Reduction of REACHABILITY to CIRCUIT VALUE

- Note that both problems are in P.
- Given a graph $G = (V, E)$, we shall construct a *variable-free* circuit $R(G)$.
- The output of $R(G)$ is true if and only if there is a path from node 1 to node n in G.
- Idea: the Floyd-Warshall algorithm.

The Gates

- The gates are
  - g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$.
  - h_{ijk} with $1 \leq i, j, k \leq n$.
- g_{ijk}: There is a path from node i to node j without passing through a node bigger than k.
- h_{ijk}: There is a path from node i to node j passing through k but not any node bigger than k.
- Input gate $g_{ij0} =$ `true` if and only if $i = j$ or $(i, j) \in E$.

The Construction

- h_{ijk} is an AND gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$, where $k = 1, 2, \ldots, n$.
- g_{ijk} is an OR gate with predecessors $g_{i,j,k-1}$ and $h_{i,j,k}$, where $k = 1, 2, \ldots, n$.
- g_{1nn} is the output gate.
- Interestingly, $R(G)$ uses no $\neg$ gates: It is a **monotone circuit**.

Reduction of CIRCUIT SAT to SAT

- Given a circuit C, we will construct a boolean expression $R(C)$ such that $R(C)$ is satisfiable iff C is.
  - $R(C)$ will turn out to be a CNF.
  - So in a sense, $R(C)$ is a 2-level circuit.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
  - g's propagate the truth values for the CNF.
- Each gate of C will be turned into equivalent clauses.
- Recall that clauses are $\wedge$-ed together by definition.

The Clauses of $R(C)$

g **is a variable gate** x**:** Add clauses $(\neg g \vee x)$ and $(g \vee \neg x)$.

- Meaning: $g \Leftrightarrow x$.

g **is a `true` gate:** Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

g **is a `false` gate:** Add clause $(\neg g)$.

- Meaning: g must be false to make $R(C)$ true.

g **is a** $\neg$ **gate with predecessor gate** h**:** Add clauses $(\neg g \vee \neg h)$ and $(g \vee h)$.

- Meaning: $g \Leftrightarrow \neg h$.

The Clauses of $R(C)$ (concluded)

g **is a** $\lor$ **gate with predecessor gates** h **and** h'**:** Add clauses $(\neg h \lor g)$, $(\neg h' \lor g)$, and $(h \lor h' \lor \neg g)$.

- Meaning: $g \Leftrightarrow (h \lor h')$.

g **is a** $\land$ **gate with predecessor gates** h **and** h'**:** Add clauses $(\neg g \lor h)$, $(\neg g \lor h')$, and $(\neg h \lor \neg h' \lor g)$.

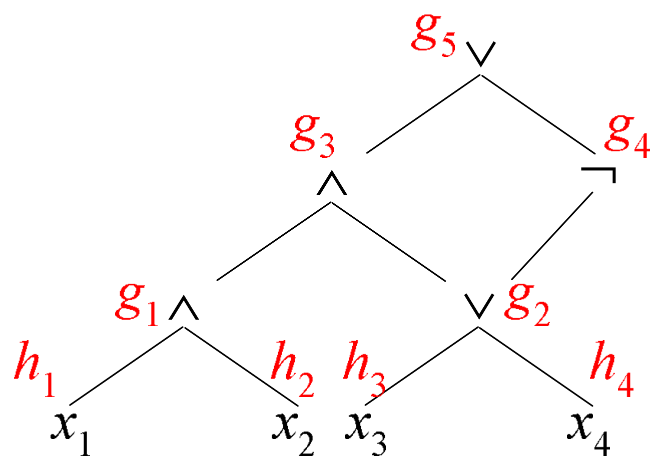
- Meaning: $g \Leftrightarrow (h \land h')$.

g **is the output gate:** Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

Note: If gate g feeds gates $h_1, h_2, \ldots$, then variable g appears in the clauses for $h_1, h_2, \ldots$ in $R(C)$.

An Example



$$
\begin{aligned}
& (h_1 \Leftrightarrow x_1) \land (h_2 \Leftrightarrow x_2) \land (h_3 \Leftrightarrow x_3) \land (h_4 \Leftrightarrow x_4) \\
\land \quad & [\, g_1 \Leftrightarrow (h_1 \land h_2) \,] \land [\, g_2 \Leftrightarrow (h_3 \lor h_4) \,] \\
\land \quad & [\, g_3 \Leftrightarrow (g_1 \land g_2) \,] \land (g_4 \Leftrightarrow \neg g_2) \land g_5.
\end{aligned}
$$

An Example (concluded)

- In general, the result is a CNF (hence with a depth of 2 even if the circuit has a higher depth).
- The CNF has size proportional to the circuit's size.
- The CNF adds new variables to the circuit's original input variables.

Composition of Reductions

Proposition 24 *If R_{12} is a reduction from L_1 to L_2 and R_{23} is a reduction from L_2 to L_3, then the composition $R_{12} \circ R_{23}$ is a reduction from L_1 to L_3.*

- Clearly $x \in L_1$ if and only if $R_{23}(R_{12}(x)) \in L_3$.
- How to compute $R_{12} \circ R_{23}$ in space $O(\log n)$, as required by the definition of reduction?

The Proof (continued)

- An obvious way is to generate $R_{12}(x)$ first and then feeding it to R_{23}.
- This takes polynomial time.[a]
  - It takes polynomial time to produce $R_{12}(x)$ of polynomial length.
  - It also takes polynomial time to produce $R_{23}(R_{12}(x))$.
- Trouble is $R_{12}(x)$ may consume up to polynomial space, much more than the logarithmic space required.

---

[a] Hence our concern below disappears had we required reductions to be in P instead of L.

The Proof (concluded)

- The trick is to let R_{23} drive the computation.
- It asks R_{12} to deliver each bit of $R_{12}(x)$ when needed.
- When R_{23} wants the ith bit, $R_{12}(x)$ will be simulated until the ith bit is available.
  - The initial $i - 1$ bits should *not* be written to the string.
- This is feasible as $R_{12}(x)$ is produced in a *write-only* manner.
  - The ith output bit of $R_{12}(x)$ is well-defined because once it is written, it will never be overwritten.

Completeness[a]

- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element?
- It is not altogether obvious that there should be a maximal element.
- Many infinite structures (such as integers and reals) do not have maximal elements.
- Hence it may surprise you that most of the complexity classes that we have seen so far have maximal elements.

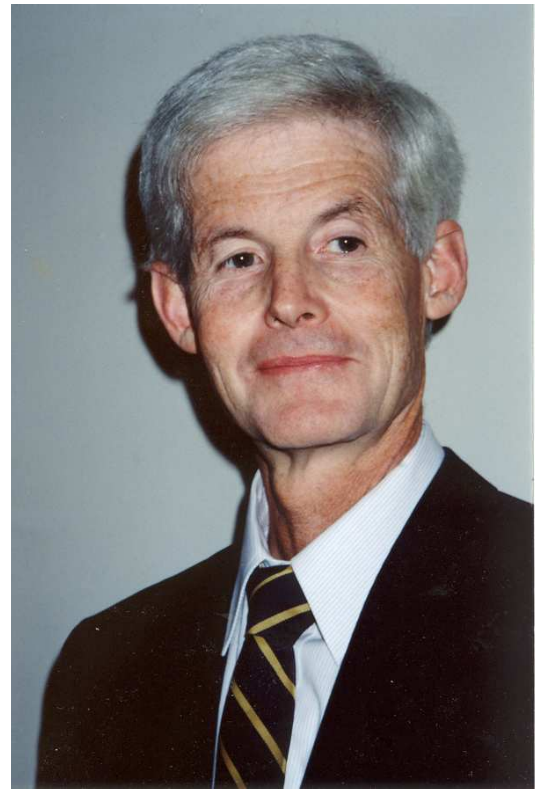
[a] Cook (1971) and Levin (1971).

Completeness (concluded)

- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if every $L' \in \mathcal{C}$ can be reduced to L.
  - Most complexity classes we have seen so far have complete problems!
- Complete problems capture the difficulty of a class because they are the hardest.

Stephen Arthur Cook (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."



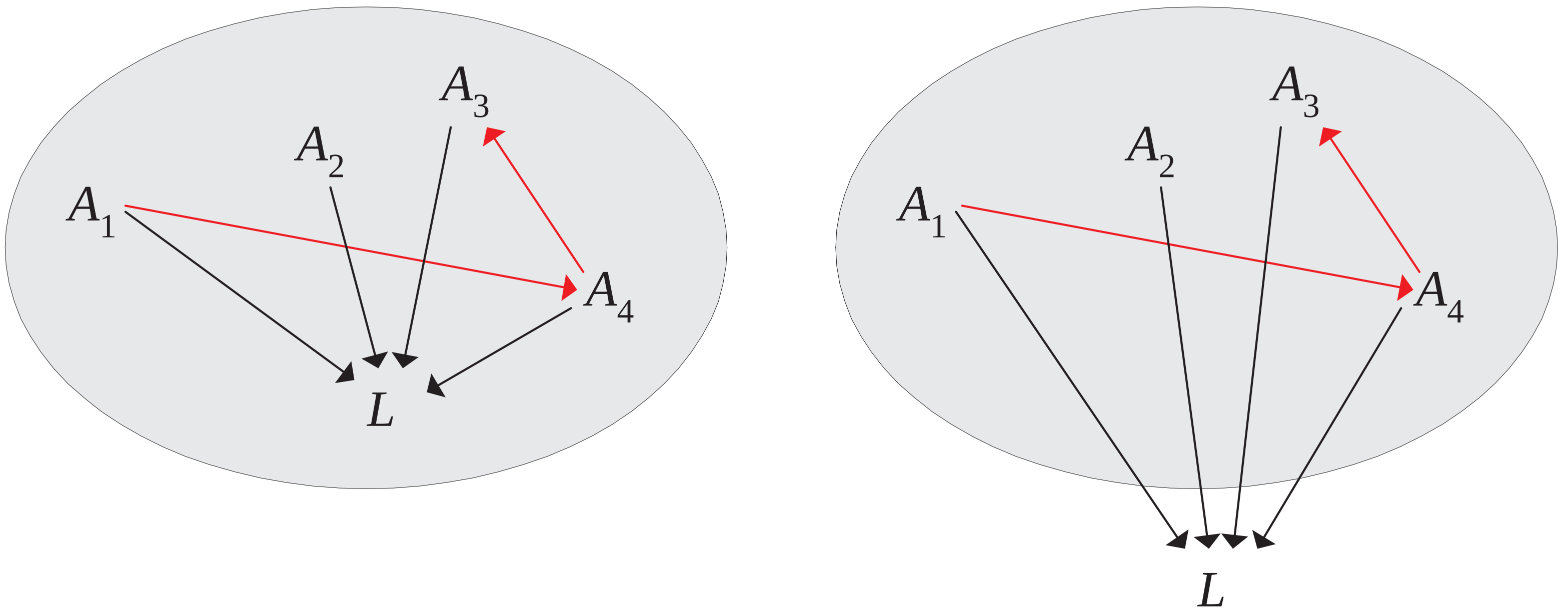
Leonid Levin (1948–)

Hardness

- Let $\mathcal{C}$ be a complexity class.
- L is **$\mathcal{C}$-hard** if every $L' \in \mathcal{C}$ can be reduced to L.
- It is not required that $L \in \mathcal{C}$.
- If L is $\mathcal{C}$-hard, then by definition, every $\mathcal{C}$-complete problem can be reduced to L.[a]

[a]Contributed by Mr. Ming-Feng Tsai (`D92922003`) on October 15, 2003.

Illustration of Completeness and Hardness



Closedness under Reduction

- A class $\mathcal{C}$ is **closed under reductions** if whenever L is reducible to L' and $L' \in \mathcal{C}$, then $L \in \mathcal{C}$.
- P, NP, coNP, L, NL, PSPACE, and EXP are all closed under reductions.

Complete Problems and Complexity Classes

Proposition 25 *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 Immediate Corollaries

Proposition 25 implies that

- P = NP if and only if an NP-complete problem in P.
- L = P if and only if a P-complete problem is in L.

Complete Problems and Complexity Classes

Proposition 26 *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)

- If 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 j.

Comments

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

$$\overbrace{\rhd \mathtt{0_s10001} \sqcup \sqcup \cdots \sqcup}^{|\,x\,|^k}$$

- A typical row may be

$$\overbrace{\rhd \mathtt{10100_q01110100} \sqcup \sqcup \cdots \sqcup}^{|\,x\,|^k}$$

Comments (concluded)

- The last rows must look like

$$\rhd \overbrace{\cdots \text{"yes"} \cdots \sqcup}^{|\,x\,|^k}$$

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

$$\begin{array}{cccccccc} \rhd & \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \sqcup \\ \rhd & & & & & & & \sqcup \\ \rhd & & & & & & & \sqcup \\ \rhd & & & & & & & \sqcup \\ \rhd & & & & & & & \sqcup \end{array}$$

A P-Complete Problem

Theorem 27 (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 four of T's 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 $T_{i-1,j+1}$.

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

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

(**state assignment** in circuit design).

The Proof (continued)

- Let 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:

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

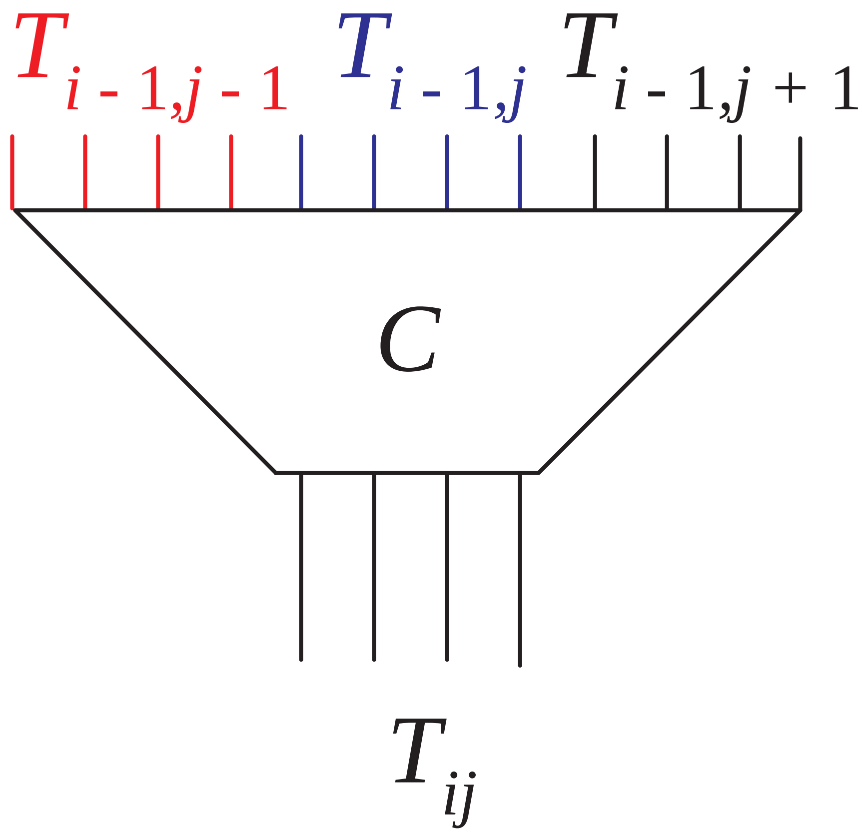
- So there are m boolean functions $F_1, F_2, \ldots, F_m$ with $3m$ inputs each such that for all $i, j > 0$ and $1 \leq \ell \leq m$,

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

The Proof (continued)

- These F_i's depend on only 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, $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}) = T_{ij}$.
  - C is like an ASIC (application-specific IC) chip.

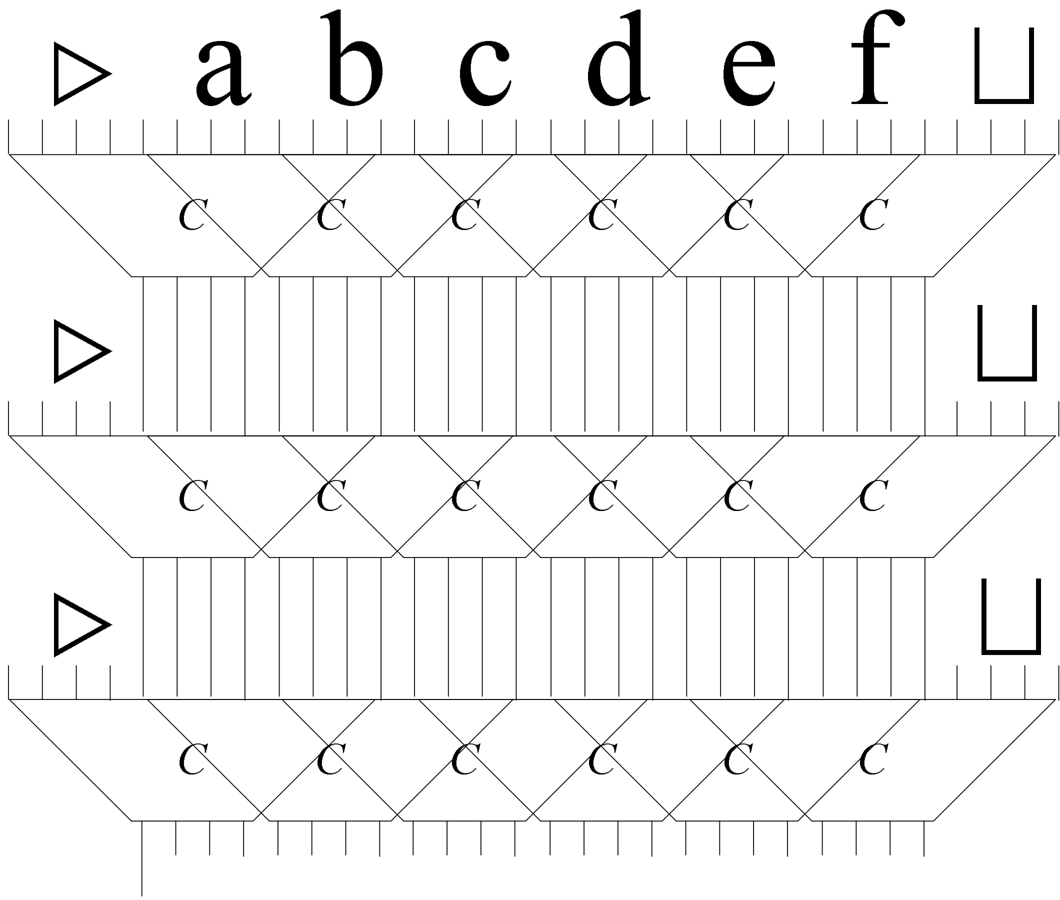
Circuit C



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" (coded as 1/0) appear at position $(|\, x\,|^k - 1, 1)$.

The Computation Tableau and $R(x)$



A Corollary

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

Corollary 28 *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.
- 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 29 MONOTONE CIRCUIT VALUE *is P-complete.*

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

Cook's Theorem: the First NP-Complete Problem

Theorem 30 (Cook (1971)) SAT *is NP-complete.*

- SAT $\in$ NP (p. 87).
- CIRCUIT SAT reduces to SAT (p. 226).
- Now we only need to show that all languages in NP can be reduced to CIRCUIT SAT.

The Proof (continued)

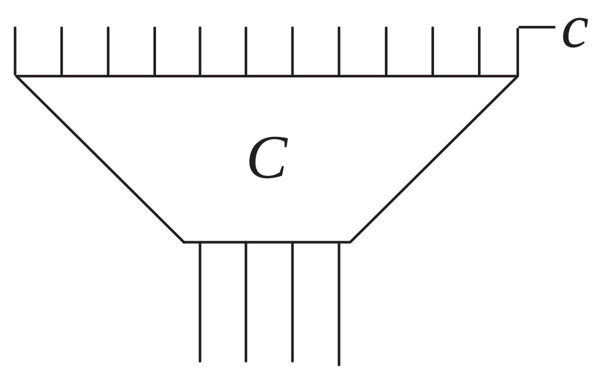
- Let single-string NTM M decide $L \in \mathrm{NP}$ in time n^k.
- Assume M has exactly *two* nondeterministic choices at each step: choices 0 and 1.
- For each input x, we construct circuit $R(x)$ such that $x \in L$ if and only if $R(x)$ is satisfiable.
- A sequence of nondeterministic choices is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

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

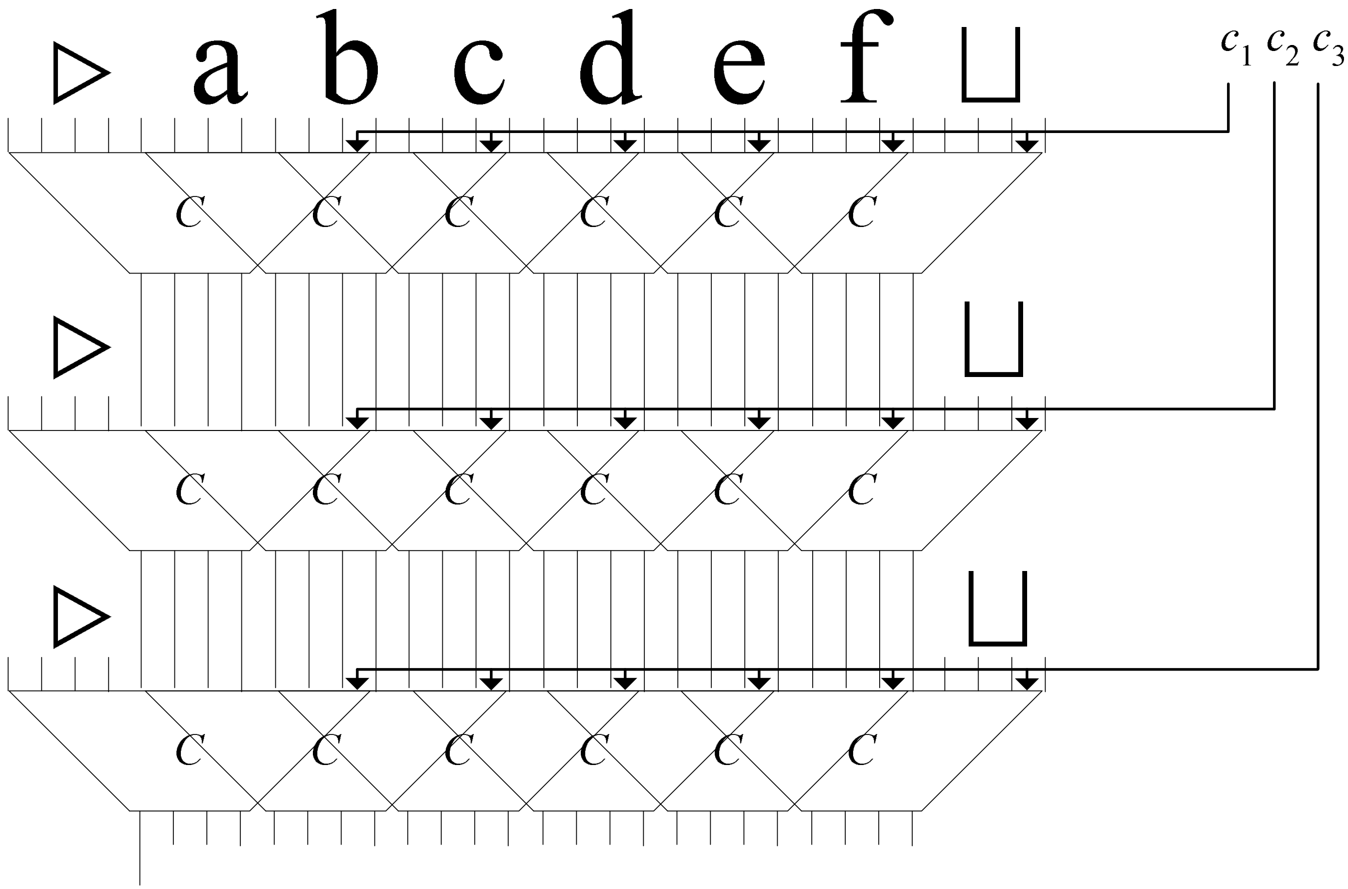
- Once B is given, the computation is *deterministic*.

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. 259.
- 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. 266) 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., $x \in L$.

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

---

[a] Proposition 31 (p. 271) remains valid if P is replaced by NP. Contributed by Mr. Cheng-Yu Lee (`R95922035`) on October 26, 2006.

An Alternative Characterization of NP

Proposition 31 (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*

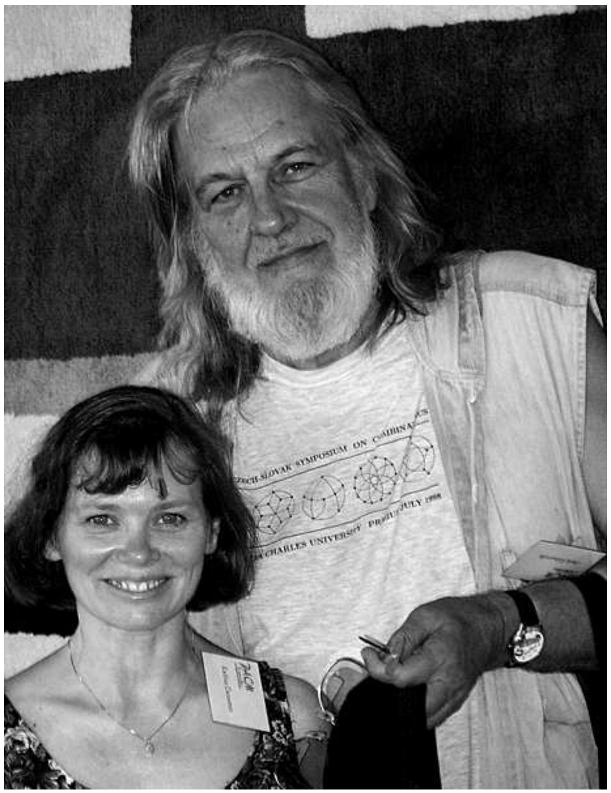
$$L = \{x : \exists y \, (x, y) \in R\}.$$

- Suppose such an R exists.
- L can be decided by this NTM:
  - On input x, the NTM guesses a y of length $\leq |\, x\,|^k$ and 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 checking with 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.
- Certificates may be hard to generate (otherwise, NP equals P), but verification must be easy.
- NP is the class of *easy-to-verify* (in P) problems.

Levin Reduction and Parsimonious Reductions

- The reduction R in Cook's theorem (p. 263) 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 co-inventor of NP-completeness.

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 timing requirement for parsimonious reduction: It runs in deterministic polynomial time.

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

- From Propositions 25 (p. 241) and Proposition 26 (p. 244), 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).

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. 263) and the reduction of CIRCUIT SAT to SAT (p. 226).
- The resulting CNF has at most 3 literals for each clause.
  - This 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, $3.52 < c_n < 4.596$.