# The Methodology of Abstraction in Computer Science

**Dadabhoy Institute of Higher Education**

**Karachi, Pakistan**

**April 13, 2010**

**Kamal Abdali**

# ABSTRACT

**Computer scientists have been able to solve a variety of large and difficult problems by using certain approaches and methodologies that seem specific to the computer science discipline.**

**An approach that has proved to be particularly powerful is that of abstraction. The abstraction approach to solving a problem generally consists of considering the problem at various levels or layers of complexity and detail, devising notations, formalisms, algebras, and algorithms appropriate for each layer, working separately at each layer, and obtaining the solution by synthesizing the results obtained at various layers. This talk will describe the abstraction approach, and illustrate it with the spectacularly successful computer science work in digital circuits, complexity theory, and VLSI design.**

# Contrast with other approaches

## Divide & Conquer

**Usually for recursive solutions**

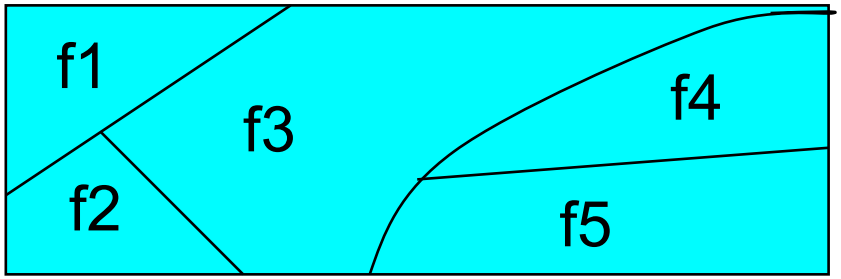
**E.g. when F(n) is expressible in terms of F(n/2):**

**F(1) = c**
**F(2n) = g(F(n),n)**
**where**
**c is a known constant,**
**g (-,-) is a known function**

## Modularization

**Where function domain can be partitioned for easier solutions for each subdomain**

**f**



# Abstraction

**Applicable when problem has different levels or layers**

**Abstraction then consists of getting rid of those details that do not contribute to solution at the level of interest.**

**Example: Building a house**

- **Architectural plan**
- **Building design specifying construction materials (concrete columns, bricks, tiles, doors, …)**
- **Electrical wiring, pipes & plumbing**
- **Paint, décor, furnishings**

# Abstraction in Switching Circuits

**Relay circuits were used in telephone exchanges and control systems. Shannon used Boolean logic to describe the functional behavior of relay networks**

- **George Boole (1815--1864): *The mathematical analysis of logic* (1847)**
- **Claude Shannon (1916--2001): *A symbolic analysis of relay and switching circuits*, Master's thesis, MIT, 1937**

# Representing switch connections as Boolean functions

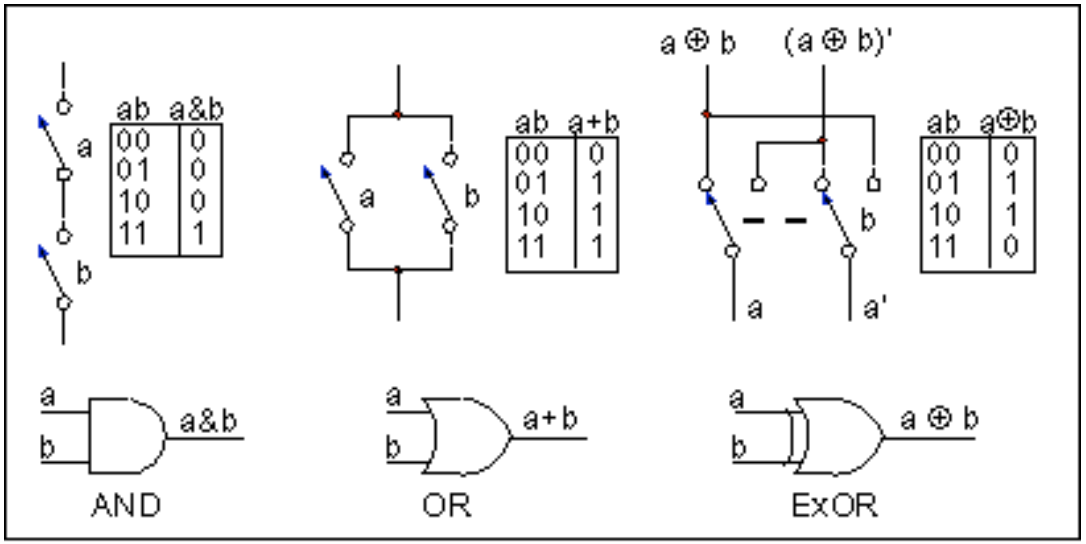



**Figure 2.** The three binary operators of Boolean Algebra with their gate and switching circuit symbols and truth tables.

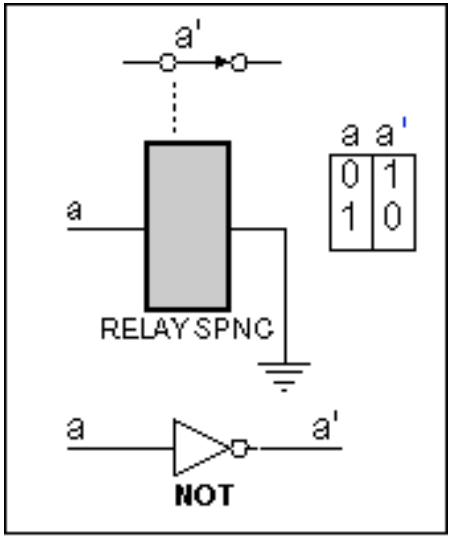


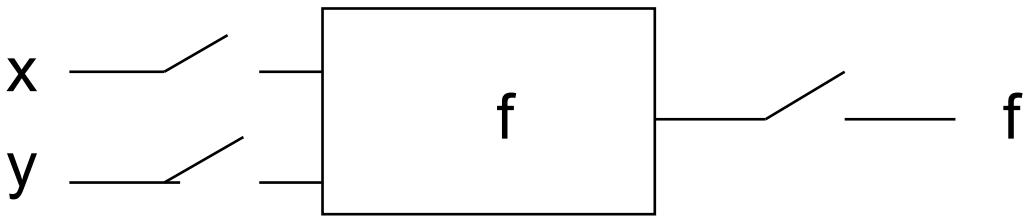

**Figure 3.** The unary operator NOT.

# Impact of Shannon's Abstraction

- Represent switching circuits as expressions of Boolean logic
- Use Boolean relations (axioms, theorems,…) to simplify expressions describing circuit behavior
- Use Boolean relations to optimize circuit design
- The formalism extends easily to other switching devices; e.g., transistors, flip-flops and similar bi-stable devices, counters, registers
- Universally used of computer architecture, fault diagnosis, fault tolerant design, …
- Extensions have been explored for non-Boolean logics, e.g., multiple-valued, probabilistic and fuzzy logics

# Example

| x | y | f |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 0 | d |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




**Choose d = 0. Then f = x. The y input is not used!**

# Abstraction in VLSI Design

- Architecture Design

  Specify system at the behavioral level.

- Logic Design:

  Specify in terms of Boolean gates (AND, OR, NOT, NAND, NOR, EXCLUSIVE OR, …)

- Circuit Design

  Account for the electrical properties of constituent devices.

- Layout Design

  Build the mask layers

# Stick Diagrams

- A stick diagram is a skeleton of a system.
- Does show all components/vias, relative placement.
- Does not show exact placement, transistor sizes, wire lengths, wire widths, tub boundaries.

# Stick Diagram Lines

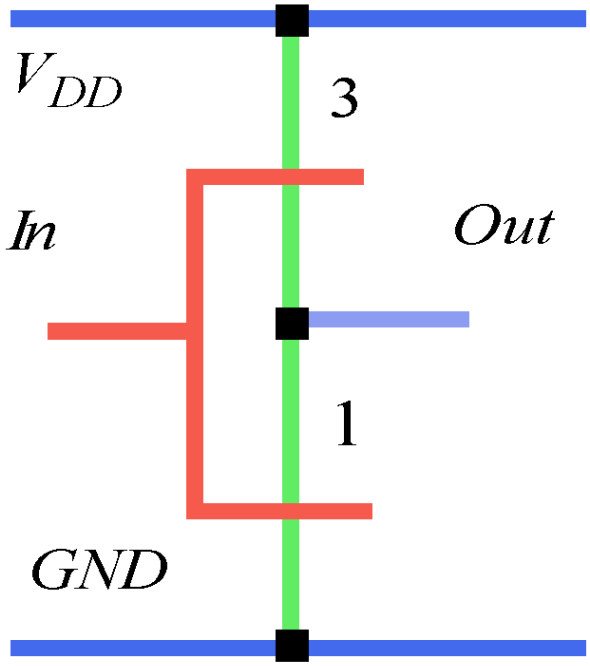
**Stick diagram lines represent chip layers**

- **Metal**

- **Polysilicon**

- **Diffusion (dope material, i.e., n or p). Usually one of n or p used, not both, so same colored line is OK**

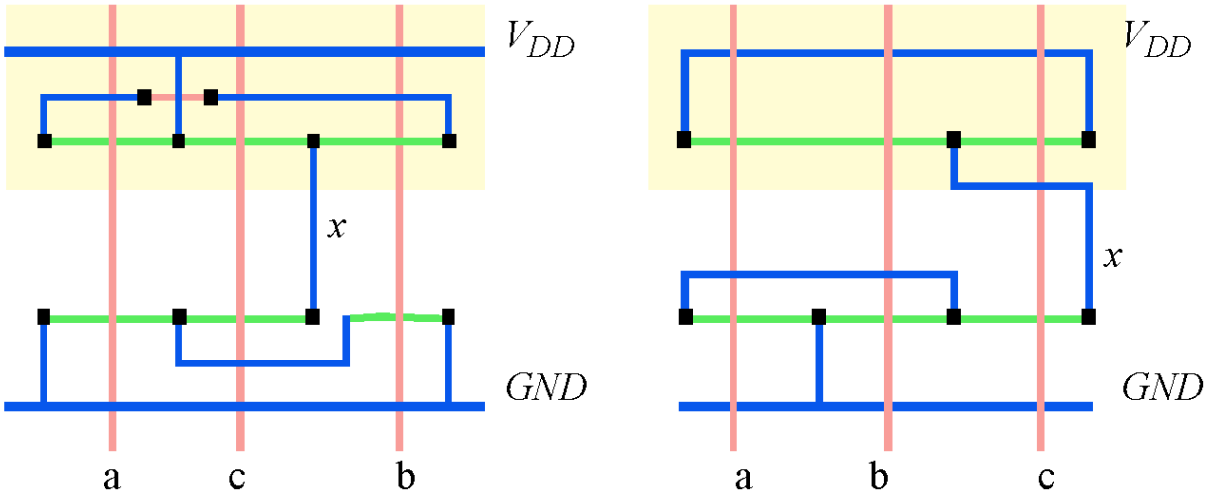
# Symbolic Layout




**Stick diagram of inverter**

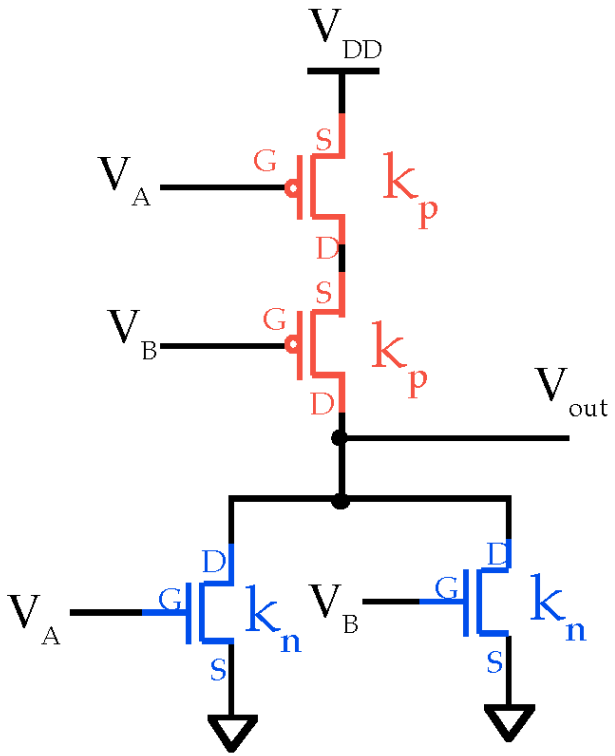
- Dimensionless layout entities
- Only topology is important
- Final layout generated by "compaction" program

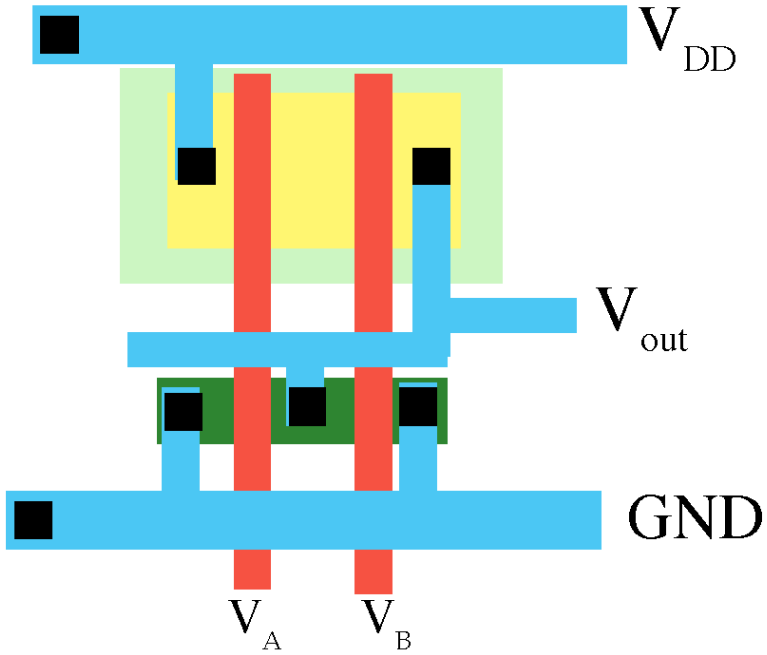
# Two Versions of (a+b).c



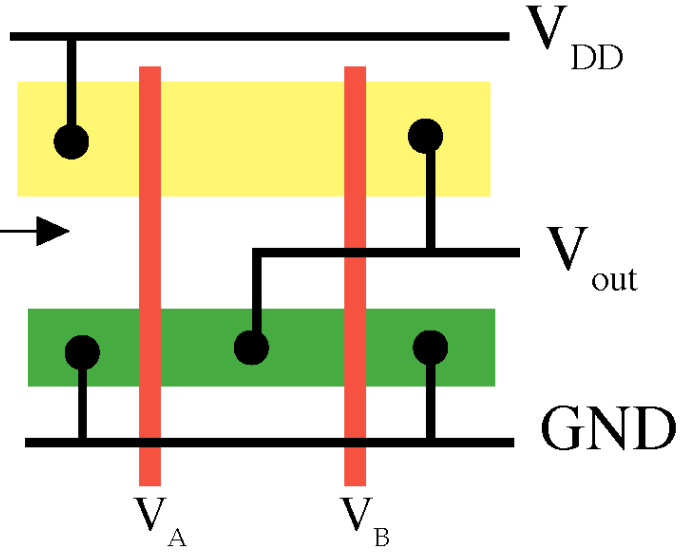

(a) Input order {*a c b*}

(b) Input order {*a b c*}




STICK DIAGRAM →




COLOR LEGEND

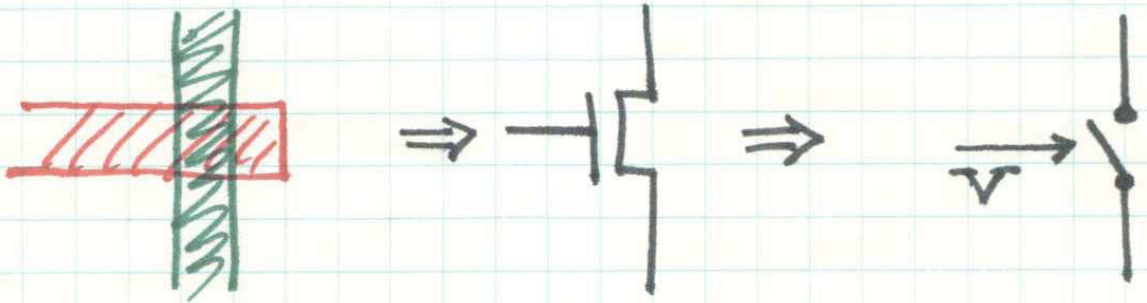
- n-Well
- p-Well
- $n^+$
- Polysilicon
- $p^+$
- Gate Oxide
- Field Oxide
- Metal 1
- Metal 2
- Metal 3
- Contact/via

I'd like to develop the basic ideas of how systems are integrated in nMOS technology.

Visualize a chip as being a sort of 3-level printed circuit board. 3-levels of conducting material are sandwiched between insulating material. SHOW VG Sequence. Name

By photolithography we can pattern the conducting material & make contact cuts thru the insulating material to form "WIRES" on the various levels. SHOW V6Seq Color codes/Level Names

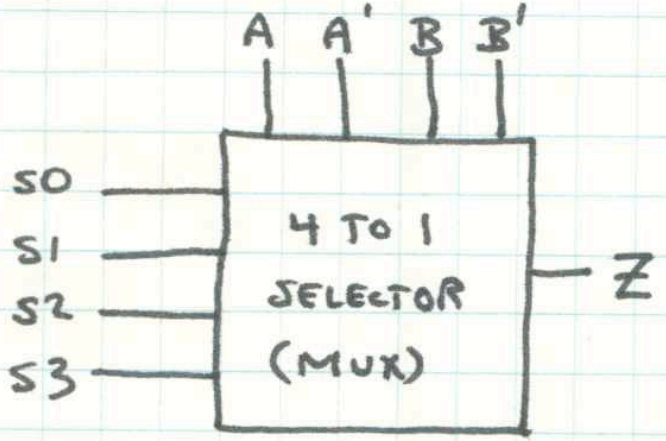
Wires on levels may cross with no functional effect, except, where RED crosses Green a transistor is created, which has the properties of a simple switch:




Slides 15-19 from Lynn Conway's VLSI Design Course, MIT, 1978

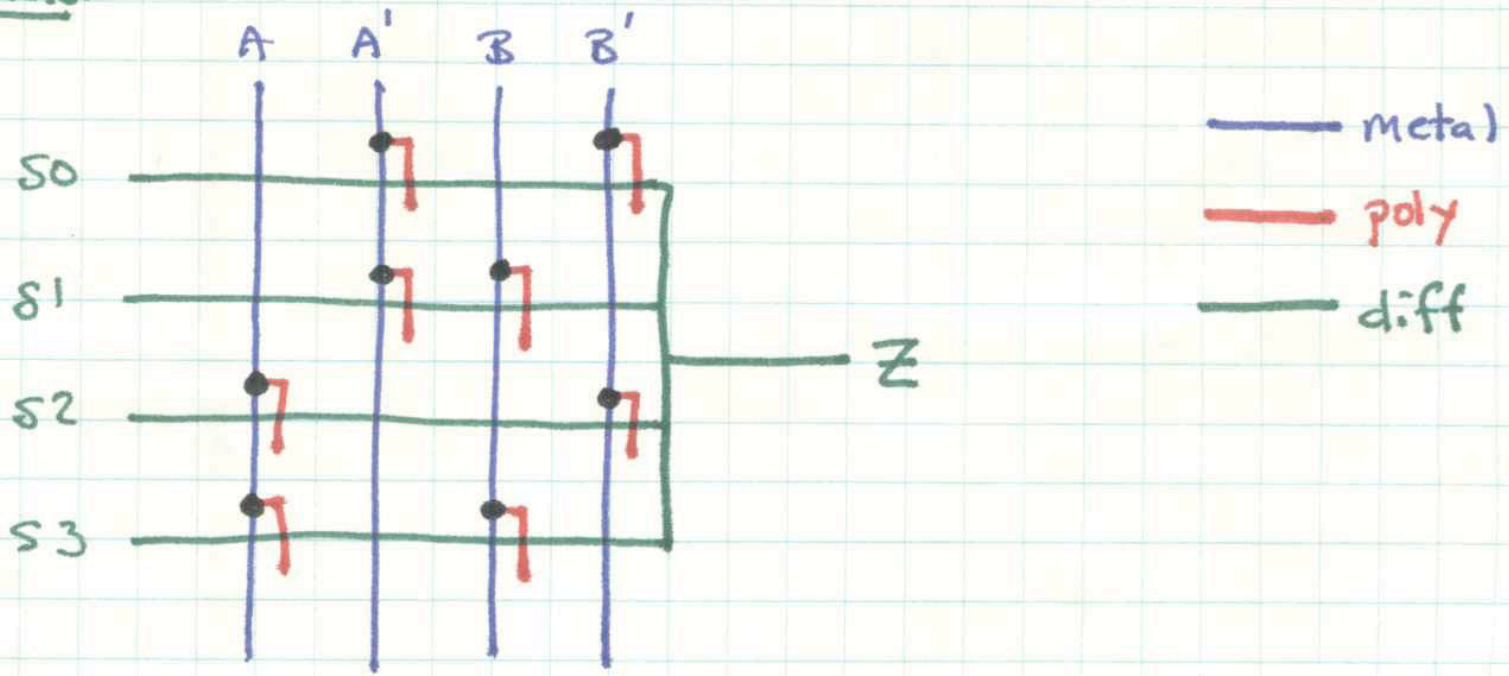
http://ai.eecs.umich.edu/people/conway/VLSI/InstGuide/InstGuide.pdf

**EXAMPLE:** CONST. STICK DIAG OF MOS INTEGRATED STRUCTURE TO IMPLEMENT A 4 TO 1 SELECTOR:

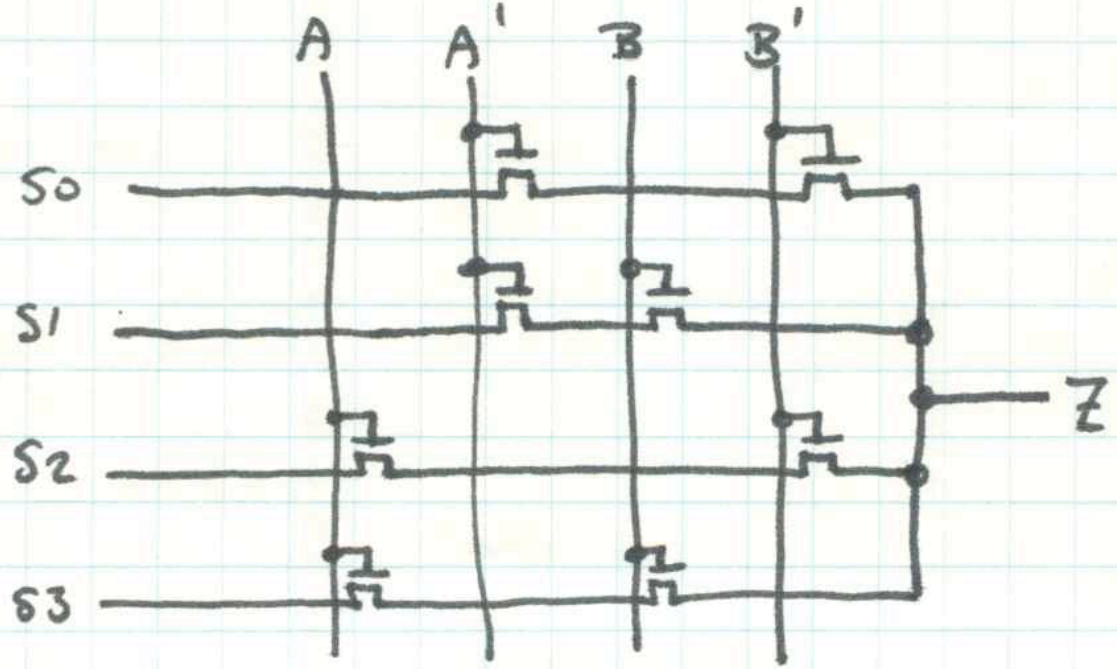



| A | B | Z |
|---|---|---|
| 0 | 0 | S0 |
| 0 | 1 | S1 |
| 1 | 0 | S2 |
| 1 | 1 | S3 |

**SOLUTION:**



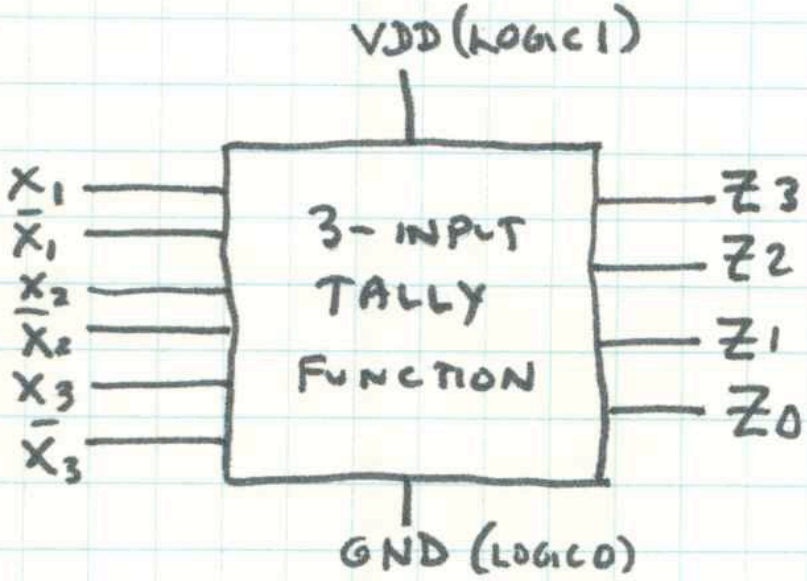
## CORRES. TRANSISTOR DIAGRAM: CIRCUIT




Interestingly, we will often find it easier & more optimal to design in this way (reminiscent of early relay switching logic) rather than use the formal methods of design synthesis using logic gates given by switching thy

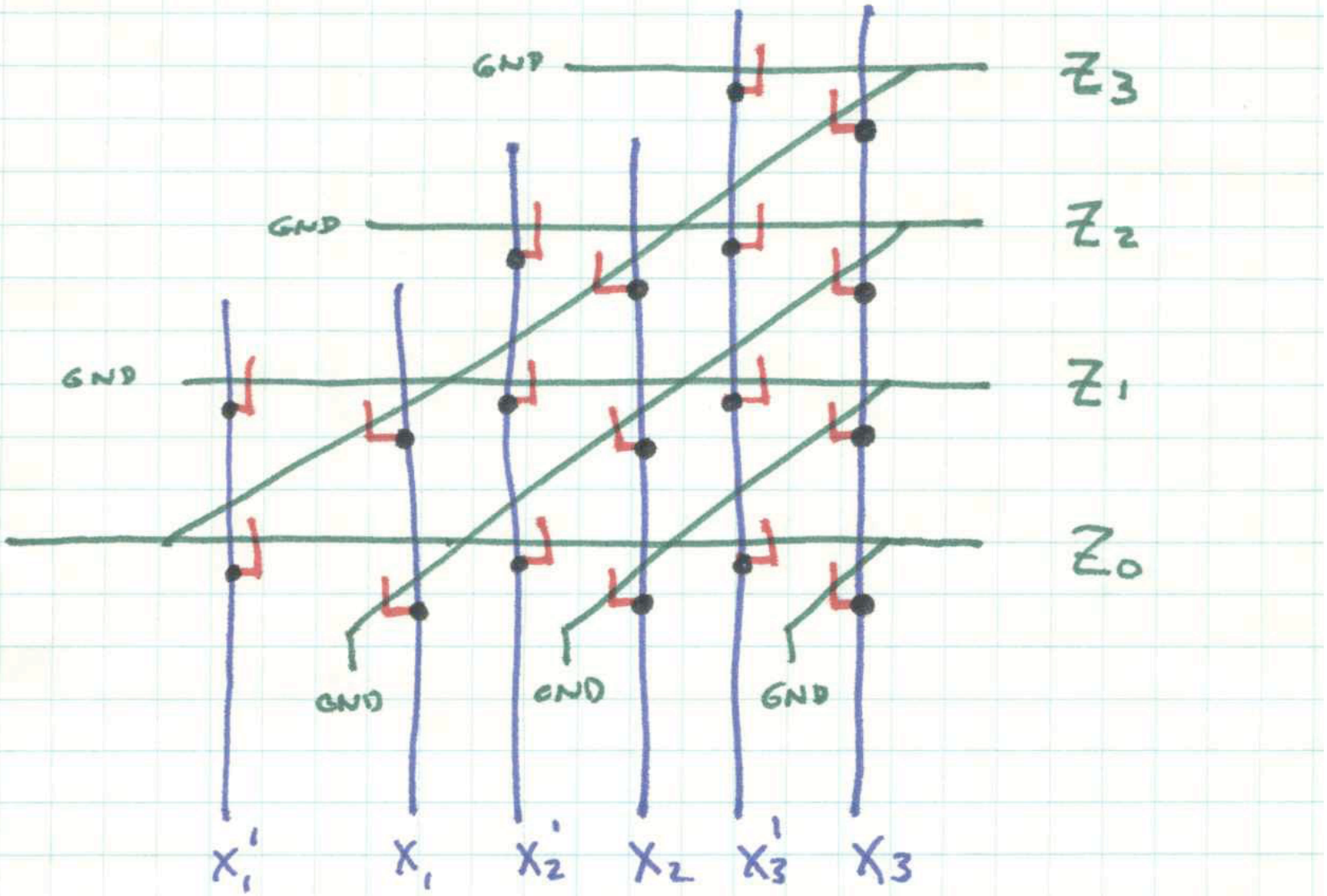
EXAMPLE: CONST. STICK DIAGRAM OF MOS INTEGRATED STRUCTURE TO IMPLEMENT A TALLY FUNCTION OF 3 INPUTS:

N INPUT VARIABLES, N+1 OUTPUTS.
IF M INPUT ARE EQ 1, OUTPUT # M = 1




| $X_1$ | $X_2$ | $X_3$ | $Z_0$ | $Z_1$ | $Z_2$ | $Z_3$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

# A SOLUTION:



# Abstraction in Algorithm Complexity

- Instead of focusing on single problems, focus on *problem classes*
- Categorize problem classes by the complexity, as measured by the amount of resources (time, computer memory, …) their solution requires
- Create a hierarchy of problem classes based on their complexity

# Decision Problems

**Decision problem.**

- **X is a set of strings.**
- **Instance: string s.**
- **Algorithm A solves problem X: A(s) = `yes` iff s ∈ X.**

**Polynomial time. Algorithm A runs in poly-time if for every string s, A(s) terminates in at most p(|s|) "steps", where p(·) is some polynomial.**

↑
length of s

**PRIMES: X = { 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, …. }**

**Algorithm. [Agrawal-Kayal-Saxena, 2002] $p(|s|) = |s|^8$.**

**ACKNOWLEDGMENT: Slides 21 – 47 adapted from**

***Slides By Kevin Wayne © 2005 Pearson-Addison Wesley***

# Definition of P

**P = Decision problems for which there is a poly-time algorithm.**

| Problem | Description | Algorithm | Yes | No |
|---|---|---|---|---|
| MULTIPLE | Is x a multiple of y? | Grade school division | `51, 17` | `51, 16` |
| RELPRIME | Are x and y relatively prime? | Euclid (300 BCE) | `34, 39` | `34, 51` |
| PRIMES | Is x prime? | AKS (2002) | `53` | `51` |
| EDIT-DISTANCE | Is the edit distance between x and y less than 5? | Dynamic programming | `neither neither` | `acgggt ttttta` |
| LSOLVE | Is there a vector x that satisfies Ax =b? | Gauss-Edmonds elimination | $\begin{bmatrix} 0 & 1 & 1 \\ 2 & 4 & -2 \\ 0 & 3 & 15 \end{bmatrix}, \begin{bmatrix} 4 \\ 2 \\ 36 \end{bmatrix}$ | $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ |

# NP

**Certification algorithm intuition.**

- **Certifier views things from "managerial" viewpoint.**
- **Certifier doesn't determine whether $s \in X$ on its own; rather, it checks a proposed proof t that $s \in X$.**

**Def.** **Algorithm C(s, t) is a certifier for problem X if for every string s, $s \in X$ iff there exists a string t such that C(s, t) = `yes`.**

"certificate" or "witness"

**NP.** **Decision problems for which there exists a poly-time certifier.**

C(s, t) is a poly-time algorithm and
$|t| \leq p(|s|)$ for some polynomial $p(\cdot)$.

**Remark.** **NP stands for nondeterministic polynomial-time.**

# Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer s, is s composite?

**Certificate.** A nontrivial factor t of s. Note that such a certificate exists iff s is composite. Moreover |t| ≤ |s|.

**Certifier.**

```
boolean C(s, t) {
   if (t ≤ 1 or t ≥ s)
      return FALSE
   else if (s is a multiple of t)
      return TRUE
   else
      return FALSE
}
```

**Instance.** s = 437,669. ⟵ 437,669 = 541 × 809

**Certificate.** t = 541 or 809.

**Conclusion.** COMPOSITES is in NP.

# Certifiers and Certificates: 3-Satisfiability

**SAT.** **Given a CNF formula $\Phi$, is there a satisfying assignment?**

**Certificate.** **An assignment of truth values to the n boolean variables.**

**Certifier.** **Check that each clause in $\Phi$ has at least one true literal.**

**Ex.**

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

instance s

$$x_1 = 1,\ x_2 = 1,\ x_3 = 0,\ x_4 = 1$$

certificate t

**Conclusion.** **SAT is in NP.**

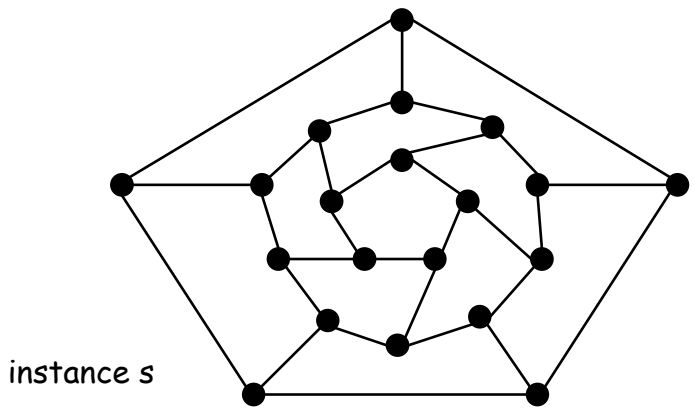
# Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** **Given an undirected graph $G = (V, E)$, does there exist a simple cycle $C$ that visits every node?**

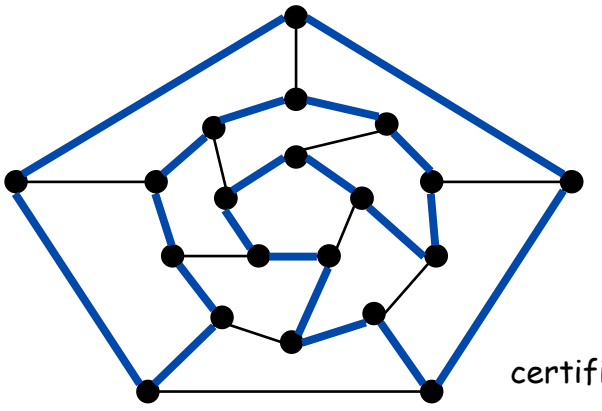
**Certificate.** **A permutation of the $n$ nodes.**

**Certifier.** **Check that the permutation contains each node in $V$ exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.**

**Conclusion.** **HAM-CYCLE is in NP.**



instance s



certificate t

# 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.** $P \subseteq NP$.

**Proof.** 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.** $NP \subseteq EXP$.

**Proof.** 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` for any 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?**


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.

# NP-Completeness

**Def.** Problem X **polynomial 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 **polynomial transforms** (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a `yes` instance of X iff y is a `yes` instance of Y.

we require |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 previous reductions were of this form.

**Open question.** Are these two concepts the same?

we abuse notation $\leq_p$ and blur distinction

# NP-Complete

**NP-complete.** A problem Y in NP with the property that for every problem X in NP, $X \leq_p Y$.

**Theorem.** Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.

**Proof.** $\Leftarrow$ If P = NP then Y can be solved in poly-time since Y is in NP.

**Proof.** $\Rightarrow$ Suppose Y can be solved in poly-time.

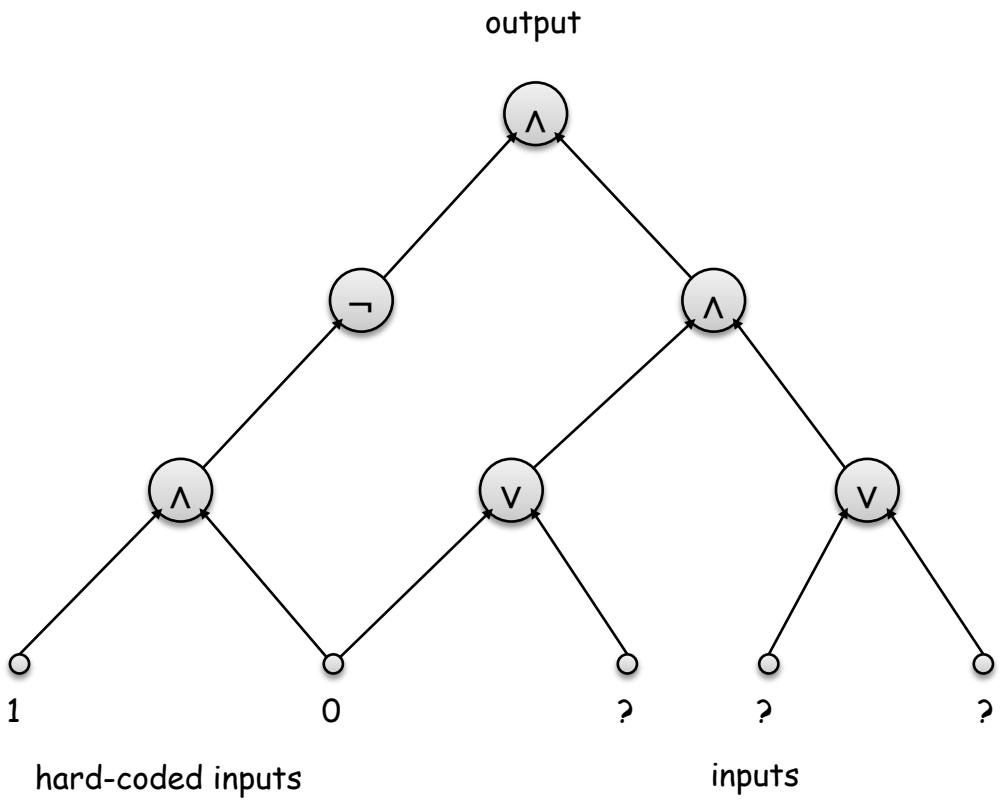
- Let X be any problem in NP. Since $X \leq_p Y$, we can solve X in poly-time. This implies NP $\subseteq$ P.
- We already know P $\subseteq$ NP. Thus P = NP. ▪

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

# Circuit Satisfiability

**CIRCUIT-SAT.** **Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?**

yes: 1 0 1



# The "First" NP-Complete Problem

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

**Proof.** **(sketch)**

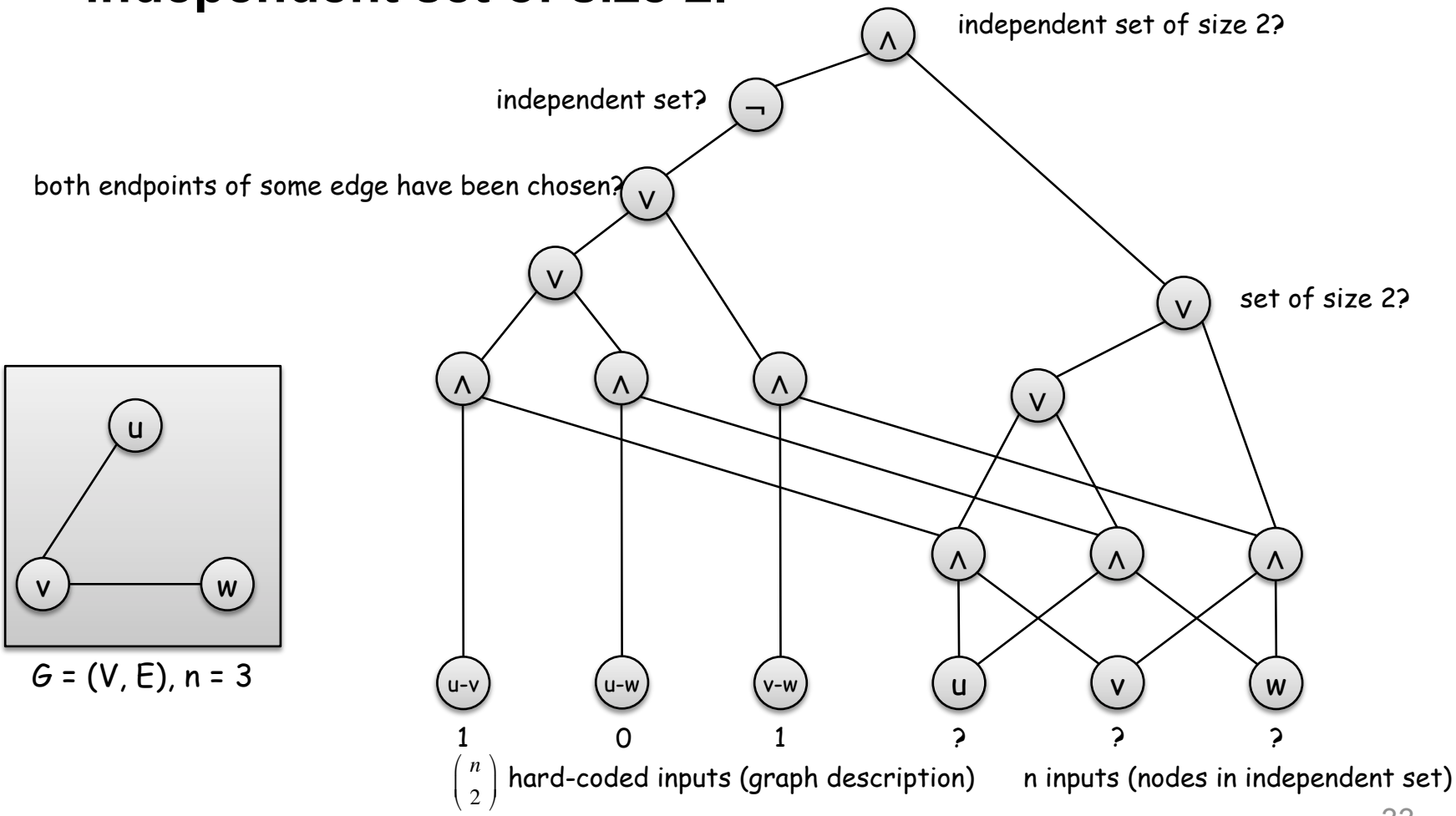
- **Any algorithm that takes a fixed number of bits** n **as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.**

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

- **Consider some problem** X **in NP. It has a poly-time certifier** C(s, t)**. To determine whether** s **is in** 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 algorithm on** |s| + p(|s|) **bits (input** s**, certificate** t**) and convert it into a poly-size circuit** K**.**
  - ♠ **first** |s| **bits are hard-coded with** s
  - ♠ **remaining** p(|s|) **bits represent bits of** t
- **Circuit** K **is satisfiable iff** C(s, t) = `yes`**.**

# Example

**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.**



# Establishing NP-Completeness

**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 NP-complete problem X.
- Step 3. Prove that $X \leq_p Y$.

**Justification.** If X is an NP-complete problem, and Y is a problem in NP with the property that $X \leq_P Y$ then Y is NP-complete.

**Proof.** Let W be any problem in NP. Then $W \leq_P X \leq_P Y$.

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

by definition of NP-complete

by assumption

# 3-SAT is NP-Complete

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

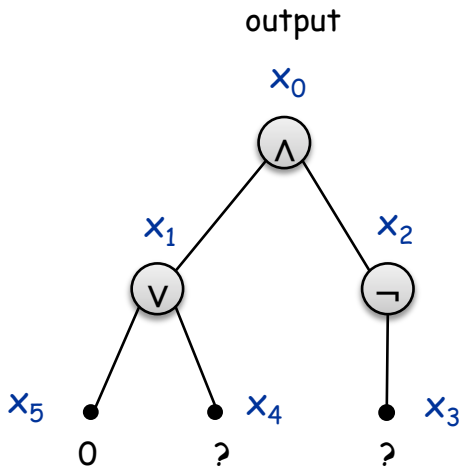
**Proof.** Suffices to show that CIRCUIT-SAT $\leq_P$ 3-SAT since 3-SAT is in NP.

- 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:
  - $x_2 = \neg\, x_3$ $\Rightarrow$ add 2 clauses: $x_2 \vee x_3$ , $\overline{x_2} \vee \overline{x_3}$
  - $x_1 = x_4 \vee x_5$ $\Rightarrow$ add 3 clauses: $x_1 \vee \overline{x_4}$ , $x_1 \vee \overline{x_5}$ , $\overline{x_1} \vee x_4 \vee x_5$
  - $x_0 = x_1 \wedge x_2$ $\Rightarrow$ 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.

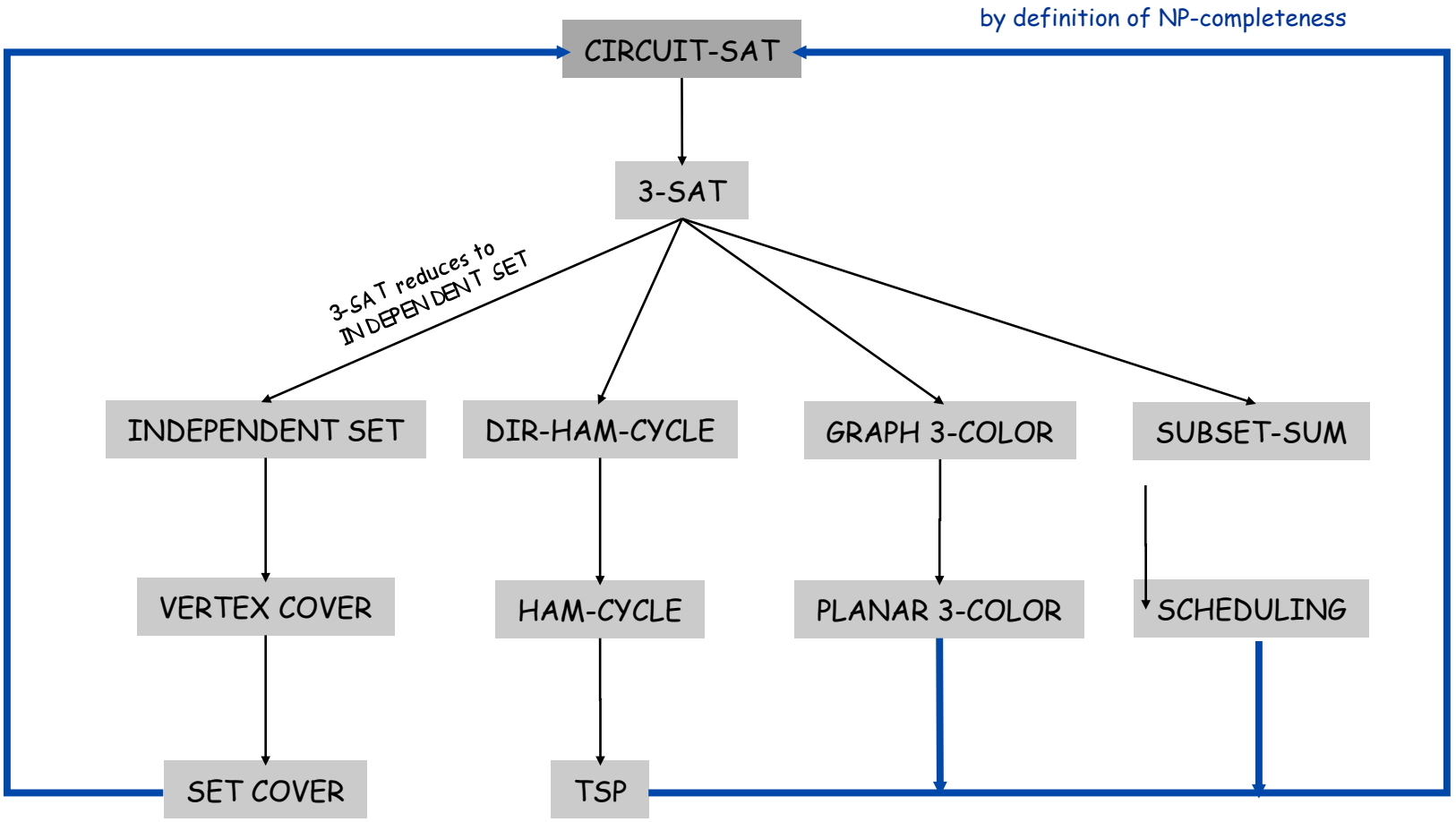
- $x_5 = 0$ $\Rightarrow$ add 1 clause: $\overline{x_5}$
- $x_0 = 1$ $\Rightarrow$ add 1 clause: $x_0$

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!**



# 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.**

# Extent and Impact of NP-Completeness

**Extent of NP-completeness. [Papadimitriou 1995]**

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

**NP-completeness can guide scientific inquiry.**

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: Istrail proves 3D problem NP-complete.

# Asymmetry of NP

**Asymmetry of NP.** We only need to have short proofs of `yes` instances.

**Ex 1.** SAT vs. TAUTOLOGY.

- Can prove a CNF formula is satisfiable by giving such an assignment
- How could we prove that a formula is not satisfiable?

**Ex 2.** HAM-CYCLE vs. NO-HAM-CYCLE.

- Can prove a graph is Hamiltonian by giving such a Hamiltonian cycle.
- How could we prove that a graph is not Hamiltonian?

**Remark.** SAT is NP-complete and SAT $\equiv_P$ TAUTOLOGY, but how do we classify TAUTOLOGY?



not even known to be in NP

# NP and co-NP

**NP.** Decision problems for which there is a poly-time certifier.

**Ex.** SAT, HAM-CYCLE, COMPOSITES.

**Def.** Given a decision problem X, its complement $\overline{X}$ is the same problem with the `yes` and `no` answers reverse.

**Ex.** $\overline{X}$ = { 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, … }
X = { 2, 3, 5, 7, 11, 13, 17, 23, 29, … }

**co-NP.** Complements of decision problems in NP.

**Ex.** TAUTOLOGY, NO-HAM-CYCLE, PRIMES.

# NP = co-NP ?

**Fundamental question.** Does NP = co-NP?

- Do `yes` instances have succinct certificates iff `no` instances do?
- Consensus opinion: no.

**Theorem.** If NP ≠ co-NP, then P ≠ NP.

**Proof idea.**

- P is closed under complementation.
- If P = NP, then NP is closed under complementation.
- In other words, NP = co-NP.
- This is the contrapositive of the theorem.

# Good Characterizations

**Good characterization.** [Edmonds 1965] NP ∩ co-NP.

- If problem X is in both NP and co-NP, then:
  - ♠ for `yes` instance, there is a succinct certificate
  - ♠ for `no` instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

**Ex.** Given a bipartite graph, is there a perfect matching.

- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes S such that |N(S)| < |S|.

# Good Characterizations

**Observation.** $P \subseteq NP \cap co\text{-}NP$.

- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in P.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

**Fundamental open question.** Does $P = NP \cap co\text{-}NP$?

- Mixed opinions.
- Many examples where problem found to have a non-trivial good characterization, but only years later discovered to be in P.
  - linear programming [Khachiyan, 1979]
  - primality testing [Agrawal-Kayal-Saxena, 2002]

**Fact.** Factoring is in $NP \cap co\text{-}NP$, but not known to be in P.

if poly-time algorithm for factoring, can break RSA cryptosystem

# PRIMES is in NP ∩ co-NP

**Theorem.** PRIMES is in NP ∩ co-NP.

**Proof.** We already know that PRIMES is in co-NP, so it suffices to prove that PRIMES is in NP.

**Pratt's Theorem.** An odd integer s is prime iff there exists an integer 1 < t < s s.t.

$$t^{s-1} \equiv 1 \pmod{s}$$
$$t^{(s-1)/p} \not\equiv 1 \pmod{s}$$

for all prime divisors $p$ of $s$-1

Input. $s$ = 437,677

Certificate. t = 17, $2^2 \times 3 \times 36{,}473$

prime factorization of s-1
also need a recursive certificate
to assert that 3 and 36,473 are prime

Certifier.

- Check $s-1 = 2 \times 2 \times 3 \times 36{,}473$.
- Check $17^{s-1} = 1 \pmod{s}$.
- Check $17^{(s-1)/2} \equiv 437{,}676 \pmod{s}$.
- Check $17^{(s-1)/3} \equiv 329{,}415 \pmod{s}$.
- Check $17^{(s-1)/36{,}473} \equiv 305{,}452 \pmod{s}$.

use repeated squaring

# FACTOR is in NP ∩ co-NP

**FACTORIZE.** Given an integer x, find its prime factorization.

**FACTOR.** Given two integers x and y, does x have a nontrivial factor less than y?

**Theorem.** FACTOR $\equiv_P$ FACTORIZE.

**Theorem.** FACTOR is in NP ∩ co-NP.

**Proof.**

- Certificate: a factor p of x that is less than y.
- Disqualifier: the prime factorization of x (where each prime factor is less than y), along with a certificate that each factor is prime.

# Primality Testing and Factoring

**Already established:** PRIMES $\leq_P$ COMPOSITES $\leq_P$ FACTOR.
**Natural question:** Does FACTOR $\leq_P$ PRIMES ?
**Consensus opinion.** No.

**State-of-the-art.**

- PRIMES is in P. ← proved in 2002
- FACTOR not believed to be in P.

**RSA cryptosystem.**

- Based on dichotomy between complexity of two problems.
- To use RSA, must generate large primes efficiently.
- To break RSA, suffixes to find efficient factoring algorithm.

# A Note on Terminology: Consensus

**NP-complete.** **A problem in NP such that every problem in NP polynomial reduces to it.**

**NP-hard.**

**A decision problem such that every problem in NP reduces to it.**

not necessarily in NP

**NP-hard search problem.** **A problem such that every problem in NP reduces to it.**

not necessarily a yes/no problem