HAMILTONIAN PATH

- A **Hamiltonian path** of a graph is a path that visits every node of the graph exactly once.
- Suppose graph G has n nodes: $1, 2, \ldots, n$.
- A Hamiltonian path can be expressed as a permutation π of $\{ 1, 2, \ldots, n \}$ such that
  - $\pi(i) = j$ means the ith position is occupied by node j.
  - $(\pi(i), \pi(i+1)) \in G$ for $i = 1, 2, \ldots, n-1$.

HAMILTONIAN PATH (concluded)

- So

$$\begin{pmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{pmatrix}.$$

- HAMILTONIAN PATH asks if a graph has a Hamiltonian path.

Reduction of HAMILTONIAN PATH to SAT

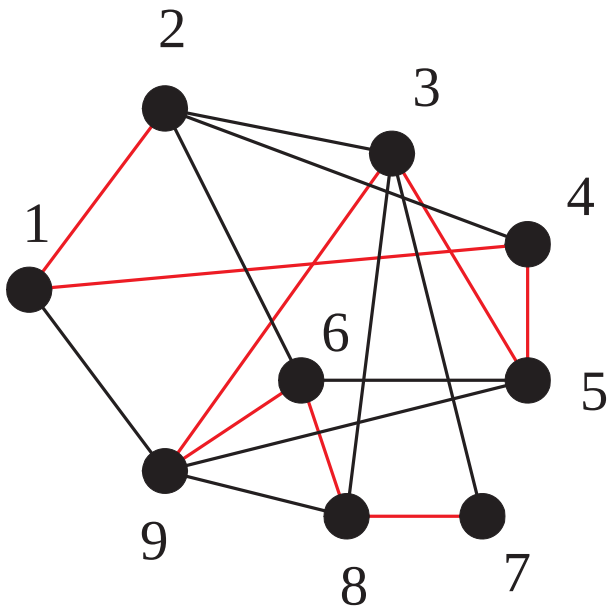
- Given a graph G, we shall construct a CNF[a] $R(G)$ such that $R(G)$ is satisfiable if and only if G has a Hamiltonian path.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- x_{ij} means

  the ith position in the Hamiltonian path is occupied by node j.[b]
- Our reduction will produce clauses.

[a] Remember that R does not have to be onto.

[b] That is, $\pi(i) = j$.

A Hamiltonian Path



$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{69} = x_{76} = x_{88} = x_{97} = 1;$
$\pi(1) = 2, \pi(2) = 1, \pi(3) = 4, \pi(4) = 5, \pi(5) = 3, \pi(6) =$
$9, \pi(7) = 6, \pi(8) = 8, \pi(9) = 7.$

The Clauses of $R(G)$ and Their Intended Meanings

1. Each node j must appear in the path.
   - $x_{1j} \vee x_{2j} \vee \cdots \vee x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \vee \neg x_{kj} (\equiv \neg(x_{ij} \wedge x_{kj}))$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \vee x_{i2} \vee \cdots \vee x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \vee \neg x_{ik} (\equiv \neg(x_{ij} \wedge x_{ik}))$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \vee \neg x_{k+1,j} (\equiv \neg(x_{k,i} \wedge x_{k+1,j}))$ for all $(i, j) \notin E$ and $k = 1, 2, \ldots, n-1$.

The Proof

- $R(G)$ contains $O(n^3)$ clauses.
- $R(G)$ can be computed efficiently (simple exercise).
- Suppose $T \models R(G)$.
- From the 1st and 2nd types of clauses, for each node j there is a unique position i such that $T \models x_{ij}$.
- From the 3rd and 4th types of clauses, for each position i there is a unique node j such that $T \models x_{ij}$.
- So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The Proof (concluded)

- The 5th type of clauses furthermore guarantee that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.
- Conversely, suppose G has a Hamiltonian path

$$(\pi(1), \pi(2), \ldots, \pi(n)),$$

  where π is a permutation.
- Clearly, the truth assignment

$$T(x_{ij}) = \texttt{true} \text{ if and only if } \pi(i) = j$$

  satisfies all clauses of $R(G)$.

A Comment[a]

- An answer to "Is $R(G)$ satisfiable?" answers the question "Is G Hamiltonian?"
- But a "yes" does not give a Hamiltonian path for G.
  - Providing a witness is not a requirement of reduction.
- A "yes" to "Is $R(G)$ satisfiable?" *plus* a satisfying truth assignment does provide us with a Hamiltonian path for G.

[a]Contributed by Ms. Amy Liu (J94922016) on May 29, 2006.

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.[a]

[a] Floyd (1962); Marshall (1962).

The Gates

- The gates are
  - g_{ijk} with $1 \le i, j \le n$ and $0 \le k \le n$.
  - h_{ijk} with $1 \le i, j, k \le n$.
- 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 if and only if C is.
  - $R(C)$ will turn out to be a CNF.
  - So $R(C)$ is basically a depth-2 circuit, where each gate has out-degree 1.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
  - The 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 $R(C)$ (continued)

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

- The conjunction of the above clauses is equivalent to

$$\begin{aligned} & [\, g \Rightarrow (h \vee h') \,] \wedge [\, (h \vee h') \Rightarrow g \,] \\ \equiv \quad & g \Leftrightarrow (h \vee h'). \end{aligned}$$

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

- It is equivalent to

$$g \Leftrightarrow (h \wedge h').$$

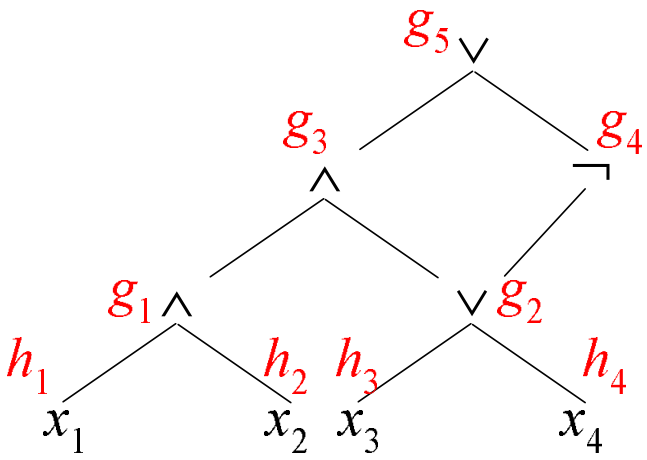
The Clauses of $R(C)$ (concluded)

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



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

An Example (continued)

- The result is a CNF.
- The CNF adds new variables to the circuit's original input variables.
- The CNF has size proportional to the circuit's number of gates.
- Had we used the idea on p. 217 for the reduction, the resulting formula may have an exponential length because of the copying.[a]

[a] Contributed by Mr. Ching-Hua Yu (D00921025) on October 16, 2012.

An Example (concluded)

- But is $R(C)$ *valid* if and only if C is?[a]
- In general, no.
- For example, the circuit equivalent to the valid $x_1 \vee \neg x_1$ is turned into

$$(h_1 \Leftrightarrow x_1) \wedge (h_2 \Leftrightarrow \neg x_1) \wedge [\, g_1 \Leftrightarrow (h_1 \vee h_2)\,] \wedge (g_1).$$

- This expression is clearly not valid.[b]
- So the reduction preserves satisfiability but *not* validity.

[a] Contributed by Mr. Han-Ting Chen (`R10922073`) on October 21, 2021.

[b] Assign `false` to g_1, e.g.

Composition of Reductions

Proposition 28 *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.*

- So reducibility is transitive.[a]

[a] See Proposition 8.2 of the textbook for a proof.

Completeness[a]

- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element (the so-called *hardest* problem)?
- It is not obvious that there should be a maximal element.
  - Many infinite structures (such as integers and real numbers) do not have maximal elements.
- Surprisingly, most of the complexity classes that we have seen so far have maximal elements!

[a]Post (1944); Cook (1971); Levin (1973).

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 of the complexity classes we have seen so far have complete problems!
- Complete problems capture the difficulty of a class because they are the hardest problems in the class.[a]

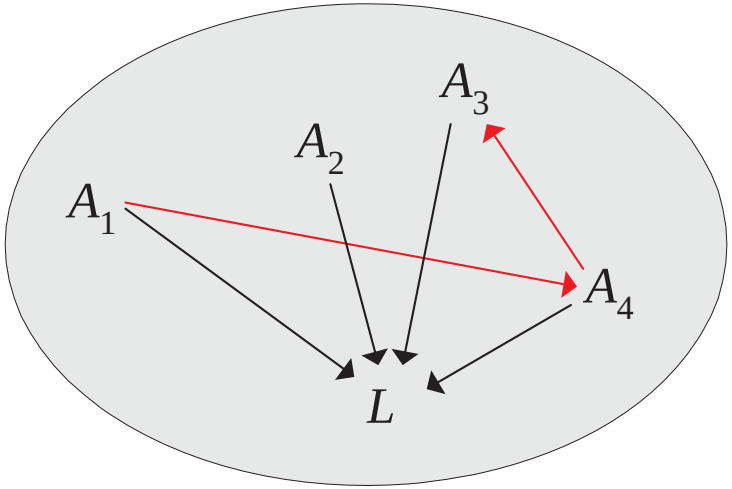
[a] See also p. 167.

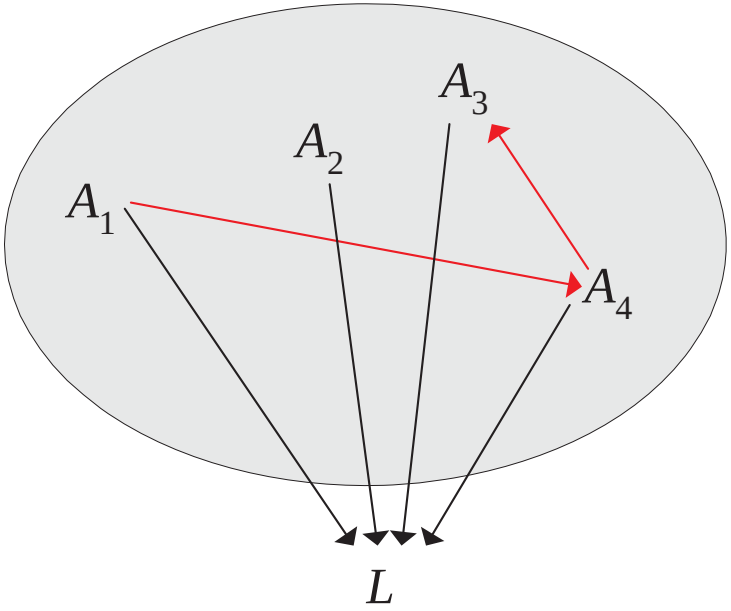
Hardness

- 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 Reductions

- A class $\mathcal{C}$ is **closed under reductions** if whenever L is reducible to L' and $L' \in \mathcal{C}$, then $L \in \mathcal{C}$.
- It is easy to show that P, NP, coNP, L, NL, PSPACE, and EXP are all closed under reductions.
- E is not closed under reductions.[a]

[a] Balcázar, Díaz, & Gabarró (1988).

Complete Problems and Complexity Classes

Proposition 29 *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 29 implies the following.

Corollary 30 *$P = NP$ if and only if an NP-complete problem is in P.*

Corollary 31 *$L = P$ if and only if a P-complete problem is in L.*

Complete Problems and Complexity Classes, Again

Proposition 32 *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 $A \in \mathcal{C}$ reduce to $L \in \mathcal{C}$ and $L \in \mathcal{C}'$.
- Since $\mathcal{C}'$ is closed under reductions, $A \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.
- The proof for $\mathcal{C}' \subseteq \mathcal{C}$ is symmetric.

Complete Problems and Complexity Classes, Again (concluded)

Proposition 33 *Let $\mathcal{C}$ be a complexity class. If L is $\mathcal{C}$-complete and L is reducible to $L' \in \mathcal{C}$, then L' is also $\mathcal{C}$-complete.*

- Every language $A \in \mathcal{C}$ reduces to L.
- By Proposition 28 (p. 300), A reduces to L'.

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 essentially 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.[a]
- 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.

[a] $|\,x\,|^k - 3$ may be safer.

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 fixed $\triangleright$ and not a $\triangleright_q$.

Some Conventions To Simplify the Table (concluded)

- M will halt before the last row is reached.
- All subsequent rows will be identical to the row where M halts.
- 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

$$\overbrace{\triangleright \mathsf{0_s 10001}\, \sqcup\, \sqcup \cdots \sqcup}^{|\, x\, |^k}$$

- A typical row looks like

$$\overbrace{\triangleright \mathsf{10100_q 01110100}\, \sqcup\, \sqcup \cdots \sqcup}^{|\, x\, |^k}$$

Comments (concluded)

- The last rows must look like

$$\overbrace{\triangleright \cdots \text{“yes”} \cdots \sqcup}^{|\,x\,|^k} \quad \text{or} \quad \overbrace{\triangleright \cdots \text{“no”} \cdots \sqcup}^{|\,x\,|^k}$$

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

$$\begin{array}{llllllll}
\triangleright & \mathrm{a} & \mathrm{b} & \mathrm{c} & \mathrm{d} & \mathrm{e} & \mathrm{f} & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
\triangleright & & & & & & & \sqcup \\
 & & & & \vdots & & &
\end{array}$$

A P-Complete Problem

Theorem 34 (Ladner, 1975) CIRCUIT VALUE *is P-complete.*

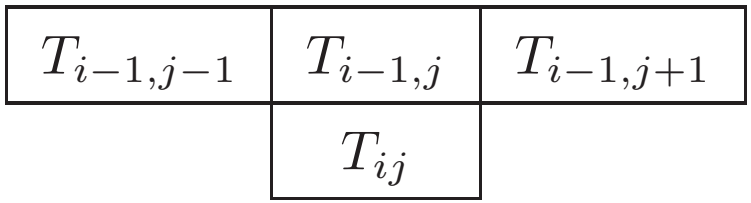
- It is easy to see that CIRCUIT VALUE $\in$ P.
- For *any* $L \in$ P, we will construct a reduction R from L to CIRCUIT VALUE.
- Given any input x, $R(x)$ is a variable-free circuit such that $x \in L$ if and only if $R(x)$ evaluates to true.
- Let M decide L in time n^k.
- Let T be the computation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

The Proof (continued)

- Recall that three out of T's four borders are known.
- So when $i = 0$, or $j = 0$, or $j = |\, x \,|^k - 1$, the value of T_{ij} is known.
  - The jth symbol of x or $\sqcup$, a $\rhd$, or a $\sqcup$, respectively.
- Consider *other* entries T_{ij}.

The Proof (continued)

- T_{ij} depends on *only* $T_{i-1,j-1}$, $T_{i-1,j}$, and $T_{i-1,j+1}$:



- T_{ij} does *not* depend on any other entries!
- T_{ij} does *not* depend on i, j, or x directly either (given $T_{i-1,j-1}$, $T_{i-1,j}$, and $T_{i-1,j+1}$).
- The dependency is thus "local."

The Proof (continued)

- 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,[a] where

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

[a] 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

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

The Proof (continued)

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

$T_{i-1,j-1}$:	$S_{i-1,j-1,1}$	$S_{i-1,j-1,2}$	$\cdots$	$S_{i-1,j-1,m}$
$T_{i-1,j}$:	$S_{i-1,j,1}$	$S_{i-1,j,2}$	$\cdots$	$S_{i-1,j,m}$
$T_{i-1,j+1}$:	$S_{i-1,j+1,1}$	$S_{i-1,j+1,2}$	$\cdots$	$S_{i-1,j+1,m}$

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

The Proof (continued)

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

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

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

The Proof (continued)

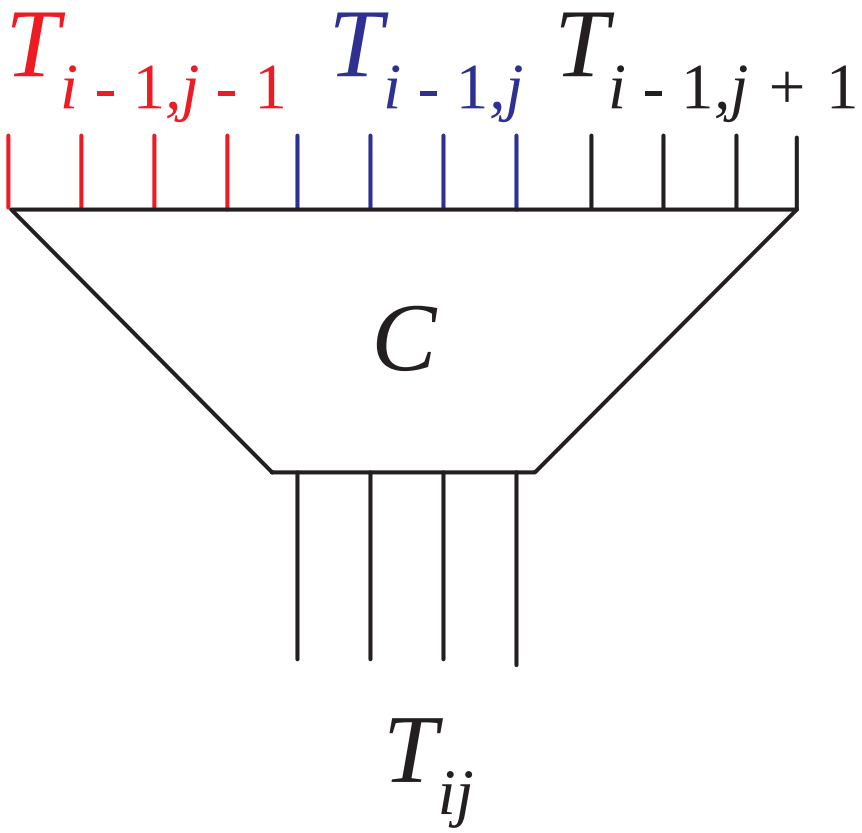
- These F_ℓs depend only on M's specification, not on x, i, or j.
- Their sizes are constant.[a]
- These boolean functions can be turned into boolean circuits.[b]
- Compose these m circuits in parallel to obtain circuit C with $3m$-bit inputs and m-bit outputs.
  - Schematically, $C(T_{i-1,j-1}, T_{i-1,j},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

[a] It means independence of the input x.

[b] Recall p. 216.

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

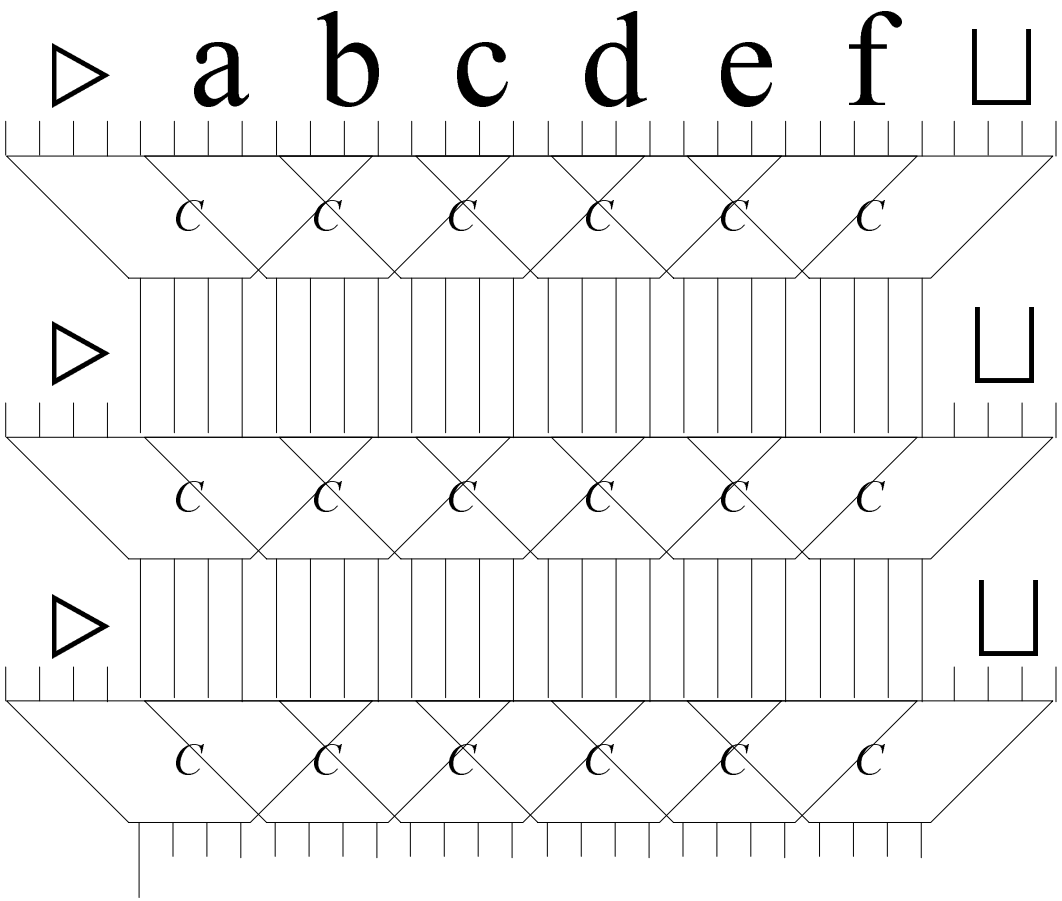
Circuit C



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 35 *If $L \in TIME(T(n))$, then it can be decided by a circuit with $O(T^2(n))$ gates.*

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 36 (Goldschlager, 1977) MONOTONE CIRCUIT VALUE *is P-complete.*

Theorem 37 (Goldschlager, 1977) PLANAR MONOTONE CIRCUIT VALUE *is P-complete.*

MAXIMUM FLOW Is P-Complete

Theorem 38 (Goldschlager, Shaw, & Staples, 1982)
MAXIMUM FLOW *is P-complete.*

Cook's Theorem: the First NP-Complete Problem

Theorem 39 (Cook, 1971) SAT *is NP-complete.*

- SAT $\in$ NP.[a]
- CIRCUIT SAT reduces to SAT.[b]
- By Proposition 33 (p. 310), it remains to show that all languages in NP can be reduced to CIRCUIT SAT.[c]

[a] Recall p. 120.

[b] Recall p. 293.

[c] 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.
- Need to do better.

[a]The circuit for Theorem 34 (p. 317) will do.

The Proof (continued)

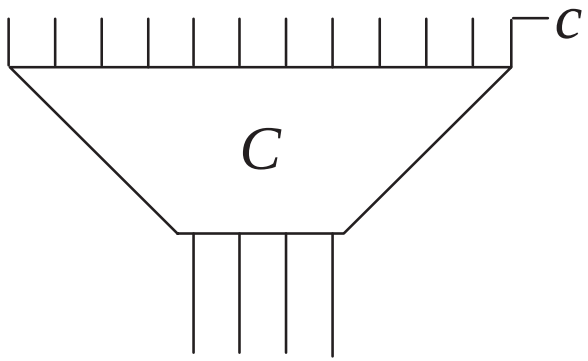
- A sequence of nondeterministic choices is a bit string

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

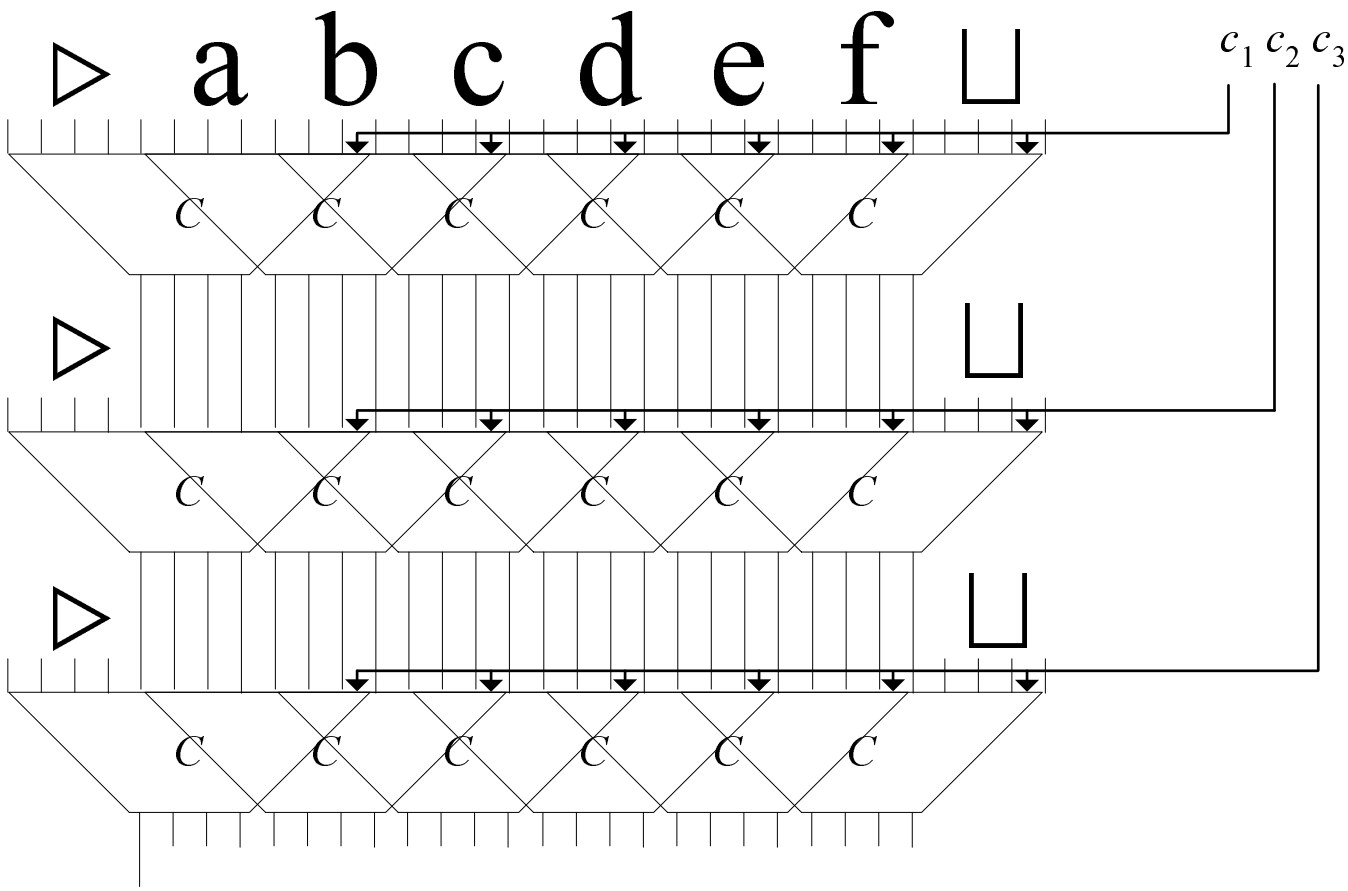
- Once B is given, the computation is *deterministic*.
- Each choice of B results in a deterministic polynomial-time computation.
- Each circuit C at time i has an *extra* binary input c corresponding to the nondeterministic choice:

$$C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}, c) = T_{ij}.$$

The Proof (continued)



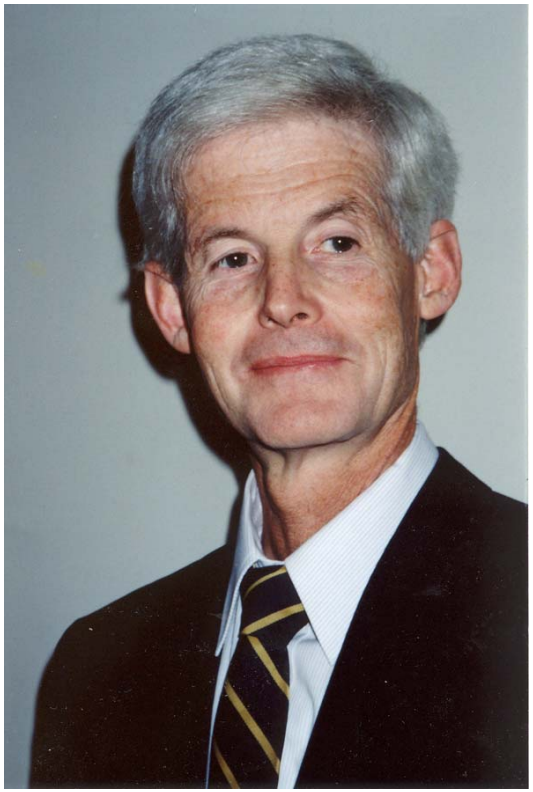
The Computation Tableau for NTMs and $R(x)$



The Proof (concluded)

- Note that $c_1, c_2, \ldots, c_{|x|^k-1}$ constitute the variables of $R(x)$.
  - Some call them the choice gates of the circuit.
- The overall circuit $R(x)$ (on p. 337) 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.

A Corollary

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

Corollary 40 *If $L \in NTIME(T(n))$, then it can be decided by a nondeterministic circuit with $O(T^2(n))$ gates.*

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 $k \geq 1$.[a]

[a] If the polynomial bound is replaced by a different class of functions, different complexity classes may arise. Contributed by Mr. Vincent Hwang (R10922138) on November 11, 2021.

An Alternative Characterization of NP

Proposition 41 (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*

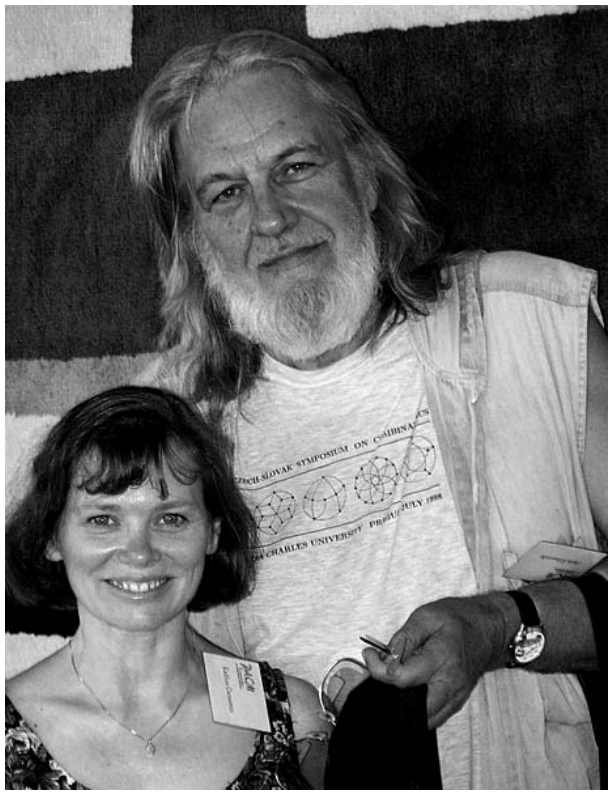
$$L = \{\, x : \exists y \, (x, y) \in R \,\}.$$

- 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 runs in polynomial time.
- 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.
- Certificates may be hard to generate,[a] but verification must be easy.
- NP is thus the class of *easy-to-verify*[b] problems.

[a] Unless P equals NP.

[b] That is, in polynomial time.

Comments (concluded)

- The degree k is not an input.
- How to find the k needed by the NTM is of no concern.[a]
- We only need to prove there *exists* an NTM that accepts L in nondeterministic polynomial time.
- If R is not required to be polynomial balanced, then its running time may not be polynomial in $|\,x\,|$.[b]
  - It is still polynomial in $|\,x\,| + |\,y\,|$.

[a] Contributed by Mr. Kai-Yuan Hou (B99201038, R03922014) on November 3, 2015.

[b] Contributed by Mr. Ri-Sheng Huang (B09902012) on November 11, 2021.

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

- From Propositions 29 (p. 306) and 32 (p. 309), 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)

Correct 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. 278),

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

- 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] Proposition 33 (p. 310).

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[a]

- Recall Cook's Theorem (p. 332) and the reduction of CIRCUIT SAT to SAT (p. 293).
- 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

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

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

Michael R. Garey (1945–)

David S. Johnson (1945–)

Larry Stockmeyer (1948–2004)

The Satisfiability of Random 3SAT Expressions

- Consider a random 3SAT expression 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 & Bourgain (1999). As of 2006, $3.52 < c_n < 4.596$.