Cantor's[a] Theorem

Theorem 7 *The set of all subsets of* $\mathbb{N}$ $(2^{\mathbb{N}})$ *is infinite and not countable.*

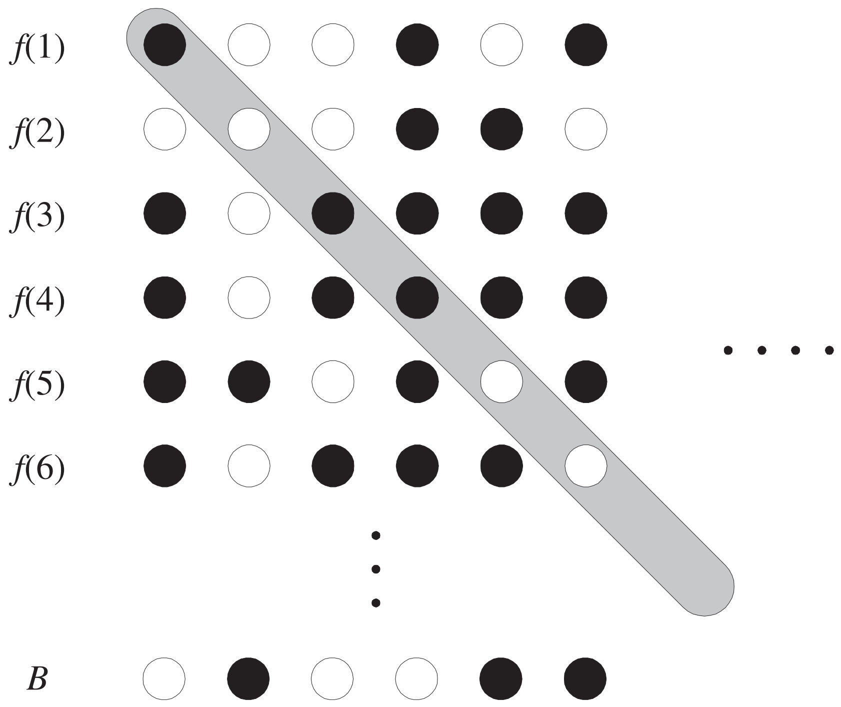
- Suppose it is countable with $f : \mathbb{N} \to 2^{\mathbb{N}}$ being a bijection.
- Consider the set $B = \{k \in \mathbb{N} : k \notin f(k)\} \subseteq \mathbb{N}$.
- Suppose $B = f(n)$ for some $n \in \mathbb{N}$.

[a] Georg Cantor (1845–1918). According to Kac and Ulam, "[If] one had to name a single person whose work has had the most decisive influence on the present spirit of mathematics, it would almost surely be Georg Cantor."

The Proof (concluded)

- If $n \in f(n)$, then $n \in B$, but then $n \notin B$ by B's definition.
- If $n \notin f(n)$, then $n \notin B$, but then $n \in B$ by B's definition.
- Hence $B \neq f(n)$ for any n.
- f is not a bijection, a contradiction.

Cantor's Diagonalization Argument Illustrated



A Corollary of Cantor's Theorem

Corollary 8 *For any set T, finite or infinite,*

$$|\,T\,| < |\,2^T\,|.$$

- The inequality holds in the finite A case.
- Assume A is infinite now.
- $|T| \leq |2^T|$: Consider $f(x) = \{x\}$.
- The strict inequality uses the same argument as Cantor's theorem.

A Second Corollary of Cantor's Theorem

Corollary 9 *The set of all functions on* $\mathbb{N}$ *is not countable.*

- It suffices to prove it for functions from $\mathbb{N}$ to $\{0, 1\}$.
- Every such function $f : \mathbb{N} \to \{0, 1\}$ determines a set

$$\{n : f(n) = 1\} \subseteq \mathbb{N}$$

  and vice versa.
- So the set of functions from $\mathbb{N}$ to $\{0, 1\}$ has cardinality $|\, 2^{\mathbb{N}} \,|$.
- Corollary 8 (p. 109) then implies the claim.

Existence of Uncomputable Problems

- Every program is a finite sequence of 0s and 1s, thus a nonnegative integer.
- Hence every program corresponds to some integer.
- The set of programs is countable.
- A function is a mapping from integers to integers.
- The set of functions is not countable by Corollary 9 (p. 110).
- So there must exist functions for which there are no programs.

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

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

- U is like a modern computer, which executes any valid machine code, or a Java Virtual machine, which executes any valid bytecode.

[a]Turing (1936).

The Halting Problem

- **Undecidable problems** are problems that have no algorithms or languages that are not recursive.
- We knew undecidable problems exist (p. 111).
- 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

- Use the universal TM U to simulate M on x.
- When M is about to halt, U enters a "yes" state.
- If $M(x)$ diverges, so does U.
- This TM accepts H.
- 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.
- Consider the program $D(M)$ that calls M_H:

if $M_H(M; M) =$ "yes" **then**
    $\nearrow$; {Writing an infinite loop is easy, right?}
else
    "yes";
end if

- Consider $D(D)$:
  - $D(D) = \nearrow \Rightarrow M_H(D; D) =$ "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 should be familiar to computer scientists.
  - Supply a C compiler to a C compiler, a Lisp interpreter to a Lisp interpreter, etc.

Self-Loop Paradoxes

Cantor's Paradox (1899): Let T be the set of all sets.[a]

- Then $2^T \subseteq T$ because 2^T is a set of sets.
- But we know $|\, 2^T \,| > |\, T \,|$ (p. 109)!
- We got a "contradiction."
- So what gives?
- Are we willing to give up Cantor's theorem?
- If not, what is a set?

[a]Recall this ontological argument for the existence of God by St Anselm (–1109) in the 11th century: If something is possible but is not part of God, then God is not the greatest possible object of thought, a contradiction.

Self-Loop Paradoxes (continued)

Russell's Paradox (1901): Consider $R = \{A : A \notin A\}$.

- If $R \in R$, then $R \notin R$ by definition.
- If $R \notin R$, then $R \in R$ also by definition.
- In either case, we have a "contradiction."

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

Liar's Paradox: "This sentence is false."

Self-Loop Paradoxes (concluded)

Sharon Stone in *The Specialist* (1994): "I'm not a woman you can trust."

Spin City: "I am not gay, but my boyfriend is."

Numbers 12:3, Old Testament: "Moses was the most humble person in all the world $[\cdots]$" (attributed to Moses).

More Undecidability

- $H^* = \{M : M \text{ halts on all inputs}\}$.
  - Given $M; x$, we construct the following machine:[a]

$$M_x(y) : M(x).$$

  - M_x halts on all inputs if and only if M halts on x.
  - In other words, $M_x \in H^*$ if and only if $M; x \in H$.
  - So if the said language were recursive, H would be recursive, a contradiction.
  - This technique is called **reduction**.

[a] Simplified by Mr. Chih-Hung Hsieh (D95922003) on October 5, 2006.

More Undecidability (concluded)

- $\{M; x : \text{there is a } y \text{ such that } M(x) = y\}$.
- $\{M; x : \text{the computation } M \text{ on input } x \text{ uses all states of } M\}$.
- $\{M; x; y : M(x) = y\}$.

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[a]

$$\forall x (R(x) \in L \text{ if and only if } x \in H).$$

- We can answer "$x \in H$?" for any x by asking $R(x) \in L$?
- This suffices to prove that L is undecidable.

[a] Contributed by Mr. Tai-Dai Chou (J93922005) on May 19, 2005.

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 $\bar{L}$.

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."

A Very Useful Corollary and Its Consequences

Corollary 12 *L is recursively enumerable but not recursive, then $\bar{L}$ is not recursively enumerable.*

- Suppose $\bar{L}$ is recursively enumerable.
- Then both L and $\bar{L}$ are recursively enumerable.
- By Lemma 11 (p. 124), L is recursive, a contradiction.

Corollary 13 *$\bar{H}$ is not recursively enumerable.*

R, RE, and coRE

RE: The set of all recursively enumerable languages.

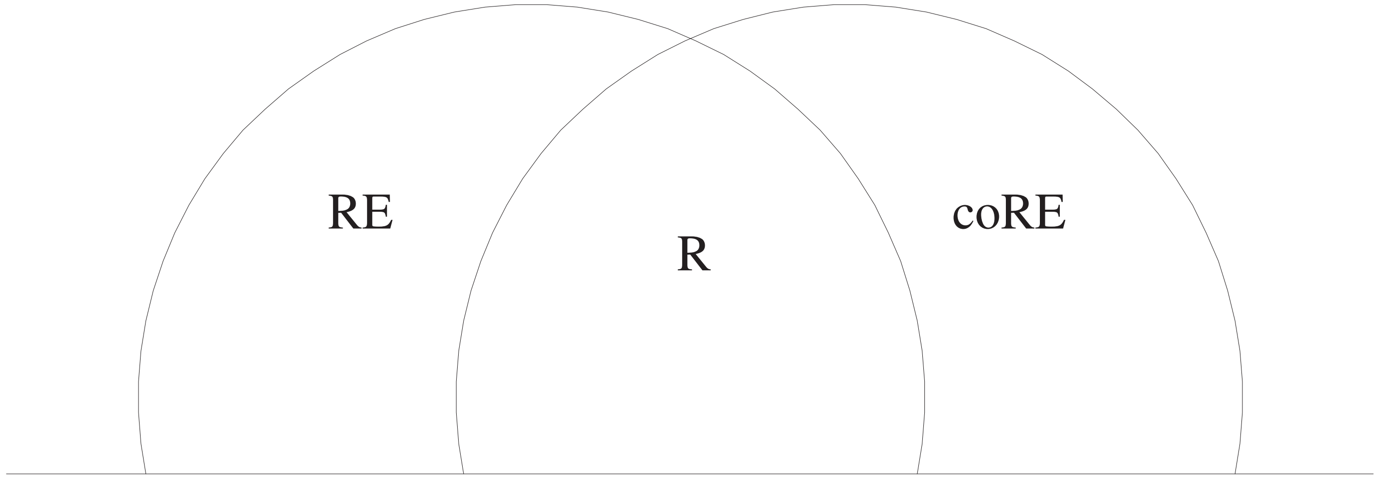
coRE: The set of all languages whose complements are recursively enumerable (note that coRE is not $\overline{\text{RE}}$).

- $\text{coRE} = \{ L : \overline{L} \in \text{RE} \}$.
- $\overline{\text{RE}} = \{ L : L \notin \text{RE} \}$.

R: The set of all recursive languages.

R, RE, and coRE (concluded)

- R = RE $\cap$ coRE (p. 124).
- There exist languages in RE but not in R and not in coRE.
  - Such as H (p. 114 and p. 115).
- There are languages in coRE but not in RE.
  - Such as $\bar{H}$ (p. 125).
- There are languages in neither RE nor coRE.


RE
R
coRE

Boolean Logic

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) in 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.
  - $\{x_1 = \texttt{true}, x_2 = \texttt{false}\}$ is appropriate to $x_1 \lor x_2$.

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, for any truth assignment T appropriate to both of them, $T \models (\phi_1 \Leftrightarrow \phi_2)$.

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.
  - Check if they give identical truth values under all 2^n truth assignments.

A Truth Table

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

De Morgan's[a] Laws

- **De Morgan's laws** say that

$$\begin{aligned} \neg(\phi_1 \land \phi_2) &= \neg\phi_1 \lor \neg\phi_2, \\ \neg(\phi_1 \lor \phi_2) &= \neg\phi_1 \land \neg\phi_2. \end{aligned}$$

- Here is a proof for the first law:

ϕ_1	ϕ_2	$\neg(\phi_1 \land \phi_2)$	$\neg\phi_1 \lor \neg\phi_2$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

[a] Augustus DeMorgan (1806–1871).

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 zero or more literals.[a]
- For example, $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$ is in CNF.
- Convention: An empty CNF is satisfiable, but a CNF containing an empty clause is not.

[a]Improved by Mr. Aufbu Huang (R95922070) on October 5, 2006.

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,

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

is a 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

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

Any Expression ϕ Can Be Converted into CNFs and DNFs (concluded)

$\phi = \phi_1 \wedge \phi_2$ **and a CNF is sought:** Make ϕ_1 and ϕ_2 CNFs.

$\phi = \phi_1 \wedge \phi_2$ **and a DNF is sought:** Let $\phi_1 = \bigvee_{i=1}^{n_1} A_i$ and $\phi_2 = \bigvee_{i=1}^{n_2} B_i$ be DNFs. Set

$$\phi = \bigvee_{i=1}^{n_1} \bigvee_{j=1}^{n_2} (A_i \wedge B_j).$$

An Example: Turn $\neg((a \wedge y) \vee (z \vee w))$ into a DNF

$$
\begin{aligned}
& \neg((a \wedge y) \vee (z \vee w)) \\
\stackrel{\neg(\mathrm{CNF} \vee \mathrm{CNF})}{=} \quad & \neg(((a) \wedge (y)) \vee (z \vee w)) \\
\stackrel{\neg(\mathrm{CNF})}{=} \quad & \neg((a \vee z \vee w) \wedge (y \vee z \vee w)) \\
\stackrel{\text{de Morgan}}{=} \quad & (\neg(a \vee z \vee w) \vee \neg(y \vee z \vee w)) \\
= \quad & ((\neg a \wedge \neg z \wedge \neg w) \vee (\neg y \wedge \neg z \wedge \neg w)).
\end{aligned}
$$

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) in 1922. Wittgenstein is one of the most important philosophers of all time. "God has arrived," the great economist Keynes (1883–1946) said of him on January 18, 1928. "I met him on the 5:15 train."

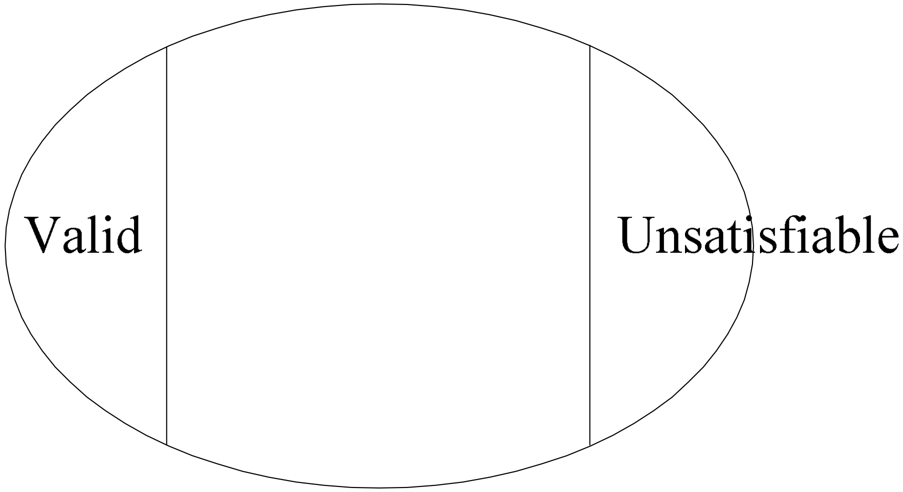
SATISFIABILITY (SAT)

- 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 exponential time 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. 242).

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 exponential time 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.

Boolean Functions

- An n-ary boolean function is a function

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

- 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.
  - $\bigvee_{T \models \phi, \text{ literal } y_i \text{ is true under } T} (y_1 \wedge \cdots \wedge y_n)$.
    * $y_1 \wedge \cdots \wedge y_n$ is the **minterm** over $\{x_1, \ldots, x_n\}$ for T.
  - The length[a] is $\leq n 2^n \leq 2^{2n}$.
  - In general, the exponential length in n cannot be avoided (p. 153)!

[a]We count the logical connectives here.

Boolean Functions (concluded)

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	1
1	0	0
1	1	1

The corresponding boolean expression:

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

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

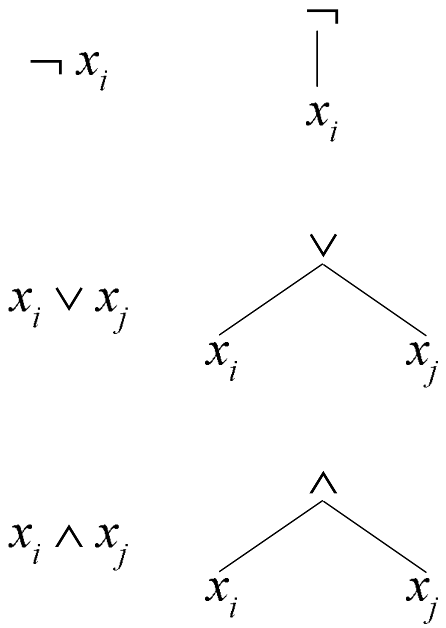
$$\{\texttt{true}, \texttt{false}, \vee, \wedge, \neg, x_1, x_2, \ldots\}.$$

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.
- The same boolean function can be computed by infinitely many boolean circuits.

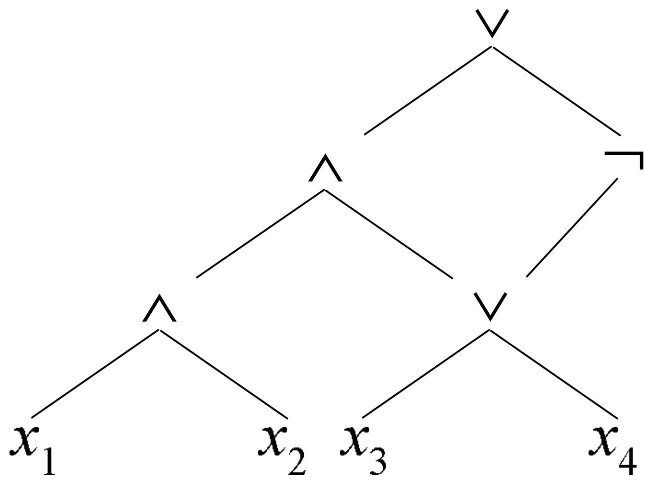
Boolean Circuits and Expressions

- 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[a]

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.
- So it suffices to prove that the number of boolean circuits with $2^n/(2n)$ or fewer gates is less than 2^{2^n}.

---

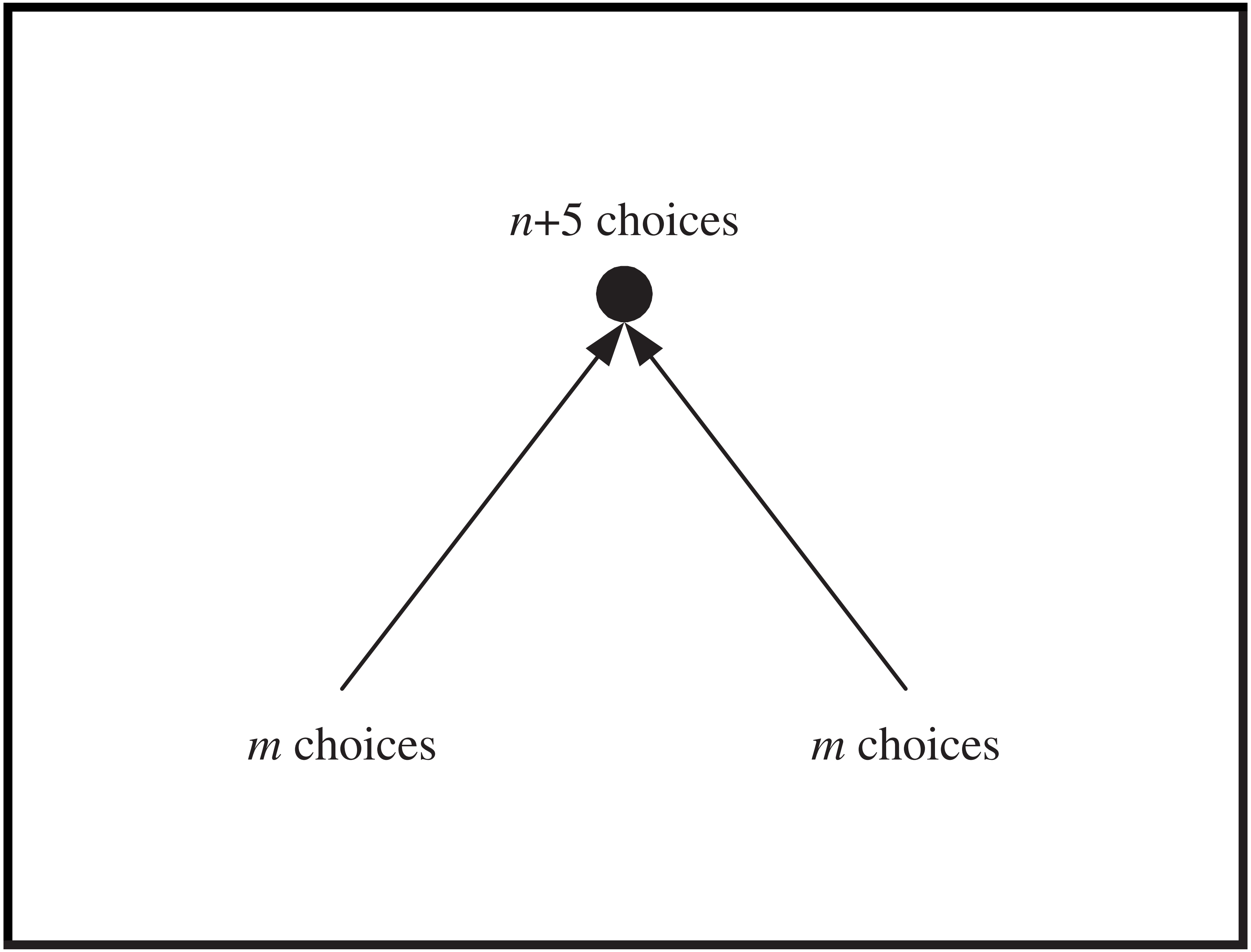
[a]Can be strengthened to "almost all boolean functions ..."

The Proof (concluded)

- There are at most $((n+5) \times m^2)^m$ boolean circuits with m or fewer gates (see next page).
- But $((n+5) \times m^2)^m < 2^{2^n}$ when $m = 2^n/(2n)$:

$$\begin{aligned} & m \log_2((n+5) \times m^2) \\ = \quad & 2^n \left(1 - \frac{\log_2 \frac{4n^2}{n+5}}{2n}\right) \\ < \quad & 2^n \end{aligned}$$

for $n \geq 2$.



n+5 choices
m choices
m choices