Composition of Reductions

Proposition 26 *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 $R_{12} \cdot R_{23}$ 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 $R_{12} \cdot R_{23}$ in space $O(\log n)$?
  - Generating $R_{12}(x)$ before feeding it to R_{23} may consume too much space because $R_{12}(x)$ is on a work string.[a]

[a]This would not be a problem if we had 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 run until the ith bit is available; the beginning $i - 1$ 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]

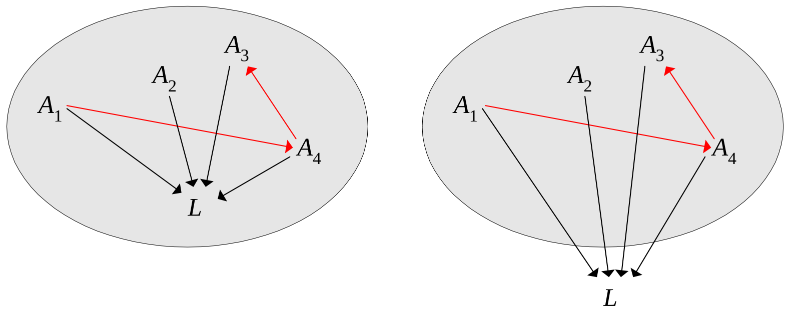
- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element?
- Let $\mathcal{C}$ be a complexity class and $L \in \mathcal{C}$.
- L is **$\mathcal{C}$-complete** if every $L' \in \mathcal{C}$ can be reduced to L.
  - Every complexity class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

[a]Cook, 1971.

Hardness

- Let $\mathcal{C}$ be a complexity class.
- L is **$\mathcal{C}$-hard** if every $L' \in \mathcal{C}$ can be reduced to L.
- Note that it is not required that $L \in \mathcal{C}$.

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 27 *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 $\mathcal{C} \subseteq \mathcal{C}'$.

Two Immediate Corollaries

Proposition 27 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 28 *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

$$\overbrace{\triangleright 0_{\mathsf{s}} 10001 \sqcup \sqcup \cdots \sqcup}^{|x|^k}$$

- A typical row may be

$$\overbrace{\triangleright 10100_{\mathsf{q}} 01110100 \sqcup \sqcup \cdots \sqcup}^{|x|^k}$$

- The last rows must look like $\overbrace{\triangleright \cdots \text{"yes"} \cdots \sqcup}^{|x|^k}$

A P-Complete Problem

Theorem 29 (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.
  - Three out of four of T's borders are known.

$\triangleright$	a	b	c	d	e	f	$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$
$\triangleright$							$\sqcup$

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: $\Sigma \cup \{\sigma_q : \sigma \in \Sigma, q \in K\}$.
- 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:

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

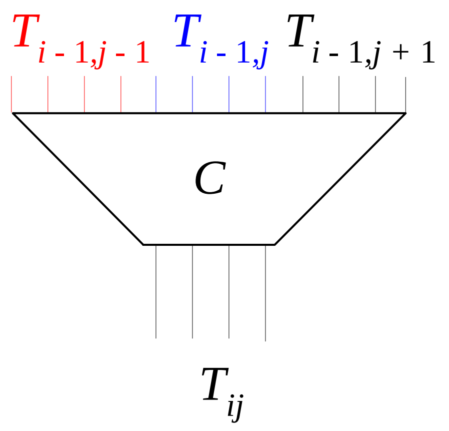
- So there are m boolean functions $F_1, F_2, \ldots, F_m$ with $3m$ inputs each such that for all $i, j > 0$,

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

The Proof (continued)

- These F_i's depend on only M's specification, not on x.
- Their sizes are fixed.
- 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.

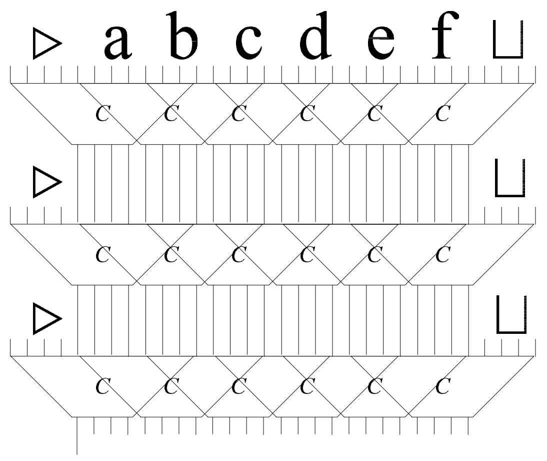
Circuit C



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 $R(x)$



$\triangleright$ a b c d e f $\sqcup$
C C C C C C
$\triangleright$ $\sqcup$
C C C C C C
$\triangleright$ $\sqcup$
C C C C C C

A Corollary

The construction in the above proof shows the following.

Corollary 30 *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 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 31 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 32 (Cook, 1971) SAT *is NP-complete.*

- SAT $\in$ NP (p. 80).
- CIRCUIT SAT reduces to SAT (p. 203).
- 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 $L \in$ 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

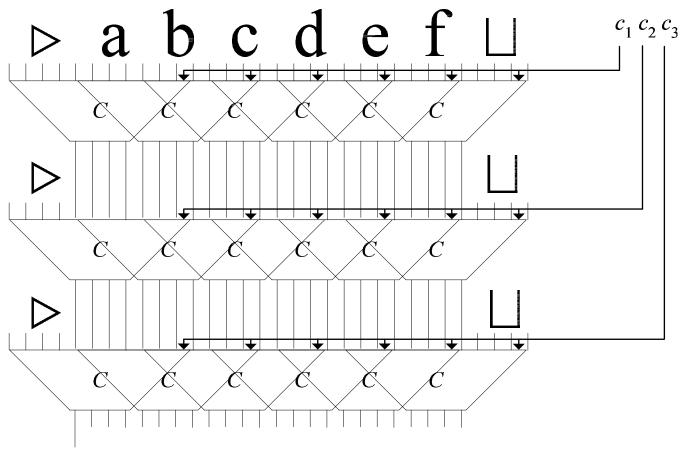
$$B = (c_0, c_1, \ldots, c_{|x|^k - 1}) \in \{0,1\}^{|x|^k}.$$

- 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. 228.
- Each circuit C at time i has an extra binary input c corresponding to the nondeterministic choice.

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



The Proof (concluded)

- The overall circuit $R(x)$ (on p. 235) 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. 232) 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 requirement for parsimonious reduction: It runs in deterministic polynomial time.

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 33 (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 $L \in \text{NP}$.
- 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.
- Clearly R is polynomially balanced because 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.

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.

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

- From Propositions 27 (p. 212) and Proposition 28 (p. 214), it is the least likely to be in P.
- Approximations.
- Special cases.
- Average performance.
- Randomized algorithms.
- Exponential-time algorithms that work well for small problems.
- "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. 232) and the reduction of CIRCUIT SAT to SAT (p. 203).
- 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.

Another Variant of 3SAT

Proposition 34 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.

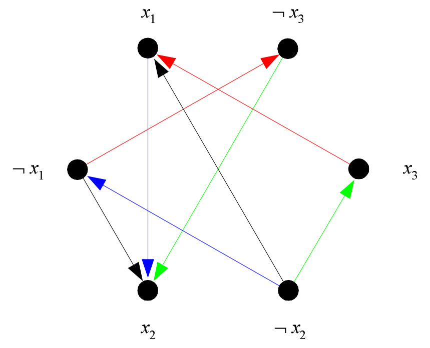
The Proof (concluded)

- Add $(\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \land \cdots \land (\neg x_k \lor x_1)$ to the expression.
  - This is logically equivalent to $x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_k \Rightarrow x_1$.
  - Each clause may have fewer than 3 literals.
- The resulting equivalent expression satisfies the condition for x.

2SAT and Graphs

- Let ϕ be an instance of 2SAT: Each clause has 2 literals.
- Define graph $G(\phi)$ as follows:
  - The nodes are the variables and their negations.
  - Add edges $(\neg\alpha, \beta)$ and $(\neg\beta, \alpha)$ to $G(\phi)$ if $\alpha \lor \beta$ is a clause in ϕ.
    - For example, if $x \lor \neg y \in \phi$, add $(\neg x, \neg y)$ and (y, x).
    - *Two* edges are added for each clause.
- Think of the edges as $\neg\alpha \Rightarrow \beta$ and $\neg\beta \Rightarrow \alpha$.
- b is reachable from a iff $\neg a$ is reachable from $\neg b$.
- Paths in $G(\phi)$ are valid implications.

Illustration: Directed Graph for $(x_1 \lor x_2) \land (x_1 \lor \neg x_3) \land (\neg x_1 \lor x_2) \land (x_2 \lor x_3)$



Properties of $G(\phi)$

Theorem 35 *ϕ is unsatisfiable if and only if there is a variable x such that there are paths from x to $\neg x$ and from $\neg x$ to x in $G(\phi)$.*

- Suppose such paths exist, but ϕ can be satisfied by a truth assignment T.
  - Without loss of generality, assume $T(x) = \texttt{true}$.
  - As there is a path from x to $\neg x$ and $T(\neg x) = \texttt{false}$, there must be an edge (α, β) on this path such that $T(\alpha) = \texttt{true}$ and $T(\beta) = \texttt{false}$.
  - Hence $(\neg\alpha \lor \beta)$ is a clause of ϕ.
  - But this clause is *not* satisfied by T, a contradiction.

The Proof (continued)

- Now suppose there is no variable with such paths in $G(\phi)$.
  - We shall construct a satisfying truth assignment.
  - It is enough that no edges go from true to false.
  - Pick any node α which has not had a truth value *and* there is no path from it to $\neg\alpha$ (always doable by assumption, why?).
  - Assign nodes reachable from α true and their negations false.
    - ∗ The negations are those nodes that can reach $\neg\alpha$.

The Proof (continued)

- (continued)
  - The above steps are well-defined.
    - ∗ If α could reach both β and $\neg\beta$, then there would be a path from $\neg\beta$ to $\neg\alpha$, hence a path from α to $\neg\alpha$!
    - ∗ If there were a path from α to a node y already assigned false, then $\neg y$ can reach $\neg\alpha$ and α had been assigned false before!
  - We keep picking such α's until we run out of them.

The Proof (concluded)

- (continued)
  - Can there be nodes α without a truth value because there is a path from α to $\neg\alpha$?
  - Well, every node must have had a truth value.
    - ∗ If α does not, then there is a path from α to $\neg\alpha$.
    - ∗ But then the algorithm could have picked $\neg\alpha$, assigning false to α!
  - The assignments make sure a false node never follows a true node.
  - Hence ϕ is satisfied by the assignments.

2SAT Is in NL $\subseteq$ P

- NL is a subset of P (p. 175).
- By Corollary 25 on p. 191, coNL equals NL.
- We need to show only that recognizing unsatisfiable expressions is in NL.
- In nondeterministic logarithmic space, we can test the conditions of Theorem 35 by guessing a variable x and testing if $\neg x$ is reachable from x *and* if $\neg x$ can reach x.
  - See the algorithm for REACHABILITY (p. 92).