BOOLEAN LOGIC

- Syntax
- Semantics
- Normal forms
- Satisfiability and validity
- Boolean functions and expressions
- Boolean circuits

(C. Papadimitriou: *Computational complexity*, Chapter 4)

Motivation

- Logic involves interesting computational problems.
- Logic is "the calculus of computer science":
  digital circuit design, programming language semantics, specification and verification, constraint programming, logic programming, databases, artificial intelligence, knowledge representation, machine learning, ...
- In computational complexity theory:
  Computational problems from logic are of central importance; they can be used to express computation at various levels.
  This leads to important connections between complexity concepts and actual computational problems.

1. Syntax

- The syntax of Boolean logic (i.e. the set of well-formed Boolean expressions) is based on the following symbols:
  - Boolean *variables* (or *atoms*): $X = \{x_1, x_2, \ldots\}$.
  - Boolean *connectives*: $\vee$ (or), $\wedge$ (and) , and $\neg$ (negation).
- The set of Boolean expressions (formulae) is the smallest set such that all Boolean variables are Boolean expressions and if ϕ_1 and ϕ_2 are Boolean expressions, so are $\neg\phi_1$, $(\phi_1 \wedge \phi_2)$, and $(\phi_1 \vee \phi_2)$.
- An expression of the form x_i or $\neg x_i$ is called a *literal* where x_i is a Boolean variable.

Example. $((x_1 \vee x_2) \wedge \neg x_3)$ is a Boolean expression but $((x_1 \vee x_2)\neg x_3)$ is not.

Some notational conventions

- Simplified notation: $(((x_1 \vee \neg x_3) \vee x_2) \vee (x_4 \vee (x_2 \vee x_5)))$ is written as $x_1 \vee \neg x_3 \vee x_2 \vee x_4 \vee x_2 \vee x_5$ or $x_1 \vee \neg x_3 \vee x_2 \vee x_4 \vee x_5$.
- Disjunctions and conjunctions involving n members:
  - $\bigvee_{i=1}^{n} \varphi_i$ stands for $\varphi_1 \vee \cdots \vee \varphi_n$.
  - $\bigwedge_{i=1}^{n} \varphi_i$ stands for $\varphi_1 \wedge \cdots \wedge \varphi_n$.
- Frequently appearing abbreviations:
  - An implication $\phi_1 \rightarrow \phi_2$ stands for $\neg\phi_1 \vee \phi_2$.
  - An equivalence $\phi_1 \leftrightarrow \phi_2$ stands for $(\neg\phi_1 \vee \phi_2) \wedge (\neg\phi_2 \vee \phi_1)$.

2. Semantics

How to interpret Boolean expressions?

- Boolean expressions are propositions that are either true or false.

  They speak about a world where certain atomic proposition (Boolean variables) are either true or false.

  This induces truth values for Boolean expressions as follows.

- A truth assignment T is mapping from a finite subset $X' \subset X$ to the set of truth values $\{\mathbf{true}, \mathbf{false}\}$.

- Let $X(\phi)$ be the set of Boolean *variables appearing in* ϕ.

  Definition. A truth assignment $T : X' \to \{\mathbf{true}, \mathbf{false}\}$ is *appropriate* to ϕ if $X(\phi) \subseteq X'$.

Satisfaction relation

- Let a truth assignment $T : X' \to \{\mathbf{true}, \mathbf{false}\}$ be appropriate to ϕ, i.e., $X(\phi) \subseteq X'$.

- $T \models \phi$ (T *satisfies* ϕ) is defined inductively as follows:

  If ϕ is a variable from X', then $T \models \phi$ iff $T(\phi) = \mathbf{true}$.

  If $\phi = \neg\phi_1$, then $T \models \phi$ iff $T \not\models \phi_1$.

  If $\phi = \phi_1 \wedge \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ and $T \models \phi_2$.

  If $\phi = \phi_1 \vee \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ or $T \models \phi_2$.

Example. Let $T(x_1) = \mathbf{true}$, $T(x_2) = \mathbf{false}$.
Then $T \models x_1 \vee x_2$ but $T \not\models (x_1 \vee \neg x_2) \wedge (\neg x_1 \wedge x_2)$.

Logical equivalence

Definition. Expressions ϕ_1 and ϕ_2 are logically *equivalent* ($\phi_1 \equiv \phi_2$) iff for all truth assignments T appropriate to both of them,

$$T \models \phi_1 \text{ iff } T \models \phi_2.$$

Example.

$(\phi_1 \vee \phi_2) \equiv (\phi_2 \vee \phi_1)$
$((\phi_1 \wedge \phi_2) \wedge \phi_3) \equiv (\phi_1 \wedge (\phi_2 \wedge \phi_3))$
$\neg\neg\phi \equiv \phi$
$((\phi_1 \wedge \phi_2) \vee \phi_3) \equiv ((\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3))$
$\neg(\phi_1 \wedge \phi_2) \equiv (\neg\phi_1 \vee \neg\phi_2)$
$\neg(\phi_1 \vee \phi_2) \equiv (\neg\phi_1 \wedge \neg\phi_2)$
$(\phi_1 \vee \phi_1) \equiv \phi_1$

3. Normal Forms

- The most frequently used normal forms for Boolean expressions are conjunctive and disjuntive normal forms (CNF/DNF).

- These forms are defined by

  CNF: $(l_{11} \vee \cdots \vee l_{1n_1}) \wedge \cdots \wedge (l_{m1} \vee \cdots \vee l_{mn_m})$

  DNF: $(l_{11} \wedge \cdots \wedge l_{1n_1}) \vee \cdots \vee (l_{m1} \wedge \cdots \wedge l_{mn_m})$

  where each l_{ij} is a literal (Boolean variable or its negation).

- A disjunction $l_1 \vee \cdots \vee l_n$ of literals is called a *clause*.

- A conjunction $l_1 \wedge \cdots \wedge l_n$ of literals is called an *implicant*.

- We can assume that normal forms do not have repeated clauses/implicants or repeated literals in clauses/implicants.

  Example. $(\neg x_1 \vee \neg x_1 \vee x_2) \equiv (\neg x_1 \vee x_2)$.

Theorem. Every Boolean expression is equivalent to one in conjunctive (disjunctive) normal form.

CNF/DNF transformation

Any Boolean expression can be transformed into CNF/DNF as follows.

- Remove $\leftrightarrow$ and $\rightarrow$:

$$\alpha \leftrightarrow \beta \quad \rightsquigarrow \quad (\neg\alpha \lor \beta) \land (\neg\beta \lor \alpha) \qquad (1)$$
$$\alpha \rightarrow \beta \quad \rightsquigarrow \quad \neg\alpha \lor \beta \qquad (2)$$

- Push negations in front of Boolean variables:

$$\neg\neg\alpha \quad \rightsquigarrow \quad \alpha \qquad (3)$$
$$\neg(\alpha \lor \beta) \quad \rightsquigarrow \quad \neg\alpha \land \neg\beta \qquad (4)$$
$$\neg(\alpha \land \beta) \quad \rightsquigarrow \quad \neg\alpha \lor \neg\beta \qquad (5)$$

☞ The result is a mixed conjunction and disjunction of literals.

CNF/DNF transformation—cont'd

The next phase depends on the normal form being pursued:

- For a CNF, move $\land$ connectives outside $\lor$ connectives:

$$\alpha \lor (\beta \land \gamma) \quad \rightsquigarrow \quad (\alpha \lor \beta) \land (\alpha \lor \gamma) \qquad (6)$$
$$(\alpha \land \beta) \lor \gamma \quad \rightsquigarrow \quad (\alpha \lor \gamma) \land (\beta \lor \gamma) \qquad (7)$$

- For a DNF, move $\lor$ connectives outside $\land$ connectives:

$$\alpha \land (\beta \lor \gamma) \quad \rightsquigarrow \quad (\alpha \land \beta) \lor (\alpha \land \gamma) \qquad (8)$$
$$(\alpha \lor \beta) \land \gamma \quad \rightsquigarrow \quad (\alpha \land \gamma) \lor (\beta \land \gamma) \qquad (9)$$

Note: Normal forms can be exponentially bigger than the original expression in the worst case.

Example

Transform $(x_1 \lor x_2) \rightarrow (x_2 \leftrightarrow x_3)$ into CNF.

$(x_1 \lor x_2) \rightarrow (x_2 \leftrightarrow x_3) \quad (1)$

$\neg(x_1 \lor x_2) \lor (x_2 \leftrightarrow x_3) \quad (2)$

$\neg(x_1 \lor x_2) \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \quad (4)$

$(\neg x_1 \land \neg x_2) \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \quad (7)$

$(\neg x_1 \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \land (\neg x_2 \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \quad (6)$

$((\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_3 \lor x_2)))$
$\quad \land \ (\neg x_2 \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \quad (6)$

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

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

4. Satisfiability and Validity

- A Boolean expression ϕ is *satisfiable* iff there is a truth assignment T appropriate to it such that $T \models \phi$.
- A Boolean expression ϕ is *valid*/*tautology* (denoted by $\models \phi$) iff for every truth assignment T appropriate to it, $T \models \phi$.
- The interconnection of satisfiability and validity:

$$\models \phi \text{ iff } \neg\phi \text{ is unsatisfiable.}$$

- Moreover, for any Boolean expressions ψ_1 and ψ_2,

$$\psi_1 \equiv \psi_2 \text{ iff } \models \psi_1 \leftrightarrow \psi_2 \text{ iff } \neg(\psi_1 \leftrightarrow \psi_2) \text{ is unsatisfiable.}$$

☞ Satisfiability forms a fundamental computational problem.

Satisfiability Problem

- *SAT problem*: Given φ in CNF, is φ satisfiable?

  Example. $(x_1 \lor \neg x_2) \land \neg x_1$ is satisfiable
  but $(x_1 \lor \neg x_2) \land \neg x_1 \land x_2$ is unsatisfiable.

- SAT can be solved in $O(n^2 2^n)$ time (e.g., truth table method).

- SAT $\in$ **NP** but SAT $\in$ **P** remains open!

> A nondeterministic Turing machine for $\varphi \in$ SAT:
> **for** all variables x in φ **do**
>   choose nondeterministically: $T(x) :=$ **true** or $T(x) :=$ **false**;
> **if** $T \models \varphi$ **then** return "yes" **else** return "no"

Horn clauses

- An interesting special case of SAT concerns *Horn clauses*, i.e., clauses (disjunction of literals) with *at most one positive literal*.

  Example. $\neg x_1 \lor x_2 \lor \neg x_3$ and $\neg x_1 \lor \neg x_3$, x_2 are Horn clauses but $\neg x_1 \lor x_2 \lor x_3$ is not.

- A Horn clause with a positive literal is called an *implication* and can be written as $(x_1 \land x_3) \rightarrow x_2$
  (or $\rightarrow x_2$ when there are no negative literals).

- HORNSAT problem:
  Given a conjunction of Horn clauses, is it satisfiable?

Polynomial Time Algorithm for HORNSAT

Algorithm *hornsat*(S)
/* Determines whether $S \in$ HORNSAT */
$T := \emptyset$ /* T is the set of true atoms */
repeat
    if there is an implication $(x_1 \land x_2 \land \cdots \land x_n) \rightarrow y$ in S
        such that $\{x_1, \ldots, x_n\} \subseteq T$ but $y \notin T$ **then**
        $T := T \cup \{y\}$
until T does not change
if for all purely negative clauses $\neg x_1 \lor \cdots \lor \neg x_n$ in S,
    there is some literal $\neg x_i$ such that $x_i \notin T$ **then**
    return S is satisfiable
else return S is not satisfiable

☞ HORNSAT $\in$ **P**.

5. Boolean Functions and Expressions

- An n-ary Boolean function is a mapping $\{\mathbf{true}, \mathbf{false}\}^n \rightarrow \{\mathbf{true}, \mathbf{false}\}$.

  Example. The connectives $\lor$, $\land$, $\rightarrow$, and $\leftrightarrow$ can be viewed as binary Boolean functions and $\neg$ is a unary function.

- Similarly, any Boolean expression ϕ can be interpreted as an n-ary Boolean function f_ϕ where $n = |X(\phi)|$.

- A Boolean expression ϕ with variables $x_1, \ldots, x_n$ *expresses* the n-ary function f if for any n-tuple of truth values $\mathbf{t} = (t_1, \ldots, t_n)$,

$$f(\mathbf{t}) = \begin{cases} \mathbf{true}, & \text{if } T \models \phi. \\ \mathbf{false}, & \text{if } T \not\models \phi. \end{cases}$$

  where T satisfies $T(x_i) = t_i$ for every $i = 1, \ldots, n$.

Proposition. Any n-ary Boolean function f can be expressed as a Boolean expression ϕ_f involving variables $x_1, \ldots, x_n$.

- The idea: model the rows of the truth table of f giving **true** as a disjunction of conjunctions.
- Let F be the set of all n-tuples $\mathbf{t} = (t_1, \ldots, t_n)$ with $f(\mathbf{t}) = \mathbf{true}$.
- For each $\mathbf{t}$, let $D_{\mathbf{t}}$ be a conjunction of literals x_i if $t_i = \mathbf{true}$ and $\neg x_i$ if $t_i = \mathbf{false}$.
- Let $\phi_f = \bigvee_{\mathbf{t} \in F} D_{\mathbf{t}}$
- Note that ϕ_f may get big in the worst case: $O(n2^n)$.

☞ Not all Boolean functions can be expressed concisely.

Example.

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	0

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

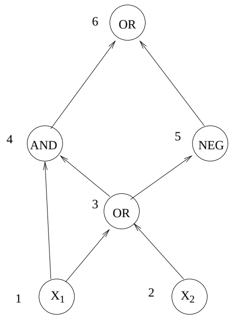
6. Boolean Circuits

A more economical way to represent Boolean functions

Syntax:

- A Boolean circuit is a graph $C = (V, E)$ where $V = \{1, 2, \ldots, n\}$ is the set of gates and C must be acyclic ($i < j$ for all edges $(i, j) \in E$) and
- all gates i have a sort $s(i) \in \{\mathbf{true}, \mathbf{false}, \wedge, \vee, \neg\} \cup \{x_1, x_2, \ldots\}$.
  - If $s(i) \in \{\mathbf{true}, \mathbf{false}\} \cup \{x_1, x_2, \ldots\}$, the indegree of i is 0 (inputs).
  - If $s(i) = \neg$, the indegree of i 1.
  - If $s(i) \in \{\vee, \wedge\}$, the indegree of i is 2.
- Node n is the output of the circuit.

Example.



In the figure sorts denoted by OR ($\vee$), AND ($\wedge$), NEG ($\neg$),

Semantics

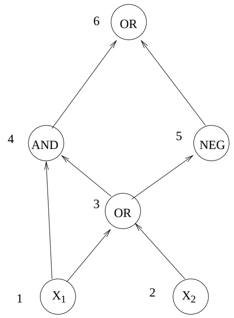
A truth assignment is a function $T : X(C) \rightarrow \{\mathbf{true}, \mathbf{false}\}$ where $X(C)$ is the set of variables appearing in a circuit C.

The truth value $T(i)$ for each gate i is defined inductively:

- If $s(i) = \mathbf{true}$, $T(i) = \mathbf{true}$ and if $s(i) = \mathbf{false}$, $T(i) = \mathbf{false}$.
- If $s(i) \in X(C)$, then $T(i) = T(s(i))$.
- If $s(i) = \neg$, then $T(i) = \mathbf{true}$ if $T(j) = \mathbf{false}$, otherwise $T(i) = \mathbf{false}$ where (j, i) is the unique edge entering i.
- If $s(i) = \wedge$, then $T(i) = \mathbf{true}$ if $T(j) = T(j') = \mathbf{true}$ else $T(i) = \mathbf{false}$ where (j, i) and (j', i) are the two edges entering i.
- If $s(i) = \vee$, then $T(i) = \mathbf{true}$ if $T(j) = \mathbf{true}$ or $T(j') = \mathbf{true}$ else $T(i) = \mathbf{false}$ where (j, i) and (j', i) are the two edges to i.
- $T(C) = T(n)$, i.e. the *value of the circuit* C.

Example.

Circuit C



Consider a truth assignment T such that $T(x_1) = T(x_2) = \mathbf{false}$.

Then

$T(1) = T(x_1) = \mathbf{false}$
$T(2) = T(x_2) = \mathbf{false}$
$T(3) = \mathbf{false}$ (as $T(1) = \mathbf{false}$, $T(2) = \mathbf{false}$)
$T(4) = \mathbf{false}$
$T(5) = \mathbf{true}$
$T(6) = \mathbf{true}$

Hence, the value of the circuit C $T(C) = T(6) = \mathbf{true}$

Boolean circuits vs. Boolean expressions

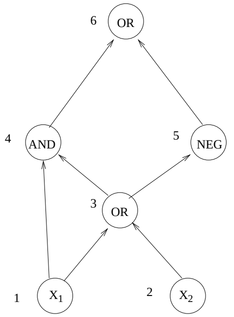
- For each Boolean circuit C, there is a corresponding Boolean expression ϕ_C.
- For each Boolean expression ϕ, there is a corresponding Boolean circuit C_ϕ such that for any T appropriate for both,

  $$T(C_\phi) = \mathbf{true} \text{ iff } T \models \phi.$$

  Idea: just introduce a new gate for each subexpression of ϕ.
- Notice that Boolean circuits allow shared subexpressions but Boolean expressions do not.

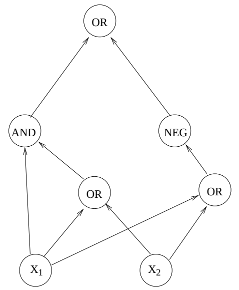
Example.

Consider the circuit C



A corresponding Boolean expression ϕ_C: $(x_1 \wedge (x_1 \vee x_2)) \vee \neg(x_1 \vee x_2)$

Boolean circuit C_{ϕ_C} for ϕ_C: $(x_1 \wedge (x_1 \vee x_2)) \vee \neg(x_1 \vee x_2)$



Note the difference: in C substructure shared.

Computational problems related with Boolean circuits

- CIRCUIT SAT:

  Given a circuit C, is there a truth assignment $T : X(C) \to \{\mathbf{true}, \mathbf{false}\}$ such that $T(C) = \mathbf{true}$?
- CIRCUIT SAT $\in \mathbf{NP}$.
- CIRCUIT VALUE:

  Given a circuit C with no variables, is it the case that $T(C) = \mathbf{true}$?
- CIRCUIT VALUE $\in \mathbf{P}$.

  (No truth assignment is needed as $X(C) = \emptyset$).

Circuits computing Boolean functions

- A Boolean circuit with variables $x_1, \ldots, x_n$ *computes* an n-ary Boolean function f if for any n-tuple of truth values $\mathbf{t} = (t_1, \ldots, t_n)$, $f(\mathbf{t}) = T(C)$ where $T(x_i) = t_i$ for $i = 1, \ldots, n$.
- Any n-ary Boolean function f can be computed by a Boolean circuit involving variables $x_1, \ldots, x_n$.
- Not every Boolean function has a concise circuit computing it.

Theorem. For any $n \geq 2$ there is an n-ary Boolean function f such that no Boolean circuit with $\frac{2^n}{2n}$ or fewer gates can compute it.

However, nobody has been able to come up with a natural family of Boolean functions that require more than a linear number of gates to compute.

Learning Objectives

- You should deeply understand the syntax and semantics of Boolean expressions — including their use in practice.
- The relationship/difference between Boolean expressions and circuits.
- Knowing the idea of representing Boolean functions in terms of Boolean expressions and circuits.
- Four computational problems related with Boolean logic and circuits: SAT, HORNSAT, CIRCUIT SAT, and CIRCUIT VALUE.