Lecture 5: Elements of Boolean logic

Ion Petre
Computer Science, Åbo Akademi University

Fall 2015
http://users.abo.fi/ipetre/computability/
Focus in this lecture on computational problems from Boolean logic

- Logic is a central tool in Computer Science
- Logic problems will be essential in discussions on NP-completeness

Content

- Boolean expressions
  - Normal forms
  - SAT, HORNSAT
- Boolean functions
- Boolean circuits
Boolean expressions: syntax

Countably infinite alphabet of Boolean variables $X=\{x_1, x_2, \ldots\}$

- Each variable can take two truth values: **true** and **false**

Syntax. A **Boolean expression** can be any one of:

a) A Boolean variable or a negation of one; (literal)
b) An expression of the form $\neg \varphi_1$; (negation)
c) An expression of the form $(\varphi_1 \lor \varphi_2)$; (disjunction)
d) An expression of the form $(\varphi_1 \land \varphi_2)$; (conjunction)

where $\varphi_1, \varphi_2$ are Boolean expressions.

Two more connectives:

- $(\varphi' \Rightarrow \varphi'')$ stands for $(\neg \varphi' \lor \varphi'')$
- $(\varphi' \Leftrightarrow \varphi'')$ stands for $(\varphi' \Rightarrow \varphi'') \land (\varphi'' \Rightarrow \varphi')$
The set of variables $X(\varphi)$ of a Boolean expression $\varphi$ is defined as follows:

- if $\varphi$ is a literal, say $\varphi = x_i$ or $\varphi = \neg x_i$, then $X(\varphi) = \{x_i\}$
- if $\varphi = \neg \varphi'$, then $X(\varphi) = X(\varphi')$
- if $\varphi = \varphi' \lor \varphi''$, then $X(\varphi) = X(\varphi') \cup X(\varphi'')$
- if $\varphi = \varphi' \land \varphi''$, then $X(\varphi) = X(\varphi') \cup X(\varphi'')$
Boolean expressions: semantic

- **A truth assignment** $T$ is a mapping from a finite set $X'$ of Boolean variables, $X' \subseteq X$, to the set of truth values \{true, false\}
  - if $\varphi$ is a Boolean expression such that $X(\varphi) \subseteq X'$, then we say that $T$ is **appropriate** for $\varphi$

- Let $T$ be an appropriate truth assignment for $\varphi$. We say that $T$ satisfies $\varphi$, denoted $T \models \varphi$, if:
  - $\varphi=x_i$ and $T(x_i)=true$;
  - $\varphi=\neg \varphi'$ and $T(\varphi')=false$;
  - $\varphi=\varphi' \lor \varphi''$ and either $T \models \varphi'$ or $T \models \varphi''$ (or both)
  - $\varphi=\varphi' \land \varphi''$ and $T \models \varphi'$ and $T \models \varphi''$

- **Example.** Let $\varphi=((x_1 \lor \neg x_2) \land x_3)$ and $T(x_1)=T(x_2)=false$, $T(x_3)=true$. Then $T \models \varphi$.

- We say that $\varphi'$ and $\varphi''$ are **equivalent**, denoted $\varphi' \equiv \varphi''$ if for any truth assignment appropriate for both, $T \models \varphi'$ iff $T \models \varphi''$
  - this is the same as $T \models (\varphi' \leftrightarrow \varphi'')$
Boolean expressions: basic properties

Let $\varphi_1$, $\varphi_2$, $\varphi_3$ be arbitrary Boolean expressions. Then:

1. $(\varphi_1 \lor \varphi_2) \equiv (\varphi_2 \lor \varphi_1)$ (commutativity)
2. $(\varphi_1 \land \varphi_2) \equiv (\varphi_2 \land \varphi_1)$ (commutativity)
3. $\neg \neg \varphi_1 \equiv \varphi_1$ (double negation)
4. $((\varphi_1 \lor \varphi_2) \lor \varphi_3) \equiv (\varphi_1 \lor (\varphi_2 \lor \varphi_3))$ (associativity)
5. $((\varphi_1 \land \varphi_2) \land \varphi_3) \equiv (\varphi_1 \land (\varphi_2 \land \varphi_3))$ (associativity)
6. $((\varphi_1 \lor \varphi_2) \land \varphi_3) \equiv (\varphi_1 \land \varphi_3) \lor (\varphi_2 \land \varphi_3)$ (distributivity)
7. $((\varphi_1 \land \varphi_2) \lor \varphi_3) \equiv (\varphi_1 \lor \varphi_3) \land (\varphi_2 \lor \varphi_3)$ (distributivity)
8. $\neg (\varphi_1 \lor \varphi_2) \equiv (\neg \varphi_1 \land \neg \varphi_2)$ (De Morgan’s law)
9. $\neg (\varphi_1 \land \varphi_2) \equiv (\neg \varphi_1 \lor \neg \varphi_2)$ (De Morgan’s law)
10. $\varphi_1 \lor \varphi_1 \equiv \varphi_1$ (idempotency)
11. $\varphi_1 \land \varphi_1 \equiv \varphi_1$ (idempotency)

Proof: use the truth table method

Note: complete symmetry of conjunction and disjunction (unlike addition and multiplication in arithmetic)
Note: we can omit the parenthesis when they separate connectives of the same kind (conjunction or disjunction)

- $\bigwedge_{i=1}^{n} \varphi_i$ stands for $(\varphi_1 \land \varphi_2 \land \ldots \land \varphi_n)$
- $\bigvee_{i=1}^{n} \varphi_i$ stands for $(\varphi_1 \lor \varphi_2 \lor \ldots \lor \varphi_n)$

We say that $\varphi$ is in conjunctive normal form (CNF) if $\varphi = \bigwedge_{i=1}^{n} C_i$, where $n \geq 1$ and each $C_i$ is the disjunction of one or more literals

- $C_i$ are called the clauses of the expression $\varphi$

We say that $\varphi$ is in disjunctive normal form (DNF) if $\varphi = \bigvee_{i=1}^{n} D_i$, where $n \geq 1$ and each $D_i$ is the conjunction of one or more literals

- $D_i$ are called the implicants of the expression $\varphi$
Normal forms

Theorem. Every Boolean expression is equivalent to one in conjunctive normal form, and to one in disjunctive normal form.

Proof: by induction on the structure of $\varphi$

- if $\varphi$ is a literal, then the statement is trivially true
- if $\varphi = \neg \varphi'$: by induction hypothesis, $\varphi'$ can be written in DNF, with implicants $D_1, \ldots, D_n$. Then $\varphi$ can be written in CNF by the De Morgan laws
  - turn the disjunction into a conjunction of $\neg D_1, \ldots, \neg D_n$, where each $\neg D_i$ is a disjunction of literals (the negations of the literals in $D_i$)
  - similar argument for writing $\varphi$ into a DNF: start from a CNF of $\varphi$
- if $\varphi = (\varphi' \lor \varphi'')$: DNF is trivial by starting from DNF of both $\varphi'$ and $\varphi''$. For CNF, start from a CNF for both $\varphi'$ and $\varphi''$
  - $\varphi' = \bigwedge_{i=1}^m C_{1i}$, $\varphi'' = \bigwedge_{j=1}^n C_{2j}$
  - then $\varphi = \bigwedge_{i=1}^m \bigwedge_{j=1}^n (C_{1i} \lor C_{2j})$
- if $\varphi = (\varphi' \land \varphi'')$: similar argument as above

Note: the algorithmic construction of a DNF/CNF may be exponential. In other words: the DNF/CNF of $\varphi$ may be exponential in the size of $\varphi$
Satisfiability and validity

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

- A Boolean expression \( \varphi \) is **valid** (or **tautology**) if \( T \models \varphi \) for all truth assignments \( T \) that are appropriate to it. In this case we write \( \models \varphi \).

- Proposition. A Boolean expression is unsatisfiable if and only if its negation is valid.
Example

- \((x_1 \lor \neg x_2) \land \neg x_1\) is satisfiable
  - \(T(x_1) = T(x_2) = false\)

- \(((x_1 \lor x_2 \lor x_3) \land (x_1 \lor \neg x_2) \land (x_2 \lor \neg x_3) \land (x_3 \lor \neg x_1) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3))\) is not satisfiable
  - CNF: one must satisfy all clauses
  - first clause: at least one of \(T(x_1), T(x_2), T(x_3)\) must be true
  - last clause: at least one of \(T(x_1), T(x_2), T(x_3)\) must be false
  - Assume without loss of generality that \(T(x_1) = true\), \(T(x_2) = false\). From the third clause it follows \(T(x_3) = false\). This leave the fourth clause unsatisfied

- \(((\neg x_1 \land \neg x_2 \land \neg x_3) \lor (\neg x_1 \land x_2) \lor (\neg x_2 \land x_3) \lor (\neg x_3 \land x_1) \lor (x_1 \land x_2 \land x_3))\) is valid
  - it is the negation of the formula above
The satisfiability problem

**SAT**: Given a Boolean expression in CNF, is it satisfiable?

- Note: the problem can be trivially decided through an exponential-time algorithm that exhaustively searches through all possible truth assignments
- Note: the problem can be easily solved by a nondeterministic polynomial algorithm: guess the satisfying truth assignment and check it
- Conclusion: SAT is in NP
  - Unknown whether it is in P
  - Another way of formulating the problem “P=NP?”
An interesting special case of SAT: Horn clauses

A clause is called a **Horn clause** if it has at most one positive literal

- all literals, with the possible exception of just one, are negations of variables
- those having only negations of variables are called purely negative clauses, the others are called implications
- the implication clauses are equivalent with formulas of type \((x_1 \land \ldots \land x_m \Rightarrow y)\)

Examples

- \((\neg x_2 \lor x_3), (\neg x_1 \lor \neg x_2 \lor \neg x_3 \lor \neg x_4), (x_1)\) are Horn clauses. The first and the last are implications; equivalent with \((x_2 \Rightarrow x_3)\) and \((\text{true} \Rightarrow x_1)\), resp.
HORNSAT: Given a conjunction of Horn clauses, is it satisfiable?

Theorem. HORNSAT is in P

Proof

- Let \( \varphi \) be a conjunction of Horn clauses. Consider the truth assignment as the set \( T \) of those variables assigned to true

- Consider first only the implications in \( \varphi \) and build a truth assignment satisfying them
  - Initially, \( T := \emptyset \) (all variables false)
  - If there is an implication \((x_1 \land \ldots \land x_m) \Rightarrow y\) such that \( \{x_1, \ldots, x_m\} \subseteq T \), add \( y \) to \( T \)
  - Repeat the above step until \( T \) stops growing

- Let \( T' \) be another truth assignment that satisfies all implications in \( \varphi \).
  - Claim: \( T \subseteq T' \)
    - if not true, consider the first time in the algorithm above when we added to \( T \) a variable \( y \notin T' \). Then clearly, the current clause is not satisfied by \( T' \), a contradiction

- Claim: \( \varphi \) is satisfiable iff \( T \models \varphi \)
  - Let \( \varphi \) be satisfiable and prove \( T \models \varphi \). Enough to prove that \( T \) satisfies all purely negative clauses of \( \varphi \)
  - Assume there is a purely negative clause of \( \varphi \), say \( (\neg x_1 \lor \ldots \lor \neg x_m) \) not satisfied by \( T \)
  - In other words, \( \{x_1, \ldots, x_m\} \subseteq T \). It follows by the above claim that \( (\neg x_1 \lor \ldots \lor \neg x_m) \) is not satisfied by any other truth assignment that satisfies all implications in \( \varphi \), i.e. \( \varphi \) is unsatisfiable

- Note: constructing \( T \) and checking whether it satisfies \( \varphi \) can be done in polynomial time
BOOLEAN FUNCTIONS
An n-ary Boolean function is a function $f: \{\text{true, false}\}^n \rightarrow \{\text{true, false}\}$.

Any Boolean expression $\varphi$ can be thought of as an n-ary Boolean function $f_\varphi$, where $n = |X(\varphi)|$:

- For any truth assignment for $\varphi$, we can define the value of $f_\varphi$ as follows: $f_\varphi = \text{true}$ if $T|=\varphi$, $f_\varphi = \text{false}$, otherwise.

- In this case we say that Boolean expression $\varphi$ expresses Boolean function $f_\varphi$.

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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>

$\varphi_f = (\neg x_1 \land \neg x_2) \lor (x_1 \land \neg x_2)$
Boolean functions

- Proposition. Any n-ary Boolean function $f$ can be expressed as a Boolean expression $\varphi_f$ involving n variables.

Proof idea
- Check the truth table for $f$; each row is a conjunction; take the disjunction of the rows having true values.

Proof
- Let $F \subseteq \{\text{true, false}\}^n$ be the n-tuples that make $f$ true.
- For each $t \in F$, let $D_t$ be the conjunction of all variables $x_i$ with $t_i = \text{true}$ and of all negations $\neg x_i$ with $t_i = \text{false}$.
- Let $\varphi_f = \bigvee_{t \in F} D_t$.
- Easy to see that for any truth assignment $T$ appropriate for $\varphi$, we have $T \models \varphi$ iff $f(t) = \text{true}$, where $t_i = T(x_i)$.

Note: The expression produced above has length (number of needed symbols) $O(n2^n)$.
- many interesting functions have short expressions.
- however, most functions do not have (details on the last 2 slides).
BOOLEAN CIRCUITS
Boolean circuits: examples
A Boolean circuit is a graph $C=(V,E)$, where $V=\{1,\ldots,n\}$ is the set of nodes (called gates) such that:

- $C$ is acyclic – this implies that we can assume (modulo relabeling of nodes) that for any edge $(i,j)$ we have $i<j$
- the indegree of all nodes is 0, 1, or 2
- each gate $i$ has a sort associated to it $s(i) \in \{\text{true, false, } \land, \lor, \neg\} \cup \{x_1,x_2,\ldots\}$
  - if $s(i) \in \{\text{true, false}\} \cup \{x_1,x_2,\ldots\}$, then $\text{indegree}(i)=0$
  - these nodes are called inputs of $C$
  - if $s(i)=\neg$, then $\text{indegree}(i)=1$
  - if $s(i)\in\{\land,\lor\}$, then $\text{indegree}(i)=2$
- node $n$ (that has no outgoing edge) is called the output of the circuit
  - we may also consider circuits with several outputs: all nodes with outdegree 0 are outputs of the circuit – compute several functions at the same time
Boolean circuits: semantics

- \(X(C)\) is the set of all Boolean variables that appear in circuit \(C\). \(X(C)=\{x \in X \mid s(i)=x, \text{ for some gate } i \text{ of } C\}\)
- A truth assignment for \(C\) is \(T:X(C) \to \{\text{true, false}\}\)
  - it is appropriate for \(C\) if it is defined for all variables in \(X(C)\)
- Define a truth value \(T(i)\) for each gate \(i\):
  - 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 \(s(i)=\neg\), then let \((j,i)\) be the edge incoming into \(i\); \(T(i)=\text{true} \iff T(j)=\text{false}\)
  - if \(s(i)=\land\), then let \((j',i)\) and \((j'',i)\) be the two edges incoming into \(i\); \(T(i)=\text{true} \iff T(j')=\text{true} \text{ and } T(j'')=\text{true}\)
  - if \(s(i)=\lor\), then let \((j',i)\) and \((j'',i)\) be the two edges incoming into \(i\); \(T(i)=\text{true} \iff T(j')=\text{true} \text{ or } T(j'')=\text{true}\)
- The value of the circuit \(C\) is \(T(C)=T(n)\)
CIRCUIT SAT, CIRCUIT VALUE

- **CIRCUIT SAT**: Given a circuit C, is there a truth assignment T appropriate to C such that T(C)=true?
  - easy to argue that CIRCUIT SAT is computationally equivalent to SAT

- **CIRCUIT VALUE**: Same for circuits with no variable gates.
  - it has a polynomial-time algorithm: compute the value of the gates in numerical order of the gates
Boolean circuits compute Boolean functions

- We say that Boolean circuit $C$ with variables $x_1, \ldots, x_n$ computes $n$-ary Boolean function $f$ if, for any $n$-tuple of truth values $t=(t_1, \ldots, t_n)$, $f(t)=T(C)$, where $T$ is defined as $T(x_i)=t_i$, for all $1 \leq i \leq n$.

- Boolean functions $\rightarrow$ Boolean expressions $\rightarrow$ Boolean circuits

- **Question**: how big a circuit one needs for an $n$-ary Boolean function?

- **Theorem.** For any $n \geq 2$ there is an $n$-ary Boolean function $f$ such that no Boolean circuit with $2^n/(2n)$ or fewer gates can compute it.
  - **Proof idea**: calculate how many $n$-ary Boolean functions exist; (over)estimate how many circuits with $m$ or fewer gates there are.

- **Challenge**: even though many exponentially difficult Boolean functions must exist, there is not yet a “natural” family of Boolean functions that require more than linear number of gates to compute.

May 21, 2018
Learning objectives

- Syntax and semantic of Boolean expressions
- CNF, DNF
- The formulation of the satisfiability problem
- Equivalence between Boolean expressions, Boolean functions and Boolean circuits