Corollary 26  Let $f(n) \geq \log n$ be proper. Then

$$\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n)).$$

- Apply Savitch’s proof to the configuration graph of the NTM on its input.
- From p. 240, the configuration graph has $O(c^f(n))$ nodes; hence each node takes space $O(f(n))$.
- But if we construct explicitly the whole graph before applying Savitch’s theorem, we get $O(c^f(n))$ space!
The Proof (continued)

• The way out is not to generate the graph at all.

• Instead, keep the graph implicit.

• We checked node connectedness only when $i = 0$ on p. 250, by examining the input graph $G$.

• Suppose we are given configurations $x$ and $y$.

• Then we go over the Turing machine’s program to determine if there is an instruction that can turn $x$ into $y$ in one step.\textsuperscript{a}

• So connectivity is checked locally and on demand.

\textsuperscript{a}Thanks to a lively class discussion on October 15, 2003.
The Proof (continued)

• The \( z \) variable in the algorithm on p. 250 simply runs through all possible valid configurations.
  
  – Let \( z = 0, 1, \ldots, O(c^f(n)) \).
  
  – Make sure \( z \) is a valid configuration before proceeding with it.\(^a\)

   * Adopt the same width for each symbol and state of the NTM and for the cursor position on the input string.\(^b\)

  – If it is not, advance to the next \( z \).

\(^a\)Thanks to a lively class discussion on October 13, 2004.

\(^b\)Contributed by Mr. Jia-Ming Zheng (R04922024) on October 17, 2017.
The Proof (concluded)

- Each $z$ has length $O(f(n))$.
- So each node needs space $O(f(n))$.
- The depth of the recursive call on p. 250 is $O(\log c^{f(n)})$, which is $O(f(n))$.
- The total space is therefore $O(f^2(n))$. 
Implications of Savitch’s Theorem

**Corollary 27** \( PSPACE = NPSPACE \).

- Nondeterminism is less powerful with respect to space.
- Nondeterminism may be very powerful with respect to time as it is not known if \( P = NP \).
Nondeterministic Space Is Closed under Complement

- Closure under complement is trivially true for deterministic complexity classes (p. 225).

- It is known that\(^a\)

\[
\text{coNSPACE}(f(n)) = \text{NSPACE}(f(n)). \tag{3}
\]

- So

\[
\text{coNL} = \text{NL}.
\]

- But it is not known whether coNP = NP.

\(^a\)Szelepscényi (1987); Immerman (1988).
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)

I thought perhaps you might be members of
that lowly section of the university
known as the Sheffield Scientific School.
F. Scott Fitzgerald (1920), “May Day”
Degrees of Difficulty

• When is a problem more difficult than another?

• B reduces to A if:
  – There is a transformation $R$ which for every problem instance $x$ of B yields a problem instance $R(x)$ of A.\(^a\)
  – The answer to “$R(x) \in A$?” is the same as the answer to “$x \in B$?”
  – $R$ is easy to compute.

• We say problem A is at least as hard as\(^b\) problem B if B reduces to A.

\(^a\)See also p. 146.
\(^b\)Or simply “harder than” for brevity.
Solving problem B by calling the algorithm for problem A once and without further processing its answer.\textsuperscript{a}

\textsuperscript{a}More general reductions are possible, such as the Turing (1939) reduction and the Cook (1971) reduction.
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), too!

\(^a\)Thanks to a lively class discussion on October 13, 2009.
Comments\textsuperscript{a}

• Suppose B reduces to A via a transformation $R$.\textsuperscript{b}

• 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.\textsuperscript{c}
  
  – Some instances of A may never appear in $R$’s range.

• But $x$ must be an \textit{arbitrary} instance for B.

\textsuperscript{a}Contributed by Mr. Ming-Feng Tsai (D92922003) on October 29, 2003.

\textsuperscript{b}Sometimes, we say “B can be reduced to A.”

\textsuperscript{c}$R(x)$ may not be onto; Mr. Alexandr Simak (D98922040) on October 13, 2009.
Is “Reduction” a Confusing Choice of Word?\textsuperscript{a}

• If B reduces to A, doesn’t that intuitively make A smaller and simpler?

• But our definition means just the opposite.

• Our definition says in this case B is a special case of A.\textsuperscript{b}

• Hence A is harder.

\textsuperscript{a}Moore & Mertens (2011).
\textsuperscript{b}See also p. 149.
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 24 (p. 237), $R$ runs in polynomial time.
  - In most cases, a polynomial-time $R$ suffices for proofs.a

- Suppose $R$ is a reduction from $L_1$ to $L_2$.

- Then solving “$R(x) \in L_2$?” is an algorithm for solving “$x \in L_1$?”b

---

aIn fact, unless stated otherwise, we will only require that the reduction $R$ run in polynomial time. It is often called a **polynomial-time many-one reduction**.

bOf course, it may not be the most efficient.
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 $A \in \text{TIME}(n^3)$ if $B$ reduces to $A$.

- But isn’t 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 is the result of flawed logic.
• 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 solved by $A$. 
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: $1, 2, \ldots, n$.

- A Hamiltonian path can be expressed as a permutation $\pi$ of $\{1, 2, \ldots, n\}$ such that
  - $\pi(i) = j$ means the $i$th position is occupied by node $j$.
  - $(\pi(i), \pi(i + 1)) \in G$ for $i = 1, 2, \ldots, n - 1$. 
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 $R(G)$ 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 $i$th position in the Hamiltonian path is occupied by node $j$.

• Our reduction will produce clauses.

\(^a\)Remember that $R$ does not have to be onto.
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.
   - $x_{1j} \lor x_{2j} \lor \cdots \lor x_{nj}$ for each $j$.

2. No node $j$ appears twice in the path.
   - $\neg x_{ij} \lor \neg x_{kj} (\equiv \neg (x_{ij} \land x_{kj}))$ 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.
   - $\neg x_{ij} \lor \neg x_{ik} (\equiv \neg (x_{ij} \land x_{ik}))$ for all $i, j, k$ with $j \neq k$.

5. Nonadjacent nodes $i$ and $j$ cannot be adjacent in the path.
   - $\neg x_{ki} \lor \neg x_{k+1,j} (\equiv \neg (x_{k,i} \land x_{k+1,j}))$ 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 $\pi$ 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 $\pi$ 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

• 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 $G$.

—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 $1 \leq i, j \leq n$ 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 $k = 1, 2, \ldots, n$.

- $g_{ijk}$ 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.
  – $R(C)$ is basically 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 ($\neg g \lor x$) and ($g \lor \neg x$).
- Meaning: $g \iff x$.

$g$ 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: $g \iff \neg h$. 
The Clauses of $R(C')$ (continued)

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

- The conjunction of the above clauses is equivalent to

$$[g \Rightarrow (h \lor h')] \land [(h \lor h') \Rightarrow g]$$

$$\equiv g \iff (h \lor h').$$

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

- It is equivalent to

$$g \iff (h \land h').$$
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 $R(C)$.
(h₁ ⇔ x₁) ∧ (h₂ ⇔ x₂) ∧ (h₃ ⇔ x₃) ∧ (h₄ ⇔ x₄)
∧ [g₁ ⇔ (h₁ ∧ h₂)] ∧ [g₂ ⇔ (h₃ ∨ h₄)]
∧ [g₃ ⇔ (g₁ ∧ g₂)] ∧ (g₄ ⇔ ¬g₂)
∧ [g₅ ⇔ (g₃ ∨ g₄)] ∧ g₅.
An Example (concluded)

- 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. 207 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.
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\textsuperscript{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!

\textsuperscript{a}Post (1944); Cook (1971); Levin (1973).
Completeness (concluded)

• Let $C$ be a complexity class and $L \in C$.
• $L$ is $C$-complete if every $L' \in C$ 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.$^a$

\[a\text{See also p. 161.}\]
Hardness

- Let \( \mathcal{C} \) be a complexity class.

- \( L \) is \( \mathcal{C} \)-\textbf{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 $C$ is **closed under reductions** if whenever $L$ is reducible to $L'$ and $L' \in C$, then $L \in C$.

- 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 $C'$ and $C$ be two complexity classes such that $C' \subseteq C$. Assume $C'$ is closed under reductions and $L$ is $C$-complete. Then $C = C'$ if and only if $L \in C'$.

• Suppose $L \in C'$ first.

• Every language $A \in C$ reduces to $L \in C'$.

• Because $C'$ is closed under reductions, $A \in C'$.

• Hence $C \subseteq C'$.

• As $C' \subseteq C$, we conclude that $C = C'$. 
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 $C'$ and $C$ be two complexity classes closed under reductions. If $L$ is complete for both $C$ and $C'$, then $C = C'$.

- All languages $A \in C$ reduce to $L \in C$ and $L \in C'$.
- Since $C'$ is closed under reductions, $A \in C'$.
- Hence $C \subseteq C'$.
- The proof for $C' \subseteq C$ is symmetric.
Complete Problems and Complexity Classes, Again (concluded)

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

- Every language $A \in C$ reduces to $L$.
- By Proposition 28 (p. 287), $A$ 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 $\_|s$ 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 $\sigma$, then the $(i, j)$th entry is a new symbol $\sigma_q$.

\(^a\)|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 \( \sigma_q \).
- 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)

- $M$ 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

\[
\begin{array}{c}
| x |^k \\
\Downarrow 0 \underbrace{10001}_{q} \underbrace{1001}_{\cdots} \\
\end{array}
\]

• A typical row looks like

\[
\begin{array}{c}
| x |^k \\
\Downarrow 10100_0 \underbrace{01110100}_{q} \underbrace{1001}_{\cdots} \\
\end{array}
\]
Comments (concluded)

- The last rows must look like

\[
\begin{array}{c}
| x |^k \\
\triangleright \cdots \text{“yes”} \cdots \square \\
\end{array}
\quad \text{or} \quad 
\begin{array}{c}
| x |^k \\
\triangleright \cdots \text{“no”} \cdots \square \\
\end{array}
\]

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

\[
\begin{array}{cccccccc}
\triangleright & a & b & c & d & e & f & \square \\
\triangleright & & & & & & & \\
\triangleright & & & & & & & \\
\triangleright & & & & & & & \\
\triangleright & & & & & & & \\
\begin{array}{c}
\vdots \\
\end{array}
\end{array}
\]
A P-Complete Problem

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

- It is easy to see that CIRCUIT VALUE ∈ 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 4 borders are known.

• So when $i = 0$, or $j = 0$, or $j = |x|^k - 1$, the value of $T_{ij}$ is known.
  – The $j$th symbol of $x$ or $\sqcup$, a $\triangleright$, or a $\sqcap$, respectively.

• Consider other entries $T_{ij}$.
The Proof (continued)

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

\[
\begin{array}{ccc}
T_{i-1,j-1} & T_{i-1,j} & T_{i-1,j+1} \\
T_{ij} \\
\end{array}
\]

- $T_{ij}$ does not depend on any other entries!

- $T_{ij}$ does not depend on $i$, $j$, or $x$ 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 $\Gamma$ 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 $\Gamma$ as an $m$-bit number,\(^a\) where

$$m = \lceil \log_2 |\Gamma| \rceil.$$\(^a\)

\(^a\)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

$$
0 \leq i \leq n^k - 1,
0 \leq j \leq n^k - 1,
1 \leq \ell \leq m.
$$
The Proof (continued)

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

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

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

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

• So truth values for the $3m$ bits determine $S_{ij\ell}$. 
The Proof (continued)

- This means there is a boolean function $F_\ell$ with $3m$ inputs such that

$$S_{ij\ell} = F_\ell(S_{i-1,j-1,1}, S_{i-1,j-1,2}, \ldots, S_{i-1,j-1,m},$$

$$S_{i-1,j,1}, S_{i-1,j,2}, \ldots, S_{i-1,j,m},$$

$$T_{i-1,j-1}, S_{i-1,j+1,1}, S_{i-1,j+1,2}, \ldots, S_{i-1,j+1,m})$$

for all $i, j > 0$ and $1 \leq \ell \leq m.$
The Proof (continued)

- These $F_\ell$’s 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 (see p. 206).
- Compose these $m$ circuits in parallel to obtain circuit $C$ with $3m$-bit inputs and $m$-bit outputs.
  
  - Schematically, $C(T_{i-1,j-1},T_{i-1,j},T_{i-1,j+1}) = T_{ij}$.\(^b\)

\(^a\)It means independence of the input $x$.
\(^b\)C is like an ASIC (application-specific IC) chip.
Circuit $C$

\[ T_{i-1,j-1} \quad T_{i-1,j} \quad T_{i-1,j+1} \]

\[ C \]

\[ T_{ij} \]
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 \( L \in \text{TIME}(T(n)) \), then a circuit with \( O(T^2(n)) \) gates can decide \( L \).
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 ¬ 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.*

- Given any general circuit, “move the ¬’s downwards” using de Morgan’s laws\(^a\) to yield a monotone circuit with the same output.

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

\(^a\)How? Need to make sure no exponential blowup.
MAXIMUM FLOW Is P-Complete

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