Exercises: ① IndepSet $\leq^P_m$ Clique

② 3-Coloring $\leq^P_m$ SAT

Formalization of "search problems":

Df: Class of languages NP:

$L \in NP \equiv \exists V \in P$ (verifier)

$\forall \alpha \in \Sigma^*: \alpha \in L \Leftrightarrow \left(\exists \beta \in \Sigma^*: |\beta| \in \text{poly}(|\alpha|) \;\&\; V(\alpha, \beta) \right)$

(β: certificate, poly: polynomial size, V: which is accepted by the verifier)

👁 $P \subseteq NP$ ... verifier does all the work & ignores β

👁 resembles proofs in logic: true statements have a proof, which is easy to verify; for false statements, no proof passes verification

Big question: Is $P = NP$?

↑ 1MB price by Clay Mathematical Institute (waits for you ☺)

Df: Language L is NP-hard $\equiv \forall K \in NP: K \leq^P_m L$

L is NP-complete $\equiv$ furthermore, $L \in NP$

Lemma: Let $K \leq^P_m L$. Then:

① if $L \in NP$, then $K \in NP$ (just compose verifier with reduction) — NP is closed under reductions

② if K is NP-hard, then L is NP-hard. ($\forall M \in NP: M \leq K \leq L \Rightarrow M \leq L$)

②' if K is NP-complete & $L \in NP$, then L is NP-complete.

(② and ②' make it easy to prove NP-completeness once we have one NP-comp. problem)

Lemma: If $L \in P$ is NP-complete, then $P = NP$.

Proof: $P \subseteq NP$ is trivial, will prove $NP \subseteq P$:

Let $K \in NP$. Then $K \leq L$, which implies $K \in P$.

Thm (Cook-Levin): SAT is NP-complete.

↳ will be proven later

MORE REDUCTIONS

3D MATCHING

Input: disjoint sets B (boys), G (girls), C (cats)

$J \subseteq B \times G \times C$ (triples)

Output: $\exists J' \subseteq J$ s.t. each element of $B \cup G \cup C$ is contained in exactly 1 triple in J'

(generalizes bipartite matching, which is in P)

3-SAT: SAT, but all clauses contain at most 3 literals (generally: k-SAT)

3,3-SAT: Furthermore, every variable occurs in at most 3 clauses. (generally: k,ℓ-SAT)
[Extension: every literal occurs at most 2 times – i.e., the 3 occurs of a variable aren't all positive nor all negative.]

Reduction: SAT $\leq^P_m$ 3-SAT

$(l_1 \vee l_2 \vee __ \vee l_k)$ → $(l_1 \vee l_2 \vee t) \wedge (l_3 \vee __ \vee l_k \vee \neg t)$

"long" clause with $k>3$ literals — replace by — $l_1 \vee l_2 \vee t$: 3 literals; $l_3 \vee __ \vee l_k \vee \neg t$: $k-1$ literals; t: new variable

New formula is satisfiable ⇔ the old one was.

Iterate until all long clauses are broken.

Reduction: 3-SAT $\leq^P_m$ 3,3-SAT

Replace variable x with $k>3$ occurrences by new variables $x_1 __ x_k$.

Add clauses $(x_1 \Rightarrow x_2), (x_2 \Rightarrow x_3), \ldots, (x_{k-1} \Rightarrow x_k), (x_k \Rightarrow x_1)$

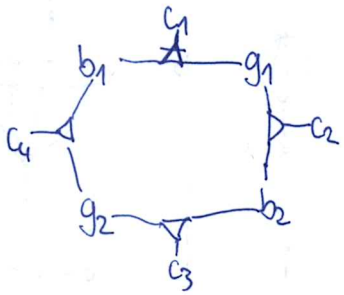
Preserves satisfiability. ($(x_2 \Rightarrow x_3)$ this is $\neg x_2 \vee x_3$)

Each new variable has at most 2 positive & at most 2 negative occurrences.

Can apply the transform for $k=3$, too.

Reduction: 3,3-SAT $\leq^P_m$ 3D-MATCHING

Choice gadget (for each variable)



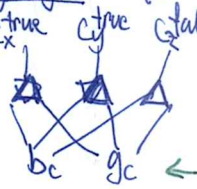
b_1, b_2, g_1, g_2 unique for this gadget

$c_1 \ldots c_4$ shared with clause gadgets

2 states: c_1, c_3 free ← logical 0

and c_2, c_4 free ← logical 1

Clause gadget (also called consistency gadget) for $x \vee y \vee \neg z$



b_c, g_c ← unique for this gadget

1 of the cats which are free if x is true (that is c_2^* or c_4)

we have (2 · #variables − #clauses)

free cats ⇒ add this many pairs of "universal cat lovers" which have triples for every cat

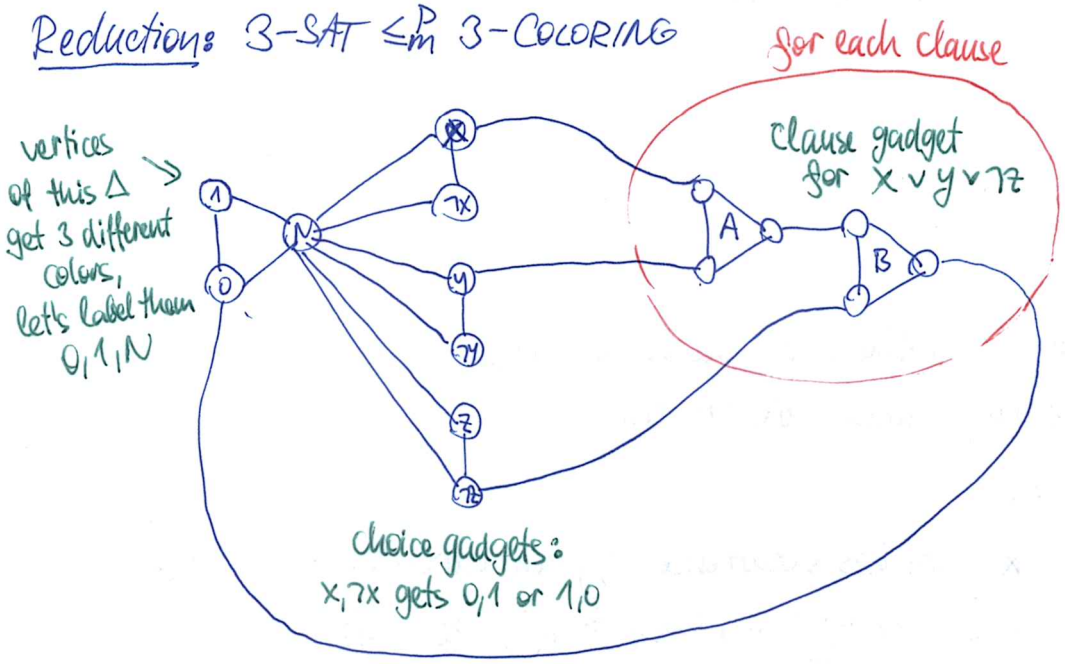
each literal occurs at most 2 times in 3,3-SAT formulas, so we have enough cats for all literals

∃ matching ⇔ ∃ satisfying assignment

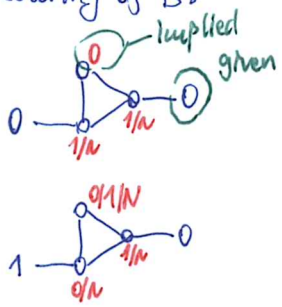
Exercise 8 3D-MATCHING $\leq_m^P$ ZOE (zero-one equations)

↳ & show that restriction of ZOE to equations with exactly 3 variables stays NP-complete.
[This is sometimes called 1-in-3-SAT: exactly 1 literal must be true ... no negations are needed. There also exists a direct reduction from 3-SAT to this problem.]

Reduction: 3-SAT $\leq_m^P$ 3-COLORING

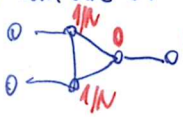


= coloring of B:



= coloring of A:

- if inputs contain 1: use coloring of B which passes 0
- if both are 0:

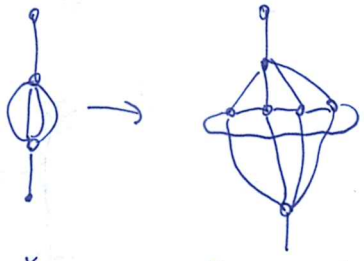


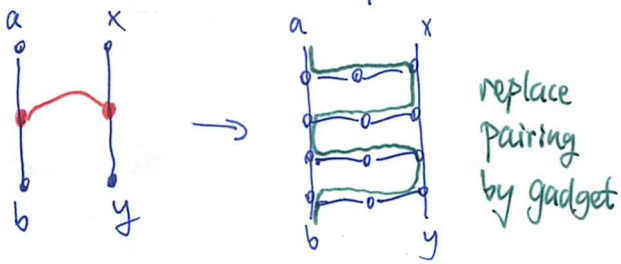
need 1 where, so the 3rd literal must be 1

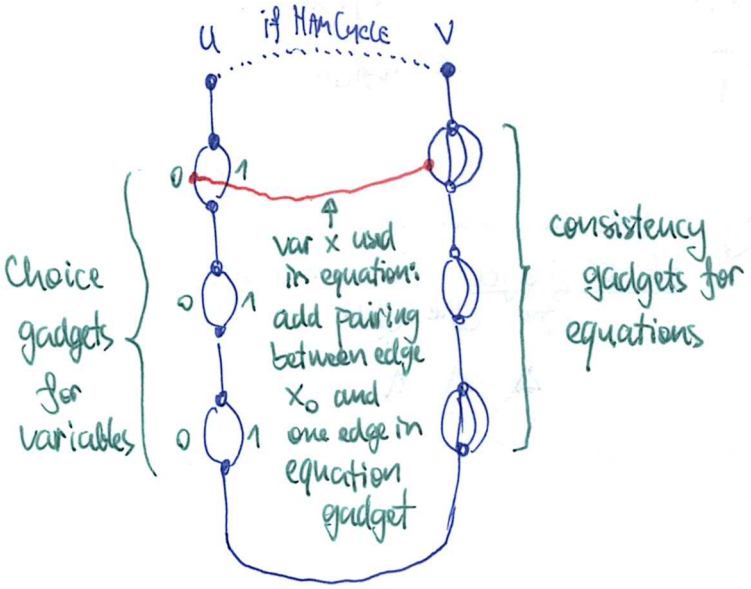
Reduction: ZOE $\leq_m^P$ HAMILTON ~~CYCLE~~ or PATH

First consider problem HAMPATH* which allows:
- parallel edges
- pairing: e ~~—~~ f ≡ must use exactly 1 edge of e, f

reduce to plain HAMILTON PATH







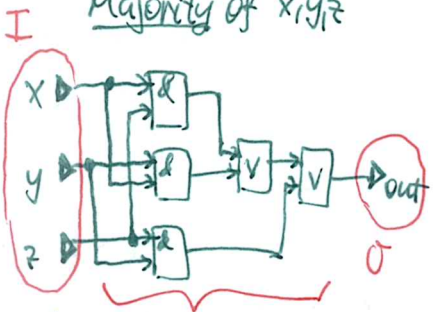
Exercises:
- SUBSET SUM problem is NP-complete: Given a finite set $X \subseteq \mathbb{N}$, $s \in \mathbb{N}$, is there $X' \subseteq X$ s.t. $\sum_{a \in X'} a = s$? (Hint: reduce from ZOE)
- 2 BANDITS: given finite $X \subseteq \mathbb{N}$, is there $X' \subseteq X$ s.t. $\sum X' = \sum (X \setminus X')$? Also NP-complete.

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: $\ell: (u,v) \in E \mapsto i \in \{1 \dots a(v)\}$

Example: Majority of x, y, z



Where:

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

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.

→ Layer #i: gates/terminals that produce output in time i, but not earlier



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

⇒ 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 (and constants). (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. (26)
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.

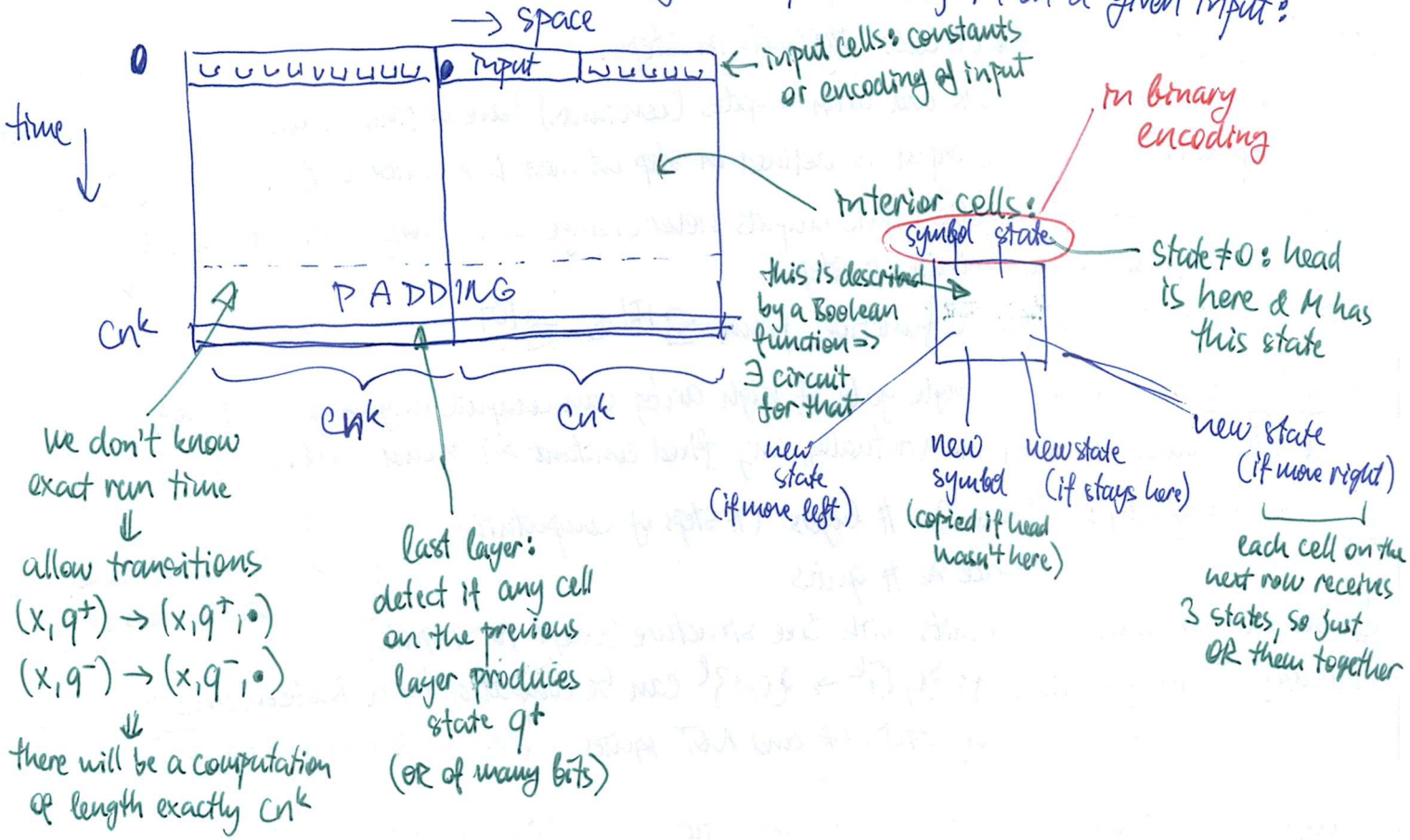
But: If we allow arbitrary sequences, we can compute undecidable problems:
$L = \{\alpha \mid |\alpha| \text{ written in binary with leading 1 removed} \in L_u\}$ (arbitrary enumeration of binary strings)

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

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.

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



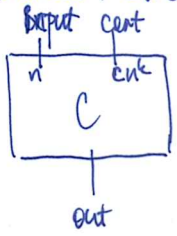
Df: CIRCUIT-SAT: given a Boolean circuit with 1 output, is there an input for which the output is true?

↳ 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 10*)
- Find Boolean circuit for V on inputs of size $n + cn^k$:



& fix input terminals to input α

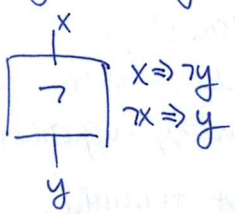
↓

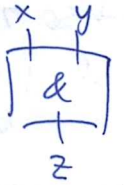
SAT for $C_\alpha(\text{cert})$ computes $\alpha \in L$

done inside the reduction when receiving input α of size n

Lemma: CIRCUIT-SAT $\leq_m^P$ SAT.

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





$x \& y \Rightarrow z$
$\neg x \Rightarrow \neg z$
$\neg y \Rightarrow \neg z$

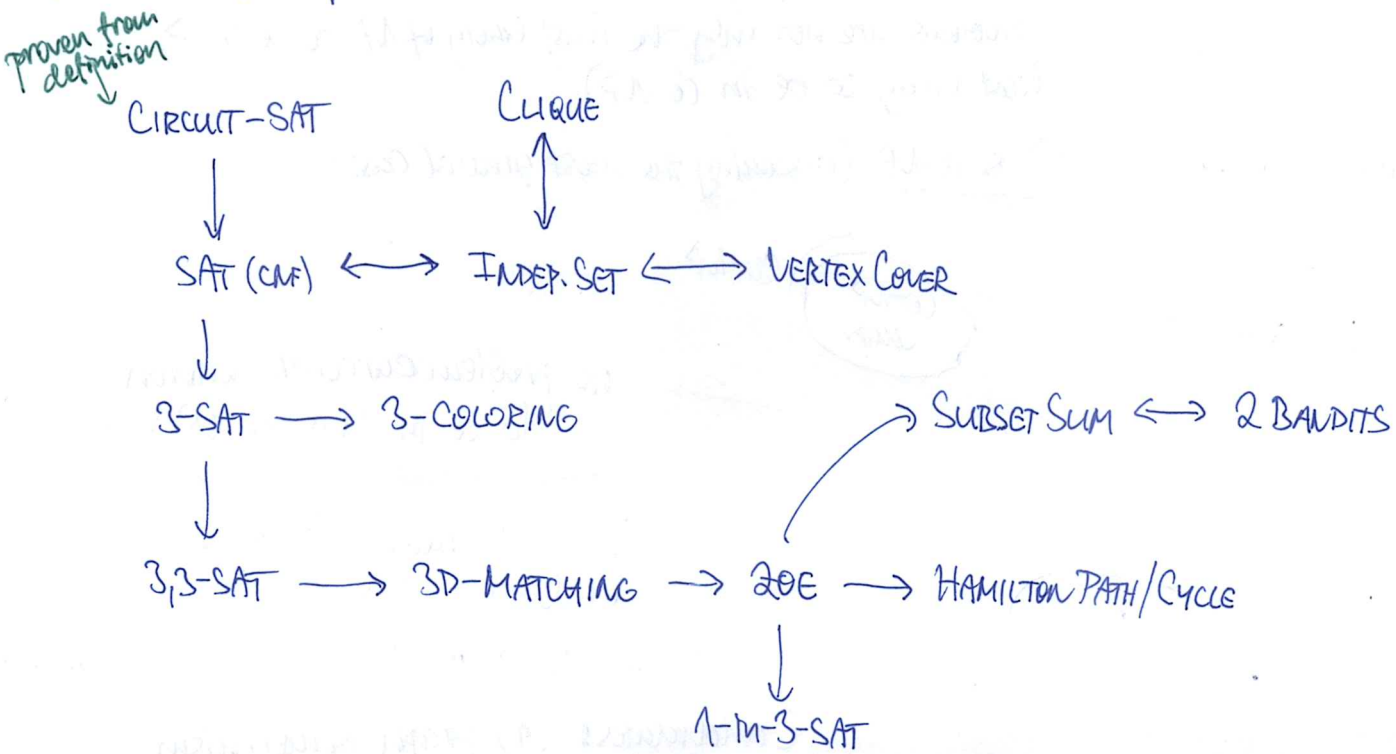
this is $\neg x \vee \neg y \vee z$ in CNF

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 ?