P, NP, EXP

P. Decision problems for which there is a poly-time algorithm.
EXP. Decision problems for which there is an exponential-time algorithm.
NP. Decision problems for which there is a poly-time certifier.

Claim. $P \subseteq NP$.
Pf. Consider any problem X in P.

- By definition, there exists a poly-time algorithm $A(s)$ that solves X.
- Certificate: $t = \varepsilon$, certifier $C(s, t) = A(s)$. ▪

Claim. $NP \subseteq EXP$.
Pf. Consider any problem X in NP.

- By definition, there exists a poly-time certifier $C(s, t)$ for X.
- To solve input s, run $C(s, t)$ on all strings t with $|t| \leq p(|s|)$.
- Return `yes`, if $C(s, t)$ returns `yes` forany of these. ▪

The Main Question: P Versus NP

Does P = NP? [Cook 1971, Edmonds, Levin, Yablonski, Gödel]

- Is the **decision** problem as easy as the **certification** problem?
- Clay $1 million prize.

would break RSA cryptography
(and potentially collapse economy)

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, …

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, …

Consensus opinion on P = NP? **Probably no.**

The Simpson's: P = NP?

Copyright © 1990, Matt Groening

Futurama: P = NP?

8.4 NP-Completeness

Polynomial Transformation

Def. Problem X polynomially reduces (**Cook**) to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of **calls** to oracle that solves problem Y.

Def. Problem X polynomially transforms (**Karp**) to problem Y if $F: \Sigma^* \rightarrow \Sigma^*$ exists such that:

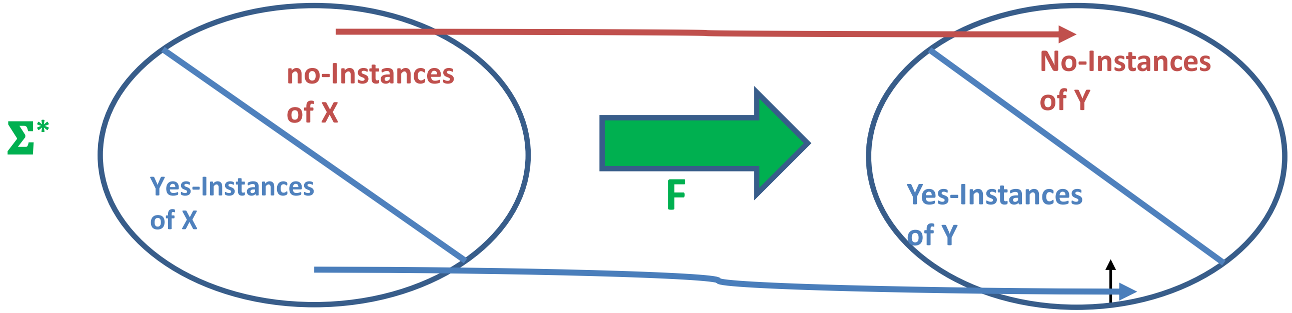
1. F can be computed in poly(|x|) time
2. x is a `yes` instance of X **iff** y= F(x) is a `yes` instance of Y

Prop. 1 implies |y| to be of size polynomial in |x|

Note. **Polynomial transformation** is **polynomial reduction** with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all our reductions will be of this form.

Open question. Are these two concepts the same with respect to NP?

we abuse notation $\leq_p$ and blur distinction



POLYNOMIAL (KARP) REDUCTIONS: Algorithmic use

THM 1. IF $X \leq_p Y$ and $Y \in P$ THEN $X \in P$ (So, class P is closed w.r.t. $\leq_p$)

Proof. By Hyp. there is a poly reduction $F: \Sigma^* \rightarrow \Sigma^*$ from X to Y and a deterministic **poly-time** algorithm *ALG* solving Y. Let p() and g() the polynomial time-**bounds** for computing F and *ALG*, respectively

Then, consider any instance $x \in \Sigma^*$ and make the following steps:

1. Compute $F(x) = y \in \Sigma^*$; (Note: Time is $p(|x|)$ and $|y| <= p(|x|)$)
2. Compute *ALG*(y); (Note: Time is $g(|y|) <= g(p(|x|)$ so it all poly(|x|) !)
3. If *ALG*(y) = yes THEN return yes ELSE return no

$$x \xrightarrow{F} y \xrightarrow{ALG} \text{YES/NO}$$

POLYNOMIAL REDUCTIONS: NP-Completeness

Def. A problem Y is **NP-Complete** **if:**

1. Y ∈ NP
2. For every problem X in **NP**, it holds: $X \leq_p Y$.

THM 2. Suppose Y is an **NP-complete** problem. Then:

Y is solvable in poly-time (i.e. Y ∈ P) **iff** P = NP

Proof. ⇐ If **P = NP** then Y can be solved in **poly-time** since Y ∈ **NP**.

Proof. ⇒ Suppose Y can be solved in **poly-time**.

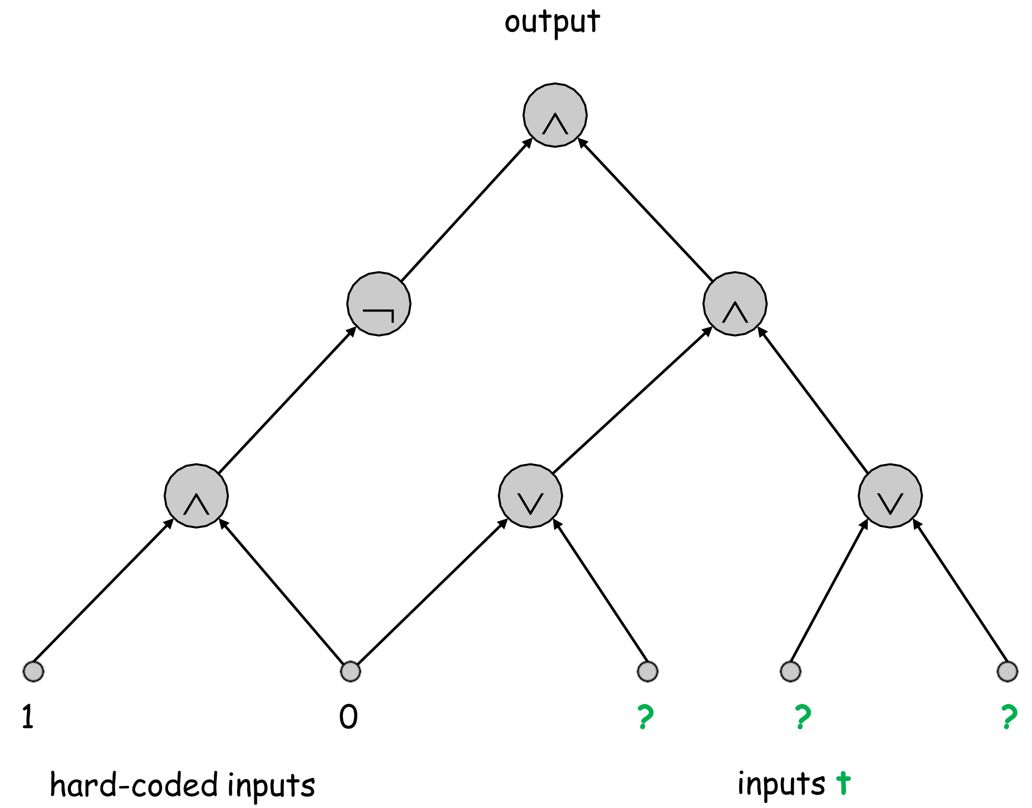
- Let X be any problem in **NP**. Since $X \leq_p Y$, from THM 1, we can solve X in **poly-time**.
- This implies NP ⊆ P.
- We already know. P ⊆ NP. Thus **P = NP**. ▪

Fundamental question. Do there exist "natural" **NP-complete** problems?

A first NP-Complete Problem: Circuit Satisfiability

CIRCUIT-SAT. Given a combinational **circuit K** built out of **AND**, **OR**, and **NOT** gates, is there a way to set the circuit **inputs** so that the **output** is 1? Namely, is circuit $K(x_1,x_2,\ldots,x_n\,;\,t_1,t_2,\ldots,t_m)$ satisfiable ?

Answer: yes!
and the **certifier** is
t = 1 0 1



The "First" NP-Complete Problem

THM. CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

Pf. (sketch)

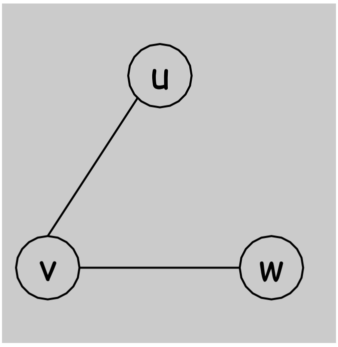
- Any **algorithm** that takes a fixed number of bits N as input and produces a yes/no answer can be represented by such a **circuit**. Moreover, if algorithm takes **poly-time**, then circuit is of **poly-size**.

sketchy part of proof; fixing the number **n** of bits is important, and reflects basic distinction between algorithms and circuits

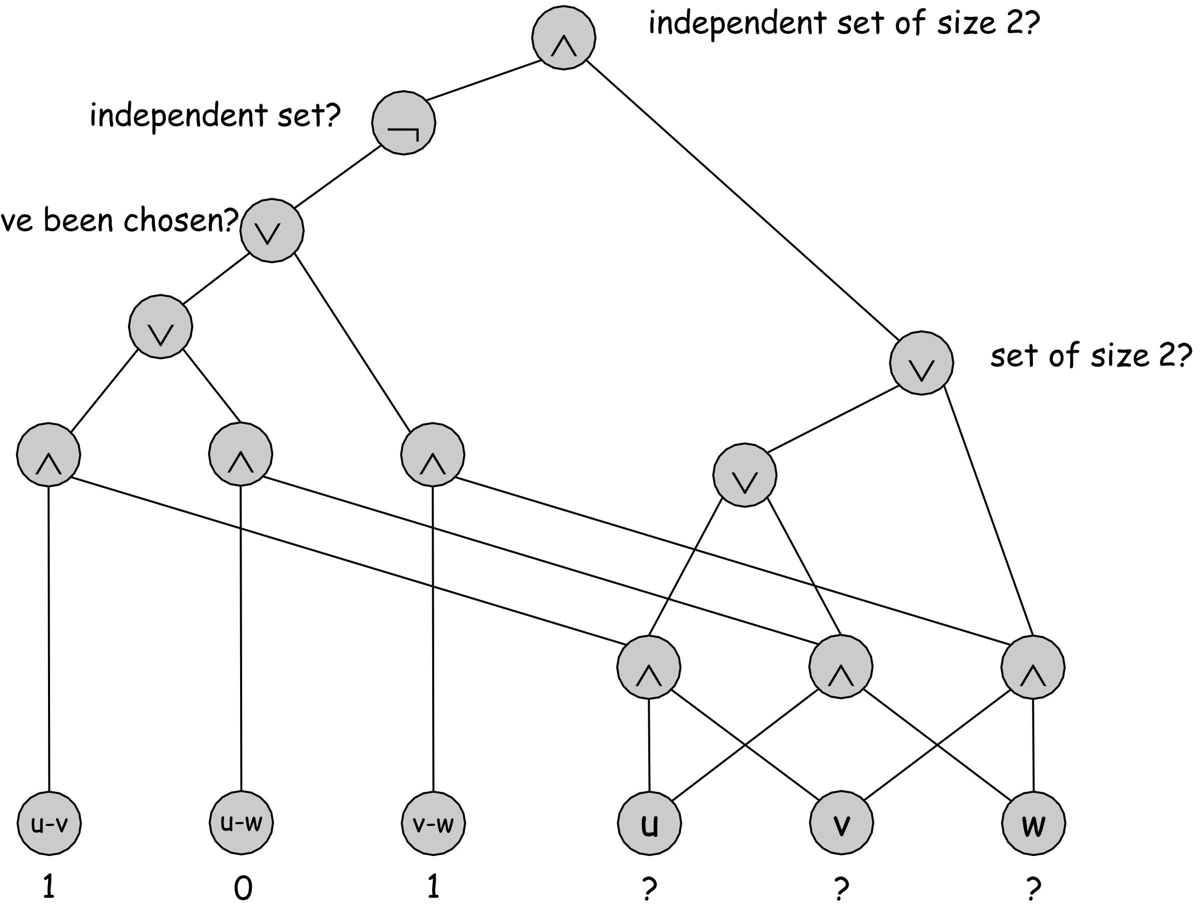
- Consider some problem X in **NP**. By Hyp. It has a **poly-time certifier C(s, t)**. To determine whether **s** ∈ X, need to know if there exists a **certificate t** of length **p(|s|)** such that **C(s, t)** = `yes`.
- View **C(s, t)** as an <u>algorithm</u> on |s| + **p(|s|) bits** (input **s**, certificate **t**) and convert it into a **poly-size circuit** **K(s_1,s_2,…,s_n ; t_1,t_2,…,t_m)** .
  - first **n=|s|** bits are **hard-coded** with input **s**
  - remaining **m**= **p(|s|)** bits represent bits of the certificate **t**
- Circuit **K** is satisfiable iff **C(s, t)** = `yes`.

Example of Reduction to Circuit SAT

Ex. Construction below creates a circuit **K** whose inputs can be set so that **K** outputs true **iff** graph **G** has an **independent set** of size **2**.



G = (V, E), m = 3



HARD CODED: graph description

t inputs (nodes in independent set)

Establishing NP-Completeness via poly-time reductions

Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

Recipe to establish NP-completeness of problem Y.

- Step 1. Show that Y is in **NP**.
- Step 2. Choose an *old* **NP-complete** problem X.
- Step 3. Prove that $X \le_p Y$.

THM. If X is an **NP-complete** problem, and Y is a problem in **NP** with the property that $X \le_P Y$ then Y is **NP-complete**, as well

Pf. Let **W** be <u>any</u> problem in **NP**. Then $W \underset{\substack{\uparrow \\ \text{by definition of} \\ \text{NP-complete}}}{\le_P} X \underset{\substack{\uparrow \\ \text{by assumption}}}{\le_P} Y$.

- By **transitivity**, $W \le_P Y$.
- Hence Y is **NP-complete**. ▪

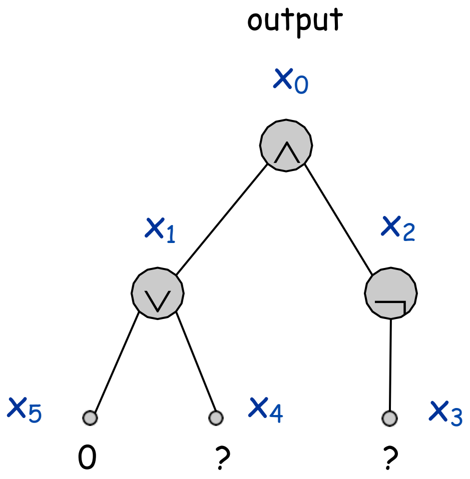
3-SAT is NP-Complete

THM. **3-SAT is NP-complete.**

Pf. Suffices to show that **CIRCUIT-SAT** $\le_P$ **3-SAT** since **3-SAT** is in **NP**.

- Let K be any circuit.
- Create a 3-SAT variable x_i for each **circuit element i**.
- Make circuit compute correct values at each node:
  - $x_2 = \neg x_3$ • add 2 clauses: $x_2 \vee x_3$, $\overline{x_2} \vee \overline{x_3}$
  - $x_1 = x_4 \vee x_5$ • add 3 clauses: $x_1 \vee \overline{x_4}$, $x_1 \vee \overline{x_5}$, $\overline{x_1} \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2$ • add 3 clauses: $\overline{x_0} \vee x_1$, $\overline{x_0} \vee x_2$, $x_0 \vee \overline{x_1} \vee \overline{x_2}$

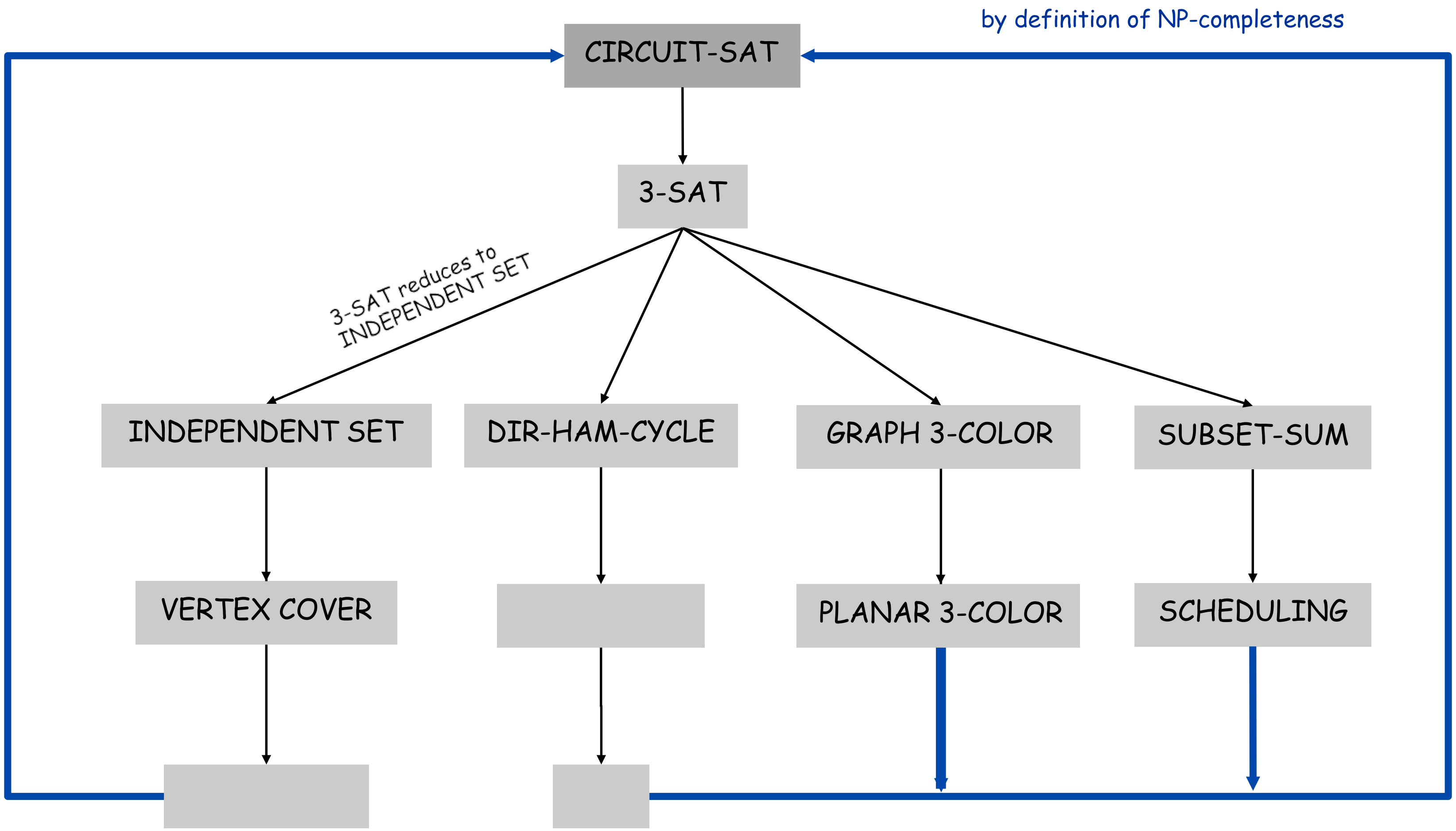
- **Hard-coded** input values and **output** value.
  - $x_5 = 0$ • add 1 clause: $\overline{x_5}$
  - $x_0 = 1$ • add 1 clause: x_0



- **Final step:** turn clauses of length < **3** into clauses of length **exactly 3**. ▪

NP-Completeness

Observation. All problems below are **NP-complete** and polynomial **reduce** to one another!



by definition of NP-completeness

CIRCUIT-SAT

3-SAT

3-SAT reduces to INDEPENDENT SET

INDEPENDENT SET

DIR-HAM-CYCLE

GRAPH 3-COLOR

SUBSET-SUM

VERTEX COVER

PLANAR 3-COLOR

SCHEDULING

Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice. Most NP problems are either known to be in P or NP-complete.

Notable exceptions. Factoring, graph isomorphism, Nash equilibrium.