Reductions and Completeness

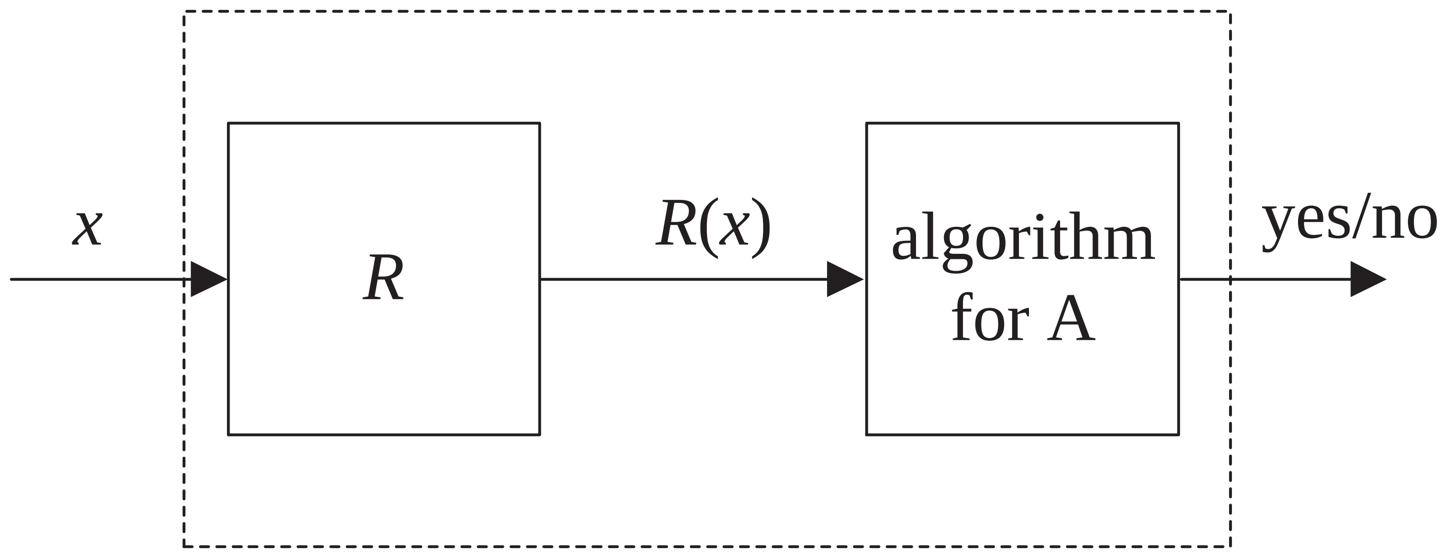
Degrees of Difficulty

- When is a problem more difficult than another?
- B **reduces to** A if there is a transformation R which for every input x of B yields an equivalent input $R(x)$ of A.
  - The answer to x for B is the same as the answer to $R(x)$ for A.
  - There must be restrictions on the complexity of computing R.
  - Otherwise, $R(x)$ might as well solve B.

Degrees of Difficulty (concluded)

- Problem A is at least as hard as problem B if B reduces to A.
- This makes intuitive sense: If A is able to solve your problem B, then A must be at least as powerful.

Reduction



Solving problem B by calling the algorithm for problem *once* and *without* further processing its answer.

Comments[a]

- Suppose B reduces to A via a transformation R.
- The input x is an instance of B.
- The output $R(x)$ is an instance of A.
- $R(x)$ may not span all possible instances of A.
- So some instances of A may never appear in the reduction.

[a] Contributed by Mr. Ming-Feng Tsai (D92922003) on October 29, 2003.

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in space $O(\log n)$.
- Furthermore, for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is said to be a (**Karp**) **reduction** from L_1 to L_2.
- Note that by Theorem 20 (p. 176), R runs in polynomial time.
- If R is a reduction from L_1 to L_2, then $R(x) \in L_2$ is a legitimate algorithm for $x \in L_1$.

A Paradox?

- Degree of difficulty is not defined in terms of *absolute* complexity.
- So a language $\mathrm{B} \in \mathrm{TIME}(n^{99})$ may be "easier" than a language $\mathrm{A} \in \mathrm{TIME}(n^3)$.
- This happens when B is reducible to A.
- But isn't this a contradiction when $\mathrm{B} \notin \mathrm{TIME}(n^k)$ for any $k < 99$?
- That is, how can a problem *requiring* n^{33} time be reducible to a problem solvable in n^3 time?

A Paradox? (concluded)

- The so-called contradiction is more apparent than real.
- When we solve the problem "$x \in$ B?" with "$R(x) \in$ A?", we must consider the time spent by $R(x)$ and its length $|\, R(x) \,|$.
- If $|\, R(x) \,| = \Omega(n^{33})$, then the time of answering "$R(x) \in$ A?" becomes $\Omega((n^{33})^3) = \Omega(n^{99})$.
- Suppose, on the other hand, that $|\, R(x) \,| = o(n^{33})$.
- Then $R(x)$ must run in time $\Omega(n^{99})$.
- In either case, there is no contradiction.

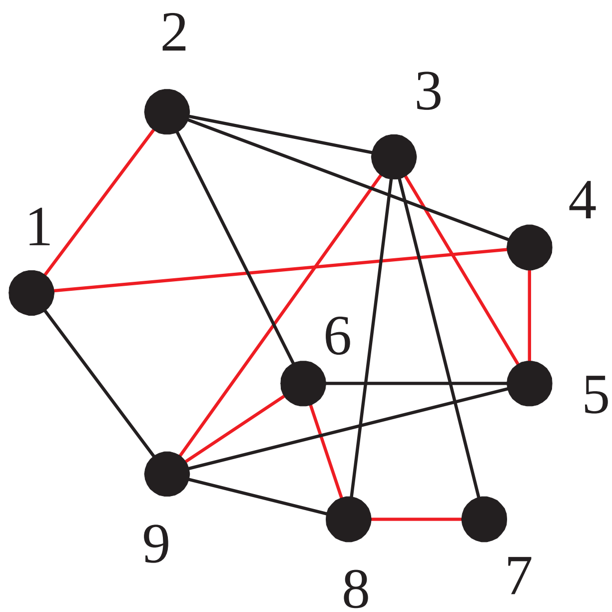
HAMILTONIAN PATH

- 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 asks if a graph has a Hamiltonian path.

Reduction of HAMILTONIAN PATH to SAT

- Given a graph G, we shall construct a CNF $R(G)$ 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.



$$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{69} = x_{76} = x_{88} = x_{97} = 1.$$

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

1. Each node j must appear in the path.
   - $x_{1j} \lor x_{2j} \lor \cdots \lor x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \lor \neg x_{kj}$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \lor x_{i2} \lor \cdots \lor x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \lor \neg x_{ik}$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \lor \neg x_{k+1,j}$ for all $(i, j) \notin G$ 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 Clauses of 1 and 2, for each node j there is a unique position i such that $T \models x_{ij}$.
- From Clauses of 3 and 4, 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)

- Clauses of 5 furthermore guarantees 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?" does answer "Is G Hamiltonian?"
- But a positive answer does not give a Hamiltonian path for G.
  - Providing witness is not a requirement of reduction.
- A positive answer 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.

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 $g_{ij0} = \texttt{true}$ 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 shall construct a boolean expression $R(C)$ such that $R(C)$ is satisfiable if and only if C is satisfiable.
  - $R(C)$ will turn out to be a CNF.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
- Each gate of C will be turned into equivalent clauses of $R(C)$.
- Recall that clauses are $\land$-ed together.

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

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

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

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

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

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

Composition of Reductions

Proposition 23 *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 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 committed 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).

Completeness (concluded)

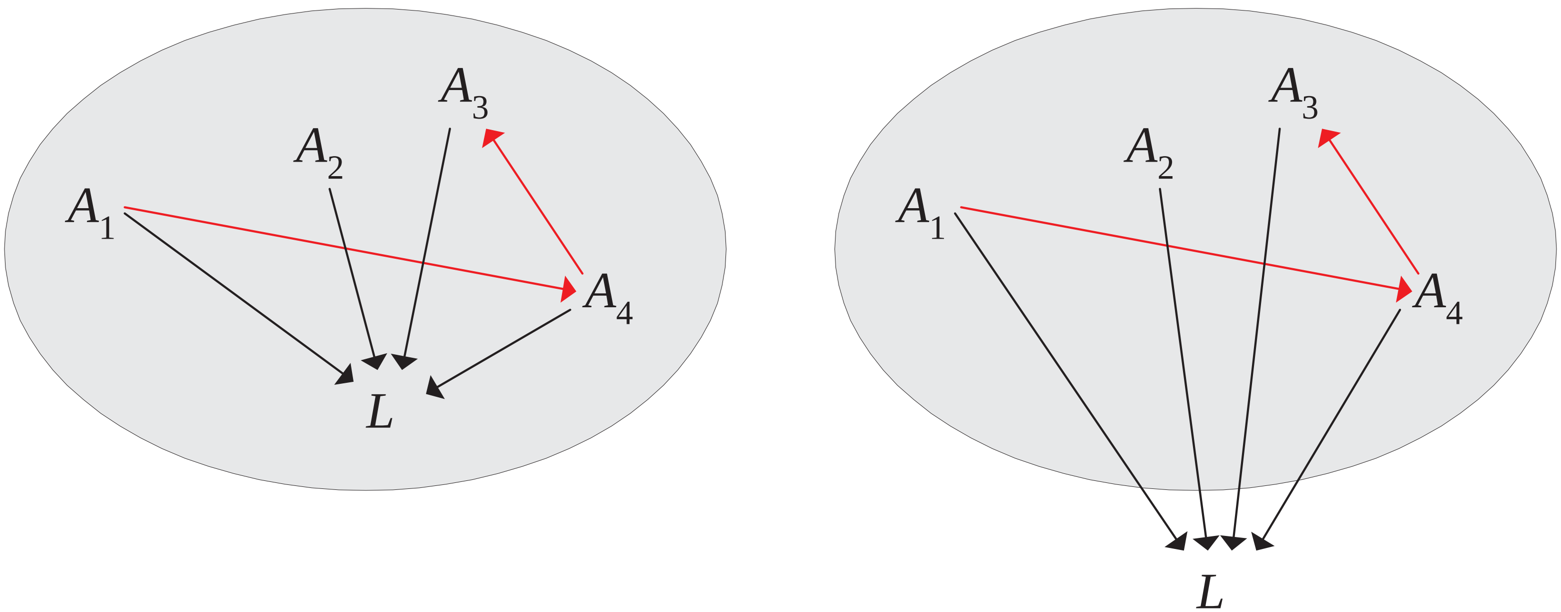
- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if every $L' \in \mathcal{C}$ can be reduced to L.
  - Most complexity classes we have seen so far have complete problems!
- Complete problems capture the difficulty of a class because they are the hardest.

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 24 *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 a complete problem for $\mathcal{C}$. 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 24 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 25 *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 (recall that it is an upper 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

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

- The last rows must look like $\overbrace{\rhd \cdots \text{“yes”} \cdots \sqcup}^{|\,x\,|^k}$

A P-Complete Problem

Theorem 26 (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 $\rhd$, and a $\sqcup$, respectively.
  - Three out of four of T's borders are known.

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

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:

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

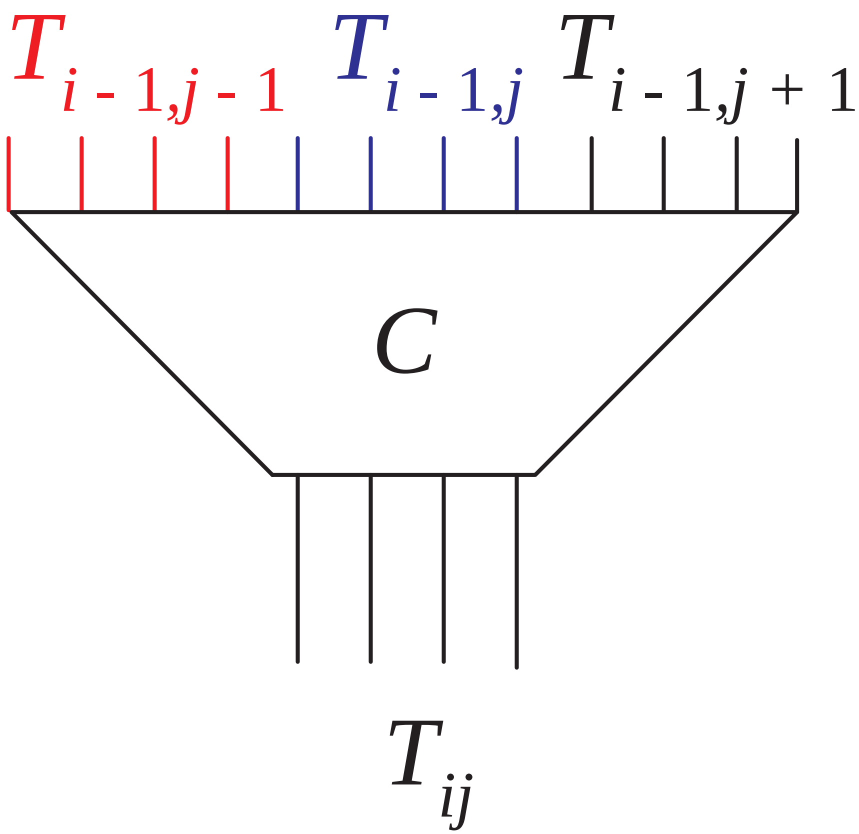
- So there are m boolean functions $F_1, F_2, \ldots, F_m$ with $3m$ inputs each such that for all $i, j > 0$,

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

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.

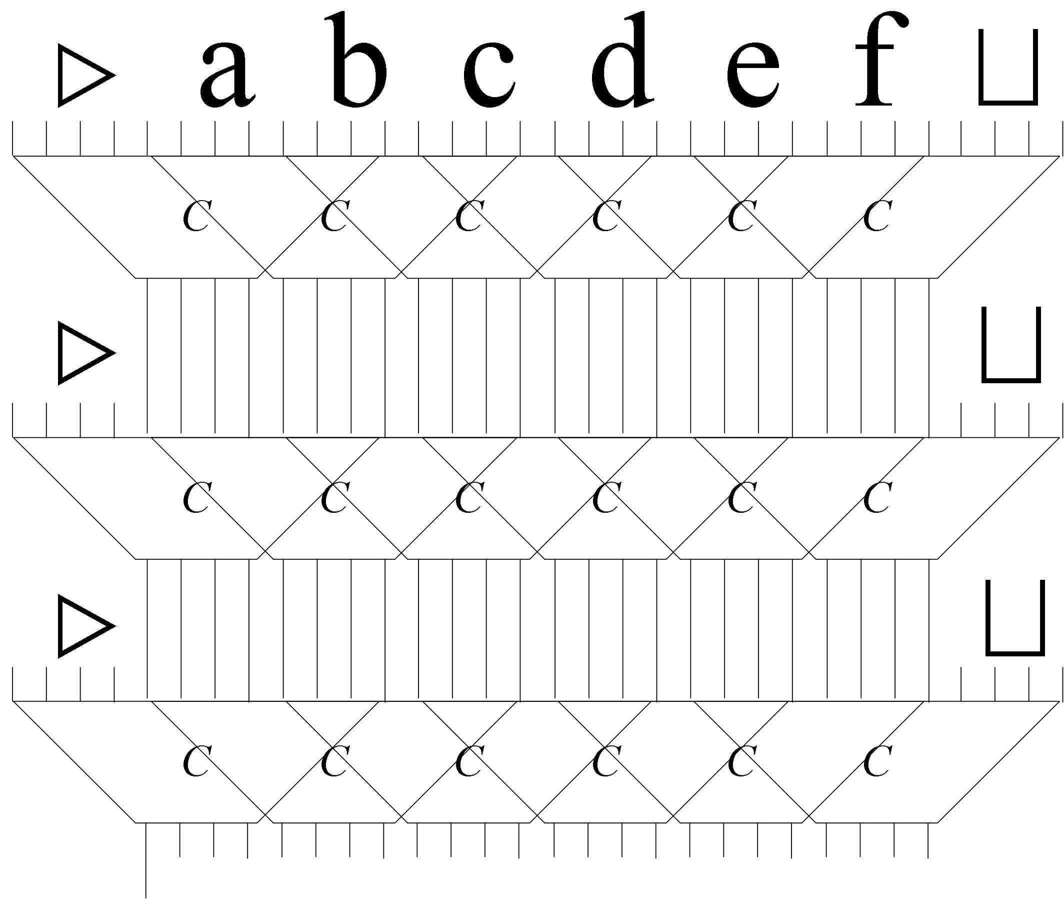
Circuit C



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 shows the following.

Corollary 27 *If* $L \in \mathit{TIME}(T(n))$*, 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 as 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 28 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 29 (Cook (1971)) SAT *is NP-complete.*

- SAT $\in$ NP (p. 80).
- CIRCUIT SAT reduces to SAT (p. 210).
- Now we only need to show that all languages in NP can be reduced to CIRCUIT SAT.

The Proof (continued)

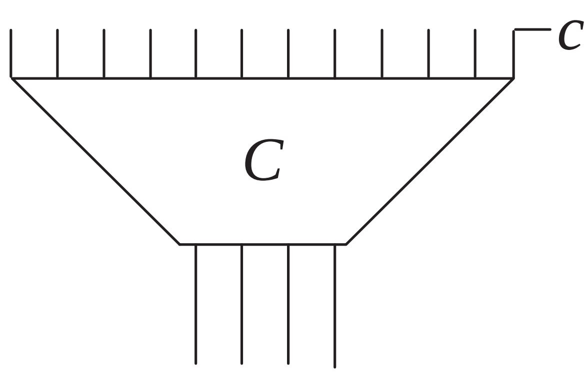
- Let single-string NTM M decide $L \in \mathrm{NP}$ in time n^k.
- Assume M has exactly *two* nondeterministic choices at each step: choices 0 and 1.
- For each input x, we construct circuit $R(x)$ such that $x \in L$ if and only if $R(x)$ is satisfiable.
- A sequence of nondeterministic choices is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

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

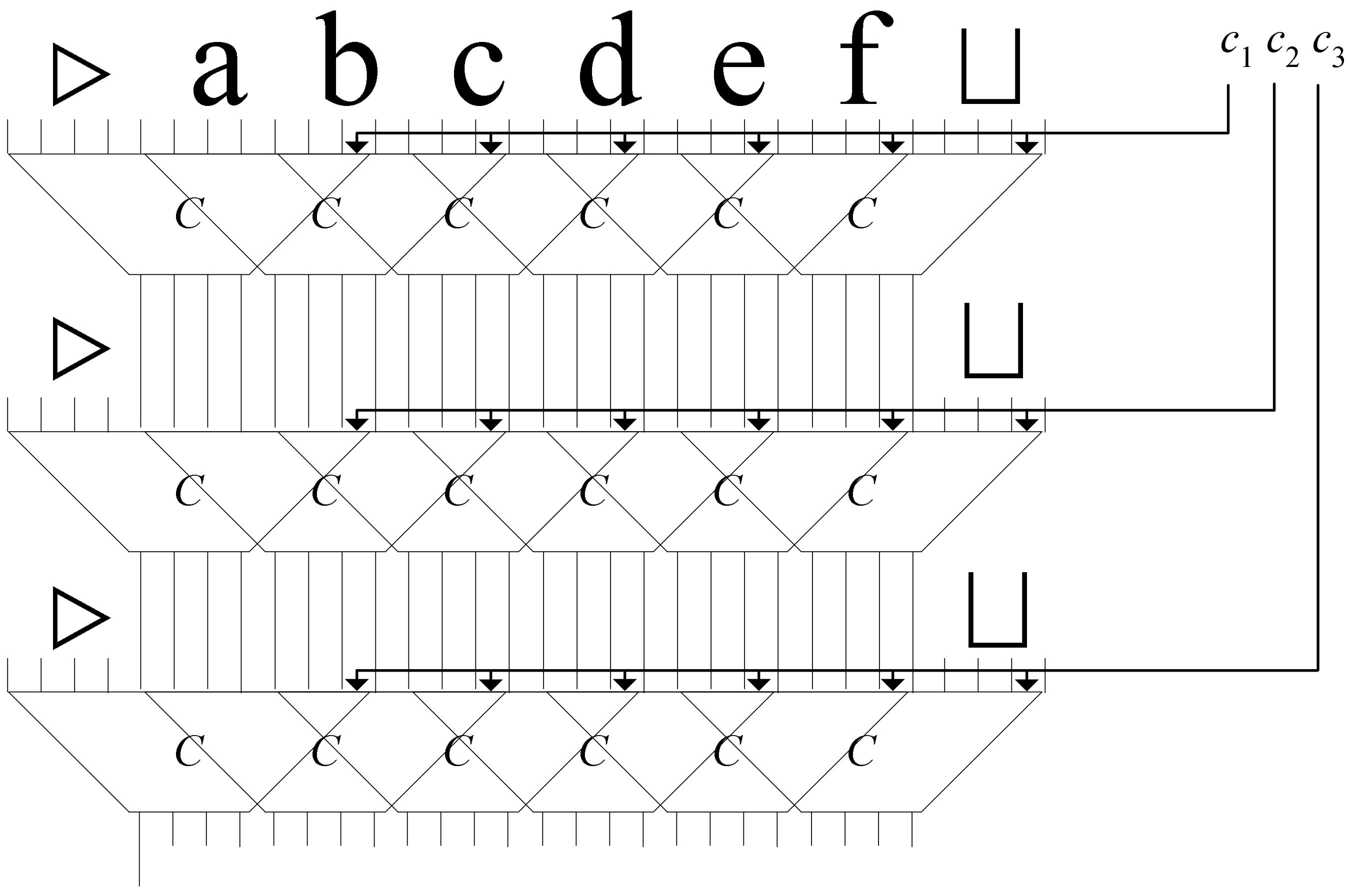
- Once B is fixed, the computation is *deterministic*.

The Proof (continued)

- Each choice of B results in a deterministic polynomial-time computation, hence a table like the one on p. 238.
- 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. 245) 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., $x \in L$.

Parsimonious Reductions

- The reduction R in Cook's theorem (p. 242) is such that
  - Each satisfying truth assignment for circuit $R(x)$ corresponds to an accepting computation path for $M(x)$.
- 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.