Binary Decision Diagrams

• Most cited document in computer science according to citeseer:
  1. Graph-Based Algorithms for Boolean Function Manipulation – Bryant (1986)
    In this paper we present a new data structure for representing Boolean functions and an associated set of...

• Introduces Reduced Ordered Binary Decision Diagrams (RO)BDD

• What is a (RO)BDD?

A compact data structure to represent boolean functions

BDD: an example:
\[ f = (a \lor b) \land c \]

• An ordered (a<b<c) binary decision diagram for f:

![Binary Decision Diagram](image)
BDD: an example:
\[ f = (a \lor b) \land c \]

- Reduction: single occurrence of terminals

- Recursively from terminals: single occurrence of any node
  - Uses a unicity table for nodes (hash table)
  - Node hash key based on: node + hash key of sons
BDD: an example:

\[ f = (a \lor b) \land c \]

- Remove useless nodes
  - Criterion: variable value does not influence truth value of formula
  - \( \iff \) both son arcs point to same node

Properties of ROBDD

- Given a boolean expression \( g \), or a function \( f : \mathbb{B}^K \rightarrow \mathbb{B} \), there is a unique BDD encoding it (given a variable order \( x_K, \ldots, x_1 \))
- Many functions have a very compact encoding as a BDD
- The constant functions \( 0 \) and \( 1 \) are represented by the nodes \( \text{Zero} \) and \( \text{One} \), respectively
- Test whether a boolean expression is constantly true or false in \( O(1) \) time, given its BDD encoding
- Test whether two boolean expressions are equivalent in \( O(1) \) time, given their BDD encoding
Properties of BDD: Choice of an order

- The variable ordering affects the size of the BDD, consider
  \[ x_K \leftrightarrow y_K \land \cdots \land x_1 \leftrightarrow y_1 \]
  - with the order \((x_K, y_K, \ldots, x_1, y_1)\)
    \[ O(K) \text{ nodes} \]
  - with the order \((x_K, \ldots, x_1, y_K, \ldots, y_1)\)
    \[ O(2^K) \text{ nodes} \]
- The BDD encoding of some functions is large (exponential) for any order
  - the expression for bit 32 of the 64-bit result of the multiplication of two 32-bit integers
- Finding the optimal ordering is an NP-complete problem
Representing a state-space using DD

- **Principle:**
  - A path in the structure represents a reachable state
  - A state $S$ is described by the value of its state variables
- **Example:**

A mutual exclusion protocol for 2 processes $p$ and $p'$

---

Example
Example

Good representation for Globally Asynchronous Locally Synchronous (GALS) systems:
- Independence of local actions (t1 and t2) is well captured

Variable Ordering issues
Growth in node size w.r.t. order

• Linear growth vs exponential growth for poor ordering clearly visible
• Linear growth of representation but exponential growth of state-space size with appropriate ordering !!
• For some problems BDD based techniques allow to go much further than explicit representation techniques

<table>
<thead>
<tr>
<th>Nb Proc</th>
<th>Nb States</th>
<th>Consecutive order</th>
<th>Interlaced order</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>244</td>
<td>40</td>
<td>260</td>
</tr>
<tr>
<td>10</td>
<td>59050</td>
<td>80</td>
<td>13321</td>
</tr>
<tr>
<td>15</td>
<td>1.434e+07</td>
<td>120</td>
<td>589838</td>
</tr>
<tr>
<td>20</td>
<td>3.486e+09</td>
<td>160</td>
<td>-out of ram-</td>
</tr>
</tbody>
</table>

Efficiency of BDD

• Computations on BDD use a cache to limit complexity
• Cache is of the form
  • Key: <operation, operand_1,..,operand_n> -> value: result
  • Where operands and result are BDD nodes
• Example :
  • Cache : contains <union, a, b> -> c if (a U b) = c
union of BDD

- union \((a,b)\)
  - If \((a=0 \text{ or } b=1)\) return \(b\);
  - If \((a=1 \text{ or } b=0)\) return \(a\);
  - If \((a=b)\) return \(a\);
  - If \((<\text{union},a,b> \rightarrow r \text{ in cache })\) return \(r\);
  - BDD \(r=\) createBDD(
    - \(0 \Rightarrow \text{union}(a[0],b[0])\),
    - \(1\Rightarrow \text{union}(a[1],b[1])\)
  )
  - Cache.add( \(<\text{union},a,b> \rightarrow r\) )
  - Return \(r\);

Complexity of BDD operations

- createBDD uses a unicity table based on node structure
  - Hash on value of son nodes
- Operation cache =>
  - complexity proportional to \(\#\text{nodes}(a) \times \#\text{nodes}(b)\) =>
    - number of nodes
  - Without it complexity would be proportional to
    - number of PATHS in the structure
- Intersection differs from union only in terminal cases
  - If \((a = 1 \text{ or } b = 0)\) return \(b\)
  - If \((a = 0 \text{ or } b = 1)\) return \(a\)
State space computation

- Classic algorithm is based on Breadth-first exploration BFS
  - Consider a system composed of k state variables (i.e., state space represented as a k-level BDD)
  - Transition relation represented using 2k variables
    \((i,j) = (i_1, j_1, \ldots, i_k, j_k)\)
    System can go from \(i\) to \(j\) in one step
  - A special synchronized product (relational product) operation is defined to apply such a transition to a system

Symbolic state space generation: Classic fixed-point iteration

Given the initial state \(s\) and the next-state function \(N\), we obtain the state space \(S\):

\[ S = N^\infty(s) \]

The number of iterations equals one plus the maximum distance \(d\) of any state from \(s\)

The peak BDD size is usually achieved well before reaching the final BDD size at step \(d\)
Combining transitions

- BDD representing transitions can be combined using union
  - Full transition relation
    - \( \text{NextAll} = \text{union}(\text{Next}(t_1)+\ldots+\text{Next}(t_n)) \)

- Algorithm for BFS(s0)
  - \( S := S_0 \)
  - \( N := 0 \)
  - While (\( N \neq S \))
    - \( N := S \)
    - \( S := S \cup \text{NextAll}(S) \)
  - Return \( S \)

State space representation size

- BFS performs better than the “intuitive algorithm”
  - Size of representation is not directly linked to number of states manipulated
  - Re-evaluating a transition on an already reached state is likely to close a cache-hit thus has experimentally low cost.

- Algorithm for newBFS(s0)
  - \( S := S_0 \)
  - \( N := S_0 \)
  - While (\( N \neq 0 \))
    - \( N := \text{NextAll}(S) \setminus S \)
    - \( S := S \cup N \)
  - Return \( S \)

“Symbolic Model Checking: 10 20 States and Beyond” (LICS’1990) Burch, Clarke, McMillan, Dill, Hwang

Only computes NextAll on newly reached states
Decomposing the transition relation
[Roig’95]

- The idea is to cluster some transitions but keep an expression of the transition relation in parts
  - Next(i) for I in 1..nbClusters = union (Next(tj),..Next(tj+k))
  - Create one cluster for each process (requires structural information)
- Not quite BFS anymore as chainings may occur
- Solves problems experienced with the size of NextAll BDD

**Algorithm for chainBFS(s0)**
- S := s0
- N := 0
- While (N != S)
  - N := S
  - For (i in 1..nbClusters)
    - S := S U Next(i) (S)
- Return S

Comparing the four approaches

| N   | |S| | New | Time (sec) | |New | Memory (MB) |
|-----|---|----|-----|-----------|---|--------------|
|     |   |    | BFS | New      | Chain | BFS | New      | Chain |

**Dining Philosophers: K = N, |S_k| = 3.4 for all k**

| N   | |S| | BFS | BFS | Chain | BFS | BFS | Chain |
|-----|---|----|-----|-----|-------|-----|-----|-------|
| 50  | 2.2 x 10^51 | 37.6 | 36.8 | 1.3 | 1.3 | 146.8 | 131.6 | 2.2 | 2.2 | 0.0 |
| 100 | 6.0 x 10^52 | 644.1 | 630.4 | 5.4 | 5.3 | >999.9 | >999.9 | 8.9 | 8.9 | 0.0 |
| 1000| 9.2 x 10^52 | 955.4 | 915.5 | —   | —   | —     | 895.2 | 895.0 | 0.3 |

**Slotted Ring Network: K = N, |S_k| = 15 for all k**

| N   | |S| | BFS | BFS | Chain | BFS | BFS | Chain |
|-----|---|----|-----|-----|-------|-----|-----|-------|
| 5   | 5.3 x 10^4  | 0.2  | 0.1  | 0.1  | 0.1  | 0.8  | 1.1  | 0.3  | 0.2 | 0.0 |
| 10  | 8.3 x 10^9   | 21.5 | 24.1 | 2.1  | 2.1  | 39.0 | 45.0 | 5.7  | 3.3 | 0.0 |
| 15  | 1.5 x 10^15  | 745.4 | 771.5 | 18.5 | 18.5 | 344.3 | 375.4 | 35.1 | 20.2 | 0.0 |

**Round Robin Mutual Exclusion: K = N+1, |S_k| = 10 for all k except |S_k| = N+1**

| N   | |S| | BFS | BFS | Chain | BFS | BFS | Chain |
|-----|---|----|-----|-----|-------|-----|-----|-------|
| 10  | 2.3 x 10^4  | 0.2  | 0.1  | 0.1  | 0.1  | 0.6  | 1.2  | 0.1  | 0.1 | 0.0 |
| 20  | 4.7 x 10^7   | 2.7  | 4.4  | 0.3  | 0.3  | 5.9  | 12.8 | 0.5  | 0.5 | 0.0 |
| 50  | 1.3 x 10^17  | 263.2 | 427.6 | 2.9  | 2.9  | 126.7 | 257.7 | 4.3  | 3.8 | 0.1 |

**FMS: K = 10, |S_k| = N+1 for all k except |S_k| = 4, |S_k| = 3, |S_k| = 2**

| N   | |S| | BFS | BFS | Chain | BFS | BFS | Chain |
|-----|---|----|-----|-----|-------|-----|-----|-------|
| 5   | 2.9 x 10^6  | 0.7  | 0.1  | 0.1  | 0.1  | 2.6  | 2.2  | 0.4  | 0.2 | 0.0 |
| 10  | 2.5 x 10^6   | 7.0  | 5.8  | 0.5  | 0.3  | 18.2 | 14.7 | 2.3  | 1.3 | 0.0 |
| 25  | 8.5 x 10^13  | 677.2 | 437.9 | 12.9 | 5.1  | 319.7 | 245.3 | 42.7 | 21.2 | 0.1 |

Performances using Smart
Using the state space representation

- The state space representation allows to easily verify safety properties
  - Can we reach a "bad" state
  - Can A and B both be true simultaneously
- State space generation is the basis for more complex temporal logic properties such as CTL
  - CTL properties can be expressed as nested fix points of the transition relation and its reverse Next-1
CTL semantics

- **EX**
  - Let $F$ be the set of states satisfying “$f$”
  - $F$ can be built by selecting states from the full state space
- **EX(F)**
  - $S := \text{Next}^{-1}(F)$
  - Return $S$
Operator EU for E(f U g)

- Let \( F \) and \( G \) be the set of states satisfying “f” and “g”
- EU\((F,G)\)
  - \( S := G \)
  - \( N := 0 \)
  - While (\( N != S \))
    - \( N := S \)
    - \( S := S \cup ( F \cap \text{Next}^{-1}(S)) \)
  - Return \( S \)

Operator EG for EG (f)

- Let \( F \) be the set of states satisfying “f”
- EG\((F)\)
  - \( S := F \)
  - \( N := 0 \)
  - While (\( N != S \))
    - \( N := S \)
    - \( S := S \cap \text{Next}^{-1}(S) \)
  - Return \( S \)
Some BDD extensions

Decision Diagrams: Widely accepted in verification tools

- SMV (US):
  - FSM/Kripke structure
  - emblematic first symbolic enabled verification tool. Now uses (NuSMV 2-Italy) library Cudd.
- Uppaal (Den-Nor):
  - Hybrid systems
  - uses Difference Bounded Matrix diagrams to represent clocks
- Prism (UK):
  - Stochastic process algebra
  - uses Matrix DD and Multi-terminal DD for stochastic verification
- Smart (US):
  - Stochastic Petri nets
  - uses integer valued DD, both CTL and stochastic solution engine (+saturation)
- Red (Taiwan):
  - Timed automata
  - Specific solution for real time systems
**Integer valued Decision Diagrams**

- **Some variants:**
  - Multi-way DD (Ciardo&Miner Icatpn'99)
  - Data Decision Diagrams (Couvreur et al. Icatpn'02)

- **Variables may have an integer domain instead of boolean domain**
  - Usually zero-suppressed, to allow arbitrary variable domains provided the actual reachable set is finite
  - Using BDD, one has to decode integer state variables into their log_2(n) bit representation
  - Problem: complexity also linked to nb arcs/node, not only number of nodes

- **Data Decision Diagram**
  - No variable order (handled in the union operation to handle incompatibilities)
  - Homomorphisms to define the transition relation

**Multi terminal Decision Diagrams**

- **Multi-terminal (MTDD [Fujita+ '97] or Algebraic DD [Bahar+ '93]):**
- Instead of single terminal 1, use several terminals
- Allows to give a correspondence between a state and a characteristic it has
  - Not just presence or absence of a state

- **Example 1:** integer terminal
  - Terminal gives the distance (number of steps) from initial state of any state
  - union handles same path with different terminals => keep the smallest terminal
  - Useful for finding shortest witness or counter-example traces

- **Example 2:** Real valued terminal
  - Used in stochastic/probabilistic systems, gives the probability of being in a state
  - union handles the approximation (2 terminals x and y considered equal if |x-y| < epsilon)
Saturation vs BFS

- Saturation algorithm introduced by Ciardo+ Tacas’01 Tacas’03
  - Fire transitions from the leaves (terminals) up to root
  - Go to ancestor of a node iff. The current node is saturated: all events that only affect this variable and variables below it have been fired until a fixpoint is reached
  - Each time a node is affected by an event, resaturate it.
- Not BFS anymore, firing order of events follows data structure
  - Huge reduction of time and space complexity
  - Good tackling of intermediate peak size effect

Saturation example

- Suppose event = if (d < 2) d++;

\[ \text{Iteration 1} \]
\[ \text{Iteration 2} \]

BFS style iterations
Saturation effect

- Nested transitive closure or fixpoint = saturation allows:
  - single traversal of the top of the tree => cost of + and h
  - less intermediate nodes

BFS style iterations

Iteration 1

Iteration 2

Useless intermediate nodes!!

Performance measures: effect of saturation

- Transitive closure allows more efficiency
  - Saturation “à la Ciardo” (Tacas’01 and ‘03)
  - Organize events by highest variable affected

<table>
<thead>
<tr>
<th>Model</th>
<th>N</th>
<th>Nb. States</th>
<th>final nodes</th>
<th>total nodes</th>
<th>time (s)</th>
<th>total nodes</th>
<th>time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dining Philosophers</td>
<td>50</td>
<td>2.23e+31</td>
<td>1387</td>
<td>13123</td>
<td>11.6</td>
<td>10739</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>4.97e+62</td>
<td>2767</td>
<td>26823</td>
<td>54.19</td>
<td>21689</td>
<td>0.18</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>2.47e+125</td>
<td>5587</td>
<td>54223</td>
<td>234</td>
<td>43589</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>9.10e+626</td>
<td>27907</td>
<td>-</td>
<td>-</td>
<td>218709</td>
<td>2.1</td>
</tr>
<tr>
<td>Slotted Ring Protocol</td>
<td>10</td>
<td>8.29e+09</td>
<td>1281</td>
<td>35898</td>
<td>83.07</td>
<td>45970</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1.46e+15</td>
<td>2790</td>
<td>118054</td>
<td>595</td>
<td>132126</td>
<td>2.26</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1.72e+52</td>
<td>29401</td>
<td>-</td>
<td>-</td>
<td>3.56e+06</td>
<td>61.58</td>
</tr>
<tr>
<td>Flexible Manufacturing System</td>
<td>25</td>
<td>8.50e+13</td>
<td>2545</td>
<td>50480</td>
<td>20.75</td>
<td>88062</td>
<td>1.88</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>4.24e+17</td>
<td>8820</td>
<td>231464</td>
<td>240.4</td>
<td>490062</td>
<td>9.78</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>1.58e+20</td>
<td>21300</td>
<td>-</td>
<td>-</td>
<td>1.72e+06</td>
<td>37.06</td>
</tr>
<tr>
<td>Kanban</td>
<td>10</td>
<td>1.01e+09</td>
<td>257</td>
<td>26862</td>
<td>20.47</td>
<td>5637</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>1.04e+16</td>
<td>3217</td>
<td>-</td>
<td>-</td>
<td>209117</td>
<td>3.96</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>1.73e+19</td>
<td>13417</td>
<td>-</td>
<td>-</td>
<td>1.22e+06</td>
<td>28.08</td>
</tr>
<tr>
<td></td>
<td>200</td>
<td>3.17e+22</td>
<td>42817</td>
<td>-</td>
<td>-</td>
<td>9.23e+06</td>
<td>238.99</td>
</tr>
</tbody>
</table>
Set Decision Diagram (SDD)

- Limits of pre-existing library (DDD) reached
  - Need for structure
- Idea: hierarchy
  - Label the arcs with a SET = Set Decision Diagram

Set Decision Diagrams: A compositional Model

- SDD arcs may be labeled by DDD
  - Hierarchical Structure
  - Adapted to composition
    - captures the similarity of repeated modules

The similarity of behavior of the philosophers is captured:
8*(1 Philosopher)
Mod_1
Mod_1
Mod_1
Mod_0
Mod_2
Mod_0
Mod_0
Mod_1
mod 1
mod 1
mod 2
mod 3
mod 4
Peak at 1000 nodes
(P1) // (P2) // (P3) // (P4) // Pi

DDD sharing & SDD sharing

State space,
4 philosophers (DDD)

P1
P2
P3
P4
(P1) // (P2) // (P3) // (P4)

With SDD, the state of a
philosopher is referenced.

P1 // P2 // P3 // P4

DDD to SDD
Model "slotted ring", 5 participants, 53856 states

(saturation disabled)
pure DDD
350 nodes
Peak 93k nodes
26 seconds

SDD ...
18 nodes
Peak at 1000 nodes
0.14 seconds
...+DDD
32 nodes
Kanban example, low parameter value (5)

Explosion of arcs per node as states per component increases

MDD (Smart)

Conclusions on Decision Diagrams

• Huge success in hardware verification
• Passage to verification of software more difficult
  • Variable length signature (creation/destruction of objects)
  • Integer or real valued variables
  • Expression of complex (code) transitions
• An active research topic
  • New Extensions of DD (i.e. hierarchy...)
  • Using DD to tackle new problems (i.e. real-valued clocks...)
  • New transition relation strategies (saturation...)
• DD are perfect for some systems, poor in other settings
  • More research is needed to provide heuristics that auto-
    configure DD libraries
Symmetry exploitation: another « symbolic » representation

Factual observation: Existence of symmetries

- In Distributed Systems:
  - Repeated Instances: clients, process ...
  - Symmetric Domains: memory addresses, pid, ...
  - Discretisation of variables with a continuous domain: sensors...
    - Large domains but existence of few values critical to control:
      - $i < \text{threshold}, i \neq 0$...

- Exploiting symmetries
  - Well Formed Petri Nets [CDFH'90]
  - Murphi [IpDill'96]
  - Extensions to partial symmetries [Capra'00,BHI'04]
Symbolic Reachability Graph: principle

- Subclass AltOK is [0..29]
- Subclass AltNOK is [30..1000,AltErr]

SRG properties

- **Symbolic reachability graph SRG**
  - Based on a notion of permutations that preserve behaviors
  - Permutations described through a partition
  - Symbolic state = equivalence class representative of behavior

- **Equivalence Classes**
  - w.r.t. Transition relation
  - Permutations are allowed if they respect the partition

- **Gain may be exponential**
  - Very few symbolic states
  - Symbolic firing => few arcs
  - The rougher the partition, the better the gain
  - Worst case SRG = RG, if partition distinguishes every element = no permutations allowed
Well-Formed Nets:
A simple Client Server protocol

Concrete Marking

Symbolic Marking

Concrete State space
(C=2, S=1, M=2)

- Dimensions
  - 24 Nodes
  - 54 Arcs

- Cycles =
  - Client to server call with message M1
  - No interlaced requests (i.e. other client remains idle)
SRG (1)
Clients are Permutable

24 nodes, 54 arcs

C1 ≠ C2

A client sends M1 to server
Two paths (C1 ≠ C2)

Graphe Quotient (2)
Les Messages Sont Permutables

24 nodes, 54 arcs

C1 ≠ C2

C1 ≈ C2

10 nodes, 16 arcs

C1 ≈ C2 et M1 ≈ M2

“a” client sends a message

“a” client sends M2

Same configuration, only single path
(identity of clients may be permuted)

Same configuration, a single path
(identity of messages permutable)
Conclusions on SRG

- Symbolic firing rule not presented due to lack of space
  - Sorry!
- SRG is compatible with many other techniques
  - Partial order reductions
  - Attempts at combining with DD...
- SRG suffers from some drawbacks
  - Requires full symmetry of the system
    - New techniques allow to construct variants of SRG with support for partial symmetry in property or system
  - Problem of defining symmetries left to user
    - But automatic symmetry analysis/detection is possible
  - Complexity of symbolic firing may be high
    - Usual trade-off time vs. memory
    - Makes it well adapted to grid computing (high CPU usage, low bandwidth)
- Well Formed nets currently being standardized by ISO under the appellation Symmetric Nets