Jean Goubault-Larrecq

Randomized complexity classes

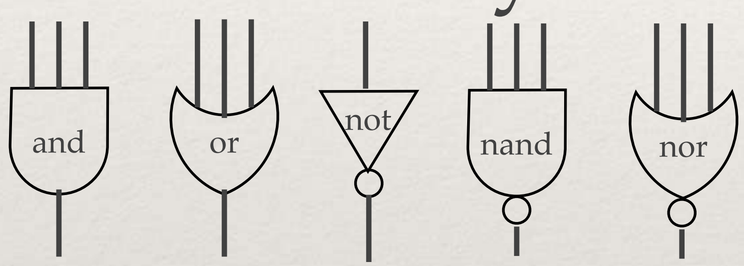
Today: **BPP** (part 2) and **P/poly**

Today

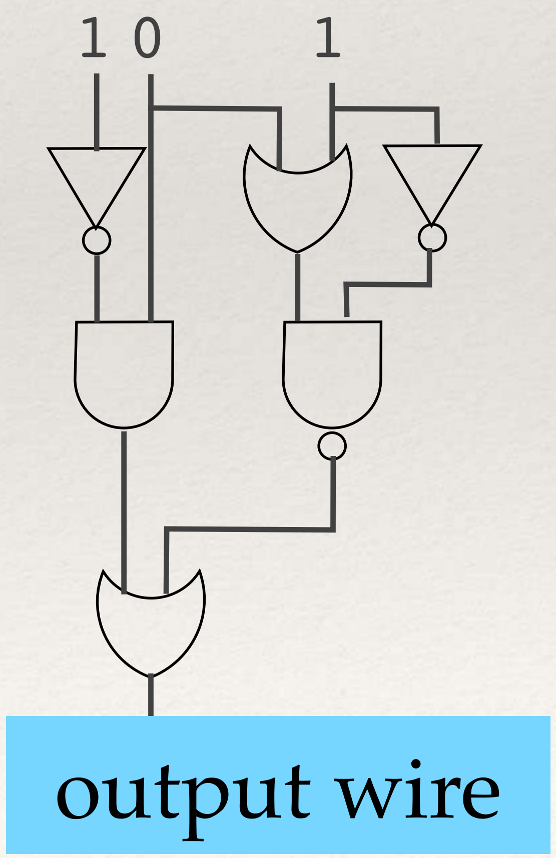
- Circuits, **P/poly**
- Adleman's theorem: $\mathbf{BPP} \subseteq \mathbf{P/poly}$
- The Karp-Lipton theorems, and consequences

Circuits

- Informally, collections of logical **gates** connected by **wires**

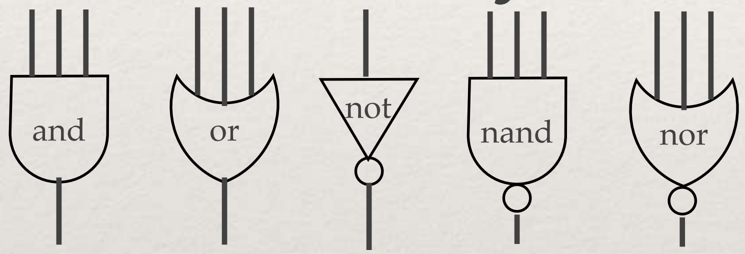


- Must be **acyclic**
- Wires can be **shared**
- **Fan-in** arbitrary here
  (e.g., 1=fan-in 0 and, 0=fan-in 0 or)



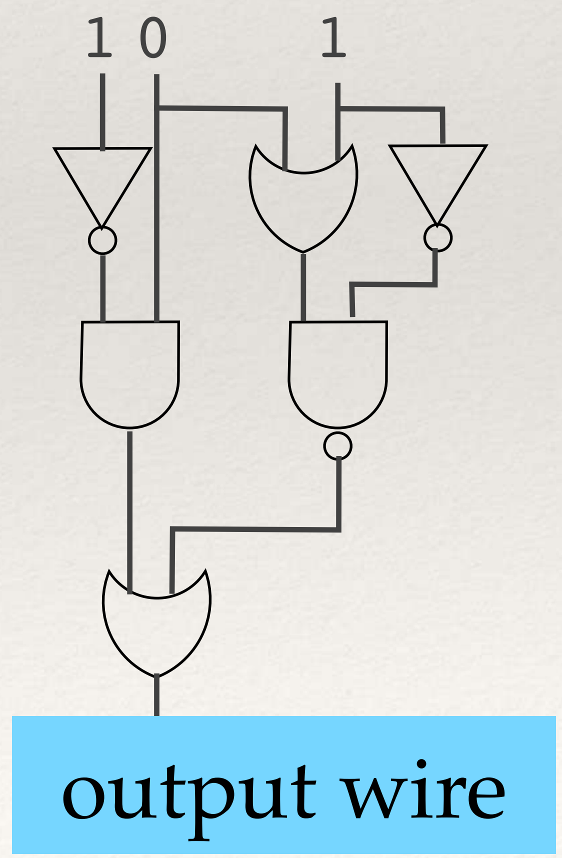
Circuits

- Informally, collections of logical **gates** connected by **wires**



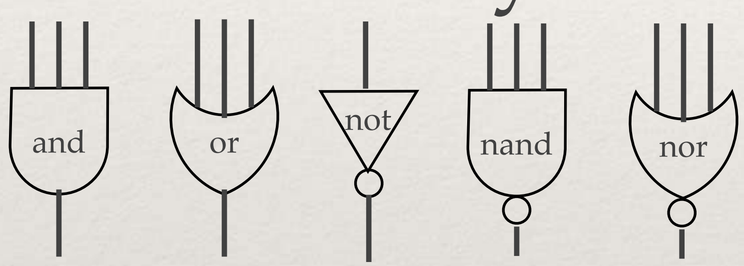
- Must be **acyclic**
- Wires can be **shared**
- **Fan-in** arbitrary here
  (e.g., 1=fan-in 0 and, 0=fan-in 0 or)

- Remember: **CIRCUIT-VALUE** is **P**-complete (for logspace reductions)



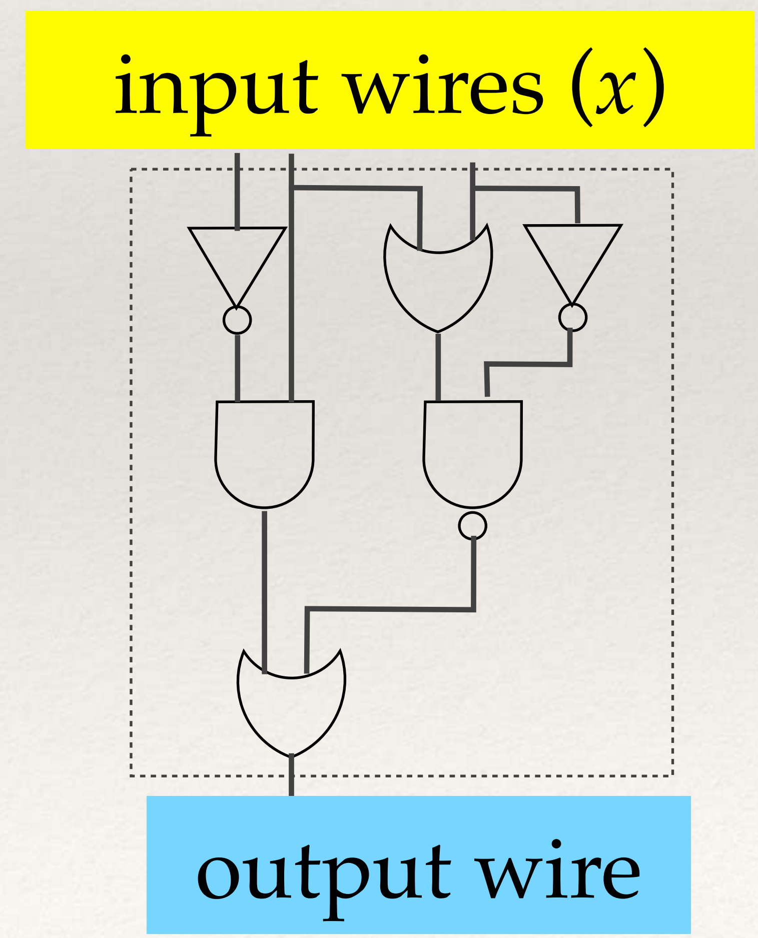
Circuits

- Informally, collections of logical **gates** connected by **wires**



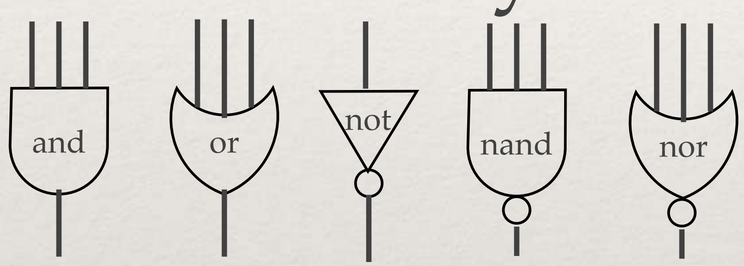
- Must be **acyclic**
- Wires can be **shared**
- **Fan-in** arbitrary here
  (e.g., 1=fan-in 0 and, 0=fan-in 0 or)

- We now consider circuits C **with input wires**



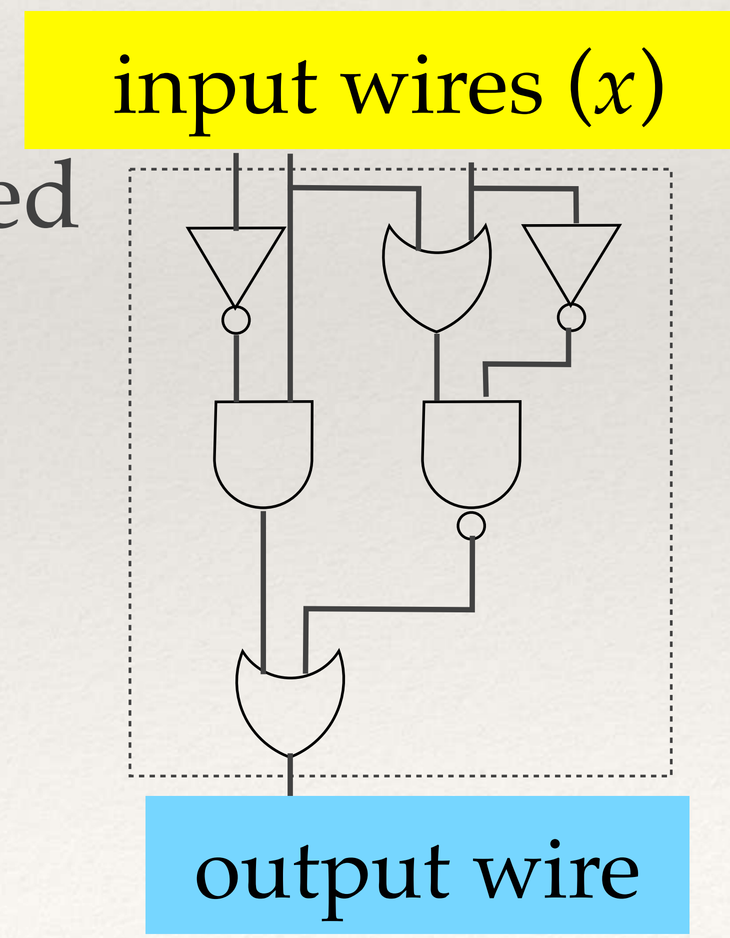
Circuits

- Informally, collections of logical **gates** connected by **wires**



- Must be **acyclic**
- Wires can be **shared**
- **Fan-in** arbitrary here
  (e.g., 1=fan-in 0 and, 0=fan-in 0 or)

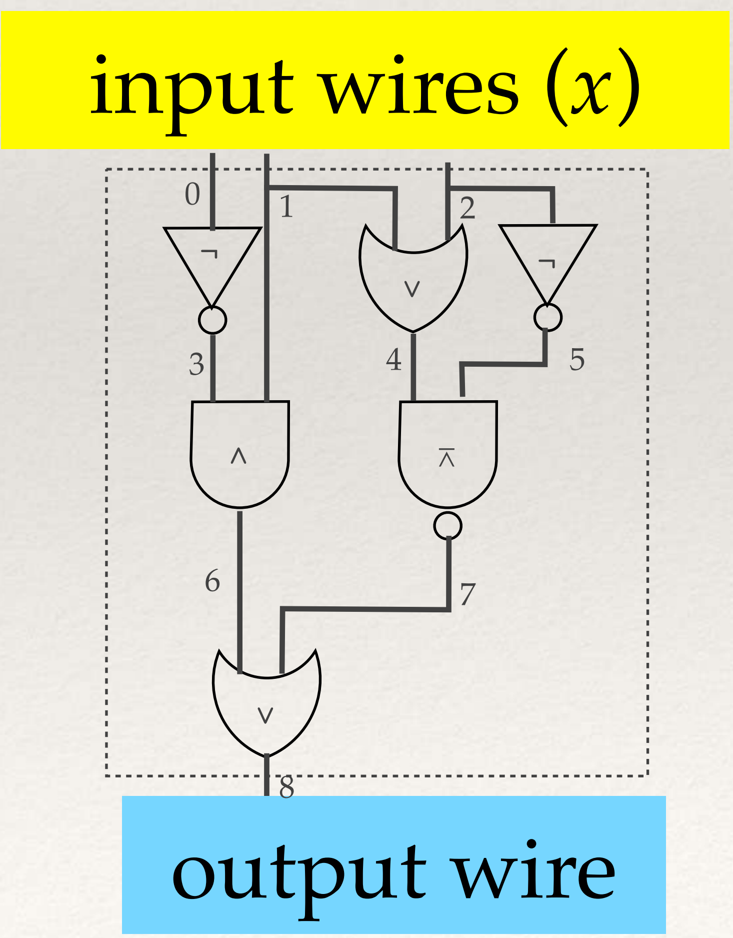
- We now consider circuits C **with input wires**
- $C[x]$ = value of C when fed input bits x



Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```



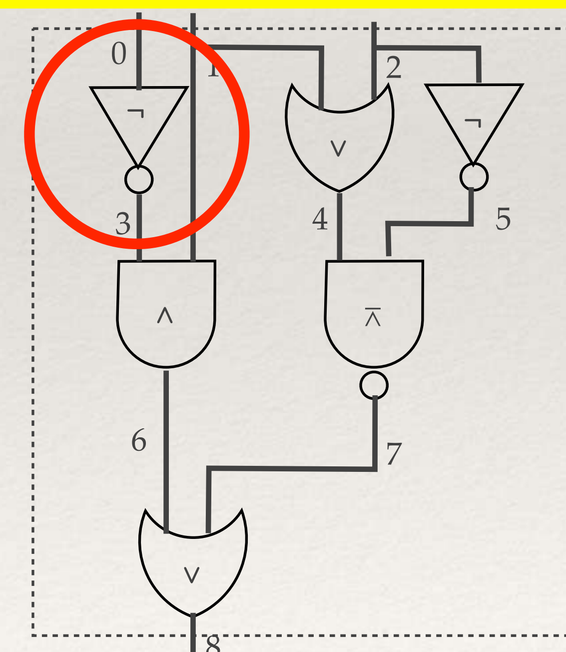
Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

wire 3 = ¬ wire 0

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```

input wires (x)



output wire

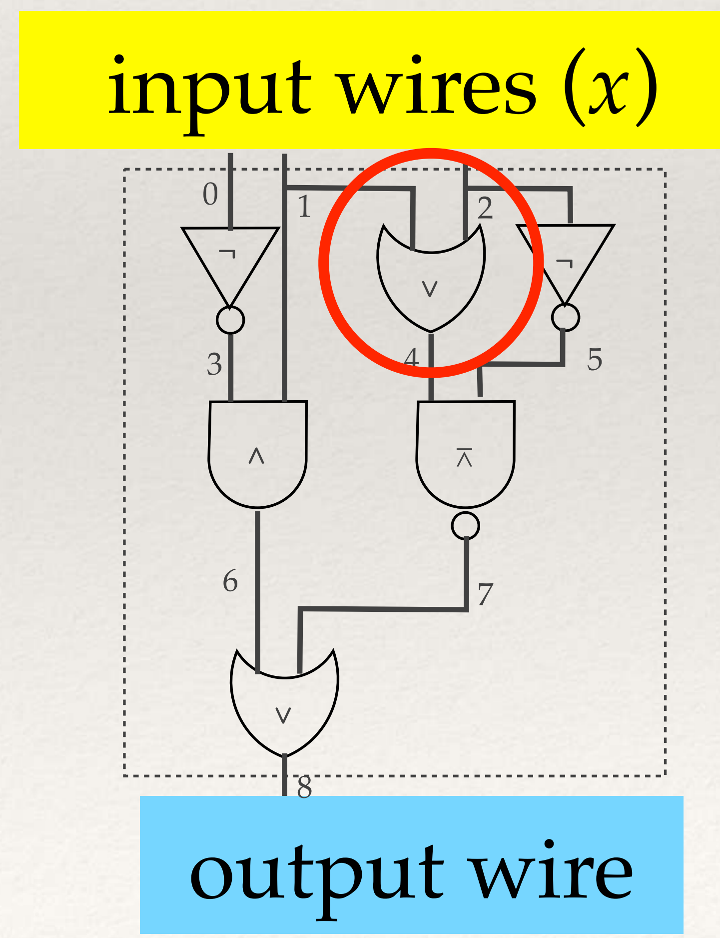
Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

wire 3 = ¬ wire 0

wire 4 = 1 ∨ 2

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```



Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

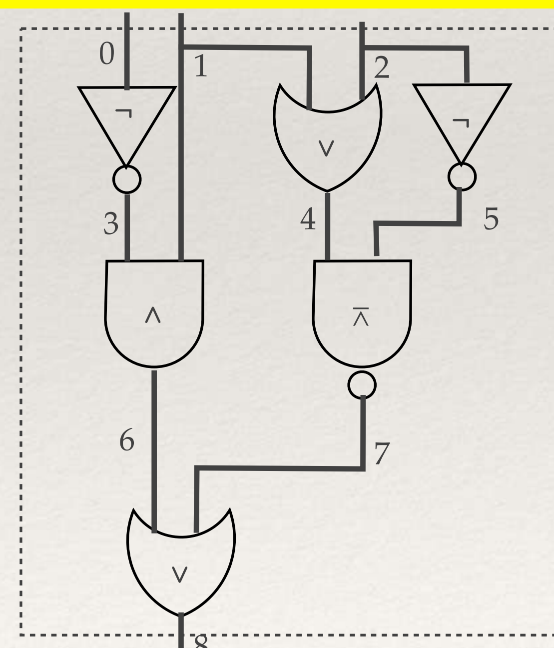
wire 3 = ¬ wire 0

wire 4 = 1 ∨ 2

etc.

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```

input wires (x)



output wire

Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

wire 3 = ¬ wire 0

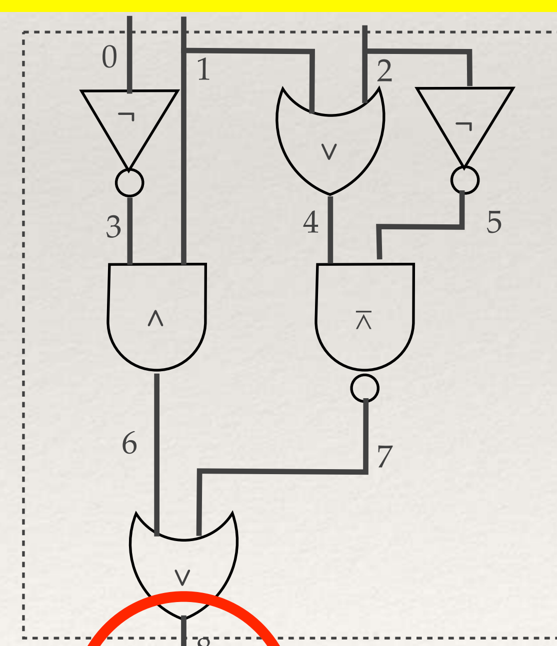
wire 4 = 1 ∨ 2

etc.

8 is output

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```

input wires (x)



output wire

Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

wire 3 = ¬ wire 0

wire 4 = 1 ∨ 2

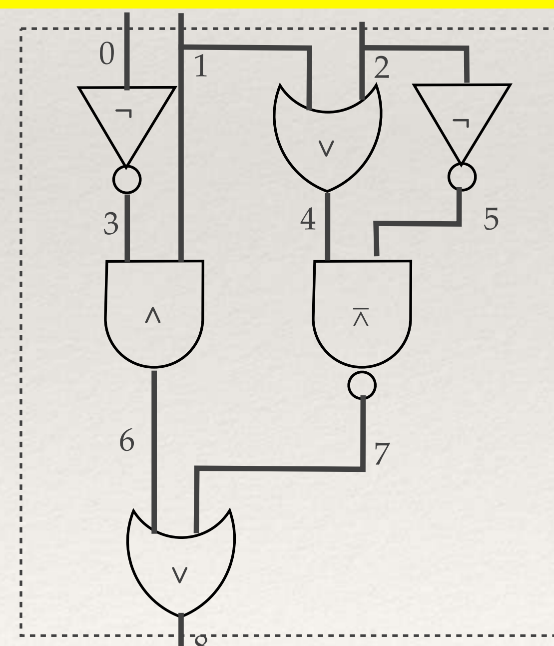
etc.

8 is output

```
3 ¬ 0
4 ∨ 1 2
5 ¬ 2
6 ∧ 1 3
7 ⊼ 4 5
8 ∨ 6 7
→8
```

wire numbers in binary

input wires (x)



output wire

Circuits, formally: net-lists

- We encode circuits as **words (net-lists)**, e.g.:

We require wire numbers to be **sorted** (implies acyclicity)
(sortedness checkable in logspace, acyclicity is **NL**-complete)

wire numbers in binary

wire 3 = ¬ wire 0

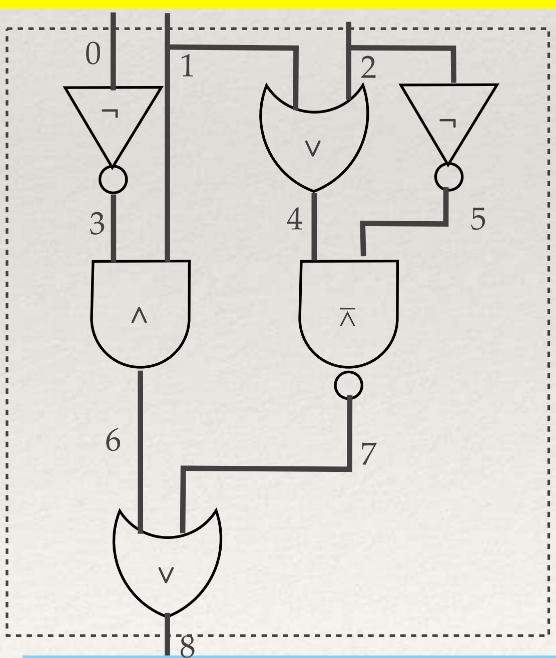
wire 4 = 1 ∨ 2

etc.

8 is output

$3 \;\neg\; 0$
$4 \;\vee\; 1 \;\; 2$
$5 \;\neg\; 2$
$6 \;\wedge\; 1 \;\; 3$
$7 \;\bar{\wedge}\; 4 \;\; 5$
$8 \;\vee\; 6 \;\; 7$
$\to 8$

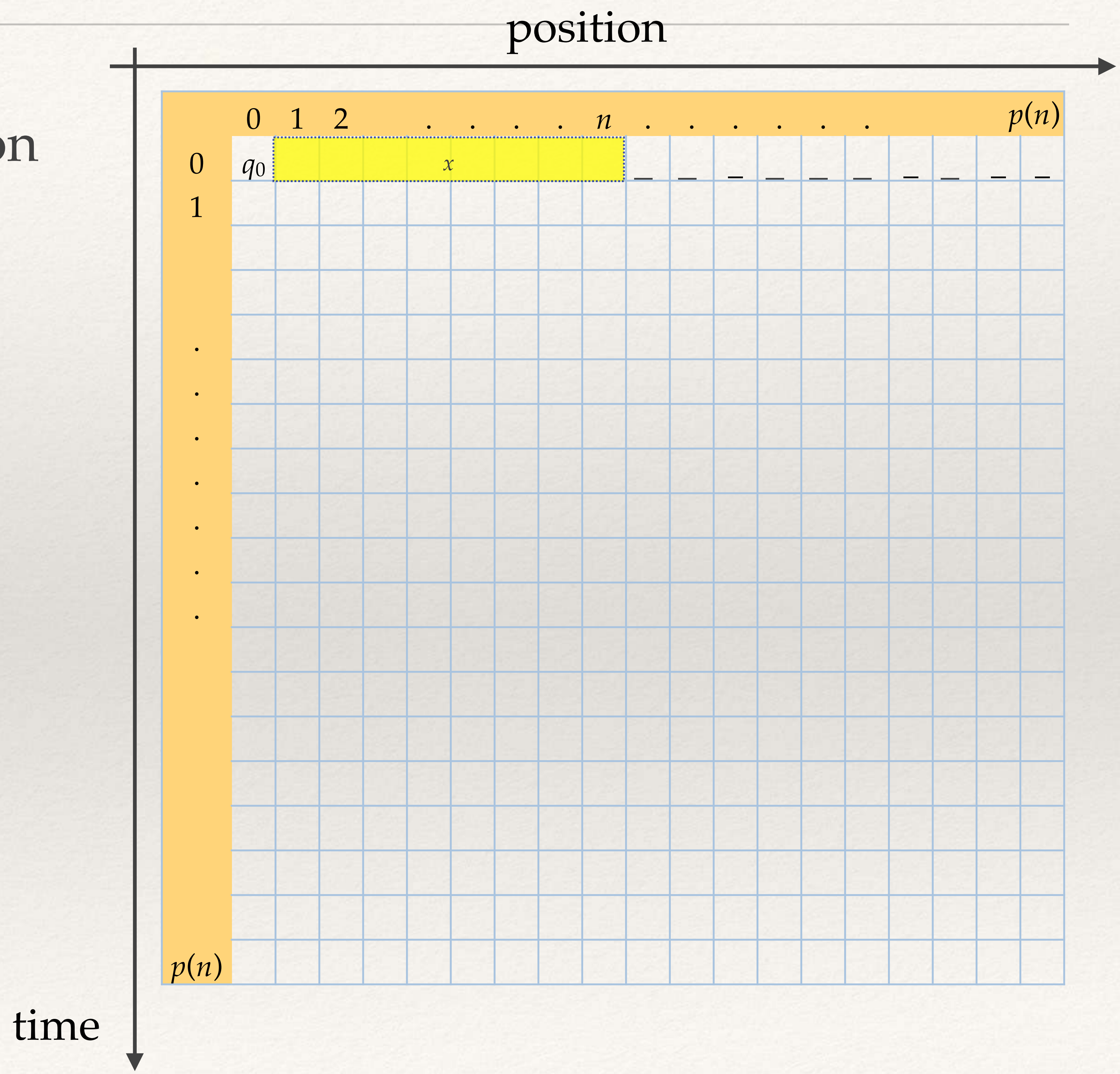
input wires (x)



output wire

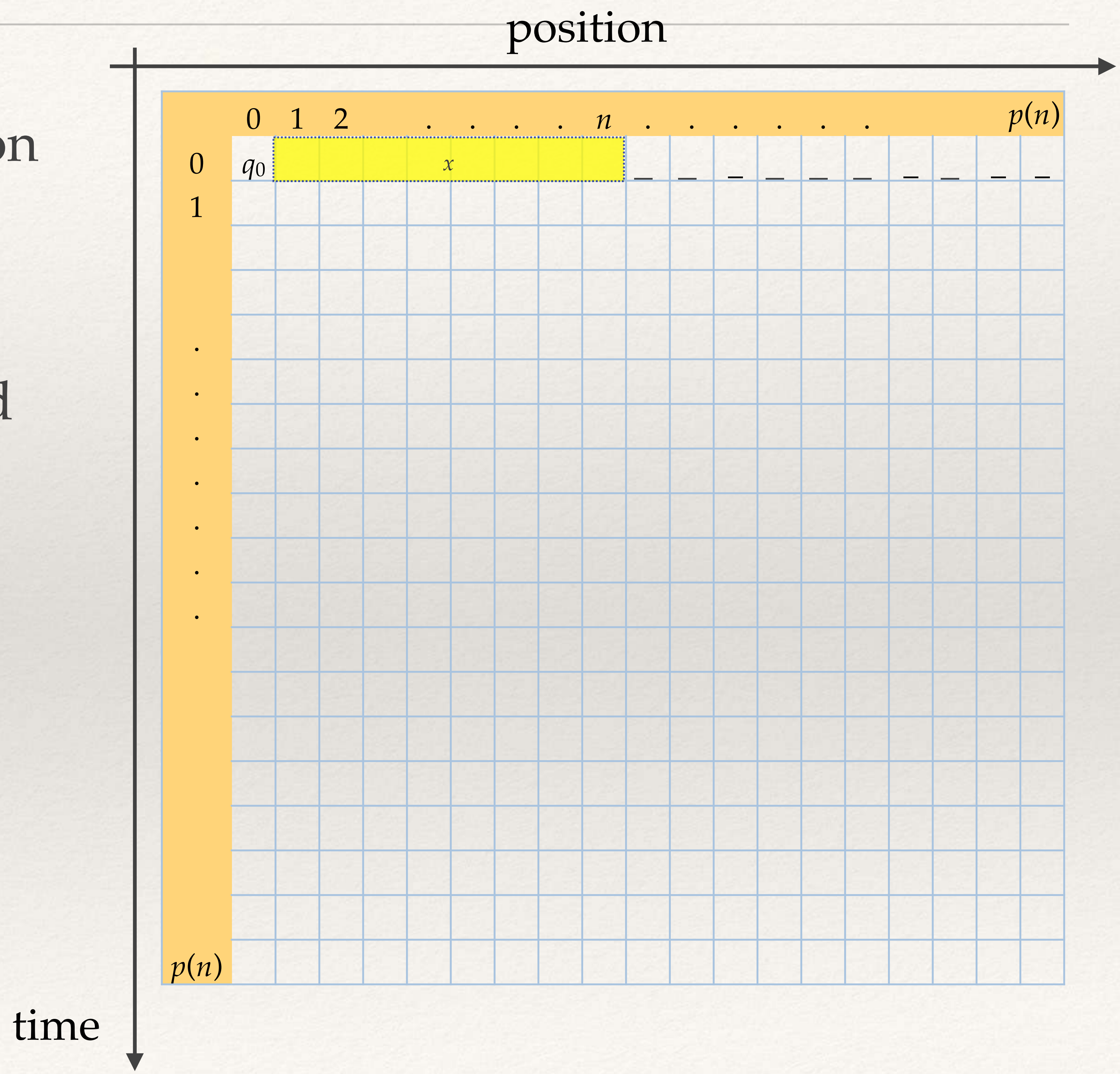
Reminder: **CIRCUIT-VALUE** is **P**-complete

- Encode $p(n)$-time TM $\mathcal{M}$ on input x by a circuit



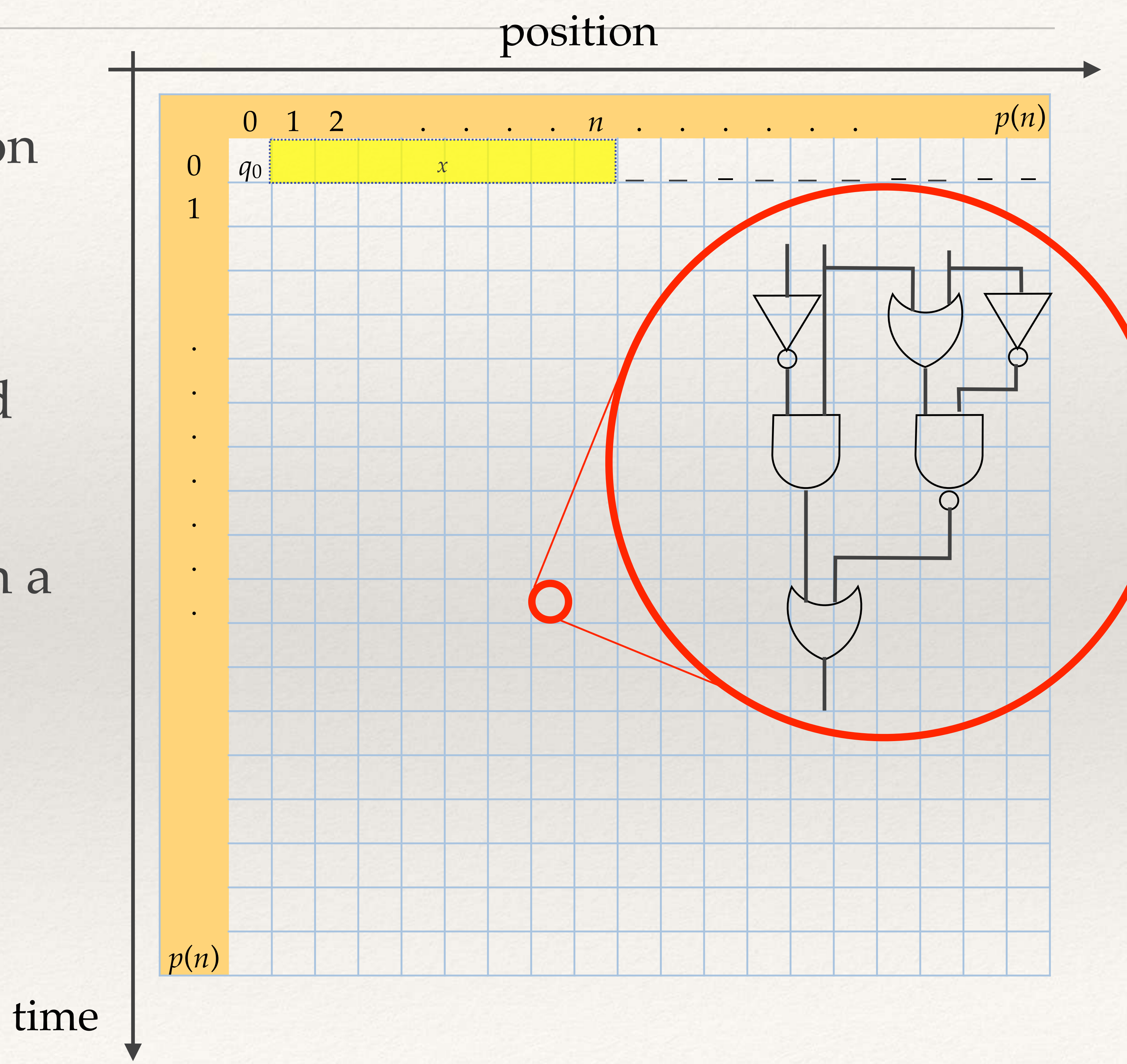
Reminder: **CIRCUIT-VALUE** is **P**-complete

- Encode $p(n)$-time TM $\mathcal{M}$ on input x by a circuit
- constant gates 1 / 0 encode initial state q_0, input x, and blanks



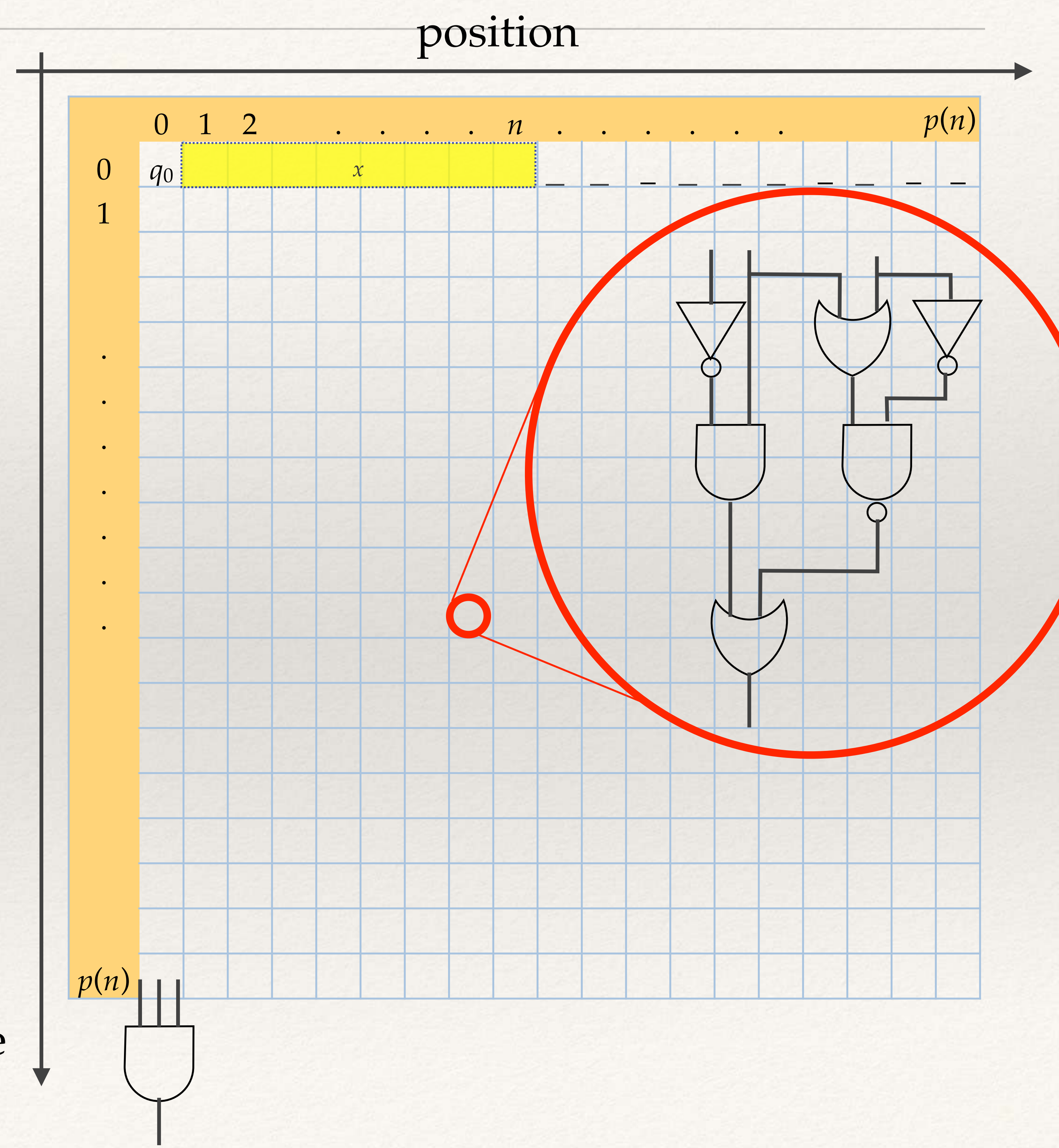
Reminder: **CIRCUIT-VALUE** is **P**-complete

- Encode $p(n)$-time TM $\mathcal{M}$ on input x by a circuit
- constant gates 1 / 0 encode initial state q_0, input x, and blanks
- each inner cell depends on a constant # cells on row above
  $\Rightarrow$ **circuit piece** of cst size (replicated $p(n)^2$ times)



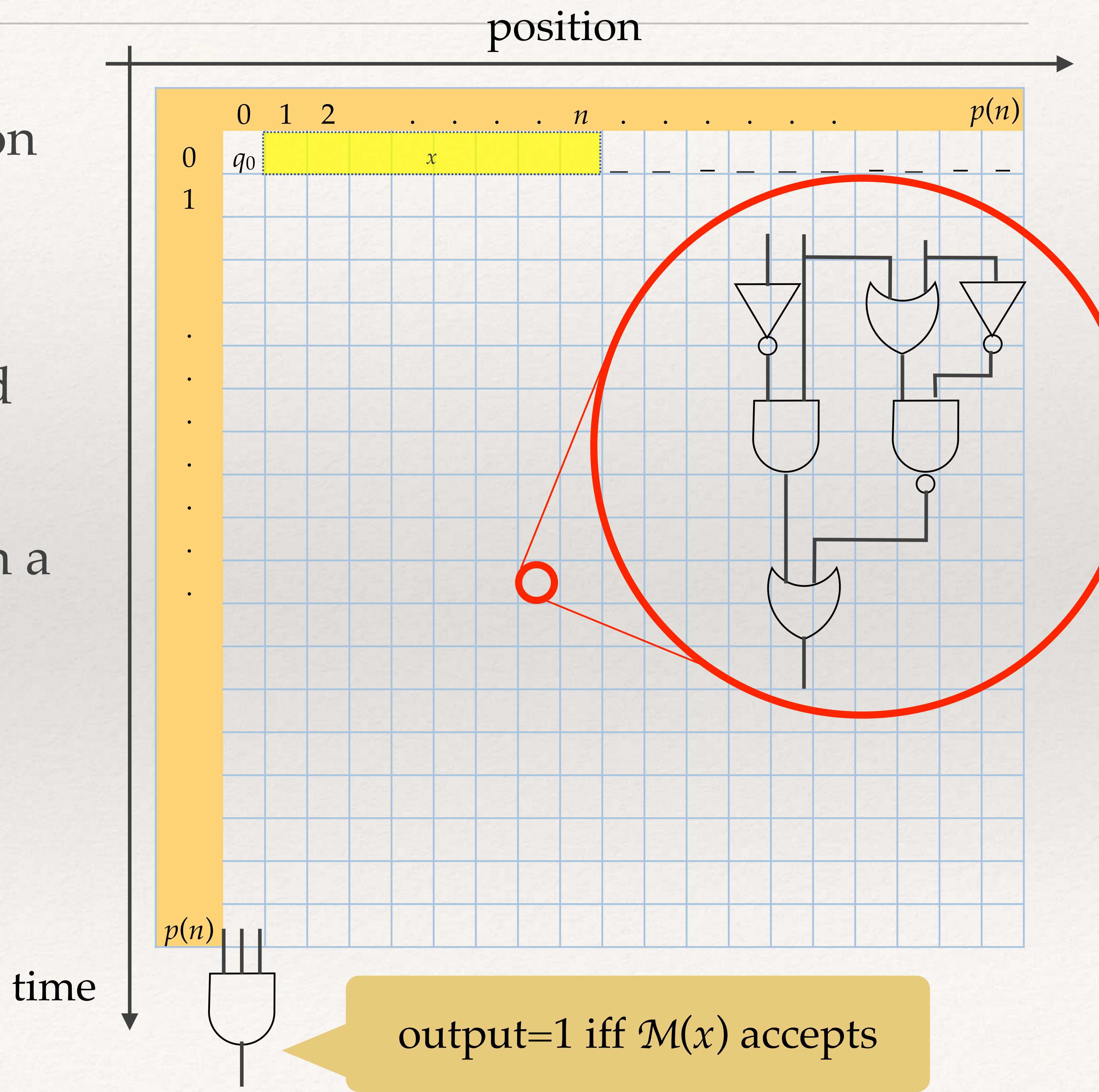
Reminder: **CIRCUIT-VALUE** is **P**-complete

- Encode $p(n)$-time TM $\mathcal{M}$ on input x by a circuit
- constant gates 1 / 0 encode initial state q_0, input x, and blanks
- each inner cell depends on a constant # cells on row above
  $\Rightarrow$ **circuit piece** of cst size (replicated $p(n)^2$ times)
- finally, a small circuit to check **acceptance.**



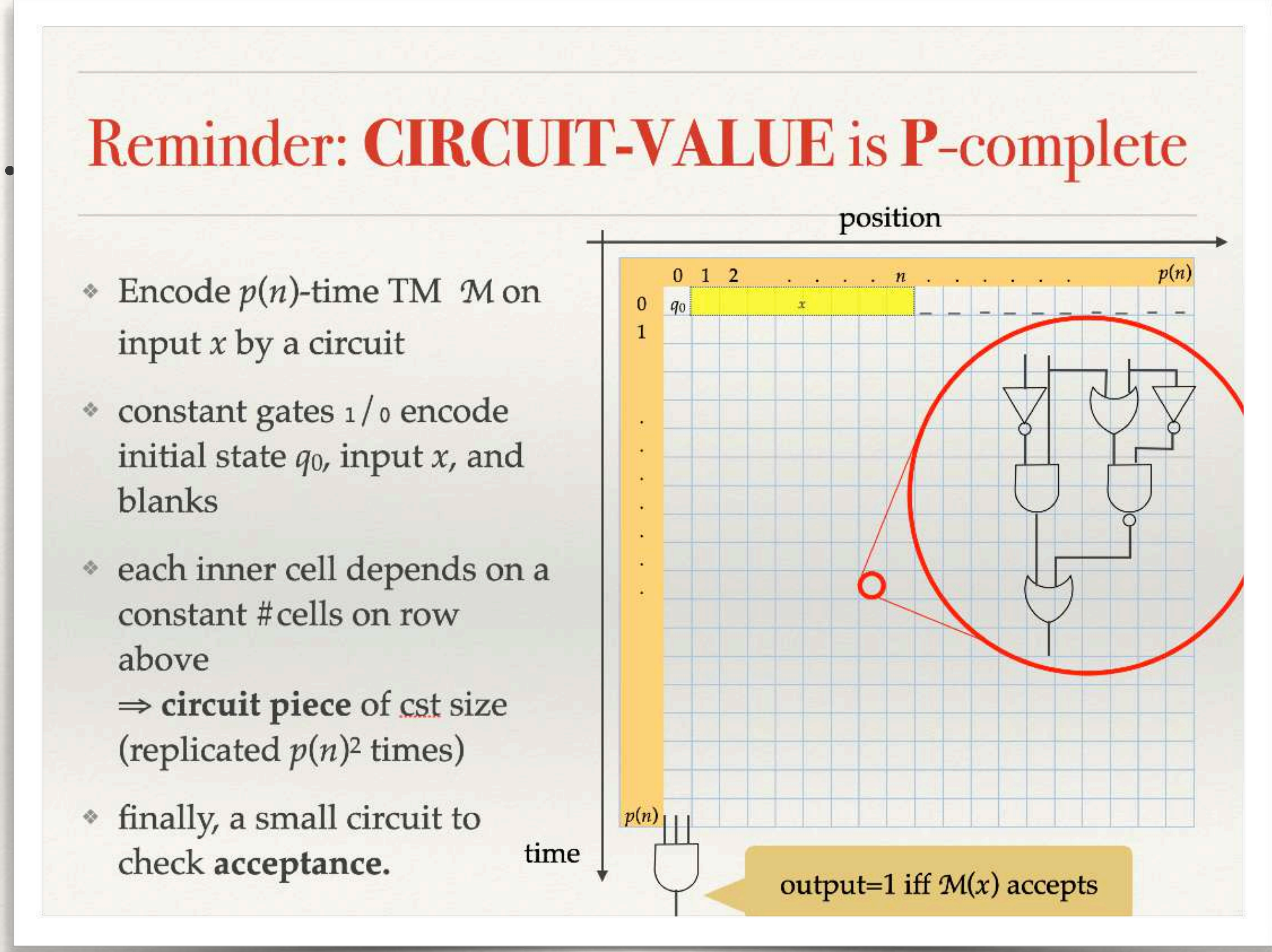
Reminder: **CIRCUIT-VALUE** is **P**-complete

- Encode $p(n)$-time TM $\mathcal{M}$ on input x by a circuit
- constant gates 1 / 0 encode initial state q_0, input x, and blanks
- each inner cell depends on a constant #cells on row above
  ⇒ **circuit piece** of cst size (replicated $p(n)^2$ times)
- finally, a small circuit to check **acceptance.**



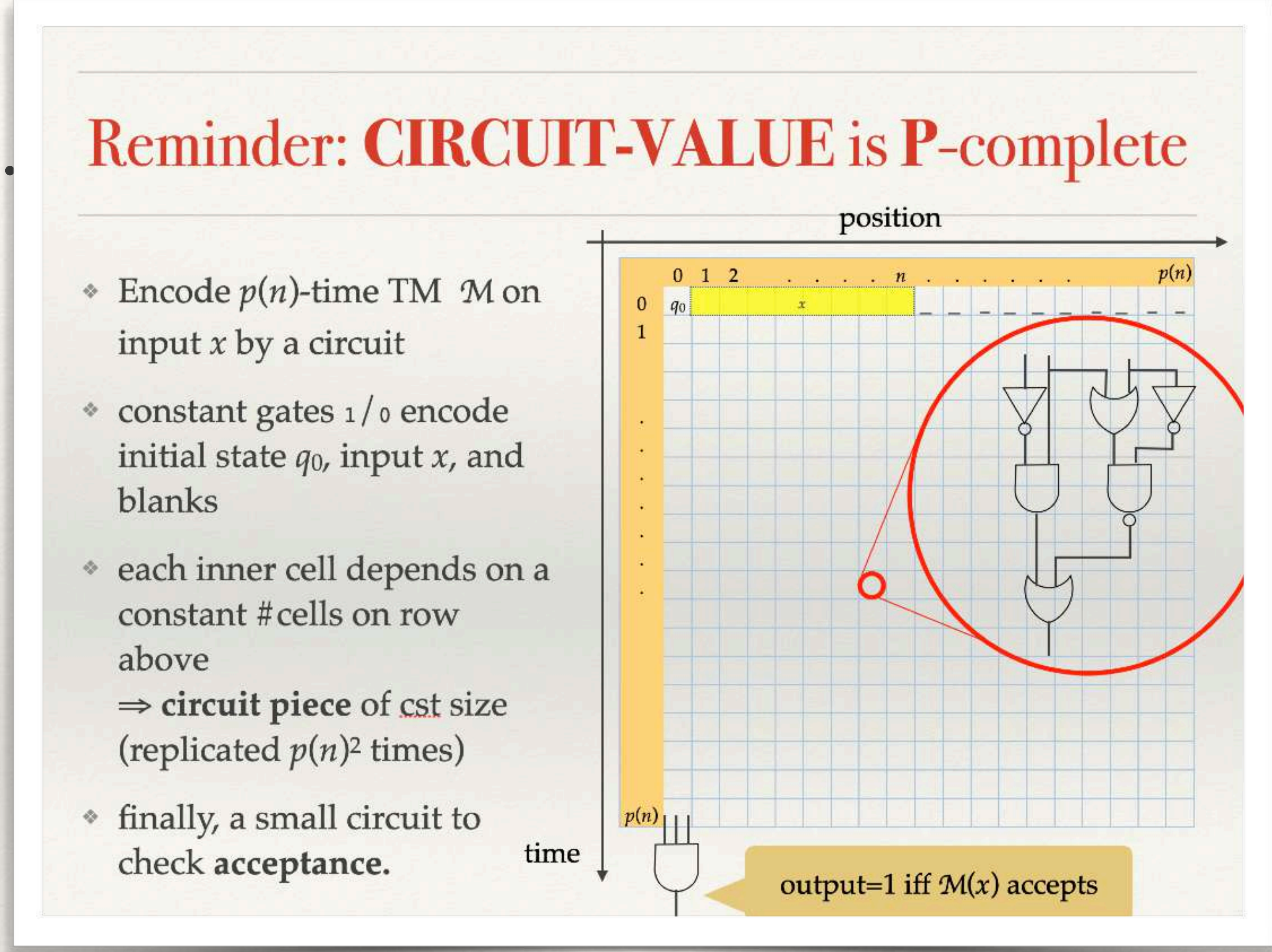
Plenty of technical details…

- Each row encodes a config. of a **one-tape** TM $\mathcal{M}$



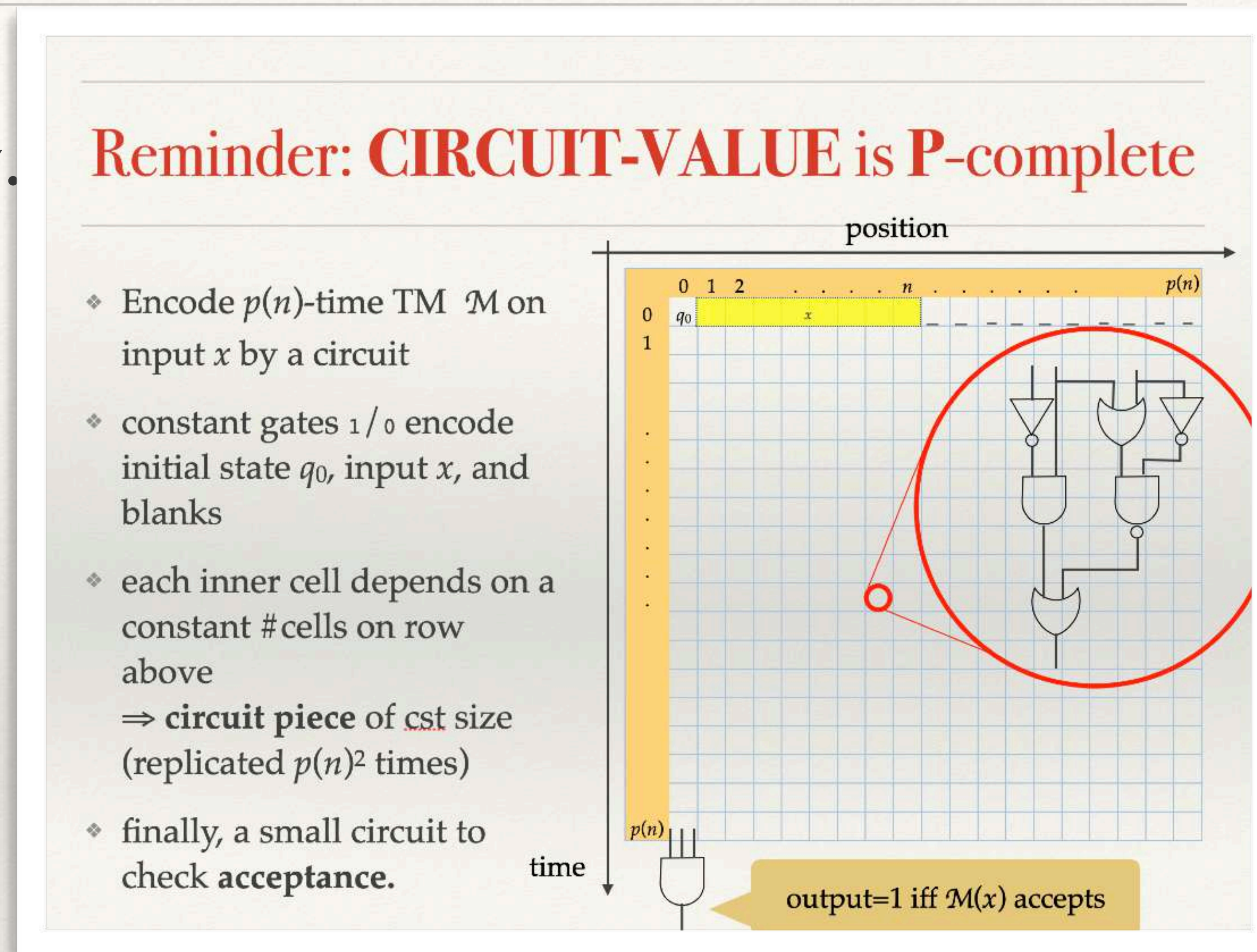
Plenty of technical details…

- Each row encodes a config. of a **one-tape** TM $\mathcal{M}$
- … in **binary**



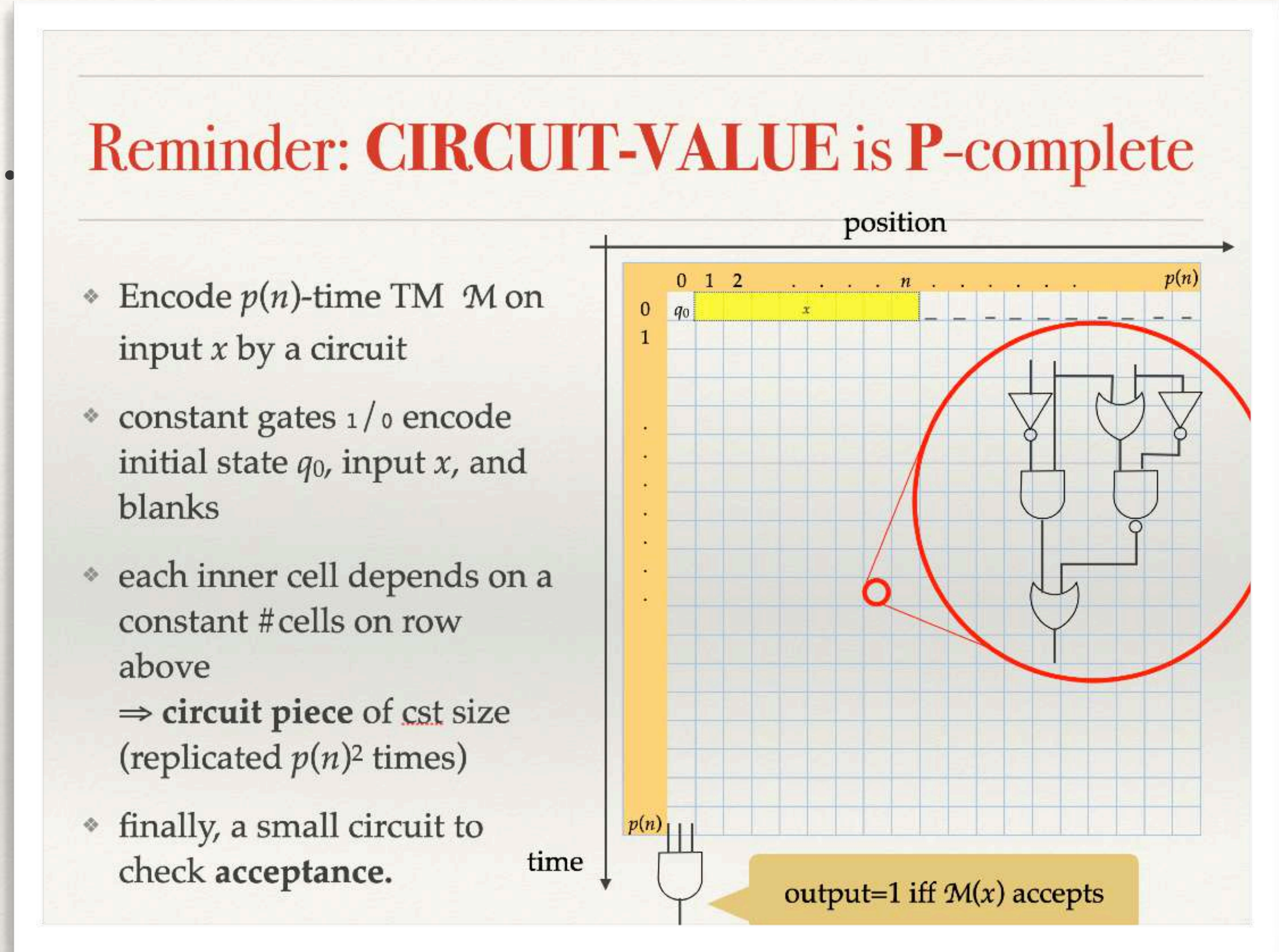
Plenty of technical details…

- Each row encodes a config. of a **one-tape** TM $\mathcal{M}$
- … in **binary**
- the machine **parks** the head at position 0 before accepting / rejecting



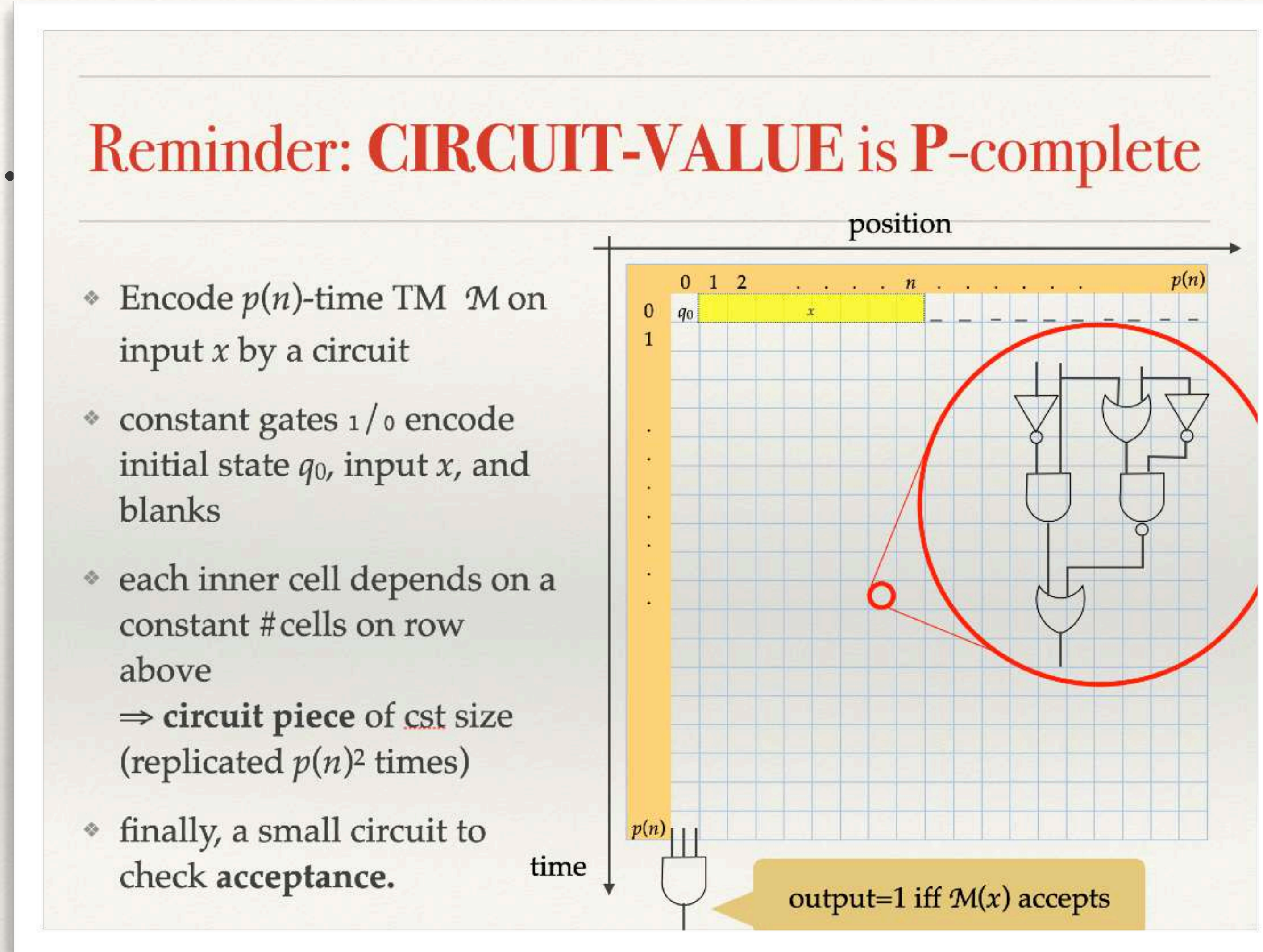
Plenty of technical details…

- Each row encodes a config. of a **one-tape** TM $\mathcal{M}$
- … in **binary**
- the machine **parks** the head at position 0 before accepting / rejecting
- … and continues working (doing **nothing**) forever (at least until time $p(n)$)



Plenty of technical details…

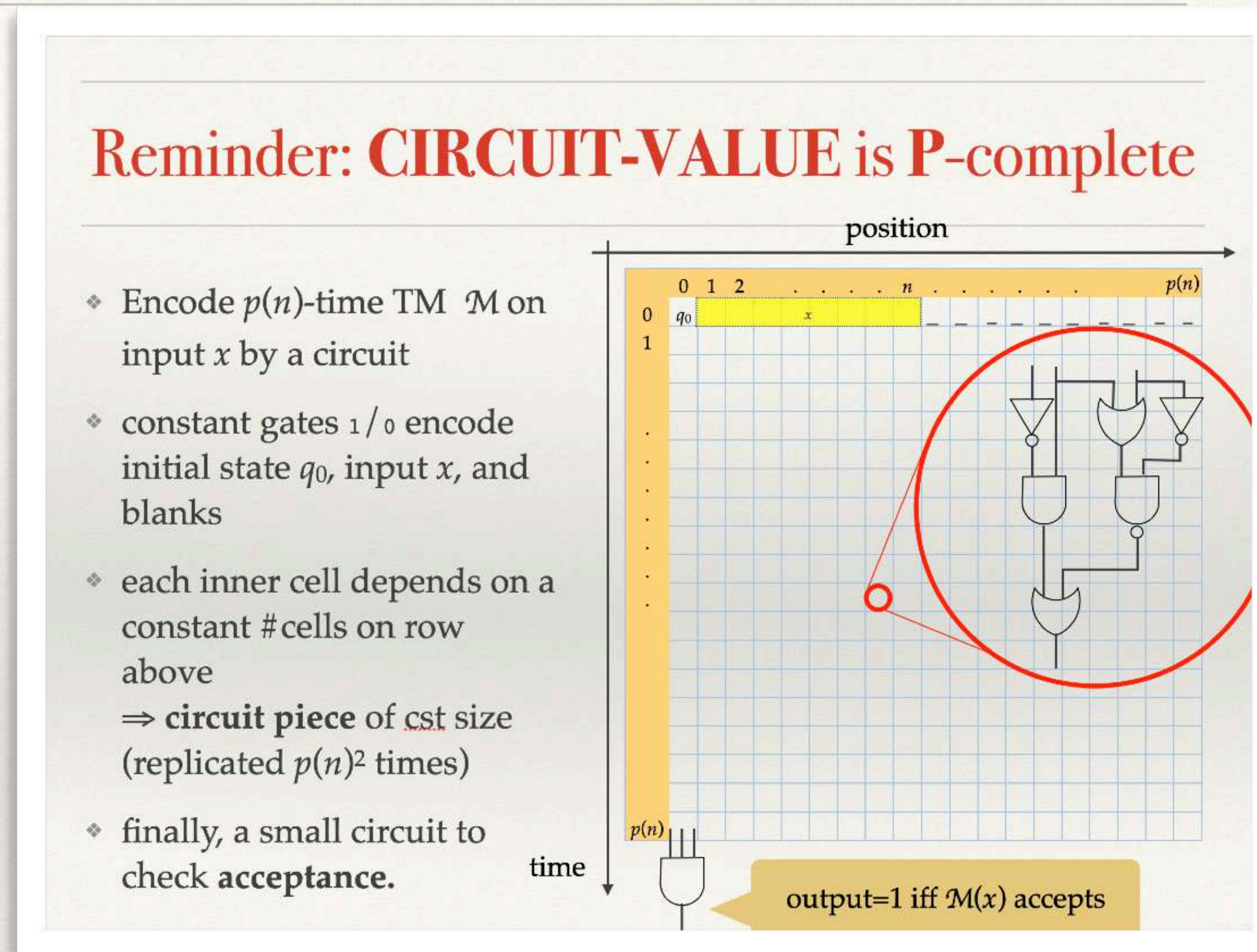
- Each row encodes a config. of a **one-tape** TM $\mathcal{M}$
- … in **binary**
- the machine **parks** the head at position 0 before accepting / rejecting
- … and continues working (doing **nothing**) forever (at least until time $p(n)$)



- Build the circuit in **logspace**: 2 nested loops from 0 to $p(n)$, with 2 **counters**

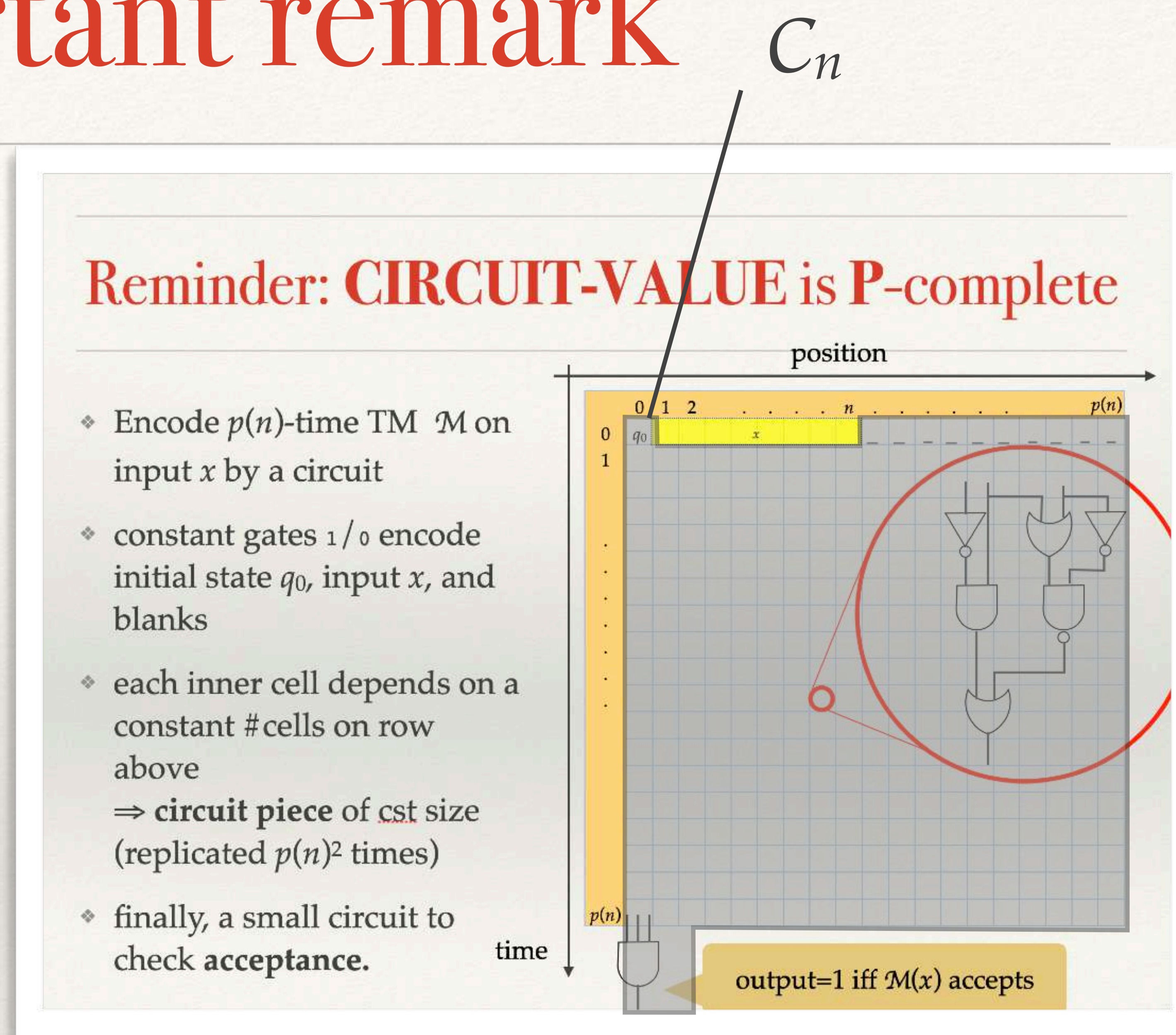
An important remark

- We can **precompile** a circuit C_n with n free input wires
  — without knowing x,
  — just its length n,
  — still in logspace



An important remark

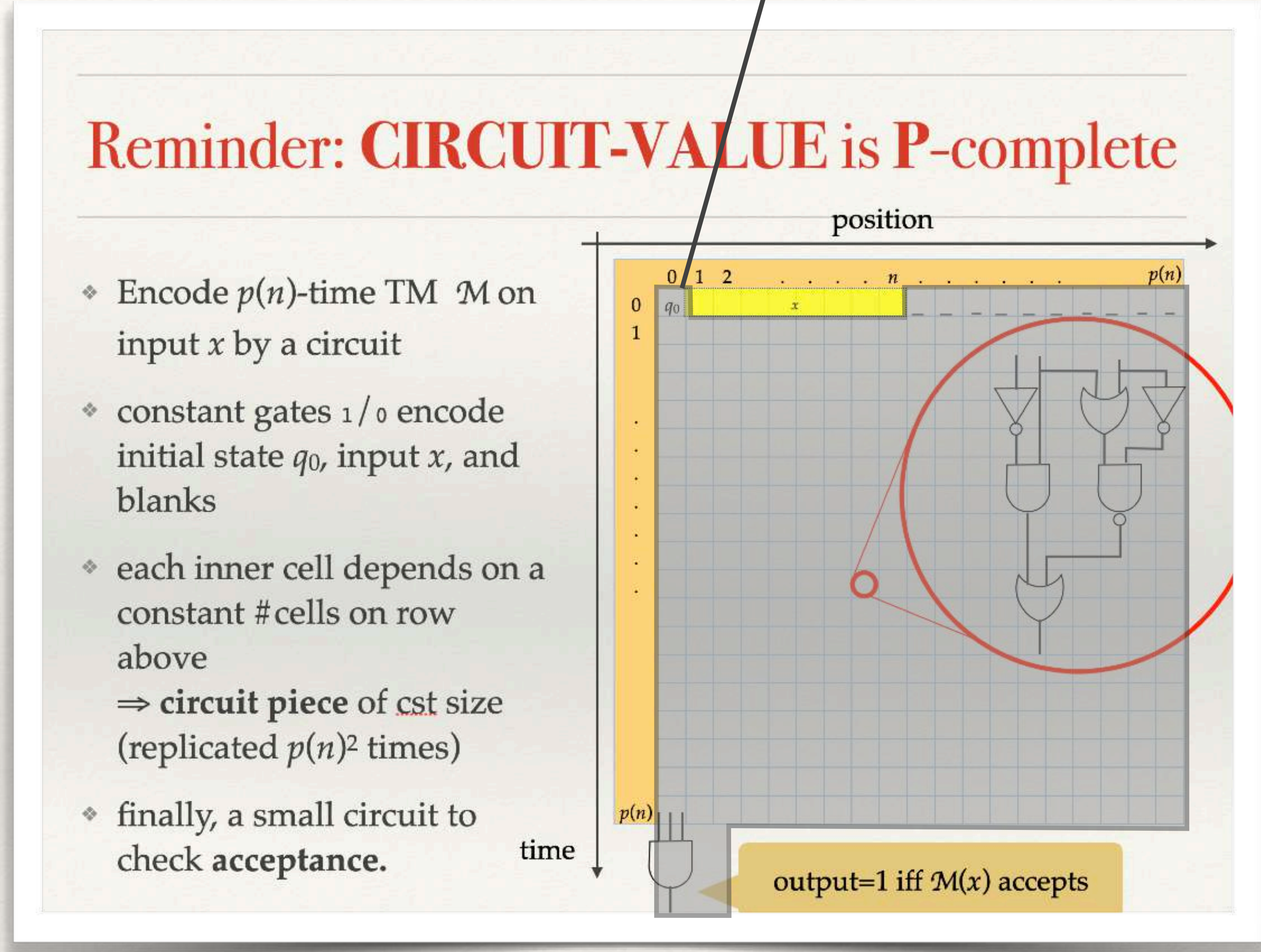
- We can **precompile** a circuit C_n with n free input wires
— without knowing x,
— just its length n,
— still in logspace



An important remark

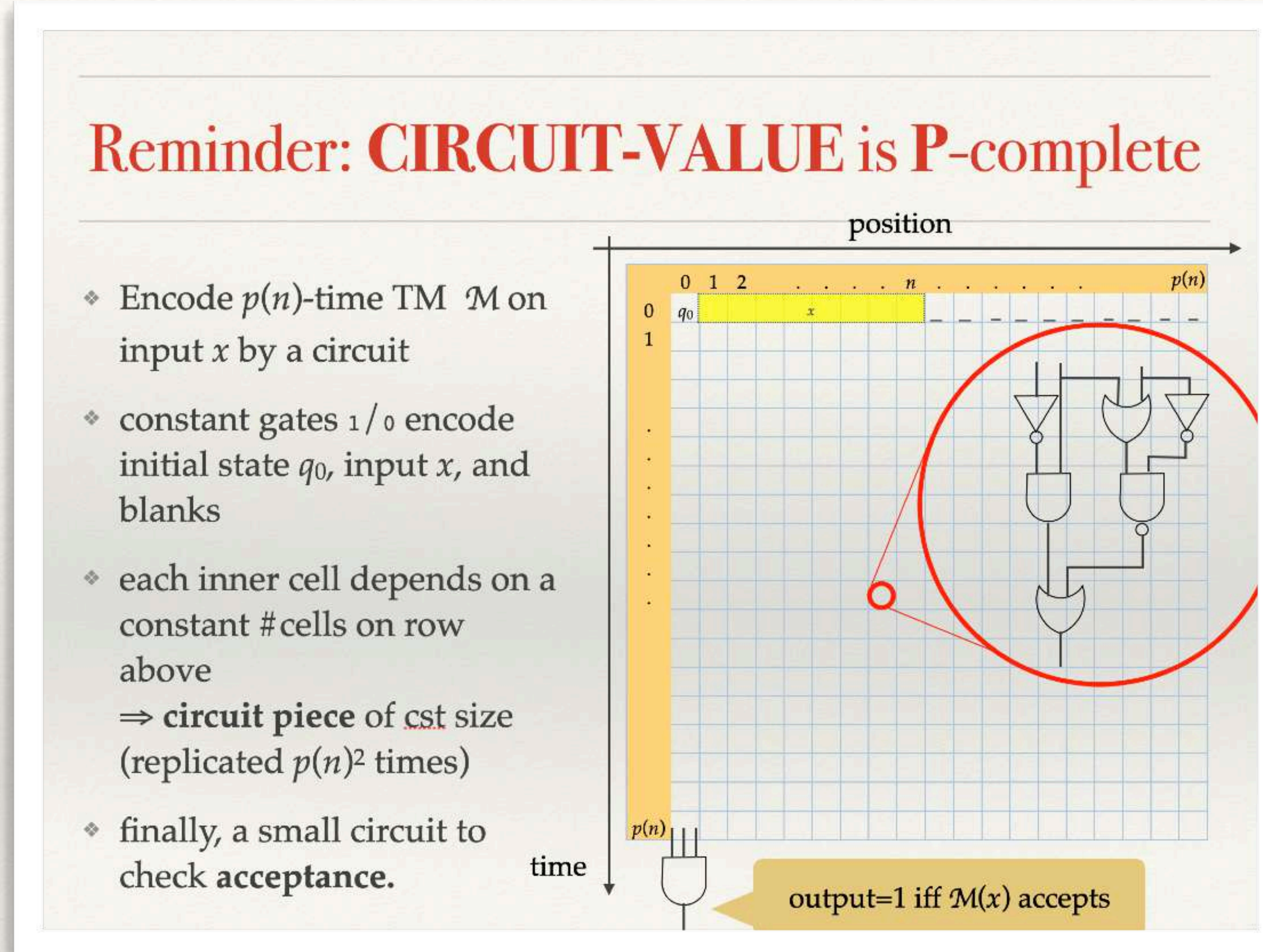
- We can **precompile** a circuit C_n with n free input wires
  — without knowing x,
  — just its length n,
  — still in logspace
- such that for **every** x of that size n,
  $\mathcal{M}(x)$ accepts $\Leftrightarrow C_n[x]=1$

C_n



Uniform P/poly

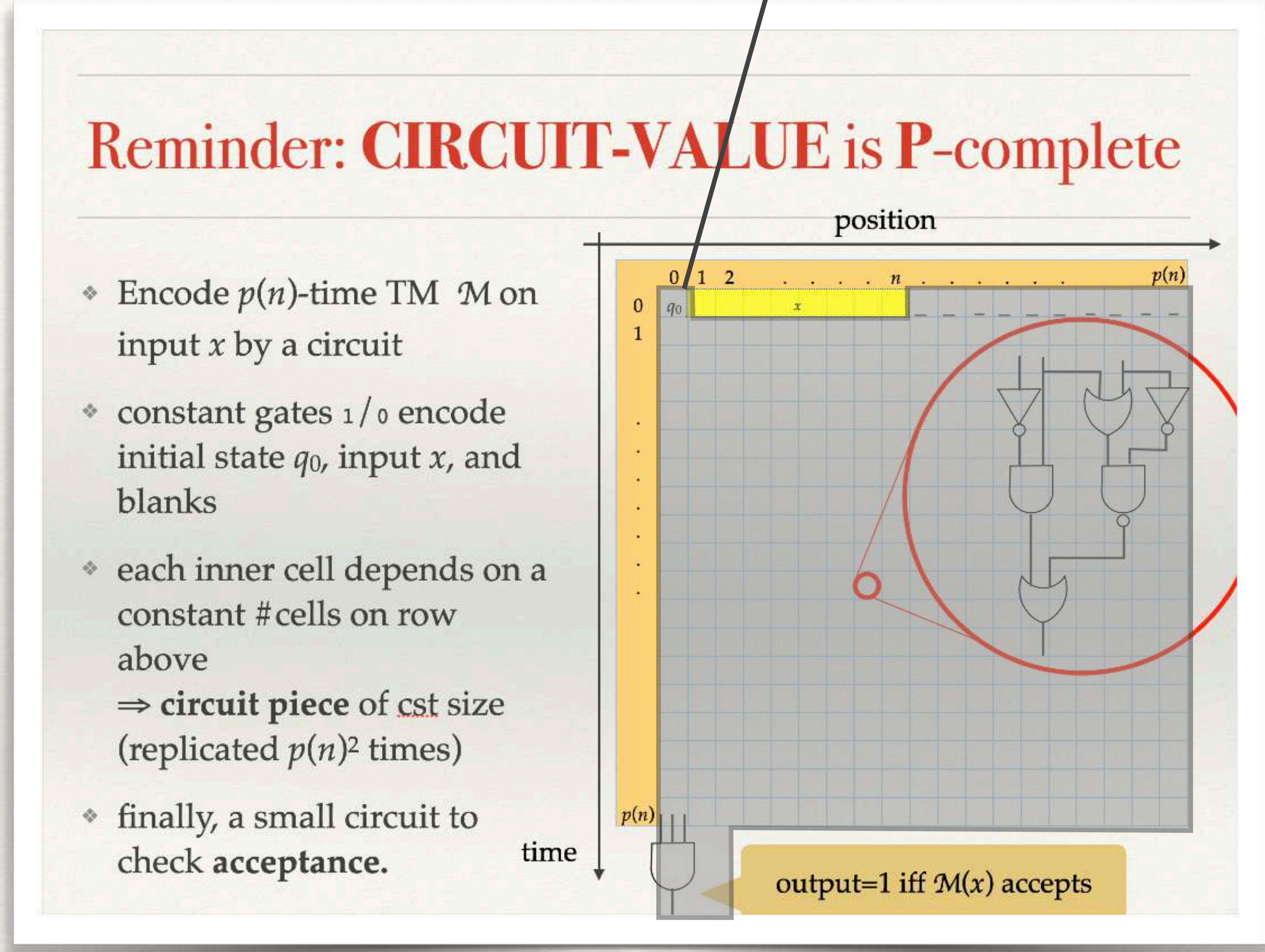
- A language L is in **uniform P/poly** iff for every n, one can build a circuit C_n
  — in space $O(\log n)$
  — such that for every input x of size $= n$, $x \in L \Leftrightarrow C_n[x]=1$



Uniform P/poly

C_n

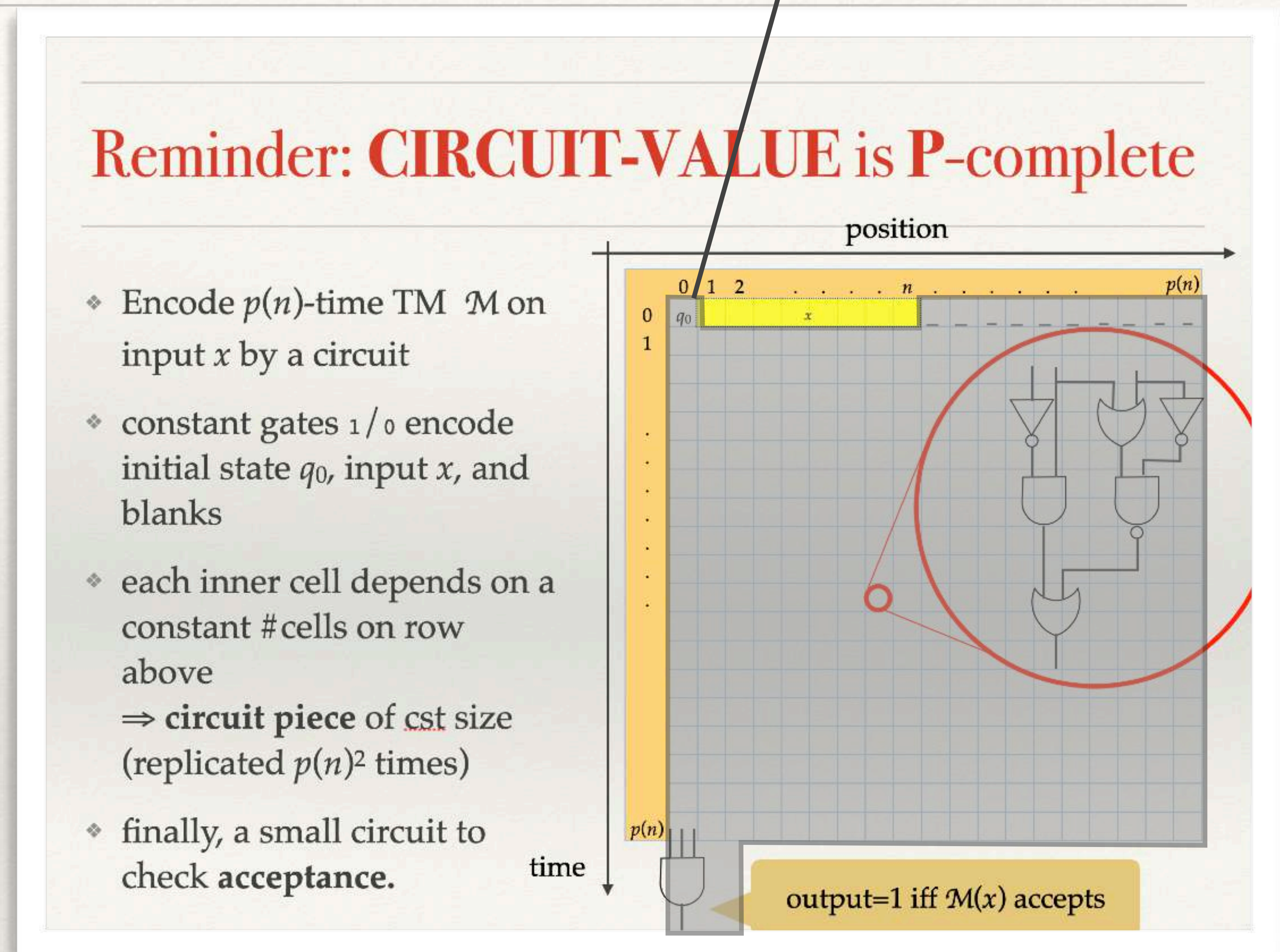
- A language L is in **uniform P/poly** iff for every n, one can build a circuit C_n
  — in space $O(\log n)$
  — such that for every input x of size $= n$, $x \in L \Leftrightarrow C_n[x]=1$



Uniform P/poly

C_n

- A language L is in **uniform P/poly** iff for every n, one can build a circuit C_n
  — in space $O(\log n)$
  — such that for every input x of size $= n$, $x \in L \Leftrightarrow C_n[x]=1$



- **Prop. P $\subseteq$ uniform P/poly.** (This is what we have just proved!)

P = Uniform P/poly

- A language L is in **uniform P/poly** iff
  for every n,
  one can build a circuit C_n
  — in space $O(\log n)$
  — such that for every
  input x of size $= n$,
  $x \in L \Leftrightarrow C_n[x]=1$
- **Prop. P $\subseteq$ uniform P/poly**.

- In fact:
  Prop. P = uniform P/poly.
- *Proof.*
  Let $L \in$ **uniform P/poly**.
  On input x (size n),
  compute C_n in space $k \log n$,
  hence in time $O(n^k)$.
  Then evaluate $C_n[x]$
  in polytime.
  Hence $L \in$ **P**.

(Non-uniform) P/poly

- A language L is in
  ~~**uniform**~~ **P/poly** iff
  for every n,
  ~~one can build~~ there is a circuit C_n
  of size $p(n)$, for some fixed polynomial p
  — ~~in space O(log n)~~
  — such that for every
  input x of size $= n$,
  $x \in L \Leftrightarrow C_n[x]=1$

(Non-uniform) P/poly

- A language L is in ~~**uniform**~~ **P/poly** iff for every n, ~~one can build~~ there is a circuit C_n ~~— in space O(log n)~~ of size $p(n)$, for some fixed polynomial p — such that for every input x of size $= n$, $x \in L \Leftrightarrow C_n[x]=1$

We no longer require to be able to **compute** C_n!

(Non-uniform) P/poly

- A language L is in ~~uniform~~ **P/poly** iff for every n, ~~one can build~~ there is a circuit C_n of size $p(n)$, for some fixed polynomial p ~~— in space O(log n)~~ — such that for every input x of size $= n$, $x \in L \Leftrightarrow C_n[x]=1$

We no longer require to be able to **compute** C_n!

- Familiarly, we say that L **has polynomial circuits**

P/poly

- **Defn.** A language L is in **P/poly** iff

  there is a family $(C_n)_{n \in \mathbb{N}}$ of circuits:

  — of size $p(n)$ (for some fixed polynomial p)

  — such that for every input x (letting n be its size)

  $$x \in L \Leftrightarrow C_n[x]=1.$$

P/poly

- **Defn.** A language L is in **P/poly** iff
  there is a family $(C_n)_{n \in \mathbb{N}}$ of circuits:
  — of size $p(n)$ (for some fixed polynomial p)
  — such that for every input x (letting n be its size)
  $$x \in L \Leftrightarrow C_n[x]=1.$$
- It was initially hoped that we could prove that some **NP**-complete languages do **not** have polynomial circuits. That would immediately imply **P**≠**NP**, since **P** ⊆ **P/poly**.

P/poly is pretty weird

- **Prop. P/poly** contains some undecidable languages.

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]=1$.

P/poly is pretty weird

- **Prop. P/poly** contains some undecidable languages.

> **Defn.** A language L is in **P/poly** iff
> there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
> — of size $p(n)$ (for some fixed polynomial p)
> — such that for every input x (letting n be its si
> $x \in L \Leftrightarrow C_n[x]=1$.

- *Proof.*
  Let L be undecidable (e.g., **HALT**).
  Then $L' = \{\text{words } 1^n \mid$
  $a_1 \ldots a_k \in L,\ n = a_1 + 2a_2 + \ldots + 2^{k-1}a_k + 2^k\}$
  is undecidable, too; and C_n is…

convert from binary to unary

If $\mathrm{bin}(n) \notin L$	If $\mathrm{bin}(n) \in L$

P/poly is pretty weird

- **Prop. P/poly** contains some undecidable languages.

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]=1$.

- *Proof.*
Let L be undecidable (e.g., **HALT**).
Then $L' = \{$words 1^n |
$a_1\ldots a_k \in L$, $n = a_1+2a_2+\ldots+2^{k-1}a_k+2^k\}$
is undecidable, too; and C_n is…

convert from binary to unary

If bin(n) $\notin L$	If bin(n) $\in L$
0 (ignores its input, size O(1))	

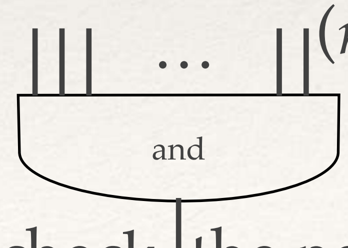
P/poly is pretty weird

- **Prop. P/poly** contains some undecidable languages.

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]=1$.

- *Proof.*
Let L be undecidable (e.g., **HALT**).
Then $L' = \{$words 1^n |
$$a_1\ldots a_k \in L,\ n = a_1+2a_2+\ldots+2^{k-1}a_k+2^k\}$$
is undecidable, too; and C_n is…

convert from binary to unary

If bin(n) $\notin L$	If bin(n) $\in L$
0 (ignores its input, size O(1))	 (n input bits) (size n log n: check the net-list!)

Weird, too: advice strings

- Imagine you wish to decide whether x is in L.
- … and you have a « cheat sheet » w_n depending only on n=size(x).
  How can this help?

Corpus ID: 115398060

Turing machines that take advice

R. Karp, R. J. Lipton · Published 1982 · Computer Science

https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/520px-Karp_mg_7725-b.cr2.jpg

https://cyber.gatech.edu/sites/default/files/styles/faculty_bio_pic/public/dick-lipton_1.jpg?itok=EkU43aPB

Weird, too: advice strings

- Imagine you wish to decide whether x is in L.
- … and you have a « cheat sheet » w_n depending only on n=size(x).
  How can this help?
- If w_n allowed to have size 2^n, then this helps **a lot** (why?)

Corpus ID: 115398060

Turing machines that take advice

R. Karp, R. J. Lipton · Published 1982 · Computer Science

https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/520px-Karp_mg_7725-b.cr2.jpg

https://cyber.gatech.edu/sites/default/files/styles/faculty_bio_pic/public/dick-lipton_1.jpg?itok=EkU43aPB

Weird, too: advice strings

- Imagine you wish to decide whether x is in L.
- … and you have a « cheat sheet » w_n depending only on n=size(x).
  How can this help?
- If w_n allowed to have size 2^n, then this helps **a lot** (why?)
- What if w_n is only allowed to have **polynomial size?**

Corpus ID: 115398060

Turing machines that take advice

R. Karp, R. J. Lipton · Published 1982 · Computer Science

https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/520px-Karp_mg_7725-b.cr2.jpg

https://cyber.gatech.edu/sites/default/files/styles/faculty_bio_pic/public/dick-lipton_1.jpg?itok=EkU43aPB

Advice strings and **P/poly** (1/2)

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]=1$.

- **Prop.** $L \in$ **P/poly** iff there is a polytime TM $\mathcal{M}$ and a family $(w_n)_{n\in\mathbb{N}}$ of so-called
  advice strings:
  — of polysize $p(n)$
  — s.t. $\forall\, x$ (size n)
  $x \in L \Leftrightarrow \mathcal{M}(x,w_n)$ accepts.

Proof.

Advice strings and **P/poly** (1/2)

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]{=}1$.

- **Prop.** $L \in$ **P/poly** iff there is a polytime TM $\mathcal{M}$ and a family $(w_n)_{n\in\mathbb{N}}$ of so-called
  advice strings:
  — of polysize $p(n)$
  — s.t. $\forall\, x$ (size n)
  $x \in L \Leftrightarrow \mathcal{M}(x,w_n)$ accepts.

Proof.

- If $L \in$ **P/poly**, then let w_n be a net-list for C_n

Advice strings and **P/poly** (1/2)

Defn. A language L is in **P/poly** iff
there is a family $(C_n)_{n\in\mathbb{N}}$ of circuits:
— of size $p(n)$ (for some fixed polynomial p)
— such that for every input x (letting n be its si
$x \in L \Leftrightarrow C_n[x]=1$.

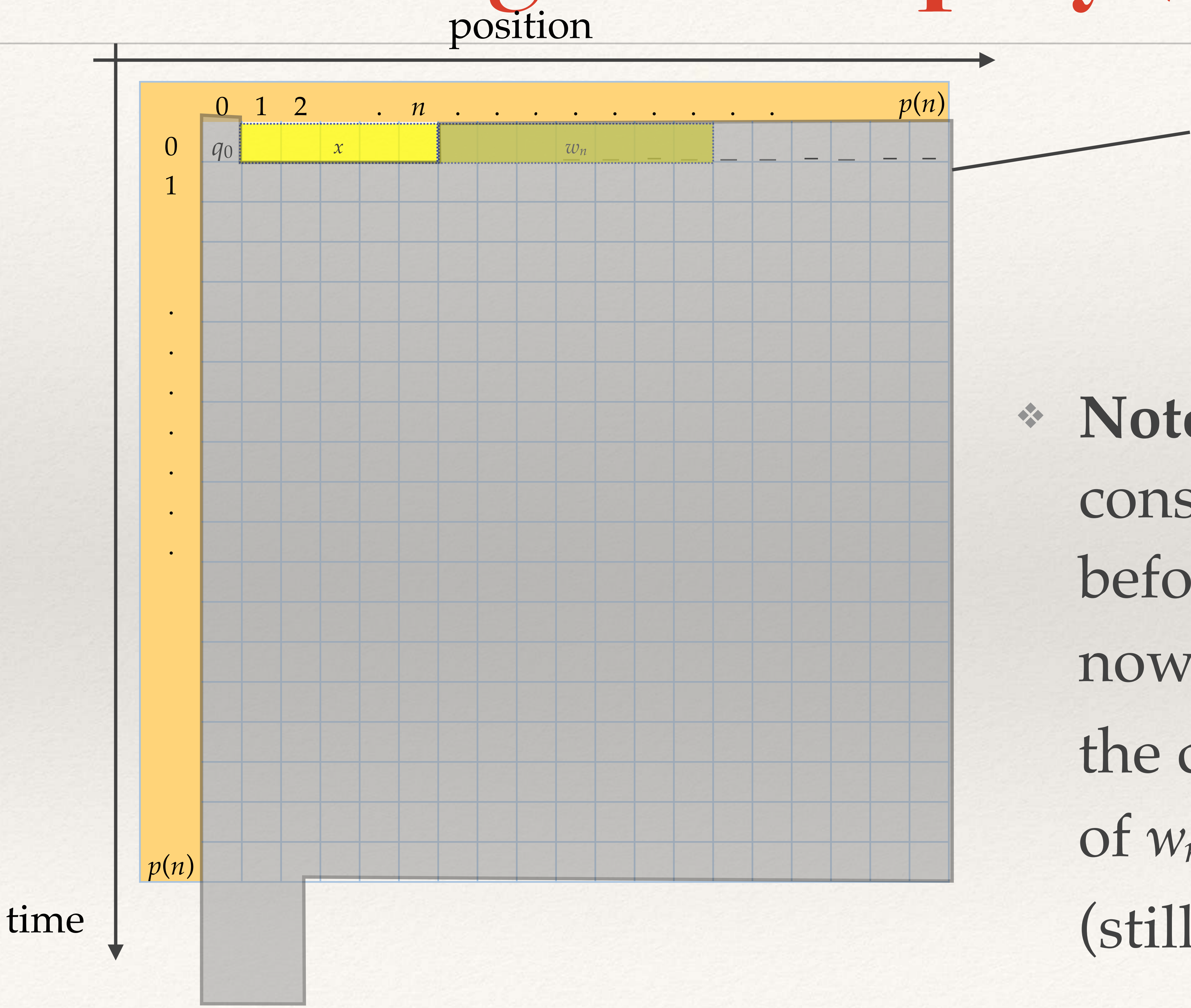
- **Prop.** $L \in$ **P/poly** iff there is a polytime TM $\mathcal{M}$ and a family $(w_n)_{n\in\mathbb{N}}$ of so-called
  advice strings:
  — of polysize $p(n)$
  — s.t. $\forall\ x$ (size n)
  $x \in L \Leftrightarrow \mathcal{M}(x,w_n)$ accepts.

Proof.

- If $L \in$ **P/poly**, then let w_n be a net-list for C_n
- If L has advice strings w_n, then…

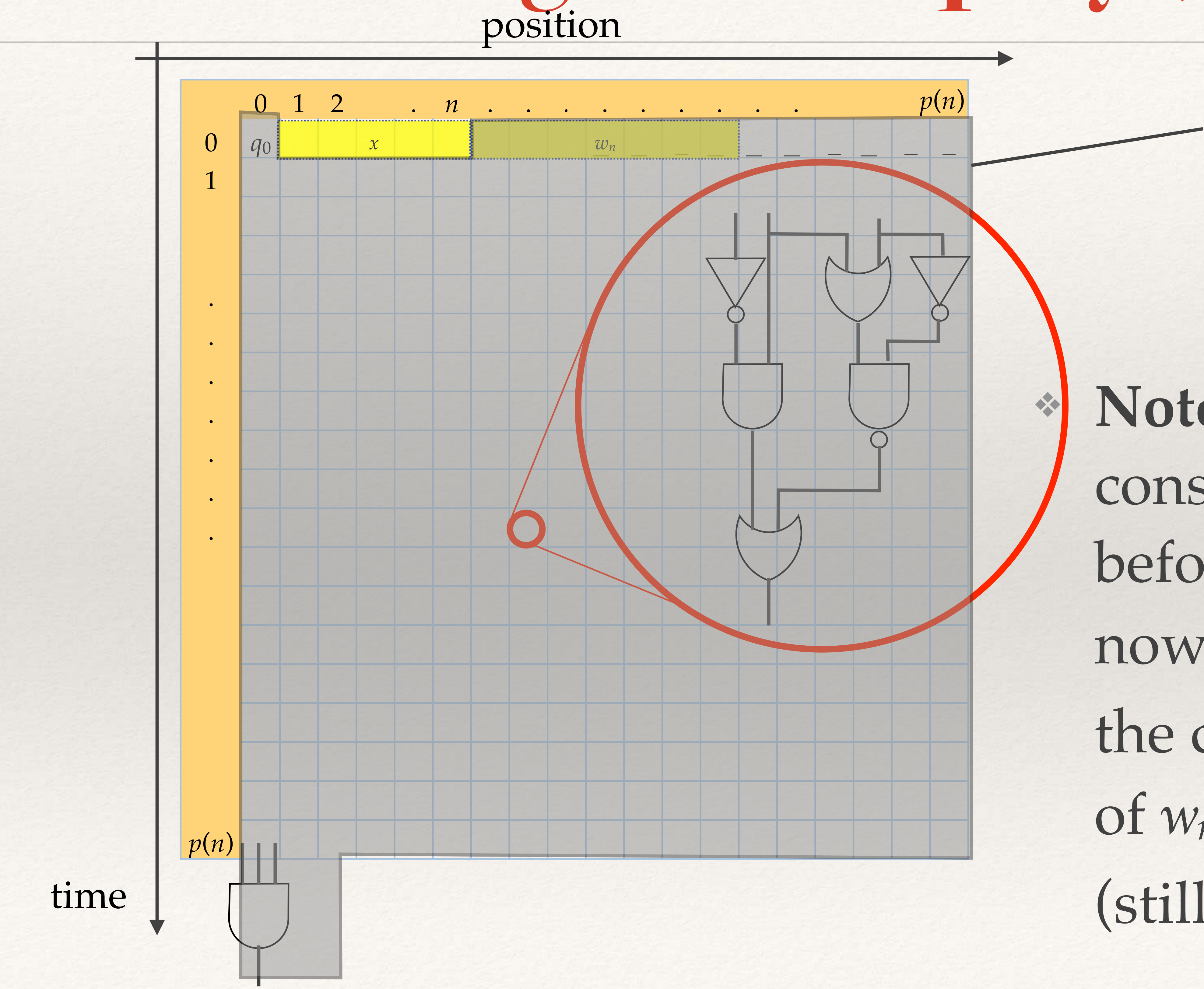
(see next slide)

Advice strings and **P/poly** (2/2)



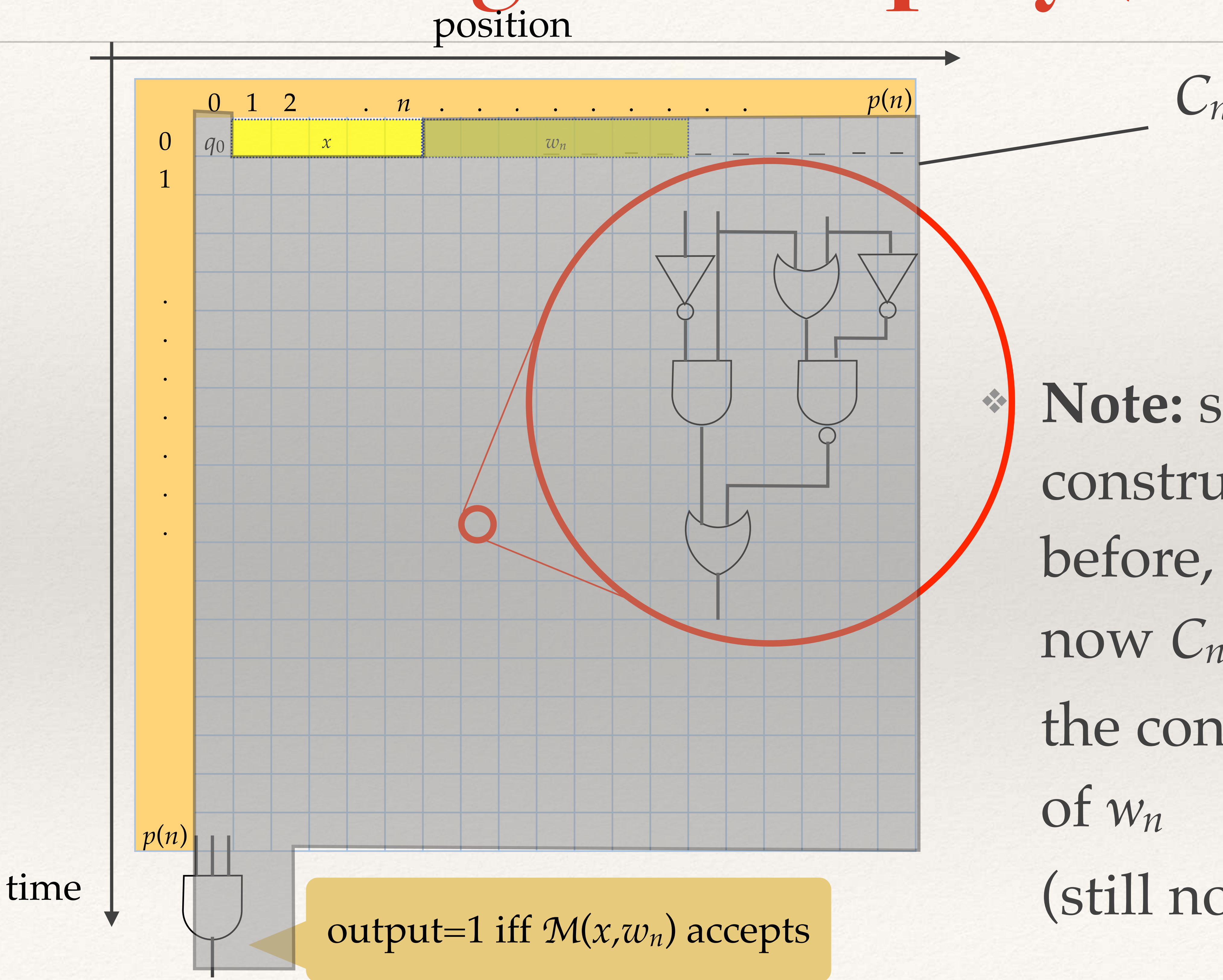
- **Note:** same construction as before, except… now C_n includes the constant bits of w_n (still not x.)

Advice strings and **P/poly** (2/2)



❖ **Note:** same construction as before, except… now C_n includes the constant bits of w_n (still not x.)

Advice strings and **P/poly** (2/2)



- **Note:** same construction as before, except… now C_n includes the constant bits of w_n (still not x.)

Adleman's Theorem

Adleman's Theorem

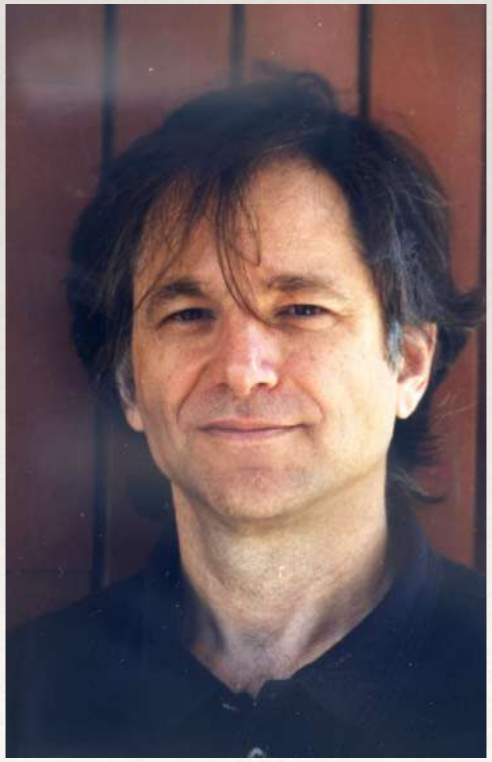
- **Theorem (Prop. 1.20). BPP $\subseteq$ P/poly.**

DOI: 10.1109/SFCS.1978.37 • Corpus ID: 15176763

Two theorems on random polynomial time

L. Adleman • Published 1978 • Computer Science •
19th Annual Symposium on Foundations of Computer Science (sfcs 1978)

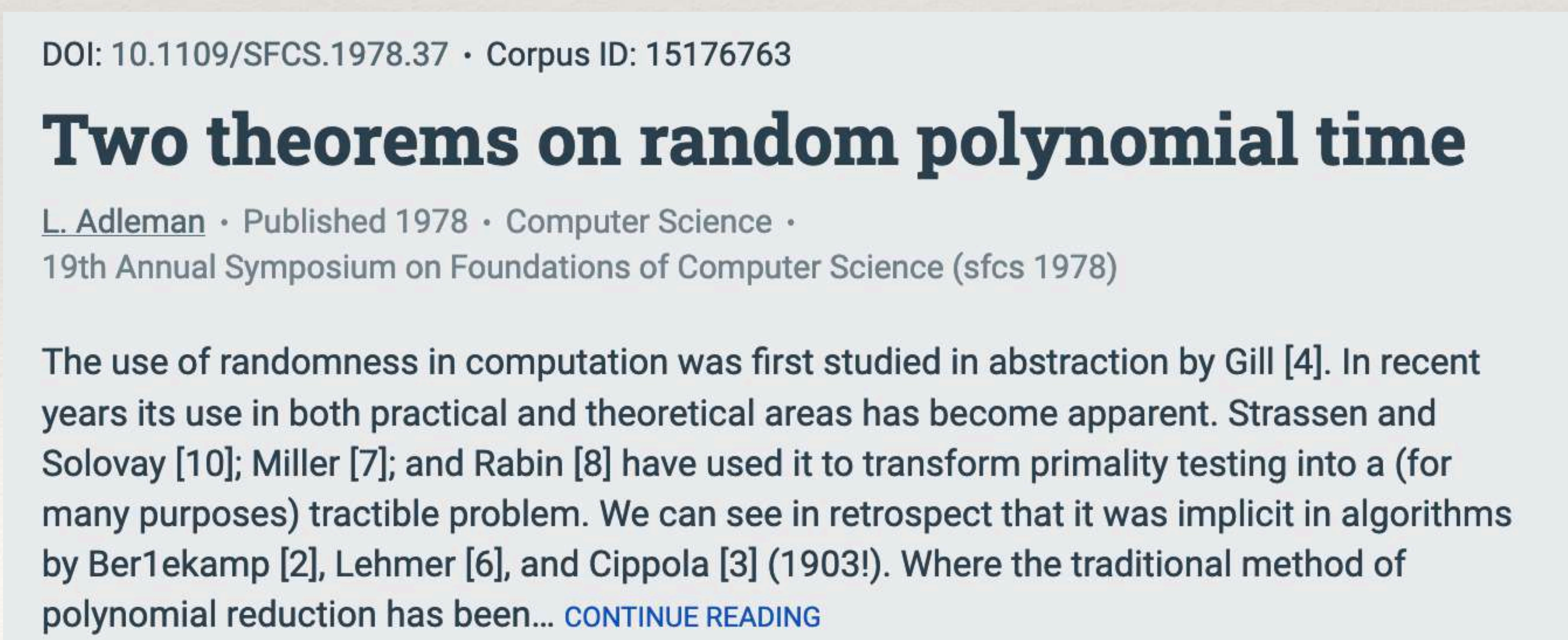
The use of randomness in computation was first studied in abstraction by Gill [4]. In recent years its use in both practical and theoretical areas has become apparent. Strassen and Solovay [10]; Miller [7]; and Rabin [8] have used it to transform primality testing into a (for many purposes) tractible problem. We can see in retrospect that it was implicit in algorithms by Ber1ekamp [2], Lehmer [6], and Cippola [3] (1903!). Where the traditional method of polynomial reduction has been... CONTINUE READING



https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Len-mankin-pic.jpg/440px-Len-mankin-pic.jpg

Adleman's Theorem

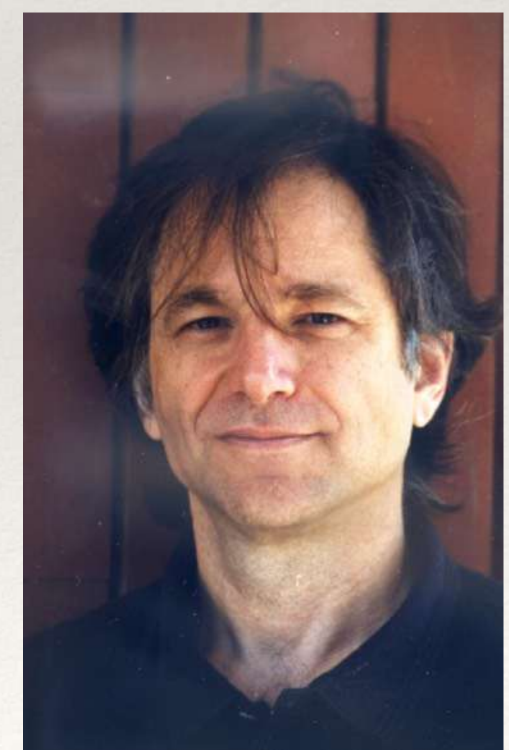
- **Theorem (Prop. 1.20). BPP $\subseteq$ P/poly.**
- Interestingly, we will be able to show the **existence** of the circuits C_n, (or the advice strings) but we won't be able to **compute** them (efficiently).


DOI: 10.1109/SFCS.1978.37 • Corpus ID: 15176763

Two theorems on random polynomial time

L. Adleman • Published 1978 • Computer Science •
19th Annual Symposium on Foundations of Computer Science (sfcs 1978)

The use of randomness in computation was first studied in abstraction by Gill [4]. In recent years its use in both practical and theoretical areas has become apparent. Strassen and Solovay [10]; Miller [7]; and Rabin [8] have used it to transform primality testing into a (for many purposes) tractible problem. We can see in retrospect that it was implicit in algorithms by Ber1ekamp [2], Lehmer [6], and Cippola [3] (1903!). Where the traditional method of polynomial reduction has been... CONTINUE READING



https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Len-mankin-pic.jpg/440px-Len-mankin-pic.jpg

The proof of Adleman's Theorem (1/2)

- Let L be in **BPP**.

A language L is in **BPP** if and only if there is a **polynomial-time** TM $\mathcal{M}$ such that for every input x (of size n):

$$\Pr_r(\mathcal{M}(x,r) \text{ errs}) \leq \varepsilon.$$

error $\varepsilon = 1/2^{q(n)}$

The proof of Adleman's Theorem (1/2)

- Let L be in **BPP**.
- Among the tapes r (of size $p(n)$), is there one such that

  for **every** x of size n, $\mathcal{M}(x,r)$ **always** gives the correct answer?

A language L is in **BPP** if and only if there is a **polynomial-time** TM $\mathcal{M}$ such that for every input x (of size n):

$$\Pr_r(\mathcal{M}(x,r) \text{ errs}) \leq \varepsilon.$$

error $\varepsilon = 1/2^{q(n)}$

The proof of Adleman's Theorem (1/2)

- Let L be in **BPP**.
- Among the tapes r (of size $p(n)$), is there one such that

  for **every** x of size n, $\mathcal{M}(x,r)$ **always** gives the correct answer?

- Let us use the probabilistic method…

A language L is in **BPP** if and only if there is a **polynomial-time** TM $\mathcal{M}$ such that for every input x (of size n):

$$\Pr_r(\mathcal{M}(x,r) \text{ errs}) \leq \varepsilon.$$

error $\varepsilon = 1/2^{q(n)}$

The proof of Adleman's Theorem (1/2)

- Let L be in **BPP**.
- Among the tapes r (of size $p(n)$), is there one such that

  for **every** x of size n, $\mathcal{M}(x,r)$ **always** gives the correct answer?

- Let us use the probabilistic method…

A language L is in **BPP** if and only if there is a **polynomial-time** TM $\mathcal{M}$ such that for every input x (of size n):

$$\Pr_r\,(\mathcal{M}(x,r) \text{ errs}) \leq \varepsilon.$$

error $\varepsilon = 1/2^{q(n)}$

- $\Pr_r(\exists\, x$ of size n, $\mathcal{M}(x,r)$ errs$)$

  $\leq \Sigma_x \Pr_r(\mathcal{M}(x,r)$ errs$)$

  $\leq 2^{n-q(n)}$

The proof of Adleman's Theorem (1/2)

A language L is in **BPP** if and only if there is a **polynomial-time** TM $\mathcal{M}$ such that for every input x (of size n):

$$\Pr_r\ (\mathcal{M}(x,r) \text{ errs}) \leq \ \varepsilon.$$

error $\varepsilon = 1/2^{q(n)}$

- Let L be in **BPP**.
- Among the tapes r (of size $p(n)$), is there one such that

  for **every** x of size n, $\mathcal{M}(x,r)$ **always** gives the correct answer?

- Let us use the probabilistic method…

- $\Pr_r(\exists\ x$ of size n, $\mathcal{M}(x,r)$ errs$)$

  $\leq \Sigma_x \Pr_r(\mathcal{M}(x,r) \text{ errs})$

  $\leq 2^{n-q(n)}$

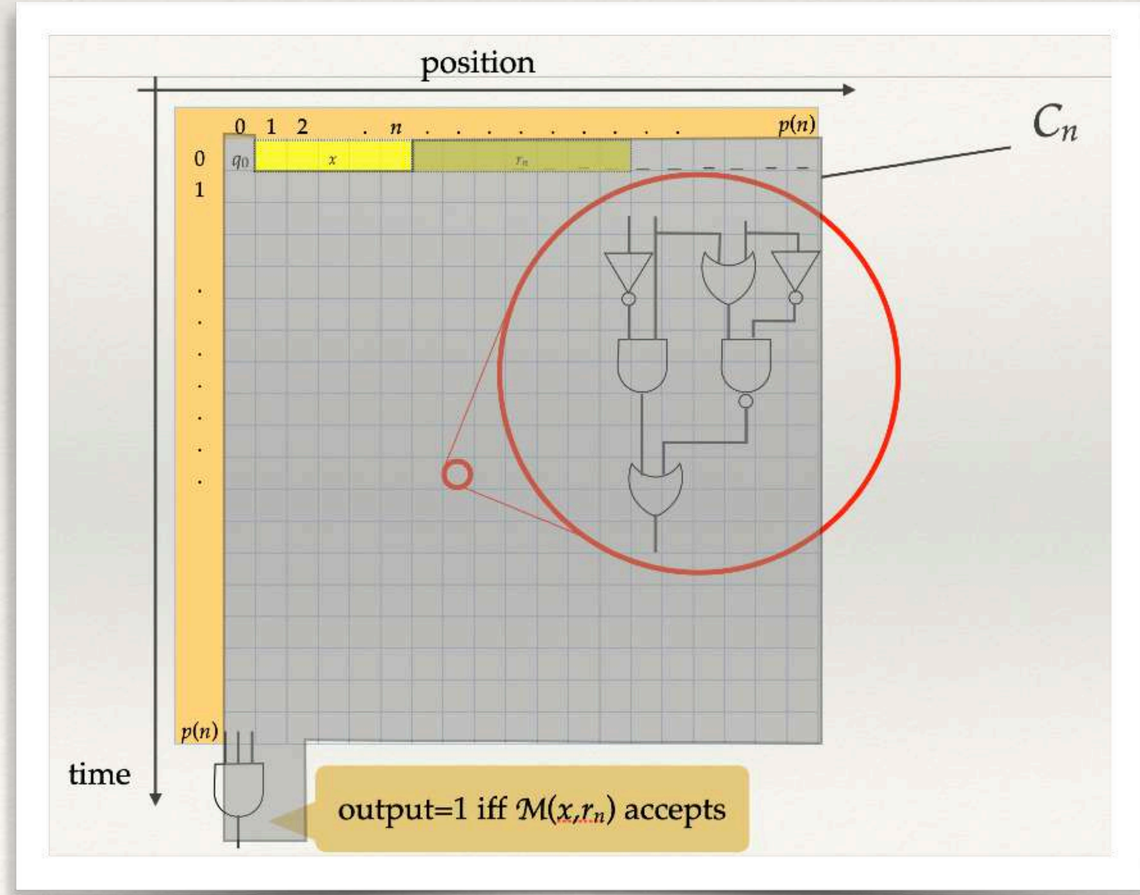
- … < 1 if we had the good taste to pick $q(n)=n+1$, say.

The proof of Adleman's Theorem (2/2)

- Let L be in **BPP**.
For each size n, there is a tape r_n (of size $p(n)$) such that for **every** x of size n, $\mathcal{M}(x,r_n)$ gives the correct answer, i.e.:
— if $x \in L$ then $\mathcal{M}(x,r_n)$ accepts
— if $x \notin L$ then $\mathcal{M}(x,r_n)$ rejects.

The proof of Adleman's Theorem (2/2)

- Let L be in **BPP**.
  For each size n, there is a tape r_n (of size $p(n)$) such that for **every** x of size n, $\mathcal{M}(x,r_n)$ gives the correct answer, i.e.:
  — if $x \in L$ then $\mathcal{M}(x,r_n)$ accepts
  — if $x \notin L$ then $\mathcal{M}(x,r_n)$ rejects.
- … Just use r_n as advice string! □



The Karp-Lipton Theorems, and consequences

(Yes, them again!)

Corpus ID: 115398060

Turing machines that take advice

R. Karp, R. J. Lipton · Published 1982 · Computer Science

https://upload.wikimedia.org/wikipedia/commons/thumb/3/3e/Karp_mg_7725-b.cr2.jpg/520px-Karp_mg_7725-b.cr2.jpg

https://cyber.gatech.edu/sites/default/files/styles/faculty_bio_pic/public/dick-lipton_1.jpg?itok=EkU43aPB

$\mathbf{co}C$

- Recall that $\Pi^{\mathrm{p}}_{k}=\mathbf{co}\Sigma^{\mathrm{p}}_{k}$ for every $k\geq 1$.
(**co**C is the class of complements of languages of C.)

co$C$

- Recall that $\Pi^{\mathrm{p}}_k = \mathbf{co}\Sigma^{\mathrm{p}}_k$ for every $k \geq 1$.
  (**co**C is the class of complements of languages of C.)
- **Fact.** **co** is monotonic: if $C \subseteq C'$, then $\mathbf{co}C \subseteq \mathbf{co}C'$.

co$C$

- Recall that Π^{p}_k=**co**Σ^{p}_k= for every $k\geq1$.
(**co**C is the class of complements of languages of C.)
- **Fact. co** is monotonic: if $C\subseteq C'$, then **co**$C\subseteq$**co**C'.
- (Already argued last time, as part of the Sipser-Gács-Lautemann theorem.)

$\mathbf{co}C$

- **Claim.** For any class C, the following are equivalent:

1. $C = \mathbf{co}C$
2. $C \subseteq \mathbf{co}C$
3. $\mathbf{co}C \subseteq C$.

co$C$

- **Claim.** For any class C, the following are equivalent:

  1. $C = \mathbf{co}C$
  2. $C \subseteq \mathbf{co}C$
  3. $\mathbf{co}C \subseteq C$.

- $2 \Rightarrow 3$: let L in $\mathbf{co}C$.

  Its complement is in C, hence in $\mathbf{co}C$ by 2.

  Therefore L is also in C.

$\mathbf{co}C$

- **Claim.** For any class C, the following are equivalent:
  1. $C = \mathbf{co}C$
  2. $C \subseteq \mathbf{co}C$
  3. $\mathbf{co}C \subseteq C$.
- $2 \Rightarrow 3$: let L in $\mathbf{co}C$.

  Its complement is in C, hence in $\mathbf{co}C$ by 2.

  Therefore L is also in C.
- $3 \Rightarrow 2$, and therefore $3 \Rightarrow 1$: similar. $1 \Rightarrow 2$: obvious. $\square$

Does PH collapse?

- We say that **PH** collapses at level 2 iff $\Sigma^p_2 = \Pi^p_2$.
  By the previous claim, equivalent to $\Pi^p_2 \subseteq \Sigma^p_2$.
- **Prop.** If $\Sigma^p_2 = \Pi^p_2$ then
  $\Sigma^p_2 = \Pi^p_2 = \Sigma^p_3 = \Pi^p_3 = \Sigma^p_4 = \ldots =$ **PH** (whence the name.)

Does PH collapse?

- We say that **PH** collapses at level 2 iff $\Sigma^p_2 = \Pi^p_2$.
  By the previous claim, equivalent to $\Pi^p_2 \subseteq \Sigma^p_2$.
- **Prop.** If $\Sigma^p_2 = \Pi^p_2$ then
  $\Sigma^p_2 = \Pi^p_2 = \Sigma^p_3 = \Pi^p_3 = \Sigma^p_4 = \ldots =$ **PH** (whence the name.)
- *Proof sketch.* Let $\exists \cdot C$ be the class of the languages
  $\{x \mid \exists y$ of poly size, $(x,y) \in L'\}$, $L' \in C$.

Does PH collapse?

- We say that **PH** <u>collapses at level 2</u> iff $\Sigma^p_2 = \Pi^p_2$. By the previous claim, equivalent to $\Pi^p_2 \subseteq \Sigma^p_2$.
- **Prop.** If $\Sigma^p_2 = \Pi^p_2$ then
  $\Sigma^p_2 = \Pi^p_2 = \Sigma^p_3 = \Pi^p_3 = \Sigma^p_4 = \ldots = \mathbf{PH}$ (whence the name.)
- *Proof sketch.* Let $\exists\cdot C$ be the class of the languages
  $\{x \mid \exists y \text{ of poly size, } (x,y) \in L'\}$, $L' \in C$.
- $\Sigma^p_3 = \exists\cdot\Pi^p_2 = \exists\cdot\Sigma^p_2 = \exists\cdot\exists\cdot\mathbf{coNP} = \exists\cdot\mathbf{coNP} = \Sigma^p_2$, then
  $\Pi^p_3 = \mathbf{co}\Sigma^p_3 = \mathbf{co}\Sigma^p_2 = \Pi^p_2 = \Sigma^p_2$, etc. $\square$

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P{}_2 \subseteq \Sigma^P{}_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y \text{ of size } p(n), (x,y) \in L'\}$, $L' \in$ **NP**.

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y \text{ of size } p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so
- $L = \{x \mid \forall y$ of size $p(n), C_{\text{size}(x,y)}[(x,y)]=1\}$

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so
- $L = \{x \mid \forall y$ of size $p(n), C_{\mathrm{size}(x,y)}[(x,y)]=1\}$
- $= \{x \mid \exists$poly size $C, \forall y$ of size $p(n), C[(x,y)]=1\}$
  $\in \Sigma^P_2$.

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so
- $L = \{x \mid \forall y$ of size $p(n), C_{\mathrm{size}(x,y)}[(x,y)]=1\}$
- $= \{x \mid \exists$poly size $C, \forall y$ of size $p(n), C[(x,y)]=1\}$
  $\in \Sigma^P_2$.

We can permute quantifiers,
because $C_{\mathrm{size(x,y)}}=C_{n+p(n)+3}$ does **not** depend on y.

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^P_2 \subseteq \Sigma^P_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^P_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so
- $L = \{x \mid \forall y$ of size $p(n), C_{\text{size}(x,y)}[(x,y)]=1\}$

  Where is the bug?
- $= \{x \mid \exists$poly size $C, \forall y$ of size $p(n), C[(x,y)]=1\}$
  $\in \Sigma^P_2$.

We can permute quantifiers,
because $C_{\text{size(x,y)}}=C_{n+p(n)+3}$ does **not** depend on y.

The first Karp-Lipton theorem

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- Let me give you a **wrong** argument first. (We will repair it later.)
- Let $L \in \Pi^p_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- L' has polynomial circuits C_n, so
- $L = \{x \mid \forall y$ of size $p(n), C_{\text{size}(x,y)}[(x,y)]{=}1\}$
- $= \{x \mid \exists$poly size $C, \forall y$ of size $p(n), C[(x,y)]{=}1\}$
  $\in \Sigma^p_2$.

Hint: this is Σ^*, not L
(just take the constant circuit 1 for C here)

We can permute quantifiers,
because $C_{\text{size(x,y)}}{=}C_{n+p(n)+3}$ does **not** depend on y.

The bug

- $L = \{x \mid \forall y \text{ of size } p(n),\ C_{\mathrm{size}(x,y)}[(x,y)]=1\}$

  $\neq \{x \mid \exists \text{poly size } C, \forall y \text{ of size } p(n),\ C[(x,y)]=1\}$:

  here we trust some divine (all-powerful) being Merlin to give us the magical circuit $C_{\mathrm{size}(x,y)}$ for C…

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

The bug

- $L = \{x \mid \forall y \text{ of size } p(n),\ C_{\text{size}(x,y)}[(x,y)]{=}1\}$
  $\neq \{x \mid \exists \text{poly size } C,\ \forall y \text{ of size } p(n),\ C[(x,y)]{=}1\}$:
  here we trust some divine (all-powerful) being Merlin to give us the magical circuit $C_{\text{size}(x,y)}$ for C…

- … but what prevents it from **cheating**?
  We must **check** that the circuit C it gives us does the job.

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set *S*,
  and you ask Merlin: « is *S* satisfiable? »
- Merlin answers: « yes »
- What can you conclude?

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set S,
  and you ask Merlin: « is S satisfiable? »
- Merlin answers: « yes »
- What can you conclude?
- Of course, nothing.

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set *S*,
  and you ask Merlin: « is *S* satisfiable?
  give me a satisfying assignment ϱ»

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set *S*,
  and you ask Merlin: « is *S* satisfiable?
  give me a satisfying assignment ϱ»

- Merlin answers: ~~« yes »~~ ϱ

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set S,
  and you ask Merlin: « is S satisfiable?
  give me a satisfying assignment ϱ»

- Merlin answers: ~~« yes »~~ ϱ
- You check $\varrho \vDash S$, **accept** if this is true, **reject** otherwise.

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

A thought experiment

- Imagine you want to solve **SAT**.
  You are given a clause set S,
  and you ask Merlin: « is S satisfiable?
  give me a satisfying assignment ϱ»

- Merlin answers: ~~« yes »~~ ϱ
- You check $\varrho \models S$, **accept** if this is true, **reject** otherwise.
- If S satisfiable, then Merlin can make you accept.
  Otherwise, you will necessarily reject.

https://www.ecranlarge.com/uploads/image/001/011/merlin-l-enchanteur-photo-merlin-disney-1011190.jpg

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ
- You retort that **SAT** is **self-reducible**:
  Given an oracle O deciding satisfiability,
  one can compute ϱ such that $\varrho \models S$ (if any).

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ
- You retort that **SAT** is **self-reducible**:
  Given an oracle O deciding satisfiability,
  one can compute ϱ such that $\varrho \models S$ (if any).

$S[A_1{:=}1]$ sat? (using O)

yes — no

yes	no
set $A_1{:=}1$	set $A_1{:=}0$
$S_1{:=}S[A_1{:=}1]$	$S_1{:=}S[A_1{:=}0]$

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ
- You retort that **SAT** is **self-reducible**:
  Given an oracle O deciding satisfiability,
  one can compute ϱ such that $\varrho \models S$ (if any).

$S[A_1{:=}1]$ sat? (using O)

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$S_1[A_2{:=}1]$ sat? (using O)

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ
- You retort that **SAT** is **self-reducible**:
  Given an oracle O deciding satisfiability,
  one can compute ϱ such that $\varrho \models S$ (if any).

$S[A_1{:=}1]$ sat? (using O)

yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$

no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$S_1[A_2{:=}1]$ sat? (using O)

yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$

no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$S_2[A_3{:=}1]$ sat? (using O)

yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$

no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

Self-reducibility

- Now Merlin complains he can only decide whether S is satisfiable (using circuits C_n), **not** find a satisfying ϱ
- You retort that **SAT** is **self-reducible**:
  Given an oracle O deciding satisfiability,
  one can compute ϱ such that $\varrho \models S$ (if any).

$S[A_1{:=}1]$ sat? (using O) — yes: set $A_1{:=}1$, $S_1{:=}S[A_1{:=}1]$ — no: set $A_1{:=}0$, $S_1{:=}S[A_1{:=}0]$

$S_1[A_2{:=}1]$ sat? (using O) — yes: set $A_2{:=}1$, $S_2{:=}S_1[A_2{:=}1]$ — no: set $A_2{:=}0$, $S_2{:=}S_1[A_2{:=}0]$

$S_2[A_3{:=}1]$ sat? (using O) — yes: set $A_3{:=}1$, $S_3{:=}S_2[A_3{:=}1]$ — no: set $A_3{:=}0$, $S_3{:=}S_2[A_3{:=}0]$

etc.

Call this the « **self-reducibility machine** »

Self-reducibility

- Instead of an oracle O, Merlin will use circuits C_m on clause sets S, S_1, S_2, …, of various sizes m.

$S[A_1{:=}1]$ sat? (using O)

set $A_1{:=}1$ — $S_1{:=}S[A_1{:=}1]$

set $A_1{:=}0$ — $S_1{:=}S[A_1{:=}0]$

$S_1[A_2{:=}1]$ sat? (using O)

set $A_2{:=}1$ — $S_2{:=}S_1[A_2{:=}1]$

set $A_2{:=}0$ — $S_2{:=}S_1[A_2{:=}0]$

$S_2[A_3{:=}1]$ sat? (using O)

set $A_3{:=}1$ — $S_3{:=}S_2[A_3{:=}1]$

set $A_3{:=}0$ — $S_3{:=}S_2[A_3{:=}0]$

etc.

Call this the « **self-reducibility machine** »

Self-reducibility

- Instead of an oracle O, Merlin will use circuits C_m on clause sets S, S_1, S_2, …, of various sizes m.
- m is bounded by n=size(S)
  (e.g., $S[A:=1]$ is obtained by removing clauses in which $+A$ appears, and removing $-A$ in the remaining clauses)

$S[A_1:=1]$ sat? (using O)

set $A_1:=1$
$S_1:=S[A_1:=1]$

set $A_1:=0$
$S_1:=S[A_1:=0]$

$S_1[A_2:=1]$ sat? (using O)

set $A_2:=1$
$S_2:=S_1[A_2:=1]$

set $A_2:=0$
$S_2:=S_1[A_2:=0]$

$S_2[A_3:=1]$ sat? (using O)

set $A_3:=1$
$S_3:=S_2[A_3:=1]$

set $A_3:=0$
$S_3:=S_2[A_3:=0]$

etc.

Call this the « **self-reducibility machine** »

A circuit for self-reducibility

- Now given (net-lists for) C_0, C_1, …, C_n as **advice** $w_{0\ldots n}$

A circuit for self-reducibility

- Now given (net-lists for) C_0, C_1, …, C_n as **advice** $w_{0\ldots n}$
- the self-reducibility machine is a poly time TM h taking (S, w) as input
— returning an environment ϱ
— satisfying S, if S is satisfiable and Merlin is **honest** (i.e., plays using the above advice $w_{0\ldots n}$ for w)

A circuit for self-reducibility

- Now given (net-lists for) C_0, C_1, …, C_n as **advice** $w_{0\ldots n}$
- the self-reducibility machine is a poly time TM h taking (S, w) as input
  — returning an environment ϱ
  — satisfying S, if S is satisfiable and Merlin is **honest** (i.e., plays using the above advice $w_{0\ldots n}$ for w)
- Note that, if size(C_n)=O(n^k) (poly), then

  size($w_{0\ldots n}$)=O(n^{k+1}) (poly again)

A circuit for self-reducibility

- Now given (net-lists for) C_0, C_1, …, C_n as **advice** $w_{0\ldots n}$
- the self-reducibility machine is a poly time TM h taking (S, w) as input
  — returning an environment ϱ
  — satisfying S, if S is satisfiable and Merlin is **honest**
  (i.e., plays using the above advice $w_{0\ldots n}$ for w)
- Note that, if size(C_n)=O(n^k) (poly), then
  size($w_{0\ldots n}$)=O(n^{k+1}) (poly again)

by the way, not quite the trick used in the lecture notes

Karp-Lipton: the proof (1/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^{P}_{2} \subseteq \Sigma^{P}_{2}$.
- Let $L \in \Pi^{P}_{2}$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.

Karp-Lipton: the proof (1/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- Let $L \in \Pi^p_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- We reduce to **SAT**
  (this will allow us to use self-reducibility!):

Karp-Lipton: the proof (1/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- Let $L \in \Pi^p_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- We reduce to **SAT**
  (this will allow us to use self-reducibility!):
- there is a polytime function f / $(x,y) \in L' \Leftrightarrow f(x,y) \in$ **SAT**

Karp-Lipton: the proof (1/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- Let $L \in \Pi^p_2$ be $\{x \mid \forall y$ of size $p(n), (x,y) \in L'\}$, $L' \in$ **NP**.
- We reduce to **SAT**
  (this will allow us to use self-reducibility!):
- there is a polytime function f / $(x,y) \in L' \Leftrightarrow f(x,y) \in$ **SAT**
- Hence $L = \{x \mid \forall y$ of size $p(n)$, $f(x,y) \in$ **SAT**$\}$

Karp-Lipton: the proof (2/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- $L = \{x \mid \forall y$ of size $p(n), f(x,y) \in \mathbf{SAT}\}$ (from last slide)

Karp-Lipton: the proof (2/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p{}_2 \subseteq \Sigma^p{}_2$.
- $L = \{x \mid \forall y$ of size $p(n), f(x,y) \in \mathbf{SAT}\}$ (from last slide)
- Now use self-reducibility:
  $L = \{x \mid \forall y$ of size $p(n), h(f(x,y), w_{0\ldots\text{size}(f(x,y))}) \vDash f(x,y)\}$

Karp-Lipton: the proof (2/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^{p}_{2} \subseteq \Sigma^{p}_{2}$.
- $L = \{x \mid \forall y$ of size $p(n)$, $f(x,y) \in \mathbf{SAT}\}$ (from last slide)
- Now use self-reducibility:

  a clause set S

  $L = \{x \mid \forall y$ of size $p(n)$, $h(f(x,y), w_{0\ldots\mathrm{size}(f(x,y))}) \models f(x,y)\}$

Karp-Lipton: the proof (2/3)

- **Theorem (Prop. 1.21).** If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then the polynomial hierarchy collapses at level 2: $\Pi^{p}{}_2 \subseteq \Sigma^{p}{}_2$.
- $L = \{x \mid \forall y \text{ of size } p(n), f(x,y) \in \mathbf{SAT}\}$ (from last slide)
- Now use self-reducibility: $L = \{x \mid \forall y \text{ of size } p(n), h(f(x,y), w_{0\ldots\mathrm{size}(f(x,y))}) \vDash f(x,y)\}$

a clause set S

the « self-reducibility machine »

Karp-Lipton: the proof (2/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p{}_2 \subseteq \Sigma^p{}_2$.
- $L = \{x \mid \forall y$ of size $p(n)$, $f(x,y) \in \mathbf{SAT}\}$ (from last slide)
- Now use self-reducibility:

  a clause set S

  $L = \{x \mid \forall y$ of size $p(n)$, $h(f(x,y), w_{0\ldots\mathrm{size}(f(x,y))}) \vDash f(x,y)\}$

  the « self-reducibility machine »

  size of advice polynomial in n=size(x)

Karp-Lipton: the proof (3/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- $L = \{x \mid \forall y$ of size $p(n), h(f(x,y), w_{0\ldots\text{size}(f(x,y))}) \models f(x,y)\}$ (last slide)

Karp-Lipton: the proof (3/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p{}_2 \subseteq \Sigma^p{}_2$.
- $L = \{x \mid \forall y$ of size $p(n),\ h(f(x,y), w_{0\ldots\text{size}(f(x,y))}) \vDash f(x,y)\}$ (last slide)
- I claim that $L = \{x \mid \exists w, \forall y$ of size $p(n),\ h(f(x,y),w) \vDash f(x,y)\}$

  (huh? that was the bug, right? No, we now **check** that $h(\ldots) \vDash f(x,y)$!)

Karp-Lipton: the proof (3/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p{}_2 \subseteq \Sigma^p{}_2$.
- $L = \{x \mid \forall y$ of size $p(n)$, $h(f(x,y), w_{0\ldots\text{size}(f(x,y))}) \vDash f(x,y)\}$ (last slide)
- I claim that $L = \{x \mid \exists w, \forall y$ of size $p(n)$, $h(f(x,y),w) \vDash f(x,y)\}$

  (huh? that was the bug, right? No, we now **check** that $h(\ldots) \vDash f(x,y)$!)
- If $x \in L$, then take $w = w_{0\ldots\text{size}(f(x,y))}$: $\forall y$, $h(f(x,y),w) \vDash f(x,y)$ ✔

Karp-Lipton: the proof (3/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- $L = \{x \mid \forall y$ of size $p(n), h(f(x,y), w_{0\ldots\mathrm{size}(f(x,y))}) \vDash f(x,y)\}$ (last slide)
- I claim that $L = \{x \mid \exists w, \forall y$ of size $p(n), h(f(x,y),w) \vDash f(x,y)\}$

  (huh? that was the bug, right? No, we now **check** that $h(\ldots) \vDash f(x,y)$!)
- If $x \in L$, then take $w{=}w_{0\ldots\mathrm{size}(f(x,y))}$: $\forall y, h(f(x,y),w) \vDash f(x,y)$ ✔
- If $x \notin L$, $\exists y, f(x,y)$ is **unsatisfiable**…

  hence whichever w we take, $h(f(x,y),w) \nvDash f(x,y)$ ✔

Karp-Lipton: the proof (3/3)

- **Theorem (Prop. 1.21).** If **NP** $\subseteq$ **P/poly**, then the polynomial hierarchy collapses at level 2: $\Pi^p_2 \subseteq \Sigma^p_2$.
- $L = \{x \mid \forall y$ of size $p(n)$, $h(f(x,y), w_{0\ldots\text{size}(f(x,y))}) \vDash f(x,y)\}$ (last slide)
- I claim that $L = \{x \mid \exists w, \forall y$ of size $p(n)$, $h(f(x,y),w) \vDash f(x,y)\}$

  (huh? that was the bug, right? No, we now **check** that $h(\ldots) \vDash f(x,y)$!)

  in Σ^p_2
- If $x \in L$, then take $w = w_{0\ldots\text{size}(f(x,y))}$: $\forall y$, $h(f(x,y),w) \vDash f(x,y)$ ✔
- If $x \notin L$, $\exists y$, $f(x,y)$ is **unsatisfiable**…

  hence whichever w we take, $h(f(x,y),w) \nvDash f(x,y)$ ✔

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{P/poly}$.

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- By previous result, it suffices to show $\Sigma^{p}_{2} \subseteq$ **P/poly**.

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{P/poly}$.
- By previous result, it suffices to show $\Sigma^{\mathrm{P}}_2 \subseteq \mathbf{P/poly}$.
- Let $L = \{x \mid \exists y \text{ of size } p(n), (x,y) \in L'\}$ where $L' \in \mathbf{coNP}$

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- By previous result, it suffices to show $\Sigma^{P}_{2} \subseteq$ **P/poly**.
- Let $L = \{x \mid \exists y$ of size $p(n), (x,y) \in L'\}$ where $L' \in$ **coNP**
- The complement of L' has poly size advice strings,
  hence L' also has poly size advice strings w_n

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{P/poly}$.
- By previous result, it suffices to show $\Sigma^{\mathrm{P}}_2 \subseteq \mathbf{P/poly}$.
- Let $L = \{x \mid \exists y \text{ of size } p(n), (x,y) \in L'\}$ where $L' \in \mathbf{coNP}$
- The complement of L' has poly size advice strings, hence L' also has poly size advice strings w_n
- $L = \{x \mid \exists y \text{ of size } p(n), \mathcal{M}((x,y), w_{\mathrm{size}(x,y)}) \text{ accepts}\}$ for some poly time TM $\mathcal{M}$.

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- $L = \{x \mid \exists y$ of size $p(n)$, $\mathcal{M}((x,y), w_{\text{size}(x,y)})$ accepts$\}$
  for some poly time TM $\mathcal{M}$ (from last slide)

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- $L = \{x \mid \exists y$ of size $p(n)$, $\mathcal{M}((x,y), w_{\text{size}(x,y)})$ accepts$\}$
  for some poly time TM $\mathcal{M}$ (from last slide)
- Let $L'' = \{(x,w) \mid \exists y$ of size $p(\text{size}(x))$, $\mathcal{M}((x,y), w)$ accepts$\}$
  This is in **NP**, hence has polynomial circuits C_n, too!

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- $L = \{x \mid \exists y$ of size $p(n)$, $\mathcal{M}((x,y), w_{\text{size}(x,y)})$ accepts$\}$
  for some poly time TM $\mathcal{M}$ (from last slide)
- Let $L'' = \{(x,w) \mid \exists y$ of size $p(\text{size}(x))$, $\mathcal{M}((x,y), w)$ accepts$\}$
  This is in **NP**, hence has polynomial circuits C_n, too!
- So $L = \{x \mid C_{\text{appropriate size}}[(x, w_{\text{size}(x,y)})]=1\}$

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- $L = \{x \mid \exists y$ of size $p(n)$, $\mathcal{M}((x,y), w_{\text{size}(x,y)})$ accepts$\}$
  for some poly time TM $\mathcal{M}$ (from last slide)
- Let $L'' = \{(x,w) \mid \exists y$ of size $p(\text{size}(x))$, $\mathcal{M}((x,y), w)$ accepts$\}$
  This is in **NP**, hence has polynomial circuits C_n, too!
- So $L = \{x \mid C_{\text{appropriate size}}[(x, w_{\text{size}(x,y)})]=1\}$

size of x + cst + size of $w_{\text{size}(x,y)}$…
polynomial in n=size(x)

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{P/poly}$.
- So $L = \{x \mid C_{\text{appropriate size}}[(x, w_{\text{size}(x,y)})]=1\}$ (from last slide)

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- So $L = \{x \mid C_{\text{appropriate size}}[(x, w_{\text{size}(x,y)})]{=}1\}$ (from last slide)

size of x + cst + size of $w_{\text{size}(x,y)}$…
polynomial in n=size(x)

The second Karp-Lipton theorem

- **Theorem (Prop. 1.22).** If **NP** $\subseteq$ **P/poly**, then **PH** $\subseteq$ **P/poly**.
- So $L = \{x \mid C_{\text{appropriate size}}[(x, w_{\text{size}(x,y)})]=1\}$ (from last slide)

size of x + cst + size of $w_{\text{size}(x,y)}$…
polynomial in n=size(x)

- Hence L is decided by the circuits

$$C_{\text{appropriate size}}[(\_, w_{\text{size}(x,y)})] \qquad \square$$

(all sizes depending only on n=size(x), not on x itself)

Conclusion

BPP cannot be too large

- **Corollary.** If **BPP** contains **NP**, then:
  — **PH** collapses at level 2 (unlikely)
  — and is included in **P/poly**.

- *Proof.*

Adleman's Theorem

Theorem (Prop. 1.20). $\mathbf{BPP} \subseteq \mathbf{P/poly}$.

The first Karp-Lipton theorem

Theorem (Prop. 1.21). If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then the polynomial hierarchy collapses at level 2: $\Pi P_2 \subseteq \Sigma P_2$.

The second Karp-Lipton theorem

Theorem (Prop. 1.22). If $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} \subseteq \mathbf{P/poly}$.