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 $x_1 \vee x_2$.

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.
  - 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

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

- Here is a proof for the first law:

ϕ_1	ϕ_2	$\neg(\phi_1 \wedge \phi_2)$	$\neg\phi_1 \vee \neg\phi_2$
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 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.
- Convention: An empty CNF is satisfiable, but a CNF containing an empty clause is not.

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 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 CNFs. 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

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

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."

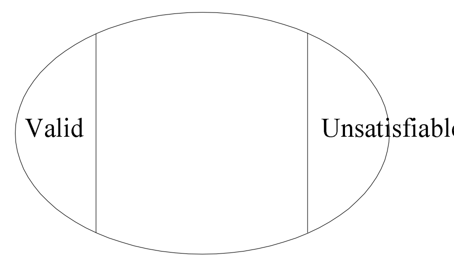
SATISFIABILITY (SAT)

- 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 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. 51).
- A most important problem in answering the P = NP problem (p. 142).

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.

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.
  - $\bigvee_{T \models \phi,\text{ literal } y_i \text{ is true under } T} (y_1 \wedge \cdots \wedge y_n)$.
    - $y_1 \wedge \cdots \wedge y_n$ is the **minterm** over $\{x_1, \ldots, x_n\}$ for T.
  - The length[a] is $\leq n2^n \leq 2^{2n}$.
  - In general, the exponential length in n cannot be avoided!

[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:

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

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

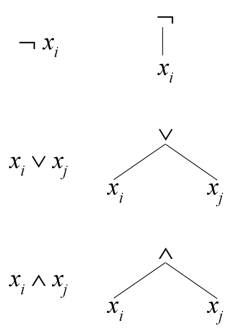
$$\{\texttt{true}, \texttt{false}, \vee, \wedge, \neg, x_1, x_2, \ldots\}.$$

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.

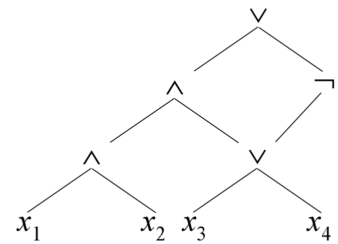
Boolean Circuits and Expressions

- 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.

Relations between Complexity Classes

Important Time Complexity Classes

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

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

Important Time Complexity Classes (concluded)

$$\begin{aligned} \text{P} &= \text{TIME}(n^k), \\ \text{NP} &= \text{NTIME}(n^k), \\ \text{E} &= \text{TIME}(2^{kn}), \\ \text{EXP} &= \text{TIME}(2^{n^k}), \end{aligned}$$

Reductions and Completeness

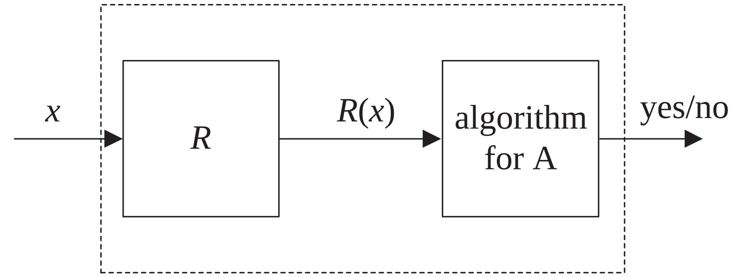
Degrees of Difficulty

- When is a problem more difficult than another?
- B **reduces to** A if there is a transformation R which for every input x of B yields an equivalent input $R(x)$ of A.
  - The answer to x for B is the same as the answer to $R(x)$ for A.
  - There must be restrictions on the complexity of computing R.
  - Otherwise, $R(x)$ might as well solve B.

Degrees of Difficulty (concluded)

- Problem A is at least as hard as problem B if B reduces to A.
- This makes intuitive sense: If A is able to solve your problem B, then A must be at least as powerful.

Reduction



Solving problem B by calling the algorithm for problem *once* and *without* further processing its answer.

Comments[a]

- Suppose B reduces to A via a transformation R.
- The input x is an instance of B.
- The output $R(x)$ is an instance of A.
- $R(x)$ may not span all possible instances of A.
- So some instances of A may never appear in the reduction.

[a] Contributed by Mr. Ming-Feng Tsai (`D92922003`) on October 29, 2003.

Reduction between Languages

- Language L_1 is **reducible to** L_2 if there is a function R computable by a deterministic TM in polynomial time.
- Furthermore, for all inputs x, $x \in L_1$ if and only if $R(x) \in L_2$.
- R is said to be a **reduction** from L_1 to L_2.
- If R is a reduction from L_1 to L_2, then $R(x) \in L_2$ is a legitimate algorithm for $x \in L_1$.

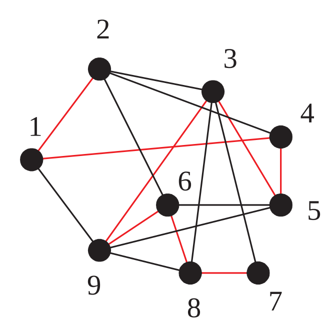
HAMILTONIAN PATH

- A **Hamiltonian path** of a graph is a path that visits every node of the graph exactly once.
- Suppose graph G has n nodes: $1, 2, \ldots, n$.
- A Hamiltonian path can be expressed as a permutation π of $\{ 1, 2, \ldots, n \}$ such that
  - $\pi(i) = j$ means the ith position is occupied by node j.
  - $(\pi(i), \pi(i+1)) \in G$ for $i = 1, 2, \ldots, n-1$.
- HAMILTONIAN PATH asks if a graph has a Hamiltonian path.

Reduction of HAMILTONIAN PATH to SAT

- Given a graph G, we shall construct a CNF $R(G)$ such that $R(G)$ is satisfiable if and only if G has a Hamiltonian path.
- $R(G)$ has n^2 boolean variables x_{ij}, $1 \leq i, j \leq n$.
- x_{ij} means

  the ith position in the Hamiltonian path is occupied by node j.



$x_{12} = x_{21} = x_{34} = x_{45} = x_{53} = x_{69} = x_{76} = x_{88} = x_{97} = 1.$

The Clauses of $R(G)$ and Their Intended Meanings

1. Each node j must appear in the path.
   - $x_{1j} \vee x_{2j} \vee \cdots \vee x_{nj}$ for each j.
2. No node j appears twice in the path.
   - $\neg x_{ij} \vee \neg x_{kj}$ for all i, j, k with $i \neq k$.
3. Every position i on the path must be occupied.
   - $x_{i1} \vee x_{i2} \vee \cdots \vee x_{in}$ for each i.
4. No two nodes j and k occupy the same position in the path.
   - $\neg x_{ij} \vee \neg x_{ik}$ for all i, j, k with $j \neq k$.
5. Nonadjacent nodes i and j cannot be adjacent in the path.
   - $\neg x_{ki} \vee \neg x_{k+1,j}$ for all $(i, j) \notin G$ and $k = 1, 2, \ldots, n-1$.

The Proof

- $R(G)$ contains $O(n^3)$ clauses.
- $R(G)$ can be computed efficiently (simple exercise).
- Suppose $T \models R(G)$.
- From Clauses of 1 and 2, for each node j there is a unique position i such that $T \models x_{ij}$.
- From Clauses of 3 and 4, for each position i there is a unique node j such that $T \models x_{ij}$.
- So there is a permutation π of the nodes such that $\pi(i) = j$ if and only if $T \models x_{ij}$.

The Proof (concluded)

- Clauses of 5 furthermore guarantees that $(\pi(1), \pi(2), \ldots, \pi(n))$ is a Hamiltonian path.
- Conversely, suppose G has a Hamiltonian path

$$(\pi(1), \pi(2), \ldots, \pi(n)),$$

where π is a permutation.

- Clearly, the truth assignment

$$T(x_{ij}) = \texttt{true} \text{ if and only if } \pi(i) = j$$

satisfies all clauses of $R(G)$.

A Comment[a]

- An answer to "Is $R(G)$ is satisfiable?" does answer "Is G Hamiltonian?"
- But a positive answer does not give a Hamiltonian path for G.
  - Providing witness is not a requirement of reduction.
- A positive answer to "Is $R(G)$ is satisfiable?" plus a satisfying truth assignment does provide us with a Hamiltonian path for G.

[a]Contributed by Ms. Amy Liu (`J94922016`) on May 29, 2006.

Reduction of REACHABILITY to CIRCUIT VALUE

- Note that both problems are in P.
- Given a graph $G = (V, E)$, we shall construct a *variable-free* circuit $R(G)$.
- The output of $R(G)$ is true if and only if there is a path from node 1 to node n in G.
- Idea: the Floyd-Warshall algorithm.

The Gates

- The gates are
  - g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$.
  - h_{ijk} with $1 \leq i, j, k \leq n$.
- g_{ijk}: There is a path from node i to node j without passing through a node bigger than k.
- h_{ijk}: There is a path from node i to node j passing through k but not any node bigger than k.
- Input gate $g_{ij0} = \texttt{true}$ if and only if $i = j$ or $(i, j) \in E$.

The Construction

- h_{ijk} is an AND gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$, where $k = 1, 2, \ldots, n$.
- g_{ijk} is an OR gate with predecessors $g_{i,j,k-1}$ and $h_{i,j,k}$, where $k = 1, 2, \ldots, n$.
- g_{1nn} is the output gate.
- Interestingly, $R(G)$ uses no $\neg$ gates: It is a **monotone circuit**.

Reduction of CIRCUIT SAT to SAT

- Given a circuit C, we shall construct a boolean expression $R(C)$ such that $R(C)$ is satisfiable if and only if C is satisfiable.
  - $R(C)$ will turn out to be a CNF.
- The variables of $R(C)$ are those of C plus g for each gate g of C.
- Each gate of C will be turned into equivalent clauses of $R(C)$.
- Recall that clauses are $\wedge$-ed together.

The Clauses of $R(C)$

g is a variable gate x: Add clauses $(\neg g \vee x)$ and $(g \vee \neg x)$.

- Meaning: $g \Leftrightarrow x$.

g is a `true` gate: Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

g is a `false` gate: Add clause $(\neg g)$.

- Meaning: g must be false to make $R(C)$ true.

g is a $\neg$ gate with predecessor gate h: Add clauses $(\neg g \vee \neg h)$ and $(g \vee h)$.

- Meaning: $g \Leftrightarrow \neg h$.

The Clauses of $R(C)$ (concluded)

g is a $\vee$ gate with predecessor gates h and h': Add clauses $(\neg h \vee g)$, $(\neg h' \vee g)$, and $(h \vee h' \vee \neg g)$.

- Meaning: $g \Leftrightarrow (h \vee h')$.

g is a $\wedge$ gate with predecessor gates h and h': Add clauses $(\neg g \vee h)$, $(\neg g \vee h')$, and $(\neg h \vee \neg h' \vee g)$.

- Meaning: $g \Leftrightarrow (h \wedge h')$.

g is the output gate: Add clause (g).

- Meaning: g must be true to make $R(C)$ true.

Composition of Reductions

Proposition 9 *If R_{12} is a reduction from L_1 to L_2 and R_{23} is a reduction from L_2 to L_3, then the composition $R_{12} \circ R_{23}$ is a reduction from L_1 to L_3.*

- Clearly $x \in L_1$ if and only if $R_{23}(R_{12}(x)) \in L_3$.
- It is also clear that $R_{12} \circ R_{23}$ can be computed in polynomial time.