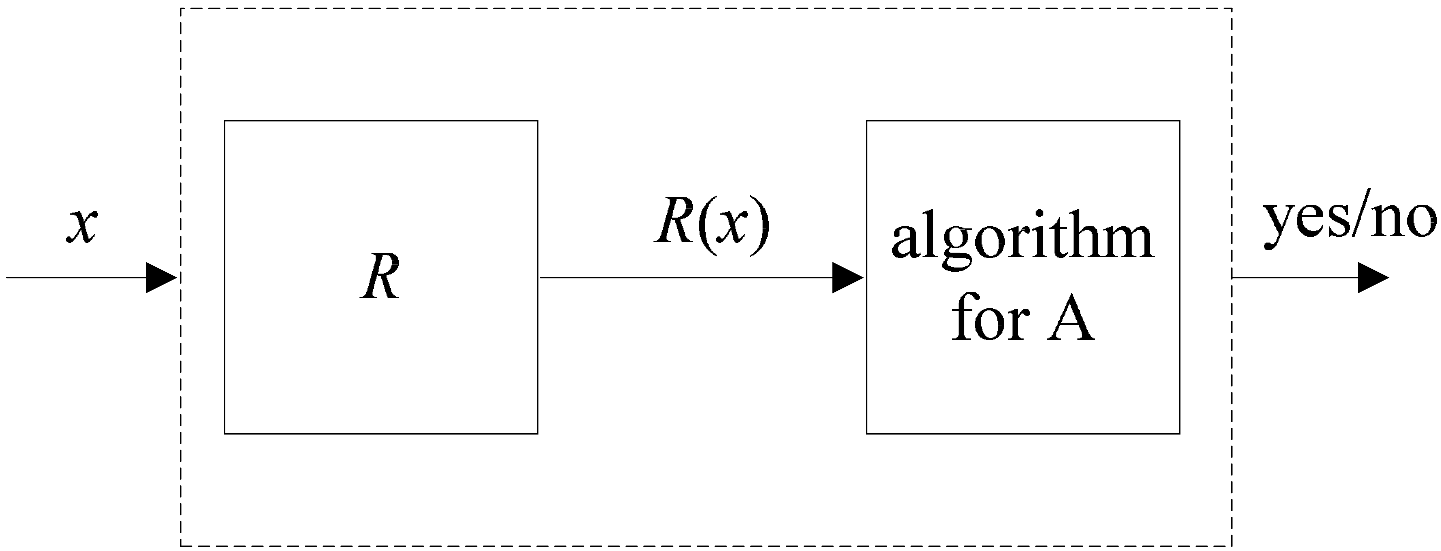
Degrees of Difficulty

- 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.
- Problem A is at least as hard as problem B if B reduces to A.

Reduction



- Solving problem B with algorithm for problem A.

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in space $O(\log n)$—hence polynomial time—such that
  - for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is called a **reduction** from L_1 to L_2.
- Degree of difficulty is not defined in terms of *absolute* complexity.
  - It is possible for a language in TIME(n^3) to be reducible to a language in TIME(n^2).
  - R can lengthen the input or may run in time n^3.

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.
- Suppose G has n nodes: $1, 2, \ldots, n$.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- In particular, x_{ij} means "node j is the ith node in the Hamiltonian path."

The Clauses of $R(G)$

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

- $R(G)$ can be computed efficiently.
- Suppose $T \models R(G)$.
- Clauses of 1 and 2 imply that for each j, there is a unique i such that $T \models x_{ij}$.
- Clauses of 3 and 4 imply that for each i, there is a unique 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 $T \models x_{ij}$.
- Clauses of 5 guarantees that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.

The Proof (continued)

- Conversely, suppose that 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 $R(G)$.

Reduction of REACHABILITY to CIRCUIT VALUE

- Note that both problems are in P.
- Given a graph G, we shall construct a *variable-free* circuit $R(G)$.
  - Incidentally, $R(G)$ will not have $\neg$ gates.
- 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 $(i, j) \in G$.

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**.
- The depth of $R(G)$ is $O(n)$, which is not optimal.

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 clauses of $R(C)$.

The Clauses of $R(C)$

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

- 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 $(\neg g \lor \neg h)$ and $(g \lor h)$.

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

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

g **is a** $\vee$ **gate with predecessor gates** h **and** h'**:** Add clauses $(\neg h \vee g)$, $(\neg h' \vee g)$, 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 22 *If R is a reduction from L_1 to L_2 and R' is a reduction from L_2 to L_3, then the composition $R \cdot R'$ is a reduction from L_1 to L_3.*

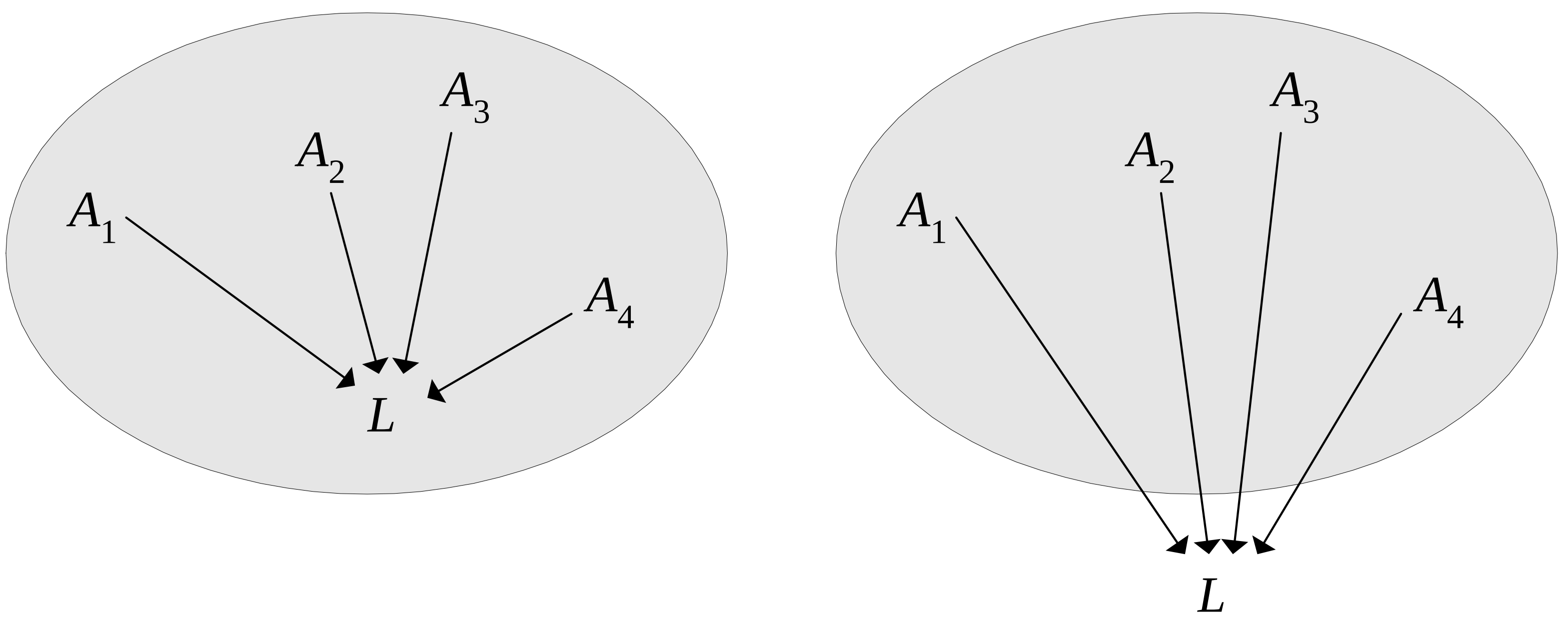
- Clearly $x \in L_1$ if and only if $R'(R(x)) \in L_3$.
- $R \cdot R'$ can be computed in space $O(\log n)$.
  - Generating $R(x)$ before feeding it to R' may consume too much space because $R(x)$ is on a work string. [No problem if we require reductions to be in P not L.]
  - The trick is to let R' drive the computation: It asks R to deliver each bit of $R(x)$ when needed.
  - Recall that $R(x)$ is produced in a *write-only* manner.

Completeness[a]

- Now that reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element?
- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if any $L' \in \mathcal{C}$ can be reduced to L.
  - Every complexity class we have seen so far has complete problems!
- Complete problems capture the difficulty of a class; they are also the hardest.

[a] Cook, 1971.

Illustration of Completeness



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 23 *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 $L \in \mathcal{C}'$.*

- Every language $A \in \mathcal{C}$ reduces to $L \in \mathcal{C}'$.
- Because $\mathcal{C}'$ is closed under reductions, $A \in \mathcal{C}'$.
- Hence $\mathcal{C} \subseteq \mathcal{C}'$.

The above proposition implies that

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

Complete Problems and Complexity Classes (continued)

Proposition 24 *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 in $\mathcal{C}$ reduce to $L \in \mathcal{C}'$.
- Since $\mathcal{C}'$ is closed under reductions, $\mathcal{C} \subseteq \mathcal{C}'$.
- The proof for $\mathcal{C}' \subseteq \mathcal{C}$ is symmetric.

Table of Computation

- Let $M = (K, \Sigma, \delta, s)$ be a 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.
- Rows are time steps (0 to $|x|^k - 1$).
- Columns are positions in the string of the TM (the same range).
- 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 has one string and halts after at most $|x|^k - 2$ steps.
  - 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.
  - If q is "yes" or "no," simply use "yes" or "no" instead of σ_q.

Some Conventions To Simplify the Table (continued)

- Modify M so that the cursor starts not at $\rhd$ but at the first symbol of the input.
- The cursor never visits the leftmost $\rhd$ by telescoping two moves of M each time the cursor is about to move to the leftmost $\rhd$.
  - The first symbol in every row is a $\rhd$ and not a $\rhd_q$.
- 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 be like $\overbrace{\rhd \cdots \text{“yes”} \cdots \sqcup}^{|x|^k}$

The First Complete Problem

Theorem 25 (Ladner, 1975) CIRCUIT VALUE *is P-complete.*

- CIRCUIT VALUE is in P.
- For any $L \in \mathrm{P}$, 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.
  - Three out of four of T's borders are known.
- 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}	

The Proof (continued)

- Let Γ denote the set of all symbols that can appear *on the table*.
- Encode each symbol of Γ as an m-bit number, where

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

  – Called **state assignment** in circuit design.

- The computation table is now a table of binary entries $S_{ij\ell}$, where $0 \leq i \leq n^k - 1$, $0 \leq j \leq n^k - 1$, and $1 \leq \ell \leq m$.
  – $S_{ij1} S_{ij2} \cdots S_{ijm}$ encodes T_{ij}.

The Proof (continued)

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

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

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

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

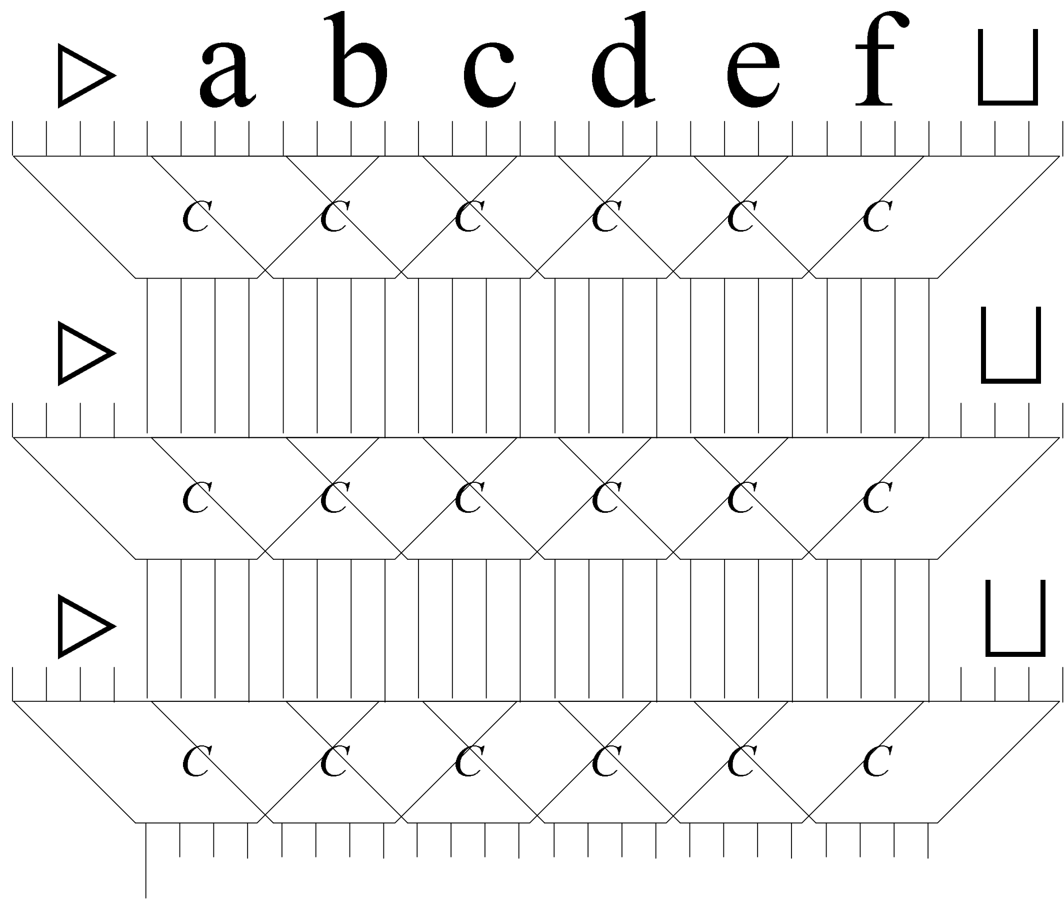
The Proof (continued)

- These F_i's depend on only M's specification, not on x.
- Their sizes are fixed.
- They can be turned into boolean circuits.
- Compose these m circuits in parallel to obtain circuit C with $3m$-bit inputs and m-bit outputs.
  - $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}) = T_{ij}$.
  - C is like an ASIC (application-specific IC) chip.

The Proof (continued)

- 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 $R(x)$



MONOTONE CIRCUIT VALUE Is P-Complete

- Monotone boolean circuits are less expressive than general circuits because they can compute only **monotone boolean functions**.
  - Their output cannot change from true to false when one input changes from false to true.
- However, MONOTONE CIRCUIT VALUE is as hard as CIRCUIT VALUE.

Corollary 26 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 27 (Cook, 1971) SAT *is NP-complete.*

- SAT is in NP (p. 61).
- CIRCUIT SAT reduces to SAT (p. 156).
- 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

$$B = (c_0, c_1, \ldots, c_{|x|^k-1}) \in \{0,1\}^{|x|^k}.$$

- 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. 175.
- Each circuit C at time i has an extra binary input c corresponding to the nondeterministic choice.
- The overall circuit $R(x)$ (on p. 180) 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., $x \in L$.

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