

Reductions and Completeness

It is unworthy of excellent men
to lose hours like slaves in the labor of
computation.
— Gottfried Wilhelm von Leibniz (1646–1716)

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 input $R(x)$ of A.^a
 - The answer to x for B is the same as the answer to $R(x)$ for A.
 - R is easy to compute.
- We say problem A is at least as hard as problem B if B reduces to A.

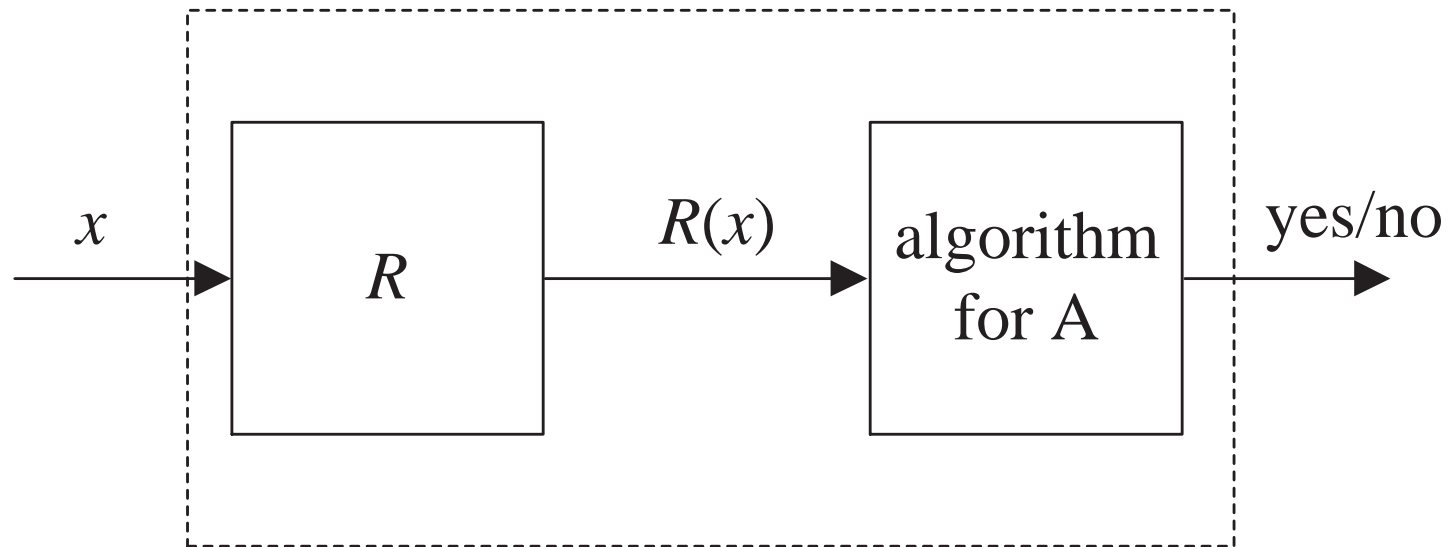
^aSee also p. 143.

Degrees of Difficulty (concluded)

- This makes intuitive sense: If A is able to solve your problem B after only a little bit of work of R , then A must be at least as hard.
 - If A is easy to solve, it combined with R (which is also easy) would make B easy to solve, too.^a
 - So if B is hard to solve, A must be hard (if not harder) as well.

^aThanks to a lively class discussion on October 13, 2009.

Reduction



Solving problem B by calling the algorithm for problem A *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 .^b
 - Some instances of A may never appear in the range of R .

^aContributed by Mr. Ming-Feng Tsai (D92922003) on October 29, 2003.

^b $R(x)$ may not be onto; Mr. Alexandr Simak (D98922040) on October 13, 2009.

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 .

Reduction between Languages (concluded)

- Note that by Theorem 23 (p. 214), R runs in polynomial time.
 - In most cases, a polynomial-time R suffices for proofs.
- Suppose R is a reduction from L_1 to L_2 .
- Then solving “ $R(x) \in L_2?$ ” is an algorithm for solving “ $x \in L_1?$ ”^a

^aBut it may not be an optimal one.

A Paradox?

- Degree of difficulty is not defined in terms of *absolute* complexity.
- So a language $B \in \text{TIME}(n^{99})$ may be “easier” than a language $A \in \text{TIME}(n^3)$.
 - This happens when B is reducible to A.
- But is this a contradiction if the best algorithm for B requires n^{99} steps?
- That is, how can a problem *requiring* n^{99} steps be reducible to a problem solvable in n^3 steps?

Paradox Resolved

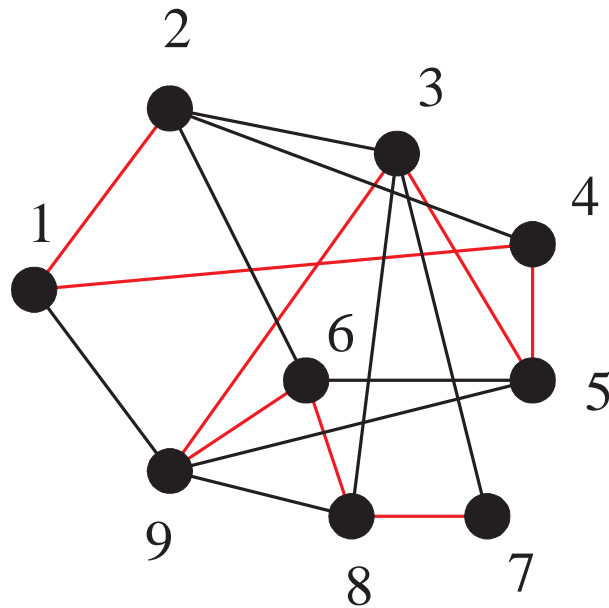
- The so-called contradiction does not hold.
- Suppose we solve the problem “ $x \in B?$ ” via “ $R(x) \in A?$ ”
- We must consider the time spent by $R(x)$ and its length $|R(x)|$ because $R(x)$ (not x) is presented to A .

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: $1, 2, \dots, n$.
- A Hamiltonian path can be expressed as a permutation π of $\{1, 2, \dots, n\}$ such that
 - $\pi(i) = j$ means the i th position is occupied by node j .
 - $(\pi(i), \pi(i + 1)) \in G$ for $i = 1, 2, \dots, n - 1$.
- 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 iff G has a Hamiltonian path.
- $R(G)$ has n^2 boolean variables x_{ij} , $1 \leq i, j \leq n$.
- x_{ij} means
“the i th position in the Hamiltonian path is occupied by node j .”



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

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.
 - $\neg x_{ki} \vee \neg x_{k+1,j}$ for all $(i, j) \notin G$ and $k = 1, 2, \dots, n - 1$.

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 $(\pi(1), \pi(2), \dots, \pi(n))$ is a Hamiltonian path.
- Conversely, suppose G has a Hamiltonian path

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

where π is a permutation.

- Clearly, the truth assignment

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

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 a 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 .

^aContributed 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 $g_{ij0} = \text{true}$ 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 $k = 1, 2, \dots, n$.
- g_{ijk} is an OR gate with predecessors $g_{i,j,k-1}$ and $h_{i,j,k}$, where $k = 1, 2, \dots, n$.
- 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.
 - $R(C)$ is a depth-2 circuit; furthermore, 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 $R(C)$ (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 $(\neg h \vee \neg h' \vee g)$.

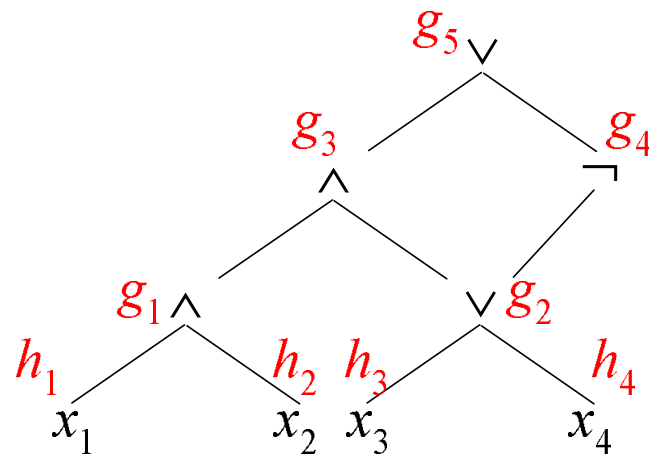
- Meaning: $g \Leftrightarrow (h \wedge h')$.

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

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

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

An Example



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

An Example (concluded)

- In general, the result is a CNF.
- The CNF has size proportional to the circuit's number of gates.
- The CNF adds new variables to the circuit's original input variables.
- Had we used the idea on p. 184 for the reduction, the resulting formula may have an exponential length because of the copying.^a

^aContributed by Mr. Ching-Hua Yu (D00921025) on October 16, 2012.

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 $R_{12} \circ R_{23}$ is a reduction from L_1 to L_3 .*

- So reducibility is transitive.

Completeness^a

- As reducibility is transitive, problems can be ordered with respect to their difficulty.
- Is there a *maximal* element?
- 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.
- Hence it may surprise you that most of the complexity classes that we have seen so far have maximal elements.

^aCook (1971) and Levin (1973).

Completeness (concluded)

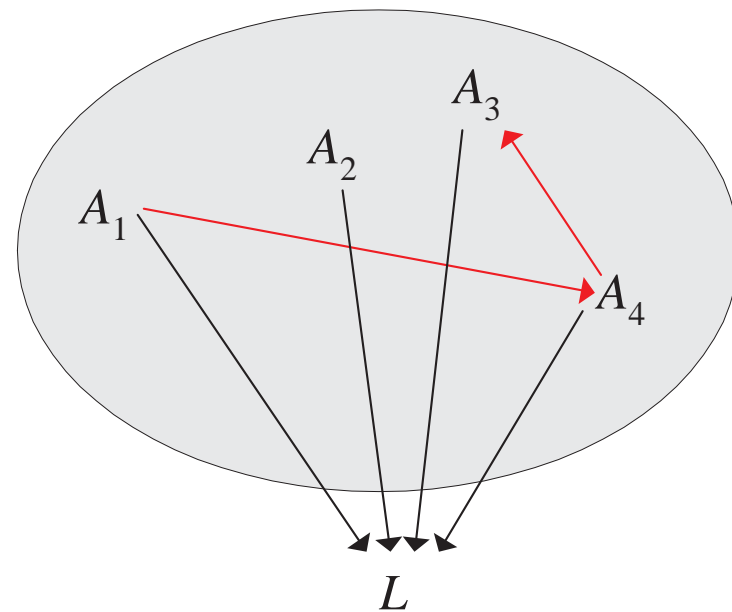
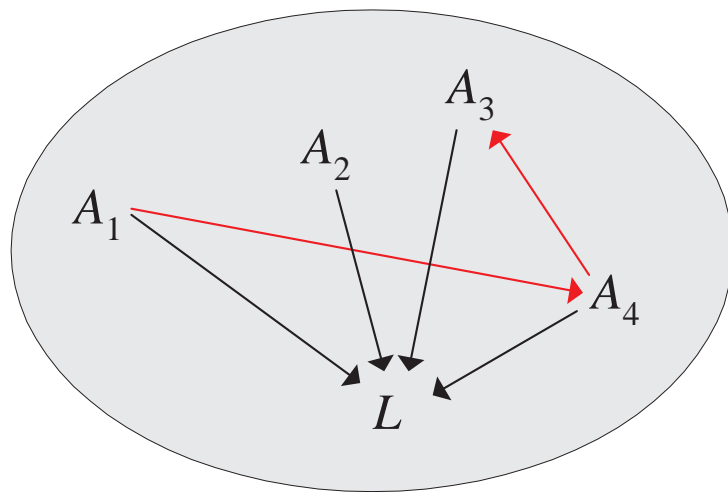
- Let \mathcal{C} be a complexity class and $L \in \mathcal{C}$.
- L is **\mathcal{C} -complete** if every $L' \in \mathcal{C}$ can be reduced to L .
 - Most complexity classes we have seen so far have complete problems!
- Complete problems capture the difficulty of a class because they are the hardest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

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

^aContributed 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.

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 \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 27 implies the following.

Corollary 28 *$P = NP$ if and only if an NP-complete problem is in P .*

Corollary 29 *$L = P$ if and only if a P-complete problem is in L .*

Complete Problems and Complexity Classes

Proposition 30 *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.
- 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 j th 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)

- Suppose 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 position j .

Comments

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

$$\begin{array}{c} |x|^k \\ \hline \triangleright 0_s 10001 \square \square \cdots \square \end{array}$$

- A typical row may look like

$$\begin{array}{c} |x|^k \\ \hline \triangleright 10100_q 01110100 \square \square \cdots \square \end{array}$$

Comments (concluded)

- The last rows must look like

$$\underbrace{\triangleright \dots \text{“yes”} \dots \square}_{|x|^k} \quad \text{or} \quad \underbrace{\triangleright \dots \text{“no”} \dots \square}_{|x|^k}$$

- Three out of the table's 4 borders are known:

$$\begin{array}{cccccc} \triangleright & \mathbf{a} & \mathbf{b} & \mathbf{c} & \mathbf{d} & \mathbf{e} & \mathbf{f} & \square \\ \triangleright & & & & & & & \square \\ \triangleright & & & & & & & \square \\ \triangleright & & & & & & & \square \\ \triangleright & & & & & & & \square \\ & & & & \vdots & & & \square \end{array}$$

A P-Complete Problem

Theorem 31 (Ladner (1975)) CIRCUI T VALUE *is P-complete.*

- It is easy to see that CIRCUI T VALUE $\in P$.
- For *any* $L \in P$, we will construct a reduction R from L to CIRCUI T 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 j th symbol of x or \sqcup , a \triangleright , and a \sqcup , respectively.
 - Recall that three out of T 's 4 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^a

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

^aCalled **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_{ijl} , where

$$0 \leq i \leq n^k - 1,$$

$$0 \leq j \leq n^k - 1,$$

$$1 \leq l \leq m.$$

The Proof (continued)

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

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

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

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

The Proof (continued)

- There is a boolean function F_ℓ with $3m$ inputs such that

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

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

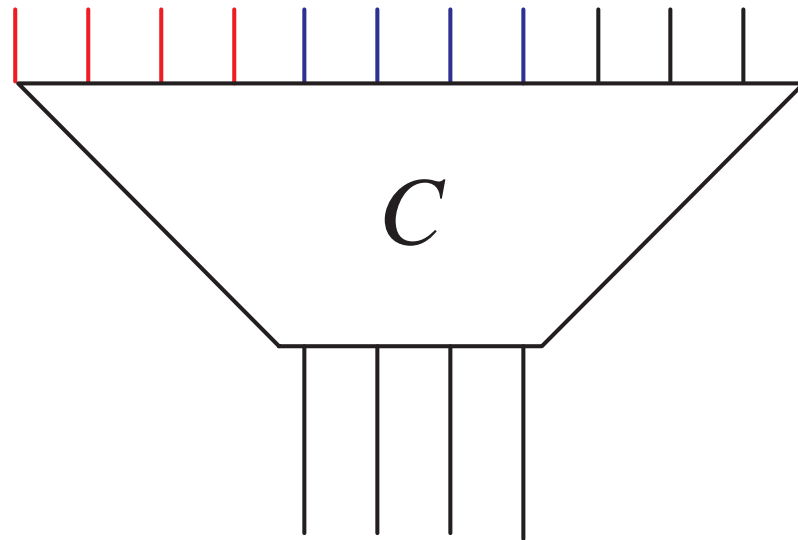
The Proof (continued)

- These F_i 's depend only on M 's specification, not on x .
- Their sizes are constant.
- These boolean functions can be turned into boolean circuits (see p. 183).
- Compose these m circuits in parallel to obtain circuit C with $3m$ -bit inputs and m -bit outputs.
 - Schematically, $C(T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}) = T_{ij}$.^a

^a C is like an ASIC (application-specific IC) chip.

Circuit C

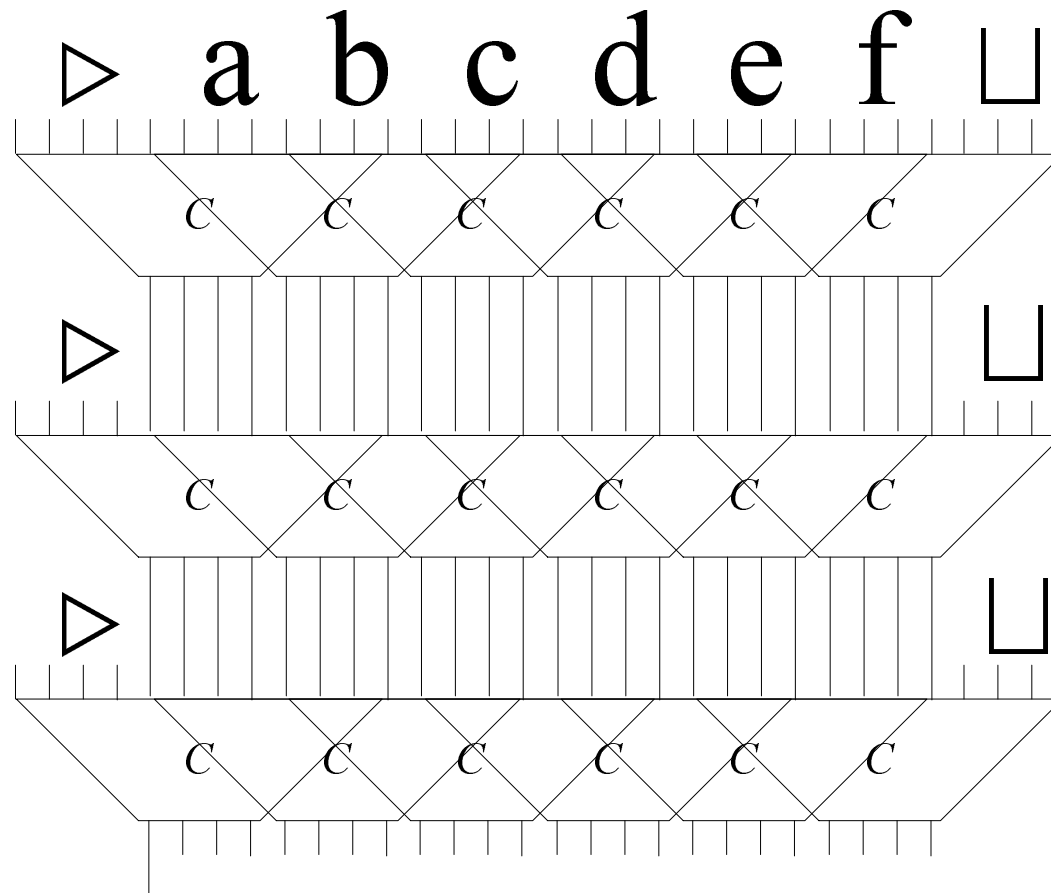
$T_{i-1,j-1}$ $T_{i-1,j}$ $T_{i-1,j+1}$



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 32 *If $L \in TIME(T(n))$, then a circuit with $O(T^2(n))$ gates can decide if $x \in L$ for $|x| = n$.*