Summary

- Introduction
- Binary Decision Diagrams
- ROBDD Example
- OBDD & ROBDD : Definition
- Operations on ROBDD
- Solving problems
- Model checking
- ROBDD & Petri nets
- Conclusion
Introduction

- Reduced Ordered Binary Decision Diagrams provide a compact representation of Boolean functions solutions space.
- ROBDD are based on BDD.
- Mostly used in problem solving and model checking.
Binary Decision Diagrams

- Is a DAG which represents both a Boolean expression and its solutions. Based on the Shannon Expansion theorem:
  \[ F(x_1, x_2, \ldots, x_n) = x_1.F(1, x_2, \ldots, x_n) + \overline{x_1}.F(0, x_2, \ldots, x_n) \]
0 is the dashed arrow while the solid arrow is 1.
ROBDD Example: Factorize nodes

\[ x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 + (x_1 + x_2)(x_3 + x_4) \]

\[
\begin{align*}
x_2(x_3 + x_4) & \\
x_1 & \\
x_3 + x_4 & \\
x_2 & \\
x_3 & \\
x_3 + x_4 & \\
x_4 & \\
0 & \\
1 & \\
\end{align*}
\]
ROBDD Example: Factorize nodes

$\frac{x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 + (x_1 + x_2)(x_3 + x_4)}{x_2(x_3 + x_4)}$

Steve Hostettler, Didier Buchs

Reduced Ordered Binary Decision Diagram
ROBDD Example: Factorize nodes

\[ x_1 \cdot x_2 \cdot \overline{x}_3 \cdot x_4 + (x_1 + x_2)(x_3 + x_4) \]

\[ x_2(x_3 + x_4) \]

\[ x_2 \cdot x_3 \cdot x_4 + x_3 + x_4 \]

\[ x_3 + x_4 \]

\[ \overline{x}_3 \cdot x_4 + x_3 + x_4 \]

\[ x_4 + x_4 \]

\[ x_4 \]

\[ x_3 \]

\[ x_2 \]

\[ x_1 \]

\[ 0 \]

\[ 1 \]
ROBDD Example: Remove useless nodes

Steve Hostettler, Didier Buchs
Reduced Ordered Binary Decision Diagram
ROBDD Example: Remove useless nodes

\[ x_1 \cdot x_2 \cdot \overline{x}_3 \cdot x_4 + (x_1 + x_2)(x_3 + x_4) \]

\[ x_2(x_3 + x_4) \]

\[ x_2 \]

\[ x_3 + x_4 \]

\[ x_4 \]

\[ 0 \]

\[ 1 \]

\[ x_2 \cdot \overline{x}_3 \cdot x_4 + \overline{x}_3 + x_4 \]
ROBDD Example: The result

\[ x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 + \ (x_1+x_2)(x_3+x_4) \]

\[ x_2(x_3+x_4) \]

\[ x_2(x_3+x_4) \leftarrow x_1 \]

\[ x_2 \leftarrow x_2 \]

\[ x_3 \leftarrow x_3 \]

\[ x_3+x_4 \]

\[ x_4 \]

\[ 0 \]

\[ 1 \]
Reduce BDD on the fly

- Isomorphic subgraphs are factorized.
- Remove node n if low(n) = high(n), i.e. left child = right child or 0 branch or 1 branch.
- Use an expression hashtable in order to known whether an expression has been computed already.
- Can be built by ROBDD composition.
- Complement function can be built in $O(1)$ by swapping the leaves.
OBDD : Definition

- Let $X$ be a set of boolean variables and $<$ a total order on $X$.
- Let $N$ be the set of nodes in the OBDD, where $L$ is the leaves and $NL$ are non-leaves. $N = L \cup NL$ and $L \cap NL = \emptyset$.
- Ordered Binary Decision diagrams
  $OBDD = \langle X, N, \text{var}, \text{val}, \text{low}, \text{high} \rangle$ are directed graph with:
  - Let $L$ be the set of leaves, $l \in L$ is labeled with a boolean value $\text{val}(l) \in \{0, 1\}$.
  - Let $NL$ be the set of non-leafes, $nl \in NL$ is labeled with a boolean variable $\text{var}(nl) \in X$.
  - $\forall n_i, n_{i+1} \in N$ :
    $n_{i+1} \in \{\text{low}(n_i), \text{high}(n_i)\} \implies (n_{i+1}$ is a leaf $\lor \text{var}(n) < \text{var}(n_{i+1})$)
- An OBDD is acyclic.
ROBDD : Definition

An OBDD is called reduced (ROBDD) iff :

- $\forall v, w \in L, \text{val}(v) = \text{val}(w) \implies v = w$.  
  $\implies$ Identical leaves are factorized.
- $\forall v \in NL, \text{low}(v) \neq \text{high}(v)$  
  $\implies$ Non-leaf may not have identical children.
- $\forall v, w \in NL :$  
  $(\text{var}(v) = \text{var}(w) \land \text{low}(v) \cong \text{low}(w) \land \text{high}(v) \cong \text{high}(w)) \implies v = w$  
  $\implies$ Nodes may not have isomorphic subgraphs.
Given a variable ordering, ROBDD are canonical (demonstration by induction on the size of the variable set): \( f, g : \mathbb{B}^n \rightarrow \mathbb{B}, f \cong g \iff \text{ROBDD}(f) \cong \text{ROBDD}(g) \).

Complexity, let \( K \) be the number of variables in the ROBDD:
- Best case complexity: \( O(K) \).
- Worst case complexity: \( O(2^K) \).
- Test whether a function is constantly true or false in \( O(1) \).

Ordering
- Is a NP complete problem.
- Dynamic optimization:
  - Reorganized the variable order on the fly.
  - Sifting algorithm.
- Heuristics.
Operations on ROBDD

- Shannon expansion theorem (Reloaded).
  - \( F(x) \odot G(x) = [F(x) \odot G(x)] + [F(\overline{x}) \odot (G(\overline{x}))]. \)
  - Requires the same variable ordering on both operand.
- Closure operations.
- Apply any boolean operator:
  - Disjunction.
  - Conjonction.
  - Exclusion.
  - Restriction.
- Variable Quantification
  - \( \forall x, F(x) \iff F(x) \land F(\overline{x}). \)
  - \( \exists x, F(x) \iff F(x) \lor F(\overline{x}). \)
- Those operators a very useful for model checking.
Solving problems

- Define the problem as a set of boolean equations.
- Build a ROBDD of the equation and then check if there is a solution.
- The N-Queens problem: How to put $n$ queens on a standard $n \times n$ chess board?
Let $C_{i,j}$ be the constraint for the position $i,j$ and $Q_{i,j}$ the presence of a queen at position $i,j$:

$$C_{1i,j} = Q_{i,j} \land \left( \bigwedge_{1 \leq k \leq n, k \neq i} \neg Q_{k,j} \cdot \neg Q_{k,j+i-k} \cdot \neg Q_{k,j+k-i} \right) \land \left( \bigwedge_{1 \leq l \leq n, l \neq j} \neg Q_{i,l} \right)$$

No other queens on diagonals and line and columns. $C_{i,j}$ must be satisfied for all $i$ and $j$.

At least one queen per column, to avoid no queens at all as a solution.
<table>
<thead>
<tr>
<th>N</th>
<th>Variables</th>
<th>Nodes</th>
<th>ROBDD size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16</td>
<td>$2^{16}$</td>
<td>30</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>$2^{25}$</td>
<td>195</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>$2^{36}$</td>
<td>133</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>$2^{49}$</td>
<td>1449</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>$2^{64}$</td>
<td>3887</td>
</tr>
</tbody>
</table>

This solution scale-up well compared to other algorithms. Look, for instance, at genetic algorithms [http://www.iba.k.u-tokyo.ac.jp/english/userlog.cgi?queenrun](http://www.iba.k.u-tokyo.ac.jp/english/userlog.cgi?queenrun).
Model checking

Problem: state space explosion!!!

Temporal logic: EX(warming)

Yes

No + counter example

Model Checker

s0

s1

s2

s4
Encoded state machines

Steve Hostettler, Didier Buchs

Reduced Ordered Binary Decision Diagram
Encoding states

Code states on 3 bits:

\[
\text{closed} \cdot \text{started} \cdot \text{warming} + \\
\text{closed} \cdot \text{started} \cdot \text{warming} + \\
\text{closed} \cdot \text{started} \cdot \text{warming} + \\
\text{closed} \cdot \text{started} \cdot \text{warming}
\]
...
Intersection

...
Encoding transitions

- Transitions are encoded with the intersection of the start-state, the final-state and the alphabet:
  \[ \tau = \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming} + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' + \text{closed}.\text{closed}'.\text{started}.\text{started}'.\text{warming}.\text{warming}' \]

- The best variable ordering for transitions is:
  \[ x_1 \text{Old} < x_1 \text{New} < x_2 \text{Old} < x_2 \text{New} < \cdots < x_i \text{Old} < y_i \text{New} < \alpha_1 < \cdots < \alpha_i. \]
The $\tau$ and the $\tau^{-1}$ functions

The $\tau$ (resp. $\tau^{-1}$) function is build using the transition (resp. reverse transition) relation.
Only need algorithms for EX, EU, EG since:

- $AX\phi \iff \neg EX(\neg \phi)$
- $AF\phi \iff \neg EG(\neg \phi)$
- $AG\phi \iff \neg EF(\neg \phi)$
- $EF\phi \iff E[true \cup \phi]$}
- $A[\phi \cup \theta] \iff \neg E[\neg \theta \cup (\neg \phi \land \neg \theta)] \land \neg EG(\neg \theta)$
Let $F$ be the set of states satisfying $\phi$:

- $S := \tau^{-1}(F)$
- Return $S$
Let $F$ (resp. $G$) be the set of states satisfying $\phi$ (resp. $\theta$):

- $S := G$
- $N := \emptyset$
- While($N \neq S$)
  - do
    - $N := S$
    - $S := S \cup (F \cap \tau^{-1}(S))$
  - done
- Return $S$
EG(\(\phi\))

Let F be the set of states satisfying \(\phi\):

- \(S := F\)
- \(N := \emptyset\)
- While( \(N \neq S\) )
  - do
    - \(N := S\);
    - \(S := S \cap \tau^{-1}(S)\)
  - done
- Return S
Let's compute the set of states that satisfy $EX(\neg warming)$:

The states satisfying $\neg warming$ are:

- $\text{closed}$
- $\text{started}$
- $\text{warming} + \text{closed} \cdot \text{started} \cdot \text{warming} + \text{closed} \cdot \text{started} \cdot \text{warming}$
\[ EX(\neg\text{warming}) = \tau^{-1} \cap \neg\text{warming} \]
$EX(\neg warming)$ The result by forgetting the target states
ROBDD & safe Petri nets

- One place, one variable (one hot encoding)
- Binary encoding
- Safeness can be checked on the fly.
If the place is k-bounded it can be encoded as $\log_2(k+1)-1$.

- Implements the transition as a binary subtraction.
- Check boundness by checking for overflow.
Parallelism

- BDD computation is highly parallelizable..
- Several Parallel BDD packages exists for diverse kind of architectures.
- Speedup 10x using 16 processors.
Conclusion

- Very powerful for particular classes of problem or model.
- Must use heuristics in order to determine a good variable ordering.
- Extensions have been proposed to deal with data types
- Intensively used for hardware verification
- Currently under investigation for software verification (DDD, SDD, ...)

Steve Hostettler, Didier Buchs

Reduced Ordered Binary Decision Diagram