T-79.5103 Computational Complexity Theory

Lecture 5: Boolean Logic

Aalto University
School of Science
Department of Computer Science

Spring 2016
Agenda

Boolean logic:
- syntax
- semantics
- normal forms
- satisfiability and validity
- Boolean functions and expressions
- Boolean circuits

(C. Papadimitriou: *Computational Complexity*, Chapter 4)
Motivation

- Logic is a fundamental representation language in many areas of computer science:
  digital circuit design, programming language semantics, specification and verification, constraint programming, logic programming, databases, artificial intelligence, knowledge representation, machine learning, . . .

- Logic involves interesting computational problems.

- In computational complexity theory:
  Computational problems from logic can be shown to be “universal” at various levels of computational difficulty. This leads to important insights and techniques for reasoning about computational complexity issues.
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 (formulas) is inductively defined as follows:
  - all Boolean variables are Boolean expressions,
  - if $\phi_1$ and $\phi_2$ are Boolean expressions, then so are $\neg\phi_1$, $(\phi_1 \land \phi_2)$, and $(\phi_1 \lor \phi_2)$, and
  - nothing else is a Boolean expression.
- When $x_i$ is a Boolean variable, expressions of the form $x_i$ and $\neg x_i$ are called literals.

Example

$((x_1 \lor x_2) \land \neg x_3)$ is a Boolean expression but $((x_1 \lor x_2)\neg x_3)$ is not.
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:
  - \(\lor_{i=1}^{n} \varphi_i\) stands for \(\varphi_1 \lor \cdots \lor \varphi_n\).
  - \(\land_{i=1}^{n} \varphi_i\) stands for \(\varphi_1 \land \cdots \land \varphi_n\).

- Other connectives can be defined as shorthands:
  - 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. Such a truth assignment induces then a truth value for any Boolean expression built from the Boolean variables.

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

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

- A truth assignment $T : X' \rightarrow \{\text{true, false}\}$ is appropriate to $\phi$ if $X(\phi) \subseteq X'$. 
The Satisfaction Relation

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

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

- If $\phi$ is a variable from $X'$, then $T \models \phi$ iff $T(\phi) = \text{true}$.
- If $\phi = \neg \phi_1$, then $T \models \phi$ iff $T \not\models \phi_1$.
- If $\phi = \phi_1 \land \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ and $T \models \phi_2$.
- If $\phi = \phi_1 \lor \phi_2$, then $T \models \phi$ iff $T \models \phi_1$ or $T \models \phi_2$.

Example

Let $T = \{x_1 \mapsto \text{true}, x_2 \mapsto \text{false} \}$.

Then $T \models x_1 \lor x_2$ but $T \not\models (x_1 \lor \neg x_2) \land (\neg x_1 \land x_2)$.
Logical equivalence

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

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

**Example**

Let $\phi$, $\phi_1$, and $\phi_2$ be expressions. We have, e.g., the following:

- $(\phi_1 \lor \phi_2) \iff (\phi_2 \lor \phi_1)$
- $((\phi_1 \land \phi_2) \land \phi_3) \iff (\phi_1 \land (\phi_2 \land \phi_3))$
- $\neg \neg \phi \iff \phi$
- $((\phi_1 \land \phi_2) \lor \phi_3) \iff ((\phi_1 \lor \phi_3) \land (\phi_2 \lor \phi_3))$
- $\neg(\phi_1 \land \phi_2) \iff (\neg \phi_1 \lor \neg \phi_2)$
- $\neg(\phi_1 \lor \phi_2) \iff (\neg \phi_1 \land \neg \phi_2)$
- $(\phi_1 \lor \phi_1) \iff \phi_1$
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:** \( (l_{1,1} \lor \cdots \lor l_{1,n_1}) \land \cdots \land (l_{m,1} \lor \cdots \lor l_{m,n_m}) \)
  
  **DNF:** \( (l_{1,1} \land \cdots \land l_{1,n_1}) \lor \cdots \lor (l_{m,1} \land \cdots \land l_{m,n_m}) \)

  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**

\[
(\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3 \lor x_4) \land (\neg x_4) \text{ is in CNF.}
\]

\[
(\neg x_1 \land x_2) \lor (\neg x_2 \land x_3 \land x_4) \lor (\neg x_4) \text{ is in DNF.}
\]

\[
(\neg x_1 \lor \neg x_1 \lor x_2) \iff (\neg x_1 \lor x_2).
\]
Every Boolean expression is equivalent to one in conjunctive (disjunctive) normal form.

Proof by construction: any Boolean expression can be transformed into CNF/DNF as follows.

- First, remove $\leftrightarrow$ and $\rightarrow$:
  \[
  \alpha \leftrightarrow \beta \quad \leadsto \quad (\neg \alpha \vee \beta) \land (\neg \beta \vee \alpha) \quad (1)
  \]
  \[
  \alpha \rightarrow \beta \quad \leadsto \quad \neg \alpha \lor \beta \quad (2)
  \]

- Then, push negations in front of Boolean variables:
  \[
  \neg \neg \alpha \quad \leadsto \quad \alpha \quad (3)
  \]
  \[
  \neg (\alpha \lor \beta) \quad \leadsto \quad \neg \alpha \land \neg \beta \quad (4)
  \]
  \[
  \neg (\alpha \land \beta) \quad \leadsto \quad \neg \alpha \lor \neg \beta \quad (5)
  \]

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 the CNF, move $\land$ connectives outside $\lor$ connectives:
  \[
  \alpha \lor (\beta \land \gamma) \quad \sim \quad (\alpha \lor \beta) \land (\alpha \lor \gamma) \quad (6)
  \]
  \[
  (\alpha \land \beta) \lor \gamma \quad \sim \quad (\alpha \lor \gamma) \land (\beta \lor \gamma) \quad (7)
  \]
- For the DNF, move $\lor$ connectives outside $\land$ connectives:
  \[
  \alpha \land (\beta \lor \gamma) \quad \sim \quad (\alpha \land \beta) \lor (\alpha \land \gamma) \quad (8)
  \]
  \[
  (\alpha \lor \beta) \land \gamma \quad \sim \quad (\alpha \land \gamma) \lor (\beta \land \gamma) \quad (9)
  \]

**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.

\[
\begin{align*}
(x_1 \lor x_2) &\rightarrow (x_2 \leftrightarrow x_3) \\
\iff &\neg(x_1 \lor x_2) \lor (x_2 \leftrightarrow x_3) \\
\iff &\neg(x_1 \lor x_2) \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \\
\iff &((\neg x_1 \land \neg x_2) \land ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \land (\neg x_2 \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \\
\iff &((\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_3 \lor x_2))) \land \\
&((\neg x_2 \lor ((\neg x_2 \land x_3) \land (\neg x_3 \lor x_2)))) \\
\iff &((\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_3 \lor x_2))) \land \\
&( (\neg x_2 \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)))) \\
\iff &((\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_3 \lor x_2))) \land \\
& ((\neg x_2 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_3 \lor x_2))) \\
\iff &((\neg x_1 \lor (\neg x_2 \lor x_3) \land (\neg x_1 \lor (\neg x_3 \lor x_2)) \land (\neg x_2 \lor x_3) \land (\neg x_2 \lor (\neg x_3 \lor x_2))) \\
\end{align*}
\]

(Simplification)
4. Satisfiability and Validity

- 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:
  $\phi$ is valid ($\models \phi$) iff $\neg \phi$ is unsatisfiable.
- Moreover, for any Boolean expressions $\psi_1$ and $\psi_2$,
  $\psi_1 \iff \psi_2$ iff $\models \psi_1 \iff \psi_2$ iff $\neg(\psi_1 \iff \psi_2)$ is unsatisfiable.

**Example**

- $(x_1 \lor \neg x_2) \land \neg x_1$ is satisfiable
- $(x_1 \lor \neg x_2) \land \neg x_1 \land x_2$ is unsatisfiable
- $(\neg x_1 \land x_2) \lor x_1 \lor \neg x_2$ is valid

Satisfiability forms a fundamental computational problem.
Definition (The Satisfiability Problem)

SAT:
INSTANCE: an expression $\phi$ in CNF
QUESTION: is $\phi$ satisfiable?

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

A nondeterministic Turing machine for $\phi \in$ SAT:
for all variables $x$ in $\phi$ do
    choose nondeterministically: $T(x) := \text{true}$ or $T(x) := \text{false}$;
if $T \models \phi$ then return “yes” else return “no”

- But whether SAT $\in$ P holds is an open question!
Horn clauses and HORNSAT

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

**Example**

The clauses \((\neg x_1 \lor x_2 \lor \neg x_3)\), \((\neg x_1 \lor \neg x_3)\), and \((x_2)\) are Horn clauses but \((\neg x_1 \lor x_2 \lor x_3)\) is not.

- A Horn clause \((\neg x_1 \lor x_2 \lor \neg x_3)\) 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).

**Definition**

HORNSAT:

INSTANCE: a conjunction \(\varphi\) of Horn clauses.

QUESTION: is \(\varphi\) satisfiable?
A Polynomial Time Algorithm for HORNSAT

Algorithm `hornsat(S)`
/* Determines whether \( S \in \text{HORNSAT} \) */
\( T := \emptyset \) /* \( T \) 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

\( \Rightarrow \) \( \text{HORNSAT} \in \mathbf{P} \).
5. Boolean Functions and Expressions

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

**Example**

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

- Any Boolean expression $\phi$ can be seen as an $n$-ary Boolean function with $n = |X(\phi)|$ by fixing the order of variables in $X(\phi)$.

- 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 $t = (t_1, \ldots, t_n)$,

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

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

**Example**

$\phi = (x_1 \lor x_2) \land \neg x_2$ expresses the Boolean function $f$ such that $f(\text{true, false}) = \text{true}$ and otherwise $f(t_1, t_2) = \text{false}$.
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 \text{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 \( t \in F \), 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: \( O(n2^n) \).

\( \downarrow \) Not all Boolean functions can be expressed concisely.

Example.

\[
\begin{array}{ccc}
\hline
x_1 & x_2 & f \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\hline
\end{array}
\]

\[
\phi_f = (\neg x_1 \land x_2) \lor (x_1 \land \neg x_2).
\]
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,
- $E$ must be acyclic ($i < j$ for all edges $(i, j) \in E$), and
- each gate $i$ has a sort $s(i) \in \{x_1, x_2, \ldots\} \cup \{true, false, \land, \lor, \neg\}$ such that
  - if $s(i) \in \{x_1, x_2, \ldots\} \cup \{true, false\}$, 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:

```
6 ∨ ¬ 5
3 ∨ 1 2
```

```
6

4 ∨

5 ¬

3 ∨

1 x1

2 x2
```
Semantics

A truth assignment for a circuit $C$ is a function

$$T : X(C) \rightarrow \{\text{true}, \text{false}\}$$

where $X(C)$ is the set of variables appearing in $C$.

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

- If $s(i) = \text{true}$, then $T(i) = \text{true}$
- If $s(i) = \text{false}$, then $T(i) = \text{false}$
- If $s(i) \in X(C)$, then $T(i) = T(s(i))$
- If $s(i) = \neg$ and $(j, i)$ is the unique edge entering $i$, then
  - $T(i) = \text{true}$ if $T(j) = \text{false}$, and
  - $T(i) = \text{false}$ otherwise
- If $s(i) = \land$ and $(j, i), (j', i)$ are the two edges entering $i$, then
  - $T(i) = \text{true}$ if $T(j) = \text{true}$ and $T(j') = \text{true}$, and
  - $T(i) = \text{false}$ otherwise
- If $s(i) = \lor$ and $(j, i), (j', i)$ are the two edges entering $i$, then
  - $T(i) = \text{true}$ if $T(j) = \text{true}$ or $T(j') = \text{true}$, and
  - $T(i) = \text{false}$ otherwise

The value of the circuit $C$ is $T(C) = T(n)$
Example

The circuit $C$: Consider a truth assignment

\[ T = \{ x_1 \mapsto \text{false}, x_2 \mapsto \text{false} \} \]

Now

\[ T(1) = T(x_1) = \text{false} \]
\[ T(2) = T(x_2) = \text{false} \]
\[ T(3) = \text{false} \text{ (as } T(1) = \text{false}, T(2) = \text{false}) \]
\[ T(4) = \text{false} \]
\[ T(5) = \text{true} \]
\[ T(6) = \text{true} \]

Hence, the value of the circuit $C$ is

\[ T(C) = T(6) = \text{true}. \]
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 $\phi$, i.e., the parse tree of the expression can be seen as a Boolean circuit.

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

The circuit \( C \):

\[ 6 \lor (4 \land 5) \lor 3 \lor (1 \lor 2) \]

and the corresponding Boolean expression \( \phi_C \):

\[ (x_1 \land (x_1 \lor x_2)) \lor \neg (x_1 \lor x_2) \]

Boolean circuit \( C_{\phi_C} \) for \( \phi_C \):

\[ \lor (\land \lor \neg) \]

Note the difference: in \( C \) substructure is shared
Computational problems related to Boolean circuits

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIRCUIT SAT:</strong></td>
</tr>
<tr>
<td>INSTANCE: a circuit $C$.</td>
</tr>
<tr>
<td>QUESTION: is there a truth assignment $T : X(C) \rightarrow {\text{true, false}}$ such that $T(C) = \text{true}$?</td>
</tr>
</tbody>
</table>

- **CIRCUIT SAT $\in \text{NP}$.**

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIRCUIT VALUE:</strong></td>
</tr>
<tr>
<td>INSTANCE: a circuit $C$ with no variables.</td>
</tr>
<tr>
<td>QUESTION: is it the case that $T(C) = \text{true}$?</td>
</tr>
</tbody>
</table>

- **CIRCUIT VALUE $\in \text{P}$.**
  
  (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 $t = (t_1, \ldots, t_n)$, $f(t) = T(C)$ 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 $\frac{2^n}{2n}$ 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

- 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.