More Undecidability

- \{M : M \text{ halts on all inputs}\}.
  - Given \( M; x \), we construct the following machine:
    * \( M_x(y) : \text{if } y = x \text{ then } M(x) \text{ else halt.} \)
  - \( M_x \) halts on all inputs if and only if \( M \) halts on \( x \).
  - So if the said language were recursive, \( H \) would be recursive, a contradiction.
    - This technique is called reduction.
- \{M; x : \text{there is a } y \text{ such that } M(x) = y\}.
- \{M; x : \text{the computation } M \text{ on input } x \text{ uses all states of } M\}.
- \{M; x; y : M(x) = y\}.
Reductions in Proving Undecidability

- Suppose we are asked to prove $L$ is undecidable.
- Language $H$ is known to be undecidable.
- We try to find a computable transformation (or reduction) $R$ such that
  $$x \in L \text{ if and only if } R(x) \in H.$$  
- This suffices to prove that $L$ is undecidable.
Complements of Recursive Languages

**Lemma 10** If $L$ is recursive, then so is $\overline{L}$.

- Let $L$ be decided by $M$ (which is deterministic).
- Swap the “yes” state and the “no” state of $M$.
- The new machine decides $\overline{L}$.
- This idea does not work if is “recursive” is replaced with “recursively enumerable” (p. 79).
Recursive and Recursively Enumerable Languages

**Lemma 11**  $L$ is recursive if and only if both $L$ and $\overline{L}$ are recursively enumerable.

- Suppose both $L$ and $\overline{L}$ are recursively enumerable, accepted by $M$ and $\overline{M}$, respectively.
- Simulate $M$ and $\overline{M}$ in an *interleaved* fashion.
- If $M$ accepts, then $x \in L$ and $M'$ halts on state “yes.”
- If $\overline{M}$ accepts, then $x \notin L$ and $M'$ halts on state “no.”
R, RE, and coRE

**RE:** The set of all recursively enumerable languages.

**coRE:** The set of all languages whose complements are recursively enumerable (note that coRE is not $\overline{RE}$).

**R:** The set of all recursive languages.
- $R = RE \cap coRE$ (p. 116).
- There exist languages in RE but not in R or coRE (such as $H$).
- There are languages in coRE but not in R or RE (such as $\overline{H}$).
- There are languages in neither RE nor coRE.
Notations

• Suppose $M$ is a TM accepting $L$.

• Write $L(M) = L$.

• If $M(x)$ is never “yes” nor $\not\rightarrow$ (as required by the definition of acceptance), we define $L(M) = \emptyset$.

• Of course, if $M(x) = \not\rightarrow$ for all $x$, then $L(M) = \emptyset$, too.
Nontrivial Properties of Sets in RE

- A property of a set accepted by a TM (a recursively enumerable set) is **trivial** if it is always true or false.
  - Is an RE set accepted by a TM? Always true.
- It can be defined by the set \( C \) of RE sets that satisfy it.
- The property is nontrivial if \( C \neq \text{RE} \) and \( C \neq \emptyset \).
- Up to now, all nontrivial properties of RE sets are undecidable (p. 113).
- In fact, Rice’s theorem confirms that.
Rice’s Theorem

Theorem 12 (Rice’s theorem) Suppose $C \neq \emptyset$ is a proper subset of the set of all recursively enumerable languages. Then the question “$L(M) \in C$?” is undecidable.

- Assume that $\emptyset \notin C$ (otherwise, repeat the proof for the class of all recursively enumerable languages not in $C$).
- Let $L \in C$ be accepted by TM $M_L$ (recall that $C \neq \emptyset$).
- Let $M_H$ accept the undecidable language $H$.
  - $M_H$ exists (p. 109).
The Proof (continued)

- Consider machine $M_x(y)$:

  $\textbf{if } M_H(x) = \text{"yes" } \textbf{then } M_L(y) \textbf{ else } \uparrow$

- If we can prove that

  $L(M_x) \in \mathcal{C}$ if and only if $x \in H$, \hspace{1cm} (2)

  then we are done because the halting problem has been reduced to deciding $L(M_x) \in \mathcal{C}$.

- We proceed to prove claim (2).
The Proof (concluded)

• Suppose $x \in H$, i.e., $M_H(x) = \text{‘yes’}$.
  
  – $M_x(y)$ determines this, and it either accepts $y$ or never halts, depending on whether $y \in L$.
  
  – Hence $L(M_x) = L \in C$.

• Suppose $M_H(x) = \uparrow$.
  
  – $M_x$ never halts.
  
  – $L(M_x) = \emptyset \notin C$. 
Consequences of Rice’s Theorem

**Corollary 13** The following properties of recursively enumerative sets are undecidable.

- *Emptiness.*
- *Finiteness.*
- *Regularity.*
- *Context-freedom.*
Boolean Logic\textsuperscript{a}

Boolean variables: $x_1, x_2, \ldots$. 

Literals: $x_i, \neg x_i$. 

Boolean connectives: $\lor, \land, \neg$. 

Boolean expressions: Boolean variables, $\neg \phi$ (negation), $\phi_1 \lor \phi_2$ (disjunction), $\phi_1 \land \phi_2$ (conjunction).

- $\bigvee_{i=1}^{n} \phi_i$ stands for $\phi_1 \lor \phi_2 \lor \cdots \lor \phi_n$. 
- $\bigwedge_{i=1}^{n} \phi_i$ stands for $\phi_1 \land \phi_2 \land \cdots \land \phi_n$. 

Implications: $\phi_1 \Rightarrow \phi_2$ is a shorthand for $\neg \phi_1 \lor \phi_2$. 

Biconditionals: $\phi_1 \Leftrightarrow \phi_2$ is a shorthand for

$(\phi_1 \Rightarrow \phi_2) \land (\phi_2 \Rightarrow \phi_1)$. 

\textsuperscript{a}Boole (1815–1864), 1847.
Truth Assignments

- A truth assignment $T$ is a mapping from boolean variables to truth values true and false.

- A truth assignment is appropriate to boolean expression $\phi$ if it defines the truth value for every variable in $\phi$.
  
  - $\{x_1 = \text{true}, x_2 = \text{false}\}$ it appropriate to $x_1 \lor x_2$. 
Satisfaction

• $T \models \phi$ means boolean expression $\phi$ is true under $T$; in other words, $T$ satisfies $\phi$.

• $\phi_1$ and $\phi_2$ are equivalent, written

$$\phi_1 \equiv \phi_2,$$

if for any truth assignment $T$ appropriate to both of them, $T \models \phi_1$ if and only if $T \models \phi_2$.

– Equivalently, $T \models (\phi_1 \Leftrightarrow \phi_2)$.
Truth Tables

• Suppose $\phi$ has $n$ boolean variables.

• A truth table contains $2^n$ rows, one for each possible truth assignment of the $n$ variables together with the truth value of $\phi$ under that truth assignment.

• A truth table can be used to prove if two boolean expressions are equivalent.

• De Morgan’s laws say that

\[ \neg(\phi_1 \land \phi_2) = \neg\phi_1 \lor \neg\phi_2 \]
\[ \neg(\phi_1 \lor \phi_2) = \neg\phi_1 \land \neg\phi_2 \]
<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \land q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Conjunctive Normal Forms

- A boolean expression $\phi$ is in **conjunctive normal form** (CNF) if

$$\phi = \bigwedge_{i=1}^{n} C_i,$$

where each **clause** $C_i$ is the disjunction of one or more literals.

- For example,

$$(x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (x_2 \lor x_3).$$

is in CNF.
Disjunctive Normal Forms

- A boolean expression $\phi$ is in **disjunctive normal form** (DNF) if

$$
\phi = \bigvee_{i=1}^{n} D_i,
$$

where each **implicant** $D_i$ is the conjunction of one or more literals.

- For example,

$$(x_1 \land x_2) \lor (x_1 \land \neg x_2) \lor (x_2 \land x_3).$$

is in DNF.
Any Expression $\phi$ Can Be Converted into CNFs and DNFs

$\phi = x_j$: This is trivially true.

$\phi = \neg \phi_1$ and a CNF is sought: Turn $\phi_1$ into a DNF and apply de Morgan’s laws to make a CNF for $\phi$.

$\phi = \neg \phi_1$ and a DNF is sought: Turn $\phi_1$ into a CNF and apply de Morgan’s laws to make a DNF for $\phi$.

$\phi = \phi_1 \lor \phi_2$ and a DNF is sought: Make $\phi_1$ and $\phi_2$ DNFs.

$\phi = \phi_1 \lor \phi_2$ and a CNF is sought: Let $\phi_1 = \bigwedge_{i=1}^{n_1} A_i$ and $\phi_2 = \bigwedge_{i=1}^{n_2} B_i$ be CNFs. Set $\phi = \bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} (A_i \lor B_j)$.

$\phi = \phi_1 \land \phi_2$: Similar.
Satisfiability

- A boolean expression $\phi$ is **satisfiable** if there is a truth assignment $T$ appropriate to it such that $T \models \phi$.
- $\phi$ is **valid** or a **tautology**, written $\models \phi$, if $T \models \phi$ for all $T$ appropriate to $\phi$.
- $\phi$ is **unsatisfiable** if and only if $\phi$ is false under all appropriate truth assignments if and only if $\neg \phi$ is valid.

---

\textsuperscript{a}Wittgenstein (1889–1951), 1922.
SATISFIABILITY (SAT)

- The **length** of a boolean expression is the length of the string encoding it.
- **SATISFIABILITY (SAT):** Given a CNF $\phi$, is it satisfiable?
- Solvable in time $O(n^22^n)$ on a TM by the truth table method.
- Solvable in polynomial time on an NTM, hence in NP (p. 80).
- A most important problem in answering the $P = NP$ problem (p. 225).
UNSATISFIABILITY (UNSAT or SAT COMPLEMENT) and VALIDITY

• UNSAT (SAT COMPLEMENT): Given a boolean expression $\phi$, is it unsatisfiable?

• VALIDITY: Given a boolean expression $\phi$, is it valid?
  – $\phi$ is valid if and only if $\neg\phi$ is unsatisfiable.
  – So UNSAT and VALIDITY have the same complexity.

• Both are solvable in time $O(n^22^n)$ on a TM by the truth table method.
Relations among \textsc{sat}, \textsc{unsat}, and \textsc{validity}

- The negation of an unsatisfiable expression is a valid expression.
- None of the three problems—satisfiability, unsatisfiability, validity—are known to be in \textsc{p}.
Horn Clauses

• A **Horn clause** is a clause with at most one *positive* literal.
  
  \[ \neg x_2 \lor x_3, \neg x_1 \lor \neg x_2 \lor \neg x_3. \]

• A Horn clause of form \( y \lor \neg x_1 \lor \neg x_2 \lor \cdots \lor \neg x_m \) can be rewritten as an implication

\[
(x_1 \land x_2 \land \cdots \land x_m) \Rightarrow y, \\
\]

where \( y \) is the positive literal.

  – If \( m = 0 \), use \texttt{true} \( \Rightarrow y \), also in implication form.

• If a Horn clause has no positive literals, we keep its *non-implication* form, \( \neg x_1 \lor \neg x_2 \lor \cdots \lor \neg x_m \).
Satisfiability of CNFs with Horn Clauses Is in P

• Interpret a truth assignment as a set $T$ of those variables that are assigned true.
  - $T \models x_i$ if and only if $x_i \in T$.
  - $x_i \notin T$ means $x_i = \text{false}$, not that $x_i$ is undetermined.

• Let $\phi$ be a conjunction of Horn clauses.

• We will prove that satisfiability of $\phi$ is in P.
The Algorithm

1: $T := \emptyset$; \{All variables are false.\}
2: \textbf{while} not all \textit{implications} are satisfied \textbf{do}
3: \hspace{1em} Pick a $(x_1 \land x_2 \land \cdots \land x_m) \Rightarrow y$ not satisfied by $T$;
4: \hspace{1em} Add $y$ to $T$; \{Make $y$ true (it was false).\}
5: \textbf{end while}
6: \textbf{if} $T \models \phi$ \textbf{then}
7: \hspace{1em} \textbf{return} “$\phi$ is satisfiable”;
8: \textbf{else}
9: \hspace{1em} \textbf{return} “$\phi$ is unsatisfiable”;
10: \textbf{end if}
Analysis of the Algorithm

- $T$ is monotonically increasing in size.
- Eventually $T$ will be large enough to make all implications (but not necessarily all Horn clauses) true.
  - Note we only make false variables true, never vice versa.
  - Reversing $y$’s truth value will not make currently satisfied implications false.
- So the **while** loop will terminate.
- By the time the **while** loop exits, all implications are satisfied by $T$.
- The running time is clearly polynomial.
Analysis of the Algorithm (continued)

- Any set $T'$ satisfying all the implications must be such that $T \subseteq T'$.
  - Otherwise, consider the first time in the execution of the algorithm at which $T \not\subseteq T'$.
  - That $(x_1 \land x_2 \land \cdots \land x_m) \Rightarrow y$ causes the insertion of $y$ to $T$ means $T \models x_1 \land x_2 \land \cdots \land x_m$ (and $T \not\models y$).
  - Hence $y \not\in T'$ but $\{x_1, x_2, \ldots, x_m\} \in T'$.
  - Hence $T \not\models (x_1 \land x_2 \land \cdots \land x_m) \Rightarrow y$, a contradiction.
Analysis of the Algorithm (concluded)

- If $T \not\models \neg x_1 \lor \neg x_2 \lor \cdots \lor \neg x_m$, then
  $$\{x_1, x_2, \ldots, x_m\} \subseteq T.$$ 

- Hence no supersets of $T$ can satisfy this clause.

- Because to satisfy all the implications must be a superset of $T$, $\phi$ is unsatisfiable.
Boolean Functions

• An $n$-ary boolean function is a function

$$f : \{\text{true, false}\}^n \rightarrow \{\text{true, false}\}.$$  

• It can be represented by a truth table.

• There are $2^n$ such boolean functions.
  
  – Each of the $2^n$ truth assignments can make $f$ true or false.
Boolean Functions (continued)

- A boolean expression expresses a boolean function.
  - Think of its truth value under all truth assignments.
- A boolean function expresses a boolean expression.
  - $\bigvee T \models \phi$, literal $y_i$ is true under $T(y_1 \land y_2 \land \cdots \land y_n)$.
  - The boolean function on p. 129 produces $p \land q$.
  - The length$^a$ is $\leq n2^n \leq 2^{2n}$.
  - In general, the exponential length in $n$ cannot be avoided (p. 150)!

$^a$We mean the logical connectives here.
Boolean Circuits

• A boolean circuit is a graph $C$ whose nodes are the gates.
• There are no cycles in $C$.
• All nodes have indegree (number of incoming edges) equal to 0, 1, or 2.
• Each gate has a sort from

$$\{\text{true}, \text{false}, \lor, \land, \neg, x_1, x_2, \ldots\}.$$
Boolean Circuits (concluded)

- Gates of sort from \{\texttt{true}, \texttt{false}, x_1, x_2, \ldots\} are the \textbf{inputs} of \(C\) and have an indegree of zero.

- The \textbf{output gate(s)} has no outgoing edges.

- A boolean circuit computes a boolean function.
Boolean Circuits and Expressions

- They are equivalent representations.
- One can construct one from the other:
An Example

$$(x_1 \land x_2) \land (x_3 \lor x_4) \lor \neg(x_3 \lor x_4)$$

- Circuits are more economical because of the possibility of sharing.
CIRCUIT SAT and CIRCUIT VALUE

CIRCUIT SAT: Given a circuit, is there a truth assignment such that the circuit outputs true?

CIRCUIT VALUE: The same as CIRCUIT SAT except that the circuit has no variable gates.

- CIRCUIT SAT ∈ NP: Guess a truth assignment and then evaluate the circuit.

- CIRCUIT VALUE ∈ P: Evaluate the circuit from the input gates gradually towards the output gate.
Some Boolean Functions Need Exponential Circuits

Theorem 14 (Shannon, 1949) For any $n \geq 2$, there is an $n$-ary boolean function $f$ such that no boolean circuits with $2^n/(2n)$ or fewer gates can compute it.

- There are $2^{2^n}$ different $n$-ary boolean functions.
- There are at most $((n + 5) \times m^2)^m$ boolean circuits with $m$ or fewer gates.
- But $((n + 5) \times m^2)^m < 2^{2^n}$ when $m = 2^n/(2n)$.
  
  $$m \log_2((n + 5) \times m^2) = 2^n \left(1 - \frac{\log_2 \left(\frac{4n^2}{n+5}\right)}{2n}\right) < 2^n$$
  
  for $n \geq 2$.

- Can be improved to “almost all boolean functions...”
Proper (Complexity) Functions

- We say that $f : \mathbb{N} \to \mathbb{N}$ is a **proper (complexity)** function if the following hold:
  - $f$ is nondecreasing.
  - There is a $k$-string TM $M_f$ such that
    $$M_f(x) = \square^f(|x|)$$
    for any $x$.
  - $M_f$ halts after $O(|x| + f(|x|))$ steps.
  - $M_f$ uses $O(f(|x|))$ space besides its input $x$. 
Examples of Proper Functions

- Most “reasonable” functions are proper: $c$, $\lceil \log n \rceil$, polynomials of $n$, $2^n$, $\sqrt{n}$, $n!$, etc.

- If $f$ and $g$ are proper, then so are $f + g$, $fg$, and $2^g$.

- Nonproper functions when serving as the time bounds for complexity classes spoil “the theory building.”
  - For example, $\text{TIME}(f(n)) = \text{TIME}(2^{f(n)})$ for some recursive function $f$ (the gap theorem).

- We shall henceforth use only proper functions in relation to complexity classes $\text{TIME}(f(n))$, $\text{SPACE}(f(n))$, $\text{NTIME}(f(n))$, and $\text{NSPACE}(f(n))$. 
Space-Bounded Computation and Proper Functions

- In the definition of space-bounded computations, 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 an output of length \( f(n) \) first.
  
  - The space-bound computation must repeat a configuration if it runs for more than \( c^{n+f(n)} \) steps for some \( c \) (p. 171).
  
  - So we can count steps to prevent infinite loops.
Precise Turing Machines

- A TM $M$ is **precise** if there are functions $f$ and $g$ such that for every $n \in \mathbb{N}$, for every $x$ of length $n$, and for every computation path of $M$,
  - $M$ halts after precise $f(n)$ steps, and
  - All of its strings are at halting of length precisely $g(n)$.
    * If $M$ is a TM with input and output, we exclude the first and the last strings.

- $M$ can be deterministic or nondeterministic.
Precise TMs Are General

Proposition 15 Suppose a (deterministic or nondeterministic) TM $M$ decides $L$ within time (space) $f(n)$, where $f$ is proper. Then there is a precise TM $M'$ which decides $L$ in time $O(n + f(n))$ (space $O(f(n))$, respectively).
The Proof

- \( M' \) on input \( x \) first simulates the TM \( M_f \) associated with the proper function \( f \) on \( x \).
- \( M_f \)'s output of length \( f(|x|) \) will serve as a “yardstick” or an “alarm clock.”
- If \( f \) is a space bound:
  - \( M' \) simulates on \( M_f \)'s output string.
  - The total space, besides the input string, is \( O(f(n)) \).
The Proof (concluded)

- If $f$ is a time bound:
  - The simulation of each step of $M$ on $x$ is matched by advancing the cursor on the “clock” string.
  - The simulation stops at the moment the “clock” string is exhausted.
  - The time bound is therefore $O(|x| + f(|x|))$. 
The Most Important Complexity Classes

- We write expressions like $n^k$ to denote the union of all complexity classes, one for each value of $k$.

- For example, $\text{NTIME}(n^k) = \bigcup_{j>0} \text{NTIME}(n^j)$.

\[
\begin{align*}
P &= \text{TIME}(n^k) \\
\text{NP} &= \text{NTIME}(n^k) \\
\text{PSPACE} &= \text{SPACE}(n^k) \\
\text{NPSPACE} &= \text{NSPACE}(n^k) \\
\text{EXP} &= \text{TIME}(2^{n^k}) \\
L &= \text{SPACE}(\log n) \\
\text{NL} &= \text{NSPACE}(\log n)
\end{align*}
\]
Complements of Nondeterministic Classes

- From p. 117, we know R, RE, and coRE are distinct.
  - coRE contains the complements of languages in RE, not the languages not in RE.

- Recall that the complement of $L$, denoted by $\bar{L}$, is the language $\Sigma^* - L$.
  - SAT COMPLEMENT is the set of unsatisfiable boolean expressions.
  - HAMILTONIAN PATH COMPLEMENT is the set of graphs without a Hamiltonian path.
The Co-Classes

• For any complexity class $\mathcal{C}$, $\text{co}\mathcal{C}$ denotes the class

  $\{\overline{L} : L \in \mathcal{C}\}$.

• Clearly, if $\mathcal{C}$ is a deterministic time or space complexity class, then $\mathcal{C} = \text{co}\mathcal{C}$.
  
  – They are said to be **closed under complement**.
  
  – A deterministic TM deciding $L$ can be converted to one that decides $\overline{L}$ within the same time or space bound by reversing the “yes” and “no” states.

• Whether nondeterministic classes for time are closed under complement is not known (p. 79).
Comments

- Then $\text{coC}$ is the class
  \[ \{ \bar{L} : L \in C \} . \]
- It is true that $x \in L$ if and only if $x \not\in \bar{L}$.
- But it is not true that $L \in C$ if and only if $L \not\in \text{coC}$.
  - $\text{coC}$ is not defined as $\bar{C}$. 