TIME COMPLEXITY AND POLYNOMIAL TIME; NON DETERMINISTIC TURING MACHINES AND NP

THURSDAY Mar 20

COMPLEXITY THEORY

Studies what can and can't be computed under limited resources such as time, space, etc

Today: Time complexity

Definition:

Suppose M is a TM that halts on all inputs.

The **running time** or **time-complexity** of M is the function $f : N \rightarrow N$, where $f(n)$ is the maximum number of steps that M uses *on any input of length n.*

MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt

Consider the language $A = \{ 0^k1^k \mid k \geq 0 \}$

On input of length **n**:

$\sim n$ **1. Scan across the tape and reject if the string is not of the form 0^i1^j**

$\sim n^2$ **2. Repeat the following if both 0s and 1s remain on the tape:**

Scan across the tape, crossing off a single 0 and a single 1

$\sim n$ **3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.**

ASYMPTOTIC ANALYSIS

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

BIG-O

Let f and g be two functions f, g : N → R^+. We say that $f(n) = O(g(n))$ if there exist positive integers c and n_0 so that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$, we say that g(n) is an asymptotic upper bound for f(n)

f asymptotically NO MORE THAN g

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$

$$2n^{4.1} + 200283n^4 + 2 \quad = O(n^{4.1})$$

$$3n\log_2 n + 5n \log_2\log_2 n \quad = O(n\log_2 n)$$

$$n\log_{10} n^{78} \quad = O(n\log_{10} n)$$

$$\log_{10} n = \log_2 n \,/\, \log_2 10$$

$$O(n\log_{10} n) = O(n\log_2 n) = O(n\log n)$$

Definition: TIME(t(n)) = { L | L is a language decided by a O(t(n)) time Turing Machine }

$A = \{ 0^k 1^k \mid k \geq 0 \} \in \mathrm{TIME}(n^2)$

Big-oh necessary

- Moral: big-oh notation **necessary** given our model of computation
  - Recall: $f(n) = O(g(n))$ if there exists c such that $f(n) \leq c\, g(n)$ for all sufficiently large n.
  - TM model incapable of making distinctions between time and space usage that differs by a constant.

Linear Speedup

Theorem: Suppose TM M decides language L in time f(n). Then for any $\varepsilon > 0$, there exists TM M' that decides L in time

$$\varepsilon f(n) + n + 2.$$

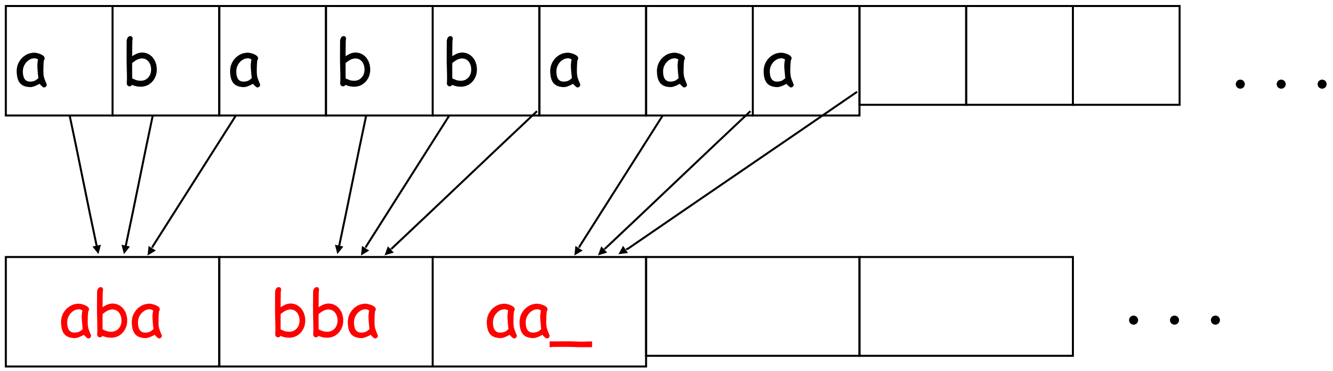
- Proof:
  - simple idea: increase "word length"
  - M' will have
    - one more tape than M
    - m-tuples of symbols of M

$$\Sigma_{new} = \Sigma_{old} \cup \Sigma_{old}^{m}$$

    - many more states

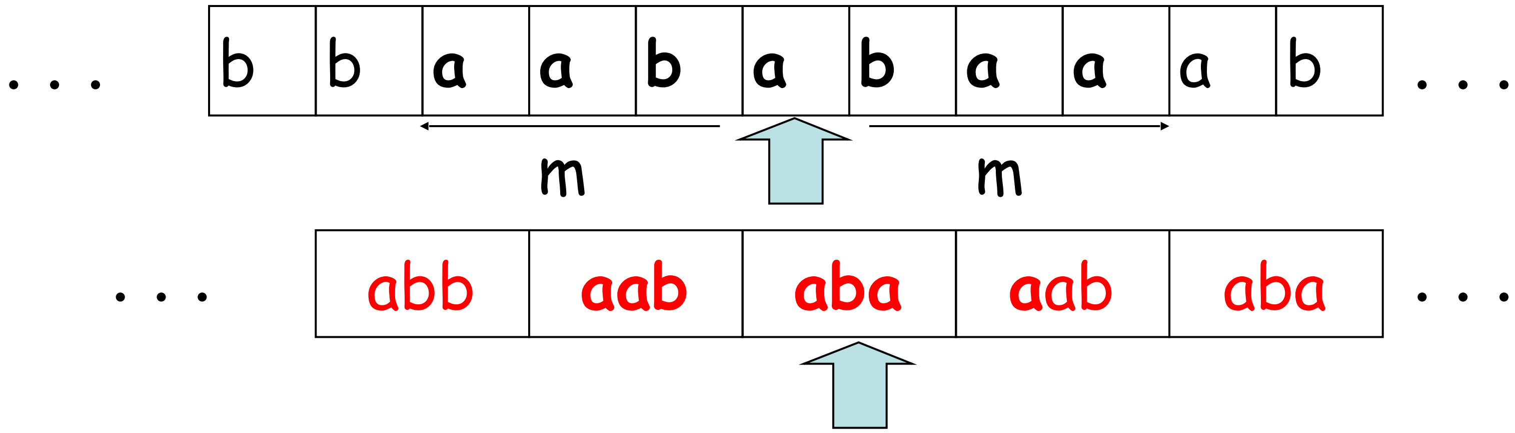
Linear Speedup

- part 1: compress input onto fresh tape



Linear Speedup

- part 2: simulate M, m steps at a time



  - 4 (L,R,R,L) steps to read relevant symbols, "remember" in state
  - 2 (L,R or R,L) to make M's changes

Linear Speedup

- accounting:
  - part 1 (copying): n + 2 steps
  - part 2 (simulation): 6 (f(n)/m)
  - set m = 6/ε
  - total: εf(n) + n + 2

Theorem: Suppose TM M decides language L in space f(n). Then for any ε > 0, there exists TM M' that decides L in space εf(n) + 2.

- Proof: same.

$A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n\log n)$

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject

```
00000000000001111111111111

x0x0x0x0x0x0xx1x1x1x1x1x1x

xxx0xxx0xxx0xxxx1xxx1xxx1x

xxxxxxx0xxxxxxxxxxxx1xxxxx

xxxxxxxxxxxxxxxxxxxxxxxxxx
```

We can prove that a one-tape TM cannot decide A in less time than O(nlog n)

***7.49 Extra Credit. Let f(n) = *o*(nlogn). Then Time(f(n)) contains only regular languages.**

where f(n) = *o*(g(n)) iff $\lim_{n\to\infty}$ f(n)/g(n) = 0

ie, for all c >0, $\exists$ n_0 such that f(n) < cg(n) for all n $\geq n_0$

f asymptotically LESS THAN g

Can A = { 0^k1^k | $k \geq 0$ } be decided in time O(n) with a two-tape TM?

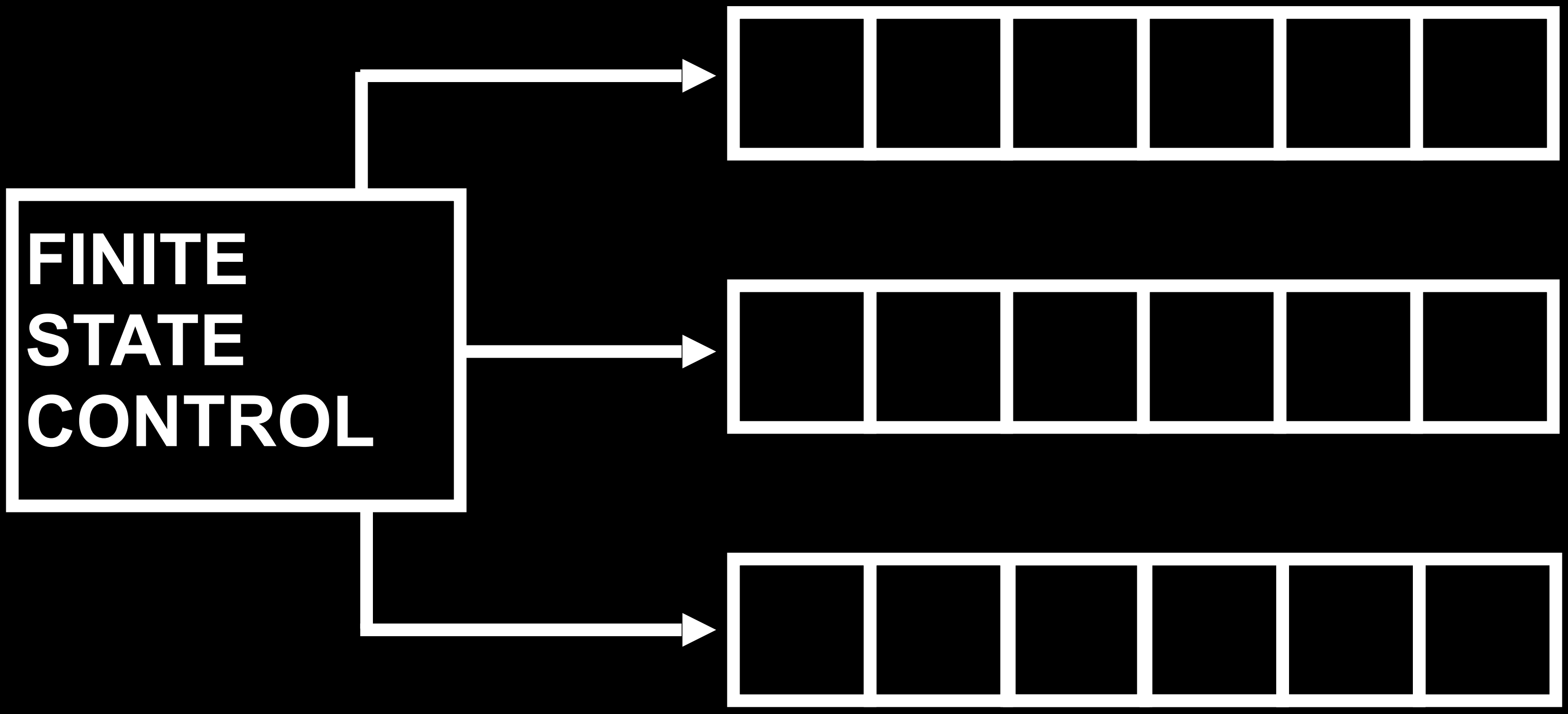
Scan all 0s and copy them to the second tape. Scan all 1s, crossing off a 0 from the second tape for each 1.

Different models of computation yield different running times for the same language!

Theorem: Let t(n) be a function such that t(n) ≥ n. Then every t(n)-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM

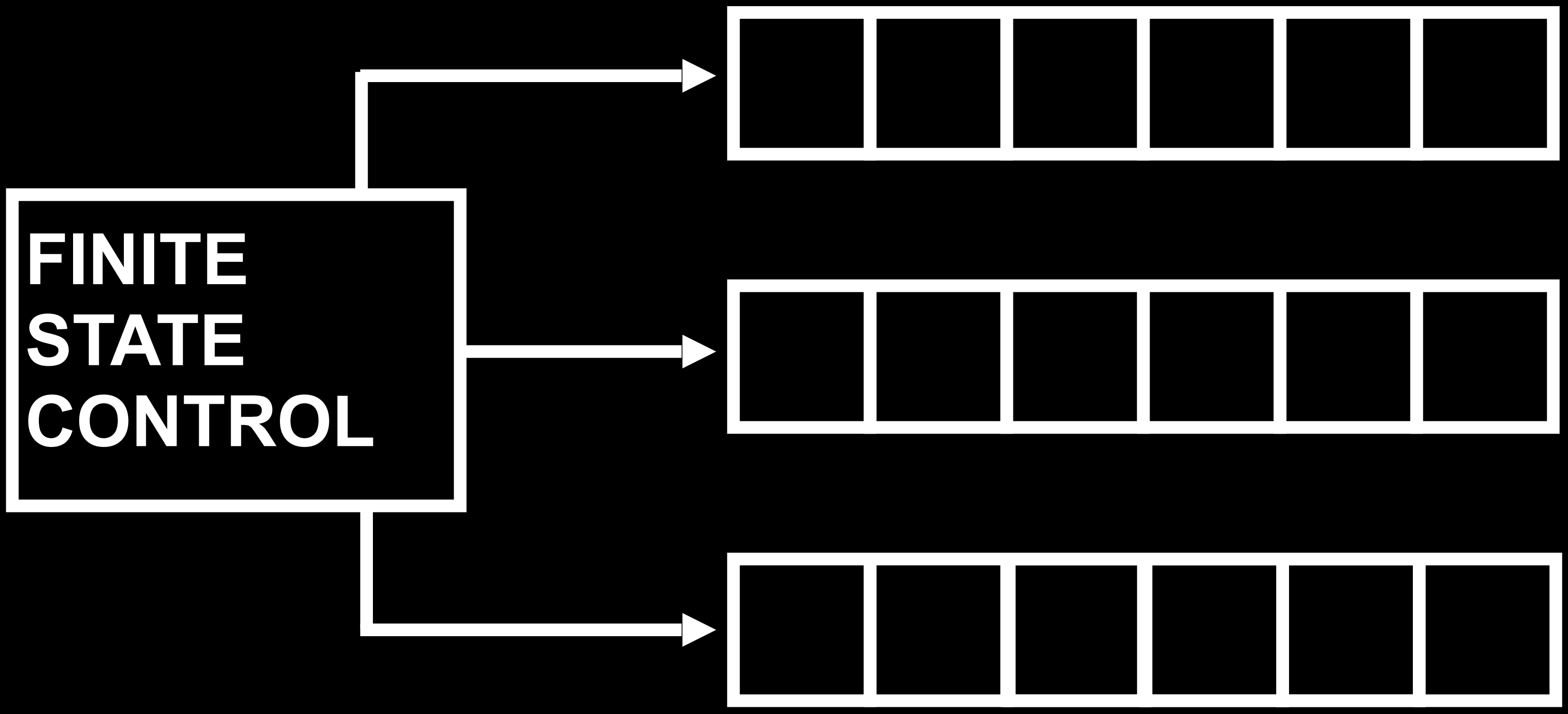
Claim: Simulating each step in the multi-tape machine uses at most O(t(n)) steps on a single-tape machine.
Hence total time of simulation is $O(t(n)^2)$.

MULTITAPE TURING MACHINES



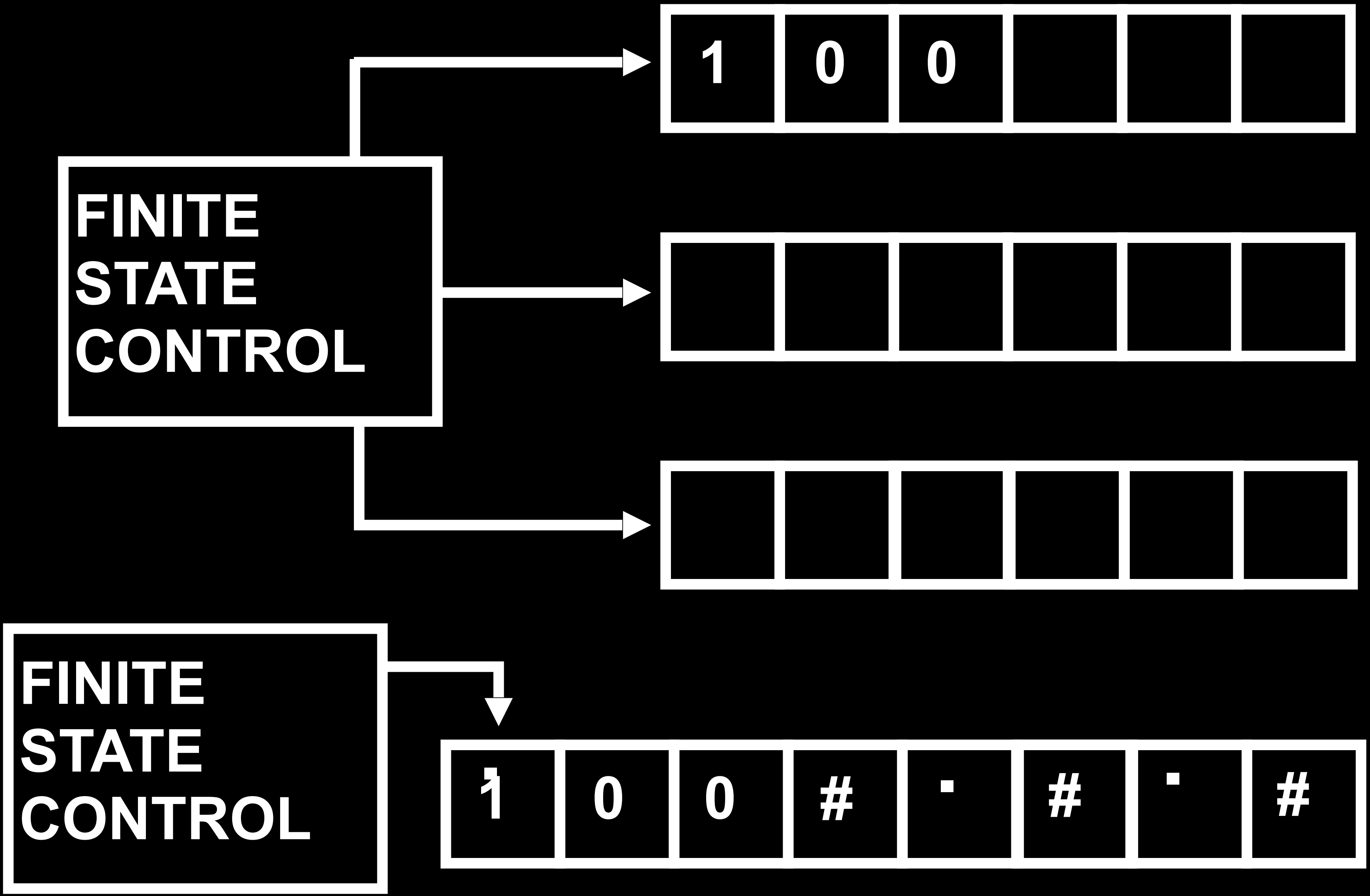
$$\delta : \mathbf{Q} \times \mathbf{\Gamma}^{\mathbf{k}} \to \mathbf{Q} \times \mathbf{\Gamma}^{\mathbf{k}} \times \{\mathbf{L},\mathbf{R}\}^{\mathbf{k}}$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

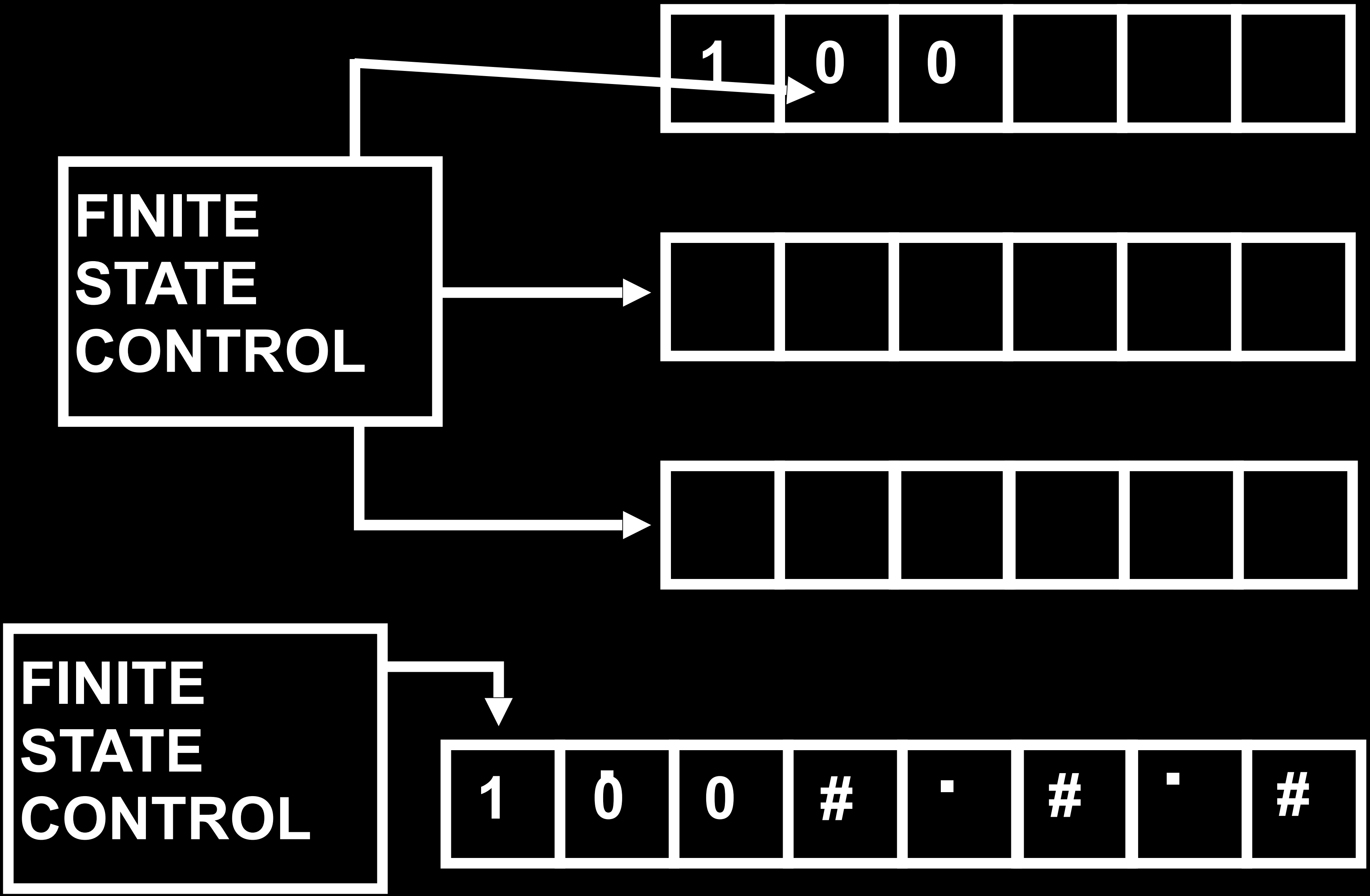


$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k$$

Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine



FINITE STATE CONTROL

FINITE STATE CONTROL

Analysis: (Note, k, the # of tapes, is fixed.)

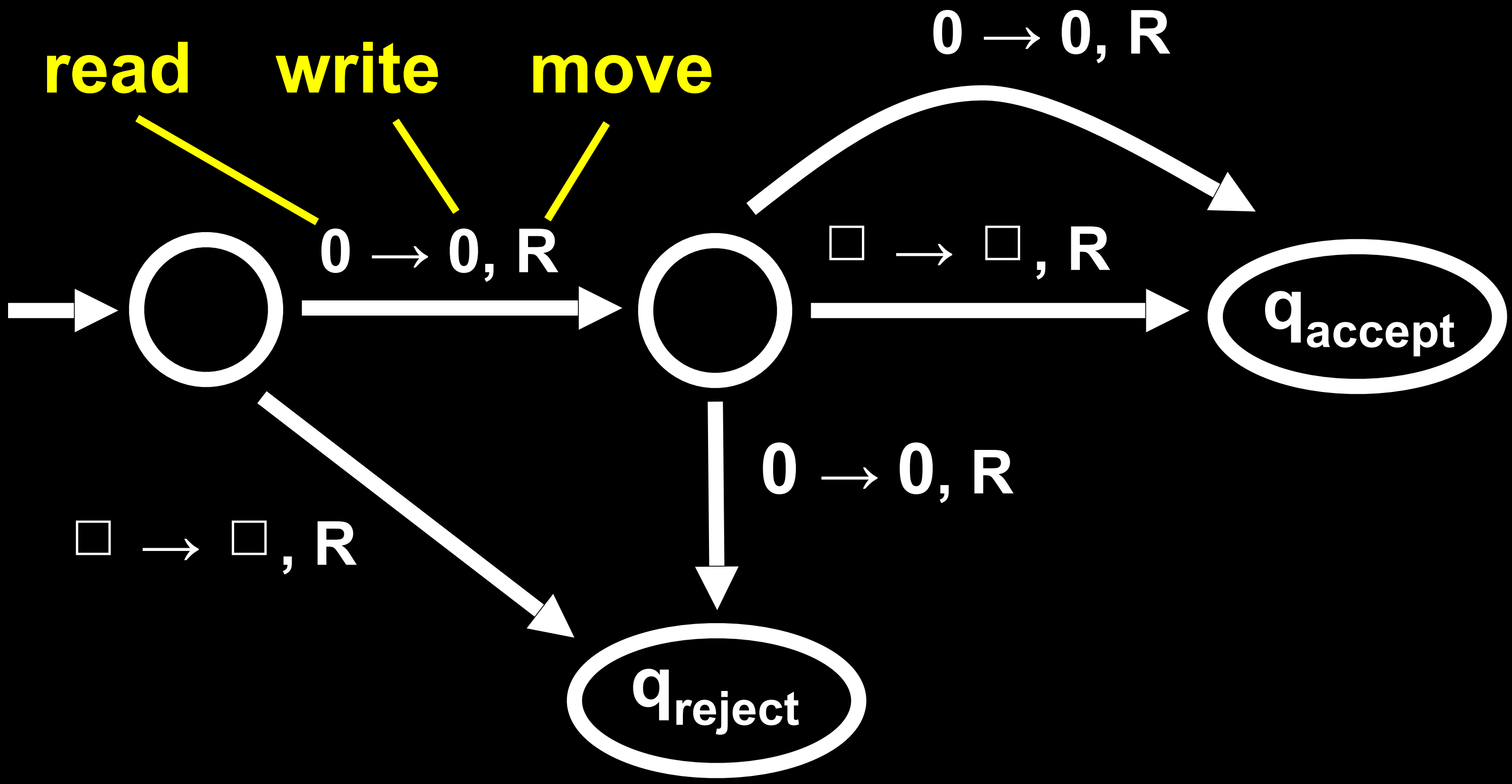
Let S be simulator

- Put S's tape in proper format: O(n) steps
- Two scans to simulate one step,

  1. to obtain info for next move O(t(n)) steps, why?
  2. to simulate it (may need to shift everything over to right possibly k times): O(t(n)) steps, why?

$$\mathbf{P} = \bigcup_{k \in \mathbf{N}} \mathbf{TIME}(n^k)$$

NON-DETERMINISTIC TURING MACHINES AND NP


read
write
move
0 → 0, R
0 → 0, R
□ → □, R
q_{accept}
0 → 0, R
□ → □, R
q_{reject}

Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

Q is a finite set of states

Σ is the input alphabet, where $\square \notin \Sigma$

Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$

$q_0 \in Q$ is the start state

$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

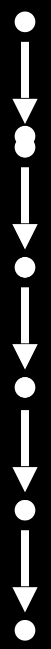
NON-DETERMINISTIC TMs

…are just like standard TMs, except:

1. The machine may proceed according to several possibilities

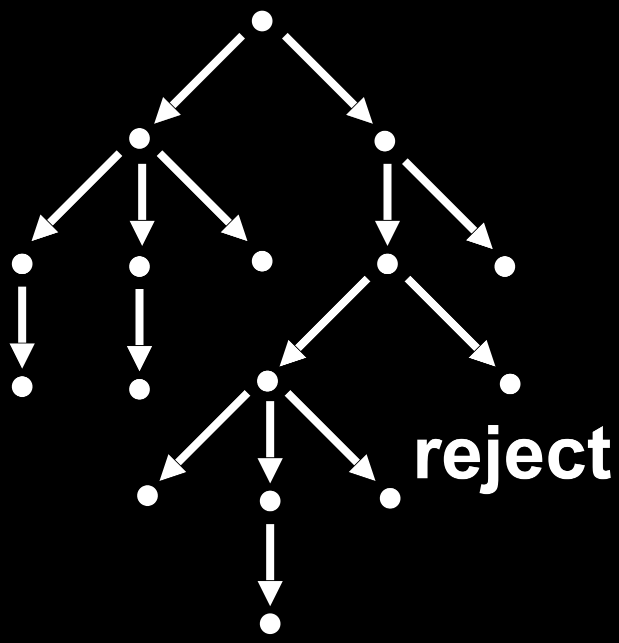
2. The machine accepts a string if there exists a path from start configuration to an accepting configuration

Deterministic Computation



accept or reject

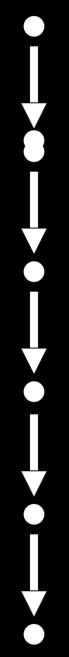
Non-Deterministic Computation



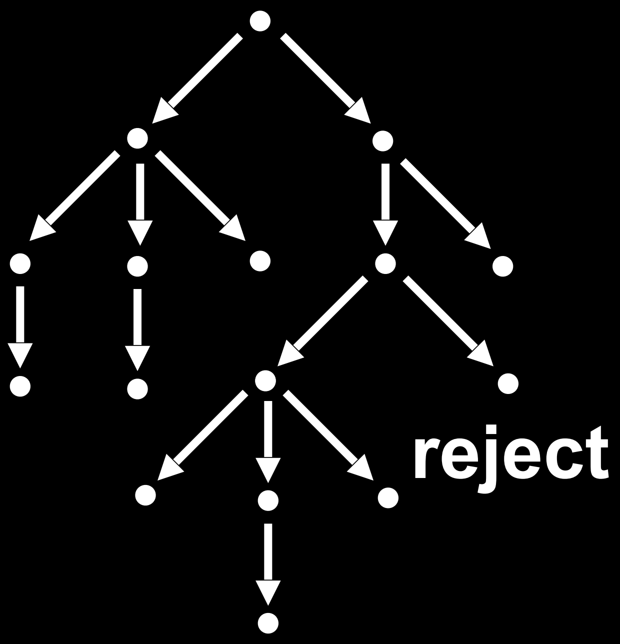
reject

accept

Deterministic Computation



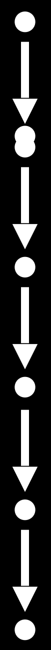
Non-Deterministic Computation



Definition*:* **Let M be a NTM that is a decider (Ie all branches halt on all inputs).**

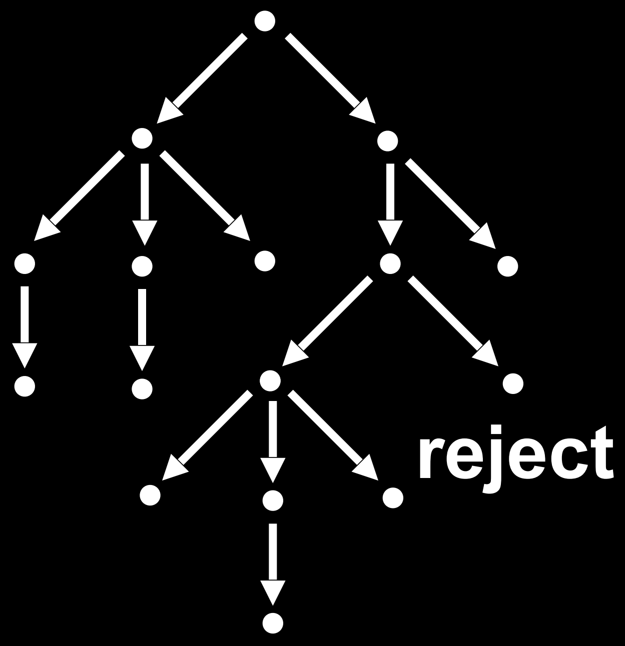
The running time or time-complexity of M is the function f : N → N, where f(n) is the maximum number of steps that M uses *on any branch of its computation on any input of length n.*

Deterministic Computation



accept or reject

Non-Deterministic Computation



reject

accept

Theorem: Let t(n) be a function such that t(n) ≥ n. Then every t(n)-time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ deterministic single tape TM

Definition: NTIME(t(n)) = { L | L is decided by a O(t(n))-time non-deterministic Turing machine }

$$\text{TIME}(t(n)) \subseteq \text{NTIME}(t(n))$$

BOOLEAN FORMULAS

logical operations

parentheses

A **satisfying assignment** is a setting of the variables that makes the formula true

$$\phi = (\neg x \land y) \lor z$$

x = 1, y = 1, z = 1 is a satisfying assignment for ϕ

variables

$$\neg(x \lor y) \land (z \land \neg x)$$

0 0 1 0

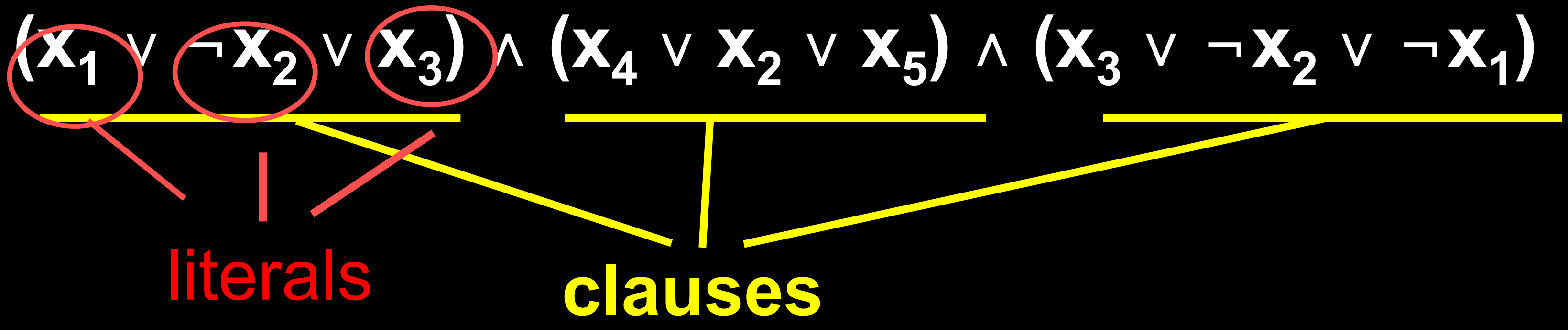
A Boolean formula is satisfiable if there exists a satisfying assignment for it

YES $a \land b \land c \land \neg d$

NO $\neg(x \lor y) \land x$

SAT = { ϕ | ϕ is a satisfiable Boolean formula }

A 3cnf-formula is of the form:



$(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)$

literals

clauses

YES $(x_1 \lor \neg x_2 \lor x_1)$

NO $(x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1)$

NO $(x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1)$

NO $(x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)$

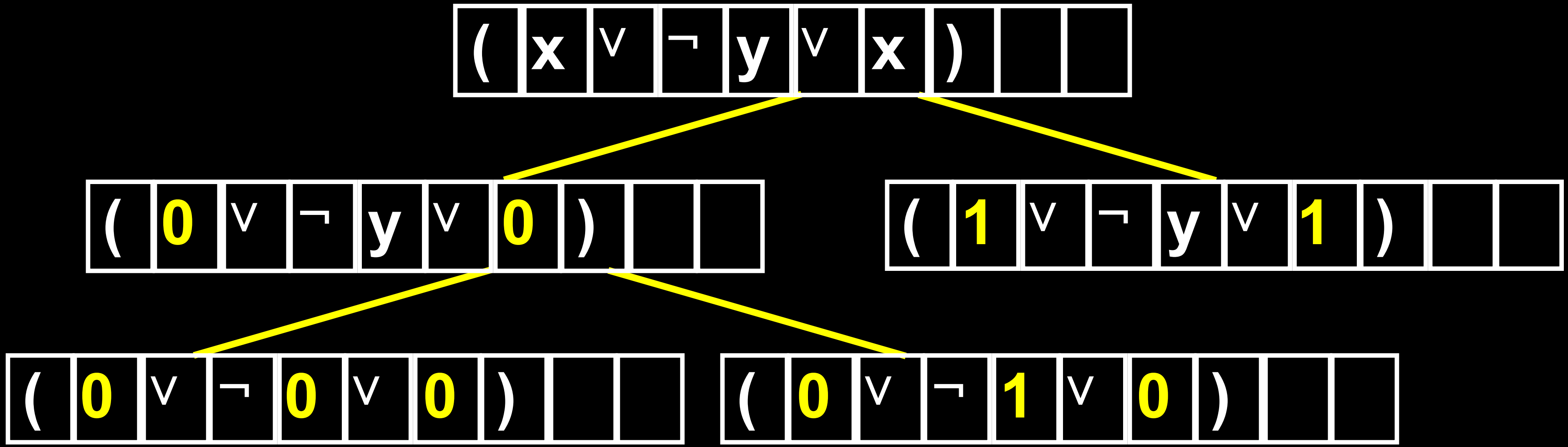
3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

Theorem: 3SAT $\in$ NTIME(n^2)

On input ϕ:

1. Check if the formula is in 3cnf
2. For each variable, non-deterministically substitute it with 0 or 1



(x ∨ ¬ y ∨ x)

(0 ∨ ¬ y ∨ 0)

(1 ∨ ¬ y ∨ 1)

(0 ∨ ¬ 0 ∨ 0)

(0 ∨ ¬ 1 ∨ 0)

3. Test if the assignment satisfies ϕ

$$\mathbf{NP} = \bigcup_{k \in \mathbf{N}} \mathbf{NTIME}(n^k)$$

Theorem: L $\in$ NP $\Leftrightarrow$ if there exists a poly-time Turing machine V(erifier) with

L = { x | $\exists$y(witness) |y| = poly(|x|) and V(x,y) accepts }

Proof:

(1) If L = { x | $\exists$y |y| = poly(|x|) and V(x,y) accepts }
then L $\in$ NP

Because we can guess y and then run V

(2) If L $\in$ NP then
L = { x | $\exists$y |y| = poly(|x|) and V(x,y) accepts }

Let N be a non-deterministic poly-time TM that decides L and define V(x,y) to accept if y is an accepting computation history of N on x

3SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ and ϕ is in 3cnf }

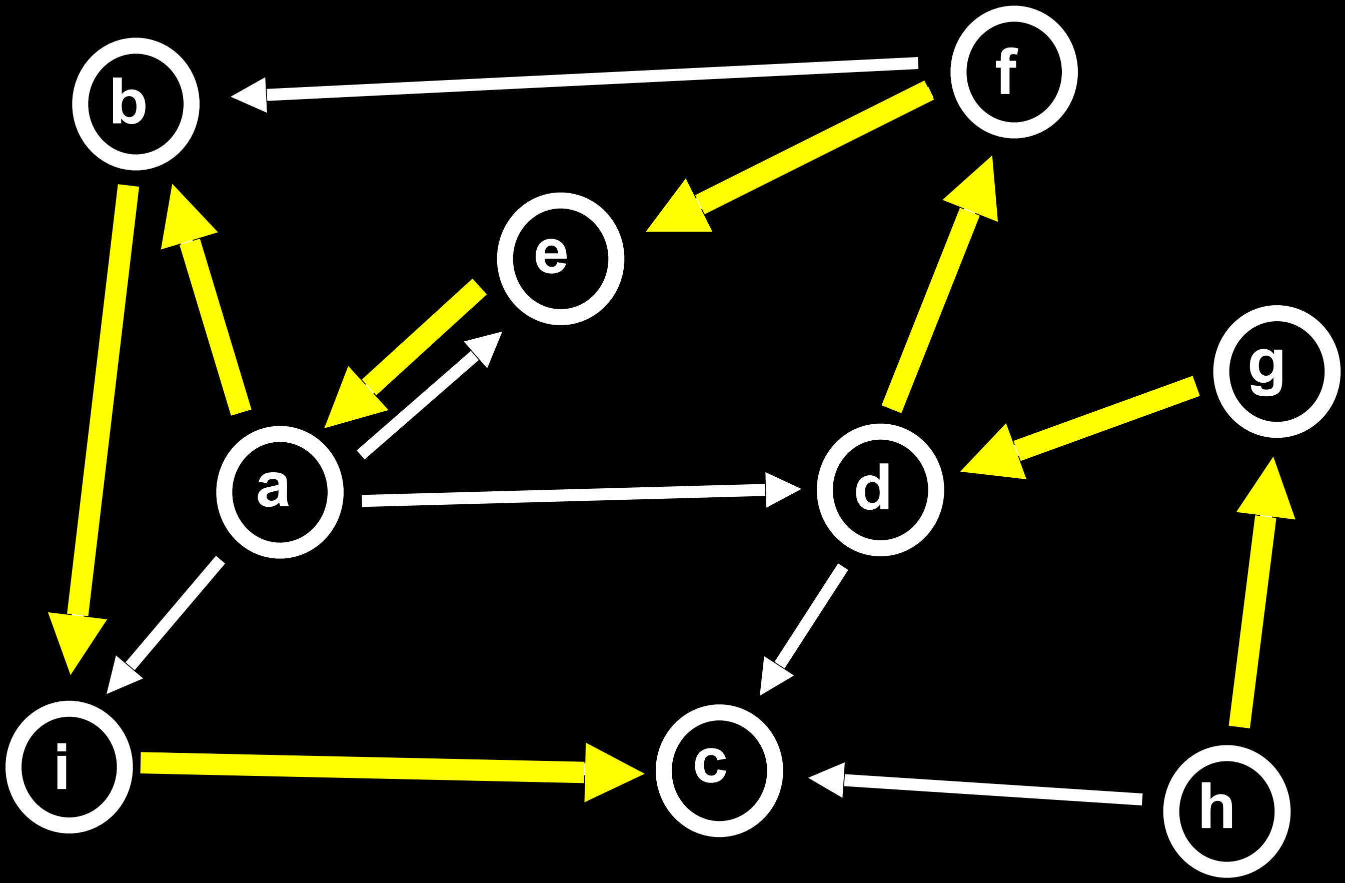
SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ }

A language is in NP if and only if there exist polynomial-length certificates* for membership to the language

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable

* that can be verified in poly-time

HAMILTONIAN PATHS

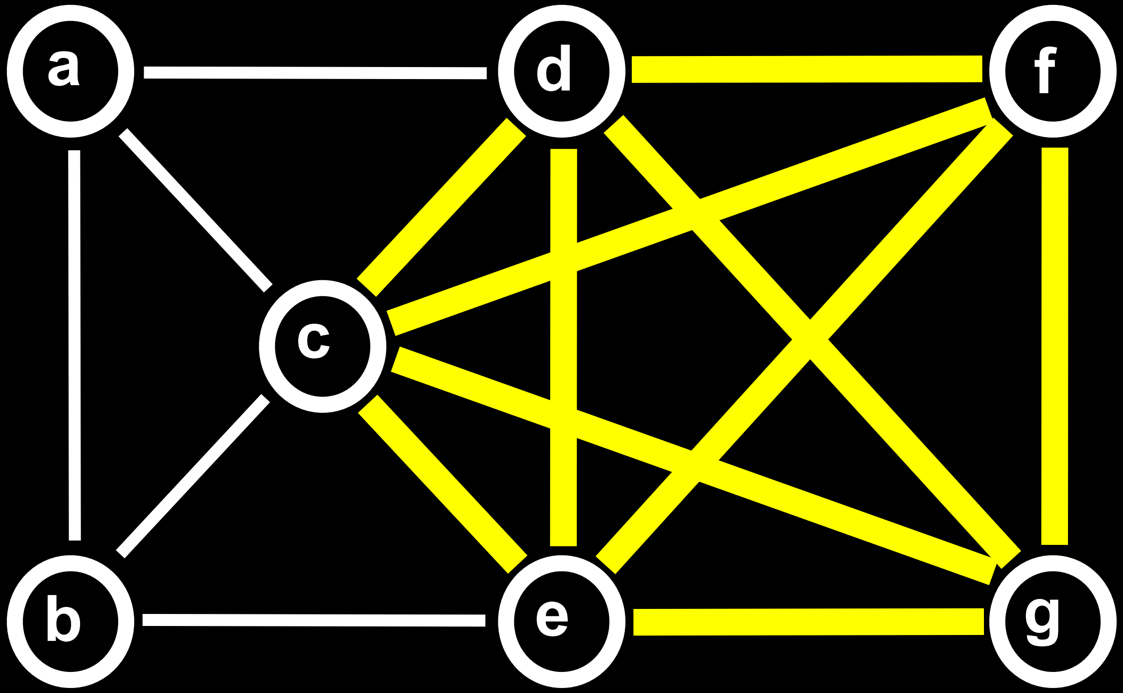


HAMPATH = { (G,s,t) | G is a directed graph with a Hamiltonian path from s to t }

Theorem: HAMPATH $\in$ NP

The Hamilton path itself is a certificate

K-**CLIQUES**



CLIQUE = { (G,k) | G is an undirected graph with a k-clique }

Theorem: CLIQUE $\in$ NP

The k-clique itself is a certificate

NP = all the problems for which once you have the answer it is easy (i.e. efficient) to verify

$$$

P = NP?

$$$

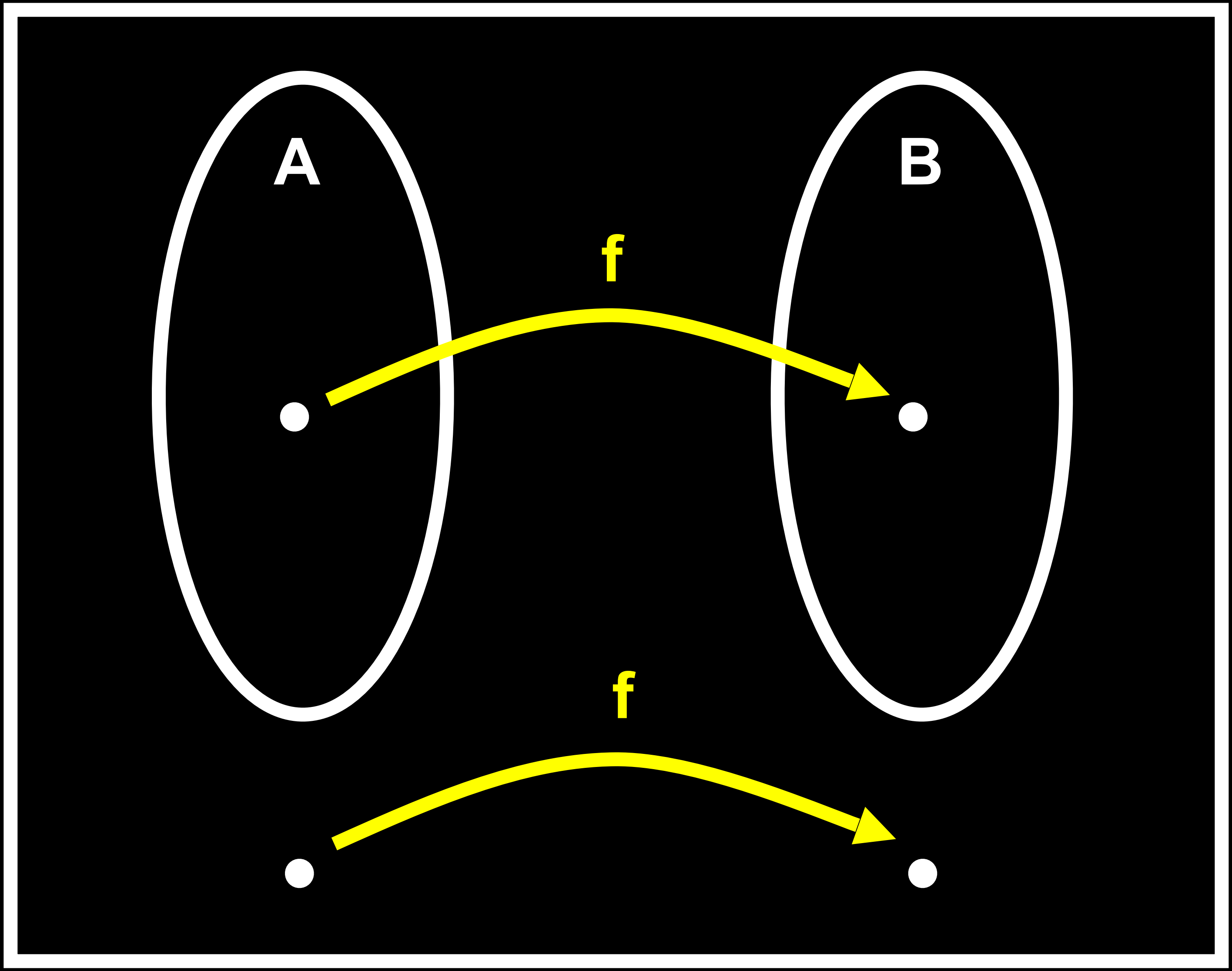
POLY-TIME REDUCIBILITY

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

Language A is polynomial time reducible to language B, written $A \leq_P B$, if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

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

f is called a polynomial time reduction of A to B


A
B
f
f

Theorem: If $A \leq_P B$ and $B \in P$, then $A \in P$

Proof: Let M_B be a poly-time (deterministic) TM that decides B and let f be a poly-time reduction from A to B

We build a machine M_A that decides A as follows:

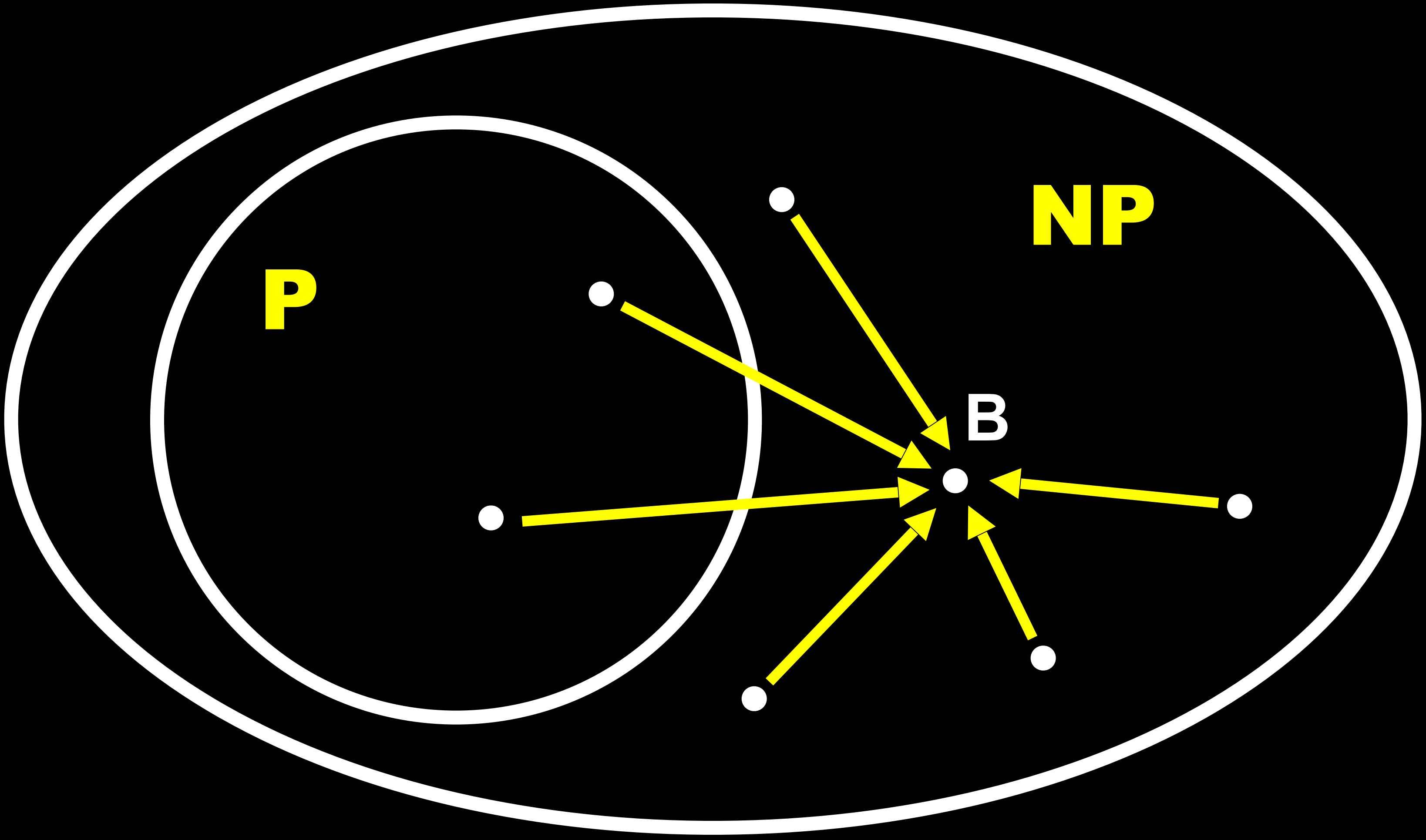
On input w:

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

Definition: A language B is NP-complete if:

1. B $\in$ NP
2. Every A in NP is poly-time reducible to B (i.e. B is NP-hard)

Suppose B is NP-Complete



So, if B is NP-Complete and B ∈ P then NP = P. **Why?**

Theorem (Cook-Levin): SAT is NP-complete

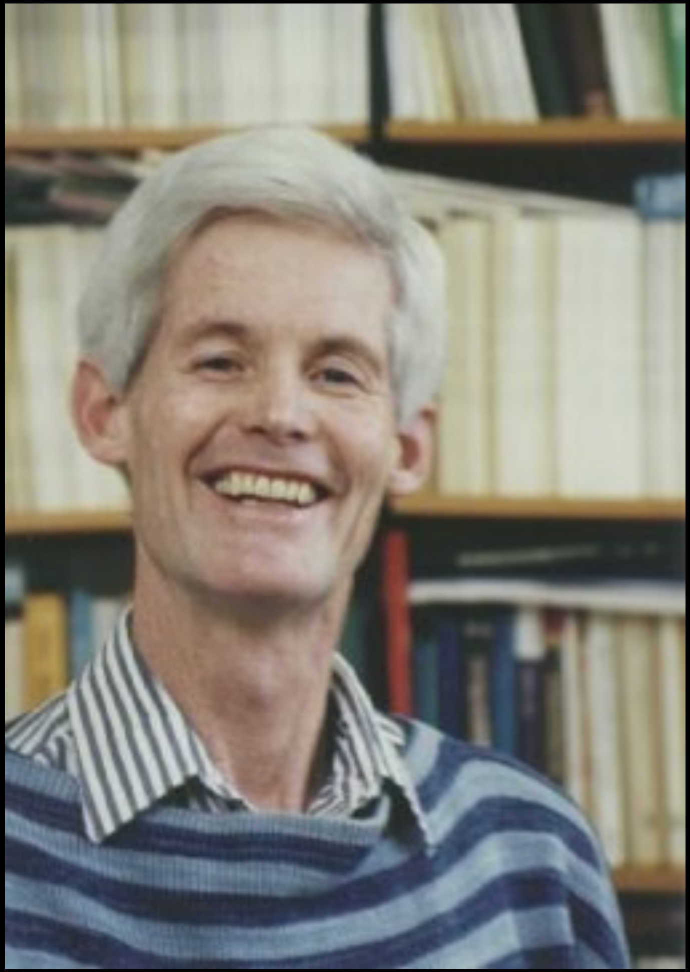
Corollary: SAT $\in$ P if and only if P = NP

NP-COMPLETENESS:
THE COOK-LEVIN THEOREM

Theorem (Cook-Levin.'71): SAT is NP-complete

Corollary: SAT $\in$ P if and only if P = NP

Leonid Levin



Steve Cook

Theorem (Cook-Levin): SAT is NP-complete

Proof:

(1) SAT $\in$ NP

(2) Every language A in NP is polynomial time reducible to SAT

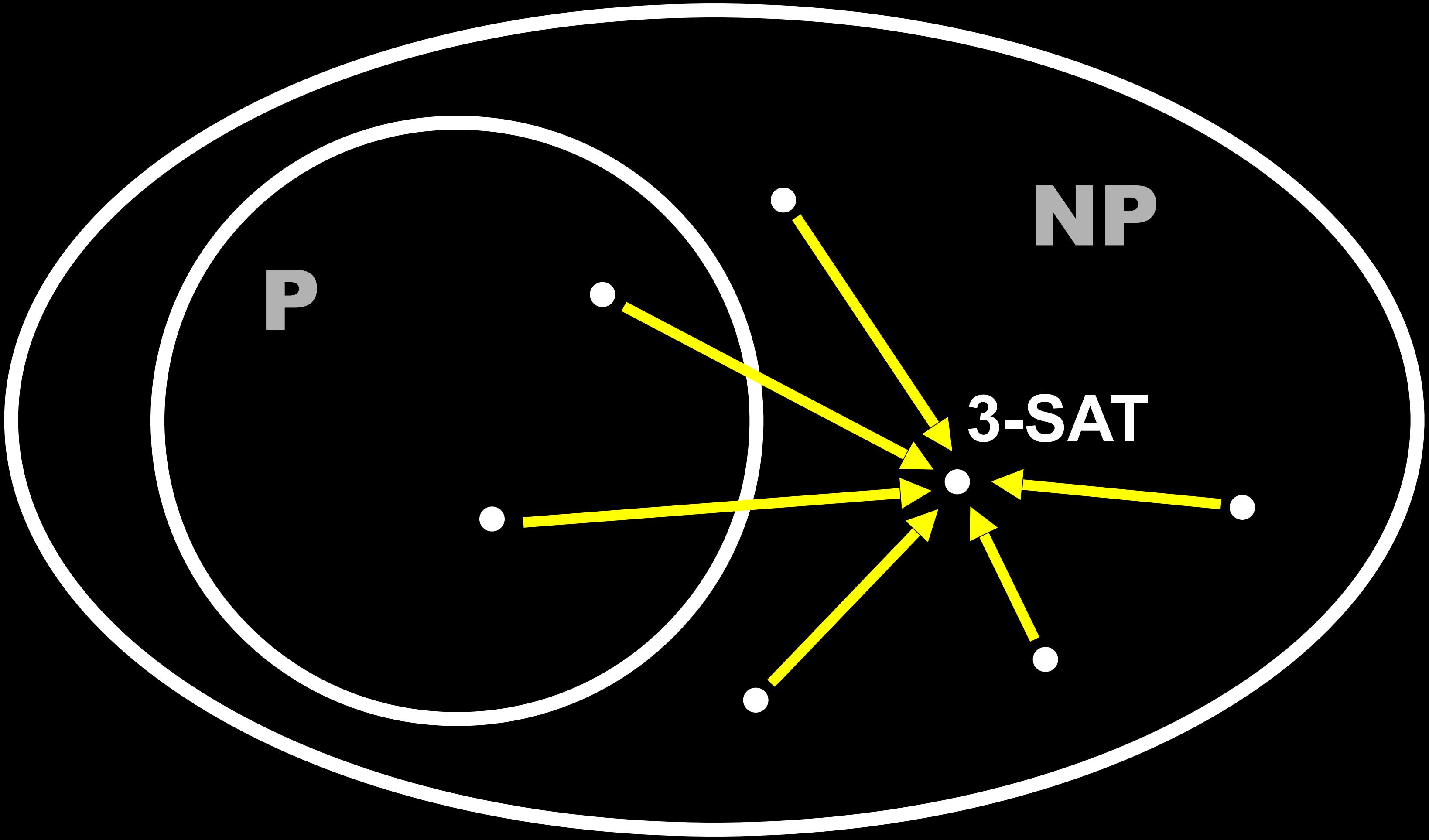
We build a poly-time reduction from A to SAT

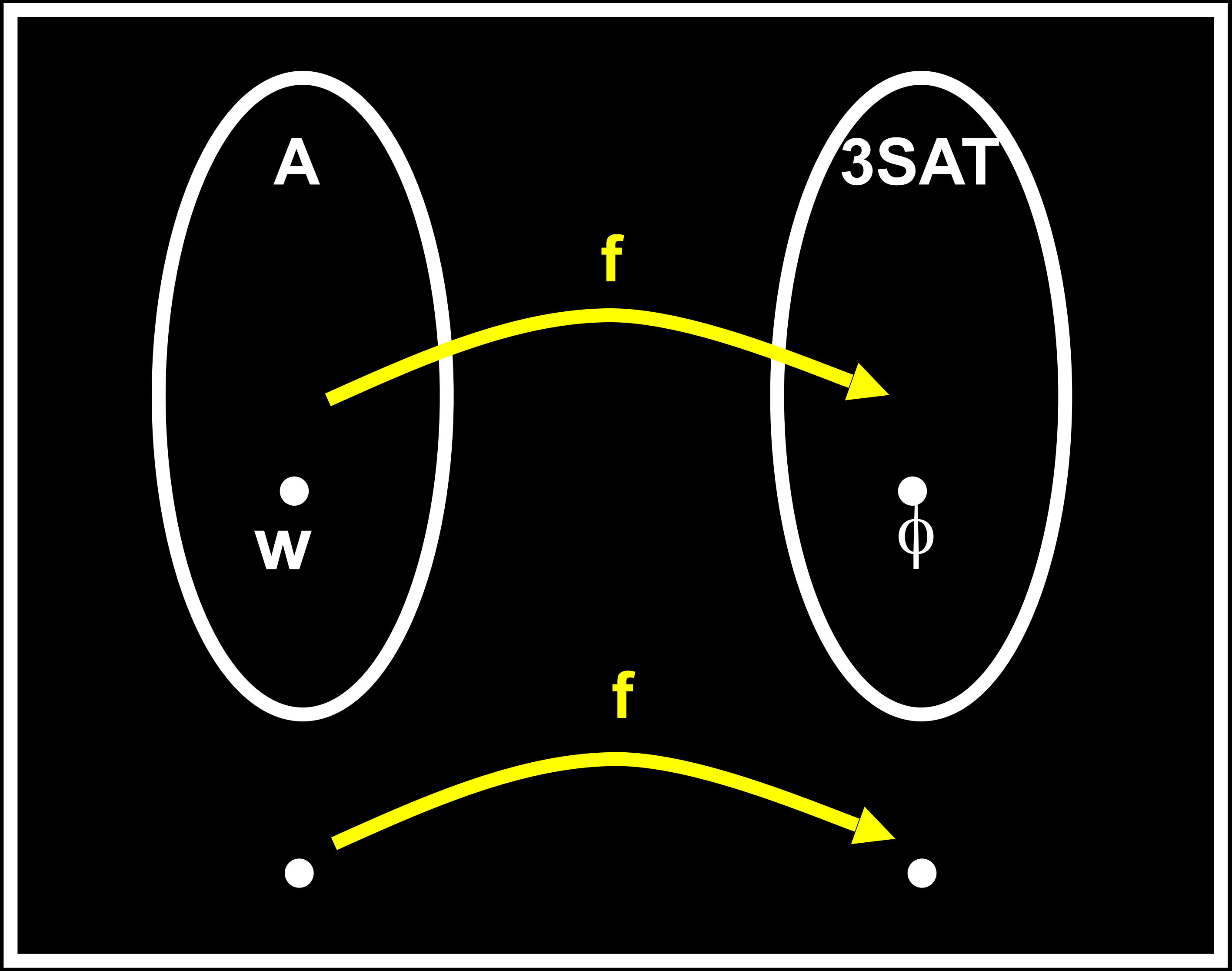
The reduction turns a string w into a 3-cnf formula ϕ such that w $\in$ A iff $\phi \in$ 3-SAT.

ϕ will *simulate* the NP machine N for A on w.

Let N be a non-deterministic TM that decides A in time n^k How do we know N exists?

So proof will also show:
3-SAT is NP-Complete



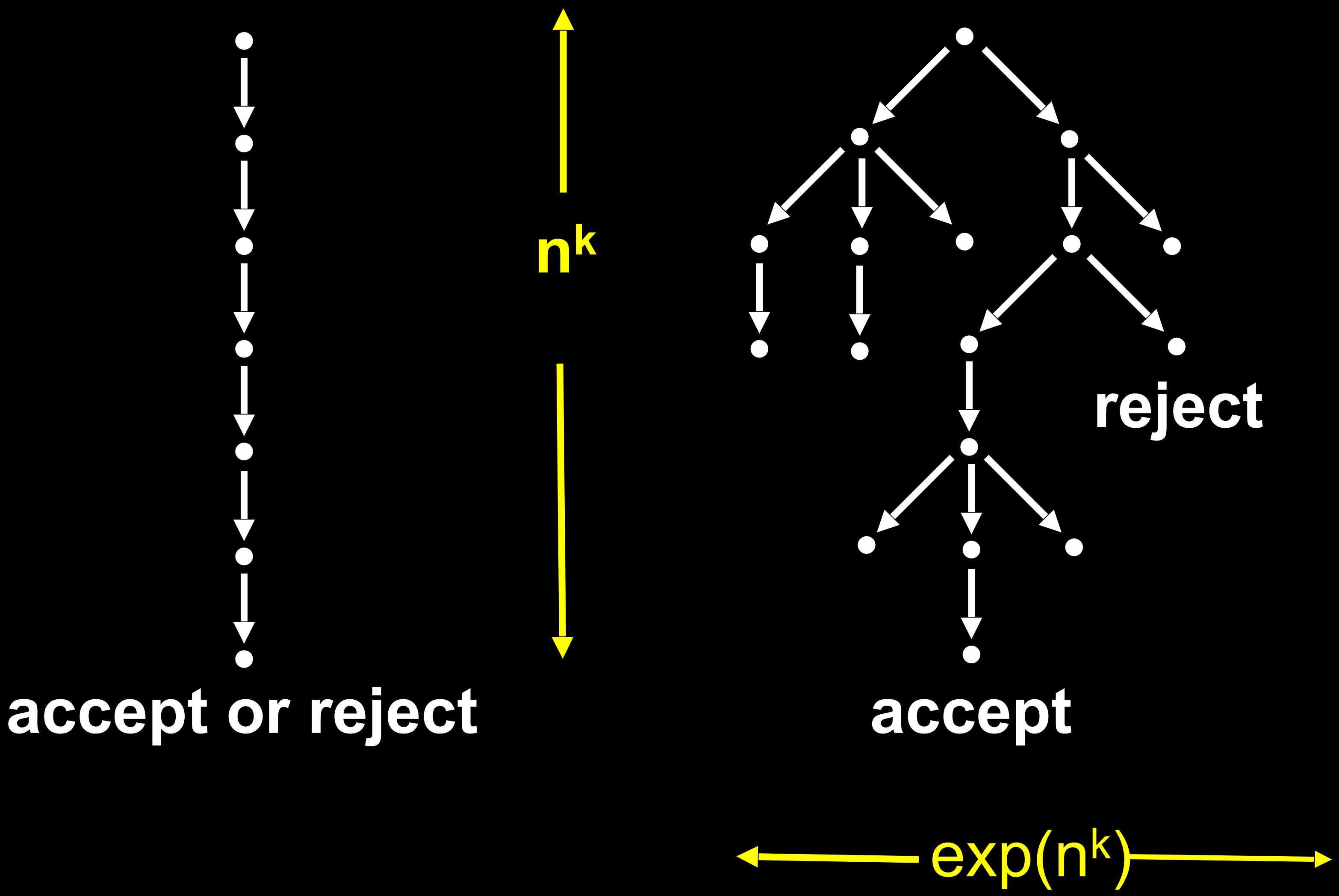


The reduction **f** turns a string **w** into a 3-cnf formula ϕ such that: **w** $\in$ A $\Leftrightarrow$ $\phi \in$ 3SAT.

ϕ will “simulate” the NP machine N for A on w.

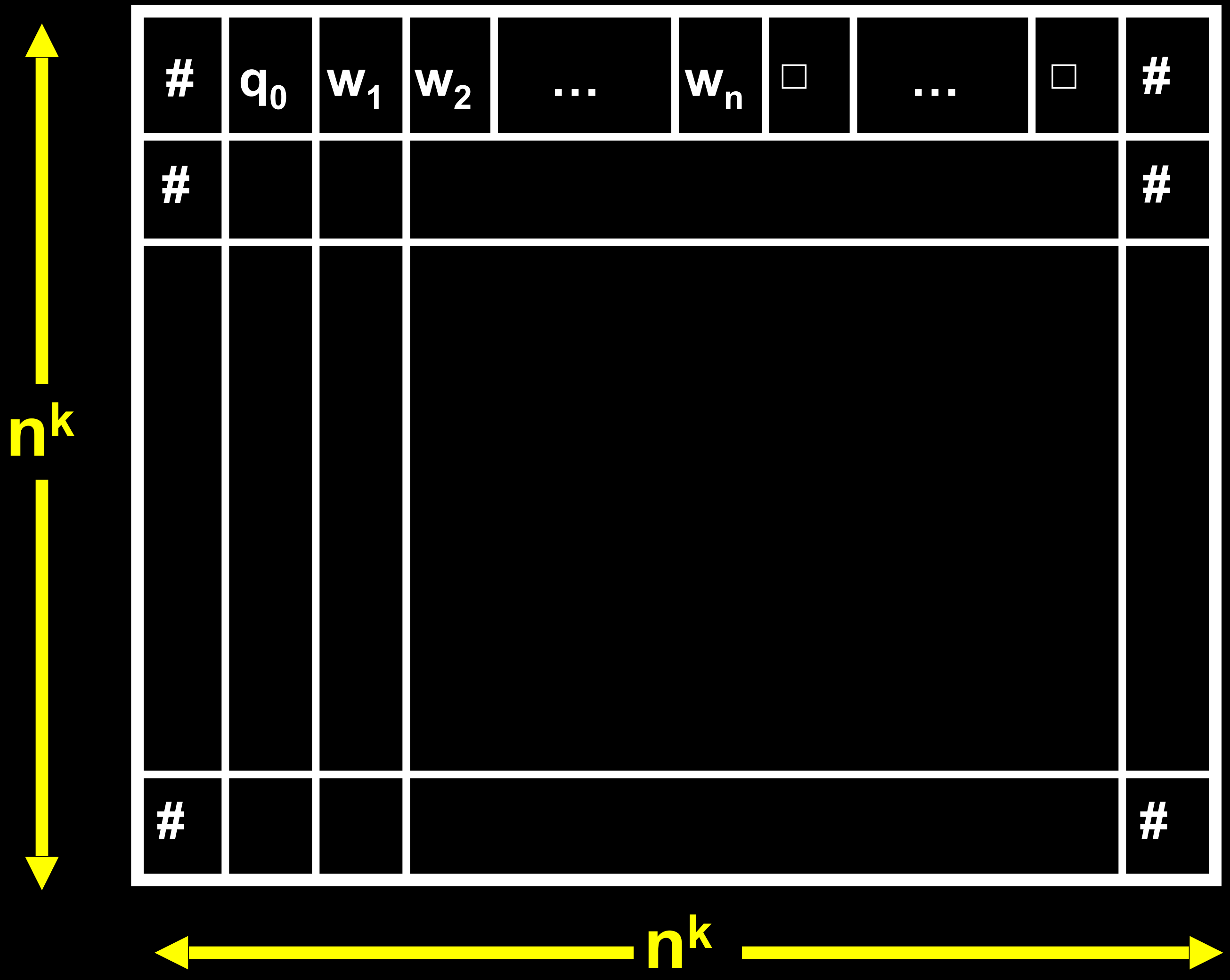
Deterministic Computation

Non-Deterministic Computation



Suppose A ∈ NTIME(n^k) and let N be an NP machine for A.

A tableau for N on w is an $n^k \times n^k$ table whose rows are the configurations of *some* possible computation of N on input w.



| q_0 | w_1 | w_2 | … | w_n | □ | … | □ |

# #

# #

A tableau is accepting if any row of the tableau is an accepting configuration

Determining whether N accepts w is equivalent to determining whether there is an accepting tableau for N on w

Given w, our 3cnf-formula ϕ will describe a *generic* tableau for N on w (in fact, essentially *generic* for N on any string w of length n).

The 3cnf formula ϕ will be satisfiable *if and only if* there is an accepting tableau for N on w.

VARIABLES of ϕ

Let $C = Q \cup \Gamma \cup \{ \# \}$

Each of the $(n^k)^2$ entries of a tableau is a cell

cell[i,j] = the cell at row i and column j

For each i and j ($1 \leq i, j \leq n^k$) and for each $s \in C$ we have a variable $x_{i,j,s}$

# variables = $|C|n^{2k}$, ie $O(n^{2k})$, since $|C|$ only depends on N

These are the variables of ϕ and represent the contents of the cells

We will have: $x_{i,j,s} = 1 \Leftrightarrow$ cell[i,j] = s

$$x_{i,j,s} = 1$$

means

cell[i, j] = s

We now design ϕ so that a satisfying assignment to the variables $x_{i,j,s}$ corresponds to an accepting tableau for N on w

The formula ϕ will be the AND of four parts:

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$$

ϕ_{cell} ensures that for each i,j, exactly one $x_{i,j,s} = 1$

ϕ_{start} ensures that the first row of the table is the *starting (initial)* **configuration of N on w**

ϕ_{accept} ensures* that an accepting configuration occurs somewhere in the table

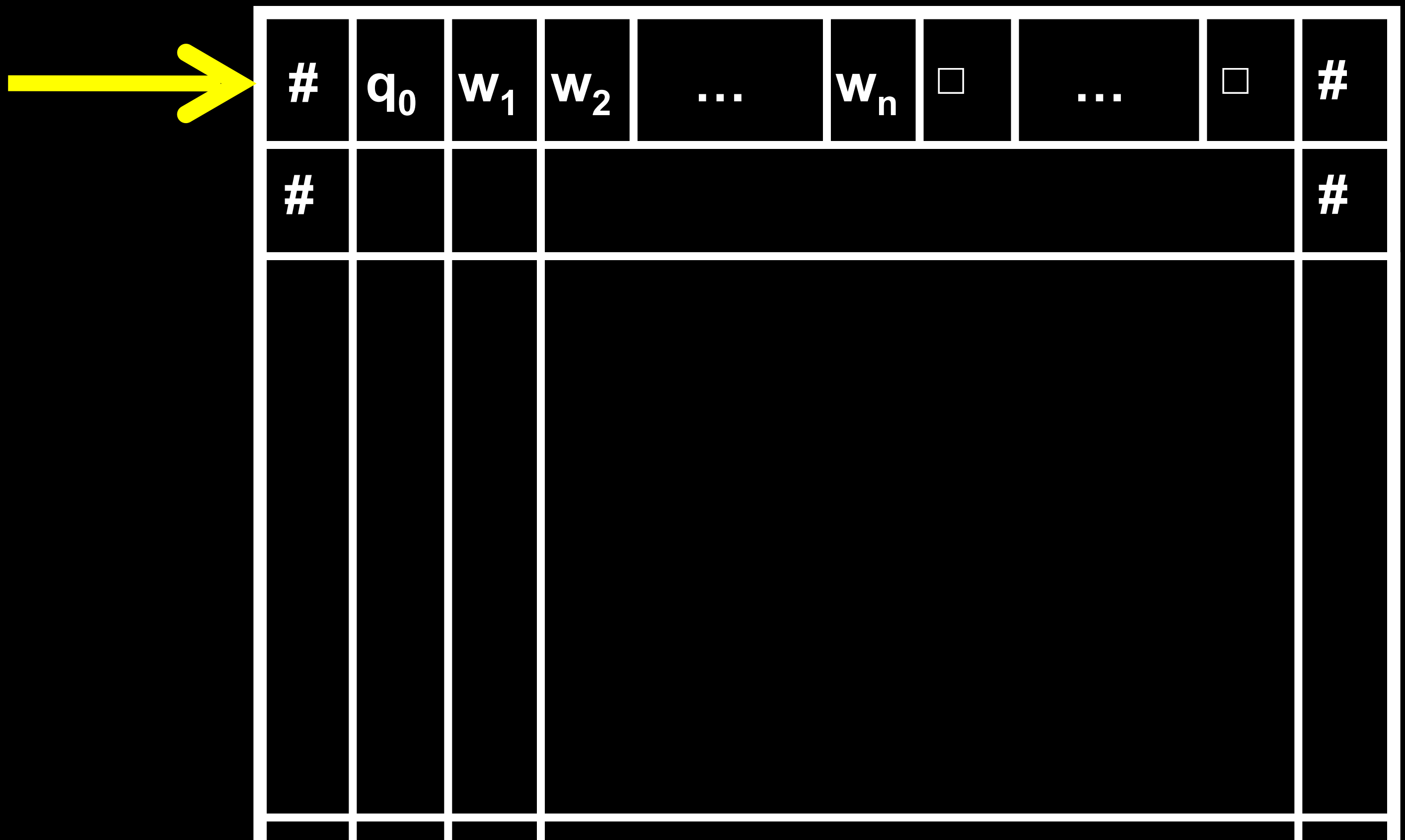
ϕ_{move} ensures* that every row is a configuration that legally follows from the previous config

*if the other components of ϕ **hold**

ϕ_{cell} ensures that for each i,j, exactly one $x_{i,j,s} = 1$

$$\phi_{cell} = \bigwedge_{1 \leq i,\, j \leq n^k} \left[\underbrace{\left(\bigvee_{s \in C} x_{i,j,s} \right)}_{\text{at least one variable is turned on}} \wedge \underbrace{\left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right)}_{\text{at most one variable is turned on}} \right]$$

$$\begin{aligned}\phi_{\text{start}} = \;& x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\Box} \wedge \dots \wedge x_{1,n^k-1,\Box} \wedge x_{1,n^k,\#}\end{aligned}$$



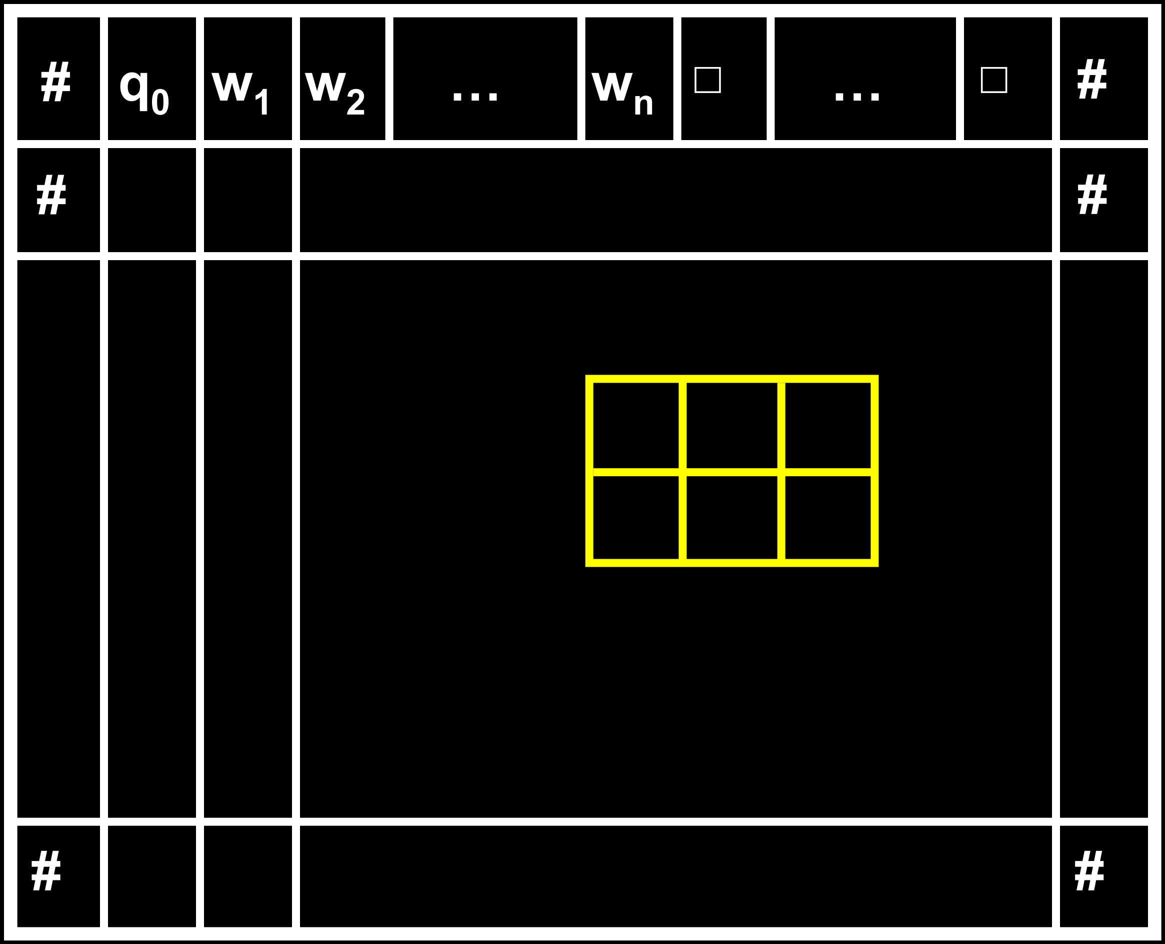
#	q_0	w_1	w_2	…	w_n	□	…	□	#
#									#

ϕ_{accept} ensures that an accepting configuration occurs somewhere in the table

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,\, j \leq n^k} x_{i,j,q_{\text{accept}}}$$

ϕ_{move} ensures that every row is a configuration that legally follows from the previous

It works by ensuring that each 2 × 3 “window” of cells is legal (does not violate N’s rules)



If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:

a	q_1	b
q_2	a	c

a	q_1	b
q_1	a	a

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	q_1	b
q_2	b	$_2$

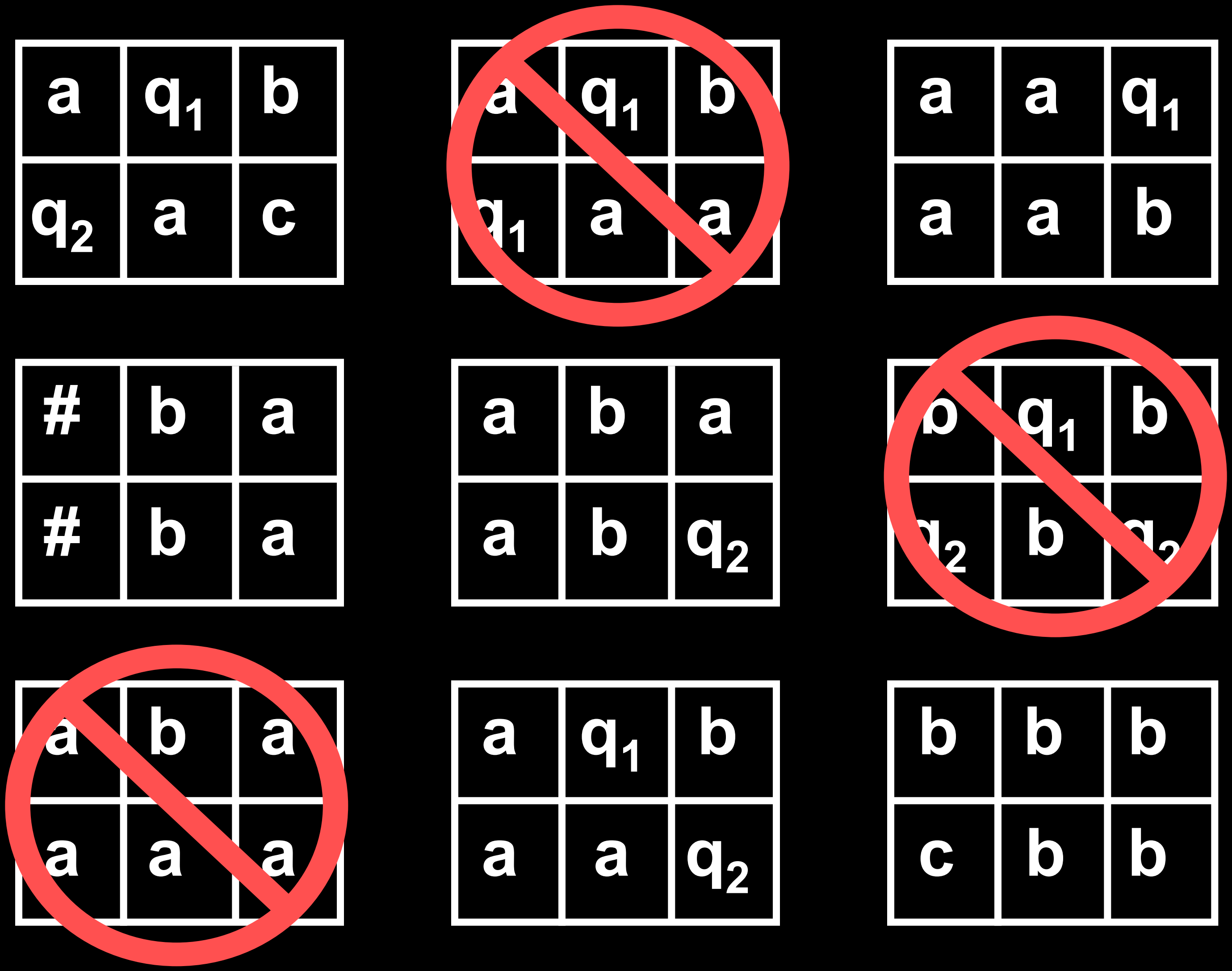
a	b	a
a	a	a

a	q_1	b
a	a	q_2

b	b	b
c	b	b

If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:



a	q_1	b
q_2	a	c

~~a~~	~~q_1~~	~~b~~
~~q_1~~	~~a~~	~~a~~

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

~~b~~	~~q_1~~	~~b~~
~~q_2~~	~~b~~	~~q_2~~

~~a~~	~~b~~	~~a~~
~~a~~	~~a~~	~~a~~

a	q_1	b
a	a	q_2

b	b	b
c	b	b

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	s	

Proof:

In upper configuration, every cell that doesn't contain the boundary symbol #, is the center top cell of a window.

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	a	
	a	

Proof:

In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol

CLAIM:

If

- the top row of the tableau is the start configuration, and
- and every window is legal,

Then

each row of the tableau is a configuration that legally follows the preceding one.

	a	
	a	

	q	
ok	ok	ok

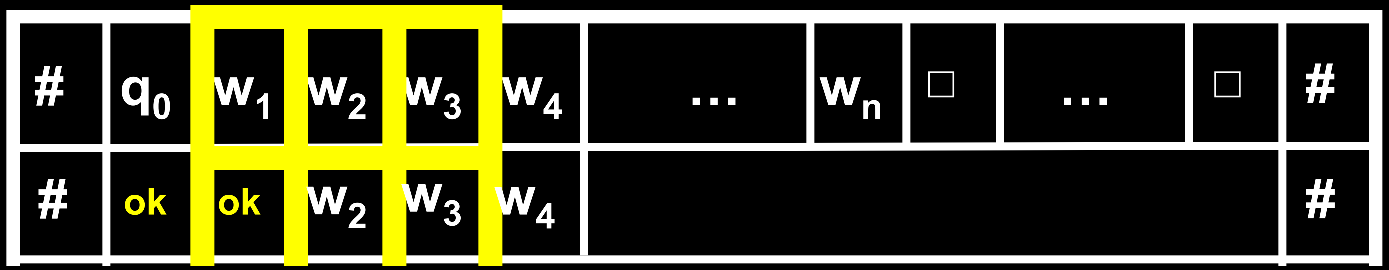
Proof:

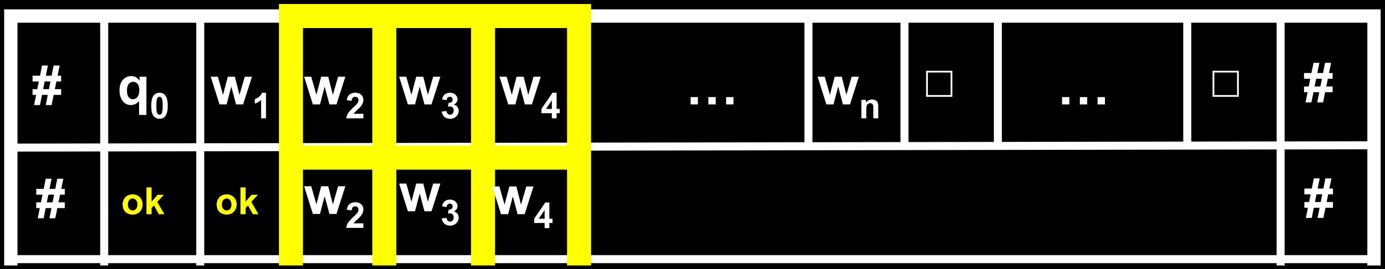
In upper configuration, every cell that doesn't contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol

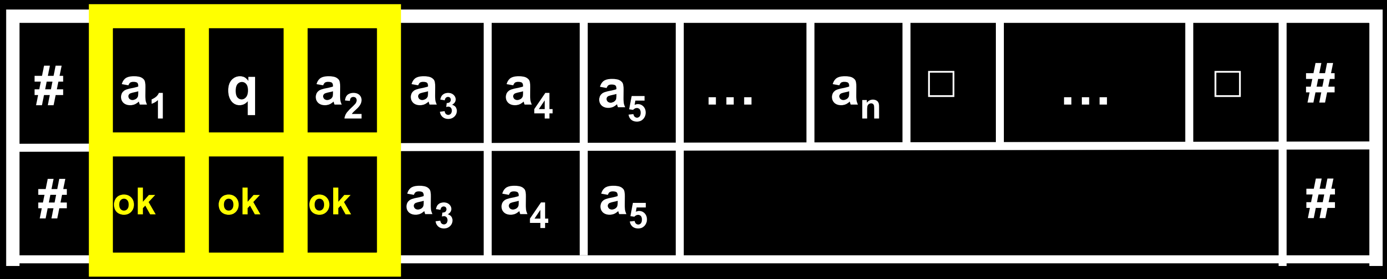
Case 2. center cell of window is a state symbol

#	q_0	w_1	w_2	w_3	w_4	…	w_n	□	…	□	#
#	ok	ok	w_2	w_3	w_4						#

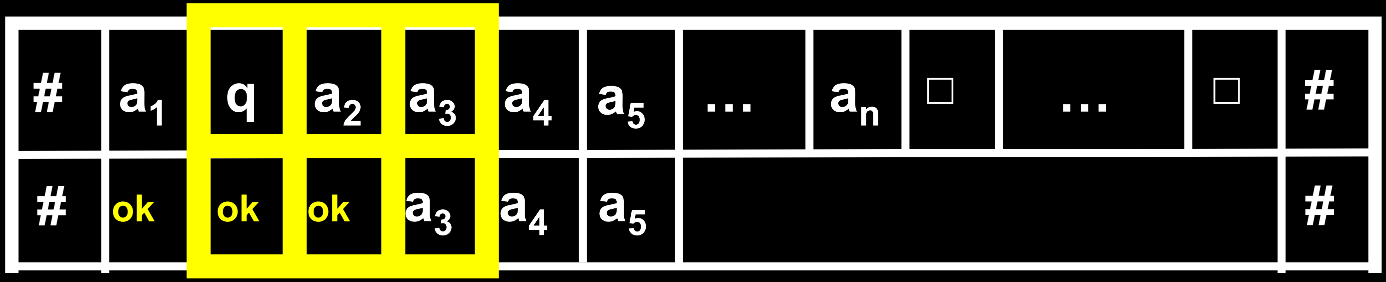




#	q_0	w_1	w_2	w_3	w_4	…	w_n	□	…	□	#
#	ok	ok	w_2	w_3	w_4						#



#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#



#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#

#	a_1	q	a_2	a_3	a_4	a_5	…	a_n	□	…	□	#
#	ok	ok	ok	a_3	a_4	a_5						#

So the lower configuration follows from the upper!!!

	col. j-1	col. j	col. j+1
row i	(i,j-1) a_1	**(i,j)** a_2	(i,j+1) a_3
row i+1	(i+1,j-1) a_4	(i+1,j) a_5	(i+1,j+ 1) a_6

The (i,j) Window

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the (i, j) window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all ($\le |C|^6$) legal sequences ($a_1, \ldots, a_6$).

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the (i, j) window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all ($\le |C|^6$) legal sequences ($a_1, \ldots, a_6$).

This disjunct is satisfiable

⟺

There is *some* assignment to the cells (ie variables) in the window (i,j) that makes the window legal

$$\phi_{move} = \bigwedge_{1 \le i, j \le n^k} (\text{ the (i, j) window is legal })$$

the (i, j) window is legal =

$$\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

This is a disjunct over all (≤ $|C|^6$) legal sequences ($a_1, \ldots, a_6$).

So ϕ_{move} is satisfiable

⇔

There is *some* assignment to each of the variables that makes *every* window legal.

$$\phi_{move} = \bigwedge_{1 \leq i,\, j \leq n^k} (\text{ the } (i, j) \text{ window is legal })$$

the (i, j) window is legal =

$$\bigvee_{a_1, \ldots, a_6} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j,+1,a} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a})$$

is a legal window

This is a disjunct over all (≤ $|C|^6$) legal sequences ($a_1, \ldots, a_6$).

Can re-write as equivalent conjunct:

$$\equiv \bigwedge_{a_1, \ldots, a_6} (\square_{i,j-1,a_1} \vee \square_{i,j,a_2} \vee \square_{i,j,+1,a} \vee \square_{i+1,j-1,a_4} \vee \square_{i+1,j,a} \vee \square_{6\, +1,j+1,a})$$

ISN'T a legal window

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$$

ϕ is satisfiable (ie, there is some assignment to each of the varialbes s.t. ϕ evaluates to 1)

⇔

there is some assignment to each of the variables s.t. ϕ_{cell} and ϕ_{start} and ϕ_{accept} and ϕ_{move} each evaluates to 1

⇔

There is some assignment of symbols to cells in the tableau such that:

- The first row of the tableau is a start configuration and
- Every row of the tableau is a configuration that follows from the preceding by the rules of N and
- One row is an accepting configuration

⇔

There is some accepting computation for N with input w

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs

We just need to make those ORs with 3 literals

If a clause has less than three variables:

$a \equiv (a \vee a \vee a)$, $(a \vee b) \equiv (a \vee b \vee b)$

3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs

We just need to make those ORs with 3 literals

If a clause has less than three variables:

$a \equiv (a \vee a \vee a)$, $(a \vee b) \equiv (a \vee b \vee b)$

If a clause has more than three variables:

$(a \vee b \vee c \vee d) \equiv (a \vee b \vee z) \wedge (\neg z \vee c \vee d)$

$(a_1 \vee a_2 \vee \ldots \vee a_t) \equiv$

$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \ldots$

$$\phi = \phi_{\mathbf{cell}} \wedge \phi_{\mathbf{start}} \wedge \phi_{\mathbf{accept}} \wedge \phi_{\mathbf{move}}$$

WHAT'S THE **LENGTH OF ϕ?**

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,\, j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right]$$

If a clause has less than three variables:

$(a \vee b) = (a \vee b \vee b)$

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i,\, j \leq n^k} \Bigg[\Bigg(\bigvee_{s \in C} x_{i,j,s} \Bigg) \wedge \Bigg(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \Bigg) \Bigg]$$

$O(n^{2k})$ clauses

$$\text{Length}(\phi_{\text{cell}}) = O(n^{2k})\ \underbrace{O(\log (n))}_{\text{length(indices)}} = O(n^{2k} \log n)$$

$$
\begin{aligned}
\phi_{\mathrm{start}} = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\
& x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\
& x_{1,n+3,\square} \wedge \ldots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#} \\
= \quad & (x_{1,1,\#} \vee x_{1,1,\#} \vee x_{1,1,\#}) \wedge \\
& (x_{1,2,q_0} \vee x_{1,2,q_0} \vee x_{1,2,q_0}) \\
& \wedge \ldots \wedge \\
& (x_{1,n^k,\#} \vee x_{1,n^k,\#} \vee x_{1,n^k,\#})
\end{aligned}
$$

$$\begin{aligned}\phi_{start} = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \cdots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\square} \wedge \cdots \wedge x_{1,n^k-1,\square} \wedge x_{1,n^k,\#}\end{aligned}$$

$$O(n^k)$$

$$\phi_{\text{accept}} = \bigvee_{1 \le i, j \le n^k} x_{i,j,q_{\text{accept}}}$$

$$(a_1 \vee a_2 \vee \ldots \vee a_t) =$$

$$(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge (\neg z_2 \vee a_4 \vee z_3) \ldots$$

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,\, j \leq n^k} x_{i,j,q_{\text{accept}}}$$

$$O(n^{2k})$$

$$\phi_{move} = \bigwedge_{1 \le i,\, j \le n^k} (\text{ the (i, j) window is legal })$$

the (i, j) window is legal =

$$\bigwedge_{\substack{a_1,\, \ldots,\, a_6 \\ \text{ISN'T a legal window}}} (\overline{x}_{i,j-1,a_1} \vee \overline{x}_{i,j,a_2} \vee \overline{x}_{i,j,+1,a_3} \vee \overline{x}_{i+1,j-1,a_4} \vee \overline{x}_{i+1,j,a_5} \vee \overline{x}_{i+1,j+1,a})$$

This is a conjunct over all ($\le |C|^6$) illegal sequences ($a_1, \ldots, a_6$).

$$O(n^{2k})$$

Theorem (Cook-Levin): 3-SAT is NP-complete

Corollary: 3-SAT $\in$ P if and only if P = NP