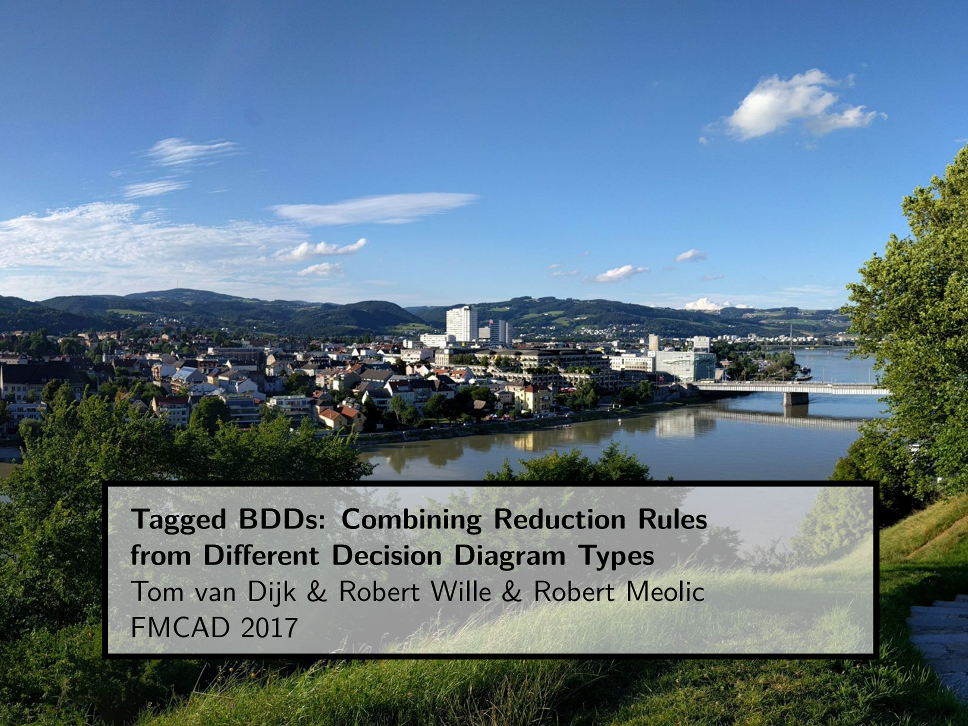


Tagged BDDs: Combining Reduction Rules from Different Decision Diagram Types

Tom van Dijk & Robert Wille & Robert Meolic

FMCAD 2017

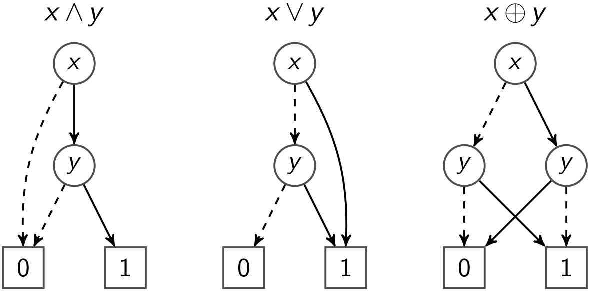
TL; DR

- 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 $\mathbb{B}^k \to \mathbb{B}$ function or a $S \subseteq \mathbb{B}^k$ set
- Paths represent valuations of $\mathbb{B}^k$
- For sets: path to $\boxed{1}$, then valuation is in the set

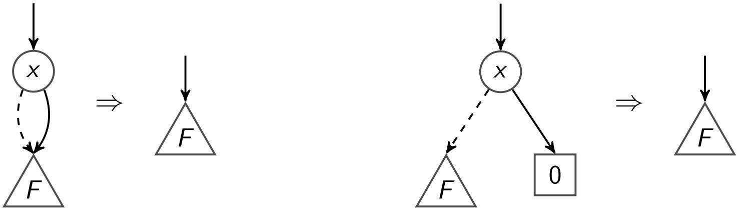


BDDs and ZBDDs

Zero-suppressed BDDs

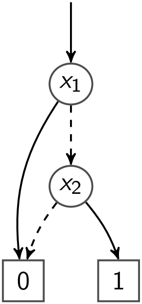
- Different "reduction rule"
  - BDD: skipped variables are dont-cares
  - ZBDD: skipped variables are set to false
- Different applications: sparse matrices, subsets

BDD reduction rule	ZBDD reduction rule

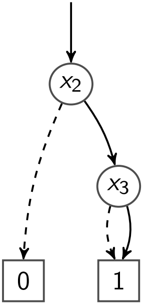


BDDs and ZBDDs

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



(a) BDD



(b) ZBDD

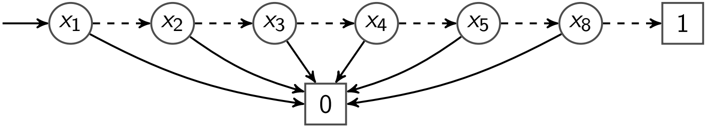
Example

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

BDDs and ZBDDs

Example

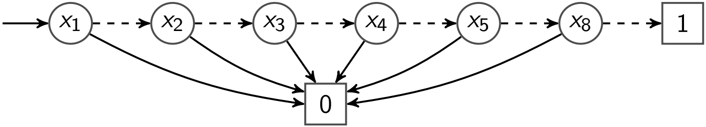
- A set on variables x_1 to x_8
- Four items: $\{00000000, 00000010, 00000100, 00000110\}$.
- As a normal BDD



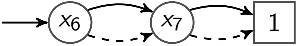
BDDs and ZBDDs

Example

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



- As a Zero-suppressed BDD



Tagged BDDs

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?

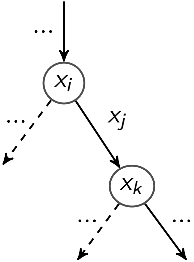
Tagged BDDs

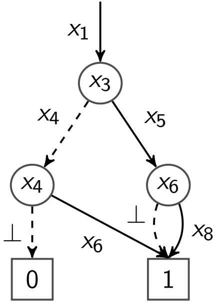
Core idea

- 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

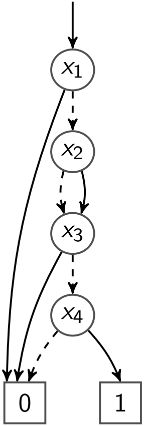
- We expect nodes between x_i and x_k
- Missing nodes $x_i < x < x_j$ due to rule 1 (BDD)
- Missing nodes $x_j \leq x < x_k$ due to rule 2 (ZBDD)





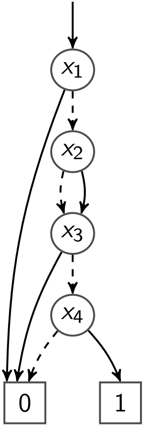
Alternating sequences

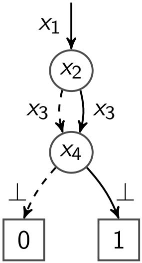
What if one variable label is not enough?



Alternating sequences

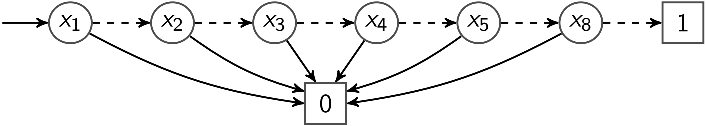
What if one variable label is not enough?



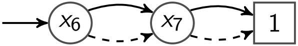


BDDs and ZBDDs

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

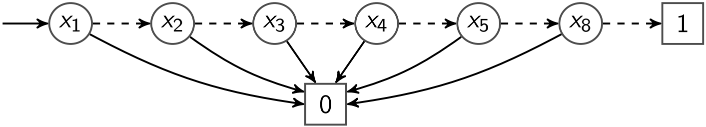


- As a Zero-suppressed BDD

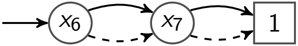


BDDs and ZBDDs

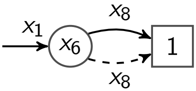
- A set on variables x_1 to x_8
- Four items: $\{00000000, 00000010, 00000100, 00000110\}$.
- As a normal BDD



- As a Zero-suppressed BDD



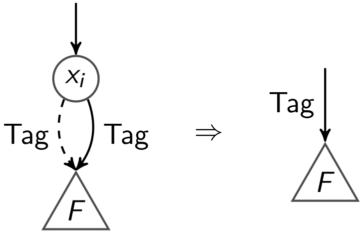
- As a Tagged BDD



TBDD rules

Bottom-up reduction rules

x_i then F(Tag) else F(Tag)

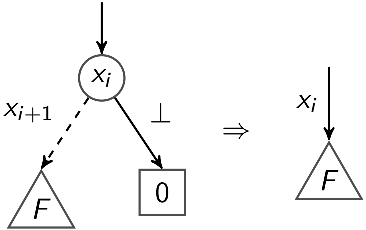


TBDD rules

Bottom-up reduction rules

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

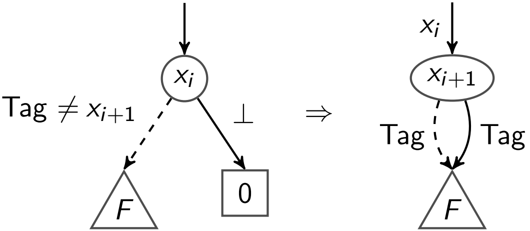
x_i then 0 else $F(x_{i+1})$



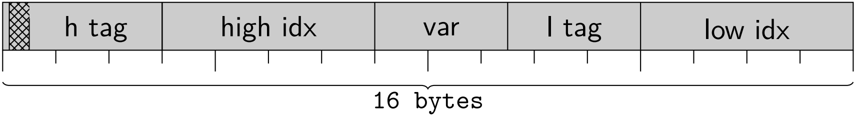
TBDD rules

Bottom-up reduction rules

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



TBDD node layout



Each TBDD node is 16 bytes

- tags and variables: 20 bits
- edge indices: 32 bits
- extra bit for complementing (low edge)

Implementation

- 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)

Outline

Application

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 $\mathcal{T}$ applied to $\mathcal{S}$
- Use a frontier set (only new states)

```
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 $\mathcal{S}$
- Use a frontier set (only new states)
- On-the-fly learning via (explicit) next-state interface

```
def ClosureFS(𝒮):
    states ← 𝒮
    frontier ← 𝒮
    𝒯 ← ∅
    while frontier ≠ ∅:
        𝒯 ← Learn(frontier, 𝒯)
        next ← RelProd(frontier, 𝒯)
        frontier ← next \ states
        states ← states ∪ frontier
    return states, 𝒯
```

Outline

Experimental evaluation

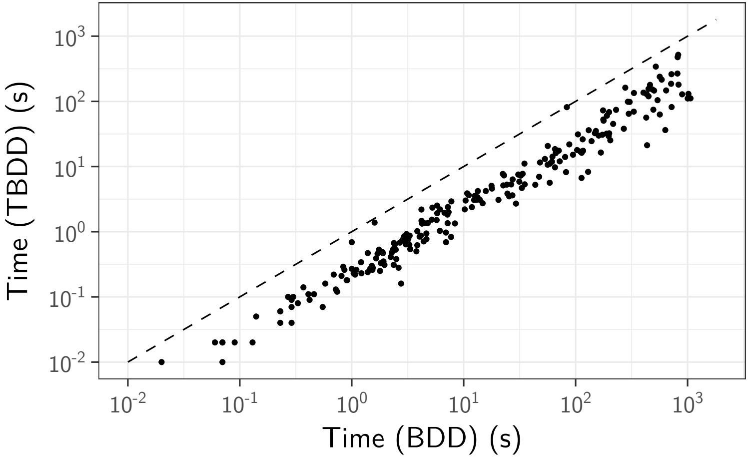
Results

- 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`

	BDD	TBDD
Time 1 core	24504 sec.	6453 sec.
Time 48 cores	14672 sec.	1075 sec.
#Nodes in the visited set	59,503,837	5,922,973

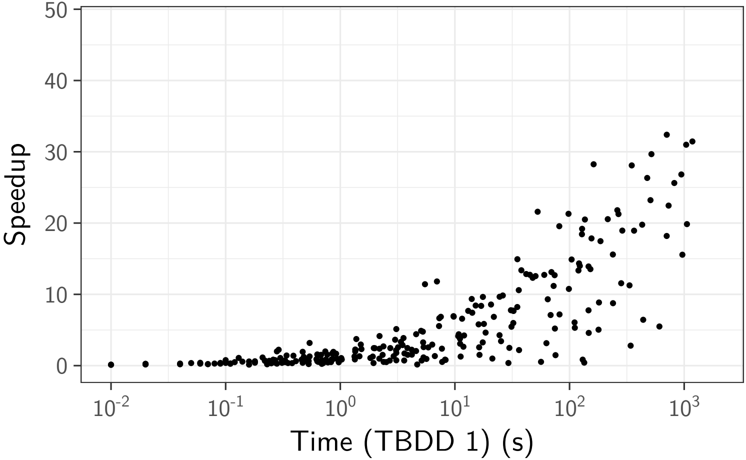
Experimental evaluation

Results: BDD vs TBDD (1 core)

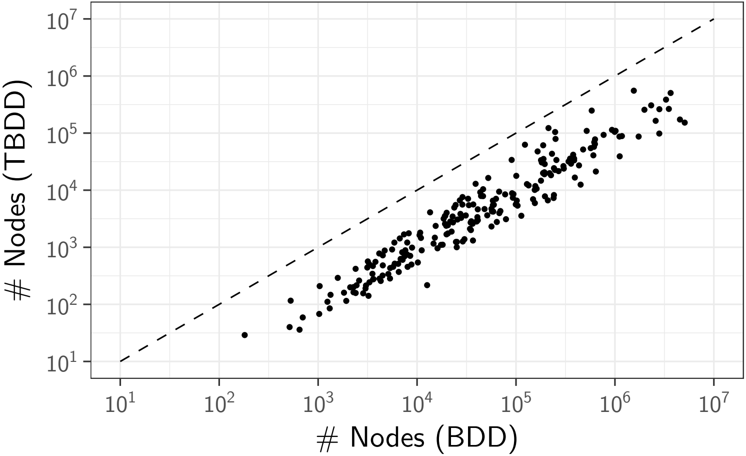


Experimental evaluation

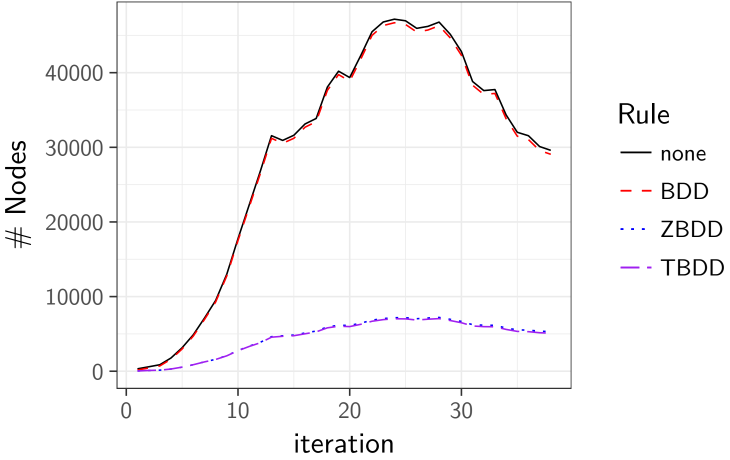
Results: TBDD speedup



Results: Number of nodes

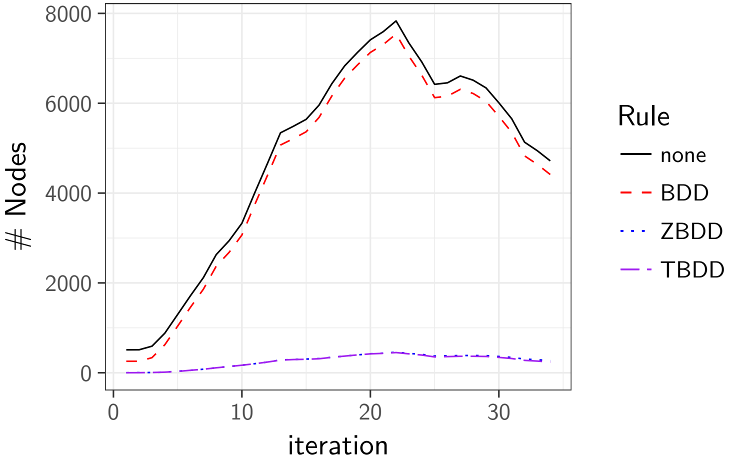


Set of visited states of `at.1`



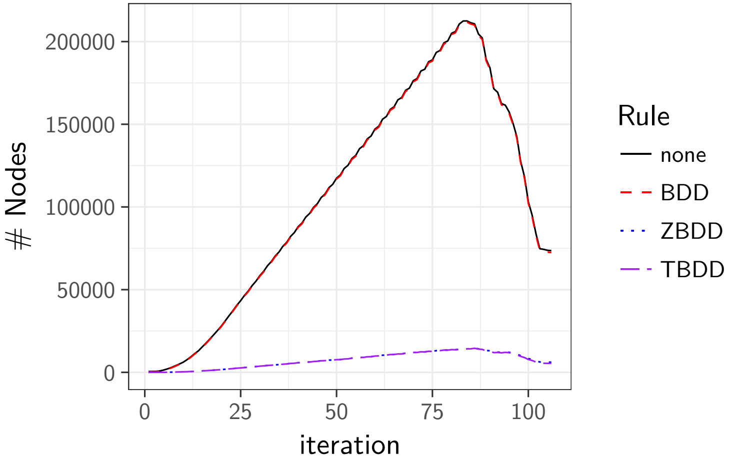
Experimental evaluation

Set of visited states of `protocols.3.visited`



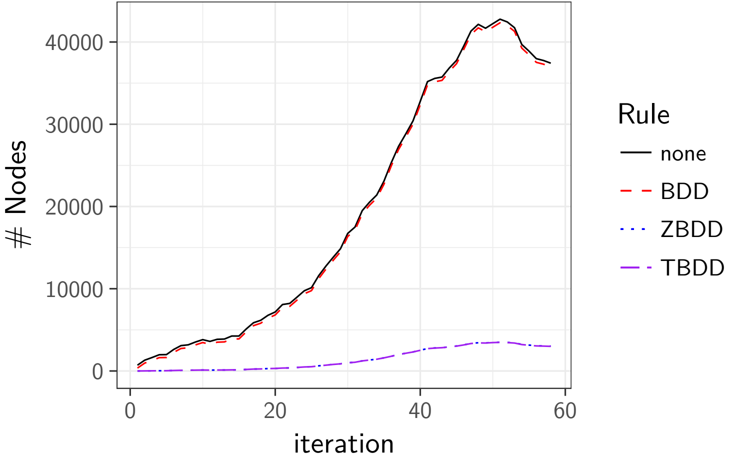
Experimental evaluation

Set of visited states of `protocols.5.visited`



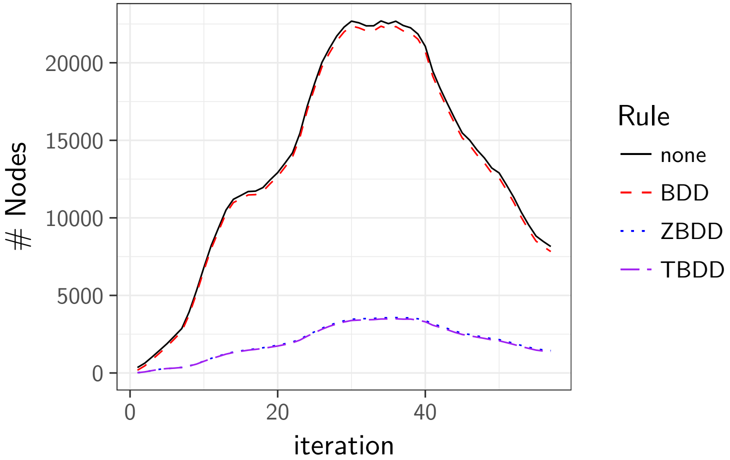
Experimental evaluation

Set of visited states of `lifts.1`



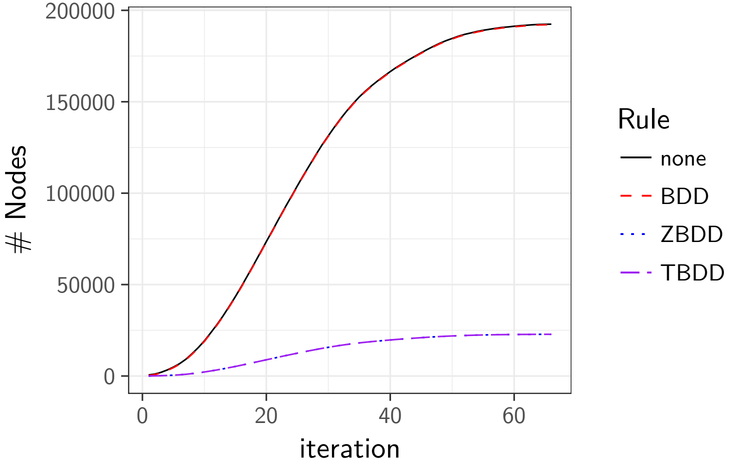
Experimental evaluation

Set of visited states of `lamport.1`



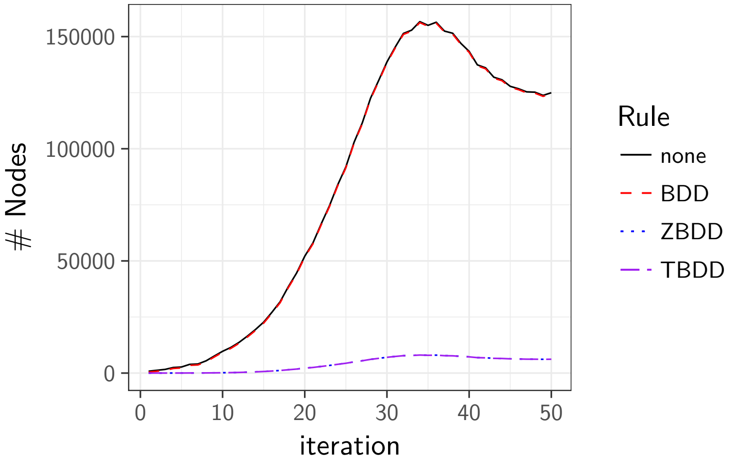
Experimental evaluation

Set of visited states of `krebs.1`



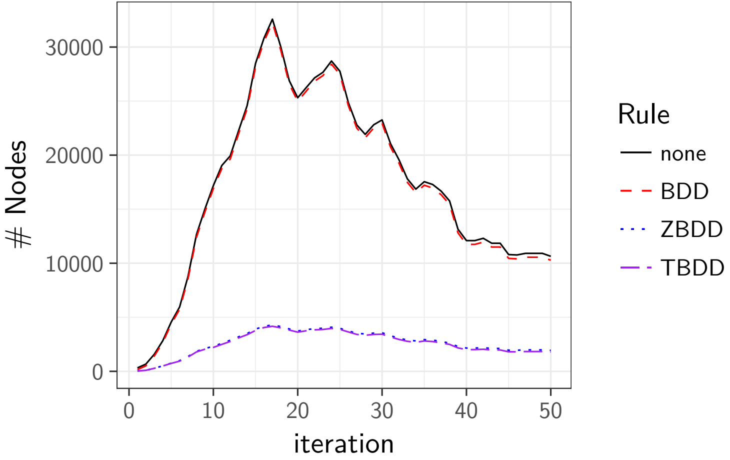
Experimental evaluation

Set of visited states of `exit.1`



Experimental evaluation

Set of visited states of `fischer.2`



Outline

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