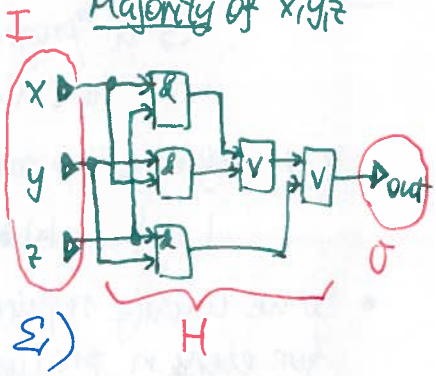
Brief detour: From formulas to Boolean circuits...

Df: A Combinatorial Circuit consists of:

- a finite alphabet Σ,
- finite sets I (input terminals) $= \{i_1, \dots, i_{|I|}\}$, O (output terminals) $= \{o_1, \dots, o_{|O|}\}$, H (gates) $= \{h_1, \dots, h_{|H|}\}$ — pairwise disjoint
- directed acyclic multigraph $(I \cup O \cup H, E)$
- arity: $a: H \to \mathbb{N}$
- assignment of functions to gates $F: h \mapsto (f_h : \Sigma^{a(h)} \to \Sigma)$
- assignment of gate inputs to incoming edges: $\mathcal{Q}: (u,v) \in E \mapsto i \in \{1 \dots a(v)\}$

Example: Majority of x,y,z



Where:

- $\forall i \in I \;\; \deg^{in}(i) = 0$
- $\forall o \in O \;\; \deg^{in}(o) = 1, \; \deg^{out}(o) = 0$
- $\forall h \in H \;\; \deg^{in}(h) = a(h) \;\; \& \;\; \forall i \in \{1 \dots a(h)\} \; \exists! \, (x,h) \in E : \mathcal{Q}((x,h)) = i$

Df: Boolean Circuit: Comb. circuit with $\Sigma = \{0,1\}$

Df: Computation of a circuit proceeds in steps.
(sketch of)

Step 0: input terminals and arity-0 gates (constants) have defined values.
Step $i+1$: gates whose input is defined in step at most i produce output.

As the graph is acyclic, gate outputs never change and every gate/terminal is defined within finite # steps.

$\Rightarrow$ the circuit computes a function from $\Sigma^{|I|}$ to $\Sigma^{|O|}$.

Bounding arity: Since a single gate of high arity can compute anything in 1 step, we will bound arity by 2. (Actually, any fixed constant >1 would work.)

Circuit complexity: Time $\approx$ # layers (# steps of computation), Space $\approx$ # gates — BTW circuits are an interesting model of parallel computing

Boolean formulas $\approx$ circuits with tree structure (except for inputs)

Lemma: Every function $f: \{0,1\}^k \to \{0,1\}^l$ can be computed by a Boolean circuit (in fact it's a formula in DNF) consisting only of AND, OR and NOT gates. (OR can be replaced by $\overline{\bar{x} \,\&\, \bar{y}}$)

Proof:
1. n-input AND/OR can be computed by a tree of 2-input ANDs/ORs.
2. Function with multiple-bit output: replace by l single-bit functions.
3. Function whose truth table contains exactly one 1:
   e.g. $\neg x_1 \,\&\, \neg x_2 \,\&\, \neg x_3 \,\&\, x_4$ ... 1 at position 0001
4. Truth table with multiple 1's: OR functions for each 1.
5. Otherwise it's constant 0.

(produces circuits of exponential size, but good enough for k, l constant)

Corollaries: ① can simulate arbitrary gates of fixed arity with $O(1)$ space/time overhead.
② can simulate arbitrary comb. circuit by a Boolean circuit (binary-encoded Σ)

Problem: A circuit handles inputs of constant size only.
↳ a "program" is a family of circuits $C_0, C_1, \ldots$ where C_n solves the problem for inputs of size n.

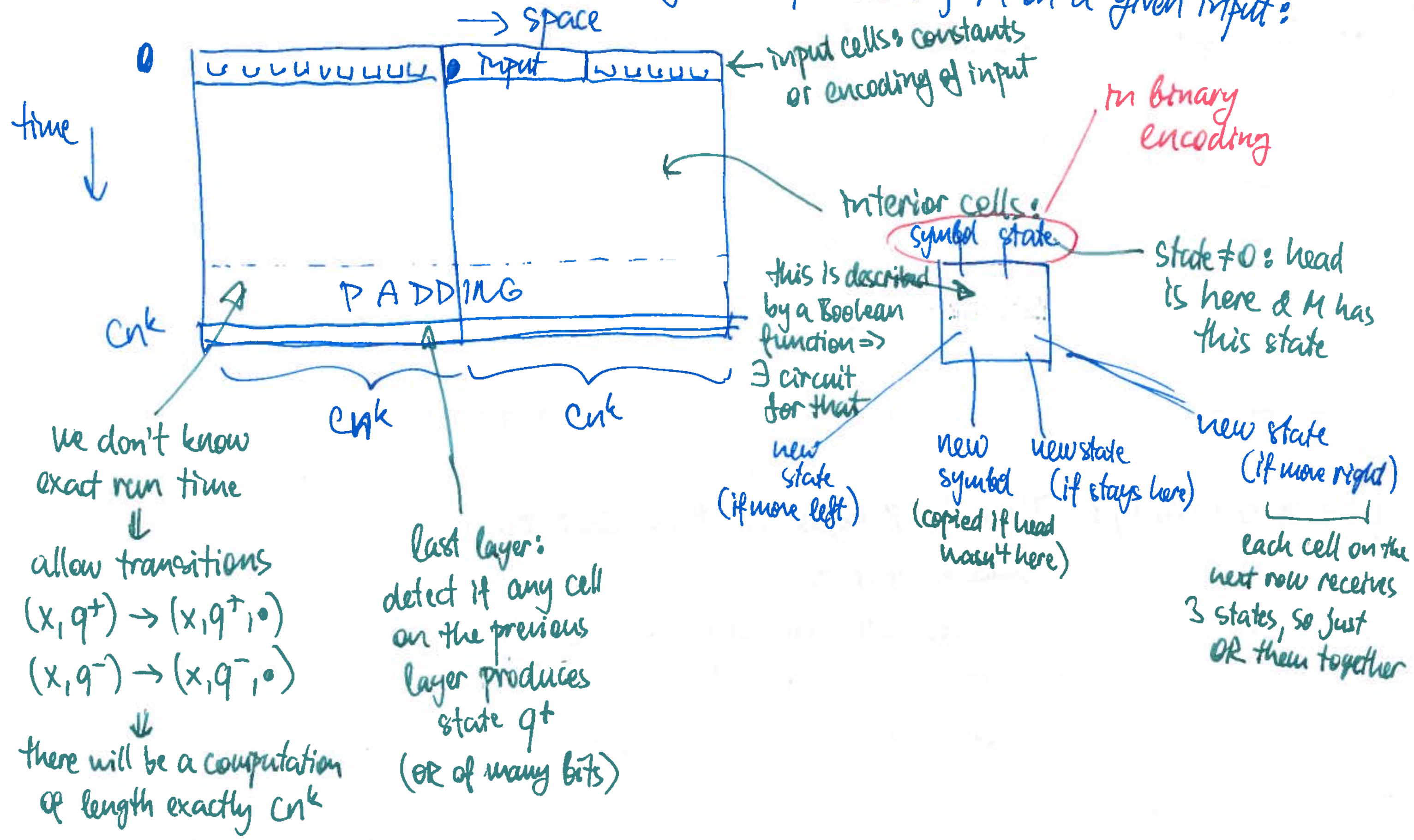
But: If we allow arbitrary sequences, we can compute undecidable problems:
$L = \{\alpha \mid |\alpha| \text{ written in binary} \in L_u\}$

- So we usually require the family to be uniform: there is an algorithm which for every n produces C_n in time $\mathrm{poly}(n)$.

So languages decidable by uniform circuit families $= P$

Theorem: For every $L \in P$ there is $f \in PF$ s.t. for every n, $f(n)$ is an (encoding of) Boolean circuit with n inputs and 1 output which decides L for strings of length n.
(with obvious meaning (computes char. function of L))

Proof: Let M be a 1-tape (WLOG) TM deciding L in time at most $c \cdot n^k$ for some $c, k \in \mathbb{N}$.
We will build a circuit producing a computation of M on a given input:



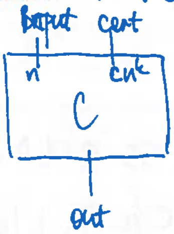
Df: CIRCUIT-SAT: given a Boolean circuit with 1 output, is there an input for which the output is true?

↳ Obviously, this is in NP.

Thm: CIRCUIT-SAT is NP-complete.

Proof: • When reducing from $L \in NP$ to C.-SAT: consider verifier $V \in P$ & upper bound cn^k for certificate size.

- Adapt verifier to accept certificates of size exactly cn^k (using reversible padding like 010^*)
- Find Boolean circuit for V on inputs of size $n + cn^k$:

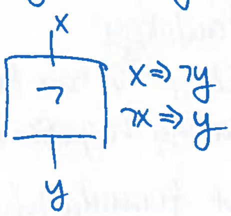


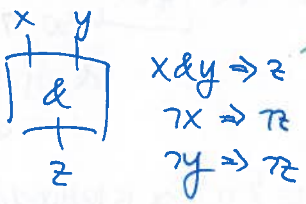
& fix input terminals to input α
↓
SAT for $C_\alpha(\text{cert})$ computes $\alpha \in L$

] done inside the reduction when receiving input α of size n

Lemma: CIRCUIT-SAT $\leq^P_m$ SAT.

Proof: Assume WLOG that all gates are AND and NOT.
Introduce new variables for gate outputs.
Add consistency-checking clauses:





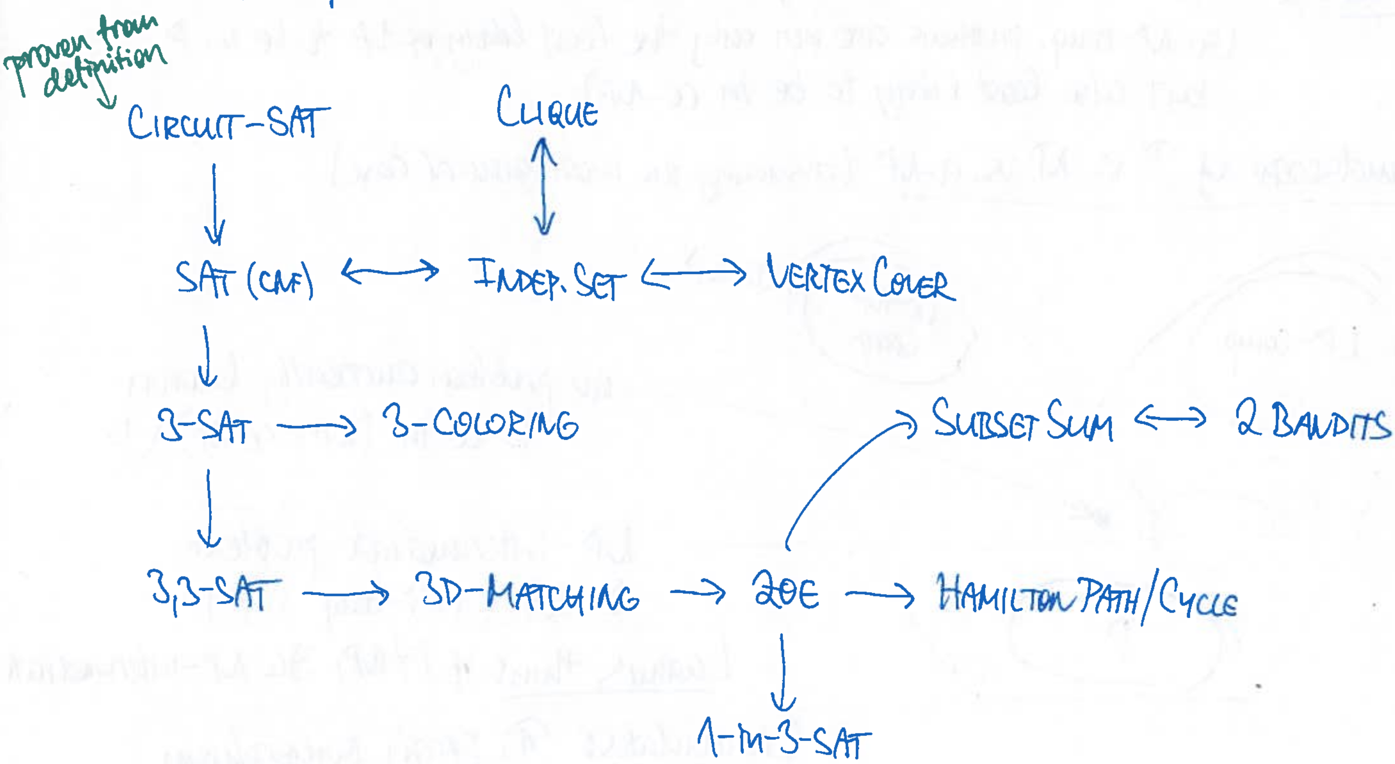
— this is $\neg x \vee \neg y \vee z$ in CNF

] this also means that SAT for general (non-CNF) formulas is reducible to SAT

BTW we produced an instance of 3-SAT :)

Corollary: SAT is NP-complete. [This is Cook-Levin theorem!]

Map of NP-complete problems we encountered until now:



Further SAT variants: 2-SAT is in P
E3,E3-SAT (clauses of size exactly 3, vars have exactly 3 occurrences)
surprise: all instances satisfiable!

The class co-NP

Df: For a language $L \subseteq \{0,1\}^*$ we define its complement $\overline{L} := \{0,1\}^* \setminus L$

Df: For a class $\mathcal{C}$ of languages: $co\text{-}\mathcal{C} := \{\overline{L} \mid L \in \mathcal{C}\}$ ⟶ ☼ $co\text{-}P = P$

Let's study co-NP...

- $P \subseteq NP \cap co\text{-}NP$ ... open if the inclusion is strict
- if $P = NP$, then $NP = co\text{-}NP$
- if $NP \neq co\text{-}NP$, then $P \neq NP$ (because $P = co\text{-}P$)
- as $K \leq^P_m L \Leftrightarrow \overline{K} \leq^P_m \overline{L}$, we have: L is NP-complete $\Leftrightarrow$ $\overline{L}$ is co-NP-complete
- certificate-based def.: $L \in co\text{-}NP \equiv \exists V \in P: (\alpha \in L \Leftrightarrow \forall \beta \in \{0,1\}, |\beta| \leq poly(|\alpha|)\ V(\alpha\beta))$

↳ so $\overline{SAT}$ is co-NP-complete

↑ this is not UNSAT (unsatisfiability for CNF), because for strings which do not encode a formula, we still have to answer 1 in $\overline{SAT}$ but 0 in UNSAT

↳ but $\overline{SAT} \leq^P_m$ UNSAT, so UNSAT is co-NP-comp.

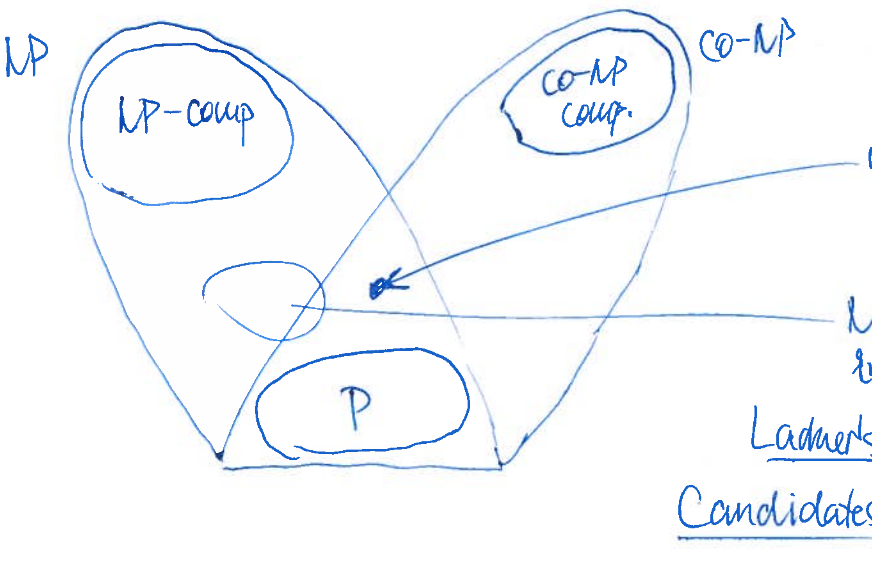
↳ $\neg \exists \vec{x}\ \varphi(\vec{x}) \Leftrightarrow \forall \vec{x}\ \neg\varphi(x)$ ⟶ so $\neg\varphi$ is a tautology & if φ is in CNF, $\neg\varphi$ can be written in DNF by propagating negation

So: TAUTOLOGY $:= \{\alpha \mid \alpha$ is (encoding of) DNF formula which is tautological$\}$ is also co-NP-complete (this is the most standard co-NP-c. problem)

Formally: TAUTOLOGY $\leq^P_m$ UNSAT $\leq^P_m$ $\overline{SAT}$

Exercise: If $L \in co\text{-}NP$ is NP-complete, then $NP = co\text{-}NP$.
(so NP-comp. problems are not only the least likely of NP to be in P, but also least likely to be in co-NP).

Landscape of P vs. NP vs. co-NP (assuming the most general case)



no problem currently known to be in $(NP \cap co\text{-}NP) \setminus P$

NP-intermediate problems in $NP \setminus (NP\text{-comp.} \cup P)$

Ladner's thm: if $P \neq NP$, $\exists L$ NP-intermediate.

Candidates: ① graph isomorphism (known to be in $DTIME(n^{\log n})$)

② factoring (given x, a, b: does x have a factor in $[a,b]$?)