Simulating Nondeterministic TMs

Theorem 4 *Suppose that language L is decided by an NTM N in time $f(n)$. Then it is decided by a 3-string deterministic TM M in time $O(c^{f(n)})$, where $c > 1$ is some constant depending on N.*

- On input x, M goes down every computation path of N using depth-first search (M does *not* know $f(n)$).
- If some path leads to "yes," then M enters the "yes" state.
- If none of the paths leads to "yes," then M enters the "no" state.

Corollary 5 $\text{NTIME}(f(n))) \subseteq \bigcup_{c>1} \text{TIME}(c^{f(n)})$.

A Nondeterministic Algorithm for Graph Reachability

```
x := 1;
for i = 2, 3, ..., n do
    Guess y ∈ {2, 3, ..., n}; {The next node.}
    if (x, y) ∈ G then
        if y = n then
            "yes"; {Node n is reached from node 1.}
        else
            x := y;
        end if
    else
        "no";
    end if
end for
"no";
```

Space Analysis

- Variables i, x, and y each require $O(\log n)$ bits.
- Testing if $(x, y) \in G$ is accomplished by consulting the input string with counters of $O(\log n)$ bit long.
- Hence REACHABILITY $\in$ NSPACE$(\log n)$.
  - REACHABILITY with more than one terminal node also has the same complexity.
- REACHABILITY is in P.

Infinite Sets

- A set is **countable** (**countably infinite**, or **denumerable**) if it is finite or if it can be put in one-one correspondence with the set of natural numbers.
  - Set of integers $\mathbb{N}$.
  - Set of positive integers.
  - Set of odd integers.
  - Set of rational numbers $(1/1, 1/2, 2/1, 1/3, 2/2, 3/1, 1/4, 2/3, 3/2, 4/1, \ldots)$.
  - Set of squared integers.

Cardinality

- Let A denote a set.
- Then 2^A denotes its **power set**, that is $\{B : B \subseteq A\}$.
  - If $|A| = k$, then $|2^A| = 2^k$.
- For any set C, define $|C|$ as C's **cardinality** (size).
- Two sets are said to have the same cardinality (written as $|A| = |B|$ or $A \sim B$) if there exists a one-to-one correspondence between their elements.
- $|A| \leq |B|$ if there is a one-to-one correspondence between A and one of B's subsets.
- $|A| < |B|$ if $|A| \leq |B|$ but $|A| \neq |B|$.
  - If $A \subseteq B$, then $|A| \leq |B|$, but if $A \subsetneq B$, then $|A| < |B|$?

Cardinality and Infinite Sets

- If A and B are infinite sets, it is possible that $A \subsetneq B$ yet $|A| = |B|$.
  - The set of integers *properly* contains the set of odd integers.
  - But the set of integers has the same cardinality as the set of odd integers.
- A lot of "paradoxes."

Hilbert's[a] Paradox of the Grand Hotel

- For a hotel with a finite number of rooms with all the rooms occupied, a new guest will be turned away.
- Now let us imagine a hotel with an infinite number of rooms, and all the rooms are occupied.
- A new guest comes and asks for a room.
- "But of course!" exclaims the proprietor, and he moves the person previously occupying Room 1 into Room 2, the person from Room 2 into Room 3, and so on ....
- The new customer occupies Room 1.

[a]David Hilbert (1862–1943).

Hilbert’s Paradox of the Grand Hotel (continued)

- Let us imagine now a hotel with an infinite number of rooms, all taken up, and an infinite number of new guests who come in and ask for rooms.
- “Certainly, gentlemen,” says the proprietor, “just wait a minute.”
- He moves the occupant Room 1 into Room 2, the occupant of Room 2 into Room 4, and so on.
- Now all odd-numbered rooms become free and the infinity of new guests can be accommodated in them.
- (“There are many rooms in my Father’s house, and I am going to prepare a place for you.” *John* 14:3.)

Galileo's[a] Paradox (1638)

- The squares of the positive integers can be placed in one-to-one correspondence with all the positive integers.
- This is contrary to the axiom of Euclid that the whole is greater than any of its proper parts.
- Resolution of paradoxes: Which notion results in better mathematics.

[a] Galileo (1564–1642).

Cantor's[a] Theorem

Theorem 6 *The set of all subsets of* N (2^N) *is infinite and not countable.*

- Suppose it is countable with $f : N \to 2^N$ being a bijection.
- Consider the set $B = \{k \in N : k \notin f(k)\} \subseteq N$.
- Suppose that $B = f(n)$ for some n.
- If $n \in f(n)$, then $n \in B$, but then $n \notin B$ by the definition of B.
- Hence $B \neq f(n)$ for any n.
- f is not a bijection, a contradiction.

---

[a] Georg Cantor (1845–1918).

Two Corollaries

- For any set T, finite or infinite,

$$|T| < |2^T|.$$

  - $|T| \leq |2^T|$ as $f(x) = \{x\}$ maps T into a subset of 2^T.
  - The inequality uses the same proof as Cantor's theorem.

- The set of all functions on N is not countable.
  - A function $f : N \to \{0, 1\}$ determines an $M \subseteq N$ in that $n \in M$ if and only if $f(n) = 1$.
  - So the set of functions from N to $\{0, 1\}$ has cardinality $|2^N|$.

Existence of Uncomputable Problems

- Every program is a sequence of 0s and 1s.
- Every program corresponds to some integer.
- The set of programs is countable.
- A function is a mapping from integers to integers.
- So there must exist functions for which there are no programs by the second corollary above.

Universal Turing Machine[a]

- A **universal Turing machine** U interprets the input as the *description* of a TM M concatenated with the *description* of an input to that machine, x.
  - Both M and x are over the alphabet of U.
- U simulates M on x so that

$$U(M; x) = M(x).$$

- Think of U as a modern computer, which can execute any valid machine code, or a Java Virtual machine, which can execute any valid Java bytecode.
- We skip the details of U.

[a] Turing, 1936.

The Halting Problem

- **Undecidable problems** are problems that have no algorithms or languages that are not recursive.
- We already knew undecidable problems must exist (p. 80).
- We now define a concrete undecidable problem, the **halting problem**:

$$H = \{M; x : M(x) \neq \nearrow\}.$$

  – Does M halt on input x?

H Is Recursively Enumerable

Proposition 7 *H is recursively enumerable.*

- Use the universal TM U to simulate M on x.
- When M is about to halt, U enters a "yes" state.
- This TM accepts H.
- Comment: Membership of x in any recursively enumerative language accepted by M can be answered by asking "$M; x \in H$?"

H Is Not Recursive

- Suppose there is a TM M_H that *decides* H.
- Write the program $D(M)$ that calls M_H:

```
if M_H(M; M) = "yes" then
    ↗; {Writing an infinite loop is easy, right?}
else
    "yes";
end if
```

- Consider now $D(D)$:
  - $D(D) = \nearrow \Rightarrow M_H(D; D) =$ "yes" $\Rightarrow D; D \in H \Rightarrow D(D) \neq \nearrow$, a contradiction.
  - $D(D) =$ "yes" $\Rightarrow M_H(D; D) =$ "no" $\Rightarrow D; D \notin H \Rightarrow D(D) = \nearrow$, a contradiction.

Comments

- Two levels of interpretations of M:
  - A sequence of 0s and 1s (data).
  - An encoding of instructions (programs).
- There are no paradoxes.
  - Concepts are familiar to computer scientists (but not philosophers or mathematicians).
  - Supply a C compiler to a C compiler, a Lisp interpreter to a Lisp interpreter, a Java compiler to a Java compiler, etc.

Self-Loop Paradoxes

Cantor's Paradox (1899):

Let T be the set of all sets.

- Then $2^T \subseteq T$.
- But we know $|2^T| > |T|$!

Russell's[a] Paradox (1901): Consider $S = \{A : A \notin A\}$.

- If $S \in S$, then $S \notin S$ by definition.
- If $S \notin S$, then $S \in S$ also by definition.

Eubulides: The Cretan says, "All Cretans are liars."

Sharon Stone, *The Specialist*: "I am not a woman you can trust."

More Undecidability

- $\{M : M$ halts on all inputs$\}$.
  - Given $M; x$, we construct the following machine:
    - $M'(y)$: if $y = x$ then $M(x)$ else halt.
  - M' 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 $M\}$.
- $\{M; x; y : M(x) = y\}$.

Properties of Recursive Languages

- If L is recursive, then so is $\bar{L}$.
  - If L is decided by M, swapping the "yes" state and the "no" state of M results in a TM that decides $\bar{L}$.
  - Can't work for recursively enumerable languages (p. 60).
- 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 M' halts on state "yes."
  - If $\bar{M}$ accepts, then 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.

- Known: $\text{R} = \text{RE} \cap \text{coRE}$.
- Known: 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.

Rice's Theorem

- Suppose M is a TM accepting L.
- Write $L(M) = L$.
- If $M(x)$ is neither "yes" nor $\nearrow$ (as required by the definition of acceptance), we define $L(M) = \emptyset$.
- Rice's theorem says any nontrivial property of TMs is undecidable.

Theorem 8 (Rice's Theorem) *Suppose that* $\mathcal{C} \neq \emptyset$ *is a proper subset of the set of all recursively enumerable languages. Then the question "*$L(M) \in \mathcal{C}$*?" is undecidable.*

The Proof

- 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 $L \neq \emptyset$).
- Let M_H accept the undecidable language H.
- 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

$$L(M_x) \in \mathcal{C} \text{ if and only if } x \in H, \tag{1}$$

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

The Proof (continued)

- We proceed to prove claim (1).
- Suppose that $x \in H$, 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 that $M_H(x) = \nearrow$.
  - M_x never halts.
  - $L(M_x) = \emptyset \notin \mathcal{C}$.

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

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.
- $T \models \phi$ 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

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

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

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.

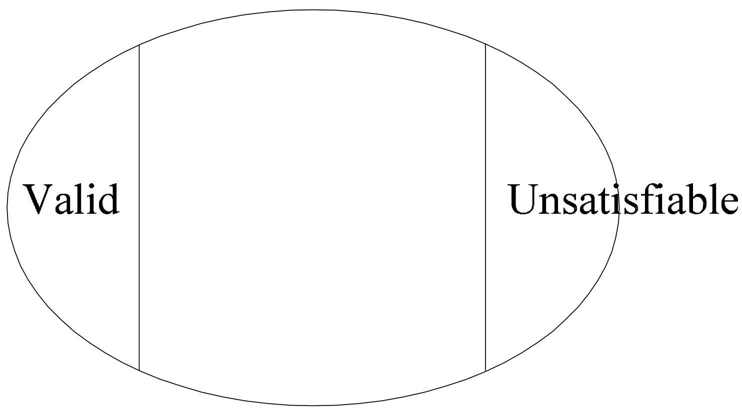
---

[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. 61).
- A most important problem in answering the P $=$ NP problem (p. 175).

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 $y \vee \neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$ 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 $\texttt{true} \Rightarrow y$, also in implication form.
- If a Horn clause has no positive literals, we keep its *non*-implication form, $\neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$.

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 $x_i \in T$.
- Let ϕ be a conjunction of Horn clauses.

The Algorithm

$T := \emptyset$; {All variables are false.}
while not all *implications* are satisfied **do**
    Pick an unsatisfied $(x_1 \wedge x_2 \wedge \cdots \wedge x_m) \Rightarrow y$;
    Add y to T; {Make y true.}
end while
if $T \models \phi$ **then**
    return "ϕ is satisfiable";
else
    return "ϕ is unsatisfiable";
end if

Analysis of the Algorithm

- It will terminate, because T is monotonically increasing in size and eventually it will be large enough to make all *implications* (but not necessarily all Horn clauses) true.
- By the time the **while** loop exits, all implications are satisfied by T.
- A T' satisfying all the implications must be such that $T \subseteq T'$.
  - Otherwise, the first time in the execution of the algorithm at which $T \not\subseteq T'$, the implication that causes insertion of y to T cannot be satisfied by T'.
- If $T \not\models \neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_m$, then $\{x_1, x_2, \ldots, x_m\} \subseteq T$ and hence no supersets of T can satisfy this clause, which means 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 be true or false.
- 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 exponential length in n cannot be avoided!

Boolean Circuits

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

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

- 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.

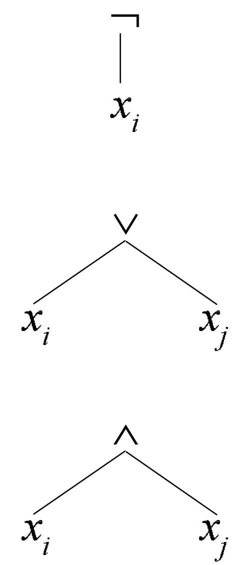
Boolean Circuits and Expressions

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

$\neg x_i$

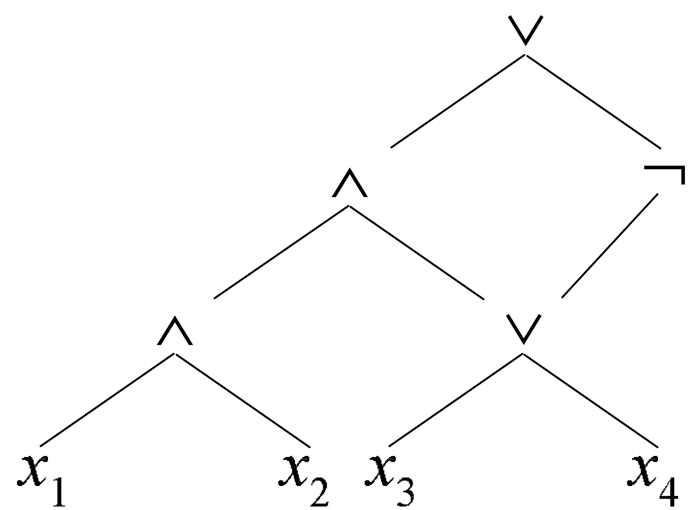
$x_i \vee x_j$

$x_i \wedge x_j$



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 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 is clearly in NP: Simply guess a truth assignment and then evaluate the circuit.
- CIRCUIT VALUE is clearly in P: Simply evaluate the circuit from the input gates gradually towards the output gate.
- CIRCUIT SAT and CIRCUIT VALUE: Is there a truth assignment of the variables of the circuit such that the resulting circuit value is true?

Some Boolean Functions Need Exponential Circuits

Theorem 9 (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…"