**Boolean Logic**

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

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

---

**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: $\lor$ (or), $\land$ (and), and $\neg$ (negation).
- The set of Boolean expressions (formulae) is the smallest set such that all Boolean variables are Boolean expressions and if $\phi_1$ and $\phi_2$ are Boolean expressions, so are $\neg\phi_1$, $(\phi_1 \land \phi_2)$, and $(\phi_1 \lor \phi_2)$.
- An expression of the form $x_i$ or $\neg x_i$ is called a literal where $x_i$ is a Boolean variable.

**Example.** $(x_1 \lor x_2) \land \neg x_3$ is a Boolean expression but $(\neg x_1 \lor x_2) \land x_3$ is not.

---

**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.

---

**Some notational conventions**

- Simplified notation: $(((x_1 \lor \neg x_3) \lor x_2) \lor (x_4 \lor (x_2 \lor x_5)))$ is written as $x_1 \lor \neg x_3 \lor x_2 \lor x_4 \lor x_2 \lor x_5$ or $x_1 \lor \neg x_3 \lor x_2 \lor x_4 \lor x_5$.
- Disjunctions and conjunctions involving $n$ members:
  - $\lor_{i=1}^{n} \phi_i$ stands for $\phi_1 \lor \cdots \lor \phi_n$.
  - $\land_{i=1}^{n} \phi_i$ stands for $\phi_1 \land \cdots \land \phi_n$.
- Frequently appearing abbreviations:
  - An implication $\phi_1 \rightarrow \phi_2$ stands for $\neg \phi_1 \lor \phi_2$.
  - An equivalence $\phi_1 \leftrightarrow \phi_2$ stands for $(\neg \phi_1 \lor \phi_2) \land (\neg \phi_2 \lor \phi_1)$.
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 propositions (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 $\{\text{true}, \text{false}\}$.

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

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

---

Logical equivalence

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

$T \models \phi_1$ iff $T \models \phi_2$.

**Example.**

$((\phi_1 \lor \phi_2) \land \phi_3) \equiv ((\phi_2 \lor \phi_1) \land \phi_3)$

$\neg ((\phi_1 \lor \phi_2) \land \phi_3) \equiv (\neg \phi_1 \lor (\neg \phi_2 \land \phi_3))$

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

$(\phi_1 \lor \phi_1) \equiv \phi_1$

---

3. Normal Forms

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

- These forms are defined by

  CNF: $(l_{11} \lor \cdots \lor l_{1n_1}) \land \cdots \land (l_{m1} \lor \cdots \lor l_{mn_m})$

  DNF: $(l_{i1} \land \cdots \land l_{in_i}) \lor \cdots \lor (l_{m1} \land \cdots \land l_{mn_m})$

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

- A disjunction $l_{1} \lor \cdots \lor l_{n}$ of literals is called a **clause**.

- A conjunction $l_{1} \land \cdots \land l_{n}$ 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.** $(\neg x_1 \lor \neg x_1 \lor x_2) \equiv (\neg x_1 \lor x_2)$.

**Theorem.** Every Boolean expression is equivalent to one in conjunctive (disjunctive) normal form.
Any Boolean expression can be transformed into CNF/DNF as follows.

- Remove \( \leftrightarrow \) and \( \rightarrow \):
  \[
  \begin{align*}
  \alpha \leftrightarrow \beta & \leadsto (\neg \alpha \lor \beta) \land (\neg \beta \lor \alpha) \\
  \alpha \rightarrow \beta & \leadsto \neg \alpha \lor \beta
  \end{align*}
  \]

- Push negations in front of Boolean variables:
  \[
  \begin{align*}
  \neg \neg \alpha & \leadsto \alpha \\
  \neg (\alpha \lor \beta) & \leadsto \neg \alpha \land \neg \beta \\
  \neg (\alpha \land \beta) & \leadsto \neg \alpha \lor \neg \beta
  \end{align*}
  \]

The result is a mixed conjunction and disjunction of literals.

The next phase depends on the normal form being pursued:

- For a CNF, move \( \land \) connectives outside \( \lor \) connectives:
  \[
  \begin{align*}
  \alpha \lor (\beta \land \gamma) & \leadsto (\alpha \lor \beta) \land (\alpha \lor \gamma) \\
  (\alpha \land \beta) \lor \gamma & \leadsto (\alpha \lor \gamma) \land (\beta \lor \gamma)
  \end{align*}
  \]

- For a DNF, move \( \lor \) connectives outside \( \land \) connectives:
  \[
  \begin{align*}
  \alpha \land (\beta \lor \gamma) & \leadsto (\alpha \land \beta) \lor (\alpha \land \gamma) \\
  (\alpha \lor \beta) \land \gamma & \leadsto (\alpha \land \gamma) \lor (\beta \land \gamma)
  \end{align*}
  \]

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

A Boolean expression \( \phi \) is **satisfiable** iff there is a truth assignment \( T \) appropriate to it such that \( T \models \phi \).

A Boolean expression \( \phi \) 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 \( \psi_1 \) and \( \psi_2 \),

\[ \psi_1 \equiv \psi_2 \text{ iff } \models \psi_1 \leftrightarrow \psi_2 \text{ iff } \neg (\psi_1 \leftrightarrow \psi_2) \text{ is unsatisfiable.} \]
Satisfiability Problem

- **SAT problem**: Given \( \varphi \) in CNF, is \( \varphi \) 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 \(\varphi\) do
  choose nondeterministically: \(T(x) := \text{true}\) or \(T(x) := \text{false}\);
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
 Cox HORNSAT \(\in\) P.
```

---

5. Boolean Functions and Expressions

- An \(n\)-ary Boolean function is a mapping \(\{\text{true}, \text{false}\}^n \rightarrow \{\text{true}, \text{false}\}\).

  **Example.** The connectives \(\lor, \land, \rightarrow, \text{ and } \leftrightarrow\) can be viewed as binary Boolean functions and \(\neg\) is a unary function.

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

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

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

  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 $\phi_f$ 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 $t = (t_1, \ldots, t_n)$ with $f(t) = \text{true}$.
- For each $t$, let $D_t$ be a conjunction of literals $x_i$ if $t_i = \text{true}$ and $\neg x_i$ if $t_i = \text{false}$.
- Let $\phi_f = \bigvee_{t \in F} D_t$
- Note that $\phi_f$ may get big in the worst case: $O(n2^n)$.

Not all Boolean functions can be expressed concisely.

Example.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Semantics

A truth assignment is a function $T : X(C) \rightarrow \{\text{true, false}\}$ 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 $s(i) = \text{true}$, $T(i) = \text{true}$ and if $s(i) = \text{false}$, $T(i) = \text{false}$.
- If $s(i) \in X(C)$, then $T(i) = T(s(i))$.
- If $s(i) = \neg$, then $T(i) = \text{true}$ if $T(j) = \text{false}$, otherwise $T(i) = \text{false}$ where $(j,i)$ is the unique edge entering $i$.
- If $s(i) = \land$, then $T(i) = \text{true}$ if $T(j) = T(j') = \text{true}$ else $T(i) = \text{false}$ where $(j,i)$ and $(j',i)$ are the two edges entering $i$.
- If $s(i) = \lor$, then $T(i) = \text{true}$ if $T(j) = \text{true}$ or $T(j') = \text{true}$ else $T(i) = \text{false}$ where $(j,i)$ and $(j',i)$ are the two edges to $i$.
- $T(C) = T(n)$, i.e. the value of the circuit $C$.

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 $s(i) \in \{\text{true, false}, \land, \lor, \neg\} \cup \{x_1,x_2,\ldots\}$.
  - If $s(i) \in \{\text{true, false}\} \cup \{x_1,x_2,\ldots\}$, the indegree of $i$ is 0 (inputs).
  - If $s(i) = \neg$, the indegree of $i$ is 1.
  - If $s(i) \in \{\lor, \land\}$, the indegree of $i$ is 2.
- Node $n$ is the output of the circuit.

Example.

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

Then

$T(1) = T(x_1) = \text{false}$

$T(2) = T(x_2) = \text{false}$

$T(3) = \text{false}$ (as $T(1) = \text{false}$, $T(2) = \text{false}$)

$T(4) = \text{false}$

$T(5) = \text{true}$

$T(6) = \text{true}$

Hence, the value of the circuit $C$

$T(C) = T(6) = \text{true}$
### Boolean circuits vs. Boolean expressions

- For each Boolean circuit $C$, there is a corresponding Boolean expression $\phi_C$.
- For each Boolean expression $\phi$, there is a corresponding Boolean circuit $C_\phi$ such that for any $T$ appropriate for both,
  \[ T(C_\phi) = \text{true} \iff T \models \phi. \]
  
  Idea: just introduce a new gate for each subexpression of $\phi$.

- Notice that Boolean circuits allow shared subexpressions but Boolean expressions do not.

### Computational problems related with Boolean circuits

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

- **CIRCUIT VALUE:**
  Given a circuit $C$ with no variables, is it the case that $T(C) = \text{true}$?
  - **CIRCUIT VALUE $\in$ 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
  \[ t = (t_1, \ldots, t_n), \quad f(t) = T(C) \]  
  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 $\frac{2^n}{n}$ 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.