The Quantified Halting Problem

- Let $f(n) \geq n$ be proper.
- Define

$$H_f \triangleq \{ M; x : M \text{ accepts input } x \text{ after at most } f(|x|) \text{ steps } \},$$

where $M$ is deterministic.
- Assume the input is binary.
\[ H_f \in \text{TIME}(f(n)^3) \]

- For each input \( M; x \), we simulate \( M \) on \( x \) with an alarm clock of length \( f(|x|) \).
  - Use the single-string simulator (p. 82), the universal TM (p. 133), and the linear speedup theorem (p. 92).
  - Our simulator accepts \( M; x \) if and only if \( M \) accepts \( x \) before the alarm clock runs out.

- From p. 89, the total running time is \( O(\ell_M k_M^2 f(n)^2) \), where \( \ell_M \) is the length to encode each symbol or state of \( M \) and \( k_M \) is \( M \)’s number of strings.

- As \( \ell_M k_M^2 = O(n) \), the running time is \( O(f(n)^3) \), where the constant is independent of \( M \).
\[ H_f \notin \text{TIME}(f([n/2])) \]

- Suppose TM \( M_{H_f} \) decides \( H_f \) in time \( f([n/2]) \).
- Consider machine:

\[
D_f(M) \quad \{
\begin{align*}
\quad & \text{if } M_{H_f}(M; M) = \text{“yes”} \\
\quad & \text{then } \text{“no”}; \\
\quad & \text{else } \text{“yes”};
\end{align*}
\}
\]

- \( D_f \) on input \( M \) runs in the same time as \( M_{H_f} \) on input \( M; M \), i.e., in time \( f(\lfloor \frac{2n+1}{2} \rfloor) = f(n) \), where \( n = |M| \).\(^a\)

\(^a\)A student pointed out on October 6, 2004, that this estimation forgets to include the time to write down \( M; M \).
The Proof (concluded)

• First,

\[
D_f(D_f) = \text{“yes”}
\]

\[
\Rightarrow D_f; D_f \notin H_f
\]

\[
\Rightarrow D_f \text{ does not accept } D_f \text{ within time } f(|D_f|)
\]

\[
\Rightarrow D_f(D_f) \neq \text{“yes” as } D_f(D_f) \text{ runs in time } f(|D_f|),
\]

a contradiction

• Similarly, \( D_f(D_f) = \text{“no”} \Rightarrow D_f(D_f) = \text{“yes.”} \)
The Time Hierarchy Theorem

**Theorem 18**  If \( f(n) \geq n \) is proper, then

\[
\text{TIME}(f(n)) \subsetneq \text{TIME}(f(2n + 1)^3).
\]

- The quantified halting problem makes it so.

**Corollary 19**  \( P \subsetneq E \).

- \( P \subseteq \text{TIME}(2^n) \) because \( \text{poly}(n) \leq 2^n \) for \( n \) large enough.
- But by Theorem 18,

\[
\text{TIME}(2^n) \subsetneq \text{TIME} \left( (2^{2n+1})^3 \right) \subseteq E.
\]
- So \( P \subsetneq E \).
The Space Hierarchy Theorem

Theorem 20 (Hennie & Stearns, 1966) If $f(n)$ is proper, then

$$\text{SPACE}(f(n)) \subsetneq \text{SPACE}(f(n) \log f(n)).$$

Corollary 21 $L \subsetneq \text{PSPACE}$. 
Nondeterministic Time Hierarchy Theorems

Theorem 22 (Cook, 1973) $\text{NTIME}(n^r) \subsetneq \text{NTIME}(n^s)$ whenever $1 \leq r < s$.

Theorem 23 (Seiferas, Fischer, & Meyer, 1978) If $T_1(n), T_2(n)$ are proper, then

$$\text{NTIME}(T_1(n)) \subsetneq \text{NTIME}(T_2(n))$$

whenever $T_1(n + 1) = o(T_2(n))$. 
The Reachability Method

• The computation of a time-bounded TM can be represented by a directed graph.

• The TM’s configurations constitute the nodes.

• There is a directed edge from node $x$ to node $y$ if $x$ yields $y$ in one step.

• The start node representing the initial configuration has zero in-degree.
The Reachability Method (concluded)

- When the TM is nondeterministic, a node may have an out-degree greater than one.
  - The graph is the same as the computation tree earlier.
  - But identical configurations are merged into one node.

- So $M$ accepts the input if and only if there is a path from the start node to a node with a “yes” state.

- It is the reachability problem.
Illustration of the Reachability Method

Initial configuration
Relations between Complexity Classes

**Theorem 24** Suppose $f(n)$ is proper. Then

1. $\text{SPACE}(f(n)) \subseteq \text{NSPACE}(f(n))$, $\text{TIME}(f(n)) \subseteq \text{NTIME}(f(n))$.
2. $\text{NTIME}(f(n)) \subseteq \text{SPACE}(f(n))$.
3. $\text{NSPACE}(f(n)) \subseteq \text{TIME}(k^{\log n} + f(n))$.

- Proof of 2:
  - Explore the computation tree of the NTM for “yes.”
  - Specifically, generate an $f(n)$-bit sequence denoting the nondeterministic choices over $f(n)$ steps.
Proof of Theorem 24(2)

• (continued)
  – Simulate the NTM based on the choices.
  – Recycle the space and repeat the above steps.
  – Halt with “yes” when a “yes” is encountered or “no” if the tree is exhausted.
  – Each path simulation consumes at most $O(f(n))$ space because it takes $O(f(n))$ time.
  – The total space is $O(f(n))$ because space is recycled.
Proof of Theorem 24(3)

• Let $k$-string NTM

$$M = (K, \Sigma, \Delta, s)$$

with input and output decide $L \in \text{NSPACE}(f(n))$.

• Use the reachability method on the configuration graph of $M$ on input $x$ of length $n$.

• A configuration is a $(2k + 1)$-tuple

$$(q, w_1, u_1, w_2, u_2, \ldots, w_k, u_k).$$
Proof of Theorem 24(3) (continued)

• We only care about 

\[(q, i, w_2, u_2, \ldots, w_{k-1}, u_{k-1}),\]

where \(i\) is an integer between 0 and \(n\) for the position of the first cursor.

• The number of configurations is therefore at most

\[|K| \times (n + 1) \times |\Sigma|^{2(k-2)f(n)} = O(c_1^{\log n + f(n)})\]  \hspace{1cm} (2)

for some \(c_1 > 1\), which depends on \(M\).

• Add edges to the configuration graph based on \(M\)’s transition function.
Proof of Theorem 24(3) (concluded)

- $x \in L \iff$ there is a path in the configuration graph from the initial configuration to a configuration of the form ("yes", $i, \ldots$).\(^a\)

- This is reachability on a graph with $O(c_1^{\log n + f(n)})$ nodes.

- It is in $\text{TIME}(c^{\log n + f(n)})$ for some $c > 1$ because reachability $\in \text{TIME}(n^j)$ for some $j$ and

$$\left[c_1^{\log n + f(n)}\right]^j = (c_1^j)^{\log n + f(n)}.$$

\(^a\)There may be many of them.
Space-Bounded Computation and Proper Functions

- In the definition of *space-bounded* computations earlier (p. 108), the TMs are not required to halt at all.

- When the space is bounded by a proper function \( f \), computations can be assumed to halt:
  - Run the TM associated with \( f \) to produce a quasi-blank output of length \( f(n) \) first.
  - The space-bounded computation must repeat a configuration if it runs for more than \( c \log n + f(n) \) steps for some \( c > 1 \).

\(^{a}\)See Eq. (2) on p. 238.
Space-Bounded Computation and Proper Functions (concluded)

• (continued)
  – So an infinite loop occurs during simulation for a computation path longer than $c \log n + f(n)$ steps.
  – Hence we only simulate up to $c \log n + f(n)$ time steps per computation path.
A Grand Chain of Inclusions

- It is an easy application of Theorem 24 (p. 235) that
  \[ L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXP. \]
- By Corollary 21 (p. 230), we know \( L \subset PSPACE. \)
- So the chain must break somewhere between \( L \) and \( EXP. \)
- It is suspected that all four inclusions are proper.
- But there are no proofs yet.

\(^a\)With input from Mr. Chin-Luei Chang (R93922004, D95922007) on October 22, 2004.
What Is Wrong with the Proof?\textsuperscript{a}

- By Theorem 24(2) (p. 235),
  \[
  \text{NL} \subseteq \text{TIME} \left( k^{O(\log n)} \right) \subseteq \text{TIME} \left( n^{c_1} \right)
  \]
  for some \( c_1 > 0 \).

- By Theorem 18 (p. 229),
  \[
  \text{TIME} \left( n^{c_1} \right) \subset \text{TIME} \left( n^{c_2} \right) \subseteq \text{P}
  \]
  for some \( c_2 > c_1 \).

- So
  \[
  \text{NL} \neq \text{P}.
  \]

\textsuperscript{a}Contributed by Mr. Yuan-Fu Shao (R02922083) on November 11, 2014.
What Is Wrong with the Proof? (concluded)

• Recall from p. 220 that $\text{TIME}(k^{O(\log n)})$ is a shorthand for

$$\bigcup_{j>0} \text{TIME}(j^{O(\log n)}).$$

• So the correct proof runs more like

$$\text{NL} \subseteq \bigcup_{j>0} \text{TIME}(j^{O(\log n)}) \subseteq \bigcup_{c>0} \text{TIME}(n^c) = P.$$  

• And

$$\text{NL} \neq P$$

no longer follows.
Nondeterministic and Deterministic Space

- By Theorem 6 (p. 114),

\[ \text{NTIME}(f(n)) \subseteq \text{TIME}(c^{f(n)}) , \]

an exponential gap.

- There is no proof yet that the exponential gap is inherent.

- How about NSPACE vs. SPACE?

- Surprisingly, the relation is only quadratic—a polynomial—by Savitch’s theorem.
Savitch’s Theorem

Theorem 25 (Savitch, 1970)

Reachability is in $\text{SPACE}(\log^2 n)$.

- Let $G(V, E)$ be a graph with $n$ nodes.
- For $i \geq 0$, let $\text{PATH}(x, y, i)$ mean there is a path from node $x$ to node $y$ of length at most $2^i$.
- There is a path from $x$ to $y$ if and only if $\text{PATH}(x, y, \lceil \log n \rceil)$ holds.
The Proof (continued)

- For $i > 0$, $\text{PATH}(x, y, i)$ if and only if there exists a $z$ such that $\text{PATH}(x, z, i - 1)$ and $\text{PATH}(z, y, i - 1)$.

- For $\text{PATH}(x, y, 0)$, check the input graph or if $x = y$.

- Compute $\text{PATH}(x, y, \lceil \log n \rceil)$ with a depth-first search on a graph with nodes $(x, y, z, i)$s (see next page).

- Like stacks in recursive calls, we keep only the current path of $(x, y, i)$s.

- The space requirement is proportional to the depth of the tree ($\lceil \log n \rceil$) times the size of the items stored at each node.

---

*Contributed by Mr. Chuan-Yao Tan on October 11, 2011.*
The Proof (continued): Algorithm for PATH\((x, y, i)\)

1: if \(i = 0\) then

2: \hspace{1em} if \(x = y\) or \((x, y) \in E\) then

3: \hspace{2em} return true;

4: \hspace{1em} else

5: \hspace{2em} return false;

6: \hspace{1em} end if

7: else

8: \hspace{1em} for \(z = 1, 2, \ldots, n\) do

9: \hspace{2em} if PATH\((x, z, i - 1)\) and PATH\((z, y, i - 1)\) then

10: \hspace{3em} return true;

11: \hspace{2em} end if

12: \hspace{1em} end for

13: \hspace{1em} return false;

14: end if
The Proof (continued)

\[ \text{PATH}(x, y, \log n) \]
\[ \text{PATH}(x, z, \log n-1) \]
\[ \text{PATH}(z, y, \log n-1) \]

"yes"  "no"  "no"
The Proof (concluded)

• Depth is \([\log n]\), and each node \((x, y, z, i)\) needs space \(O(\log n)\).

• The total space is \(O(\log^2 n)\).
The Relation between Nondeterministic and Deterministic Space Is Only Quadratic

**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. 238, the configuration graph has $O(cf(n))$ nodes; hence each node takes space $O(f(n))$.
- But if we construct *explicitly* the whole graph before applying Savitch’s theorem, we get $O(cf(n))$ 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. 248, 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.\(^a\)
- So connectivity is checked locally and on demand.

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

- The $z$ variable in the algorithm on p. 248 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 a fixed 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. 248 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. 223).

- 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 if B reduces to A.

\(^a\)See also p. 145.
\(^b\)Or simply “harder than” for brevity.
Solving problem B by calling the algorithm for problem A once and without further processing its answer.\(^a\)

\(^a\)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

- Suppose B reduces to A via a transformation $R$.\(^b\)
- 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.\(^c\)
  - Some instances of A may never appear in $R$’s range.
- But $x$ must be a general instance for B.

\(^a\)Contributed by Mr. Ming-Feng Tsai (D92922003) on October 29, 2003.
\(^b\)Sometimes, we say “B can be reduced to A.”
\(^c\)\(R(x)\) 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. 148.
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. 235), $R$ 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 “$x \in L_1$?”\(^b\)

---

\(^a\)In fact, unless stated otherwise, we will only require that the reduction $R$ run in polynomial time.

\(^b\)Of course, it may not be the best.
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$^a$ $R(G)$ 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 $(i, j) \not\in E$ 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

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

  satisfies all clauses of \(R(G)\).
A Comment\textsuperscript{a}

- 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 $G$.

\textsuperscript{a}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

\[a^{\text{Floyd (1962); Marshall (1962).}}\]
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 $\lor$ 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

$\left[ g \Rightarrow (h \lor h') \right] \land \left[ (h \lor h') \Rightarrow g \right] \\
\equiv g \iff (h \lor h').$

$g$ 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 $R(C)$. 
An Example

\[
\begin{align*}
(h_1 & \Leftrightarrow x_1) \land (h_2 \Leftrightarrow x_2) \land (h_3 \Leftrightarrow x_3) \land (h_4 \Leftrightarrow x_4) \\
\land & [g_1 \Leftrightarrow (h_1 \land h_2)] \land [g_2 \Leftrightarrow (h_3 \lor h_4)] \\
\land & [g_3 \Leftrightarrow (g_1 \land g_2)] \land (g_4 \Leftrightarrow \neg g_2) \\
\land & [g_5 \Leftrightarrow (g_3 \lor g_4)] \land g_5.
\end{align*}
\]
An Example (concluded)

• 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. 205 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.
Completeness\textsuperscript{a}

- As reducibility is transitive, problems can be ordered with respect to their difficulty.

- Is there a \textit{maximal} element (the so-called \textit{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 $\mathcal{C}$ 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 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.\(^a\)

\(^a\)See also p. 159.
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 $L$.  

---

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