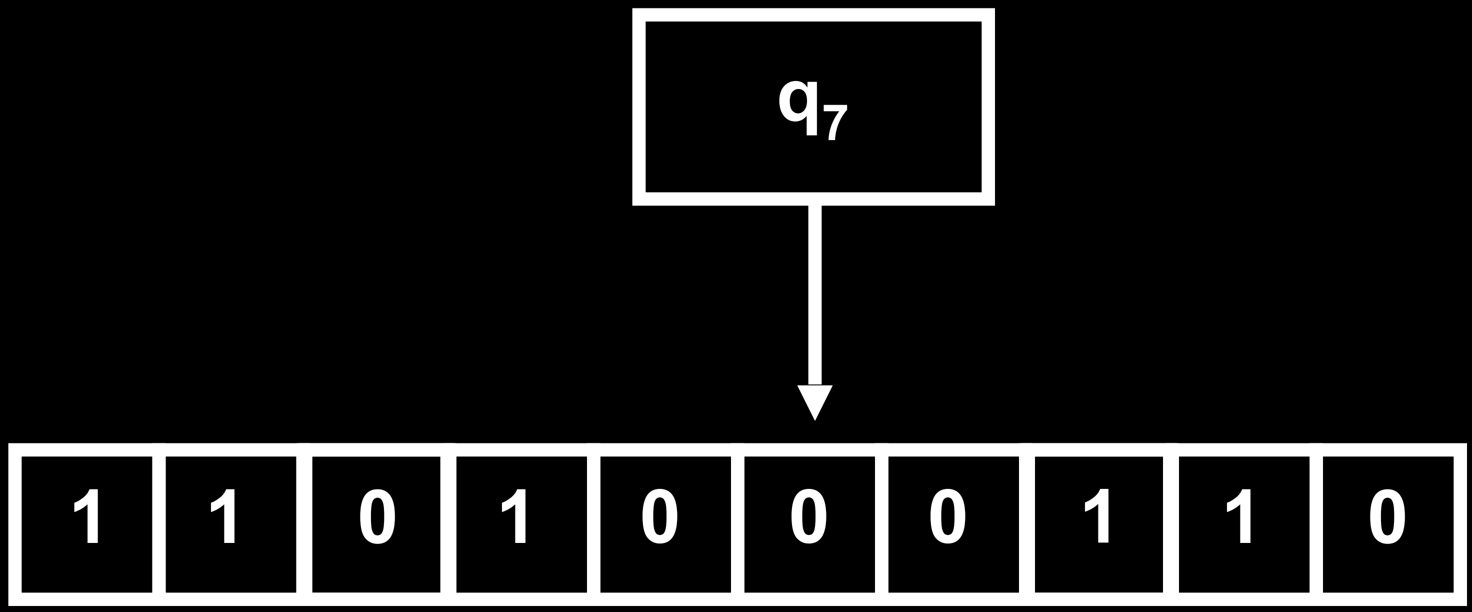
Map (M,w) to a PDA P_w that recognizes Σ* if and only if M does not accept w

So, (M, w) $\notin A_{TM} \Leftrightarrow P_w \in ALL_{PDA}$

P_w will recognize all (and only those) strings that are NOT accepting computation histories for M on w

CONFIGURATIONS

11010$q_7$00110



COMPUTATION **HISTORIES**

An **accepting computation history** is a sequence of configurations $C_1,C_2,\ldots,C_k$, where

1. C_1 is the start configuration,
2. C_k is an accepting configuration,
3. Each C_i follows from C_{i-1}

An **rejecting computation history** is a sequence of configurations $C_1,C_2,\ldots,C_k$, where

1. C_1 is the start configuration,
2. C_k is a rejecting configuration,
3. Each C_i follows from C_{i-1}

COMPUTATION **HISTORIES**

An accepting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. **C_1 is the start configuration,**
2. **C_k is an accepting configuration,**
3. **Each C_i follows from C_{i-1}**

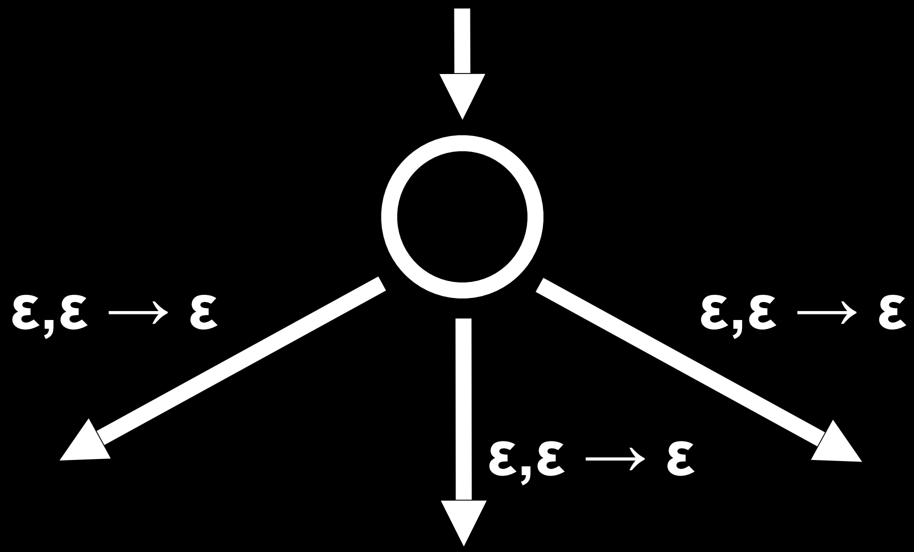
An rejecting computation history is a sequence of configurations $C_1, C_2, \ldots, C_k$, where

1. **C_1 is the start configuration,**
2. **C_k is a rejecting configuration,**
3. **Each C_i follows from C_{i-1}**

M accepts w if and only if there exists an accepting computation history that starts with $C_1 = q_0 w$

P will recognize all strings (read as sequences of configurations) that:

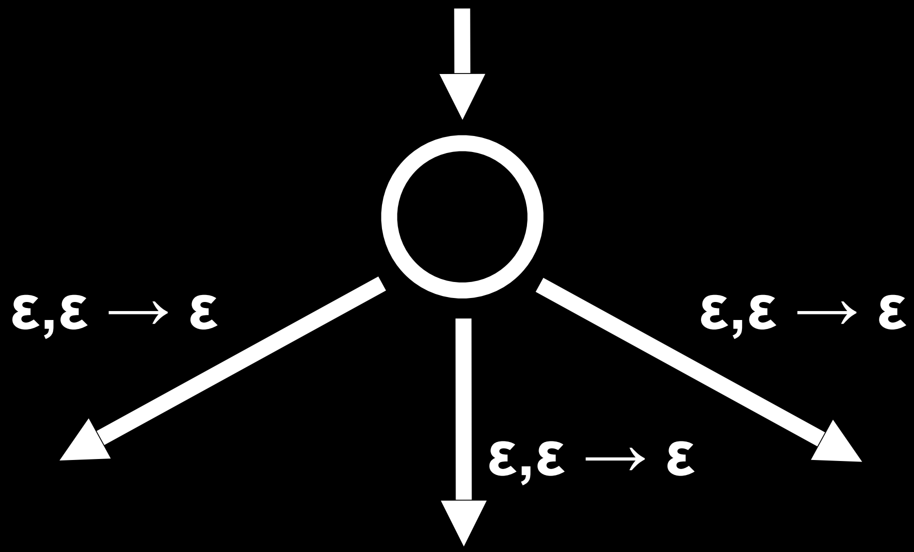
1. Do not start with C_1 or
2. Do not end with an accepting configuration or
3. Where some C_i does not properly yield C_{i+1}



Non-deterministic checks for 1, 2, and 3.

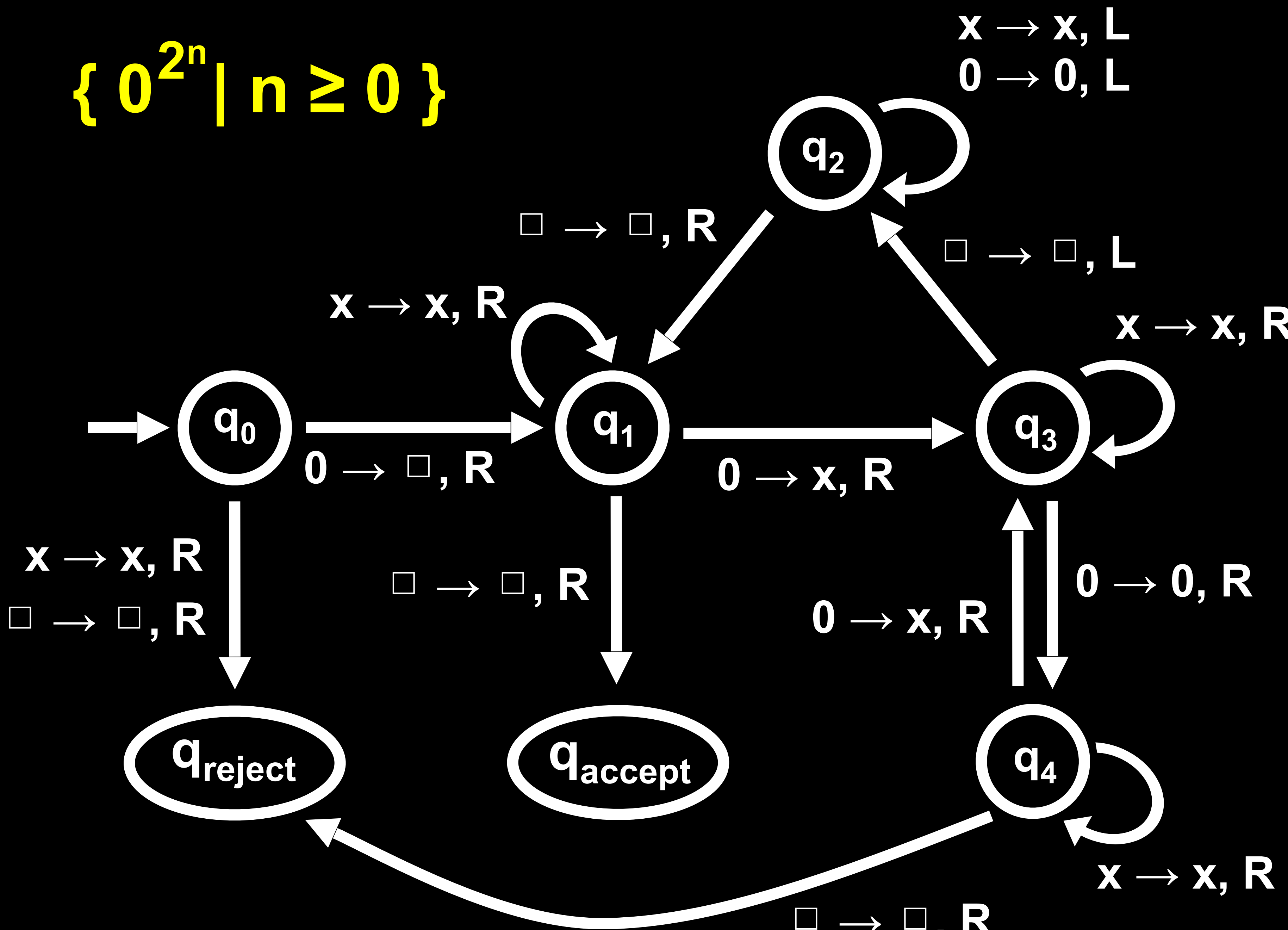
P will **reject** all strings **(read as sequences of configurations)** that:

1. Start with C_1 **and**
2. End with an accepting configuration **and**
3. Where each C_i properly yields C_{i+1}

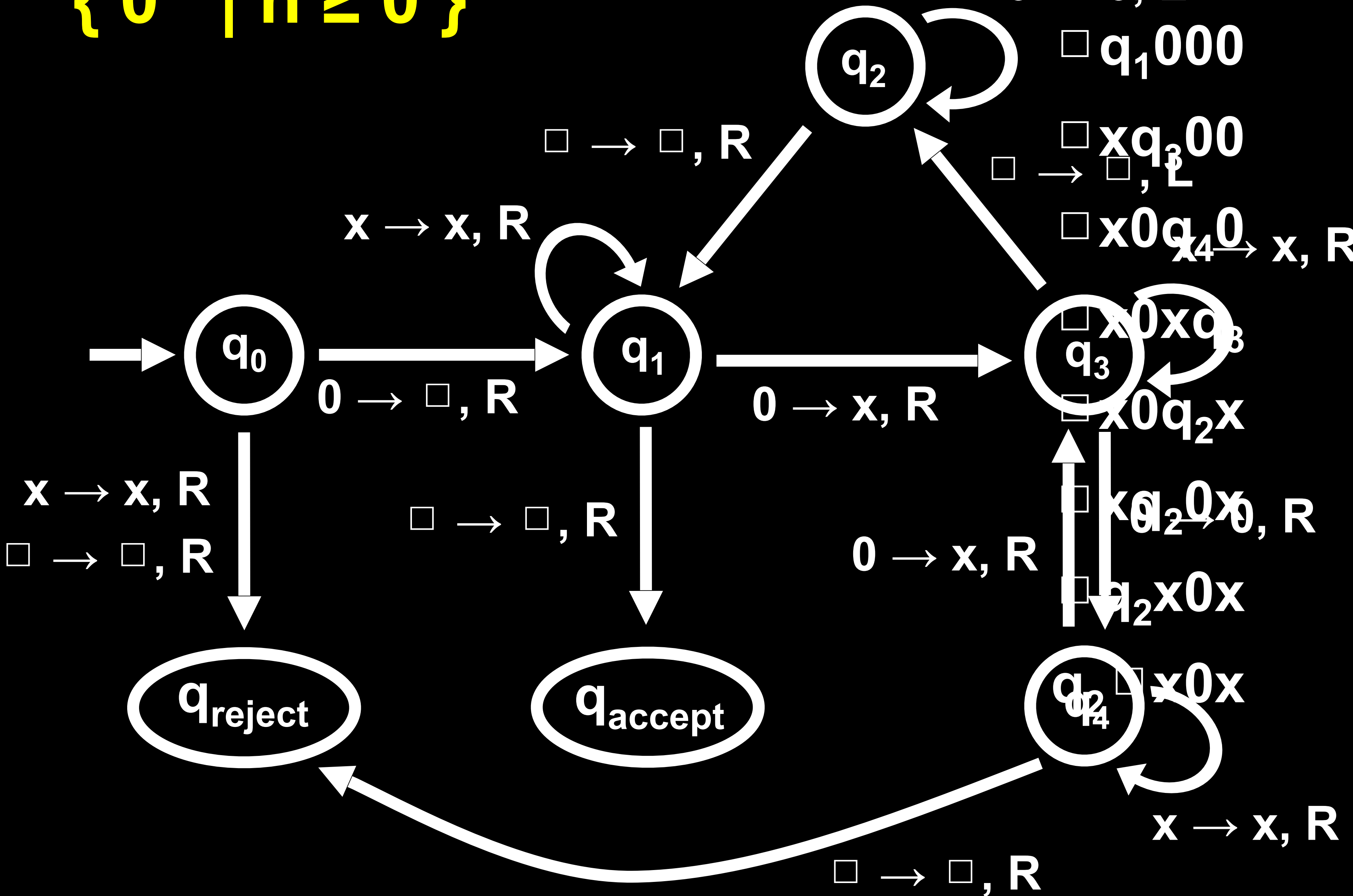


Non-deterministic checks for 1, 2, and 3.

$\{ 0^{2^n} \mid n \geq 0 \}$



$\{ 0^{2^n} \mid n \geq 0 \}$



P recognizes all strings except accepting computation histories :

$\#C_1\# \, C_2^R \#C_3 \#C_4^R \#C_5 \#C_6^R \#\ldots.\# \, C_k$

If i is odd, put C_i on stack and see if C_{i+1}^R follows properly:

For example,

If $=uaq_ibv$ and $\delta(q_i,b) = (q_j,c,R)$,

then C_i properly yields $C_{i+1} \Leftrightarrow C_{i+1} = uacq_jv$

P recognizes all strings except accepting computation histories :

$\#C_1\# \ C_2^R \#C_3 \#C_4^R \#C_5 \#C_6^R \#\ldots.\# \ C_k$

If i is odd, put C_i on stack and see if C_{i+1}^R follows properly:

For example,

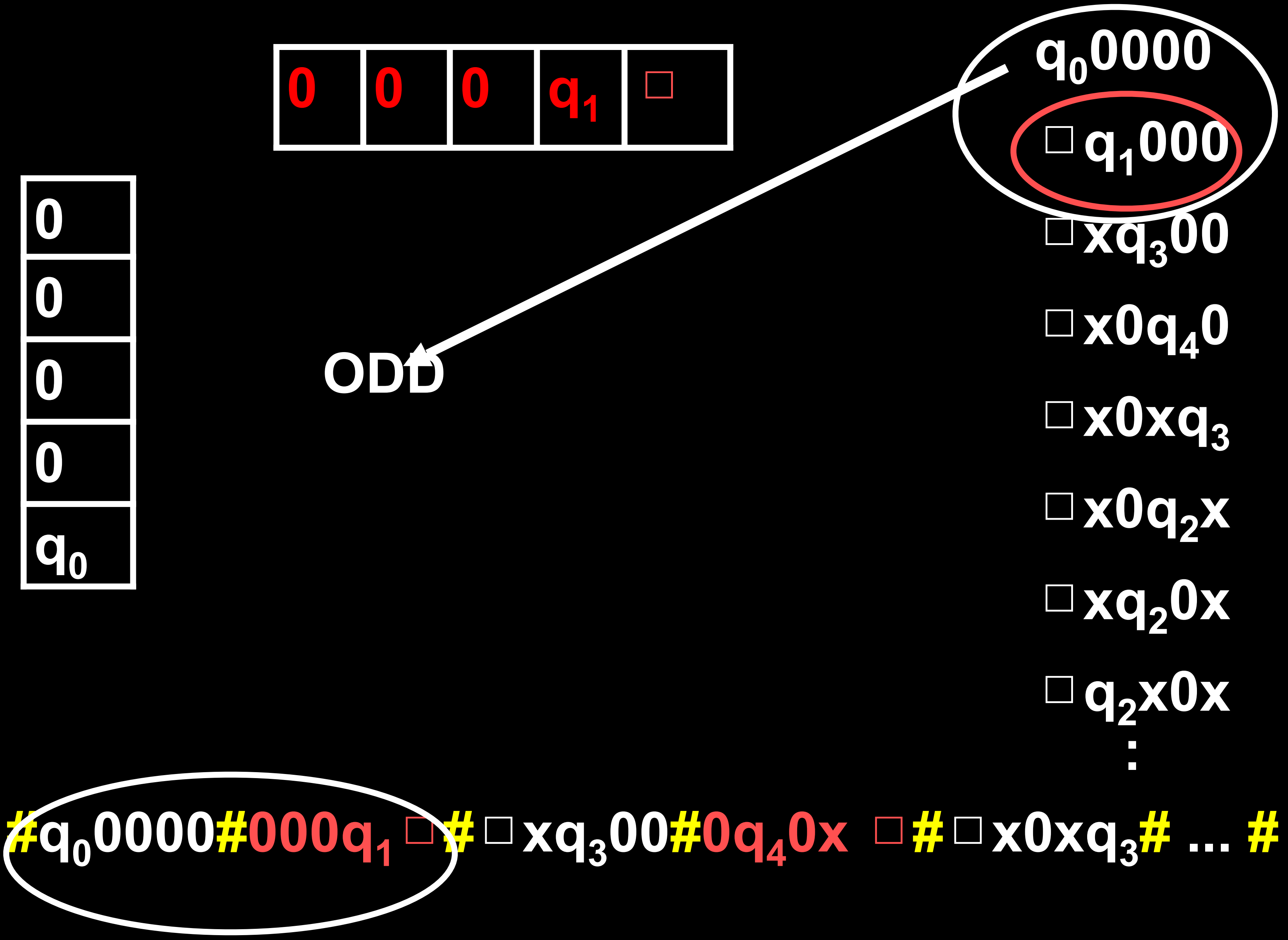
If $=uaq_ibv$ and $\delta(q_i,b) = (q_j,c,L)$,

then C_k properly yields $C_{k+1} \Leftrightarrow C_{k+1} = uq_jacv$

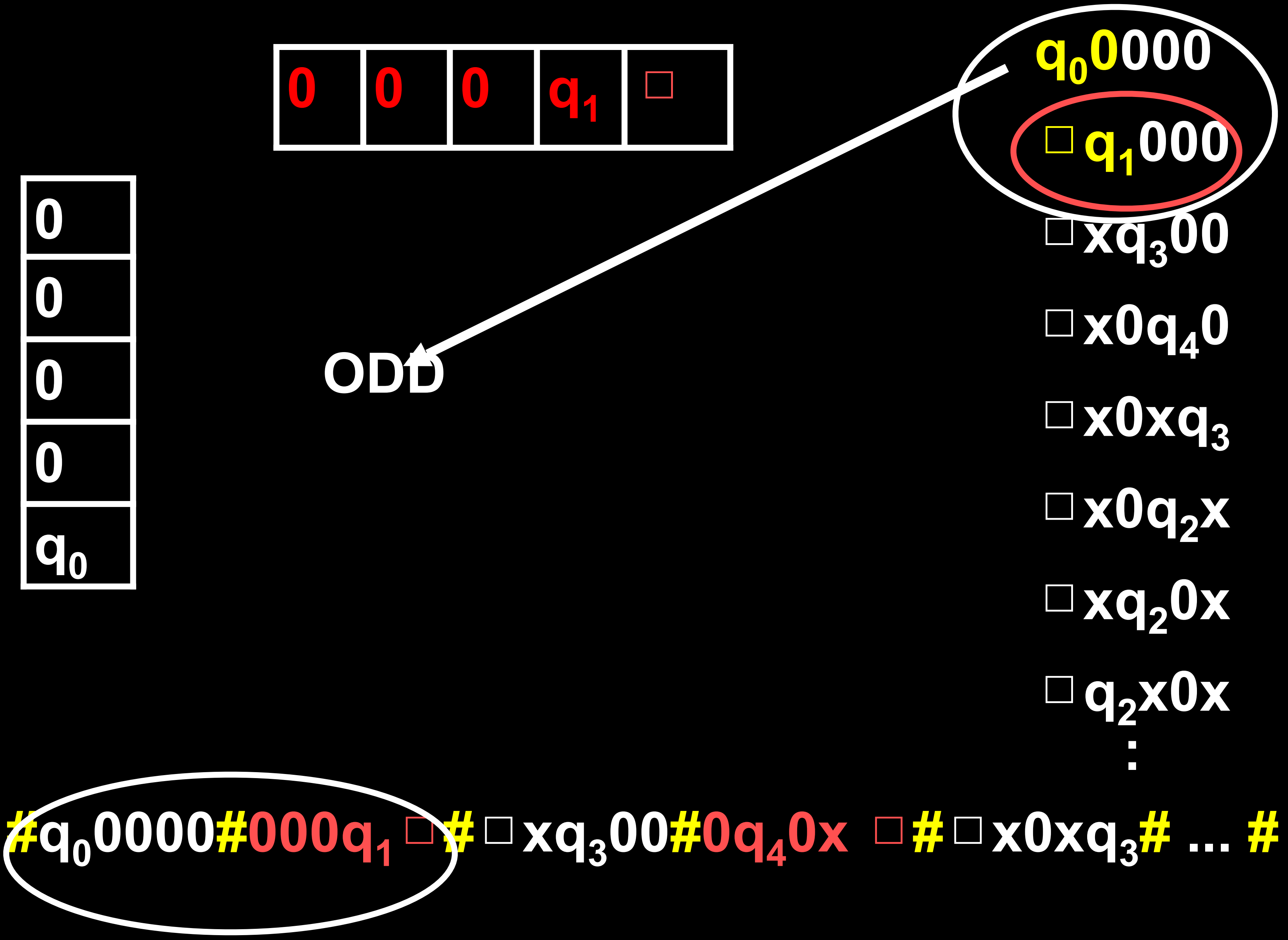
P recognizes all strings except accepting computation histories :

$\#C_1\# \; C_2^R \#C_3 \#C_4^R \#C_5 \#C_6^R \#\ldots\# \; C_k$

If i is even, put C_i^R on stack and see if C_{i+1} follows properly.



0 0 0 q1 □
0
0
0
0
q0
ODD
q0 0000
□ q1 000
□ xq3 00
□ x0q4 0
□ x0xq3
□ x0q2 x
□ xq2 0x
□ q2 x0x
:
#q0 0000#000q1 □#□xq3 00#0q4 0x □#□x0xq3# ... #



0 0 0 q_1 □
0
0
0
0
q_0
ODD
$q_0$0000
□$q_1$000
□x$q_3$00
□x0$q_4$0
□x0xq_3
□x0q_2x
□x$q_2$0x
□q_2x0x
:
#$q_0$0000#000q_1□#□x$q_3$00#0$q_4$0x□#□x0xq_3# ... #

0	0	0	q_1	□

0
0
0
0
q_0

ODD

$q_0$0000

□ $q_1$000

□ x$q_3$00

□ x0$q_4$0

□ x0xq_3

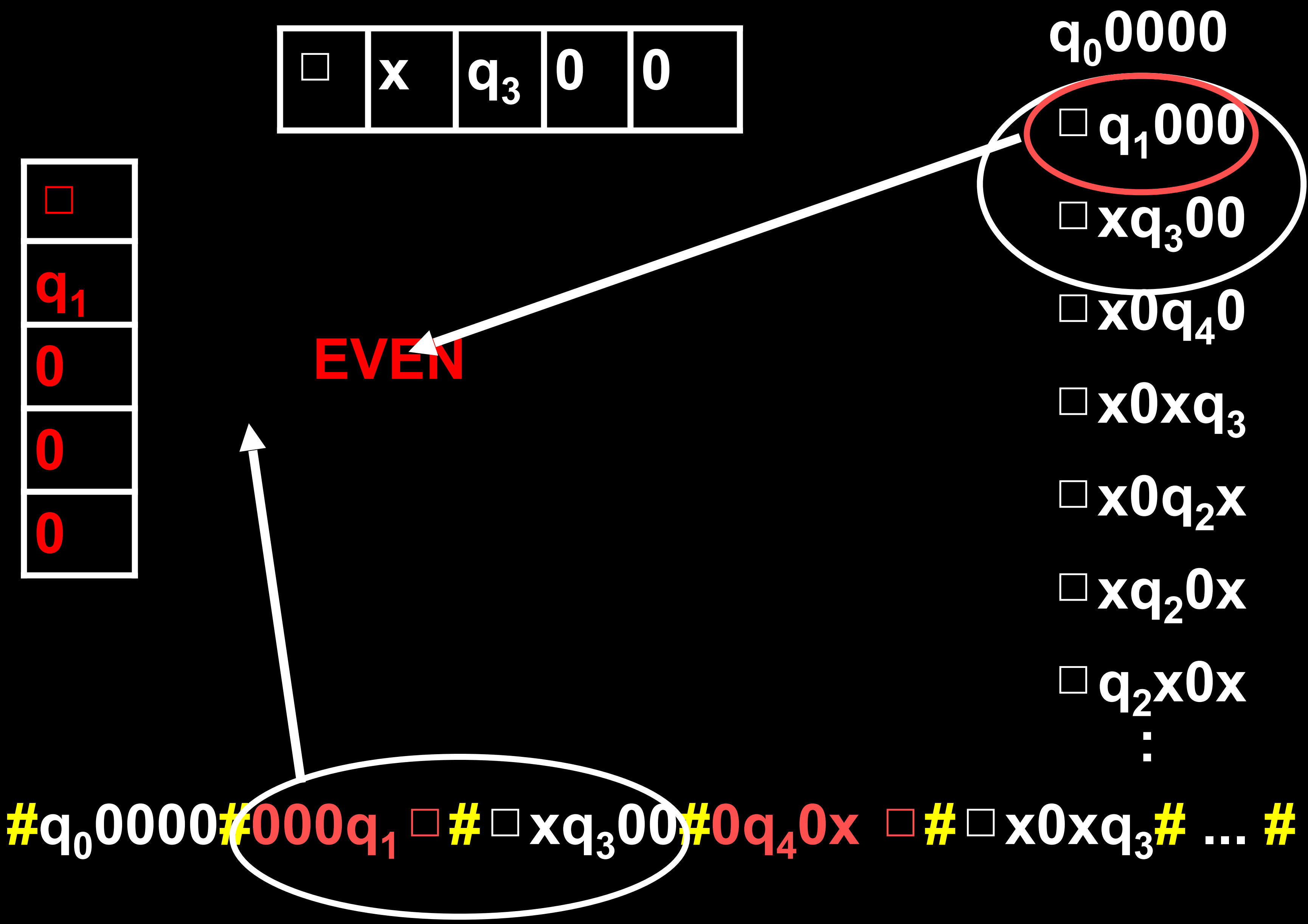
□ x0q_2x

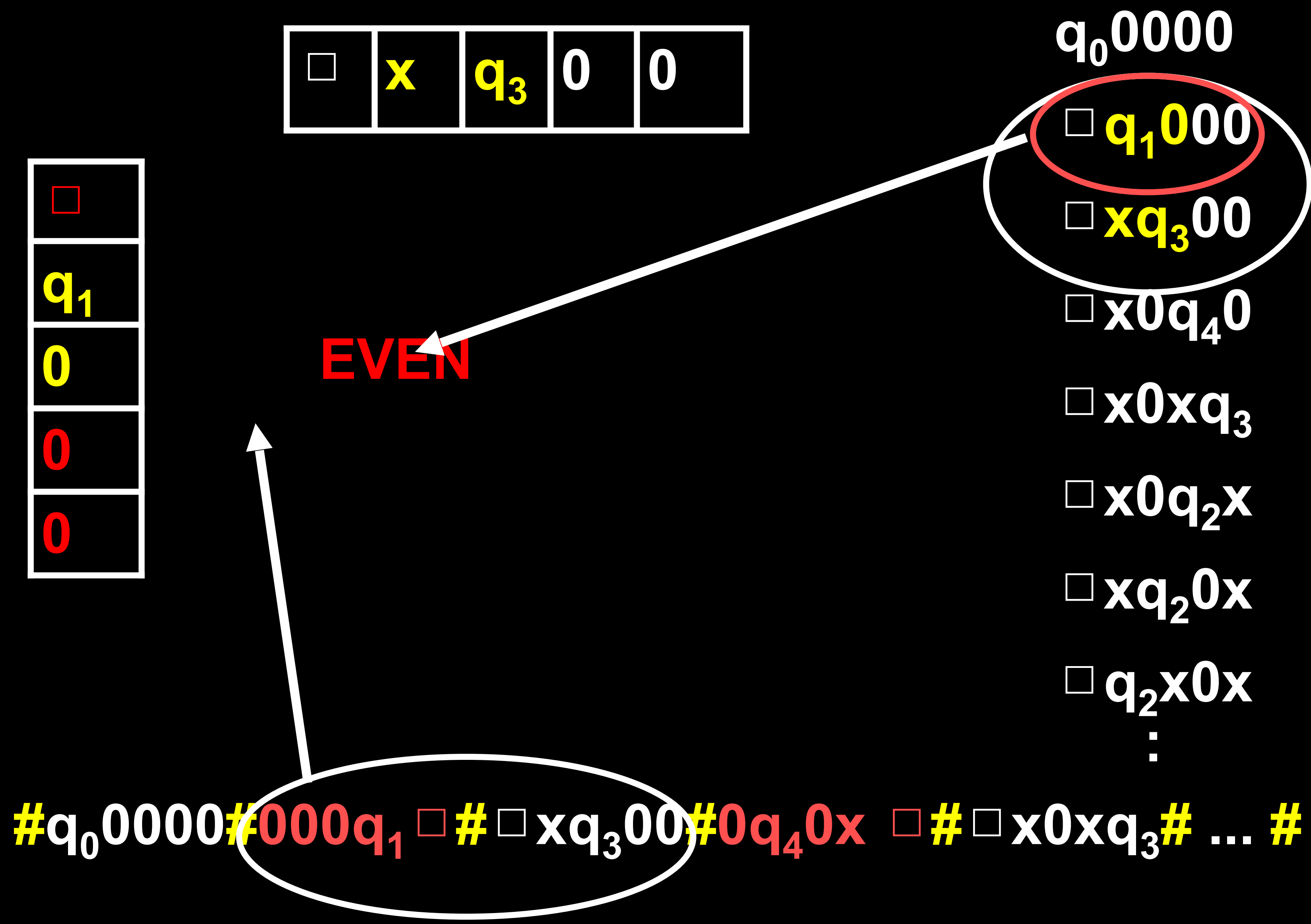
□ x$q_2$0x

□ q_2x0x

:

#$q_0$0000#000q_1 □# □ x$q_3$00#0$q_4$0x □# □ x0xq_3# ... #


□ x q3 0 0
□
q1
0
0
0
EVEN
q0 0000
□ q1 000
□ x q3 00
□ x 0 q4 0
□ x 0 x q3
□ x 0 q2 x
□ x q2 0 x
□ q2 x 0 x
:
#q0 0000#000q1 □#□xq3 00#0q4 0x □#□x0xq3# ... #



□ x q3 0 0
□ q1 0 0 0
EVEN
q0 0000
□ q1 000
□ xq3 00
□ x0q4 0
□ x0xq3
□ x0q2 x
□ xq2 0x
□ q2 x0x
:
#q0 0000#000q1 □#□xq3 00#0q4 0x □#□x0xq3# ... #

A_{TM} = { (M,w) | M is a TM that accepts string w }

ALL_{PDA} = { P | P is a PDA and L(P) = Σ* }

CLAIM: $A_{TM} \leq_m \neg ALL_{PDA}$

$\neg A_{TM} \leq_m ALL_{PDA}$

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

---

f: (M,w) → P_w where

P_w (s) = accept iff s is NOT an accepting computation of M(w)

So, (M, w) $\notin A_{TM} \Leftrightarrow P_w \in ALL_{PDA}$

So, (M, w) $\in A_{TM} \Leftrightarrow P_w \in \neg ALL_{PDA}$

EXPLAIN THE PROOF TO YOUR NEIGHBOR

A_{TM} = { (M,w) | M is a TM that accepts string w }

$HALT_{TM}$ = { (M,w) | M is a TM that halts on string w }

E_{TM} = { M | M is a TM and L(M) = $\emptyset$ }

REG_{TM} = { M | M is a TM and L(M) is regular}

EQ_{TM} = {(M, N) | M, N are TMs and L(M) =L(N)}

ALL_{PDA} = { P | P is a PDA and L(P) = Σ^* }

ALL UNDECIDABLE

Which are SEMI-DECIDABLE?

What about complements?

WWW.FLAC.WS

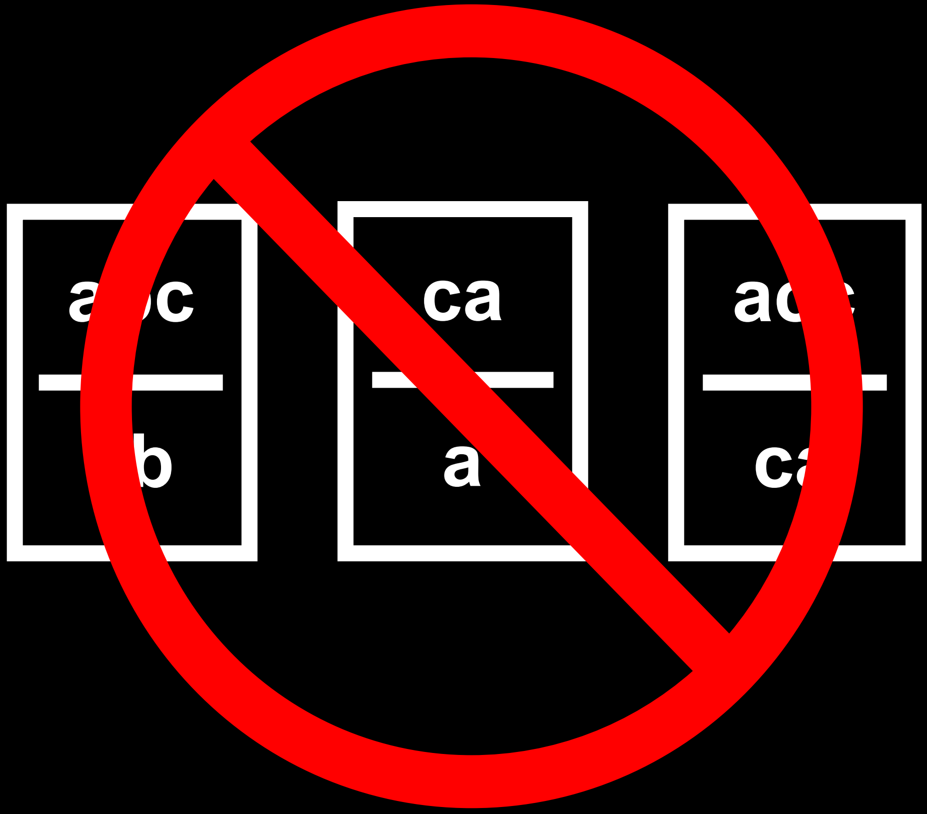
Read chapter 5.1-5.3 of the book for next time

THE PCP **GAME**

ba	a	b	b
a	ab	bcb	a

aaa	a	a	c
a	c	aa	a

$$\left[\frac{b}{ca}\right] \quad \left[\frac{a}{ab}\right] \quad \left[\frac{ca}{a}\right] \quad \left[\frac{abc}{c}\right]$$


ca
a

GENERAL RULE #1

If every top string is longer than the corresponding bottom one, there can’t be a match

caa	acc	b	aab	c
a	a	b	aa	a

GENERAL RULE #2

If there is a domino with the same string on the top and on the bottom, there is a match

POST CORRESPONDENCE **PROBLEM**

Given a collection of dominos, is there a match?

PCP = { P | P is a set of dominos with a match }

PCP is *undecidable*!

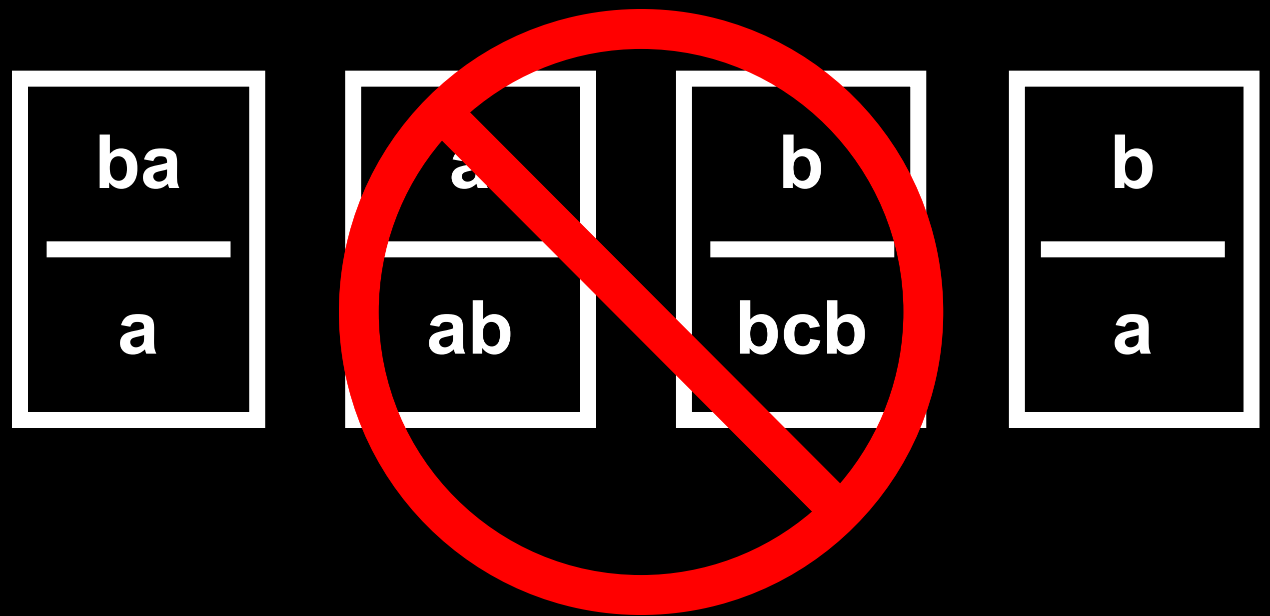
THE FPCP GAME

… is just like the PCP game except that a match has to start with the first domino

FPCP

aaa	a	a	c
a	c	aa	a

FPCP



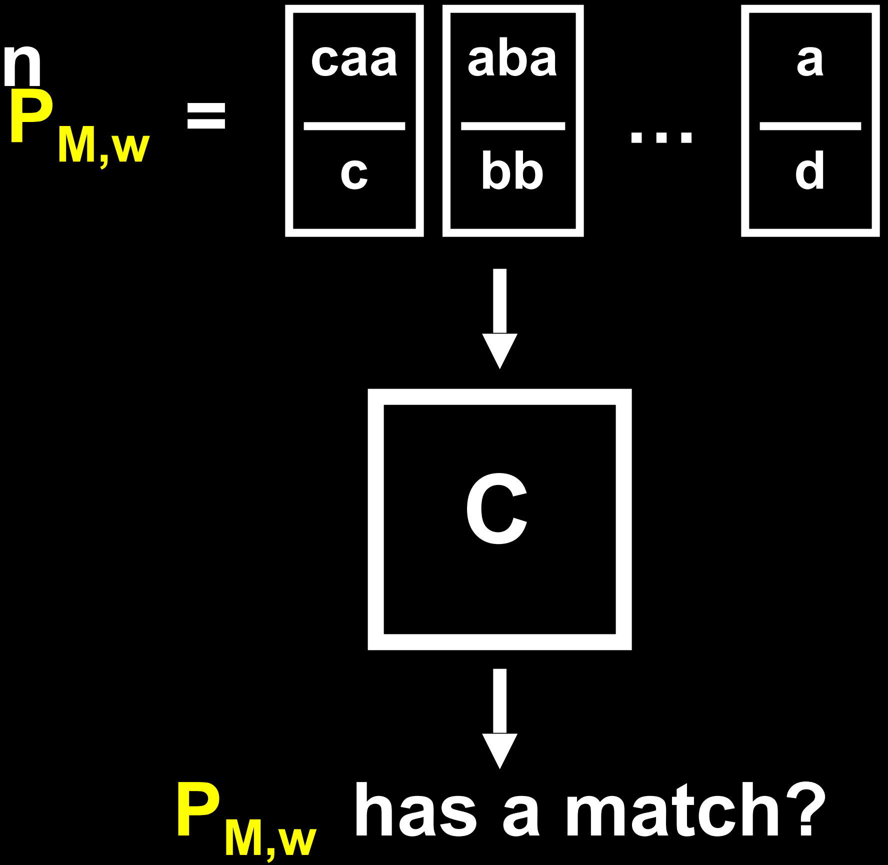
Theorem: FPCP is undecidable

Proof: Assume machine C decides FPCP

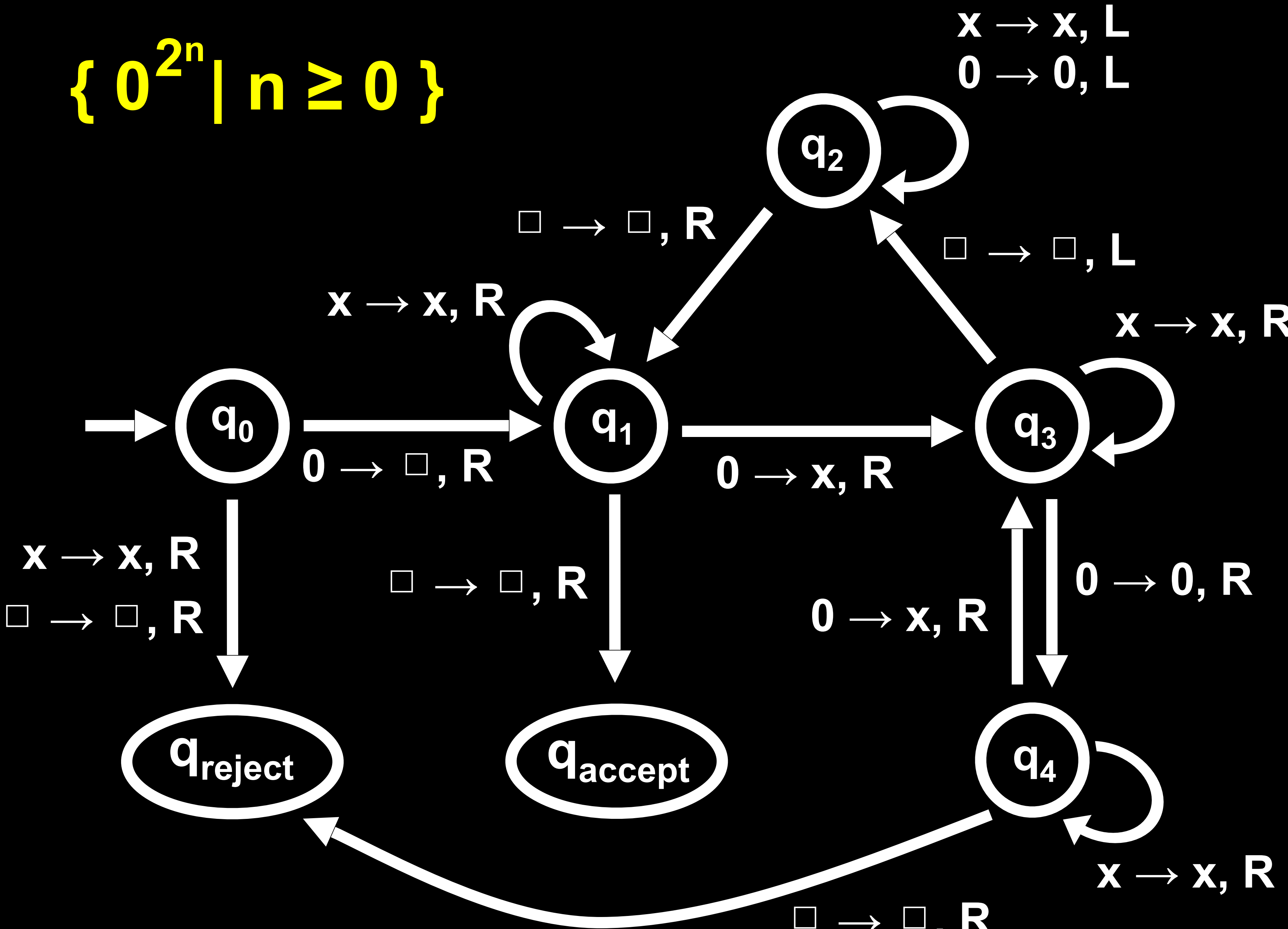
We will show how to use C to decide A_{TM}

Given (M,w)

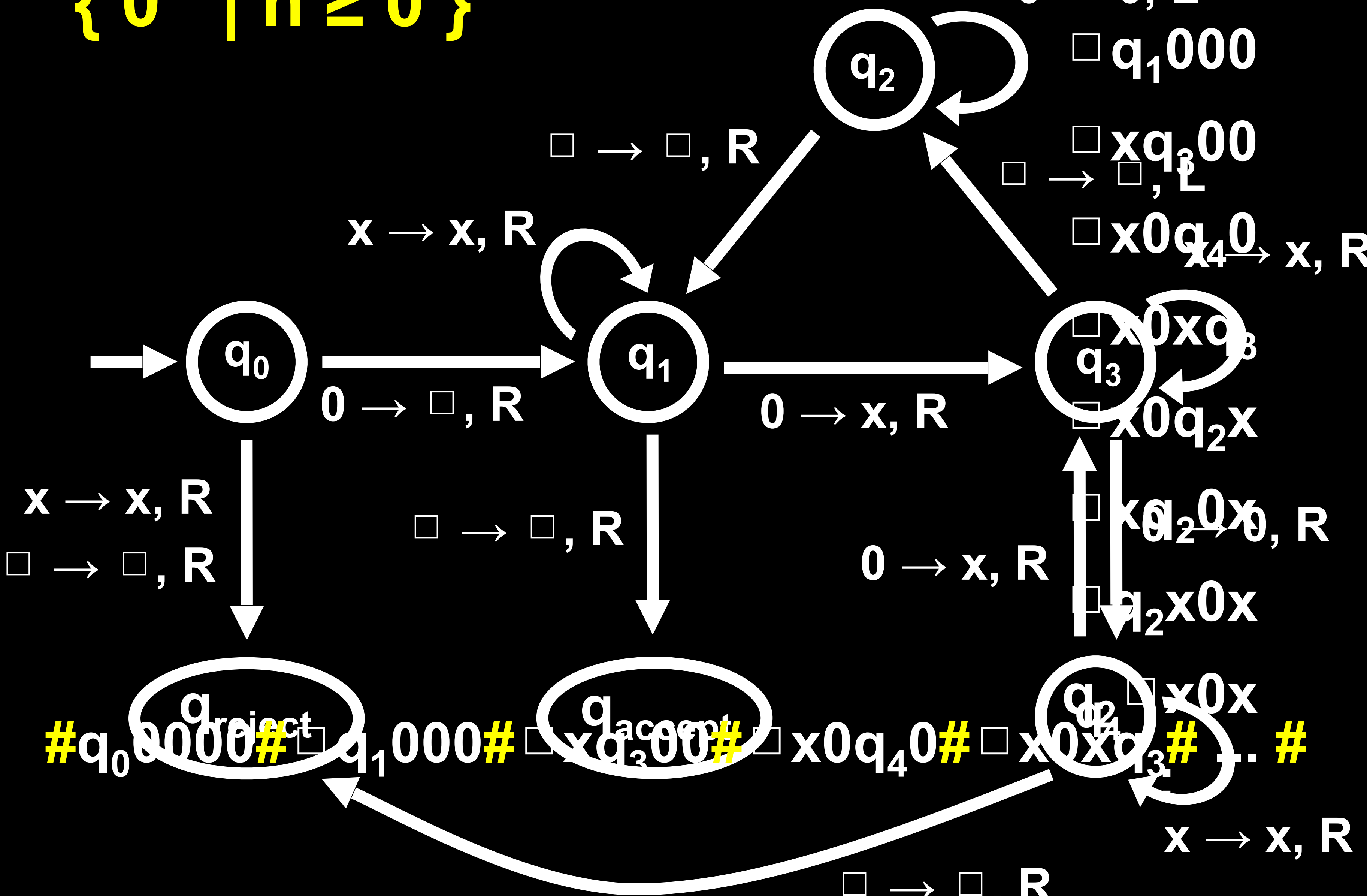
we will construct a set of dominos $P_{M,w}$ where a match is an accepting computation history for M on w



$\{ 0^{2^n} \mid n \geq 0 \}$



$\{ 0^{2^n} \mid n \geq 0 \}$



$q_0$0000

□$q_1$000

□x$q_3$00

□x0$q_4$0

□x0xq_3

□x0q_2x

□x$q_2$0x

□q_2x0x

q_2□x0x

#$q_0$0000#□$q_1$000#□x$q_3$00#□x0$q_4$0#□x0xq_3# … #

Given (M,w), we will construct an instance $P_{M,w}$ of FPCP in 7 steps

Assume M on w never attempts to move off left hand edge of tape

STEP 1

Put $\left[\dfrac{\#}{\#q_0w_1w_2\ldots w_n\#}\right]$ **into P**

For start configuration

START

STEP 2

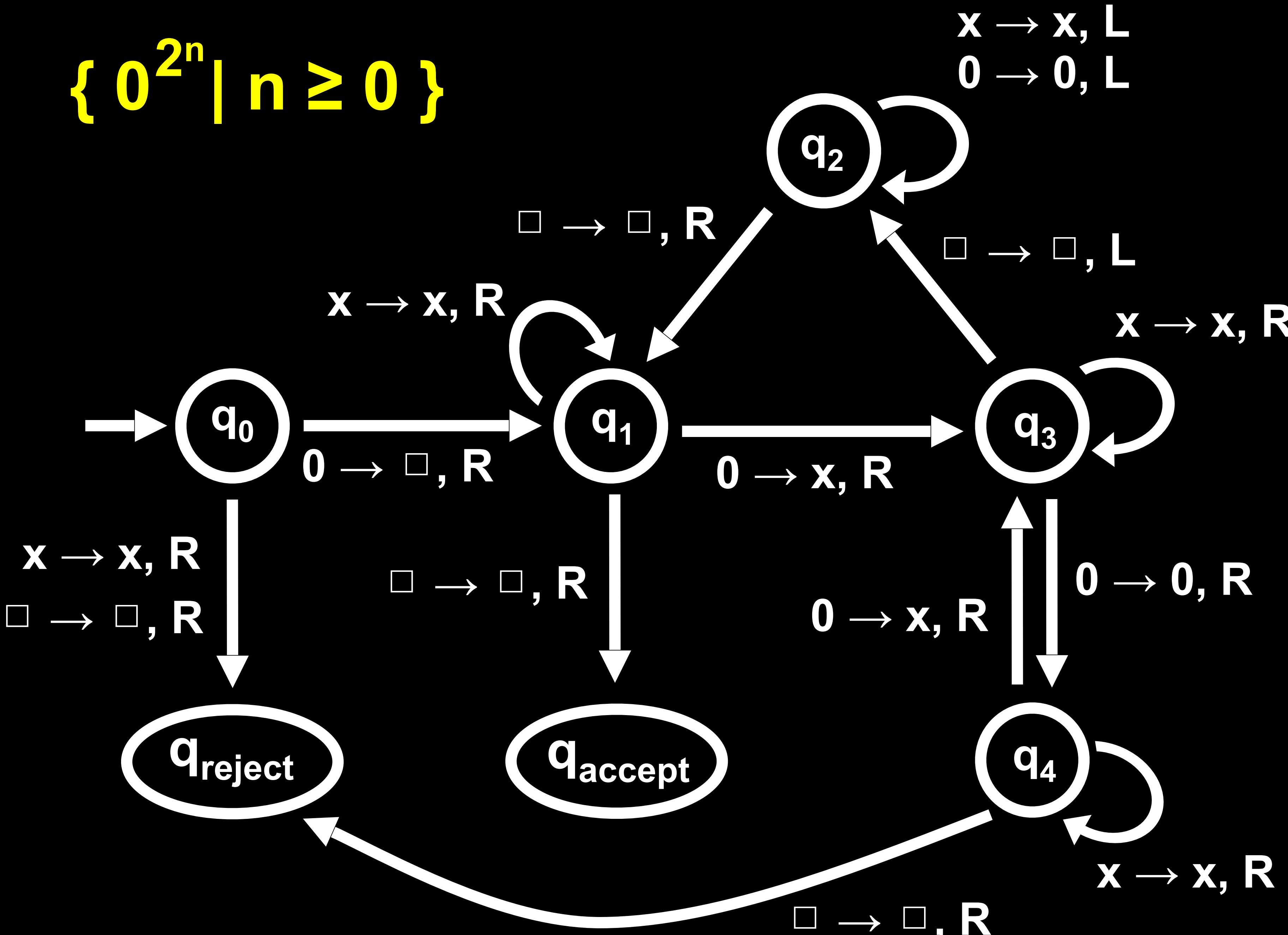
If $\delta(q,a) = (p,b,R)$ then add $\left[\frac{qa}{bp}\right]$

STEP 3

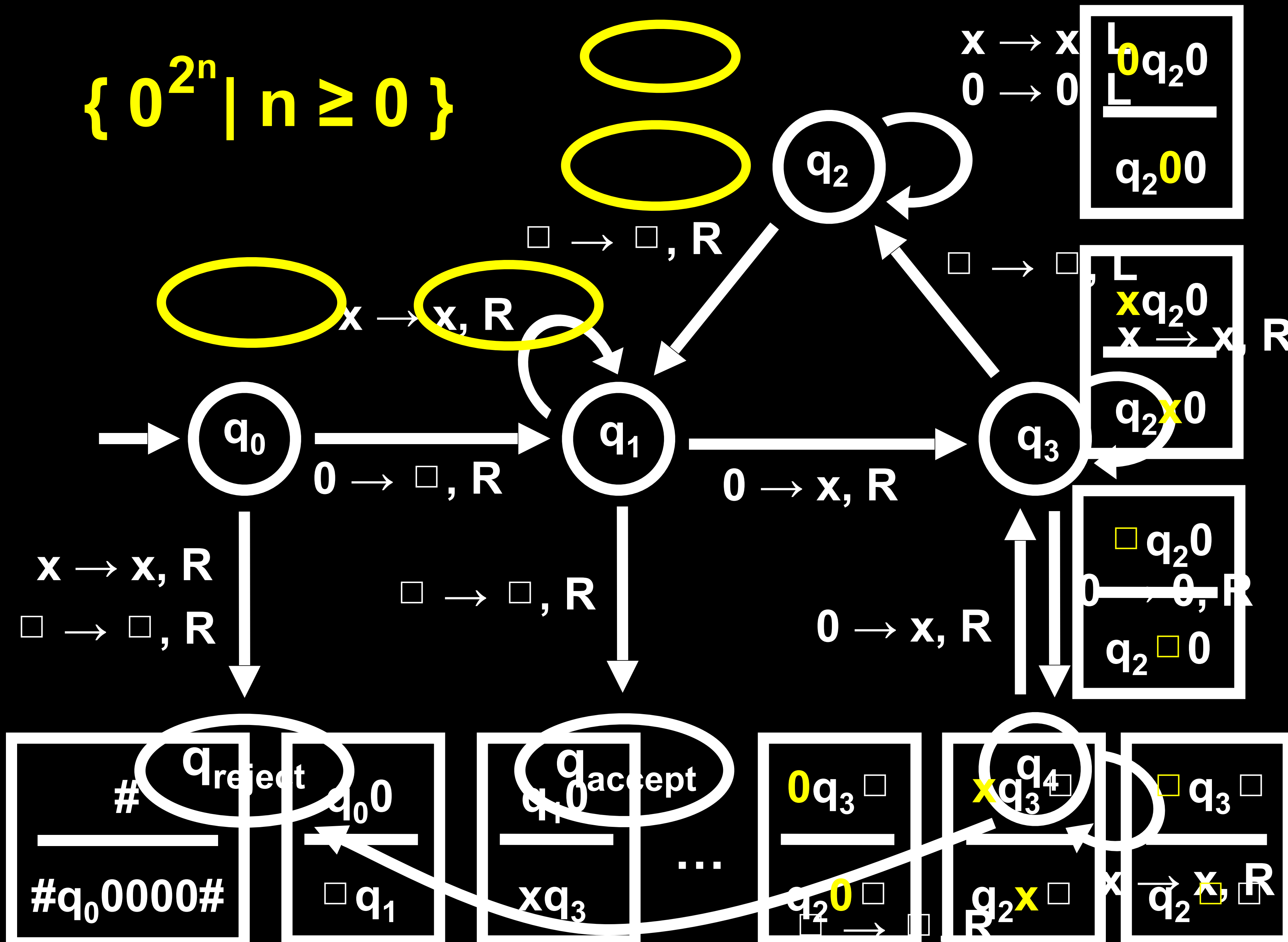
If $\delta(q,a) = (p,b,L)$ then add $\left[\frac{cqa}{pcb}\right]$ for all $c \in \Gamma$

RULES

$\{ 0^{2^n} \mid n \geq 0 \}$



$\{ 0^{2^n} \mid n \geq 0 \}$



STEP 4

add $\left[\frac{a}{a}\right]$ **for all a ∈ Γ**

For tape cells not adjacent to head

STEP 5

add $\left[\frac{\#}{\#}\right]$ $\left[\frac{\#}{\square\#}\right]$

For configuration separator

To simulate the blanks on the right hand side of tape

CONTINUE

STEP 4

add $\left[\frac{a}{a}\right]$ for all $a \in \Gamma$

STEP 5

add $\left[\frac{\#}{\#}\right]$ $\left[\frac{\#}{\square\#}\right]$

Adds pseudo-steps after TM halts (catch up)

STEP 6

add $\left[\frac{a q_{acc}}{q_{acc}}\right]$ $\left[\frac{q_{acc} a}{q_{acc}}\right]$ for all $a \in \Gamma$

#	$q_0 0$	$q_1 0$	$q_1 x$	$q_0 x$	$q_0 \square$	$q_1 \square$	$q_2 \square$
$\#q_0 0000\#$	$\square q_1$	$x q_3$	$x q_1$	$x q_r$	$\square q_r$	$\square q_a$	$\square q_1$

$q_3 x$	$q_3 0$	$q_4 0$	$q_4 x$	$q_4 \square$	$0 q_2 0$	$\square q_2 0$	$x q_2 0$
$x q_3$	$0 q_4$	$x q_3$	$x q_4$	$\square q_r$	$q_2 00$	$q_2 \square 0$	$q_2 x 0$

$0 q_3 \square$	$x q_3 \square$	$\square q_3 \square$	$0 q_3 x$	$x q_3 x$	$\square q_3 x$	x	0	$\square$
$q_2 0 \square$	$q_2 x \square$	$q_2 \square \square$	$q_2 0 x$	$q_2 x x$	$q_2 \square x$	x	0	$\square$

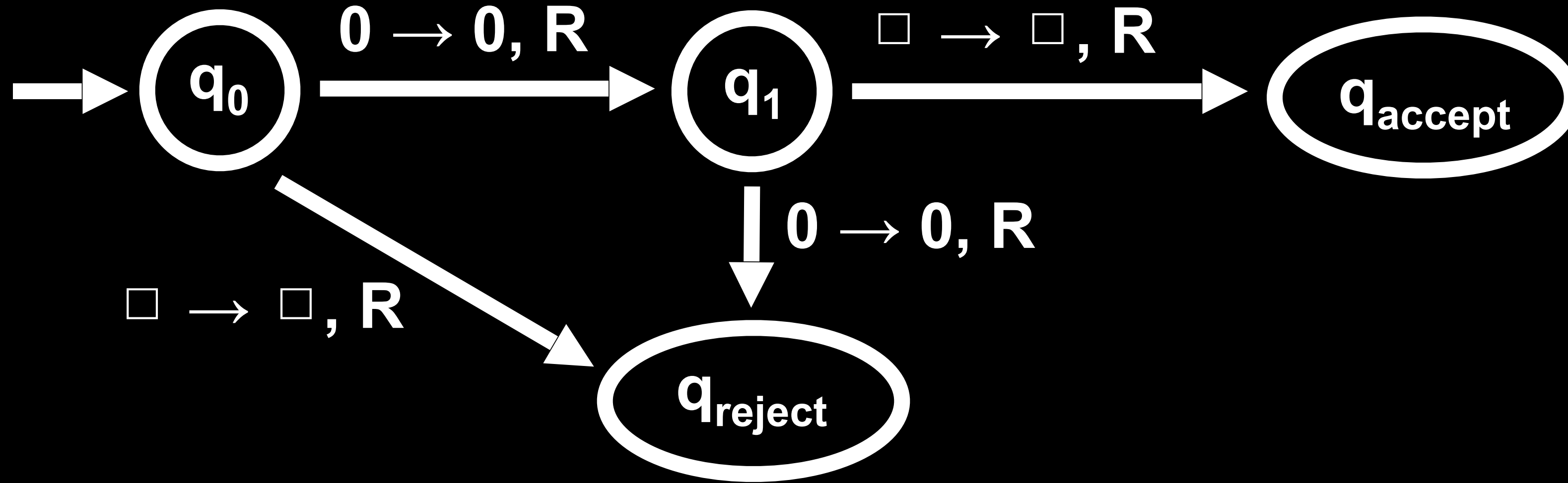
#	#	$0 q_{acc}$	$q_{acc} 0$	$x q_{acc}$	$q_{acc} x$	$\square q_{acc}$	$q_{acc} \square$
#	$\square \#$	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}

$\square$	$q_1 0$	0	0
$\square$	$x q_3$	0	0

STEP 7

add $\left[\frac{q_{acc}\#\#}{\#}\right]$

END



#	$q_0 0$	$q_1 0$	q_0 □	q_1 □	0	□
$\#q_0 0\#$	$0 q_1$	$0 q_{rej}$	□ q_{rej}	□ q_{acc}	0	□

#	#	□ q_{acc}	$0 q_{acc}$	q_{acc} □	$q_{acc} 0$	$q_{acc}\#\#$
#	□#	q_{acc}	q_{acc}	q_{acc}	q_{acc}	#

#	$q_0 0$	#	0	q_1 □	#	0	□ q_{acc}	#	$0 q_{acc}$	#
$\#q_0 0\#$	$0 q_1$	□#	0	□ q_{acc}	#	0	q_{acc}	#	q_{acc}	#

Given (M,w), we can construct an instance of FPCP that has a match if and only if M accepts w

Can convert an instance of FPCP into one of PCP:

Let $u = u_1u_2\ldots u_n$, define:

$$\star u = * u_1 * u_2 * u_3 * \ldots * u_n$$

$$u\star = u_1 * u_2 * u_3 * \ldots * u_n *$$

$$\star u\star = * u_1 * u_2 * u_3 * \ldots * u_n *$$

FPCP:

t_1	t_2	…	t_k
b_1	b_2	…	b_k

PCP:

$\star t_1$	$\star t_1$	$\star t_2$	…	$\star t_k$	$*\diamond$
$\star b_1\star$	$b_1\star$	$b_2\star$	…	$b_k\star$	$\diamond$

Given (M,w), we can construct an instance of PCP that has a match if and only if M accepts w