Boolean Functions and Circuits

Gianpiero Cabodi  Stefano Quer
Politecnico di Torino
Torino, Italy
{gianpiero.cabodi,stefano.quer}@polito.it
http://staff.polito.it/{gianpiero.cabodi,stefano.quer}/

Boolean Expressions

- If \( B = \{0, 1\} \) a Boolean Function is
  \[ y = f(X) : B^n \rightarrow B \]
- With
  \[ X = (x_1, x_2, ..., x_n) \in B^n \]
  \[ x_1, x_2, ... \text{ are variables} \]
  \[ x_1, x_2' \text{ are literals} \]
- Basically
  \[ f \] maps each vertex of \( B^n \) to 0 or 1

Boolean Operations:
AND, OR, NOT

- Given two Boolean functions
  \[ f : B^n \rightarrow B \quad g : B^n \rightarrow B \]
  we define
    - The AND operation
      \[ h = f \land g \]
      \[ h = \{ x | f(x)=1 \land g(x)=1 \} \]
    - The OR operation
      \[ h = f \lor g \]
      \[ h = \{ x | f(x)=1 \lor g(x)=1 \} \]
    - The COMPLEMENT operation
      \[ h = \overline{f} \]
      \[ h = \{ x | \overline{f(x)} = 0 \} \]

Cofactor and Quantification

- Given a Boolean function \( f : B^n \rightarrow B \)
  with the input variable \( (x_1,x_2,...,x_i,...,x_n) \), we define
    - The positive cofactor
      \[ h = f_{x_i} \]
      \[ h = \{ x | f(x_1,x_2,...,1,...,x_n)=1 \} \]
    - The negative cofactor
      \[ h = f_{x_i} \]
      \[ h = \{ x | f(x_1,x_2,...,0,...,x_n)=1 \} \]
    - The existential quantification of variable \( x_i \)
      \[ h = \exists x_i . F \]
      \[ h = \{ x | f(x_1,x_2,...,0,...,x_n)=1 \lor f(x_1,x_2,...,1,...,x_n)=1 \} \]
    - The universal quantification of variable \( x_i \)
      \[ h = \forall x_i . F \]
      \[ h = \{ x | f(x_1,x_2,...,0,...,x_n)=1 \land f(x_1,x_2,...,1,...,x_n)=1 \} \]
**Characteristic function**

- Given a set \( A \)
- We define the Characteristic Function \( \chi_A(s) \) of the set \( A \) as

\[
\chi_A(s) = \begin{cases} 
1 & \text{IFF } s \in A \\
0 & \text{IFF } s \notin A
\end{cases}
\]

\( \chi_A(s) = 0 \)
\( \chi_A(s) = 1 \)

**Representation of Boolean Functions**

- What do we need?
  - A good data structure for Boolean formulas !!!
- We need representations for Boolean Functions for two reasons
  - A mechanism to build a data structure that represents the problem
  - A set of algorithm to manipulate the representation used
  - A decision procedure to decide about SAT or UNSAT, i.e., to perform Boolean reasoning

**Classical Methods**

- Canonical Forms
  - Canonical: one and only one representation for each function, i.e., data structure uniquely represents function
  - Decision procedure is trivial (e.g., just pointer comparison)
  - Example: Reduced Ordered Binary Decision Diagrams
  - Problem: Size of data structure is in general exponential
- NON Canonical Forms
  - Systematic search for satisfying assignment
  - Size of data structure is linear
  - Problem: decision may take an exponential amount of time

**Non-Classical Methods**

- Truth Table
- DNF
- CNF
- Automata

**Example**

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Classical Canonical Methods**

- Truth Table
  - \( F \) = Graphical/Tabular Representation
  - Two-Level Normal Forms
  - Canonical Disjunctive Normal Form (cDNF)
    - \( F = (x_1^* \land x_2^* \land ... \land x_n^*) \lor ... \lor (x_1^* \land x_2^* \land ... \land x_n^*) \)
  - Canonical Conjunctive Normal Form (cCNF)
    - \( F = (x_1^* \lor x_2^* \lor ... \lor x_n^*) \land ... \land (x_1^* \lor x_2^* \lor ... \lor x_n^*) \)
  - Automata
    - \( F \) = Graphical/Graph Representation
    - (Reduced Automatas are a Canonical Representation)

**Pros**

- Unique representation (one and only for each function)
- Constant Time Comparison (same representation)

**Cons**

- Exponential Size
- Complex Resolution Algorithms
- Satisfiability is NP-complete (Cook) (i.e., resolution algorithms require exponential time)

**Examples**

- \( F \) satisfiability requires polynomial time, tautology is co-NP
  - \( F \) is \( \leq_{m} \) tautology
  - Conversion CNF \( \iff \) DNF is exponential

**Example**

- \( F = (x_0 \lor x_1 \lor x_2 \land (x_3 \lor x_4 \land ... \land (x_{n-1} \lor x_n)) \)
Classical NON Canonical Methods

- **Disjunctive Normal Form (DNF)**
  \[ F = (x_1^* \land \ldots \land x_n^*) \lor \ldots \lor (x_1^* \land \ldots \land x_n^*) \]

- **Conjunctive Normal Form (CNF)**
  \[ F = (x_1^* \lor \ldots \lor x_n^*) \land \ldots \land (x_1^* \lor \ldots \lor x_n^*) \]

- **Automata**
  - \( F = \) Graphical/Graph Representation
  - (Not-Reduced Automatas)

**Pros**
- Non-Exponential Representation’s Size

**Cons**
- Non-Unique representation (more representations for each function)
- Complex Algorithms for Comparison
- Complex Algorithms for Conversions

Non Classical Methods

- **Decision Diagrams**
  - BDDs - Binary Decision Diagrams
  - ZBDDs - Zero Suppressed Binary Decision Diagrams
  - Etc.

- **Boolean Circuits**
  - RBCs – Reduced Boolean Circuits
  - BEDs – Boolean Expression Diagrams
  - AIGs – And Inverter Graphs

**Definition**
- A Boolean circuit is a directed graph \( C(G,N) \) where \( G \) are the gates and \( N \subseteq G \times G \) is the set of directed edges (nets) connecting the gates.
- Some of the vertices are designated
  - Inputs: \( \{ x \} \)
  - Outputs: \( \{ 0, 1 \} \)
- Each gate \( g \) is assigned a Boolean function \( f_g \) which computes the output of the gate in terms of its inputs
- The fanin \( \text{Fin}(g) \) of a gate \( g \) are all predecessor vertices of \( g \)
- The fanout \( \text{FOut}(g) \) of a gate \( g \) are all successor vertices of \( g \)
- The cone \( \text{Cone}(g) \) of a gate \( g \) is the transitive fanin of \( g \) and \( g \) itself.
- The support \( \text{Support}(g) \) of a gate \( g \) are all inputs in its cone
  \[ \text{Support}(g) = \text{Cone}(g) \land I \]

Boolean Circuits

- Used for two main purposes
  - As representation for Boolean reasoning engine
  - As target structure for logic implementation which gets restructured in a series of logic synthesis steps until result is acceptable

- Efficient representation for most Boolean problems we have in CAD
  - Memory complexity is same as the size of circuits we are actually building
  - Close to input representation and output representation in logic synthesis

Binary Decision Diagram (BDD)

- **Graph representation of a Boolean function** \( f \)
  - Vertices represent decision nodes for variables
  - Two children represent the two subfunctions
  - \( f(x = 0) \) and \( f(x = 1) \) (cofactors)
  - Restrictions on ordering and reduction rules
  - Can make a BDD representation canonical
Example

- \( F(I) = \{2, 4\} \)
- \( F(O) = \{7, 9\} \)
- \( \text{CONE}(6) = \{1, 2, 4, 6\} \)
- \( \text{SUPPORT}(6) = \{1, 2\} \)

And Inverter Graphs (AIGs)

- Base data structure uses two-input AND function for vertices and INVERTER attributes at the edges (individual bit)
  - Use De Morgan’s law to convert OR operation etc.
- Hash table to identify and reuse structurally isomorphic circuits