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 $G$.

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 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 \((\neg g \lor x)\) 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: \( g \leftrightarrow \neg h \).

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

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

\( g \) is the output gate: Add clause \((g)\).
- Meaning: \( g \) must be true to make \( R(C) \) true.

\[ \]

Composition of Reductions

**Proposition 24** 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 $i$th bit, $R_{12}(x)$ will be simulated until the $i$th 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 $i$th output bit of $R_{12}(x)$ is well-defined because once it is written, it will never be overwritten.

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.

Completeness

- 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.

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 $L$.

\[\text{Contributed by Mr. Ming-Feng Tsai (D92922003) on October 15, 2003.}\]
Closedness under Reduction

- A class $\mathcal{C}$ is **closed under reductions** if whenever $L$ is reducible to $L'$ and $L' \in \mathcal{C}$, then $L \in \mathcal{C}$.
- $\text{P, NP, coNP, L, NL, PSPACE, and EXP}$ are all closed under reductions.

Complete Problems and Complexity Classes

**Proposition 25** 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 $\mathcal{C} = \mathcal{C}'$.

On the other hand, suppose $\mathcal{C} = \mathcal{C}'$.

- As $L$ is $\mathcal{C}$-complete, $L \in \mathcal{C}$.
- Thus, trivially, $L \in \mathcal{C}'$. 

The Proof (concluded)
Two Immediate Corollaries
Proposition 25 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 26 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 $L \in C$ reduce to $L \in C'$.
• Since $C'$ is closed under reductions, $L \in C'$.
• Hence $C \subseteq C'$.
• The proof for $C' \subseteq C$ 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 $\_\_$ 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 $\sigma_q$. 
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 $\triangleright_q$.

Comments

- Each row is essentially a configuration.
- If the input $x = 010001$, then the first row is
  \[
  \begin{array}{c}
  |x|^k \\
  \triangleright 010001 \\
  \end{array}
  \]
- A typical row may be
  \[
  \begin{array}{c}
  |x|^k \\
  \triangleright 10100_{q,0}1110100 \\
  \end{array}
  \]
- The last rows must look like $\triangleright \cdots \text{"yes"} \cdots$

A P-Complete Problem

Theorem 27 (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 $j$th symbol of $x$ or $\bigcup$, a $\triangleright$, and a $\bigcup$, respectively.
  - Three out of four of $T$’s borders are known.

> a b c d e f □

• Consider other entries $T_{ij}$.

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

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

• 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, 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

\[
0 \leq i \leq n^k - 1, \\
0 \leq j \leq n^k - 1, \\
1 \leq \ell \leq m.
\]
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.

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 $(|x|^k - 1, 1)$. 

Circuit $C$

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

\[
C
\]

\[
T_{ij}
\]

The Computation Tableau and $R(x)$

\[
\begin{array}{cccccccc}
\uparrow & a & b & c & d & e & f & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]

\[
\begin{array}{cccccccc}
\uparrow & c & c & c & c & c & c & \downarrow \\
\hline
\hline
\hline
\end{array}
\]
A Corollary

The construction in the above proof shows the following.

**Corollary 28** If $L \in \text{TIME}(T(n))$, then a circuit with $O(T^2(n))$ gates can decide if $x \in L$ for $|x| = n$.

---

**MONOTONE CIRCUIT VALUE is P-Complete**

Despite their limitations, MONOTONE CIRCUIT VALUE is as hard as CIRCUIT VALUE.

**Corollary 29** MONOTONE CIRCUIT VALUE is P-complete.

- Given any general circuit, we can “move the ¬’s downwards” using de Morgan’s laws. (Think!)

---

**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 ¬ gates.
- MONOTONE CIRCUIT VALUE is CIRCUIT VALUE applied to monotone circuits.

---

**Cook’s Theorem: the First NP-Complete Problem**

**Theorem 30 (Cook (1971))** SAT is NP-complete.

- SAT $\in$ NP (p. 84).
- CIRCUIT SAT reduces to SAT (p. 213).
- 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 $L \in \text{NP}$ 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 $B = (c_1, c_2, \ldots, c_{|x|^{k-1}}) \in \{0,1\}^{|x|^k}$.
• Once $B$ is fixed, the computation is deterministic.

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

The Proof (concluded)

• Each choice of $B$ results in a deterministic polynomial-time computation, hence a table like the one on p. 241.
• 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 overall circuit $R(x)$ (on p. 248) 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. 245) 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.

Wir müssen wissen, wir werden wissen.
(We must know, we shall know.)
— David Hilbert (1900)

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 $k \geq 1$. 

NP-Complete Problems
An Alternative Characterization of NP

Proposition 31 (Edmonds (1965)) Let $L \subseteq \Sigma^*$ be a language. Then $L \in \text{NP}$ 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 $L = \{x : (x, y) \in R$ 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 25 (p. 224) and Proposition 26 (p. 227), it is the least likely to be in P.
• Your options are:
  – Approximations.
  – Special cases.
  – Average performance.
  – Randomized algorithms.
  – Exponential-time algorithms that work well in practice.
  – “Heuristics” (and pray).