Complements of Recursive Languages

The **complement** of L, denoted by $\bar{L}$, is the language $\Sigma^* - L$.

Lemma 9 *If L is recursive, then so is $\bar{L}$.*

- Let L be decided by a deterministic M.
- Swap the "yes" state and the "no" state of M.
- The new machine decides $\bar{L}$.[a]

[a] Recall p. 118.

Recursive and Recursively Enumerable Languages

Lemma 10 (Kleene's theorem; Post, 1944) *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 halt on state "yes" because $x \in L$.
- If $\bar{M}$ accepts, then halt on state "no" because $x \notin L$.[a]
- The other direction is trivial.

[a] Either M or $\bar{M}$ (but not both) must accept the input and halt.

A Very Useful Corollary and Its Consequences

Corollary 11 *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 10 (p. 164), L is recursive, a contradiction.

Corollary 12 *$\bar{H}$ is not recursively enumerable.*[a]

[a] Recall that $\bar{H} \stackrel{\Delta}{=} \{ M; x : M(x) = \nearrow \}$.

R, RE, and coRE

RE: The set of all recursively enumerable languages.

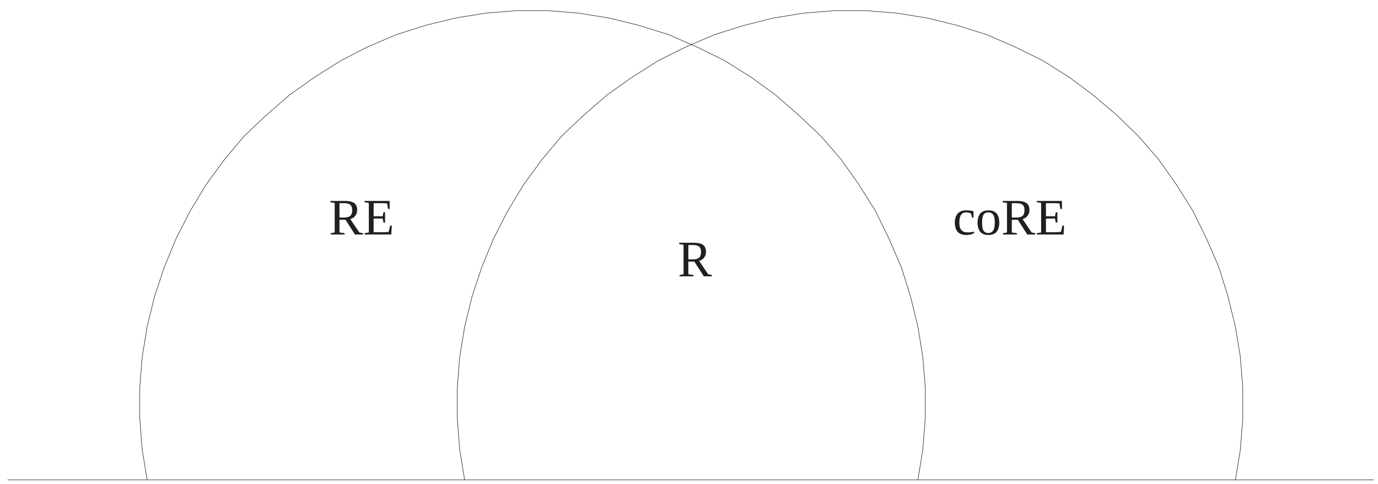
coRE: The set of all languages whose complements are recursively enumerable.

R: The set of all recursive languages.

- Note that coRE is not $\overline{\text{RE}}$.
  - $\text{coRE} \triangleq \{ L : \overline{L} \in \text{RE} \} = \{ \overline{L} : L \in \text{RE} \}$.
  - $\overline{\text{RE}} \triangleq \{ L : L \notin \text{RE} \}$.

R, RE, and coRE (concluded)

- R $=$ RE $\cap$ coRE (p. 164).
- There exist languages in RE but not in R and not in coRE.
  - Such as H (p. 144, p. 145, and p. 165).
- There are languages in coRE but not in RE.
  - Such as $\bar{H}$ (p. 165).
- There are languages in neither RE nor coRE.


RE
R
coRE

H Is Complete for RE[a]

- Let L be any recursively enumerable language.
- Assume M accepts L.
- Clearly, one can decide whether $x \in L$ by asking if $M ; x \in H$.
- Hence *all* recursively enumerable languages are reducible to H!
- H is said to be **RE-complete**.

[a] Post (1944).

Notations

- The language *accepted* by TM M is written as $L(M)$.
- If $M(x) = \nearrow$ for all x, then $L(M) = \emptyset$.
- If $M(x)$ is never "yes" nor $\nearrow$ (as required by the definition of acceptance), we also let $L(M) = \emptyset$.

Nontrivial Properties of Sets in RE

- A **property** of the recursively enumerable languages can be defined by the set $\mathcal{C}$ of all the recursively enumerable languages that satisfy it.
  - The property of *finite* recursively enumerable languages is
  $$\{\, L : L = L(M) \text{ for a TM } M,\, L \text{ is finite} \,\}.$$
  - The property of *recursiveness* is
  $$\{\, L : L = L(M) \text{ for a TM } M,\, L \text{ is recursive} \,\}.$$

Nontrivial Properties of Sets in RE (continued)

- A property is **trivial** if $\mathcal{C} = \text{RE}$ or $\mathcal{C} = \emptyset$.
  - Answer to a trivial property (about the language a TM accepts) is either always "yes"or always "no."
  - It is either possessed by *all* recursively enumerable languages or by *none*.
- Here is a trivial property (always yes): Does the TM accept a recursively enumerable language?[a]
- Here is a trivial property (always no): Does the TM accept a language that is finite and infinite?

[a]Or, $L(M) \in \text{RE}$? Formally, $\{ L : L = L(M) \text{ for a TM } M, L \in \text{RE} \}$.

Nontrivial Properties of Sets in RE (continued)

- A property is **nontrivial** if $\mathcal{C} \neq \text{RE}$ and $\mathcal{C} \neq \emptyset$.
  - In other words, answer to a nontrivial property is "yes" for some TMs and "no" for others.
  - It is possessed by *some* recursively enumerable languages but *not* by *all*.
- Here is a nontrivial property: Does the TM accept an empty language?[a]
  - Some machines do, but some machines do not.

---

[a] Or, $L(M) = \emptyset$? That is, does it go into an infinite loop on all inputs?

Nontrivial Properties of Sets in RE (concluded)

- Determining whether a property is trivial may not be trivial!
- Up to now, all nontrivial properties (of recursively enumerable languages) are undecidable.[a]
- In fact, Rice's theorem confirms that.

[a] Such as the universal halting problem H^* on p. 161.

Rice's Theorem

Theorem 13 (Rice, 1956) *Suppose* $\mathcal{C} \neq \emptyset$ *and* $\mathcal{C} \subsetneq$ *RE.*[a] *Then the question "*$L(M) \in \mathcal{C}$*?" is undecidable.*

- Note that the input is a TM program M.
- Assume that $\emptyset \notin \mathcal{C}$ (otherwise, repeat the proof for $\text{RE} - \mathcal{C}$.
- 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. 144).

---

[a] A nontrivial property, i.e.

The Proof (continued)

- Construct machine $M_x(y)$:

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

- On the next page, we will prove that

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

  - As a result, the halting problem is reduced to deciding $L(M_x) \in \mathcal{C}$.
  - Hence $L(M_x) \in \mathcal{C}$ must be undecidable,[a] and we are done.

---

[a]By Theorem 8 (p. 156).

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.
  - $L(M_x) = \emptyset \notin \mathcal{C}$.

Comments

- Rice's theorem is about nontrivial properties of the languages accepted by Turing machines.
- It says they are undecidable.
- Rice's theorem is *not* about Turing machines' operations themselves, such as

  Does this TM contain 5 states?
  Does this TM take more than 1,000 steps on ϵ?
- Both are decidable, and the answers are contingent.

Comments (concluded)

- Rather, it is about

  Does this TM accept a language acceptable by one that contains 5 states?
  Does this TM accept a language acceptable by one that takes more than 1,000 steps on ϵ?

- Because both properties are nontrivial,[a] they are undecidable by Rice's theorem.

---

[a] Why?

Consequences of Rice's Theorem

Corollary 14 *The following properties of recursively enumerative sets are undecidable.*

- *Emptiness.*
- *Nonemptiness.*
- *Finiteness.*
- *Recursiveness.*
- Σ^*.
- *Regularity.*[a]
- *Context-freedom.*[b]

---

[a] Is it a regular language?

[b] Is it a context-free language?

Undecidability in Logic and Mathematics

- First-order logic is undecidable (answer to Hilbert's (1928) *Entscheidungsproblem*).[a]
- Natural numbers with addition and multiplication is undecidable.[b]
- Rational numbers with addition and multiplication is undecidable.[c]

---

[a] Church (1936).
[b] Rosser (1937).
[c] Robinson (1948).

Undecidability in Logic and Mathematics (concluded)

- Natural numbers with addition and equality is decidable and complete.[a]
- Elementary theory of groups is undecidable.[b]

---

[a] Presburger's Master's thesis (1928), his only work in logic. The direction was suggested by Tarski. Mojžesz Presburger (1904–1943) died in a concentration camp during World War II.

[b] Tarski (1949).

Julia Hall Bowman Robinson (1919–1985)

Alfred Tarski (1901–1983)

Boolean Logic

Christianity is either false or true.
— Girolamo Savonarola (1497)

Both of us had said the very same thing.
Did we both speak the truth
—or one of us did
—or neither?
— Joseph Conrad (1857–1924),
Lord Jim (1900)

Boolean Logic[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 $\phi_1 \lor \phi_2 \lor \cdots \lor \phi_n$ (**multiple conjunction**).
- $\bigwedge_{i=1}^{n} \phi_i$ stands for $\phi_1 \land \phi_2 \land \cdots \land \phi_n$ (**multiple disjunction**).

---

[a] George Boole (1815–1864) in 1847.

Boolean Logic (concluded)

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

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

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 $x_1 \lor x_2$.
  - $\{ x_2 = \texttt{true}, x_3 = \texttt{false} \}$ is not 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 $T \models \phi_2$.

Truth Table[a]

- Suppose ϕ has n boolean variables.
- A **truth table** contains 2^n rows.
- Each row corresponds to one truth assignment of the n variables and records the truth value of ϕ under it.
- A truth table can be used to prove if two boolean expressions are equivalent.
  - Just check if they give identical truth values under all appropriate truth assignments.

---

[a]Post (1921); Wittgenstein (1922). Here, 1 is used to denote `true`; 0 is used to denote `false`. This is called the **standard representation** (Beigel, 1993).

A Truth Table

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

A Second Truth Table

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

A Third Truth Table

p	$\neg p$
0	1
1	0

Proof of Equivalency by the Truth Table:

$$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$$

p	q	$p \Rightarrow q$	$\neg q \Rightarrow \neg p$
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

De Morgan's Laws[a]

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

$$\neg(\phi_1 \land \phi_2) \equiv \neg\phi_1 \lor \neg\phi_2,$$
$$\neg(\phi_1 \lor \phi_2) \equiv \neg\phi_1 \land \neg\phi_2.$$

- Here is a proof of 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) or William of Ockham (1288–1348).

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,
$$(x_1 \lor x_2) \land (x_1 \lor \neg x_2) \land (x_2 \lor x_3).$$
- Convention: An empty CNF is satisfiable, but a CNF containing an empty clause is unsatisfiable.

[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**[a] or simply **term** D_i is the conjunction of zero or more literals.

  - For example,

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

---

[a] D_i implies ϕ, thus the term.

Clauses and Implicants

- The $\bigvee$ of clauses yields a clause.
  - For example,

$$
\begin{aligned}
& (x_1 \lor x_2) \lor (x_1 \lor \neg x_2) \lor (x_2 \lor x_3) \\
= \quad & x_1 \lor x_2 \lor x_1 \lor \neg x_2 \lor x_2 \lor x_3.
\end{aligned}
$$

- The $\bigwedge$ of implicants yields an implicant.
  - For example,

$$
\begin{aligned}
& (x_1 \land x_2) \land (x_1 \land \neg x_2) \land (x_2 \land x_3) \\
= \quad & x_1 \land x_2 \land x_1 \land \neg x_2 \land x_2 \land x_3.
\end{aligned}
$$

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.
- 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.
- Apply de Morgan's laws to make a DNF for ϕ.

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

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

- Make ϕ_1 and ϕ_2 DNFs.

$\phi = \phi_1 \lor \phi_2$ and a CNF is sought:

- Turn ϕ_1 and ϕ_2 into CNFs,[a]

$$\phi_1 = \bigwedge_{i=1}^{n_1} A_i, \quad \phi_2 = \bigwedge_{j=1}^{n_2} B_j.$$

- Set

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

[a] Corrected by Mr. Chun-Jie Yang (R99922150) on November 9, 2010.

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:

- Turn ϕ_1 and ϕ_2 into DNFs,

$$\phi_1 = \bigvee_{i=1}^{n_1} A_i, \quad \phi_2 = \bigvee_{j=1}^{n_2} B_j.$$

- Set

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

An Example: Turn $\neg((a \land y) \lor (z \lor w))$ into a DNF

$$
\begin{aligned}
& \neg((a \land y) \lor (z \lor w)) \\
\stackrel{\neg(\text{CNF}\lor\text{CNF})}{=} \;& \neg(((a) \land (y)) \lor ((z \lor w))) \\
\stackrel{\neg(\text{CNF})}{=} \;& \neg((a \lor z \lor w) \land (y \lor z \lor w)) \\
\stackrel{\text{de Morgan}}{=} \;& \neg(a \lor z \lor w) \lor \neg(y \lor z \lor w) \\
\stackrel{\text{de Morgan}}{=} \;& (\neg a \land \neg z \land \neg w) \lor (\neg y \land \neg z \land \neg w).
\end{aligned}
$$

Functional Completeness

- A set of logical connectives is called **functionally complete** if every boolean expression is equivalent to one involving *only* these connectives.
- The set $\{ \neg, \lor, \land \}$ is functionally complete.
  - Every boolean expression can be turned into a CNF, which involves only $\neg$, $\lor$, and $\land$.
- The sets $\{ \neg, \lor \}$ and $\{ \neg, \land \}$ are functionally complete.[a]
  - By the above result and de Morgan's laws.
- $\{ \text{NAND} \}$ and $\{ \text{NOR} \}$ are functionally complete.[b]

---

[a] Post (1921).

[b] Peirce (c. 1880); Sheffer (1913).

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.

---

[a]Wittgenstein (1922). Wittgenstein is one of the most important philosophers of all time. Russell (1919), "The importance of 'tautology' for a definition of mathematics was pointed out to me by my former pupil Ludwig Wittgenstein, who was working on the problem. I do not know whether he has solved it, or even whether he is alive or dead." "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 (concluded)

- ϕ is **unsatisfiable** or a **contradiction** if ϕ is false under all appropriate truth assignments.
  - Or, equivalently, if $\neg\phi$ is valid (prove it).
- ϕ is a **contingency** if ϕ is neither a tautology nor a contradiction.

Ludwig Wittgenstein (1889–1951)

Wittgenstein (1922), "Whereof one cannot speak, thereof one must be silent."

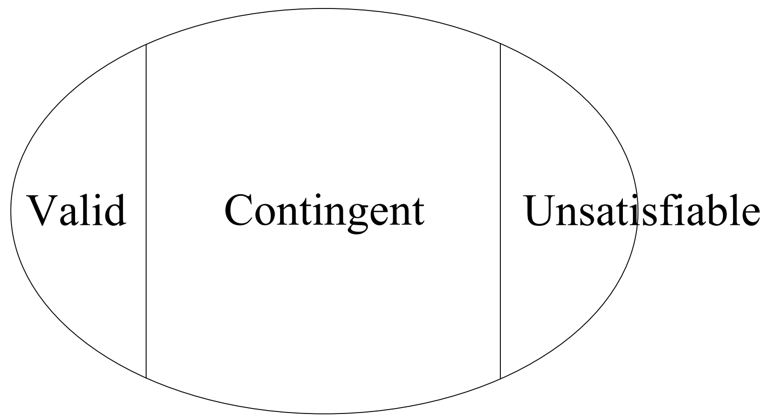
SATISFIABILITY (SAT)

- 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. 120).
- A most important problem in settling the "P $\stackrel{?}{=}$ NP" problem (p. 332).

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.
  - ϕ and $\neg\phi$ are basically of the same length.
  - So UNSAT and VALIDITY have the same complexity.
- Both are solvable in exponential time by the truth table method.

Relations among SAT, UNSAT, and VALIDITY



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

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.
  - We can assign `true` or `false` to f for each of the 2^n truth assignments.

Boolean Functions (continued)

Assignment	Truth value
1	`true` or `false`
2	`true` or `false`
$\vdots$	$\vdots$
2^n	`true` or `false`

- A boolean expression expresses a boolean function.
  - Think of its truth values under all possible truth assignments.

Boolean Functions (continued)

- A boolean function expresses a boolean expression.
  - $\bigvee_{T \models \phi, \text{ literal } y_i \text{ is true in "row" } T} (y_1 \wedge \cdots \wedge y_n)$.[a]
    - The implicant $y_1 \wedge \cdots \wedge y_n$ is called the **minterm** over $\{ x_1, \ldots, x_n \}$ for T.
  - The size[b] is $\leq n2^n \leq 2^{2n}$.
  - This DNF is optimal for the parity function, for example.[c]

---

[a] Similar to **programmable logic array**. This is called the **table lookup representation** (Beigel, 1993).

[b] We count only the literals here.

[c] Du & Ko (2000).

Boolean Functions (continued)

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 Functions (concluded)

Corollary 15 *Every n-ary boolean function can be expressed by a boolean expression of size $O(n2^n)$.*

- In general, the exponential length in n cannot be avoided (p. 221).
- The size of the truth table is also $O(n2^n)$.[a]

[a]There are 2^n n-bit strings.

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

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

  – There are $n + 5$ sorts.

Boolean Circuits (concluded)

- Gates with a 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.
- A boolean function can be realized by *infinitely many* equivalent boolean circuits.

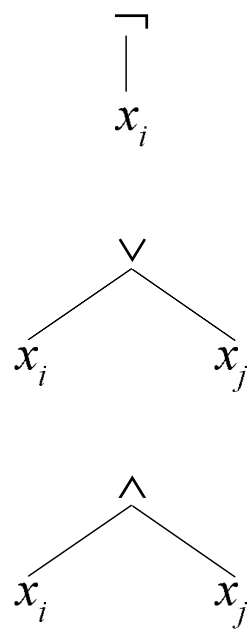
Boolean Circuits and Expressions

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

$\neg x_i$

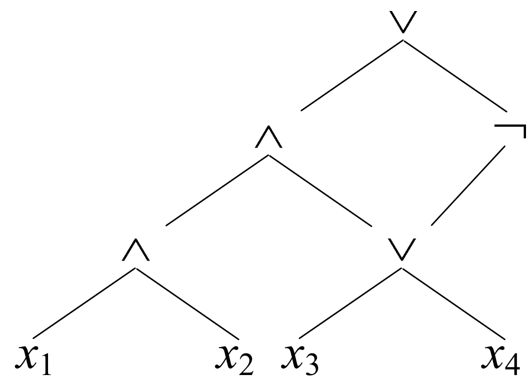
$x_i \vee x_j$

$x_i \wedge x_j$



An Example

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



- Circuits are potentially more economical because of the possibility of "sharing."[a]

[a]But see p. 294 for an efficient equivalent boolean expression. Contributed by Mr. Han-Ting Chen (`R10922073`) on October 14, 2021.

CIRCUIT SAT and CIRCUIT VALUE

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

- CIRCUIT SAT $\in$ NP: Guess a truth assignment and then evaluate the circuit.[a]

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

- CIRCUIT VALUE $\in$ P: Evaluate the circuit from the input gates gradually towards the output gate.

---

[a]Essentially the same algorithm as the one on p. 120.

Some[a] Boolean Functions Need Exponential Circuits[b]

Theorem 16 *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 (p. 211).
- So it suffices to prove that there are fewer than 2^{2^n} boolean circuits with up to $2^n/(2n)$ gates.

---

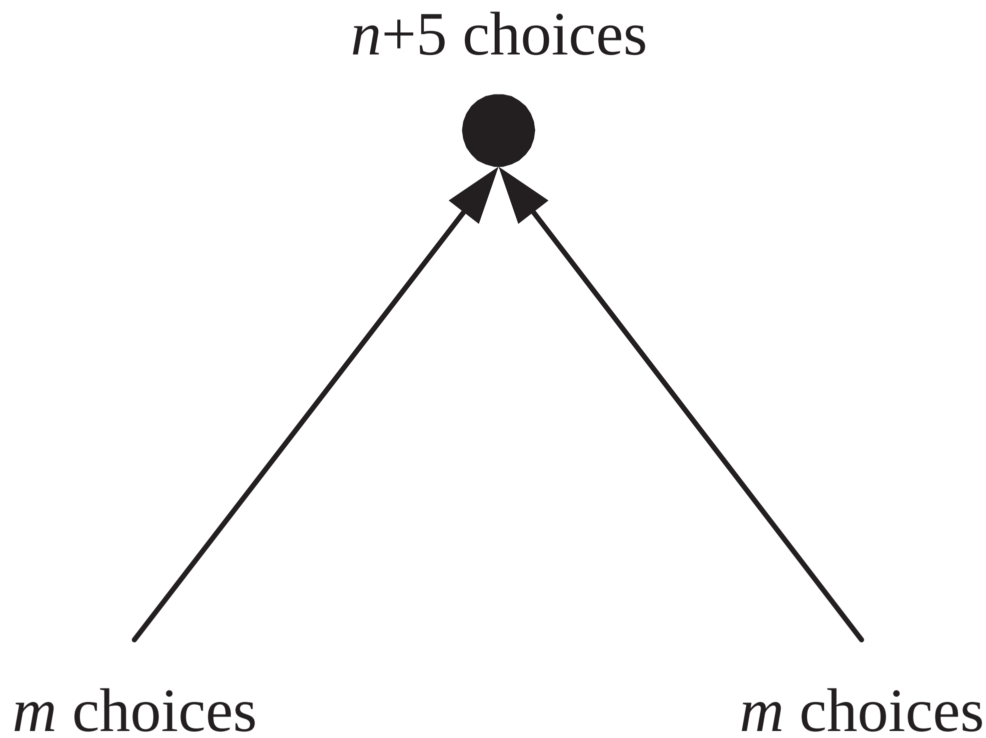
[a] Can be strengthened to "Almost all" (Lupanov, 1958).
[b] Riordan & Shannon (1942); Shannon (1949).

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 $m = 2^n/(2n)$:

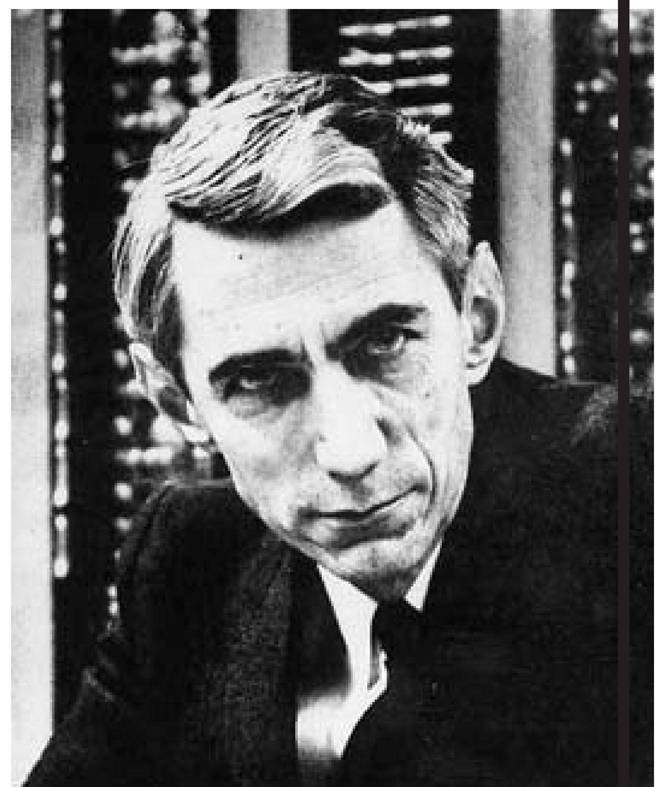
$$\begin{aligned} & m \log_2((n+5) \times m^2) \\ = \quad & 2^n \left(1 - \frac{\log_2 \frac{4n^2}{n+5}}{2n} \right) \\ < \quad & 2^n \end{aligned}$$

for $n \geq 2$.



n+5 choices
m choices
m choices

Claude Elwood Shannon (1916–2001)



Howard Gardner (1987), "[Shannon's master's thesis is] possibly the most important, and also the most famous, master's thesis of the century."

Comments

- The lower bound $2^n/(2n)$ is rather tight because an upper bound is $n2^n$ (p. 213).
- The proof counted the number of circuits.
  - Some circuits may not be valid at all.
  - Different circuits may also compute the same function.
- Both are fine because we only need an upper bound on the number of circuits.
- We do not need to consider the *outgoing* edges because they have been counted as incoming edges.[a]

[a] If you prove the theorem by considering outgoing edges, the bound will not be good. (Try it!)