Complete Problems and Complexity Classes

Proposition 26 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes such that $\mathcal{C}' \subseteq \mathcal{C}$. Assume $\mathcal{C}'$ is closed under reductions and L is $\mathcal{C}$-complete. Then $\mathcal{C} = \mathcal{C}'$ if and only if $L \in \mathcal{C}'$.*

- Suppose $L \in \mathcal{C}'$ first.
- Every language $A \in \mathcal{C}$ reduces to $L \in \mathcal{C}'$.
- Because $\mathcal{C}'$ is closed under reductions, $A \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- As $\mathcal{C}' \subseteq \mathcal{C}$, we conclude that $\mathcal{C} = \mathcal{C}'$.

The Proof (concluded)

- On the other hand, suppose $\mathcal{C} = \mathcal{C}'$.
- As L is $\mathcal{C}$-complete, $L \in \mathcal{C}$.
- Thus, trivially, $L \in \mathcal{C}'$.

Two Important Corollaries

Proposition 26 implies the following.

Corollary 27 *$P = NP$ if and only if an NP-complete problem in P.*

Corollary 28 *$L = P$ if and only if a P-complete problem is in L.*

Complete Problems and Complexity Classes

Proposition 29 *Let $\mathcal{C}'$ and $\mathcal{C}$ be two complexity classes closed under reductions. If L is complete for both $\mathcal{C}$ and $\mathcal{C}'$, then $\mathcal{C} = \mathcal{C}'$.*

- All languages $\mathcal{L} \in \mathcal{C}$ reduce to $L \in \mathcal{C}'$.
- Since $\mathcal{C}'$ is closed under reductions, $\mathcal{L} \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- The proof for $\mathcal{C}' \subseteq \mathcal{C}$ is symmetric.

Table of Computation

- Let $M = (K, \Sigma, \delta, s)$ be a single-string polynomial-time deterministic TM deciding L.
- Its computation on input x can be thought of as a $|\,x\,|^k \times |\,x\,|^k$ table, where $|\,x\,|^k$ is the time bound.
  - It is a sequence of configurations.
- Rows correspond to time steps 0 to $|\,x\,|^k - 1$.
- Columns are positions in the string of M.
- The (i, j)th table entry represents the contents of position j of the string *after* i steps of computation.

Some Conventions To Simplify the Table

- M halts after at most $|\,x\,|^k - 2$ steps.
  - The string length hence never exceeds $|\,x\,|^k$.
- Assume a large enough k to make it true for $|\,x\,| \geq 2$.
- Pad the table with $\sqcup$s so that each row has length $|\,x\,|^k$.
  - The computation will never reach the right end of the table for lack of time.
- If the cursor scans the jth position at time i when M is at state q and the symbol is σ, then the (i, j)th entry is a *new* symbol σ_q.

Some Conventions To Simplify the Table (continued)

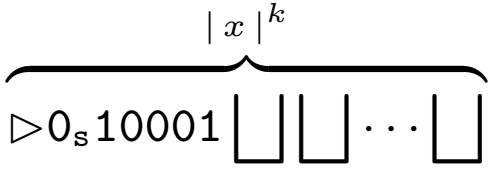
- If q is "yes" or "no," simply use "yes" or "no" instead of σ_q.
- Modify M so that the cursor starts not at $\triangleright$ but at the first symbol of the input.
- The cursor never visits the leftmost $\triangleright$ by telescoping two moves of M each time the cursor is about to move to the leftmost $\triangleright$.
- So the first symbol in every row is a $\triangleright$ and not a $\triangleright_q$.

Some Conventions To Simplify the Table (concluded)

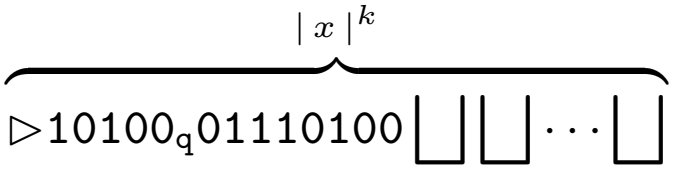
- Suppose M has halted before its time bound of $|\, x \,|^k$, so that "yes" or "no" appears at a row before the last.
- Then all subsequent rows will be identical to that row.
- M accepts x if and only if the $(|\, x \,|^k - 1, j)$th entry is "yes" for some position j.

Comments

- Each row is essentially a configuration.
- If the input $x = 010001$, then the first row is

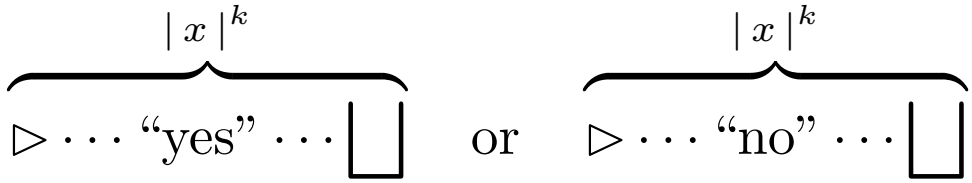


- A typical row may look like

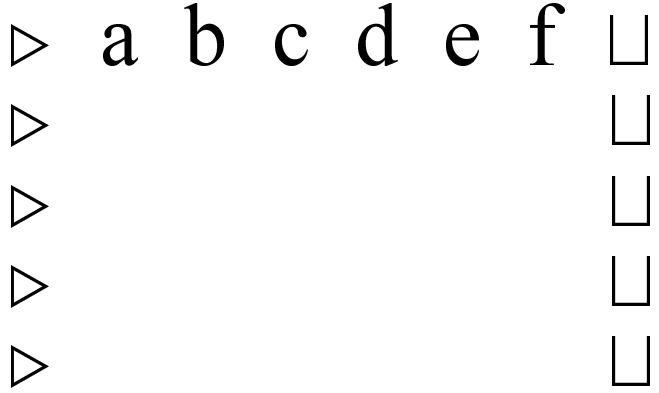


Comments (concluded)

- The last rows must look like



- Three out of the table's 4 borders are known:



A P-Complete Problem

Theorem 30 (Ladner (1975)) CIRCUIT VALUE *is P-complete.*

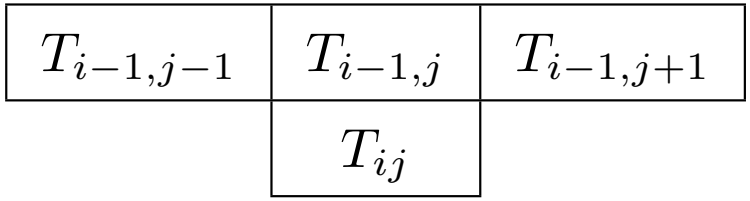
- It is easy to see that CIRCUIT VALUE $\in$ P.
- For *any* $L \in$ P, we will construct a reduction R from L to CIRCUIT VALUE.
- Given any input x, $R(x)$ is a variable-free circuit such that $x \in L$ if and only if $R(x)$ evaluates to true.
- Let M decide L in time n^k.
- Let T be the computation 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 $\bigsqcup$, a $\triangleright$, and a $\bigsqcup$, 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 $T_{i-1,j+1}$.



- 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[a]

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

[a] **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

$$0 \le i \le n^k - 1,$$
$$0 \le j \le n^k - 1,$$
$$1 \le \ell \le m.$$

The Proof (continued)

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

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

- There is a boolean function F_ℓ with $3m$ inputs such that

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

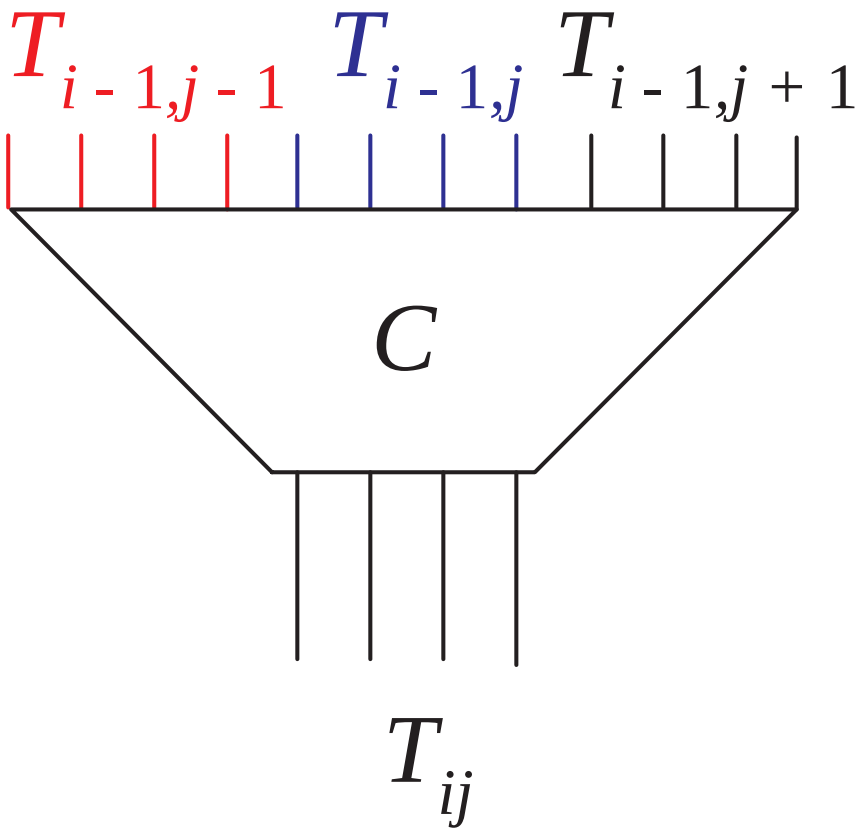
  where for all $i, j > 0$ and $1 \le \ell \le m$.

The Proof (continued)

- These F_i's depend only on M's specification, not on x.
- Their sizes are fixed.
- These boolean functions can be turned into boolean circuits.
- 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.

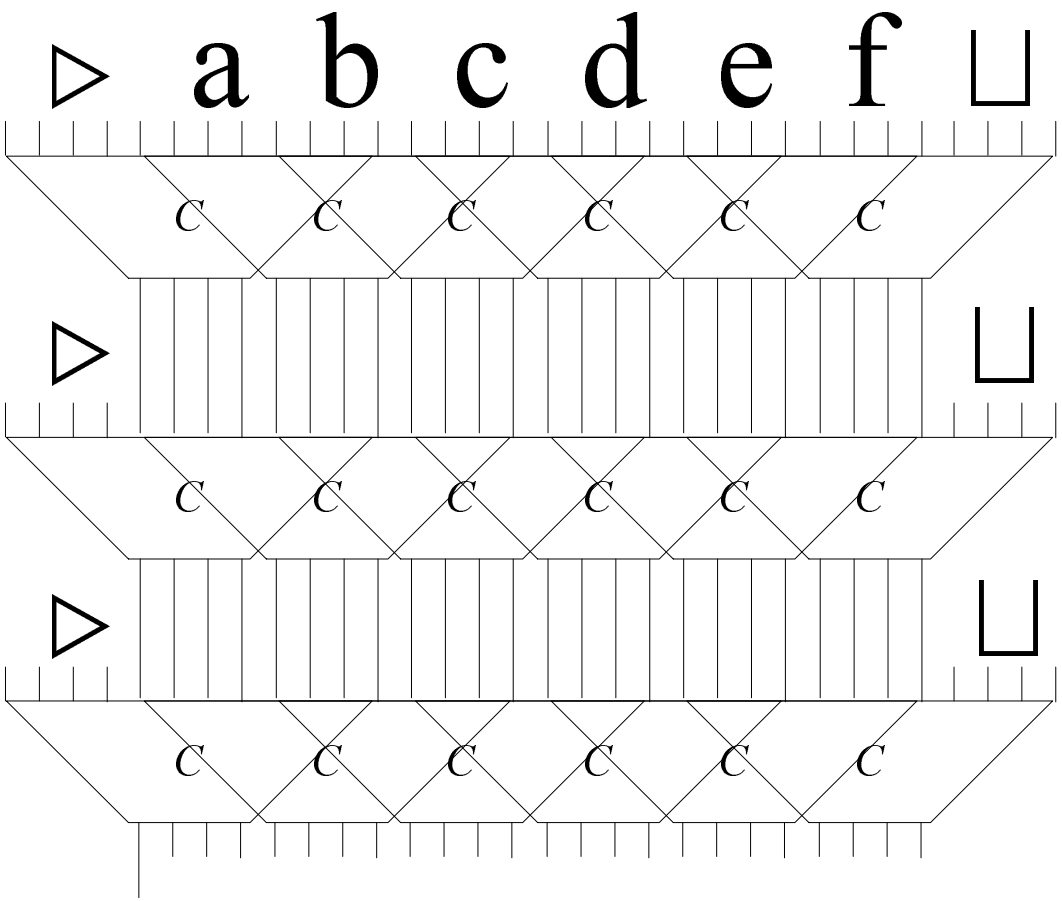
Circuit C



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 columns.
- $R(x)$ 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 31 *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 32 MONOTONE CIRCUIT VALUE *is P-complete.*

- Given any general circuit, we can "move the $\neg$'s downwards" using de Morgan's laws. (Why?)

Cook's Theorem: the First NP-Complete Problem

Theorem 33 (Cook (1971)) SAT *is NP-complete.*

- SAT $\in$ NP (p. 90).
- CIRCUIT SAT reduces to SAT (p. 231).
- Now we only need to show that all languages in NP can be reduced to CIRCUIT SAT.

The Proof (continued)

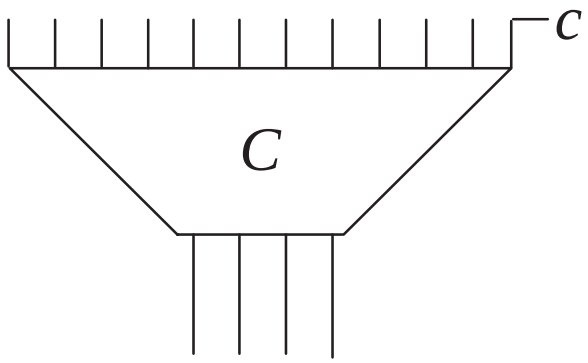
- Let single-string NTM M decide $L \in \text{NP}$ in time n^k.
- Assume M has exactly *two* nondeterministic choices at each step: choices 0 and 1.
- For each input x, we construct circuit $R(x)$ such that $x \in L$ if and only if $R(x)$ is satisfiable.
- A sequence of nondeterministic choices is a bit string

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

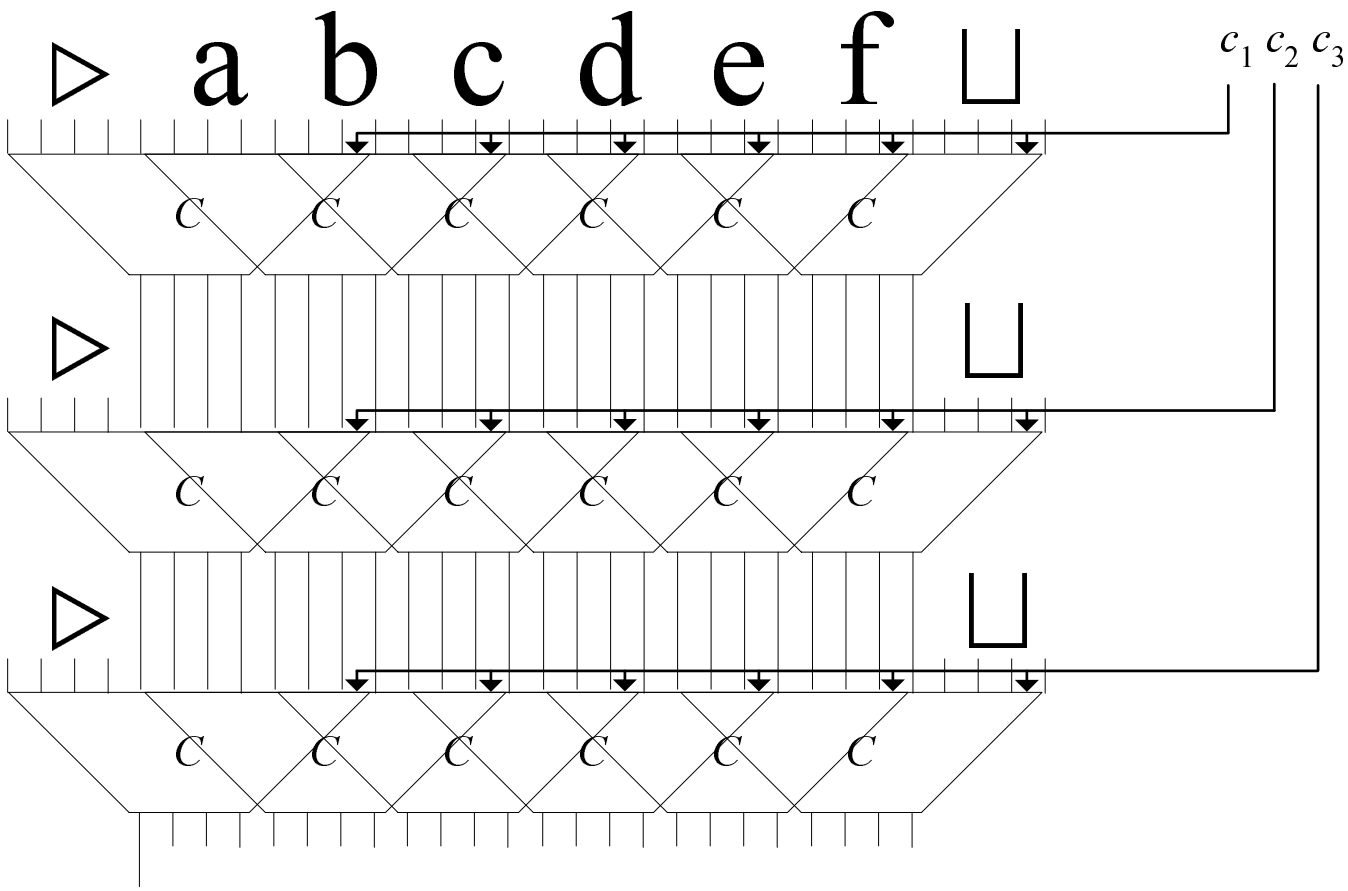
- Once B is given, the computation is *deterministic*.

The Proof (continued)

- Each choice of B results in a deterministic polynomial-time computation.
- So each choice of B results in a table like the one on p. 260.
- Each circuit C at time i has an extra binary input c corresponding to the nondeterministic choice: $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}, c) = T_{ij}$.



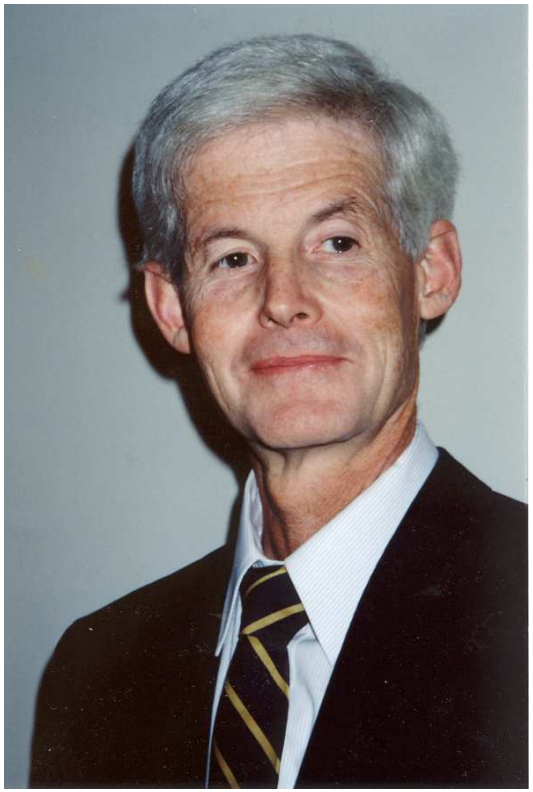
The Computation Tableau for NTMs and $R(x)$



The Proof (concluded)

- The overall circuit $R(x)$ (on p. 267) is satisfiable 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).

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 34 (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*

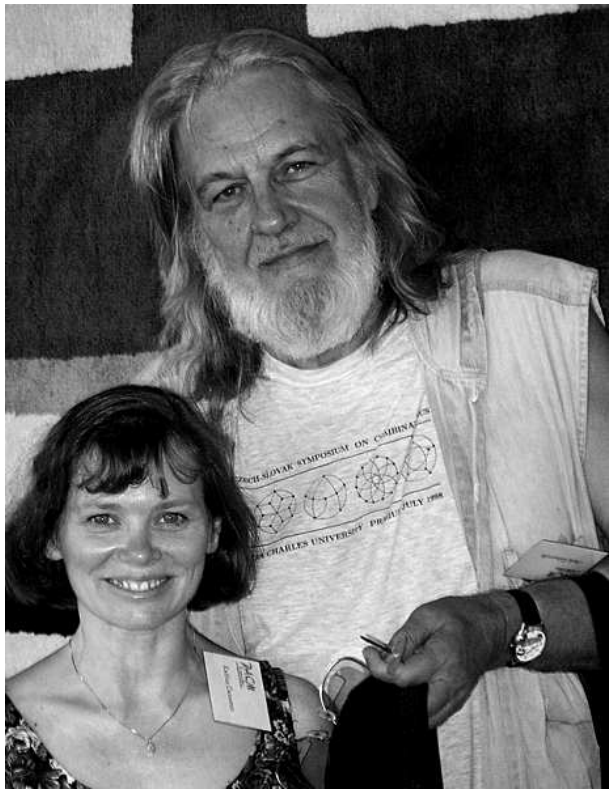
$$L = \{x : \exists y \, (x, y) \in R\}.$$

- 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 $L \in \text{NP}$.
- 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



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 and Parsimonious Reductions

- The reduction R in Cook's theorem (p. 264) 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, "Mathematicians often think that historical evidence is that NP is exponential. Historical evidence is quite strongly in the other direction."

Levin Reduction and Parsimonious Reductions (concluded)

- Furthermore, the proof gives a one-to-one and onto mapping between the set of certificates for x and the set of satisfying assignments for $R(x)$.
- So the number of satisfying truth assignments for $R(x)$ equals that of $M(x)$'s accepting computation paths.
- This kind of reduction is called **parsimonious**.
- We will loosen the requirement for parsimonious reduction: It runs in deterministic polynomial time.

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

- From Propositions 26 (p. 242) and Proposition 29 (p. 245), 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, in Shasha & Lazere (1998)

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

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,

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

3SAT Is NP-Complete

- Recall Cook's Theorem (p. 264) and the reduction of CIRCUIT SAT to SAT (p. 231).
- 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.
- Note: The overall reduction remains parsimonious.

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 35 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.)*

- 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, where $x_1, x_2, \ldots, x_k$ are k *new* variables.

The Proof (concluded)

- Add $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \cdots \wedge (\neg x_k \vee x_1)$ to the expression.
  - It is logically equivalent to

$$x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_k \Rightarrow x_1.$$

  - Note that each clause above has only 2 literals.
- The resulting equivalent expression satisfies the condition 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. 201).
- By Eq. (2) on p. 211, coNL equals NL.
- We need to show only that recognizing unsatisfiable expressions is in NL.
- See the textbook for proof.

Generalized 2SAT: MAX2SAT

- Consider a 2SAT expression.
- Let $K \in \mathbb{N}$.
- MAX2SAT is the problem of whether there is a truth assignment that satisfies at least K of the clauses.
- MAX2SAT becomes 2SAT when K equals the number of clauses.
- MAX2SAT is an optimization problem.
- MAX2SAT $\in$ NP: Guess a truth assignment and verify the count.

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)

- Any truth assignment that satisfies $x \vee y \vee z$ can be *extended* to satisfy 7 of the 10 clauses *and no more*.
- Any other truth assignment can be extended to satisfy only 6 of them.
- 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 (concluded)

- We now show that K clauses of $R(\phi)$ can be satisfied if and only if ϕ is satisfiable.
- Suppose $7m$ clauses of $R(\phi)$ can be satisfied.
  - 7 clauses must be satisfied in each group because each group can have at most 7 clauses satisfied.
  - Hence all clauses of ϕ must be satisfied.
- Suppose all clauses of ϕ are satisfied.
  - Each group can set its w_i appropriately to have 7 clauses satisfied.

Michael R. Garey (1945–)

David S. Johnson (1945–)

Larry Stockmeyer (1948–2004)