More Undecidability

- $\{M : M$ halts on all inputs$\}$.
  - Given $M; x$, we construct the following machine:
    - $M_x(y)$: if $y = x$ then $M(x)$ 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**.
- $\{M; x :$ there is a y such that $M(x) = y\}$.
- $\{M; x :$ the computation M on input x 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 $\bar{L}$.*

- Let L be decided by M (which is deterministic).
- Swap the "yes" state and the "no" state of M.
- The new machine decides $\bar{L}$.
- 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 $\bar{L}$ are recursively enumerable.*

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

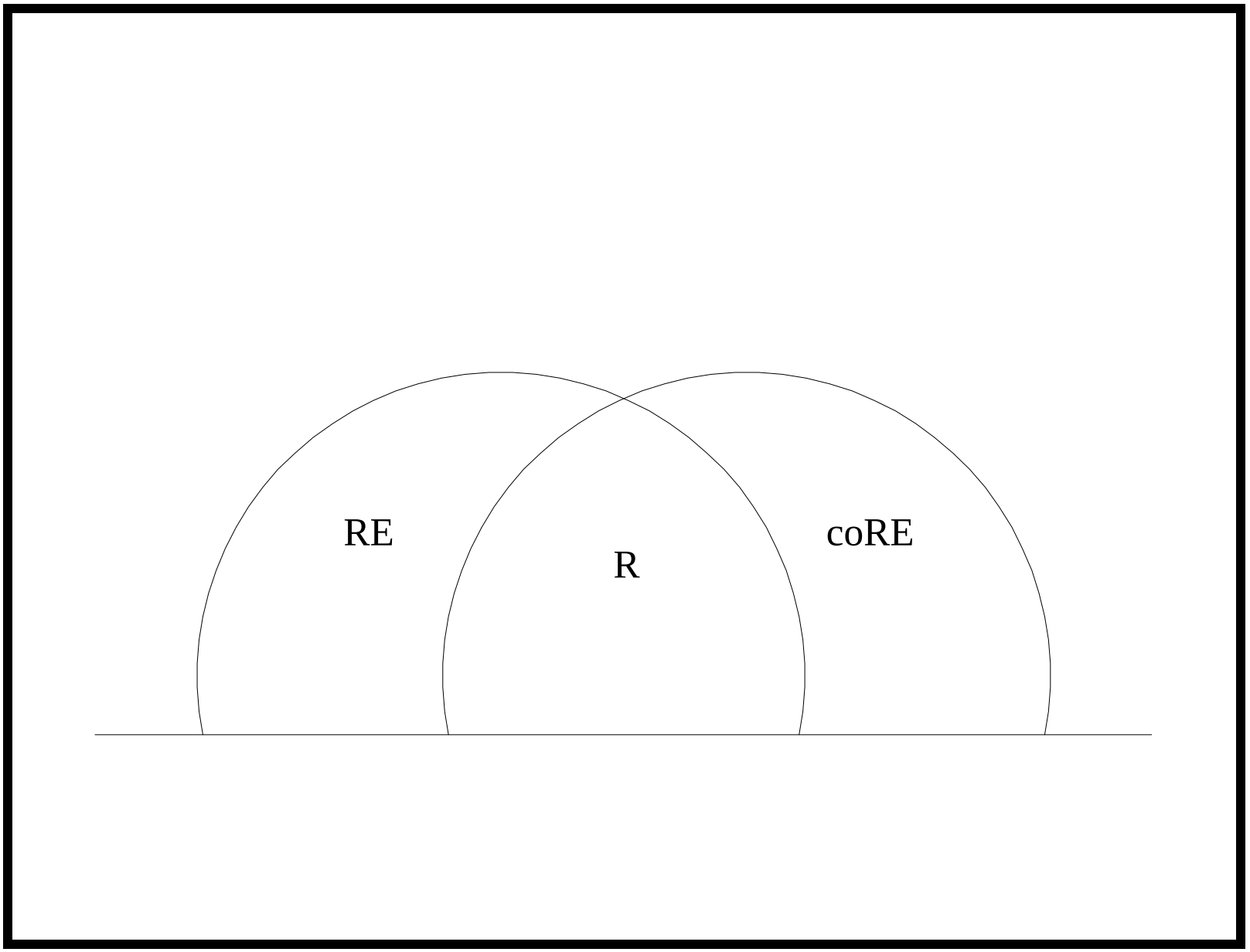
R, RE, and coRE

RE: The set of all recursively enumerable languages.

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

R: The set of all recursive languages.

- $\text{R} = \text{RE} \cap \text{coRE}$ (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 $\bar{H}$).
- 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 $\nearrow$ (as required by the definition of acceptance), we define $L(M) = \emptyset$.
- Of course, if $M(x) = \nearrow$ 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 $\mathcal{C}$ of RE sets that satisfy it.
- The property is nontrivial if $\mathcal{C} \neq \text{RE}$ and $\mathcal{C} \neq \emptyset$.
- 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 $\mathcal{C} \neq \emptyset$ is a proper subset of the set of all recursively enumerable languages. Then the question "$L(M) \in \mathcal{C}$?" is undecidable.*

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

The Proof (continued)

- Consider machine $M_x(y)$:

$$\textbf{if } M_H(x) = \text{“yes”} \textbf{ then } M_L(y) \textbf{ else } \nearrow$$

- If we can prove that

$$L(M_x) \in \mathcal{C} \text{ if and only if } x \in H, \tag{2}$$

  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., $M_H(x) =$ "yes."
  - $M_x(y)$ determines this, and it either accepts y or never halts, depending on whether $y \in L$.
  - Hence $L(M_x) = L \in \mathcal{C}$.
- Suppose $M_H(x) = \nearrow$.
  - M_x never halts.
  - $L(M_x) = \emptyset \notin \mathcal{C}$.

Consequences of Rice's Theorem

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

- *Emptiness.*
- *Finiteness.*
- *Regularity.*
- *Context-freedom.*

Boolean Logic[a]

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

Literals: x_i, $\neg x_i$.

Boolean connectives: $\vee, \wedge, \neg$.

Boolean expressions: Boolean variables, $\neg\phi$ (**negation**), $\phi_1 \vee \phi_2$ (**disjunction**), $\phi_1 \wedge \phi_2$ (**conjunction**).

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

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

Biconditionals: $\phi_1 \Leftrightarrow \phi_2$ is a shorthand for $(\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$.

---

[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 ϕ if it defines the truth value for every variable in ϕ.
  - $\{x_1 = \texttt{true}, x_2 = \texttt{false}\}$ it appropriate to $x_1 \vee x_2$.

Satisfaction

- $T \models \phi$ means boolean expression ϕ is true under T; in other words, T **satisfies** ϕ.
- ϕ_1 and ϕ_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 ϕ 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 ϕ under that truth assignment.
- A truth table can be used to prove if two boolean expressions are equivalent.
- **De Morgan's laws** say that

$$\begin{aligned} \neg(\phi_1 \wedge \phi_2) &= \neg\phi_1 \vee \neg\phi_2 \\ \neg(\phi_1 \vee \phi_2) &= \neg\phi_1 \wedge \neg\phi_2 \end{aligned}$$

Truth Table

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Conjunctive Normal Forms

- A boolean expression ϕ 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,

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3).$$

  is in CNF.

Disjunctive Normal Forms

- A boolean expression ϕ 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,

$$(x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2) \vee (x_2 \wedge x_3).$$

  is in DNF.

Any Expression ϕ Can Be Converted into CNFs and DNFs

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

$\phi = \neg\phi_1$ **and a CNF is sought:** Turn ϕ_1 into a DNF and apply de Morgan's laws to make a CNF for ϕ.

$\phi = \neg\phi_1$ **and a DNF is sought:** Turn ϕ_1 into a CNF and apply de Morgan's laws to make a DNF for ϕ.

$\phi = \phi_1 \vee \phi_2$ **and a DNF is sought:** Make ϕ_1 and ϕ_2 DNFs.

$\phi = \phi_1 \vee \phi_2$ **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 $\phi = \bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} (A_i \vee B_j)$.

$\phi = \phi_1 \wedge \phi_2$**:** Similar.

Satisfiability

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

[a] Wittgenstein (1889–1951), 1922.

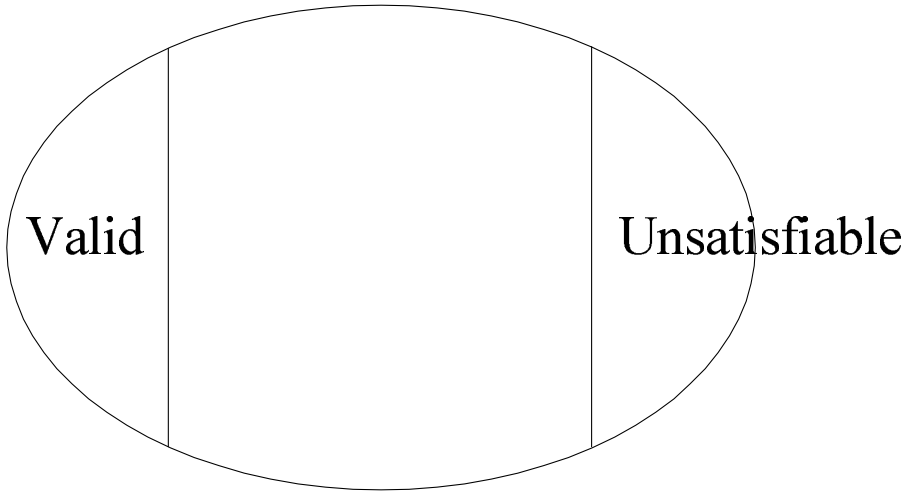
SATISFIABILITY (SAT)

- The **length** of a boolean expression is the length of the string encoding it.
- SATISFIABILITY (SAT): Given a CNF ϕ, is it satisfiable?
- Solvable in time $O(n^2 2^n)$ 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 ϕ, is it unsatisfiable?
- VALIDITY: Given a boolean expression ϕ, is it valid?
  - ϕ is valid if and only if $\neg\phi$ is unsatisfiable.
  - So UNSAT and VALIDITY have the same complexity.
- Both are solvable in time $O(n^2 2^n)$ on a TM by the truth table method.

Relations among SAT, UNSAT, and VALIDITY



- The negation of an unsatisfiable expression is a valid expression.
- None of the three problems—satisfiability, unsatisfiability, validity—are known to be in P.

Horn Clauses

- A **Horn clause** is a clause with at most one *positive* literal.
  - $\neg x_2 \vee x_3$, $\neg x_1 \vee \neg x_2 \vee \neg x_3$.
- A Horn clause of form $y \vee \neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$ can be rewritten as an implication

$$(x_1 \wedge x_2 \wedge \cdots \wedge x_m) \Rightarrow y,$$

  where y is the positive literal.
  - If $m = 0$, use `true` $\Rightarrow y$, also in implication form.
- If a Horn clause has no positive literals, we keep its *non*-implication form, $\neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$.

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 $x_i = \texttt{false}$ not that x_i is undetermined.
- Let ϕ be a conjunction of Horn clauses.
- We will prove that satisfiability of ϕ is in P.

The Algorithm

1: $T := \emptyset$; {All variables are false.}
2: **while** not all *implications* are satisfied **do**
3: Pick a $(x_1 \wedge x_2 \wedge \cdots \wedge x_m) \Rightarrow y$ not satisfied by T;
4: Add y to T; {Make y true (it was false).}
5: **end while**
6: **if** $T \models \phi$ **then**
7: **return** "ϕ is satisfiable";
8: **else**
9: **return** "ϕ is unsatisfiable";
10: **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 $(x_1 \wedge x_2 \wedge \cdots \wedge x_m) \Rightarrow y$ causes the insertion of y to T means $T \models x_1 \wedge x_2 \wedge \cdots \wedge x_m$ (and $T \not\models y$).
  - Hence $y \notin T'$ but $\{x_1, x_2, \ldots, x_m\} \in T'$.
  - Hence $T \not\models (x_1 \wedge x_2 \wedge \cdots \wedge x_m) \Rightarrow y$, a contradiction.

Analysis of the Algorithm (concluded)

- If $T \not\models \neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$, 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, ϕ is unsatisfiable.

Boolean Functions

- An n-ary boolean function is a function

$$f : \{\texttt{true}, \texttt{false}\}^n \to \{\texttt{true}, \texttt{false}\}.$$

- It can be represented by a truth table.
- There are 2^{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.
  - $\bigvee_{T \models \phi, \text{ literal } y_i \text{ is true under } T}(y_1 \wedge y_2 \wedge \cdots \wedge y_n)$.
  - The boolean function on p. 129 produces $p \wedge q$.
  - 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

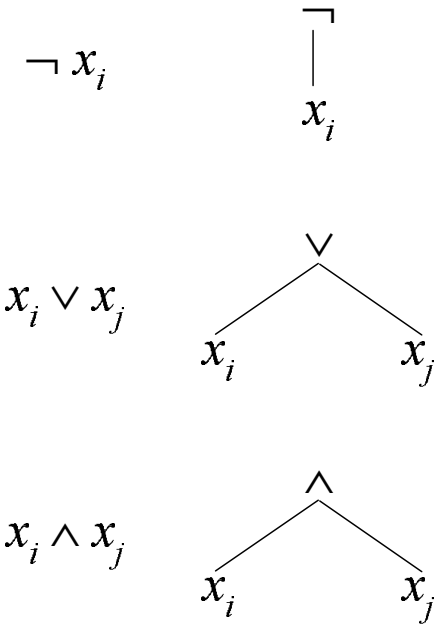
$$\{\texttt{true}, \texttt{false}, \vee, \wedge, \neg, x_1, x_2, \ldots\}.$$

Boolean Circuits (concluded)

- Gates of sort from $\{\texttt{true}, \texttt{false}, x_1, x_2, \ldots\}$ are the **inputs** of C and have an indegree of zero.
- The **output gate**(s) has no outgoing edges.
- A boolean circuit computes a boolean function.

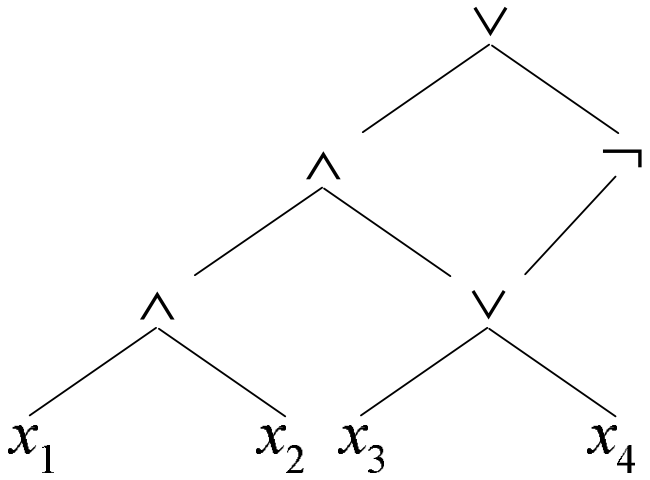
Boolean Circuits and Expressions

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



An Example

$$((x_1 \wedge x_2) \wedge (x_3 \vee x_4)) \vee (\neg(x_3 \vee x_4))$$



- 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 $\in$ NP: Guess a truth assignment and then evaluate the circuit.
- CIRCUIT VALUE $\in$ 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 $m = 2^n/(2n)$.
  - $m \log_2((n+5) \times m^2) = 2^n(1 - \frac{\log_2 \frac{4n^2}{n+5}}{2n}) < 2^n$ 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 $M_f(x) = \sqcap^{f(|x|)}$ 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, $\mathrm{NTIME}(n^k) = \bigcup_{j>0} \mathrm{NTIME}(n^j)$.

$$
\begin{aligned}
\mathrm{P} &= \mathrm{TIME}(n^k) \\
\mathrm{NP} &= \mathrm{NTIME}(n^k) \\
\mathrm{PSPACE} &= \mathrm{SPACE}(n^k) \\
\mathrm{NPSPACE} &= \mathrm{NSPACE}(n^k) \\
\mathrm{EXP} &= \mathrm{TIME}(2^{n^k}) \\
\mathrm{L} &= \mathrm{SPACE}(\log n) \\
\mathrm{NL} &= \mathrm{NSPACE}(\log n)
\end{aligned}
$$

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 $\mathcal{C}$, co$\mathcal{C}$ denotes the class

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

- 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 $\bar{L}$ 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 co$\mathcal{C}$ is the class

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

- It is true that $x \in L$ if and only if $x \notin \bar{L}$.
- But it is *not* true that $L \in \mathcal{C}$ if and only if $L \notin$ co$\mathcal{C}$.
  - co$\mathcal{C}$ is not defined as $\bar{\mathcal{C}}$.