Tagged BDDs: Combining Reduction Rules from Different Decision Diagram Types
Tom van Dijk & Robert Wille & Robert Meolic
FMCAD 2017
TL; DR

- **Normal BDDs:**
  Great with symmetries and dont-cares

- **Zero-suppressed BDDs:**
  Great with subsets and sparse matrices

- **Tagged BDDs:**
  Combining both types to use both features simultaneously
  TBDDs implemented in the BDD package “Sylvan”

- **Application:**
  On-the-fly symbolic learning of transition systems (LTSmin)
BDDs and ZBDDs

Binary Decision Diagrams

- A BDD is a directed acyclic graph encoding a $\mathbb{B}^k \rightarrow \mathbb{B}$ function or a $S \subseteq \mathbb{B}^k$ set
- Paths represent valuations of $\mathbb{B}^k$
- For sets: path to 1, then valuation is in the set

$$x \land y$$

$$x \lor y$$

$$x \oplus y$$
Zero-suppressed BDDs

- Different “reduction rule”
  - BDD: skipped variables are don't-cares
  - ZBDD: skipped variables are set to false
- Different applications: sparse matrices, subsets

<table>
<thead>
<tr>
<th>BDD reduction rule</th>
<th>ZBDD reduction rule</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="BDD Diagram" /></td>
<td><img src="image2" alt="ZBDD Diagram" /></td>
</tr>
</tbody>
</table>
BDDs and ZBDDs

Two decision diagrams representing \( f(x_1, x_2, x_3) = \overline{x_1}x_2 \), i.e., the set \( \{ \overline{x_1}x_2x_3, \overline{x_1}x_2\overline{x_3} \} \)
BDDs and ZBDDs

Example

- A set on variables $x_1$ to $x_8$
- Four items: \{00000000, 00000010, 00000100, 00000110\}. 
Example

- A set on variables $x_1$ to $x_8$
- Four items: \{00000000, 00000010, 00000100, 00000110\}.
- As a normal BDD

\[\begin{array}{c}
\text{x}_1 \rightarrow \text{x}_2 \rightarrow \text{x}_3 \rightarrow \text{x}_4 \rightarrow \text{x}_5 \rightarrow \text{x}_8 \rightarrow 1 \\
\end{array}\]
BDDs and ZBDDs

Example

- A set on variables $x_1$ to $x_8$
- Four items: $\{00000000, 00000010, 00000100, 00000110\}$.
- As a normal BDD

![Normal BDD Diagram]

- As a Zero-suppressed BDD

![Zero-suppressed BDD Diagram]
Core idea

- Apply both types of reduction
  - Rule 1: skip nodes with identical edges
  - Rule 2: skip nodes with true edge to false

- How to distinguish which rule was applied?
Core idea

- Apply both types of reduction
  - Rule 1: skip nodes with identical edges
  - Rule 2: skip nodes with true edge to false
- How to distinguish which rule was applied?
- **Solution**: a variable label (“tag”) on every edge
  - Missing nodes $x_i < x_{\text{tag}}$ due to rule 1
  - Missing nodes $x_i \geq x_{\text{tag}}$ due to rule 2
- Maximally apply both rules!
Examples

- We expect nodes between $x_i$ and $x_k$
- Missing nodes $x_i < x < x_j$ due to rule 1 (BDD)
- Missing nodes $x_j \leq x < x_k$ due to rule 2 (ZBDD)
Alternating sequences

What if one variable label is not enough?
Alternating sequences

What if one variable label is not enough?
BDDs and ZBDDs

- A set on variables $x_1$ to $x_8$
- Four items: \{00000000, 00000010, 00000100, 00000110\}.
- As a normal BDD

![Normal BDD Diagram]

- As a Zero-suppressed BDD

![Zero-suppressed BDD Diagram]
BDDs and ZBDDs

- A set on variables $x_1$ to $x_8$
- Four items: \{00000000, 00000010, 00000100, 00000110\}.
- As a normal BDD

- As a Zero-suppressed BDD

- As a Tagged BDD
TBDD rules

Bottom-up reduction rules

\[ x_i \text{ then } F(\text{Tag}) \text{ else } F(\text{Tag}) \]
TBDD rules

Bottom-up reduction rules

Note: $x_{i+1}$ is $\perp$ if $x_i$ is the last variable.

\[ x_i \text{ then } 0 \text{ else } F(x_{i+1}) \]

\[ F \]

\[ x_i \]

\[ 0 \]

\[ \Rightarrow \]

\[ F \]
Bottom-up reduction rules

\[ x_i \text{ then } 0 \text{ else } F(\text{Tag}) \]

\[ \text{Tag } \neq x_{i+1} \]

\[ \bot \Rightarrow \]

\[ x_i \]

\[ x_{i+1} \]

\[ F \]

\[ 0 \]

\[ \text{Tag} \]

\[ \text{Tag} \]
Each TBDD node is 16 bytes

- tags and variables: 20 bits
- edge indices: 32 bits
- extra bit for complementing (low edge)
Manipulating the 16-byte TBDD nodes
- Primitive for making nodes (tbdd_makenode)
  - This implements the three rules

Operations
- Binary operators
- Negation (not trivial like BDDs)
- Function domain extension (not trivial like BDDs)
- Abstraction (exists, forall)
- Relational product (for transition systems)
- Other functions (counting, etc)

Garbage collection and parallelism boilerplate (framework)
On-the-fly symbolic transition learning

- Learn a model symbolically starting from an initial state
- Interleave reachability with transition learning of new states
- Encode obtained new transitions in BDD
- Number of variables per state unknown (init to 0)
- BDDs and ZBDDs should both be strong
On-the-fly transition learning

- Compute reflexive transitive closure of $T$ applied to $S$
- Use a frontier set (only new states)

```python
def ClosureFS(S, T):
    states ← S
    frontier ← S
    while frontier ≠ ∅:
        next ← RelProd(frontier, T)
        frontier ← next \ states
        states ← states ∪ frontier
    return states
```
On-the-fly transition learning

- Compute reflexive transitive closure of $\mathcal{T}$ applied to $S$
- Use a frontier set (only new states)
- On-the-fly learning via (explicit) next-state interface

```python
def ClosureFS(S):
    states ← S
    frontier ← S
    $\mathcal{T} ← \emptyset$
    while frontier $\neq \emptyset$:
        $\mathcal{T} ← \text{Learn}(\text{frontier}, \mathcal{T})$
        next ← $\text{RelProd}(\text{frontier}, \mathcal{T})$
        frontier ← next \ states
    states ← states $\cup$ frontier
    return states, $\mathcal{T}$
```
Experimental evaluation

Results

- Implemented in Sylvan (parallel BDD package)
- Using LTSmin on the BEEM model database
- 48-core machine (4 processors \times 12 cores)
- Publicly available: http://fmv.jku.at/tbdd

<table>
<thead>
<tr>
<th></th>
<th>BDD</th>
<th>TBDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time 1 core</td>
<td>24504 sec.</td>
<td>6453 sec.</td>
</tr>
<tr>
<td>Time 48 cores</td>
<td>14672 sec.</td>
<td>1075 sec.</td>
</tr>
<tr>
<td>#Nodes in the visited set</td>
<td>59,503,837</td>
<td>5,922,973</td>
</tr>
</tbody>
</table>
Experimental evaluation

Results: BDD vs TBDD (1 core)

![Graph showing the comparison between Time (BDD) and Time (TBDD)].
Experimental evaluation

Results: TBDD speedup

![Graph showing TBDD speedup vs. time (TBDD 1) in seconds. The x-axis represents time in seconds, ranging from $10^{-2}$ to $10^3$, and the y-axis represents speedup, ranging from 0 to 50.]
Experimental evaluation

Results: Number of nodes

![Graph showing the relationship between number of nodes in BDDs and TBDDs. The x-axis represents the number of nodes in BDDs, ranging from 10^1 to 10^7, and the y-axis represents the number of nodes in TBDDs, also ranging from 10^1 to 10^7. The data points are scattered along a linear trend line, indicating a consistent growth rate.]
Experimental evaluation

Set of visited states of $\text{at}.1$

![Graph showing the number of nodes visited over iterations for different rules: none, BDD, ZBDD, TBDD. The graph indicates the performance of these rules in terms of the number of nodes.]
Experimental evaluation

Set of visited states of protocols.3.visited

Graph showing the number of nodes visited over iterations for different rules:
- none
- BDD
- ZBDD
- TBDD

The graph indicates how the number of visited states changes with each iteration for each rule.
Experimental evaluation

Set of visited states of protocols.5.visited

![Graph showing the number of nodes versus iteration for different rules: none, BDD, ZBDD, TBDD. The graph peaks around iteration 75 and then decreases.](image)

Rule
- none
- BDD
- ZBDD
- TBDD
Experimental evaluation

Set of visited states of lifts.1

Rule
- none
- BDD
- ZBDD
- TBDD

Tom van Dijk & Robert Wille & Robert Meolic
Tagged BDDs
Experimental evaluation

Set of visited states of lamport.1

![Graph showing the set of visited states of lamport.1 for different rules: none, BDD, ZBDD, and TBDD. The x-axis represents iteration, and the y-axis represents the number of nodes. The graph illustrates the performance of each rule over iteration, with none showing the highest nodes count.](image-url)
Experimental evaluation

Set of visited states of \textit{krebs.1}

![Graph showing the set of visited states of \textit{krebs.1} for different rules.](image)

- **Rule**: none, BDD, ZBDD, TBDD

- **Graph**:
  - Y-axis: Number of nodes
  - X-axis: Iteration
  - Different line styles and colors for each rule:
    - none (solid line)
    - BDD (red dashed line)
    - ZBDD (blue dotted line)
    - TBDD (purple dotted line)
Experimental evaluation

Set of visited states of exit.1

![Graph showing the set of visited states over iterations for different rules. The x-axis represents the iteration, and the y-axis represents the number of nodes. The graph compares different rules: none, BDD, ZBDD, and TBDD. The BDD rule shows a significant increase in the number of nodes compared to the other rules, peaking around iteration 30 before decreasing. The ZBDD and TBDD rules show a steady increase, with ZBDD starting higher than TBDD.]
Experimental evaluation

Set of visited states of fischer.2

![Graph showing the set of visited states for fischer.2 with different rule sets: none, BDD, ZBDD, and TBDD. The graph plots the number of nodes against iteration.]
Contributions and Conclusions

▶ Tagged BDDs to combine BDD and ZBDD rules.
▶ Implemented in Sylvan with multi-core implementation
  https://www.github.com/trolando/sylvan
▶ Faster than normal BDDs on BEEM models
▶ Good parallel performance
▶ BEEM models disappointing w.r.t. BDD rule

Future Directions

▶ Apply to better symbolic models, e.g., Petri nets?
▶ Study other rules and more complicated rules
▶ Dynamic variable reordering and other operations

Supported by FWF, RiSE, COST Act. IC1405,
JKU, University of Maribor, FMT Twente