Implementation Verification: Equivalence Checking

Prof. Kurt Keutzer
EECS
UC Berkeley

With thanks to Srinivas Devadas, MIT

Design Process

Design: specify and enter the design intent

Verify: verify the correctness of design and implementation

Implement: refine the design through all phases
Design Verification

Is the design consistent with the original specification?
Is what I think I want what I really want?

Implementation Verification

Is the implementation consistent with the original design intent?
Is what I implemented what I wanted?
Manufacture Verification (Test)

Is the manufactured circuit consistent with the implemented design?
Did they build what I wanted?

Implementation verification for ASIC’s

Apply gate-level simulation ("the golden simulator") at each step to verify functionality:
- 0-1 behavior on regression test set

and timing:
- maximum delay of circuit across critical paths
Advantages of gate-level simulation
- verifies timing and functionality simultaneously
- approach well understood by designers

Disadvantages of gate-level simulation?
- computationally intensive - only 1 - 10 clock cycles of 100K gate design per 1 CPU second
- incomplete - results only as good as your vector set - easy to overlook incorrect timing/behavior
Alternative - Static Sign-off

Use static analysis techniques to verify:
- functionality: formal equivalence-checking techniques
- and timing: use static timing analysis – discussed earlier in the semester, but don’t forget about false paths!

Problem: RTL to RTL Verification

After verification RTL may still be modified
- RTL level improvements for:
  - performance
  - power
  - area
  - testability

Need to verify that new RTL is correct
**Problem: RTL to Gates Verification**

Verify the gate level implementation is consistent with the RTL level design

Errors may have occurred due to

- synthesis (heaven forbid!!)
- manual intervention

---

**Problem: Gates to Gates Verification**

Verify the modified gate level implementation is consistent with the RTL level design

Errors may have occurred due to

- Incorrect synthesis or module generation (heaven forbid!!)
- Test insertion
- Scan chain reordering
- Clock tree synthesis
- Post layout “tweaks”
Problem: Layout to Gates Verification (LVS)

Verify the modified gate level implementation is consistent with the RTL level design

Errors may have occurred due to
  – Errors in physical design tools
  – Manual changes in layout

Verification is primarily graphical or "topological"

Solving Layout to Gates Verification (LVS)

Extract gate level models from physical level

Graphically compare extracted model against gate-level schematic (layout versus schematic)

Flag any discrepancies
Solving Gates to Gates Verification

``specification''

``implementation''

Extract combinational portions

``spec''

``implementation''
Combinational Equivalence Checking

Given combination circuits C1 and C2 (Boolean functions B1 and B2) how can we practically prove that C1 is equivalent to C2?

Presumes equivalence-relation given (or discovered) between sequential circuits

Approaches
- Reasoning in the propositional calculus/Satisfiability
- Set-theoretic approaches (used in 2-level examples)
- Symbolic simulation (used in 2-level examples)
- Symbolic manipulation
  - graph isomorphism
  - structural reductions
- Canonical forms - BDD’s and variants
- Test-oriented methods
  - static, dynamic learning

These techniques form the foundation of modern equivalence checking/implemention verification
2-level circuits

\[(F \iff G) \iff (F \rightarrow G) \cdot (G \rightarrow F)\]

\[\iff (\overline{F} \lor G) \land (F \lor \overline{G})\]

Now, treating \(F\) and \(G\) as sets of cubes we can check if

\[\overline{(F \cup G)} \cap (F \cup \overline{G}) \iff 1\]

Which is feasible for most 2-level circuits/SOP expressions/DNF formulas

Worked well in the espresso era – doesn’t generalize to multilevel

Multilevel: Structural Methods

Combinational circuit 1

Combinational circuit 2

unmapped circuit 1

unmapped circuit 2

Compare them as graphs
Looks tough – why?
Turns out to be easy – why?
Structural Methods

Compare them as graphs
Looks tough – graph isomorphism
Turns out to be easy – DAGs
This helps but runs out of gas soon.

More powerful: Testing

Given two single-output circuits A and B
Are A and B equivalent can be posed as: Is there a test for F s-a-0?

If F s-a-0 is redundant, A ≡ B else test vector produces different outputs for A and B.
SAT Again

This time ask whether there is an input on which Circuit 1 and Circuit 2 differ? This time we don’t expect one!

Circuit 2 – Any structural similarities (found earlier) are shared

Circuit 1

More powerful: Comparison Mitre

Primary Inputs, Register and Black Box Outputs

spec

implementation

COMPARE

0 or 1
Canonical Forms: Binary Decision Tree

Do not have to store entire set of nodes, but have to enumerate them (slight improvement over two-level tautology).

Decision Graph

Share nodes in tree $\Rightarrow$ graph.

FULLY SHARED 2N nodes

partial sharing
**Definition of a Binary Decision Diagram**

A Binary Decision Diagram having root vertex $v$ denotes a Boolean function $f_v$.

1. If $v$ is a terminal vertex:
   (a) if $\text{value}(v) = 1$, then $f_v = 1$
   (b) if $\text{value}(v) = 0$, then $f_v = 0$

2. If $v$ is a nonterminal vertex with $\text{index}(v) = n$ then $f_v$ is the function:

   $$f_v(x_1, \ldots, x_n) = !f_{\text{low} v}(x_1, \ldots, x_{n-1}) + f_{\text{high} v}(x_1, \ldots, x_{n-1})$$

**Definition of an Ordered BDD**

A Binary Decision Diagram is ordered iff:

1. If $v$ is a non-terminal vertex:
   (a) if $\text{low}(v)$ is a non-terminal then, $\text{index}(v) < \text{index}(\text{low}(v))$ and
   (b) if $\text{high}(v)$ is a non-terminal then, $\text{index}(v) < \text{index}(\text{high}(v))$ and

This property implies the property of freedom in BDDs: In traversing any path from a vertex in a OBDD to its root then we encounter each decision variable at most once.
Ordered Binary Decision Diagram

Inputs satisfy ordering restriction. Each node/vertex $v$ in the graph has $\text{index}(v)$. Two children are $\text{low}(v)$ and $\text{high}(v)$. 0 and 1 are terminal vertices, others are non-terminal.

- $\text{index}(v) < \text{index}(\text{low}(v))$ for all $v$
- $\text{index}(v) < \text{index}(\text{high}(v))$

Ordered BDDs Enough?

Storage is always a problem for Ordered Binary Decision Diagram (OBDD) can we simplify them further?
Reduced, Ordered BDDs

An Ordered Binary Decision Diagram (OBDD) may still have "redundant" vertices.

Definition: An OBDD is reduced, if it contains no vertex \( v \) with \( \text{low}(v) = \text{high}(v) \), nor does it contain distinct vertices \( v \) and \( v' \) such that the subgraphs rooted by \( v \) and \( v' \) are isomorphic.

Can reduce an OBDD in \( O(|G| \log |G|) \) time.

Some Properties of a ROBDD

\[ f = x_1x_2x_3 + x_1x_2x_3 + x_1x_2x_3 \]

\[ f = x_1x_3 + x_1x_2x_3 \]

\[ f = ac + \overline{a}bc + acd + \overline{a}b\overline{c}d \]

Disjoint cover
Proof that ROBDDs are canonical - 1

Theorem (R. Bryant): If $G, G'$ are ROBDD's of a Boolean function $f$ with $k$ inputs then $G$ and $G'$ are identical.

Base Case: $i=0$, $f$ has 0 inputs.
$f$ can be the $0$ or $1$ ROBDD.
In either case $G$ and $G'$ are identical.

Induction Hypothesis: Suppose that for any Boolean function $f$ with $i < k$ inputs then if $H, H'$ are each ROBDD, with the same ordering, of the Boolean function $f$ then $H, H'$ are identical.

Let $G, G'$ be ROBDDs for $f$ under the same ordering.
Let $x_i$ be the input with lowest index (i.e. the root of the ROBDD) in the ROBDDs $G, G'$

Proof that ROBDDs are canonical - 2

By hypothesis, $f_0 \equiv f_0' \ f_1 \equiv f_1'$.
Let us consider a number of cases regarding sharing between $f_0, f_1$, and $f_0', f_1'$
If there is no sharing of vertices between $f_0, f_1$ and $f_0', f_1'$, then …

---

Kurt Keutzer 33

Kurt Keutzer 34
Proof that ROBDDs are canonical -2

By hypothesis, \( f_0 \equiv f_0' \) \( f_1 \equiv f_1' \).

Let us consider a number of cases regarding sharing between \( f_0, f_1 \), and \( f_0', f_1' \).

If there is no sharing of vertices between \( f_0, f_1 \) and \( f_0', f_1' \), then \( G \) is identical to \( G' \).

\[ f_0, f_0' \text{ identical} \]
\[ f_1, f_1' \text{ identical} \]
\[ Xi \text{ identical} \]

Proof that ROBDDs are canonical - 3

Suppose a vertex \( u \) is shared across \( f_0, f_1 \).

Then if there is a corresponding single \( u' \) shared in \( f_0', f_1' \) then \( G \), and \( G' \) are identical.
Proof that ROBDDs are canonical - 3

Suppose a vertex $u$ is shared across $f_0$, $f_1$.

Then if there is a corresponding single $u'$ shared in $f_0'$, $f_1'$ then $G$ and $G'$ are identical.

By the induction hypothesis the bdd rooted in $u$, $u'$ are the same.

Proof that ROBDDs are canonical – 4a

Alternatively, if $u$ in $G$ is realized as two (or more) vertices $u', u''$, in $G'$, then $G$, $G'$ are not identical:

What about this case?
Proof that ROBDDs are canonical – 4b

Alternatively, if \( u \) in \( G \) is realized as two (or more) vertices \( u', u'' \), in \( G' \), then \( G, G' \) are not identical:

But the ROBDDs rooted at \( u', u'' \) both realize the same Boolean function with the same ordering. So \( G' \) is not reduced because there are two such vertices in \( G' \). But this contradicts the assumption that \( G, G' \) are each ROBDDs.

Therefore, in each case \( G \) is identical to \( G' \). Therefore ROBDDs are a canonical representation.

ROBDDs are Canonical - use 1

Given an ordering, a logic function has a unique ROBDD.

Given two circuits, checking their equivalence reduces to a Directed Acyclic Graph isomorphism check between their respective ROBDDs

- can be done in linear time in \( |G_1| (= |G_2|) \).
- constructing ROBDD for a given function and ordering could take exponential time.
ROBDD - approach 2

Given two single-output circuits A and B

What is the ROBDD of this function? If 0 then circuits A and B are equivalent. Else they are not.

ROBDD Construction

Given ordering and multilevel network. ROBDD of a b

Proceed through network, constructing the ROBDD for each gate output, by applying the gate operator to the ROBDDs of the gate inputs.

Example worked through in extra slides for this lecture.
Sensitivity to Ordering

Given a function with \( n \) inputs, one input ordering may require exponential \# vertices in ROBDD, while other may be linear in size.

\[
f = x_1 x_2 + x_3 x_4 + x_5 x_6
\]

\[
x_1 x_2 x_3 x_4 x_5 x_6
\]

\[
x_1 x_4 x_2 x_3 x_6
\]

Summary of ROBDD checking procedure

Given circuits C1 and C2 to be verified for equivalence

A1) create the ``comparison mitre'' circuit D1
A2) find a variable ordering for the ROBDD for D1
A3) build the ROBDD and check for 0
or
B1) find a variable ordering for the ROBDD's of C1, C2
B2) build the ROBDD for each of C1, C2
B3) Check to see that the DAGs are isomorphic
Heuristic Input Ordering

BDD can be viewed as a multiplexor-based multilevel circuit.

Look at an (optimized) multilevel network and decide ordering for the BDD.

Generalize to multiple levels.
Resolv...
**Solving RTL-to-Gates Verification**

1. **Step 1:** (formally) translate HDL source into netlist
   - HDL "specification"
   - RTL Synthesis
   - netlist
   - Combinational logic
   - clk

2. **Step 2:** Perform gates-to-gates verification
   - gate-level implementation

**Solving RTL-to-RTL Verification**

1. **Step 1:** (formally) translate both HDL sources into netlists
   - HDL "specification"
   - RTL Synthesis
   - netlist
   - Combinational logic
   - clk

2. **Step 2:** Perform gate-to-gate verification on netlists
   - HDL implementation
   - RTL Synthesis
   - netlist
   - Combinational logic
   - clk
Current status of equivalence checking

Equivalence checking is one of the great successes of EDA in the late 90's

Equivalence checkers are now able to routinely verify complex (>10M gate) integrated circuit designs

Coupled with static timing analysis it has enabled “static-signoff”

Current technology leaders are Encounter Conformal from Cadence (Verplex) and Formality from Synopsys. Good proprietary (e.g. IBM/verity) solutions exist

Static sign-off methodology more widely used

Successful equivalence checkers must orchestrate a number of different approaches

- syntactic equivalence
- automatic test pattern generation-like approaches
- BDD-based techniques
- pattern-reduction methods

A few open problems remain

Open problems in implementation verification

More robust equivalence checking

Verification of equivalence between sequential circuits in which there is no obvious register-equivalence

- retimed circuits
- circuits with differing state assignments

Better diagnostics when circuits are not equivalent

Implementation verification between RTL and behavioral models
Retimed circuits

Circuits are equivalent (modulo some initial state issues) but it is not possible to show that they are equivalent using Boolean equivalence.

Encoding Problems

Some logic specifications are “symbolic” rather than binary-valued

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>+</td>
</tr>
<tr>
<td>SUB</td>
<td>-</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive-OR</td>
</tr>
<tr>
<td>INC</td>
<td>Increment</td>
</tr>
</tbody>
</table>

Can assign any binary code to the symbolic values, so long as they are different.
## Different State Encodings

<table>
<thead>
<tr>
<th>Circuit 1</th>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Different state encodings make circuits no longer amenable to combinational logic equivalence checking.

<table>
<thead>
<tr>
<th>Circuit 2</th>
<th>Symbol</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>INC</td>
<td>01</td>
<td></td>
</tr>
</tbody>
</table>

## Different Encodings

![Graphical representation of ALU operations]

- **ALU ``ADD''s on 00**
- **ALU ``ADD''s on 11**
Building ROBDD: Procedure Apply

Compute $f_1 \ <op> \ f_2$

<op> can be AND, OR, XOR, XNOR, etc.

To apply the operator to the ROBDDs represented by $f_1$ and $f_2$

1) If $v_1$ and $v_2$ are terminal vertices, simply generate a terminal vertex $u$ with
   \[ \text{value}(u) = \text{value}(v_1) <op> \ \text{value}(v_2) \]

2) Else if index($v_1$) = index($v_2$) = $i$
   Call algorithm apply recursively on low($v_1$) and low($v_2$) to generate a new vertex $u$, low($u$),
   high($v_1$) and high($v_2$) to generate high($u$), after creating vertex $u$, index($u$) = $i$
**Procedure Apply - 2**

3) If \( \text{index}(v_1) = i \), but \( \text{index}(v_2) > i \), then create a new vertex \( u \) having index \( i \), and apply algorithm recursively on \( \text{low}(v_1) \) and \( v_2 \) to generate \( \text{low}(u) \), and on \( \text{high}(v_1) \) and \( v_2 \) to generate \( \text{high}(u) \).

4) If \( \text{index}(v_2) = i \), but \( \text{index}(v_1) > i \), then create a new vertex \( u \) having index \( i \), and apply algorithm recursively on \( \text{low}(v_2) \) and \( v_1 \) to generate \( \text{low}(u) \), and on \( \text{high}(v_2) \) and \( v_1 \) to generate \( \text{high}(u) \).

\( O(G_1 \cdot G_2) \) complexity (though recursive).

“Multiplying” the two graphs.

---

**ROBDD Construction - 1**

Given ordering and multilevel network.

ROBDD of \( a \) \( b \)

- \( a \) \( b \) + \( c \) 
  - \( a \) \( b \) 
  - \( c \) 
  - \( d \) 

Begin with ROBDDS for primary inputs

Proceed through network, constructing the ROBDD for each gate output, by applying the gate operator to the ROBDDs of the gate inputs
Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

**ROBDD Construction – 2a**

Build ROBDD of \(a \times b\) using apply

\[a \times b + c\]

Proceed to AND gate

If \(\text{index}(v_1) = i\), but \(\text{index}(v_2) > i\), then create a new vertex \(u\) having index \(i\), and apply algorithm recursively on \(\text{low}(v_1)\) and \(v_2\) to generate \(\text{low}(u)\), and on \(\text{high}(v_1)\) and \(v_2\) to generate \(\text{high}(u)\).

**ROBDD Construction – 2c**

Build ROBDD of \(a \times b\) using apply

\[a \times b + c\]

Proceed to AND gate

If \(\text{index}(v_1) = i\), but \(\text{index}(v_2) > i\), then create a new vertex \(u\) having index \(i\), and apply algorithm recursively on \(\text{low}(v_1)\) and \(v_2\) to generate \(\text{low}(u)\), and on \(\text{high}(v_1)\) and \(v_2\) to generate \(\text{high}(u)\).
Given ordering \(<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>\) and multilevel network.

ROBDD Construction – 2d

Build ROBDD of \(a \times b\) using apply

If index(v₁) = i, but index(v₂) > i, then create a new vertex \(u\) having index i, and apply algorithm recursively on low(v₁) and v₂ to generate low(u), and on high(v₁) and v₂ to generate high(u).

ROBDD Construction – 3a

Build ROBDD of \(a \times b\) using apply

If index(v₂) = i, but index(v₁) > i, then create a new vertex \(u'\) having index i, and apply algorithm recursively on low(v₂) and v₁ to generate low(u'), and on high(v₂) and v₁ to generate high(u').
Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

**ROBDD Construction – 3b**

Build ROBDD of $a \times b$ using apply

If index($v_2$) = i, but index($v_1$) > i, then create a new vertex $u'$ having index i, and apply algorithm recursively on low($v_2$) and $v_1$ to generate low($u'$), and on high($v_2$) and $v_1$ to generate high($u'$).

**ROBDD Construction – 3c**

Build ROBDD of $a \times b$ using apply

If index($v_2$) = i, but index($v_1$) > i, then create a new vertex $u'$ having index i, and apply algorithm recursively on low($v_2$) and $v_1$ to generate low($u'$), and on high($v_2$) and $v_1$ to generate high($u'$).
**ROBDD Construction – 3d**

Given ordering $<<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>>$ and multilevel network.

**ROBDD Construction – 3e**

Given ordering $<<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>>$ and multilevel network.
**ROBDD Construction – 3f**

Given ordering \(<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>>\) and multilevel network.

Proceed to AND gate

Build ROBDD of \(a \cdot b\) using apply

\[
egin{align*}
\text{v1} & \quad \text{AND} \quad \text{v2} \\
0 & \quad = & & 0
\end{align*}
\]

If \(v_1\) and \(v_2\) are terminal vertices, simply generate a terminal vertex \(u\) with

\[
\text{value}(u) = \text{value}(v_1) \text{ <op> value}(v_2)
\]

\[
\text{high}(u') = \text{high}(u)
\]

**ROBDD Construction – 3g**

Given ordering \(<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>>\) and multilevel network.

Proceed to AND gate

Build ROBDD of \(a \cdot b\) using apply

\[
egin{align*}
\text{v1} & \quad \text{AND} \quad \text{v2} \\
0 & \quad = & & 0
\end{align*}
\]

If \(v_1\) and \(v_2\) are terminal vertices, simply generate a terminal vertex \(u\) with

\[
\text{value}(u) = \text{value}(v_1) \text{ <op> value}(v_2)
\]

\[
\text{high}(u') = \text{high}(u)
\]
ROBDD Construction – 4a

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Build ROBDD of a * b using apply

After returning from recursion:
If index(v₁) = i, but index(v₂) > i, then create a new vertex u having index i, and apply algorithm recursively on low(v₁) and v₂ to generate low(u), and on high(v₁) and v₂ to generate high(u).

ROBDD Construction – 4b

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Build ROBDD of a * b using apply

If index(v₁) = i, but index(v₂) > i, then create a new vertex u having index i, and apply algorithm recursively on low(v₁) and v₂ to generate low(u), and on high(v₁) and v₂ to generate high(u).
ROBDD Construction – 4c

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Proceed to AND gate

Build ROBDD of $a \cdot b$ using apply

After returning from recursion:
If index($v_1$) = $i$, but index($v_2$) $> i$, then create a new vertex $u$ having index $i$, and apply algorithm recursively on low($v_1$) and $v_2$ to generate low($u$), and on high($v_1$) and $v_2$ to generate high($u$).

ROBDD Construction – 4d

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Proceed to AND gate

Build ROBDD of $a \cdot b$ using apply

If index($v_2$) $= i$, but index($v_1$) $> i$, then create a new vertex $u'$ having index $i$, and apply algorithm recursively on low($v_2$) and $v_1$ to generate low($u'$), and on high($v_2$) and $v_1$ to generate high($u'$).
Build ROBDD of $a \cdot b$ using apply

If $\text{index}(v_2) = i$, but $\text{index}(v_1) > i$, then create a new vertex $u'$ having index $i$, and apply algorithm recursively on $\text{low}(v_2)$ and $v_1$ to generate $\text{low}(u')$, and on $\text{high}(v_2)$ and $v_1$ to generate $\text{high}(u')$. 

ROBDD Construction – 4f

Given ordering $<a,1>, <b,2>, <c,3>, <d,4>, <0,100>, <1,100>$ and multilevel network.
Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Build ROBDD of a * b using apply

If \(v_1\) and \(v_2\) are terminal vertices, simply generate a terminal vertex \(u\) with 
\[
\text{value}(u) = \text{value}(v_1) \text{ <op> } \text{value}(v_2)
\]

\(\text{low}(u')\)

\(\text{high}(u')\)
ROBDD Construction – 4i

Given ordering \(<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>\) and multilevel network.

Proced to AND gate

\[ a \otimes b + c \]

Build ROBDD of \(a \times b\) using apply

If index \((v_2) = i\), but index \((v_1) > i\), then create a new vertex \(u'\) having index \(i\), and apply algorithm recursively on \(low(v_2)\) and \(v_1\) to generate \(low(u')\), and on \(high(v_2)\) and \(v_1\) to generate \(high(u')\).

ROBDD Construction – 4j

Given ordering \(<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>\) and multilevel network.

Proced to AND gate

\[ a \otimes b + c \]

Build ROBDD of \(a \times b\) using apply

If \(v_1\) and \(v_2\) are terminal vertices, simply generate a terminal vertex \(u\) with

\[ value(u) = value(v_1) \text{ <op> } value(v_2) \]

where \(<op>\) is the operation \(\otimes\).
ROBDD Construction – 4k

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Build ROBDD of $a \times b$ using apply

If $v_1$ and $v_2$ are terminal vertices, simply generate a terminal vertex $u$ with

$$\text{value}(u) = \text{value}(v_1) \text{<op>} \text{value}(v_2)$$

ROBDD Construction – 4l

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

Build ROBDD of $a \times b$ using apply

After returning from recursion:
If $\text{index}(v_1) = i$, but $\text{index}(v_2) > i$, then create a new vertex $u$ having index $i$, and apply algorithm recursively on $\text{low}(v_1)$ and $v_2$ to generate $\text{low}(u)$, and on $\text{high}(v_1)$ and $v_2$ to generate $\text{high}(u)$. 
ROBDD Construction – 4m

Given ordering \(\langle a, 1 \rangle, \langle b, 2 \rangle, \langle c, 3 \rangle, \langle d, 4 \rangle, \langle 0, 100 \rangle, \langle 1, 100 \rangle\) and multilevel network.

Is this an ROBDD?

![Diagram of ROBDD Construction – 4m](image)

ROBDD Construction – 4n

Given ordering \(\langle a, 1 \rangle, \langle b, 2 \rangle, \langle c, 3 \rangle, \langle d, 4 \rangle, \langle 0, 100 \rangle, \langle 1, 100 \rangle\) and multilevel network.

Reduce

![Diagram of ROBDD Construction – 4n](image)
ROBDD Construction - 5

Given ordering <<a,1>,<b,2>,<c,3>,<d,4>,<0,100>,<1,100>> and multilevel network.

\[
\text{Reduce}
\]

\[
a \ b + c
\]

Example OR’ing of ROBDDs

\[
f_1 = x_1 x_3 = \overline{x}_1 + \overline{x}_3
\]

\[
f_2 = x_2 x_3
\]

New created graph

After reduction

\[
f = \overline{x}_1 + x_1 x_2 + x_1 \overline{x}_2 \overline{x}_3 = \overline{x}_1 + \overline{x}_3 + x_2
\]