A P-Complete Problem

Theorem 32 (Ladner (1975)) CIRCUIT VALUE *is P-complete.*

- It is easy to see that CIRCUIT VALUE $\in$ P.
- For *any* $L \in$ P, we will construct a reduction R from L to CIRCUIT VALUE.
- Given any input x, $R(x)$ is a variable-free circuit such that $x \in L$ if and only if $R(x)$ evaluates to true.
- Let M decide L in time n^k.
- Let T be the computation table of M on x.

The Proof (continued)

- When $i = 0$, or $j = 0$, or $j = |\, x \,|^k - 1$, then the value of T_{ij} is known.
  - The jth symbol of x or $\sqcup$, a $\triangleright$, and a $\sqcup$, respectively.
  - Recall that three out of T's 4 borders are known.

The Proof (continued)

- Consider *other* entries T_{ij}.
- T_{ij} depends on only $T_{i-1,j-1}$, $T_{i-1,j}$, and $T_{i-1,j+1}$:[a]

$T_{i-1,j-1}$	$T_{i-1,j}$	$T_{i-1,j+1}$
	T_{ij}	

- Let Γ denote the set of all symbols that can appear on the table: $\Gamma = \Sigma \cup \{\sigma_q : \sigma \in \Sigma, q \in K\}$.
- Encode each symbol of Γ as an m-bit number, where[b]

$$m = \lceil \log_2 |\,\Gamma\,| \rceil.$$

[a]The dependency is "local."

[b]Called **state assignment** in circuit design.

The Proof (continued)

- Let the m-bit binary string $S_{ij1} S_{ij2} \cdots S_{ijm}$ encode T_{ij}.
- We may treat them interchangeably without ambiguity.
- The computation table is now a table of binary entries $S_{ij\ell}$, where

$$
\begin{aligned}
0 \leq i \leq n^k - 1, \\
0 \leq j \leq n^k - 1, \\
1 \leq \ell \leq m.
\end{aligned}
$$

The Proof (continued)

- Each bit $S_{ij\ell}$ depends on only $3m$ other bits:

$$\begin{array}{lllll} T_{i-1,j-1}: & S_{i-1,j-1,1} & S_{i-1,j-1,2} & \cdots & S_{i-1,j-1,m} \\ T_{i-1,j}: & S_{i-1,j,1} & S_{i-1,j,2} & \cdots & S_{i-1,j,m} \\ T_{i-1,j+1}: & S_{i-1,j+1,1} & S_{i-1,j+1,2} & \cdots & S_{i-1,j+1,m} \end{array}$$

- So truth values for the $3m$ bits determine $S_{ij\ell}$.

The Proof (continued)

- This means there is a boolean function F_ℓ with $3m$ inputs such that

$$
\begin{aligned}
& S_{ij\ell} \\
= \;& F_\ell(\overbrace{S_{i-1,j-1,1}, S_{i-1,j-1,2}, \ldots, S_{i-1,j-1,m}}^{T_{i-1,j-1}}, \\
& \overbrace{S_{i-1,j,1}, S_{i-1,j,2}, \ldots, S_{i-1,j,m}}^{T_{i-1,j}}, \\
& \overbrace{S_{i-1,j+1,1}, S_{i-1,j+1,2}, \ldots, S_{i-1,j+1,m}}^{T_{i-1,j+1}}),
\end{aligned}
$$

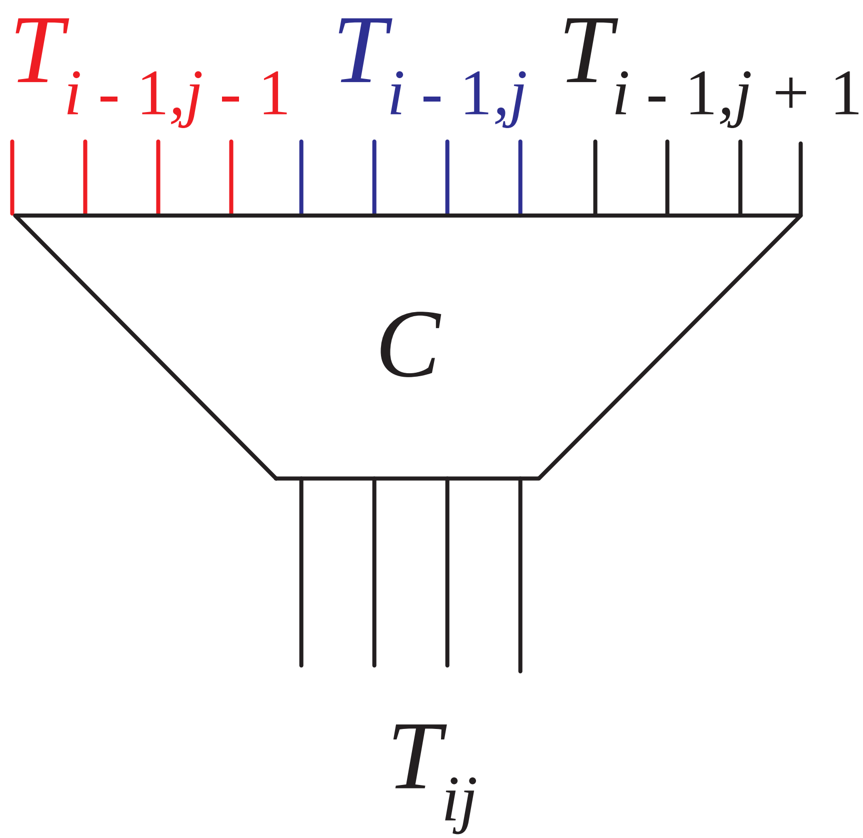
where for all $i, j > 0$ and $1 \leq \ell \leq m$.

The Proof (continued)

- These F_ℓ's depend only on M's specification, not on x, i, or j.
- Their sizes are constant.
- These boolean functions can be turned into boolean circuits (see p. 208).
- Compose these m circuits in parallel to obtain circuit C with $3m$-bit inputs and m-bit outputs.
  - Schematically, $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}) = T_{ij}$.[a]

[a] C is like an ASIC (application-specific IC) chip.

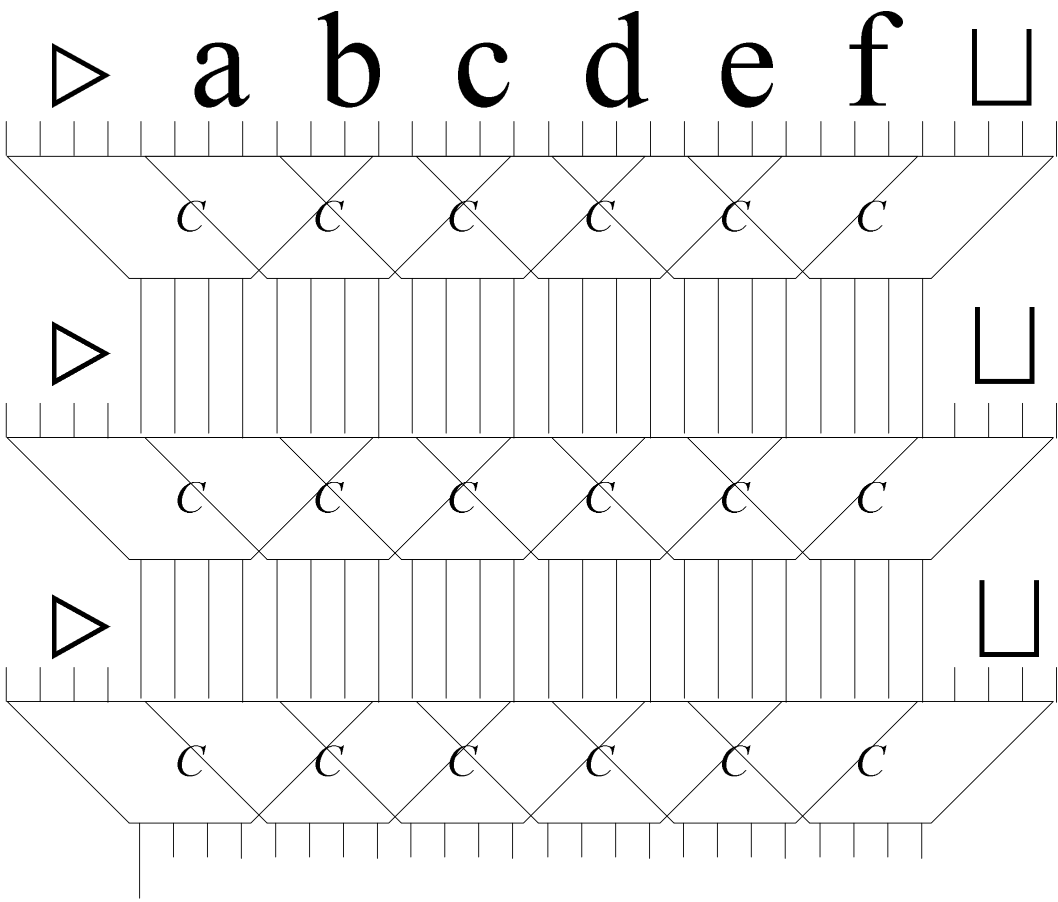
Circuit C



The Proof (concluded)

- A copy of circuit C is placed at each entry of the table.
  - Exceptions are the top row and the two extreme column borders.
- $R(x)$ consists of $(|x|^k - 1)(|x|^k - 2)$ copies of circuit C.
- Without loss of generality, assume the output "yes"/"no" appear at position $(|x|^k - 1, 1)$.
- Encode "yes" as 1 and "no" as 0.

The Computation Tableau and $R(x)$



A Corollary

The construction in the above proof yields the following, more general result.

Corollary 33 *If $L \in TIME(T(n))$, then a circuit with $O(T^2(n))$ gates can decide if $x \in L$ for $|x| = n$.*

MONOTONE CIRCUIT VALUE

- A **monotone** boolean circuit's output cannot change from true to false when one input changes from false to true.
- Monotone boolean circuits are hence less expressive than general circuits.
  - They can compute only *monotone* boolean functions.
- Monotone circuits do not contain $\neg$ gates (prove it).
- MONOTONE CIRCUIT VALUE is CIRCUIT VALUE applied to monotone circuits.

MONOTONE CIRCUIT VALUE Is P-Complete

Despite their limitations, MONOTONE CIRCUIT VALUE is as hard as CIRCUIT VALUE.

Corollary 34 MONOTONE CIRCUIT VALUE *is P-complete.*

- Given any general circuit, "move the ¬'s downwards" using de Morgan's laws[a] to yield a monotone circuit with the same output.

---

[a] How? Need to make sure no exponential blowup.

Cook's Theorem: the First NP-Complete Problem

Theorem 35 (Cook (1971)) SAT *is NP-complete.*

- SAT $\in$ NP (p. 119).
- CIRCUIT SAT reduces to SAT (p. 279).
- Now we only need to show that all languages in NP can be reduced to CIRCUIT SAT.[a]

[a] As a bonus, this also shows CIRCUIT SAT is NP-complete.

The Proof (continued)

- Let single-string NTM M decide $L \in \text{NP}$ in time n^k.
- Assume M has exactly *two* nondeterministic choices at each step: choices 0 and 1.
- For each input x, we construct circuit $R(x)$ such that $x \in L$ if and only if $R(x)$ is satisfiable.
- Equivalently, for each input x, $M(x) =$ "yes" for some computation path if and only if $R(x)$ is satisfiable.
- How to come up with a polynomial-sized $R(x)$ when there are exponentially many computation paths?

The Proof (continued)

- A straightforward proof is to construct a variable-free circuit $R_i(x)$ for the ith computation path.[a]
- Then add a small circuit to output 1 if and only if there is an $R_i(x)$ that outputs a "yes."
- Clearly, the resulting circuit outputs 1 if and only if M accepts x.
- But, it is too large because there are exponentially many computation paths.

---

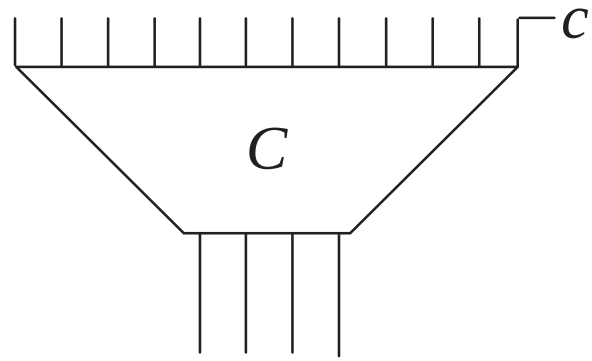
[a] The circuit for Theorem 32 (p. 300) will do.

The Proof (continued)

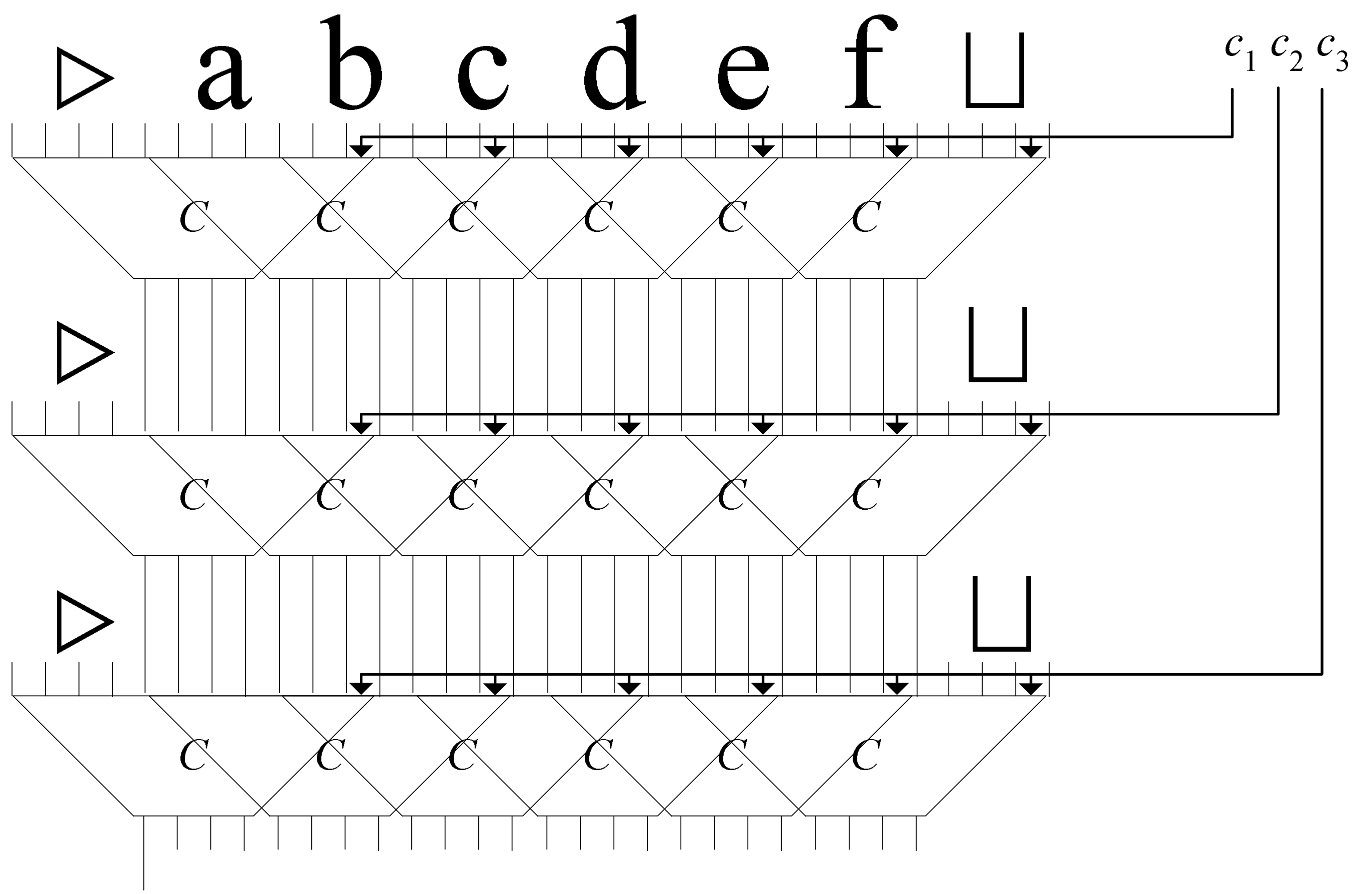
- A sequence of nondeterministic choices is a bit string

$$B = (c_1, c_2, \ldots, c_{|\,x\,|^k - 1}) \in \{0,1\}^{|\,x\,|^k - 1}.$$

- Once B is given, the computation is *deterministic*.
- Each choice of B results in a deterministic polynomial-time computation.
- Each circuit C at time i has an extra binary input c corresponding to the nondeterministic choice: $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}, c) = T_{ij}$.


c
C

The Computation Tableau for NTMs and $R(x)$

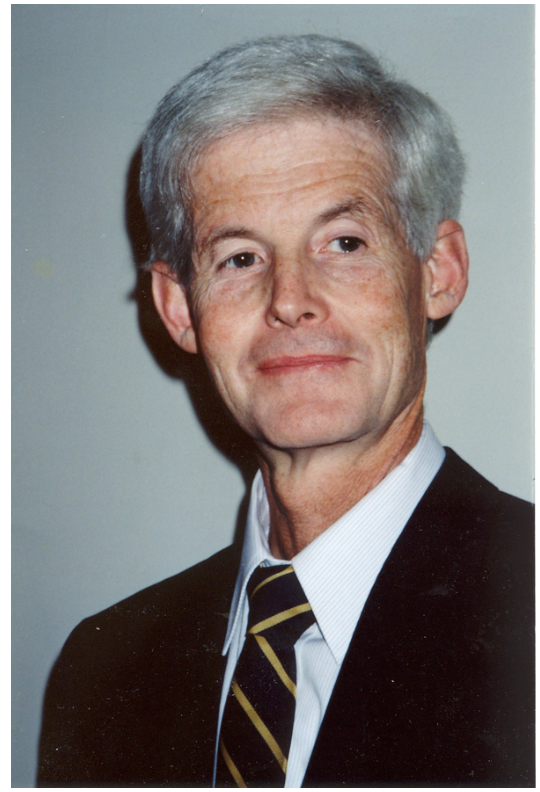


The Proof (concluded)

- Note that $c_1, c_2, \ldots, c_{|x|^k - 1}$ constitute the variables of $R(x)$.
- The overall circuit $R(x)$ (on p. 318) is satisfiable if and only if there is a truth assignment B such that the computation table accepts.
- This happens if and only if M accepts x, i.e., $x \in L$.

Stephen Arthur Cook[a] (1939–)

Richard Karp, “It is to our everlasting shame that we were unable to persuade the math department [of UC-Berkeley] to give him tenure.”



[a]Turing Award (1982). See http://conservancy.umn.edu/handle/107226 for an interview in 2002.

NP-Complete Problems

Wir müssen wissen, wir werden wissen.
(We must know, we shall know.)
— David Hilbert (1900)

I predict that scientists will one day adopt a new principle: "NP-complete problems are hard." That is, solving those problems efficiently is impossible on any device that could be built in the real world, whatever the final laws of physics turn out to be.
— Scott Aaronson (2008)

Two Notions

- Let $R \subseteq \Sigma^* \times \Sigma^*$ be a binary relation on strings.
- R is called **polynomially decidable** if

$$\{x; y : (x, y) \in R\}$$

  is in P.
- R is said to be **polynomially balanced** if $(x, y) \in R$ implies $|y| \leq |\, x \,|^k$ for some $k \geq 1$.

An Alternative Characterization of NP

Proposition 36 (Edmonds (1965)) *Let $L \subseteq \Sigma^*$ be a language. Then $L \in NP$ if and only if there is a polynomially decidable and polynomially balanced relation R such that*

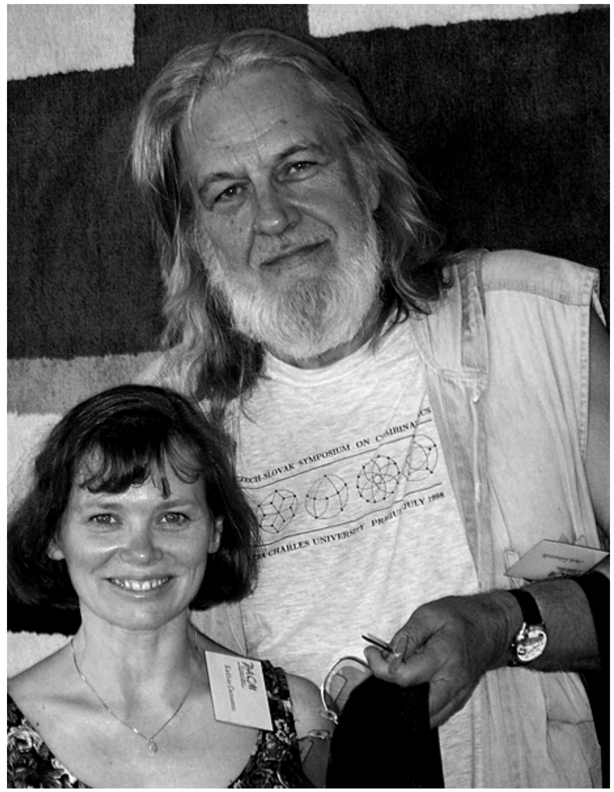
$$L = \{x : \exists y\, (x, y) \in R\}.$$

- Suppose such an R exists.
- L can be decided by this NTM:
  - On input x, the NTM guesses a y of length $\leq |\, x\,|^k$.
  - It then tests if $(x, y) \in R$ in polynomial time.
  - It returns "yes" if the test is positive.

The Proof (concluded)

- Now suppose $L \in \mathrm{NP}$.
- NTM N decides L in time $|\,x\,|^k$.
- Define R as follows: $(x, y) \in R$ if and only if y is the encoding of an accepting computation of N on input x.
- R is polynomially balanced as N is polynomially bounded.
- R is polynomially decidable because it can be efficiently verified by consulting N's transition function.
- Finally $L = \{x : (x, y) \in R \text{ for some } y\}$ because N decides L.

Jack Edmonds (1934–)



Comments

- Any "yes" instance x of an NP problem has at least one **succinct certificate** or **polynomial witness** y.
- "No" instances have none.
- Certificates are short and easy to verify.
  - An alleged satisfying truth assignment for SAT, an alleged Hamiltonian path for HAMILTONIAN PATH, etc.
- Certificates may be hard to generate,[a] but verification must be easy.
- NP is the class of *easy-to-verify* (i.e., in P) problems.

[a]Unless P equals NP.

Levin Reduction

- The reduction R in Cook's theorem (p. 313) is such that
  - Each satisfying truth assignment for circuit $R(x)$ corresponds to an accepting computation path for $M(x)$.
- It actually yields an efficient way to transform a certificate for x to a satisfying assignment for $R(x)$, and vice versa.
- A reduction with this property is called a **Levin reduction**.[a]

[a] Levin is the co-inventor of NP-completeness, in 1973.

Leonid Levin (1948–)

Leonid Levin (1998), "Mathematicians often think that historical evidence is that NP is exponential. Historical evidence is quite strongly in the other direction."

You Have an NP-Complete Problem (for Your Thesis)

- From Propositions 28 (p. 290) and Proposition 31 (p. 293), it is the least likely to be in P.
- Your options are:
  - Approximations.
  - Special cases.
  - Average performance.
  - Randomized algorithms.
  - Exponential-time algorithms that work well in practice.
  - "Heuristics" (and pray that it works *for your thesis*).

I thought NP-completeness was an interesting idea:
I didn't quite realize its potential impact.
— Stephen Cook (1998)

I was indeed surprised by Karp's work
since I did not expect so many
wonderful problems were NP-complete.
— Leonid Levin (1998)

Use of Reduction in Proving NP-Completeness

- Recall that L_1 reduces to L_2 if there is an efficient function R such that for all inputs x (p. 265),

$$x \in L_1 \text{ if and only if } R(x) \in L_2.$$

- When L_1 is known to be NP-complete and when $L_2 \in \text{NP}$, then L_2 is NP-complete.[a]
- A common mistake is to focus on solving $x \in L_1$ or solving $R(x) \in L_2$.
- The correct way is to, given a certificate for $x \in L_1$ (a satisfying truth assignment, e.g.), construct a certificate for $R(x) \in L_2$ (a Hamiltonian path, e.g.), and vice versa.

---

[a] Because NP is closed under reductions (p. 289).

3SAT

- k-SAT, where $k \in \mathbb{Z}^+$, is the special case of SAT.
- The formula is in CNF and all clauses have *exactly* k literals (repetition of literals is allowed).
- For example,

$$(x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor x_1 \lor \neg x_2) \land (x_1 \lor \neg x_2 \lor \neg x_3).$$

3SAT Is NP-Complete

- Recall Cook's Theorem (p. 313) and the reduction of CIRCUIT SAT to SAT (p. 279).
- The resulting CNF has at most 3 literals for each clause.
  - This accidentally shows that 3SAT where each clause has *at most* 3 literals is NP-complete.
- Finally, duplicate one literal once or twice to make it a 3SAT formula.
  - So

$$x_1 \lor x_2 \quad \text{becomes} \quad x_1 \lor x_2 \lor x_2.$$

The Satisfiability of Random 3SAT Expressions

- Consider a random 3SAT expressions ϕ with n variables and cn clauses.
- Each clause is chosen independently and uniformly from the set of all possible clauses.
- Intuitively, the larger the c, the less likely ϕ is satisfiable as more constraints are added.
- Indeed, there is a c_n such that for $c < c_n(1 - \epsilon)$, ϕ is satisfiable almost surely, and for $c > c_n(1 + \epsilon)$, ϕ is unsatisfiable almost surely.[a]

[a] Friedgut and Bourgain (1999). As of 2006, $3.52 < c_n < 4.596$.

Another Variant of 3SAT

Proposition 37 3SAT *is NP-complete for expressions in which each variable is restricted to appear at most three times, and each literal at most twice. (*3SAT *here requires only that each clause has* at most *3 literals.)*

The Proof (continued)

- Consider a general 3SAT expression in which x appears k times.
- Replace the first occurrence of x by x_1, the second by x_2, and so on.
  - $x_1, x_2, \ldots, x_k$ are k *new* variables.

The Proof (concluded)

- Add $(\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \cdots \land (\neg x_k \lor x_1)$ to the expression.
  - It is logically equivalent to
$$x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_k \Rightarrow x_1.$$
  - So $x_1, x_2, \ldots, x_k$ must assume an identical truth value for the whole expression to be satisfied.
- Note that each clause $\neg x_i \lor x_j$ above has only 2 literals.
- The resulting equivalent expression satisfies the conditions for x.

An Example

- Suppose we are given the following 3SAT expression

$$\cdots (\neg x \vee w \vee g) \wedge \cdots \wedge (x \vee y \vee z) \cdots .$$

- The transformed expression is

$$\cdots (\neg x_1 \vee w \vee g) \wedge \cdots \wedge (x_2 \vee y \vee z) \cdots (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_1).$$

  - Variable x_1 appears 3 times.
  - Literal x_1 appears once.
  - Literal $\neg x_1$ appears 2 times.

2SAT Is in NL $\subseteq$ P

- Let ϕ be an instance of 2SAT: Each clause has 2 literals.
- NL is a subset of P (p. 246).
- By Eq. (3) on p. 257, coNL equals NL.
- We need to show only that recognizing *unsatisfiable* 2SAT expressions is in NL.
- See the textbook for the complete proof.

Generalized 2SAT: MAX2SAT

- Consider a 2SAT expression.
- Let $K \in \mathbb{N}$.
- MAX2SAT asks whether there is a truth assignment that satisfies at least K of the clauses.
  - MAX2SAT becomes 2SAT when K equals the number of clauses.

Generalized 2SAT: MAX2SAT (concluded)

- MAX2SAT is an optimization problem.
  - With binary search, one can nail the maximum number of satisfiable clauses of the 2SAT expression.
- MAX2SAT $\in$ NP: Guess a truth assignment and verify the count.
- We now reduce 3SAT ϕ to MAX2SAT.

MAX2SAT Is NP-Complete[a]

- Consider the following 10 clauses:

$$(x) \wedge (y) \wedge (z) \wedge (w)$$
$$(\neg x \vee \neg y) \wedge (\neg y \vee \neg z) \wedge (\neg z \vee \neg x)$$
$$(x \vee \neg w) \wedge (y \vee \neg w) \wedge (z \vee \neg w)$$

- Let the 2SAT formula $r(x, y, z, w)$ represent the conjunction of these clauses.
- The clauses are symmetric with respect to x, y, and z.
- How many clauses can we satisfy?

[a]Garey, Johnson, and Stockmeyer (1976).

The Proof (continued)

All of x, y, z are true: By setting w to true, we satisfy $4 + 0 + 3 = 7$ clauses, whereas by setting w to false, we satisfy only $3 + 0 + 3 = 6$ clauses.

Two of x, y, z are true: By setting w to true, we satisfy $3 + 2 + 2 = 7$ clauses, whereas by setting w to false, we satisfy $2 + 2 + 3 = 7$ clauses.

The Proof (continued)

One of x, y, z is true: By setting w to false, we satisfy $1 + 3 + 3 = 7$ clauses, whereas by setting w to true, we satisfy only $2 + 3 + 1 = 6$ clauses.

None of x, y, z is true: By setting w to false, we satisfy $0 + 3 + 3 = 6$ clauses, whereas by setting w to true, we satisfy only $1 + 3 + 0 = 4$ clauses.

The Proof (continued)

- A truth assignment that satisfies $x \vee y \vee z$ can be *extended* to satisfy 7 of the 10 clauses of $r(x, y, z, w)$, *and no more*.
- A truth assignment that does *not* satisfy $x \vee y \vee z$ can be extended to satisfy only 6 of them, *and no more*.
- The reduction from 3SAT ϕ to MAX2SAT $R(\phi)$:
  - For each clause $C_i = (\alpha \vee \beta \vee \gamma)$ of ϕ, add **group** $r(\alpha, \beta, \gamma, w_i)$ to $R(\phi)$.
- If ϕ has m clauses, then $R(\phi)$ has $10m$ clauses.
- Finally, set $K = 7m$.

The Proof (continued)

- We now show that K clauses of $R(\phi)$ can be satisfied if and only if ϕ is satisfiable.
- Suppose $K = 7m$ clauses of $R(\phi)$ can be satisfied.
  - 7 clauses of each group $r(\alpha, \beta, \gamma, w_i)$ must be satisfied because each group can have at most 7 clauses satisfied.[a]
  - Hence each clause $C_i = (\alpha \vee \beta \vee \gamma)$ of ϕ is satisfied by the same truth assignment.
  - So ϕ is satisfied.

[a]If 70% of the world population are male and if at most 70% of each country's population are male, then each country must have exactly 70% male population.

The Proof (concluded)

- Suppose ϕ is satisfiable.
  - Let T satisfy all clauses of ϕ.
  - Each group $r(\alpha, \beta, \gamma, w_i)$ can set its w_i appropriately to have 7 clauses satisfied.
  - So $K = 7m$ clauses are satisfied.

Michael R. Garey (1945–)

David S. Johnson (1945–)

Larry Stockmeyer (1948–2004)

NAESAT

- The NAESAT (for "not-all-equal" SAT) is like 3SAT.
- But there must be a satisfying truth assignment under which no clauses have all three literals equal in truth value.
- Equivalently, there is a truth assignment such that each clause has a literal assigned true and a literal assigned false.

NAESAT Is NP-Complete[a]

- Recall the reduction of CIRCUIT SAT to SAT on p. 279ff.
- It produced a CNF ϕ in which each clause has 1, 2, or 3 literals.
- Add the same variable z to all clauses with fewer than 3 literals to make it a 3SAT formula.
- Goal: The new formula $\phi(z)$ is NAE-satisfiable if and only if the original circuit is satisfiable.

[a] Karp (1972).

The Proof (continued)

- Suppose T NAE-satisfies $\phi(z)$.
  - $\bar{T}$ takes the opposite truth value of T on every variable.
  - $\bar{T}$ also NAE-satisfies $\phi(z)$.
  - Under T or $\bar{T}$, variable z takes the value false.
  - This truth assignment $\mathcal{T}$ must satisfy all the clauses of ϕ.
    * Because z is not the reason that makes $\phi(z)$ true under $\mathcal{T}$.
  - So $\mathcal{T} \models \phi$.
  - So the original circuit is satisfiable.

The Proof (concluded)

- Suppose there is a truth assignment that satisfies the circuit.
  - Then there is a truth assignment T that satisfies every clause of ϕ.
  - Extend T by adding $T(z) = \texttt{false}$ to obtain T'.
  - T' satisfies $\phi(z)$.
  - So in no clauses are all three literals false under T'.
  - In no clauses are all three literals true under T'.
    * Need to review the detailed construction on p. 280 and p. 281.

Richard Karp[a] (1935–)

[a]Turing Award (1985).