Formal Equivalence Checking - II

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Dept. of Electrical Engineering
Indian Institute of Technology Bombay, Mumbai
viren@ee.iitb.ac.in

EE 709: Testing & Verification of VLSI Circuits
Lecture – 7 (Jan 18, 2012)
Formal Equivalence Checking

- BDD is canonical form of representation
- Shannon’s expansion theorem

\[ f(x_1, x_2, \ldots, x_i, \ldots, x_n) = x_i \cdot f(x_1, x_2, \ldots, x_i=1, \ldots, x_n) + x'_i \cdot f(x_1, x_2, \ldots, x_i=0, \ldots, x_n) \]
Example OBDD

- Canonical representation of Boolean function
  - For given variable ordering
    - Two functions equivalent if and only if graphs isomorphic
      - Can be tested in linear time
    - Desirable property: *simplest form is canonical.*
Effect of Variable Ordering

Good Ordering

\[(a_1 \land b_1) \lor (a_2 \land b_2) \lor (a_3 \land b_3)\]

Bad Ordering

Linear Growth

Exponential Growth

Jan 18, 2012
EE-709@IITB
### Sample Function Classes

<table>
<thead>
<tr>
<th>Function Class</th>
<th>Best</th>
<th>Worst</th>
<th>Ordering Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU (Add/Sub)</td>
<td>linear</td>
<td>exponential</td>
<td>High</td>
</tr>
<tr>
<td>Symmetric</td>
<td>linear</td>
<td>quadratic</td>
<td>None</td>
</tr>
<tr>
<td>Multiplication</td>
<td>exponential</td>
<td>exponential</td>
<td>Low</td>
</tr>
</tbody>
</table>

#### General Experience
- Many tasks have reasonable OBDD representations
- Algorithms remain practical for up to 500,000 node OBDDs
- Heuristic ordering methods generally satisfactory
**ROBDD sizes & variable ordering**

- **Bad News** 🔥
  - Finding optimal variable ordering NP-Hard
  - Some functions have exponential BDD size for all orders *e.g.* multiplier

- **Good News** 😊
  - Many functions/tasks have reasonable size ROBDDs
  - Algorithms remain practical up to 500,000 node OBDDs
  - Heuristic ordering methods generally satisfactory

- **What works in Practice** 🔄
  - Application-specific heuristics *e.g.* DFS-based ordering for combinational circuits
  - Dynamic ordering based on variable sifting (*R. Rudell*)
Operations with BDD (1/5)

- **Restriction**: A restriction to a function to \( x = d \), denoted \( f|_{x=d} \), where \( x \in \text{var}(f) \), and \( d \in \{0,1\} \), is equal to \( f \) after assigning \( x = d \).

- Given BDD of \( f \), deriving BDD of \( f|_{x=d} \) is simple.
Let $v_1, v_2$ denote root nodes of $f_1, f_2$ respectively, with $\text{var}(v_1) = x_1$ and $\text{var}(v_2) = x_2$.

If $v_1$ and $v_2$ are leafs, $f_1 \text{ OP } f_2$ is a leaf node with value $\text{val}(v_1) \text{ OP } \text{val}(v_2)$. 
If $x_1 = x_2 = x$, apply Shannon’s expansion

$$f_1 \text{ OP } f_2 = x \cdot (f_1|_{x=0} \text{ OP } f_2|_{x=0}) + x' \cdot (f_1|_{x=1} \text{ OP } f_2|_{x=1})$$
Operations with BDD (4/5)

\[ f_1 \mid_{x=0} \quad \text{BDD for} \quad f_2 \mid_{x=0} \]

\[ f_1 \mid_{x=1} \quad \text{BDD for} \quad f_2 \mid_{x=1} \]

\[ f_2 \mid_{x=0} \quad \text{BDD for} \quad f_2 \mid_{x=1} \]

\[ f_1 \mid_{x=0} \quad \text{OP} \quad f_2 \mid_{x=0} \]

\[ f_1 \mid_{x=1} \quad \text{OP} \quad f_2 \mid_{x=1} \]
Else suppose $x_1 < x_2 = x$, in variable order

$$f_1 \text{ OP } f_2 = x_1 (f_1|_{x_1=0} \text{ OP } f_2) + x_1' (f_1|_{x_1=1} \text{ OP } f_2)$$
Operations with BDD: Example

\[ f_1 = X_1 \text{XOR} X_2 \]
\[ f_2 = X_2 \]

BDD for \( f_2 \mid_{x_1=0} \) OP \( f_2 \)

BDD for \( f_1 \mid_{x_1=1} \) OP \( f_2 \)
Operations with BDD: Example

\[ f_1 = X_1 \text{XOR} X_2 \]

\[ f_2 = X_2 \]

\[ f_2 \mid_{x_1=0} \text{OP} \ f_2 = X_1 \text{XOR} X_2 \]

\[ f_2 \mid_{x_1=1} \text{OP} \ f_2 = X_1 \text{XOR} X_2 \]
Operations with BDD: Example

\[ f = x_1 \cdot x_2 \]

\[ g = x_1' \cdot x_2' \]

\[ f + g \]
From circuits to BDD
Variants of decision diagrams

- **Multiterminal BDDs (MTBDD)** – Pseudo Boolean functions $\mathbb{B}^n \rightarrow \mathbb{N}$, terminal nodes are integers
- **Ordered Kronecker Functional Decision Diagrams (OKFDD)** – uses XOR in OBDDs
- **Binary Moment Diagrams (BMD)** – good for arithmetic operations and word-level representation
- **Zero-suppressed BDD (ZDD)** – good for representing sparse sets
- **Partitioned OBDDs (POBDD)** – highly compact representation which retains most of the features of ROBDDs
- **BDD packages** –
  - CUDD from Univ. of Colorado, Boulder,
  - CMU BDD package from Carnegie Mellon Univ.
  - In addition, companies like Intel, Fujitsu, Motorola etc. have their own internal BDD packages
Formal Equivalence Checking

- **Satisfiability Formulation**
  - Search for input assignment giving different outputs

- **Branch & Bound**
  - Assign input(s)
  - Propagate forced values
  - Backtrack when cannot succeed

- **Challenge**
  - Must prove all assignments fail
    - Co-NP complete problem
  - Typically explore significant fraction of inputs
  - Exponential time complexity
SAT Problem definition

Given a CNF formula, f :

- A set of variables, \( V \)
- Conjunction of clauses \( (a,b,c) \)
- Each clause: disjunction of literals over \( V \)

Does there exist an assignment of Boolean values to the variables, \( V \) which sets at least one literal in each clause to ‘1’?

Example :

\[
(a + b + c)(\overline{a} + c)(a + \overline{b} + c)
\]

\( a = b = c = 1 \)
DPLL algorithm for SAT

Given: CNF formula $f(v_1, v_2, \ldots, v_k)$, and an ordering function $\text{Next Variable}$

Example:

$$(a + b)(a + c)(a + b)$$

$C_1$, $C_2$, $C_3$

$\text{CONFLICT!}$

SAT!
DPLL algorithm: Unit clause rule

**Rule:** Assign to **true** any single literal clauses.

\[(a + b + c) \quad \text{||} \quad 0 \quad \text{||} \quad 0\]

Apply Iteratively: **Boolean Constraint Propagation (BCP)**

\[a(\overline{a} + c)(\overline{b} + c)(a + b + \overline{c})(\overline{c} + e)(\overline{d} + e)(c + d + \overline{e})\]

\[c(\overline{b} + c)(\overline{c} + e)(\overline{d} + e)(c + d + \overline{e})\]

\[e(\overline{d} + e)\]

\[c = 1\]
Anatomy of a modern SAT solver

**SAT Solver**

- **DPLL Algorithm**
- **Efficient BCP**
- **Clause database management**
  - Discard *useless* clauses (e.g. inactive or large clauses)
  - Efficient garbage collection
- **Search Restarts**
  - To correct for bad choices in variable ordering
  - Restart algorithm “periodically”
  - Retain some/all recorded clauses

**Conflict-driven learning**
Conflict driven search pruning *(GRASP)*

1. Non-chronological backtracking
2. Conflict-clause recording

Silva & Sakallah ‘95
Variable ordering

• Significantly impacts size of search tree
• Ordering schemes can be static or dynamic
• Conventional wisdom (pre-chaff):
  – Satisfy most number of clauses OR
  – Maximize BCP
  – *e.g.* DLIS, MOMs, BOHM etc.
Variable ordering: New ideas

- **New wisdom:** Recorded clauses key in guiding search

- **Conflict-driven variable ordering:**
  - Chaff (DAC’01): Pick var. appearing in *most* number of *recent* conflict clauses
  - BerkMin (DATE’02): Pick var. *involved* in most number of *recent* conflicts

- **Semi-static in nature, for efficiency**
  - Statistics updated on each conflict

- **Side-effect:** Better cache behavior
Efficient Boolean Constraint Propagation

- **Observation**: BCP almost 80% of compute time, under clause recording
- **Traditional implementation**: 
  - Each clause: Counter for #literals set to false
  - Assign. to variable ‘x’: Update all clauses having x, $\overline{x}$
- **New Idea**: Only need to monitor event when # free literals in a clause goes from 2 to 1
  - Need to *watch* only 2 literals per clause: SATO (Zhang’97), Chaff (DAC’01)
SAT solvers today

- **Capacity:**
  - Formulas up to a *million variables* and *3-4 million clauses* can be solved in *few hours*
  - Only for *structured instances e.g.* derived from real-world circuits & systems

- **Tool offerings:**
  - **Public domain**
    - GRASP: Univ. of Michigan
    - SATO: Univ. of Iowa
    - zChaff: Princeton University
    - BerkMin: Cadence Berkeley Labs.
  - **Commercial**
    - PROVER: Prover Technologies
Thank you