CS21
Decidability and Tractability

Lecture 16-17

February 12-14, 2018

Outline

- The complexity class P
  - examples of problems in P
- The complexity class EXP
- Time Hierarchy Theorem
- hardness and completeness
- an EXP-complete problem
- the complexity class NP
  - alternate characterization of NP
  - 3-SAT is NP-complete

Time complexity

- Recall:
  - language is a set of strings
  - a complexity class is a set of languages
  - complexity classes we've seen:
    - Regular Languages, Context-Free Languages, Decidable Languages, RE Languages, co-RE languages

Definition: TIME(t(n)) = {L : there exists a TM M that decides L in time O(t(n))}

Time Complexity

- interested in a coarse classification of problems. For this purpose,
  - treat any polynomial running time as "efficient" or "tractable"
  - treat any exponential running time as inefficient or "intractable"

Key definition: "P" or "polynomial-time" is

$$P = \cup_{k \geq 1} \mathrm{TIME}(n^k)$$

A puzzle

- Find an efficient algorithm to solve the following problem:
- Input: sequence of pairs of symbols

  e.g. (A, b), (E, D), (d, C), (B, a)
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

A puzzle

- Find an efficient algorithm to solve the following problem.
- Input: sequence of pairs of symbols

  e.g. (A, b), (E, D), (d, C), (b, a)
- Goal: determine if it is possible to circle at least one symbol in each pair without circling upper and lower case of same symbol.

2SAT

- This is a disguised version of the language

  2SAT = {formulas in Conjunctive Normal Form with 2 literals per clause for which there exists a satisfying truth assignment}

  – CNF = “AND of ORs”

  (A, b), (E, D), (d, C), (b, a)

  $$(x_1 \vee \neg x_2) \wedge (x_5 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1)$$

  – satisfying truth assignment = assignment of TRUE/FALSE to each variable so that whole formula is TRUE

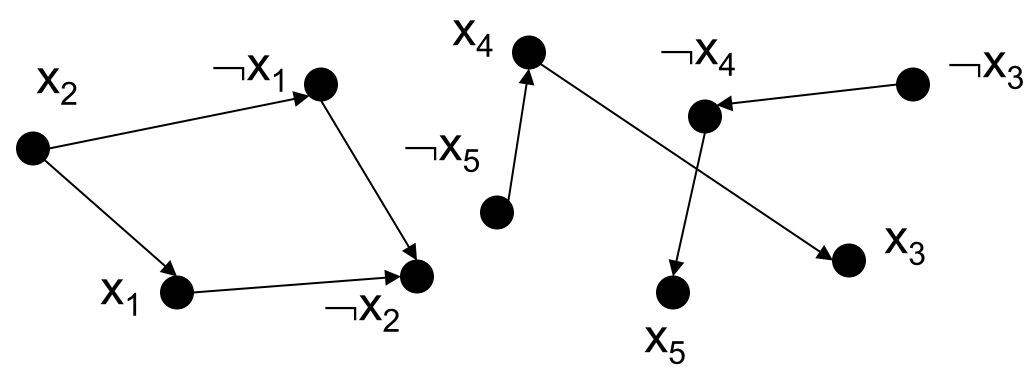
2SAT

<u>**Theorem**</u>: There is a polynomial-time algorithm deciding 2SAT ("2SAT 2 P").

Proof: algorithm described on next slides.

Algorithm for 2SAT

- Build a graph with separate nodes for each literal.
  - add directed edge (x, y) iff formula includes clause $(\neg x \vee y)$ or $(y \vee \neg x)$ (equiv. to $x \Rightarrow y$)



e.g. $(x_1 \vee \neg x_2) \wedge (x_5 \vee x_4) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_2 \vee \neg x_1)$

Algorithm for 2SAT

Claim: formula is unsatisfiable iff there is some variable x with a path from x to $\neg x$ and a path from $\neg x$ to x in derived graph.

- Proof ($\Leftarrow$)
  - edges represent implication $\Rightarrow$. By transitivity of $\Rightarrow$, a path from x to $\neg x$ means $x \Rightarrow \neg x$, and a path from $\neg x$ to x means $\neg x \Rightarrow x$.

Algorithm for 2SAT

- Proof ($\Rightarrow$)
  - to construct a satisfying assign. (if no x with a path from x to $\neg$x and a path from $\neg$x to x):
    - pick unassigned literal s with no path from s to $\neg$s
    - assign it TRUE, as well as all nodes reachable from it; assign negations of these literals FALSE
    - note: path from s to t and s to $\neg$t implies path from $\neg$t to $\neg$s and t to $\neg$s, implies path from s to $\neg$s
    - note: path s to t (assigned FALSE) implies path from $\neg$t (assigned TRUE) to $\neg$s, so s already assigned at that point.

Algorithm for 2SAT

- Algorithm:
  - build derived graph
  - for every pair x, $\neg$x check if there is a path from x to $\neg$x and from $\neg$x to x in the graph
- Running time of algorithm (input length n):
  - O(n) to build graph
  - O(n) to perform each check
  - O(n) checks
  - running time $O(n^2)$. 2SAT $\in$ P.

Another puzzle

- Find an efficient algorithm to solve the following problem.
- Input: sequence of *triples* of symbols

  e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a)
- Goal: determine if it is possible to circle at least one symbol in each *triple* without circling upper and lower case of same symbol.

3SAT

- This is a disguised version of the language

  3SAT = {formulas in Conjunctive Normal Form with 3 literals per clause for which there exists a satisfying truth assignment}

  e.g. (A, b, C), (E, D, b), (d, A, C), (c, b, a)

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

- observe that this language is in TIME(2^n)

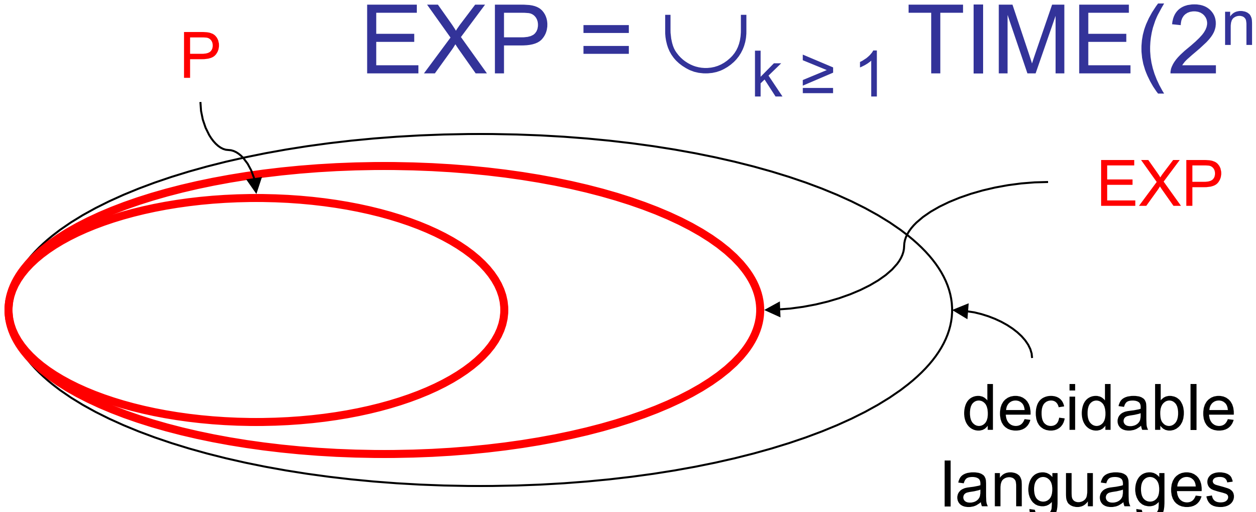
Time Complexity

Key definition: "P" or "polynomial-time" is

$$P = \cup_{k \geq 1} \text{TIME}(n^k)$$

Definition: "EXP" or "exponential-time" is

$$EXP = \cup_{k \geq 1} \text{TIME}(2^{n^k})$$



EXP

$$P = \cup_{k \geq 1} \mathrm{TIME}(n^k)$$

$$EXP = \cup_{k \geq 1} \mathrm{TIME}(2^{n^k})$$

- Note: $P \subseteq EXP$.
- We have seen 3SAT $\in$ EXP.
  - **does not rule out possibility that it is in P**
- Is P different from EXP?

Time Hierarchy Theorem

<u>Theorem</u>: For every *proper complexity function* $f(n) \geq n$:

$$\mathbf{TIME(f(n)) \not\supseteq TIME(f(2n)^3)}.$$

- Note: $P \subseteq TIME(2^n) \not\supseteq TIME(2^{(2n)3}) \subseteq EXP$
- Most natural functions (and 2^n in particular) are proper complexity functions. We will ignore this detail in this class.

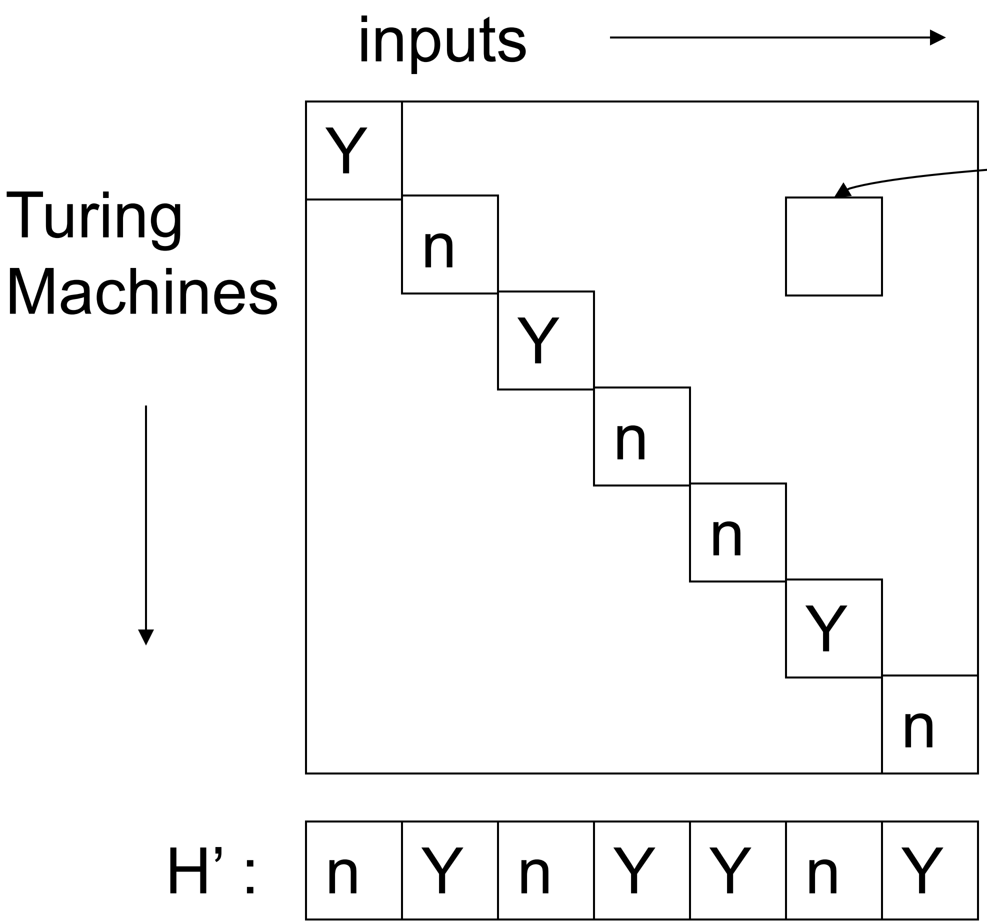
Time Hierarchy Theorem

<u>**Theorem**</u>: For every *proper complexity function* f(n) ≥ n:

$$\mathbf{TIME(f(n))} \not\subseteq \mathbf{TIME(f(2n)^3)}.$$

- Proof idea:
  - use diagonalization to construct a language that is not in TIME(f(n)).
  - constructed language comes with a TM that decides it and runs in time $f(2n)^3$.

Recall proof for Halting Problem

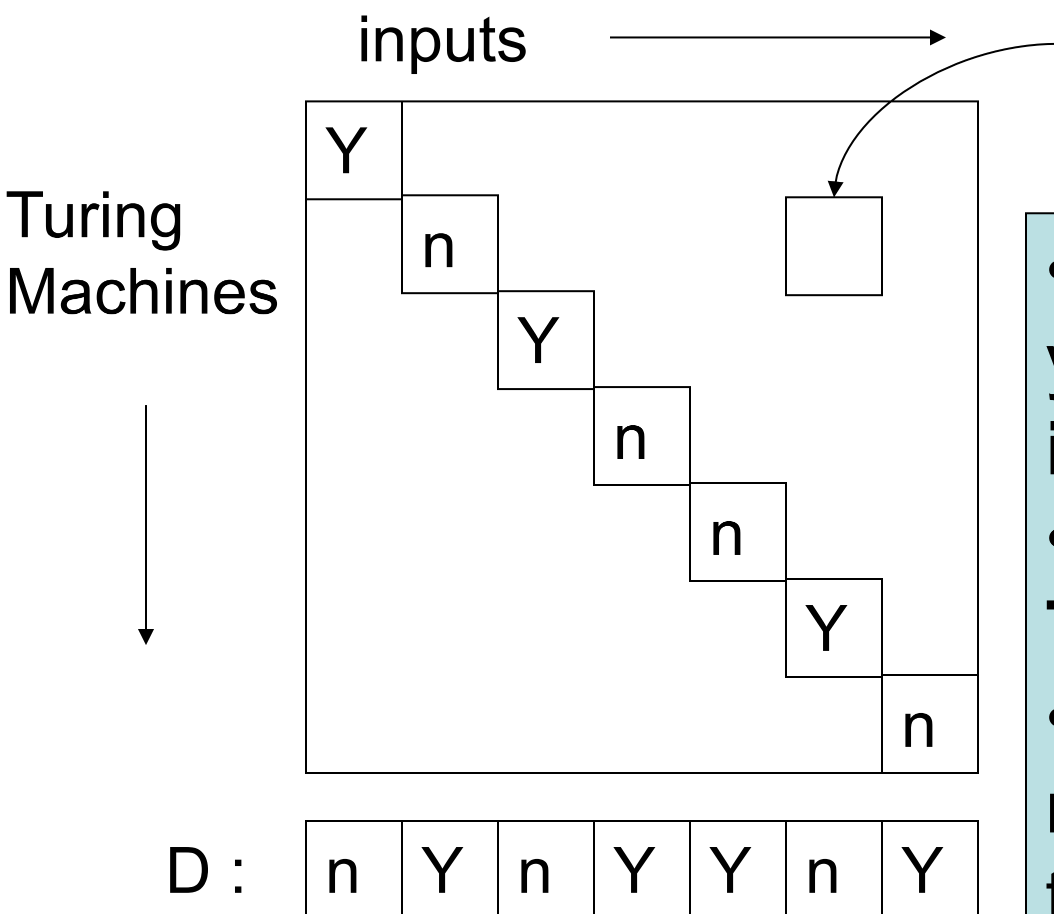


box (M, x): does M halt on x?

The existence of H which tells us yes/no for each box allows us to construct a TM H' that cannot be in the table.

Proof of Time Hierarchy Theorem

box (M, x): does M accept x in time f(n)?



- TM SIM tells us yes/no for each box **in time g(n)**
- rows include all of **TIME(f(n))**
- construct TM D running in time g(2n) that is not in table

Proof of Time Hierarchy Theorem

- Proof:
  - SIM is TM deciding language

    { <M, x> : M accepts x in $\leq f(|x|)$ steps }
  - Claim: SIM runs in time $g(n) = f(n)^3$.
  - define new TM D: on input <M>
    - if SIM accepts <M, <M>>, reject
    - if SIM rejects <M, <M>>, accept
  - D runs in time $g(2n)$

Proof of Time Hierarchy Theorem

- Proof (continued):
  - suppose M in **TIME(f(n))** decides L(D)
    - M(<M>) = SIM(<M, <M>>) ≠ D(<M>)
    - but M(<M>) = D(<M>)
  - contradiction.

Proof of Time Hierarchy Theorem

- Claim: there is a TM SIM that decides

  {<M, x> : M accepts x in $\leq f(|x|)$ steps}

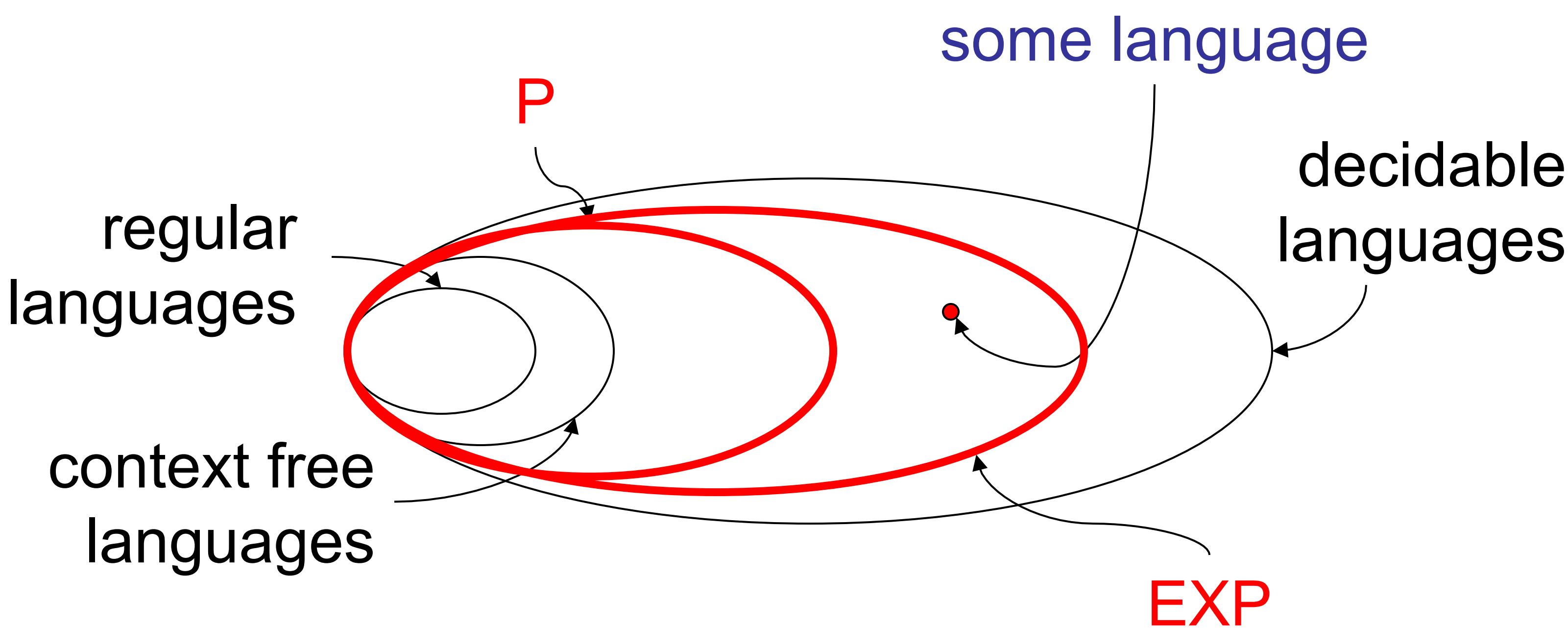
  and runs in time $g(n) = f(n)^3$.
- Proof sketch: SIM has 4 work tapes
  - contents and "virtual head" positions for M's tapes
  - M's transition function and state
  - $f(|x|)$ "+"s used as a clock
  - scratch space

Proof of Time Hierarchy Theorem

- Proof sketch (continued): 4 work tapes
    - contents and "virtual head" positions for M's tapes
    - M's transition function and state
    - f(|x|) "+"s used as a clock
    - scratch space
  - initialize tapes
  - simulate step of M, advance head on tape 3; repeat.
  - can check running time is as claimed.

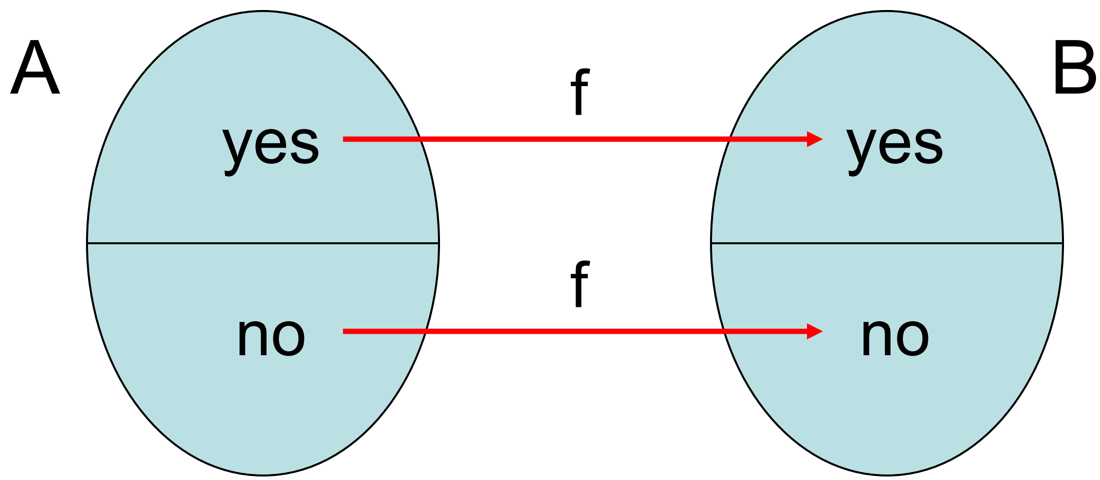
So far…

- We have defined the complexity classes P (polynomial time), EXP (exponential time)



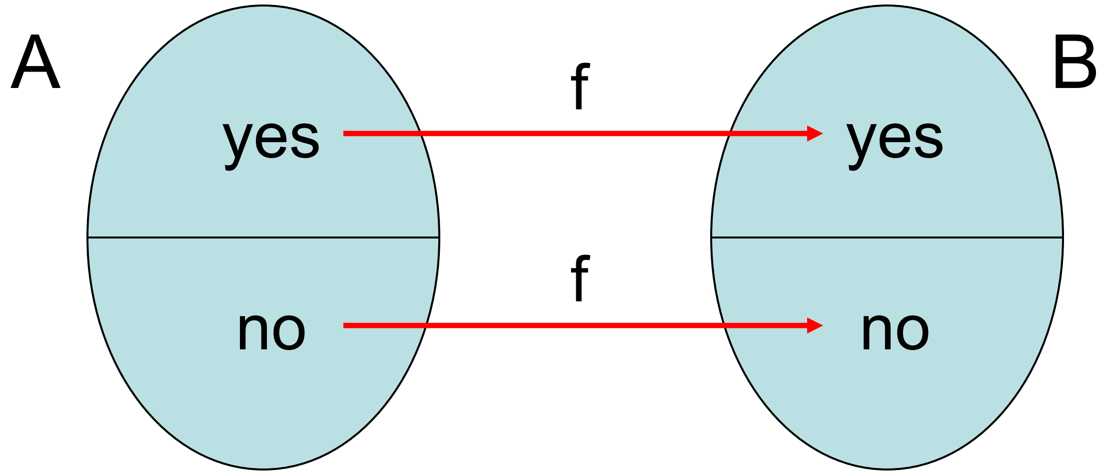
Poly-time reductions

- Type of reduction we will use:
  - "many-one" poly-time reduction (commonly)
  - "mapping" poly-time reduction (book)



reduction from language A to language B

Poly-time reductions



- function f should be poly-time computable

Definition: $f : \Sigma^* \rightarrow \Sigma^*$ is poly-time computable if for some $g(n) = n^{O(1)}$ there exists a $g(n)$-time TM M_f such that on every $w \in \Sigma^*$, M_f halts with $f(w)$ on its tape.

Poly-time reductions

<u>Definition</u>: $A \leq_P B$ ("A reduces to B") if there is a poly-time computable function f such that for all w

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

- as before, condition equivalent to:
  - YES maps to YES *and* NO maps to NO
- as before, meaning is:
  - B is at least as "hard" (or expressive) as A

Poly-time reductions

Theorem: if $A \leq_P B$ and $B \in P$ then $A \in P$.

Proof:

- a poly-time algorithm for deciding A:
- on input w, compute f(w) in poly-time.
- run poly-time algorithm to decide if $f(w) \in B$
- if it says "yes", output "yes"
- if it says "no", output "no"

Example

- 2SAT = {CNF formulas with 2 literals per clause for which there exists a satisfying truth assignment}
- L = {directed graph G, and list of pairs of vertices $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$, such that there is no i for which [u_i is reachable from v_i in G and v_i is reachable from u_i in G]}
- We gave a poly-time reduction from 2SAT to L.
- determined that 2SAT $\in$ P from fact that L $\in$ P

Hardness and completeness

- Reasonable that can efficiently transform one problem into another.

- Surprising:
  - can often find a special language L so that **every** language in a given complexity class reduces to L!
  - powerful tool

Hardness and completeness

- Recall:
  - a language L is a set of strings
  - a complexity class C is a set of languages

Definition: a language L is C-hard if for every language $A \in C$, A poly-time reduces to L; i.e., $A \leq_P L$.

meaning: L is at least as “hard” as anything in C

Hardness and completeness

- Recall:
  - a language L is a set of strings
  - a complexity class C is a set of languages

<u>Definition</u>: a language L is C-complete if L is C-hard and $L \in C$

meaning: L is a "hardest" problem in C

An EXP-complete problem

- Version of A_{TM} with a time bound:

  ATM_B = {<M, x, m> : M is a TM that accepts x within at most m steps}

<u>Theorem</u>: ATM_B is EXP-complete.

Proof:

– what do we need to show?

An EXP-complete problem

- ATM_B = {<M, x, m> : M is a TM that accepts x within at most m steps}
- Proof that ATM_B is EXP-complete:
  – Part 1. Need to show $ATM_B \in$ EXP.
    - simulate M on x for m steps; accept if simulation accepts; reject if simulation doesn't accept.
    - running time $m^{O(1)}$.
    - n = length of input $\geq \log_2 m$
    - running time $\leq m^k = 2^{(\log m)k} \leq 2^{(kn)}$

An EXP-complete problem

- ATM_B = {<M, x, m> : M is a TM that accepts x within at most m steps}
- Proof that ATM_B is EXP-complete:
  - Part 2. For each language $A \in$ EXP, need to give poly-time reduction from A to ATM_B.
  - for a given language $A \in$ EXP, we know there is a TM M_A that decides A in time $g(n) \leq 2^{n^k}$ for some k.
  - what should reduction f(w) produce?

An EXP-complete problem

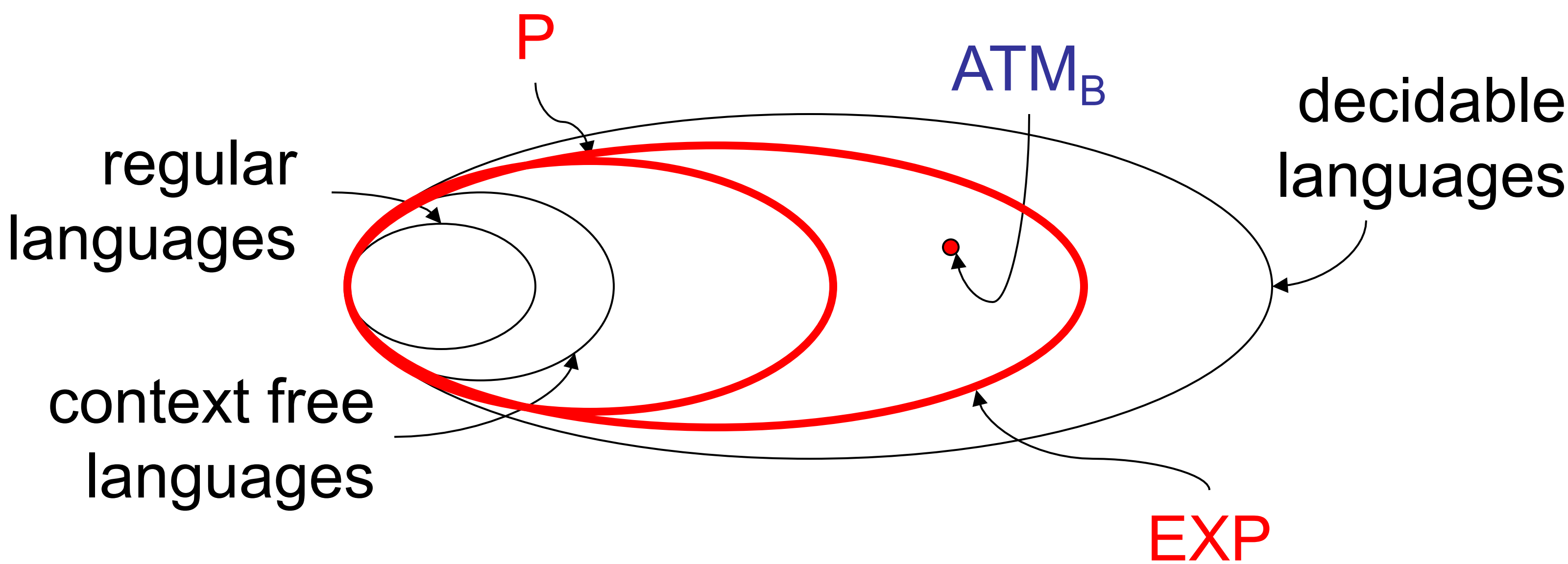
- ATM_B = {<M, x, m> : M is a TM that accepts x within at most m steps}
- Proof that ATM_B is EXP-complete:
  - f(w) = $\langle M_A, w, m \rangle$ where $m = 2^{|w|^k}$
  - is f(w) poly-time computable?
    - hardcode M_A and k…
  - YES maps to YES?
    - $w \in A \Rightarrow \langle M_A, w, m \rangle \in ATM_B$
  - NO maps to NO?
    - $w \notin A \Rightarrow \langle M_A, w, m \rangle \notin ATM_B$

An EXP-complete problem

- A C-complete problem is a surrogate for the entire class C.
- For example: if you can find a poly-time algorithm for ATM_B then there is automatically a poly-time algorithm for every problem in EXP (i.e., EXP = P).
- Can you find a poly-time alg for ATM_B?

An EXP-complete problem

- Can you find a poly-time alg for ATM_B?
- NO! we showed that P (EXP.
- ATM_B is not tractable (intractable).



Back to 3SAT

- Remember 3SAT $\in$ EXP

  3SAT = {formulas in CNF with 3 literals per clause for which there exists a satisfying truth assignment}

- It seems hard. Can we show it is intractable?
  - formally, can we show 3SAT is EXP-complete?

Back to 3SAT

- can we show 3SAT is EXP-complete?
- Don't know how to. Believed unlikely.
- One reason: there is an important positive feature of 3SAT that doesn't seem to hold for problems in EXP (e.g. ATM_B):

3SAT is decidable in polynomial time by a nondeterministic TM

Nondeterministic TMs

- Recall: nondeterministic TM
- informally, TM with several possible next configurations at each step
- formally, A NTM is a 7-tuple

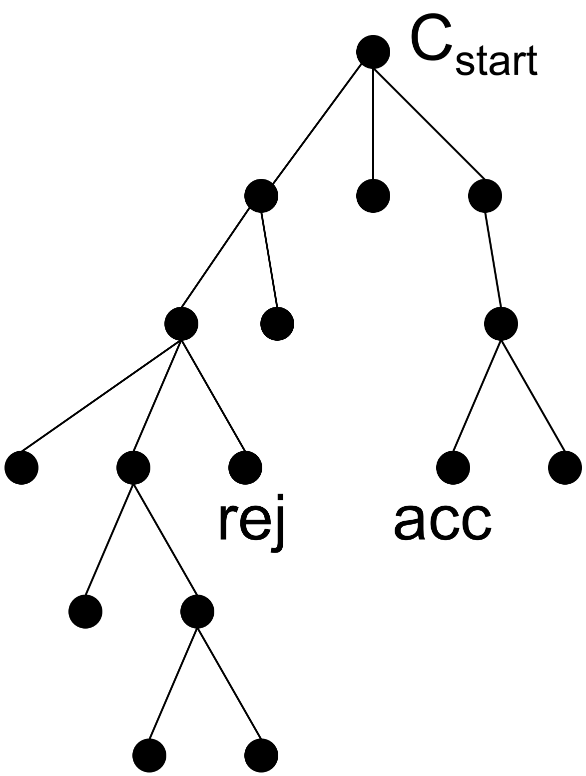
  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where:

  – everything is the same as a TM except the transition function:

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

Nondeterministic TMs

visualize computation of a NTM M as a tree



- nodes are configurations
- leaves are accept/reject configurations
- M accepts if and only if there exists an accept leaf
- M is a decider, so no paths go on forever
- running time is max. path length

The class NP

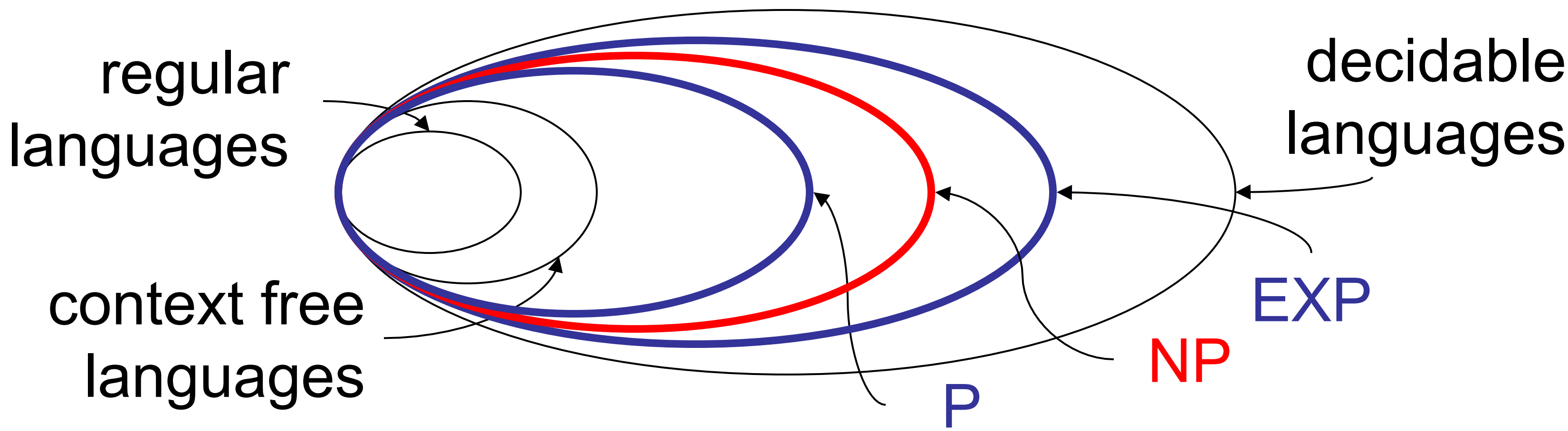
Definition: TIME(t(n)) = {L : there exists a TM M that decides L in time O(t(n))}

$$P = \cup_{k \geq 1} \text{TIME}(n^k)$$

Definition: NTIME(t(n)) = {L : there exists a NTM M that decides L in time O(t(n))}

$$NP = \cup_{k \geq 1} \text{NTIME}(n^k)$$

NP in relation to P and EXP



- P$\subseteq$ NP (poly-time TM ***is*** a poly-time NTM)
- NP$\subseteq$ EXP
  - configuration tree of n^k-time NTM has $\leq b^{n^k}$ nodes
  - can traverse entire tree in $O(b^{n^k})$ time

we do not know if either inclusion is proper

An NP-complete problem

- Version of A_{TM} with a unary time bound, and NTM instead of TM:

  $ANTM_U$ = {<M, x, 1^m> : M is a NTM that accepts x within at most m steps}

Theorem: $ANTM_U$ is NP-complete.

Proof:

– what do we need to show?

An NP-complete problem

- $ANTM_U$ = {<M, x, 1^m> : M is a NTM that accepts x within at most m steps}
- Proof that $ANTM_U$ is NP-complete:
  - Part 1. Need to show $ANTM_U \in NP$.
    - simulate NTM M on x for m steps; do what M does
    - running time $m^{O(1)}$.
    - n = length of input $\geq$ m
    - running time $\leq m^k \leq n^k$

An NP-complete problem

- $ANTM_U = \{<M, x, 1^m> : M$ is a NTM that accepts x within at most m steps$\}$
- Proof that $ANTM_U$ is NP-complete:
  - Part 2. For each language $A \in NP$, need to give poly-time reduction from A to $ANTM_U$.
  - for a given language $A \in NP$, we know there is a NTM M_A that decides A in time $g(n) \leq n^k$ for some k.
  - what should reduction $f(w)$ produce?

An NP-complete problem

- $ANTM_U$ = {<M, x, 1^m> : M is a NTM that accepts x within at most m steps}
- Proof that $ANTM_U$ is NP-complete:
  - f(w) = <M_A, w, 1^m> where m = $|w|^k$
  - is f(w) poly-time computable?
    - hardcode M_A and k…
  - YES maps to YES?
    - $w \in A \Rightarrow$ <M_A, w, 1^m> $\in ANTM_U$
  - NO maps to NO?
    - $w \notin A \Rightarrow$ <M_A, w, 1^m> $\notin ANTM_U$

An NP-complete problem

- If you can find a poly-time algorithm for ANTM_U then there is automatically a poly-time algorithm for every problem in NP (i.e., NP = P).

- No one knows if can find a poly-time alg for ANTM_U…

Cook-Levin Theorem

- Gateway to proving lots of natural, important problems NP-complete is:

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

- Recall: 3SAT = {φ : φ is a CNF formula with 3 literals per clause for which there exists a satisfying truth assignment}

Poly-time verifiers

- NP = {L : L decide… e NTM}

“witness” or “certificate”

- Very useful alternate definition of NP:

efficiently verifiable

<u>Theorem</u>: language L is in NP if… it is expressible as:

$$L = \{ x \mid \exists\, y, |y| \leq |x|^k, (x, y) \in R \}$$

where R is a language in P.

- poly-time TM M_R deciding R is a “verifier”

Poly-time verifiers

- Example: 3SAT expressible as

  3SAT = {φ : φ is a 3-CNF formula for which $\exists$ assignment A for which $(\varphi, A) \in R$}

  R = {(φ, A) : A is a sat. assign. for φ}

  – satisfying assignment A is a "witness" of the satisfiability of φ (it "certifies" satisfiability of φ)

  – R is decidable in poly-time

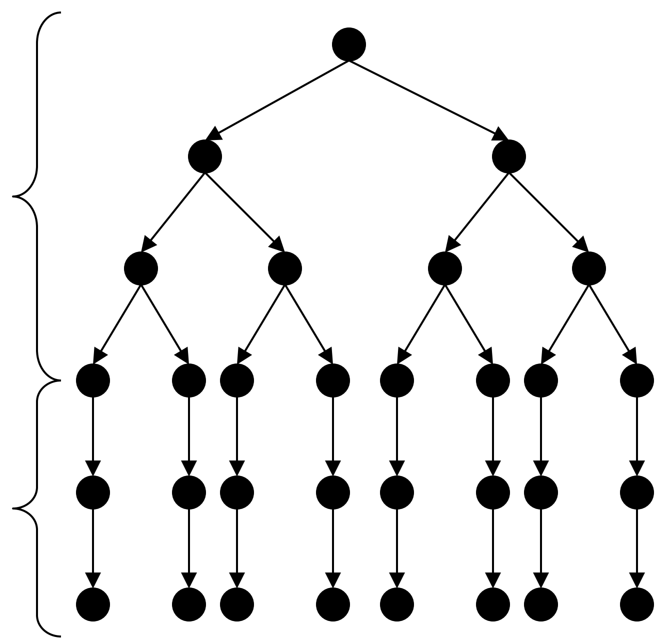
Poly-time verifiers

$$L = \{ x \mid \exists\, y, |y| \leq |x|^k, (x, y) \in R \}$$

<u>Proof</u>: ($\Leftarrow$) give poly-time NTM deciding L

phase 1: "guess" y with $|x|^k$ nondeterministic steps

phase 2: decide if $(x, y) \in R$



Poly-time verifiers

Proof: ($\Rightarrow$) given L $\in$ NP, describe L as:

$$L = \{ x \mid 9\ y, |y| \leq |x|^k, (x, y) \in R \}$$

- L is decided by NTM M running in time n^k
- define the language

  R = { (x, y) : y is an accepting computation history of M on input x}
- check: accepting history has length $\leq |x|^k$
- check: M accepts x iff $\exists y, |y| \leq |x|^k, (x, y) \in R$

Cook-Levin Theorem

- Gateway to proving lots of natural, important problems NP-complete is:

<u>Theorem</u> (Cook, Levin): 3SAT is NP-complete.

- Recall: 3SAT = {φ : φ is a CNF formula with 3 literals per clause for which there exists a satisfying truth assignment}

Cook-Levin Theorem

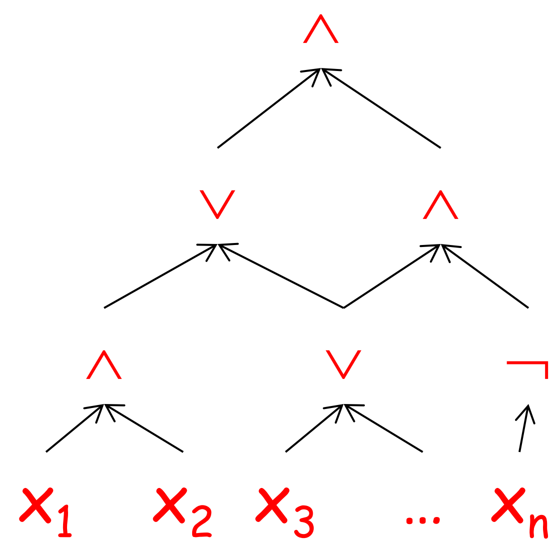
- Proof outline
  - show CIRCUIT-SAT is NP-complete

    CIRCUIT-SAT = {C : C is a Boolean circuit for which there exists a satisfying truth assignment}

  - show 3SAT is NP-complete (reduce from CIRCUIT SAT)

Boolean Circuits

- Boolean circuit C
  - directed acyclic graph
  - nodes: AND ($\wedge$); OR ($\vee$); NOT ($\neg$); variables x_i



- C computes function $f:\{0,1\}^n \rightarrow \{0,1\}$ in natural way
  - identify C with function f it computes
- size = # nodes

Boolean Circuits

- every function $f:\{0,1\}^n \rightarrow \{0,1\}$ computable by a circuit of size at most $O(n2^n)$
  - AND of n literals for each x such that $f(x) = 1$
  - OR of up to 2^n such terms

CIRCUIT-SAT is NP-complete

<u>Theorem:</u> CIRCUIT-SAT is NP-complete

CIRCUIT-SAT = {C : C is a Boolean circuit for which there exists a satisfying truth assignment}

Proof:

– Part 1: need to show CIRCUIT-SAT $\in$ NP.

  - can express CIRCUIT-SAT as:

CIRCUIT-SAT = {C : C is a Boolean circuit for which $\exists$ x such that (C, x) $\in$ R}

R = {(C, x) : C is a Boolean circuit and C(x) = 1}

CIRCUIT-SAT is NP-complete

CIRCUIT-SAT = {C : C is a Boolean circuit for which there exists a satisfying truth assignment}

Proof:

- Part 2: for **each** language $A \in$ NP, need to give poly-time reduction from A to CIRCUIT-SAT
- for a given language $A \in$ NP, we know

$$A = \{x \mid \exists\, y, |y| \le |x|^k, (x, y) \in R\}$$

and there is a (deterministic) TM M_R that decides R in time $g(n) \le n^c$ for some c.

CIRCUIT-SAT is NP-complete

- Tableau (configurations written in an array) for machine M_R on input $w = (x, y)$:

w_1/q_s	w_2	…	w_n	…	_
w_1	w_2/q_1	…	w_n	…	_
w_1/q_1	a	…	w_n	…	_
		⋮			⋮
_$/q_a$	_	…	_	…	_

- height = time taken = $|w|^c$
- width = space used $\leq |w|^c$

CIRCUIT-SAT is NP-complete

- Important observation: contents of cell in tableau determined by 3 others above it:

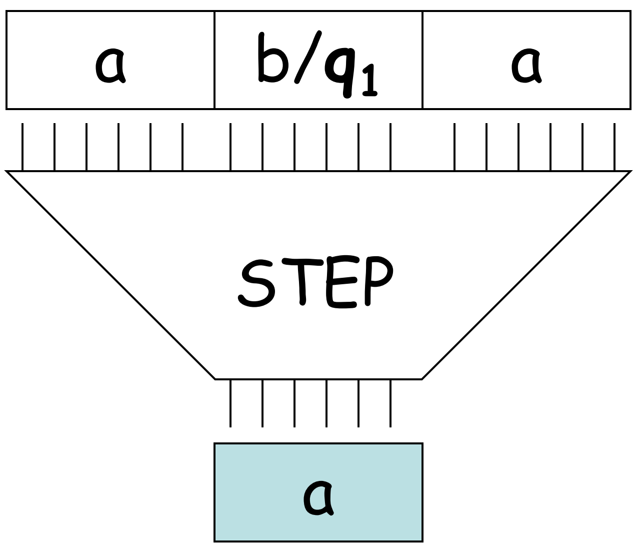
a/q_1	b	a
	b/q_1	

a	b/q_1	a
	a	

a	b	a
	b	

CIRCUIT-SAT is NP-complete

- Can build Boolean circuit STEP
  - input (binary encoding of) 3 cells
  - output (binary encoding of) 1 cell



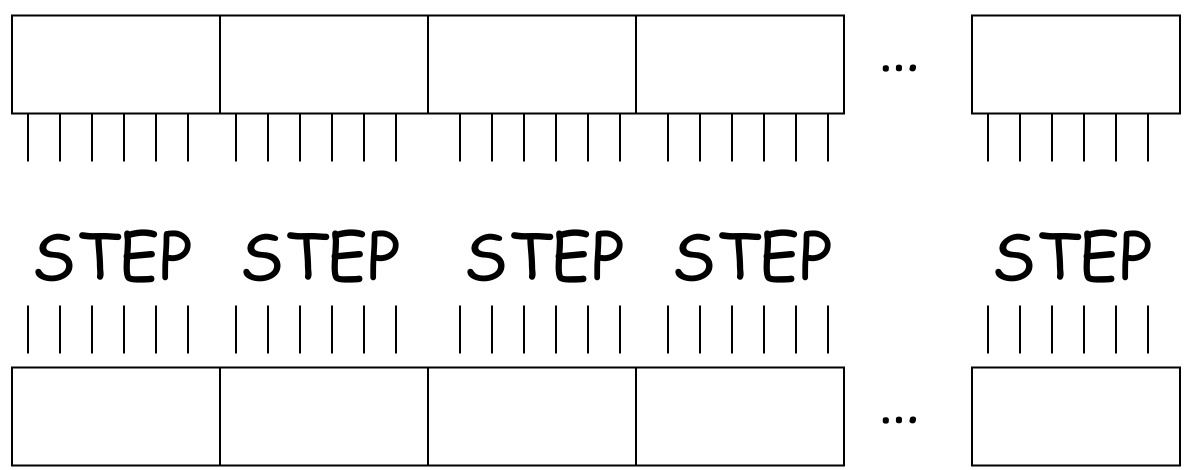
- each output bit is some function of inputs
- can build circuit for each
- size is independent of size of tableau

CIRCUIT-SAT is NP-complete

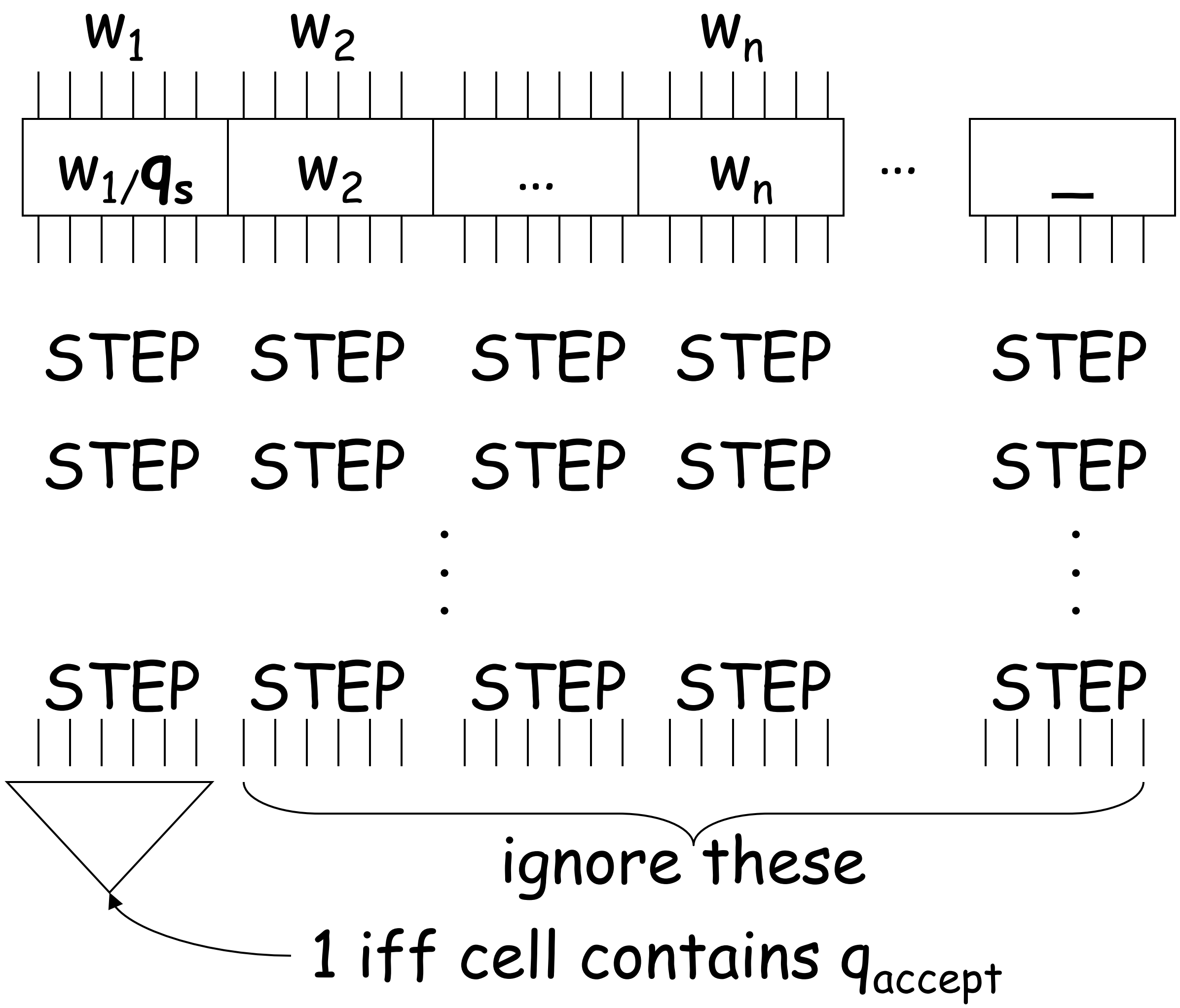
Tableau for M_R on input $w = (x, y)$

w_1/q_s	w_2	…	w_n	…	_
w_1	w_2/q_1	…	w_n	…	_
		⋮			⋮

- $|w|^c$ copies of STEP compute row i from i-1



CIRCUIT-SAT is NP-complete



This circuit $C_{M_R, w}$ has inputs $w_1w_2\ldots w_n$

and $C(w) = 1$ iff M_R accepts input w.

Size = $O(|w|^{2c})$

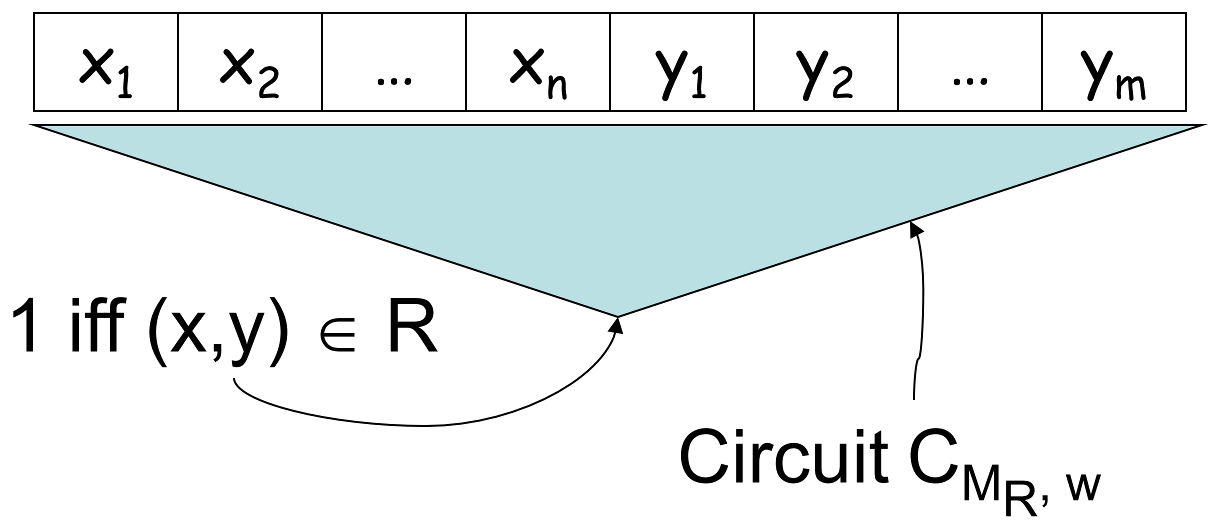
CIRCUIT-SAT is NP-complete

– recall: we are reducing language A:

$$A = \{ x \mid \exists\, y, |y| \leq |x|^k, (x, y) \in R \}$$

to CIRCUIT-SAT.

– f(x) produces the following circuit:



– hardwire input x

– leave y as variables

CIRCUIT-SAT is NP-complete

– is f(x) poly-time computable?
  - hardcode M_R, k and c
  - circuit has size $O(|w|^{2c})$; $|w| = |(x,y)| \le n + n^k$
  - each component easy to describe efficiently from description of M_R

– YES maps to YES?
  - $x \in A \Rightarrow 9$ y, M_R accepts (x, y) $\Rightarrow f(x) \in$ CIRCUIT-SAT

– NO maps to NO?
  - $x \notin A \Rightarrow 8$ y, M_R rejects (x, y) $\Rightarrow f(x) \notin$ CIRCUIT-SAT