

Lecture 2

Welcome to INTRO TO COMPLEXITY!

First a quick recap of class policies/grading/etc

Me: Toniann Pitassi (Toni)

TAs: Conlan Olson
Rain Yang

Course

Webpage: www.cs.columbia.edu/~toni/courses/cs4236/complexity.html

INTRODUCTION TO COMPLEXITY THEORY

COMS 4236, SPRING 2026

General Information

- Instructor: [Tonian Pitassi](#) email: tp2684@columbia.edu; OH: Tues 5-6, CSB 513
- Class Time/Place: Fridays: 1:10-3:40 415 Schapiro
- Textbook: Introduction to the Theory of Computation, by M. Sipser
- For some lectures we will follow [Lectures in Computational Complexity](#) of Jin-Yi Cai and the book [Computational Complexity: A Modern Approach](#) of Sanjeev Arora and Boaz Barak.
- Lecture topics and homeworks are posted on the [Lectures/Weekly Info](#) page

Teaching Assistants

- Conlan Olson, email: co2583@columbia.edu; OH: Wed 5-6, Location: CSB 453
- Rain Yang, email: ry2553@columbia.edu; OH: Wed 4-5, Location: CSB 453

Quick Links

- [Lectures/Weekly Info](#): Schedule of lectures, tests/quizzes, and homeworks, links to readings and lectures.
- [Courseworks](#): Has links to a range of resources including the Video Library (for course videos), and Gradescope.
- [Gradescope](#): Submit assignments and view grades here. (You can set up Gradescope by clicking the "Gradescope" module on Courseworks.)

Course Description:

In this course, we study Computational Complexity, one of the most active areas of theoretical Computer Science, devoted to understanding the intrinsic computational hardness of problems, and the limitations of efficient algorithms. Along the way we will uncover some very important ideas, including: connections between computational hardness and randomness, and between computational hardness and cryptography; Below is a list of topics we plan to cover in the course:

- Time complexity classes (P, NP, PH, EXP), Reductions and Complete problems.
- Space complexity classes (Logspace, NL, RL, PSPACE). Graph connectivity, Catalytic Space
- Interactive proofs, $IP=PSPACE$.
- Randomized complexity classes BPP, RP; randomness in crypto/privacy; hardness vs randomness.
- Boolean circuit and formulas. Lower bounds for constant-depth circuits.
- Communication complexity.
- Approximation algorithms, PCP theorem.

Prerequisites

- An introductory course on theory of computation, e.g., COMS 3261. In particular you should be familiar with Turing machines, decidability, and the complexity classes P and NP, at the level of Chapters 3,4,7 of Sipser, Introduction to the Theory of Computation. You should also be comfortable with asymptotic notation (Big-O and Big-Omega), and run-time analysis.
- A course on algorithms will be helpful but not required. This course is mostly theoretical, so you

In this course, we study Computational Complexity, one of the most active areas of theoretical Computer Science, devoted to understanding the intrinsic computational hardness of problems, and the limitations of efficient algorithms. Along the way we will uncover some very important ideas, including: connections between computational hardness and randomness, and between computational hardness and cryptography; Below is a list of topics we plan to cover in the course:

- Time complexity classes (P, NP, PH, EXP), Reductions and Complete problems.
- Space complexity classes (Logspace, NL, RL, PSPACE). Graph connectivity, Catalytic Space
- Interactive proofs, $IP=PSPACE$.
- Randomized complexity classes BPP, RP; randomness in crypto/privacy; hardness vs randomness.
- Boolean circuit and formulas. Lower bounds for constant-depth circuits.
- Communication complexity.
- Approximation algorithms, PCP theorem.

Prerequisites

- An introductory course on theory of computation, e.g., COMS 3261. In particular you should be familiar with Turing machines, decidability, and the complexity classes P and NP, at the level of Chapters 3,4,7 of Sipser, Introduction to the Theory of Computation. You should also be comfortable with asymptotic notation (Big-O and Big-Omega), and run-time analysis.
- A course on algorithms will be helpful but not required. This course is mostly theoretical, so you should have a good grasp of discrete math, probability theory and proofs (See Chapter 0 of Sipser's book).

Grading

- Four homeworks (2% each). HWs posted on Wednesdays and due by 11:59pm the next week (submit thru Gradescope). HW1: Posted Sept 23, Due Oct 1; HW2: Oct 7, due Oct 15; HW3: Nov 4, due Nov 12; HW4: Dec 2, due Dec 10.
- Three in-class tests (20% each) (Oct 9, Oct 23, Nov 20) and a final exam (30%) (Dec 18). See [Lectures/Weekly Info](#) for important details and more info.
- Please read the [Syllabus](#) carefully for information on grading, collaboration, AI policy, and important dates.

Lectures

For some lectures we will follow [Lectures in Computational Complexity](#) of Jin-Yi Cai (LC below), the book [Computational Complexity: A Modern Approach](#) of Sanjeev Arora and Boaz Barak (MA below), and the book [Communication Complexity](#) of Anup Rao and Amir Yehudayoff (CC below). TC denotes the [Sipser book](#)

Date Topic

Sep 11	Introduction; Review: Turing machines, Time Complexity, P vs NP, NP-Completeness
Sep 18	Polynomial Hierarchy; Hierarchy theorems, Space Complexity
Sep 25	Space Complexity: Classes L, NL, PSPACE; NL=coNL; PSPACE=NPSpace
Oct 2	Space Complexity cont'd, Catalytic Space
Oct 9	Test 1. Catalytic Space, Cook-Mertz and Williams' algorithms
Oct 16	Randomized algorithms: RP, BPP and examples; hardness vs. randomness; randomness in cryptography
Oct 23	Test 2 Randomness Continued
Oct 30	Interactive Proofs; #SAT in IP, IP=PSPACE
Nov 6	PCP Theorem, Hardness of Approximation
Nov 13	Intro Communication Complexity, Deterministic, Rank, Randomized CC
Nov 20	Test 3. Communication Complexity: Randomized Communication, Set Disjointness
Nov 27	Holiday (no class)
Dec 4	Intro Circuit Complexity, Karp-Lipton Theorem
Dec 11	(Last Class) TFNP and proof complexity, Wrap-up
Dec 18	Final Exam (See Exam Schedule for Time/Location)

Readings

[Notes](#); TC 3.1-3.2, and 7; LC 2.1-2.2
 TC 7 and 9.1, LC 2.3-2.5
 TC Chap 8 (also LC Chap 3, MA Chap 4)
 TC Chap 8 (also LC Chap 3, MA Chap 4)

[Simulating time in \$\text{sqrt-space}\$ \(R. Williams\)](#)
 TC Sec 10.2; Also LC 5.1.1-5.1.3, 5.4-5.5, and MA Chap 7.

TC Sec 10.4 (also MA Chap 8)
 MA Sec 18.1-18.2
 CC pp. 15-37

CC Chap 3, See also Sec 2-12.4 of [Notes \(S. Lovett\)](#)

Sec 4.1, 4.2 of LC (also Chap 6 of MA)
 CC pp. 97-101

Homework

[HW1 Due 10/01 11:59pm](#)

[HW2 Due 10/15 11:59pm](#)

[HW3 Due 11/12 11:59pm](#)

[HW4 Due 12/10 11:59pm](#)

Grading

HOMEWORK

	POSTED	DUE (gradescope)
HW1	Sept 23	Oct 1
HW2	Oct 7	Oct 15
HW3	Nov 4	Nov 12
HW4	Dec 2	Dec 10

10 do
Best 3
out of 4

TESTS

(in class)

Test 1	Oct 9	20 do
Test 2	Oct 23	20 do
Test 3	Nov 20	20 do

FINAL :

Dec 18 (tentative. see exam schedule)

30 do

Today:

→ Time Complexity:

P , NP , $coNP$, EXP , NP -completeness

Time hierarchy Theorem

→ Space complexity

TMs

Input tape (read only)

Work tapes (read/write)

Output tape (write only)

Time Complexity (quick recap last class)

$\text{TIME}(T(n))$ = set of all Languages $L \subseteq \{0,1\}^*$ that are decided by a TM in time $T(n)$

Defn $P = \bigcup_{k \geq 0} \text{Time}(n^k)$

Languages decidable in polynomial time

Nondeterminism

Recall normal TMs are deterministic: For every configuration there is exactly one possible transition

Nondeterministic TMs generalize deterministic TMs.

NTMs are not a realistic computational model.

But invented to help understand a certain large class of problems seemingly not in P, but nonetheless solvable can be efficiently verified

Nondet TMs: Like ordinary TMs but now transition function

is "nondeterministic": $\delta(q, a, d, e) \rightarrow \{ \text{set of transitions} \}$

ex: $\delta(q_2, a, b) \rightarrow \left\{ \begin{array}{l} q_3, a, c, L \\ q_{rej}, a, b, R \end{array} \right\}$ $\leftarrow$ one work tape + input tape
example for a decision problem

Defn A NTM M decides $L \subseteq \{0,1\}^*$ if:

$x \in L \Rightarrow$ there exists a sequence of transitions (=computation path)
s.t. $M(x)$ halts & accepts

$x \notin L \Rightarrow$ for all possible computation paths, $M(x)$ does not
halt & accept

Defn Runtime of NTM M on $x = \max$ runtime (until we halt)
over all computation paths of M on x

Defn $\text{NTIME}(f(n)) = \{ L \subseteq \{0,1\}^* \mid \exists \text{ NTM } M \text{ that decides } L$
and $\forall x$ runtime of $M(x)$ is $\leq f(|x|)$.

Defn 1 $NP := \bigcup_{k \geq 0} NTIME(n^k)$

Alternative "Verifier" Definition of NP

Defn 2 $L \in NP$ iff $\exists$ poly $p(n) = n^c$ and a deterministic TM. V
such that $\uparrow$
verifier

(1) $V(x, *)$ runs in time poly in $|x|$

(2) $\forall x \in \{0,1\}^*$ $x \in L \iff \exists u$ such that $V(x, u) = 1$
 $\uparrow$
"witness"

claim: Definitions 1, 2 are equivalent

Examples of NP Languages

- ① 3SAT : Input : 3CNF formula F over $x_1, \dots, x_n$
Output : 1 if there is an assignment $\alpha \in \{0,1\}^n$ that satisfies F .

Example $F = (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2 \vee x_4 \vee \bar{x}_1) \wedge (x_2 \vee x_3)$

$V(F, \alpha)$: check if $F(\alpha) = 1$
if yes $\rightarrow$ accept
else $\rightarrow$ reject

Check 2 defs of NP are satisfied for 3SAT

Examples of NP Languages

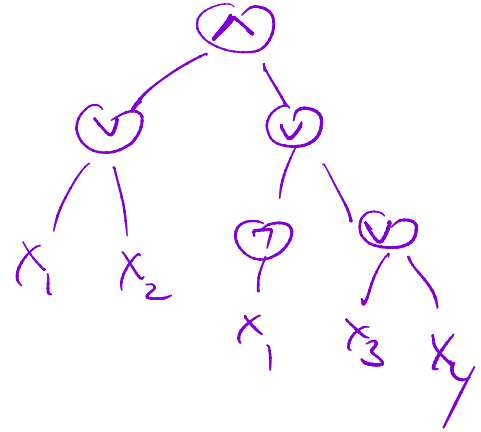
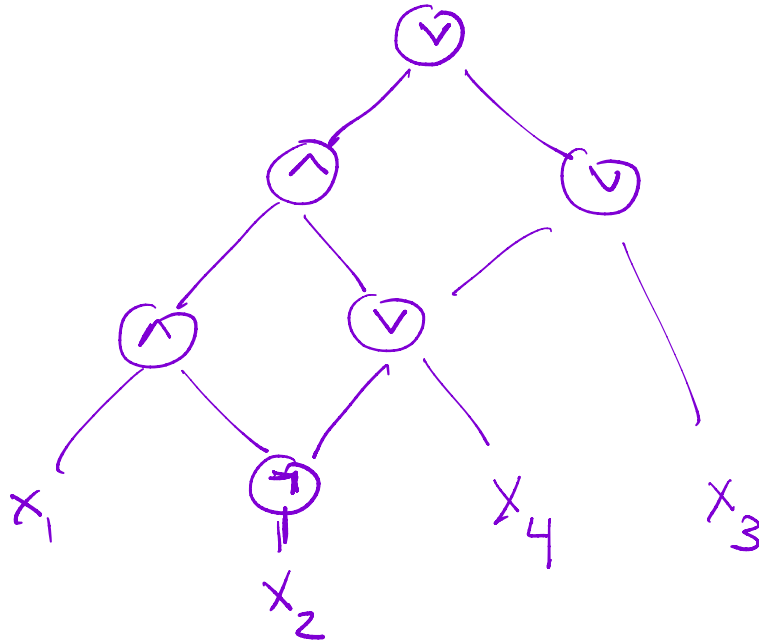
② Circuit SAT Input: Boolean circuit C with n inputs $x_1, \dots, x_n$

Output: 1 (accept) if there exists a Boolean assignment $\alpha \in \{0,1\}^n$ to $x_1 \dots x_n$ such that $C(\alpha) = 1$

3CNF : is a type of circuit

Example:

$$F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$$



Cook-Levin Thm

Circuit-SAT = $\{ C \mid C \text{ is a satisfiable Boolean circuit} \}$ is NP-complete

Proof Idea

1. Circuit-SAT $\in$ NP: $V(C, \alpha)$ $\alpha \in \{0,1\}^n$

$V(C, \alpha)$: evaluate C on input α
and accept if $C(\alpha)$ outputs 1
or reject

2. Circuit-SAT NP-hard: (See Sipser Thm 7.37)

Cook-Levin Thm

$\text{Circuit-SAT} = \{ C \mid C \text{ is a satisfiable Boolean circuit} \}$

2. Circuit-SAT NP-hard: (See Sipser Thm 4.37)

Idea: Let $L \in \text{NP}$, + let V be verifier for L :

$x \in L \Rightarrow \exists y \text{ s.t. } V(x, y) \text{ accepts}$

$x \notin L \Rightarrow \forall y \ V(x, y) \text{ rejects}$

runtime of $V(x, y) = O(|x|^k)$

given $(V, x) \longrightarrow C(z_1, \dots, z_m)$ circuit

such that $\exists$ an assignment y st $V(x, y)$ accepts $\Leftrightarrow$

$\exists$ setting to $z_1, \dots, z_m$ such that $C(z_1, \dots, z_m)$ accepts

$z_1, \dots, z_m$ encodes "tableaux" of V on inputs x, y

Other NP-complete Languages

I. 3SAT : Input is 3CNF formula $C = C_1 \wedge C_2 \wedge \dots \wedge C_m$
each C_i is size ≤ 3 clause

$$\text{Ex. } C = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_4 \vee x_5) \wedge (x_3 \vee \neg x_2 \vee x_4)$$

accept iff $\exists$ assignment $\alpha \in \{0,1\}^n$ to $x_1, \dots, x_n$
s.t. $C(\alpha) = 1$

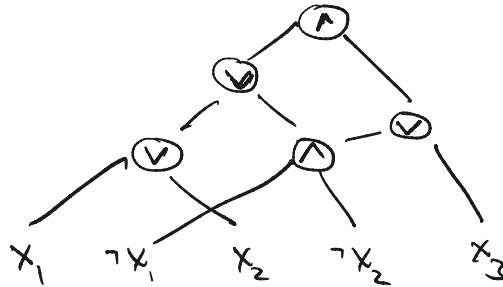
1. $3SAT \in NP$

2. $3SAT$ is NP hard.

show $CIRCUIT-SAT \leq_P 3SAT$

Ex Let C be a circuit over $x_1 \dots x_n$

C :



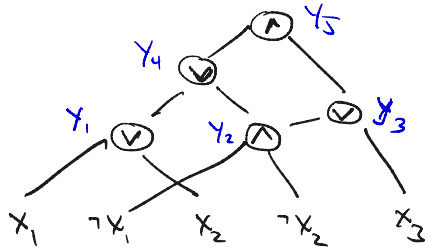
Create 3CNF F_c :

Variables: $x_1 \dots x_n$ plus new vars one per intermediate gate of C

clauses: express that value of y_i is "correct" for each y_i and output is 1

Ex Let C be a circuit over $x_1 \dots x_n$

C :



Create 3CNF F_c :

Variables: $x_1 \dots x_n$ plus new vars one per intermediate gate of C

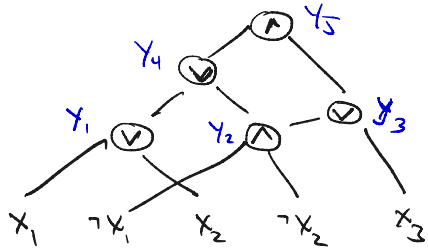
clauses:

$$\begin{aligned}
 Y_1 &\equiv x_1 \vee x_2 & \Leftrightarrow & (Y_1 \rightarrow x_1 \vee x_2) \wedge (x_1 \vee x_2 \rightarrow Y_1) \Leftrightarrow (\neg Y_1 \vee x_1 \vee x_2) \wedge \\
 Y_2 &\equiv \neg x_1 \wedge \neg x_2 & & (\neg x_1 \vee Y_1) \wedge (\neg x_2 \vee Y_1) \\
 Y_3 &\equiv Y_2 \vee x_3 \\
 Y_4 &\equiv Y_1 \vee Y_2 \\
 Y_5 &\equiv Y_4 \wedge Y_3 \\
 Y_5 &\equiv 1
 \end{aligned}$$

convert to CNF
↓

Ex Let C be a circuit over $x_1 \dots x_n$

C :



Create 3CNF F_C :

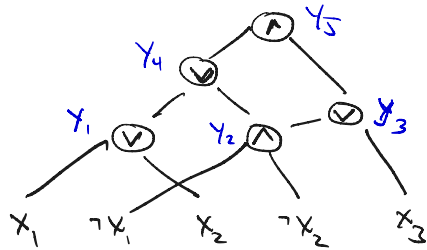
Variables: $x_1 \dots x_n$ plus new vars one per intermediate gate of C

clauses:

$$\begin{aligned}
 Y_1 &\equiv x_1 \vee x_2 &\Leftrightarrow (\neg Y_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee Y_1) \wedge (\neg x_2 \vee Y_1) \\
 Y_2 &\equiv \neg x_1 \wedge \neg x_2 &\Leftrightarrow (Y_2 \Rightarrow \neg x_1 \wedge \neg x_2) \wedge (\neg x_1 \wedge \neg x_2 \Rightarrow Y_2) \Leftrightarrow \\
 Y_3 &\equiv Y_2 \vee x_3 &(\neg Y_2 \vee \neg x_1)(\neg Y_2 \vee \neg x_2)(x_1 \vee x_2 \vee Y_2) \\
 Y_4 &\equiv Y_1 \vee Y_2 \\
 Y_5 &\equiv Y_4 \wedge Y_3 \\
 Y_5 &\equiv 1
 \end{aligned}$$

Ex Let C be a circuit over $x_1 \dots x_n$

C :



Create 3CNF F_c :

Variables: $x_1 \dots x_n$ plus new vars one per intermediate gate of C

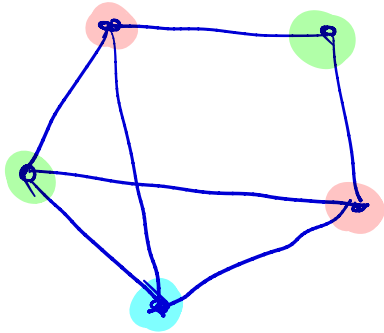
clauses:

$$\begin{aligned}
 y_1 &\equiv x_1 \vee x_2 &\Leftrightarrow (\neg y_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee y_1) \wedge (\neg x_2 \vee y_1) \\
 y_2 &\equiv \neg x_1 \wedge \neg x_2 &\Leftrightarrow (\neg y_2 \vee \neg x_1) \wedge (\neg y_2 \vee \neg x_2) \wedge (x_1 \vee x_2 \vee y_2) \\
 y_3 &\equiv y_2 \vee x_3 &\Leftrightarrow (\neg y_3 \vee y_2 \vee x_3) \wedge (\neg y_2 \vee y_3) \wedge (\neg x_3 \vee y_3) \\
 y_4 &\equiv y_1 \vee y_2 &\Leftrightarrow (\neg y_4 \vee y_1 \vee y_2) \wedge (\neg y_1 \vee y_4) \wedge (\neg y_2 \vee y_4) \\
 y_5 &\equiv y_4 \wedge y_3 &\Leftrightarrow (\neg y_5 \vee y_4) \wedge (\neg y_5 \vee y_3) \wedge (\neg y_4 \vee \neg y_3 \vee y_5) \\
 y_5 &\equiv 1 &\Leftrightarrow (y_5)
 \end{aligned}$$

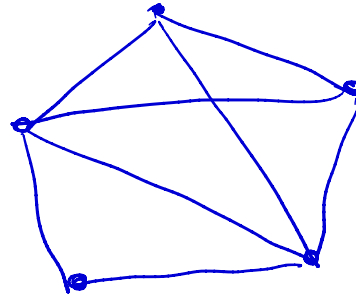
Other NP-complete Languages

II. 3COL Input: undirected graph $G = (V, E)$
Output: 1 iff G is 3-colorable

Ex



↑
yes! 3-colorable



↑
No. Not 3-colorable

$|V| = n$: input size $\Theta(n^2)$

Other NP-complete Languages

II. 3COL Input: undirected graph $G = (V, E)$

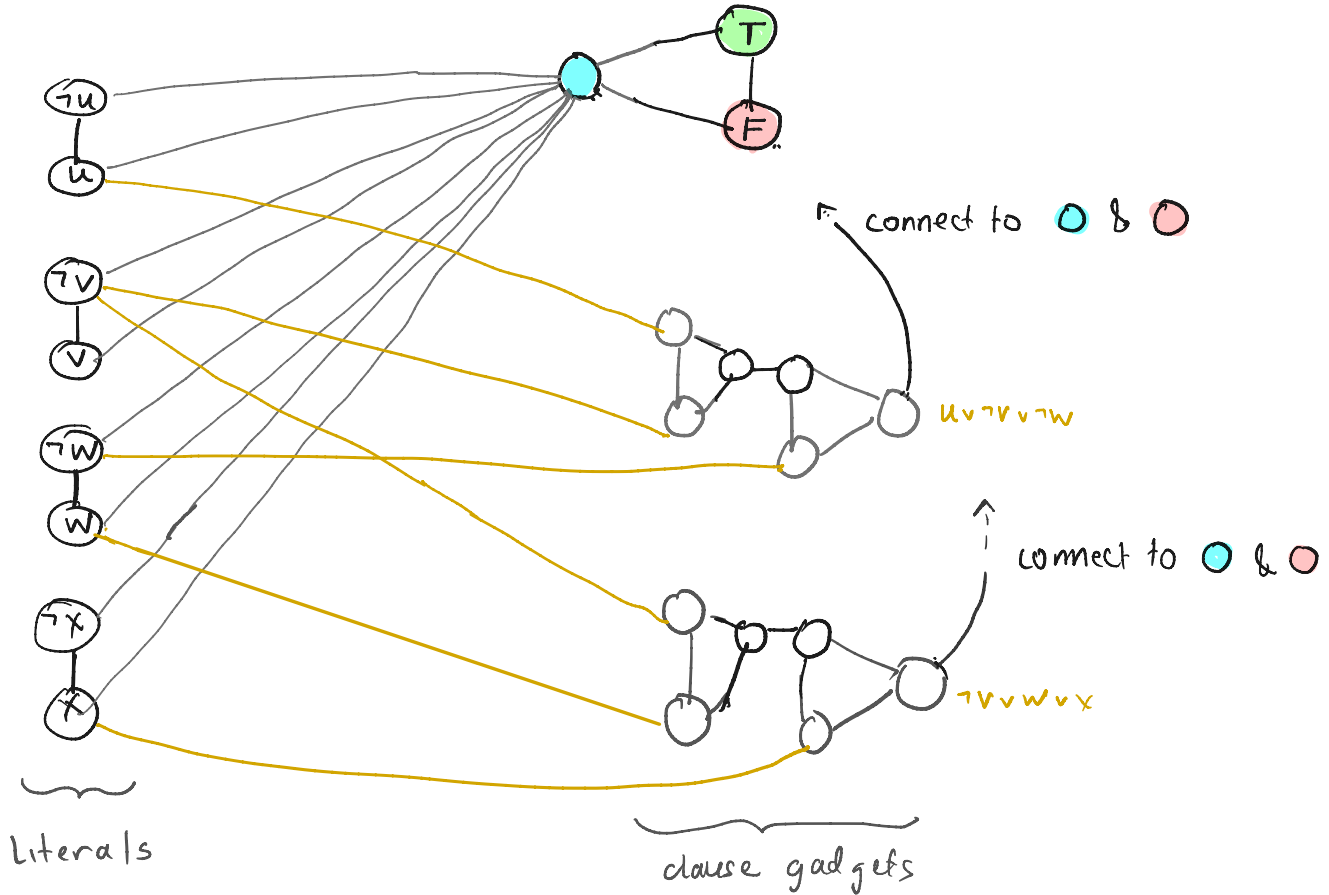
Output: 1 iff G is 3-colorable

1. In NP: verifier $V(G, \vec{c})$: $\vec{c} \in \{R, B, G\}^{|V|}$
check if $\vec{c}$ is a valid 3-coloring of G

2. NP-hard: Show $3SAT \leq_p 3COL$

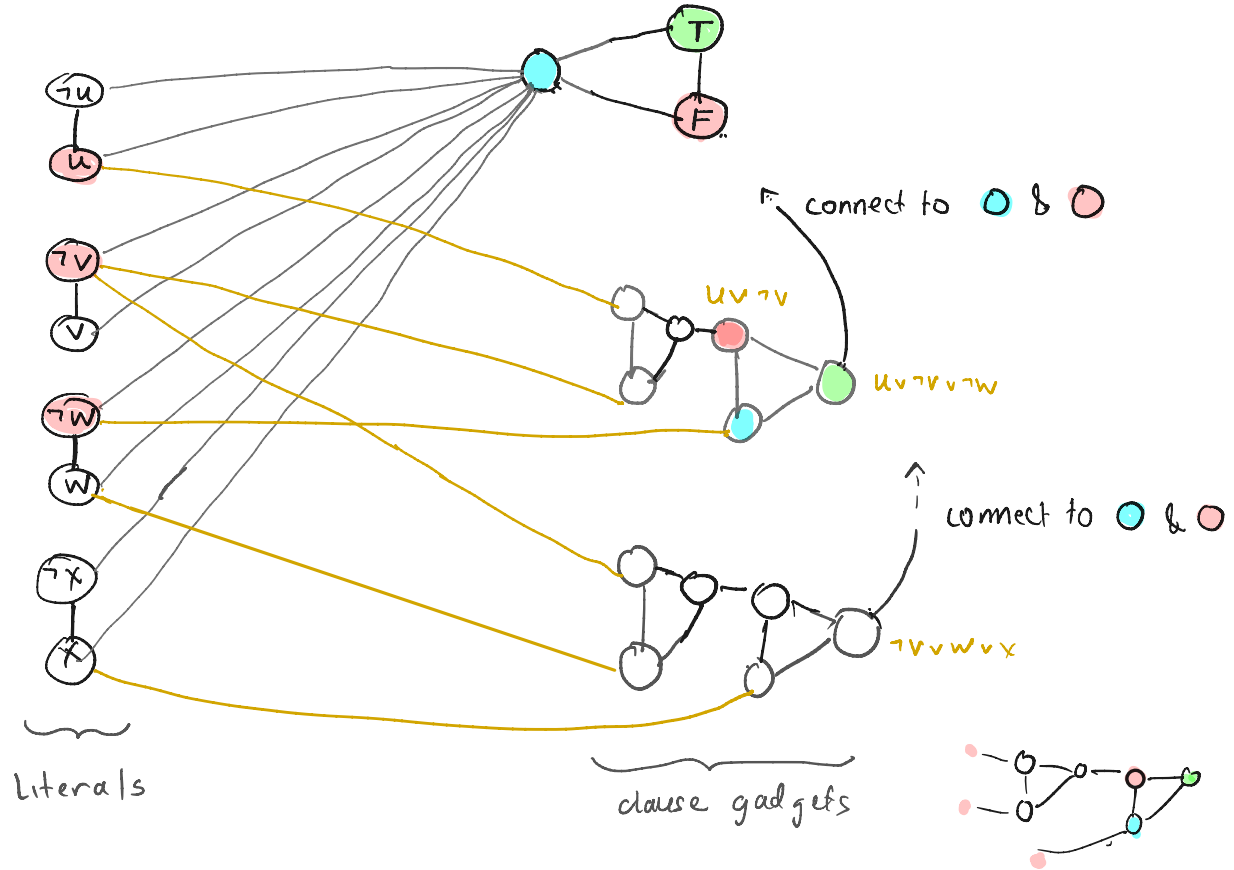
Example

$$F = \underbrace{(u \vee \neg v \vee \neg w)}_{C_1} \wedge \underbrace{(\neg v \vee w \vee x)}_{C_2}$$



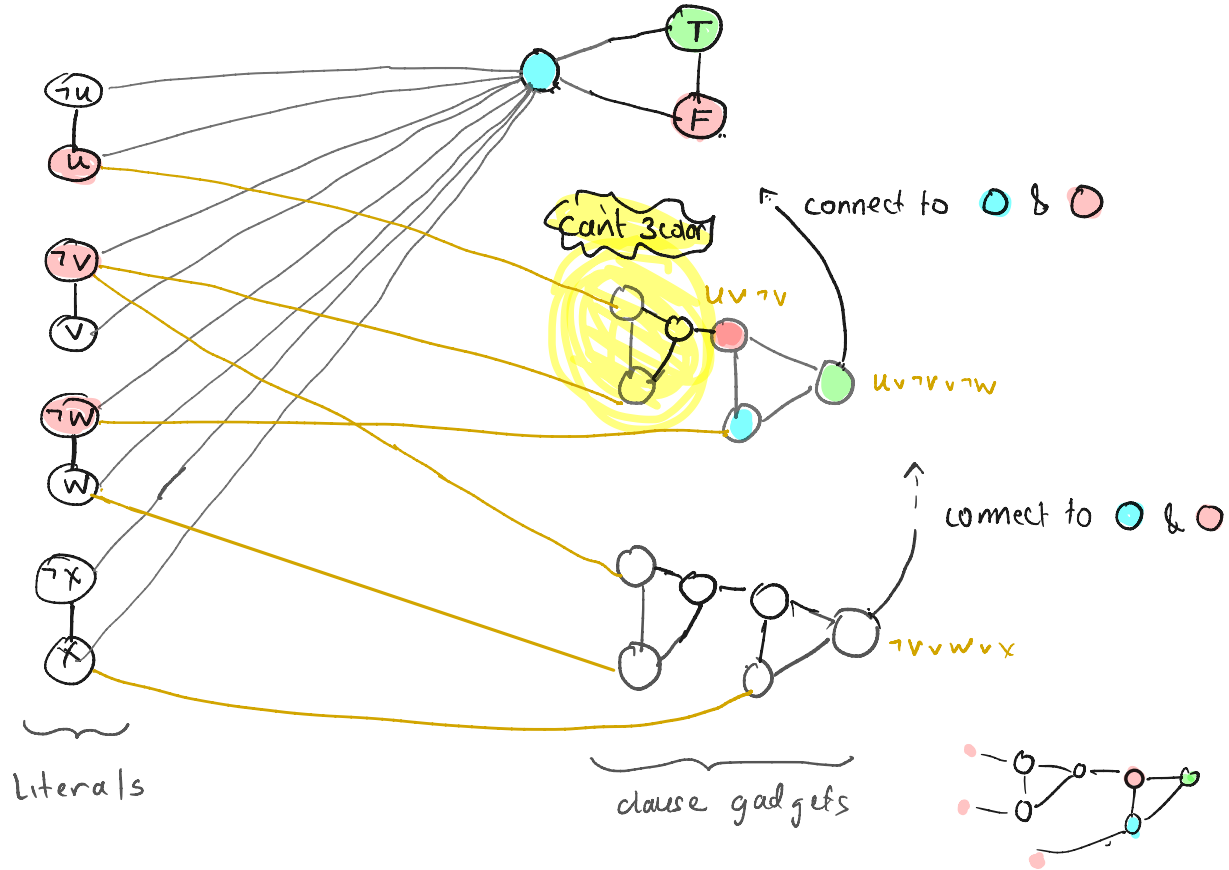
Example $F = \underbrace{(u \vee \neg v \vee \neg w)}_{C_1} \wedge \underbrace{(\neg v \vee w \vee x)}_{C_2}$

* Each clause gadget is 3-colorable iff at least one of the input literals is colored $\textcircled{T}$



Example $F = \underbrace{(u \vee \neg v \vee \neg w)}_{C_1} \wedge \underbrace{(\neg v \vee w \vee x)}_{C_2}$

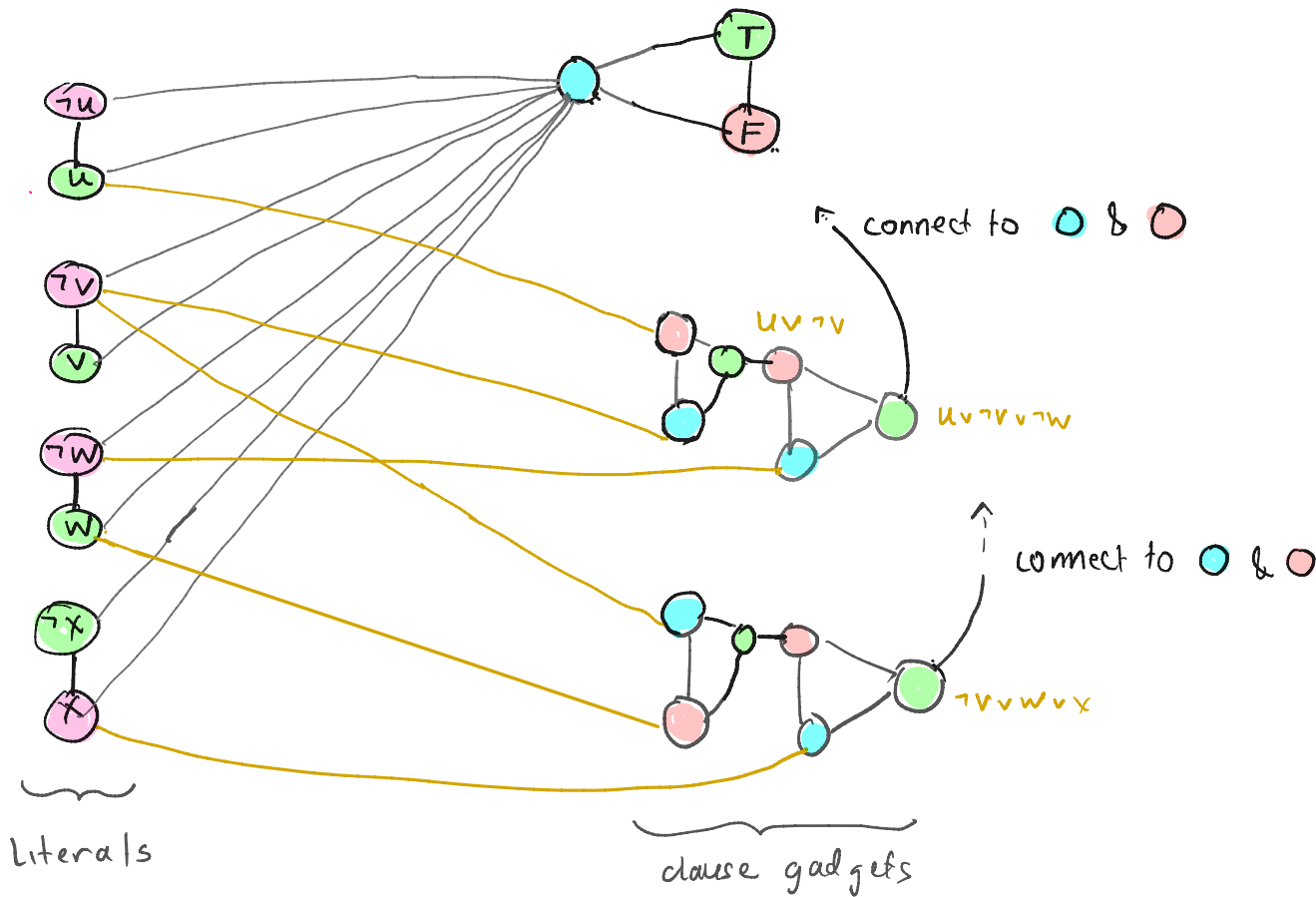
* Each clause gadget is 3-colorable iff at least one of the input literals is colored T



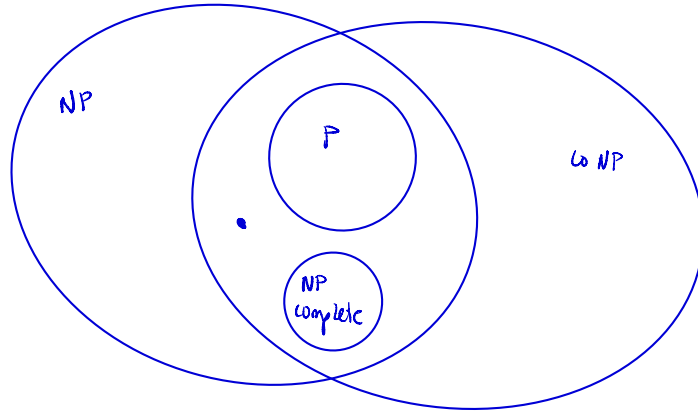
Example

$$F = (\underbrace{u \vee \neg v \vee \neg w}_{C_1}) \wedge (\underbrace{\neg v \vee w \vee x}_{C_2})$$

$u = 1$
 $v = 1$
 $w = 1$
 $x = 0$



If $P \neq NP$:



$L \in NP$ iff there is a verifier $V(x, y)$ such that

$$x \in L \Leftrightarrow \exists y \ V(x, y) = 1$$

$L \in coNP$ iff there is a verifier $V(x, y)$ s.t.

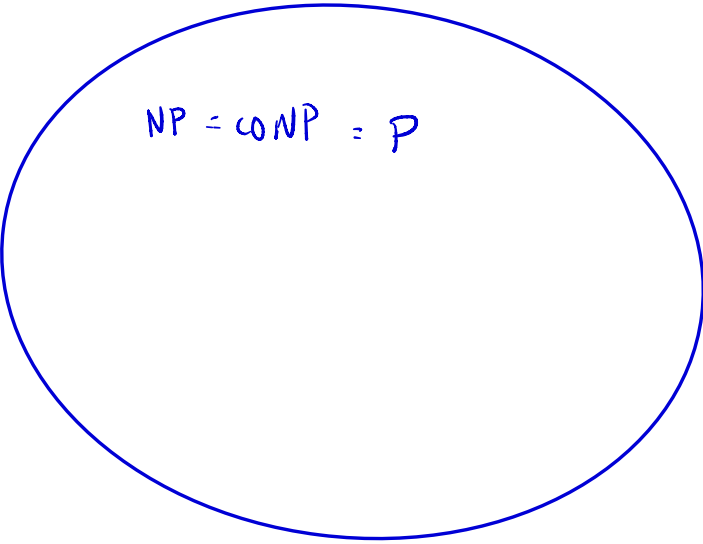
$$x \in L \Leftrightarrow \forall y \ V(x, y) \neq 1$$

3 UNSAT

Ladner's Thm: If $P \neq NP$ then there are languages in NP but not in P and not NP-complete

example: Decision version of Factoring: $L = \{(N, i) \mid i^{\text{th}} \text{ bit of largest factor of } N \text{ is } 1\}$

If $P = NP$:


$$NP = \text{coNP} = P$$

Time hierarchy Theorem (sipser 9.1)

$$\text{TIME}(n^2) \subsetneq \text{TIME}(n^3)$$

(There is a problem $L \in \{0,1\}^*$ solvable in time $O(n^3)$
but not in time $O(n^2)$.)

Actually shows: $P \subsetneq E \subsetneq EXP$
 ↑ ↑
 time($2^{O(n)}$) time($2^{\text{poly}(n)}$)

Time Hierarchy Thm

Assume $t_1(n) \ll t_2(n)$. Ex $t_1(n) = n^2$ $t_2(n) = n^5$

Then $\exists L \in \text{Time}(t_2(n))$, but $L \notin \text{TIME}(t_1(n))$

Notation (encoding)

z an object of type T

$\langle z \rangle_T :=$ standard encoding of z

Ex 1 $z = \mathbb{N}$ $\langle z \rangle_{\mathbb{N}} =$ binary encoding of z

Ex 2 $z = \text{all TMs}$ $\langle z \rangle_{\text{TM}} =$ binary encoding of TM z

PF sketch high level is diagonalization

L will be defined by a time $t_2(n)$ TM, D , that decides it

	M_1	M_2	M_3	...
x_1	0			
x_2		0		
x_3			0	
$\vdots$				$\ddots$

Columns: all TMs

rows: all $x \in \{0,1\}^*$ that encode
a TM

so entry (x_i, M_i) :

M_i encoded by x_i

that is, $\langle M_i \rangle_{TM} = x_i$

PF sketch high level is diagonalization

	M_1	M_2	M_3	$\vdots$
x_1	0			
x_2		0		
x_3			0	
$\vdots$				$\ddots$

L will be defined by a time $t_2(n)$ TM, D , that decides it

D on input x_i :

- (1) Let $\langle M_i \rangle_{TM} = x_i$. (if x_i is not code of any TM, halt & reject)
- (2) Use $t_2(|x_i|)$ time to simulate $M_i(x_i)$ for as many steps as it can
- (3) If $M_i(x_i)$ finishes and accepts in this much time $\Rightarrow D(x_i)$ halts & rejects
- (4) Else $[M_i(x_i)]$ rejects or doesn't finish in this much time $\Rightarrow D(x_i)$ halts & accepts

D on input x_i :

- (1) Let $\langle M_i \rangle_{TM} = x_i$ (if x_i is not code of any TM, halt & reject)
- (2) Use $t_2(|x_i|)$ time to simulate $M_i(x_i)$ for as many steps as it can
- (3) If $M_i(x_i)$ finishes and accepts in this much time $\Rightarrow D(x_i)$ halts & rejects
- (4) Else $[M_i(x_i)]$ rejects or doesn't finish in this much time $\Rightarrow D(x_i)$ halts & accepts

Define $L = \{x \mid D(x) \text{ accepts}\}$

Claim 1: $L \in \text{TIME}(t_2(n))$ by construction

Claim 2: $L \notin \text{TIME}(t_1(n))$ for $t_1(n) \ll t_2(n)$

Claim 2 $L \notin \text{TIME}(t_1(n))$ for $t_1(n) \ll t_2(n)$

Proof

• $D(x)$ runs in time $t_2(|x|)$.

- Assumption: $\forall x_i$ $D(x_i)$ is able to simulate TM $M_i = \langle x_i \rangle_{TM}$ for at least $t_1(|x_i|)$ steps.
- Assumption implies $L \notin \text{TIME}(t_1(n))$ since $\forall M_i$ that halts in $t_1(n)$ time, $D(x_i)$ halts and $D(x_i) \neq M_i(x_i)$ by construction

Assumption is true: overhead of simulating M on x for $t_1(|x|)$ steps:

say $t_1(n) = n^2$

D on x_i ($\langle M_i \rangle = x_i$):

- Tape 1: "clock": stores time steps on tape 1
- Tape 2: store contents of M_i tapes
- Tape 3: store program x_i (encoding M_i)
- Tape 4: Workspace for D

Assumption is true: overhead of simulating M on x for $t_i(|x|)$ steps:

say $t_i(n) = n^2$

D on x_i ($\langle M_i \rangle = x_i$):

Tape 1: "clock": stores time step on tape 1

Tape 2: store contents of M_i tapes

Tape 3: store program x_i (encoding M_i)

Tape 4: Workspace for D

To simulate one step of M_i on x_i :

- Scan tape 2 to write on tape 4 the symbols under M_i 's heads
- Scan Tapes 3, 4 to figure out M_i 's next transition and update Tape 2
- Increment clock (Tape 1) by +1

Assumption is true: overhead of simulating M on x for $t_1(|x|)$ steps:

say $t_1(n) = n^2$

D on x_i ($\langle M_i \rangle = x_i$):

Tape 1: "clock": stores time step on tape 1

Tape 2: store contents of M_i tapes

Tape 3: store program x_i (encoding M_i)

Tape 4: Workspace for D

To simulate one step of M_i on x_i :

- Scan tape 2 to write on tape 4 the symbols under M_i 's heads $\leftarrow O(t_1(|x_i|)^2)$
- Scan Tapes 3, 4 to figure out M_i 's next transition and update Tape 2 $\leftarrow O(t_1(|x_i|)^2)$
- Increment clock (Tape 1) by +1 $\leftarrow O(t_1(|x_i|)^2)$

$\therefore$ can simulate one step of M_i in time $O(t_1(|x_i|)^2) \leq t_1(|x_i|)^3$

so if $t_1(n) = n^2$ can pick $t_2 = (n^2)^3 = n^6$

Better simulation is possible:

$$\forall t_1(n) \geq n \text{ and } t_2(n) \geq t_1(n) \cdot \log(t_1(n))$$

$$\exists L \in \text{TIME}(t_2(n)) , L \notin \text{TIME}(t_1(n))$$



says for example : for any $k \geq 2$ $k \in \mathbb{N}$
there exists $L \in \text{TIME}(n^k)$ but
 $L \notin \text{TIME}(n^{k-1})$

Space Complexity

Defn A TM M has space complexity $S(n)$ if

$\forall x, |x|=n$, M scans $\leq S(n)$ cells on every work tape on input x (need not halt)

Defn $L \in \text{SPACE}(S(n))$ if $\exists$ a deterministic TM M that decides L and has $\text{SPACE } O(S(n))$

Note space can be $\ll$ length of input

Defn $\text{LogSPACE} = \text{SPACE}(\log n)$

$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k)$

Space Complexity

Defn A non-det TM M has non-det space complexity $S(n)$ if

$\forall x, |x|=n$, M scans $\leq S(n)$ cells on every work tape on input x (need not halt)

Defn $L \in \text{NSPACE}(S(n))$ if $\exists$ a non-deterministic TM M that decides L and has $\text{NSPACE } O(S(n))$

Note space can be $\ll$ length of input

Defn " L " = $\text{LogSPACE} = \text{SPACE}(\log n)$ " NL " = $\text{NSPACE}(\log n)$

$$\text{PSPACE} = \bigcup_{k \geq 0} \text{SPACE}(n^k)$$

Example Reachability

Input: Directed graph g , vertices s, t

Accept iff there is a (directed) path from s to t in g

If $g = (V, E)$ $|V| = n$ then $|g| = n^2$ (adjacency matrix)

so input $\langle g, s, t \rangle$ has length $n^2 + 2\log n = O(n^2)$

Lemma 1 Reachability $\in \text{NSPACE}(\log n)$

Nondeterministic SPACE $\log n$ algorithm:

Idea: "guess" path from s to t . Do guesses online to keep space $O(\log n)$

Worktape 1: stores current vertex visited ($\log n$ bits)

initially tape 1 contains s

Worktape 2: guess next vertex to visit ($\log n$ bits)

Worktape 3: counter in binary (halt when counter $\geq n$)

Nondeterministic SPACE $\log n$ algorithm:

Idea: "guess" path from s to t . Do guesses online to keep space $O(\log n)$

Worktape 1: stores current vertex visited ($\log n$ bits)
initially tape 1 contains s

Worktape 2: guess next vertex to visit ($\log n$ bits)

Worktape 3: counter in binary (halt when counter $\geq n$)
initially $t = 1$

Loop:

- If $t = n+1$ halt + reject
- Let v_1 be vertex Name stored on tape 1
- guess a vertex v_2 + store Name in binary on tape 2
- If edge $(v_1, v_2) \in E$ then halt + reject
- Else: replace v_1 with v_2 on tape 1
- If $v_2 = t$ halt + accept
- $t = t + 1$

Lemma 2 Reachability $\in \text{SPACE}(\log^2 n)$

← deterministic $\log^2 n$ space

Idea: Divide and conquer / recursive algorithm

$\exists$ path from $s \rightsquigarrow t$ of length $\leq n$ iff

$\exists$ vertex x such that: (i) $\exists$ path $s \rightsquigarrow x$ of length $\leq \frac{n}{2}$ and
(ii) $\exists$ path $s \rightsquigarrow t$ " " $\leq \frac{n}{2}$

↑
can be implemented using $O(\log^2 n)$ space

Main tape 1: stack that stores recursive history of calls

Recursive Alg:

PATH (x, y, i) :

if $i = 0$ then if $(x, y) \in E \Rightarrow \text{return TRUE}$
else $\Rightarrow \text{return FALSE}$

else ($i > 0$): for each $z \in V$:

If $\text{PATH}(x, z, i-1) = \text{True}$

and $\text{PATH}(z, y, i-1) = \text{TRUE} \Rightarrow \text{return TRUE}$

else $\Rightarrow \text{return FALSE}$

Theorem (Savitch) Let $S(n) \geq \log n$. Then

$$NSPACE(S(n)) \subseteq SPACE(S^2(n))$$

Pf (Key idea: a nondet space S alg can be viewed as reachability problem on a directed graph with $2^{O(S)}$ vertices.)

Let L be decided by a space $S(n)$ nondet TM, M

Let G_n be "configuration" graph of M (assume M has one work tape)

A "configuration" : contents of work tape, location of work head,
location of input head and value

since M uses space $\leq S(n)$, there are $\leq 2^{O(S(n))}$ distinct configurations

Vertices of $G_n = \{ \text{all possible configurations} \}$

$(v \rightarrow v')$ iff M can transition from config v to v' on one step.

Theorem (Savitch) Let $S(n) \geq \log n$. Then

$$NSPACE(S(n)) \subseteq SPACE(S^2(n))$$

Pf (Key idea: a nondet space $S(n)$ algorithm can be viewed as a reachability problem on a directed graph with $2^{O(S(n))}$ vertices

Let L be decided by a space $S(n)$ nondet TM, M

Let G_n be "configuration" graph of M (assume M has one work tape)

A "configuration" : contents of work tape, location of work head,
location of input head and value

since M uses space $\leq S(n)$, there are $\leq 2^{O(S(n))}$ distinct configurations

Vertices of $G_n = \{ \text{all possible configurations} \}$

$(v \rightarrow v')$ iff M can transition from config v to v' on one step.

Let v_x = start config on input x , and let v_{acc} = accept config.

then $x \in L \Leftrightarrow v_{acc}$ is reachable from v_x in G_n .

By Lemma 2, there is a deterministic space $S^2(n)$ algorithm $\square$