

# Relational Product of BDDs in External Memory

---

**Steffan Christ Sølvesten**, Jaco van de Pol

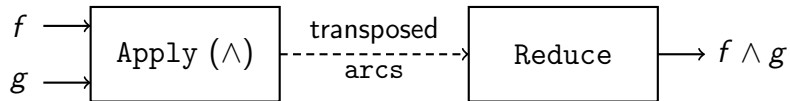
SPIN 2025

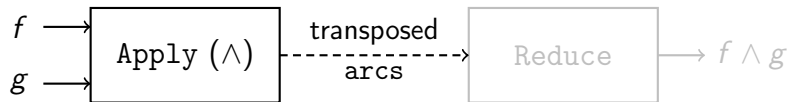


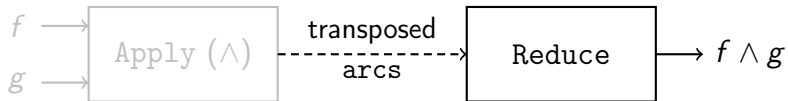
$$Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'}) \equiv ( \exists \vec{x}. S_{\vec{x}} \wedge T_{\vec{x}, \vec{x}'} )[\vec{x}' / \vec{x}]$$

[TACAS 22]

$$Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'}) \equiv ( \exists \vec{x}. S_{\vec{x}} \wedge T_{\vec{x}, \vec{x}'} ) [\vec{x}' / \vec{x}]$$



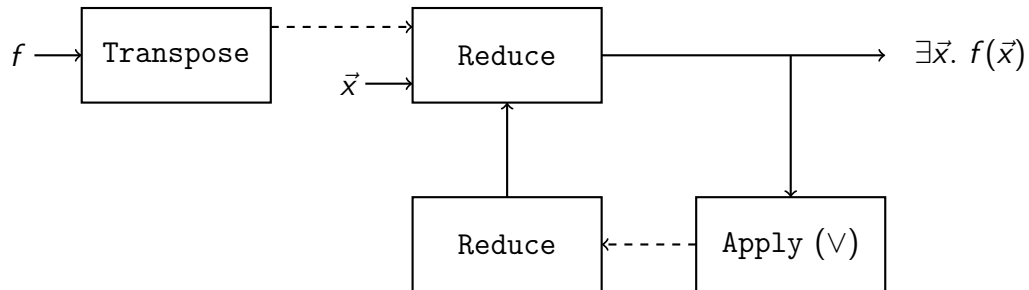




[TACAS 22]

$$\text{Next}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'}) \equiv ( \exists \vec{x}. S_{\vec{x}} \wedge T_{\vec{x}, \vec{x}'} ) [\vec{x}' / \vec{x}]$$

[TACAS 25]





[TACAS 22]

$$Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'}) \equiv ( \exists \vec{x}. S_{\vec{x}} \wedge T_{\vec{x}, \vec{x}'} ) [\vec{x}' / \vec{x}]$$

[TACAS 25]

$$\begin{array}{c}
\text{[TACAS 22]} \\
| \\
\text{Next}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'}) \equiv ( \underbrace{\exists \vec{x}. S_{\vec{x}} \wedge}_{\text{[TACAS 25]}} T_{\vec{x}, \vec{x}'} ) \underbrace{[\vec{x}' / \vec{x}]}_{\text{[SPIN 25]}}
\end{array}$$

# Replace

## Definition

A relabelling  $\pi$  is monotonic if  $x_i < x_j \implies \pi(x_i) < \pi(x_j)$

## Lemma

*If  $\pi$  is monotonic, then the BDD  $f(\vec{x})$  is isomorphic to  $f(\pi(\vec{x}))$ .*

# Replace

## Definition

A relabelling  $\pi$  is monotonic if  $x_i < x_j \implies \pi(x_i) < \pi(x_j)$

## Lemma

*If  $\pi$  is monotonic, then the BDD  $f(\vec{x})$  is isomorphic to  $f(\pi(\vec{x}))$ .*

- One can apply  $\pi$  in a single linear scan.

$\mathcal{O}(N)$  time,  $2 \cdot \text{scan}(N)$  I/Os, and  $N$  external space.

# Replace

## Definition

A relabelling  $\pi$  is monotonic if  $x_i < x_j \implies \pi(x_i) < \pi(x_j)$

## Lemma

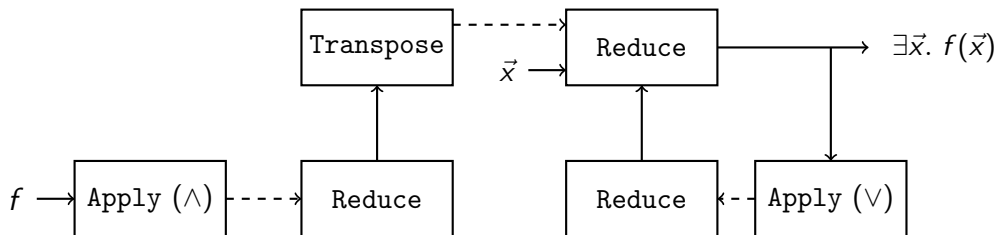
*If  $\pi$  is monotonic, then the BDD  $f(\vec{x})$  is isomorphic to  $f(\pi(\vec{x}))$ .*

- **One can apply  $\pi$  in a single linear scan.**  
 $\mathcal{O}(N)$  time,  $2 \cdot \text{scan}(N)$  I/Os, and  $N$  external space.
- **One can incorporate  $\pi$  into a (succeeding) top-down Apply sweep.**  
 $\mathcal{O}(N)$  time, 0 I/Os, and 0 external space.
- **One can incorporate  $\pi$  into a (preceeding) bottom-up Reduce sweep.**  
 $\mathcal{O}(n)$  time, 0 I/Os, and 0 external space.

# AndExists

## Observation

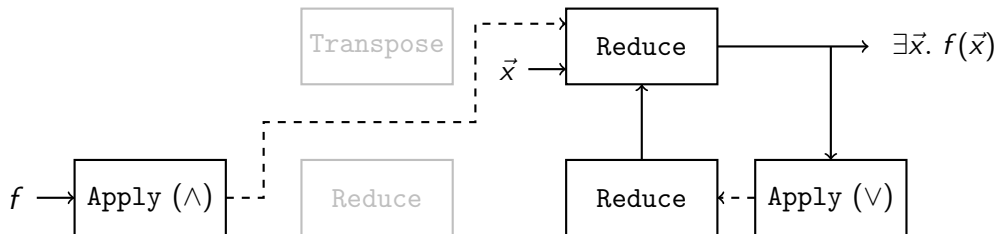
The I/O-efficient And [TACAS 22] and Exists [TACAS 25] operations can be merged:



# AndExists

## Observation

The I/O-efficient And [TACAS 22] and Exists [TACAS 25] operations can be merged:



# AndExists

## Observation

The I/O-efficient And [TACAS 22] and Exists [TACAS 25] operations can be merged:

- **The outer accumulating Reduce sweep of the Exists can do double-duty as the Reduce sweep of the preceeding And operation.**

This saves  $\Theta(\text{sort}(N))$  time and I/Os.



# AndExists

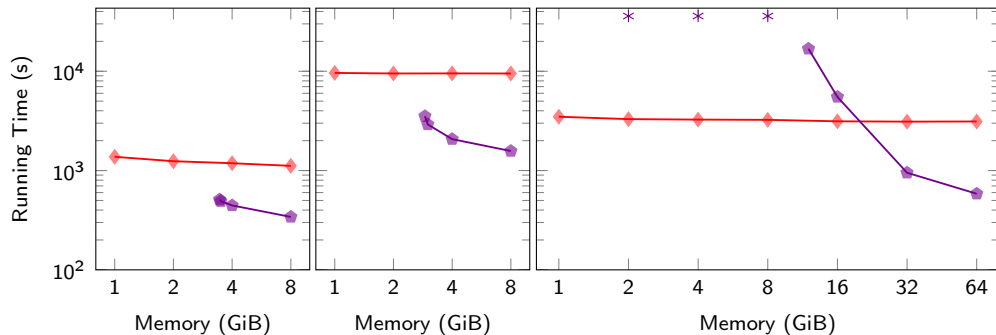
## Observation

The I/O-efficient And [TACAS 22] and Exists [TACAS 25] operations can be merged:

- **The outer accumulating Reduce sweep of the Exists can do double-duty as the Reduce sweep of the preceeding And operation.**  
This saves  $\Theta(\text{sort}(N))$  time and I/Os.
- **The And operation can prune subtrees that trivially will become redundant during the succeeding Exists.**  
This can save up to  $\mathcal{O}(\text{sort}(N^{2^k}))$  time and I/Os.  
In practice, this only saves up to  $\mathcal{O}(\text{sort}(N))$  time and I/Os.



## Experiment: $\text{Next}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$



(a) GPUForwardProgress 20a

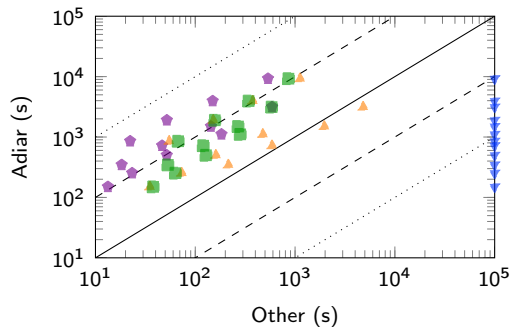
(b) SmartHome 16

(c) ShieldPPPs 10a

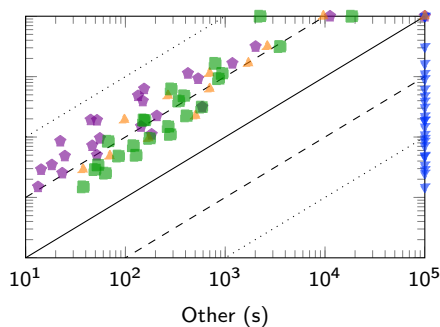
—◆— Adiar —■— BuDDy

Relational Product for MCC models with a  $2^{25}$  state space BDD. Timeouts are marked as stars.

## Experiment: $\text{Next}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$ & $\text{Prev}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$



(a)  $\text{Next}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$

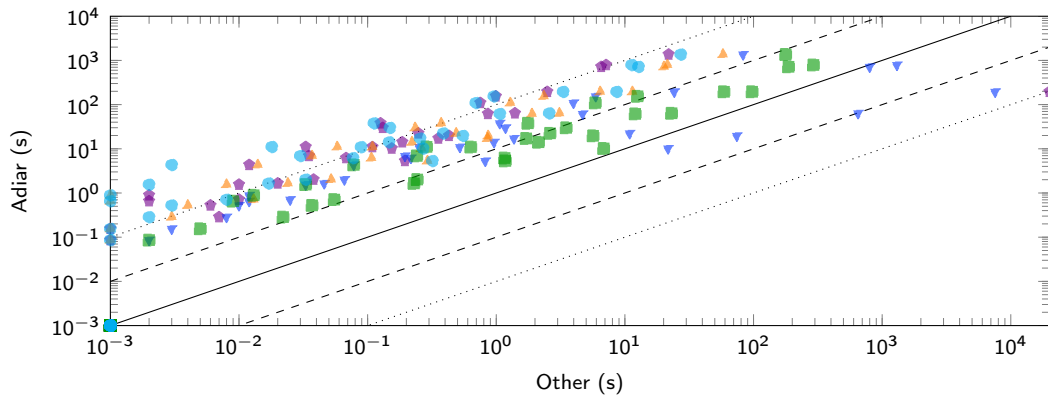


(b)  $\text{Prev}(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$

◆ BuDDy ▼ CAL ▲ CUDD ■ LibBDD

Relational Product for MCC models, 384 GiB of memory, and  $2^{22}, \dots, 2^{25}$  state space BDDs.

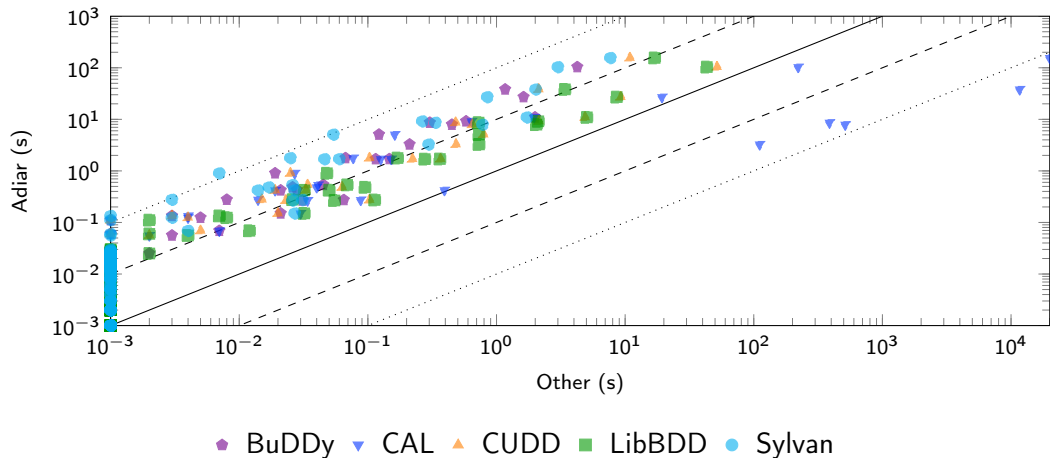
## Experiment: Reachability



◆ BuDDy ▼ CAL ▲ CUDD ■ LibBDD ● Sylvan

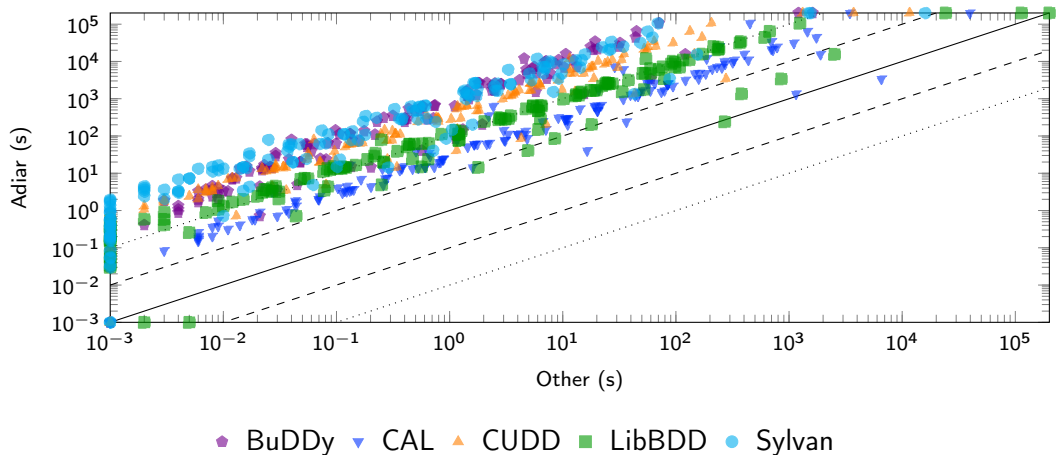
16 Petri Nets [MCC 21–23] with 384 GiB of memory.

## Experiment: Deadlock Detection



16 Petri Nets [MCC 21–23] and 59 Boolean Networks [AEON, PyBoolNet] with 384 GiB RAM.

## Experiment: SCC Decomposition



16 Petri Nets [MCC 21–23] and 59 Boolean Networks [AEON, PyBoolNet] with 384 GiB RAM.

# Conclusions and Future Work

- To improve the I/O-efficient  $Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$ , focus on *AndExists*.
  - Factor of  $\sim 2\times$  by using a *AndExists* instead of *And* and *Exists* for conventional depth-first implementations [1]. This may explain the sudden performance gap.
  - For larger instances, less than  $\frac{1}{10}$ th of the time is spent on the *And*.

- 
- [1] Van Dijk et al.: *A Comparative Study of BDD packages for Probabilistic Symbolic Model Checking*. (2015)
  - [2] Van Dijk: *Sylvan – Multi-core Decision Diagrams*. (2016)
  - [3] Van Dijk et al.: *Multi-core on-the-fly saturation*. (2019)
  - [4] Brand et al.: *A Decision Diagram Operation for Reachability*. (2023).
  - [5] Marmorstein & Siminiceanu: *The Saturation algorithm for Symbolic State-space Exploration*. (2006).



# Conclusions and Future Work

- To improve the I/O-efficient  $Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$ , focus on *AndExists*.
  - Factor of  $\sim 2\times$  by using a *AndExists* instead of *And* and *Exists* for conventional depth-first implementations [1]. This may explain the sudden performance gap.
  - For larger instances, less than  $\frac{1}{10}$ th of the time is spent on the *And*.
- Deal with small BDDs using Depth-first Recursion.

- 
- [1] Van Dijk et al.: *A Comparative Study of BDD packages for Probabilistic Symbolic Model Checking*. (2015)
  - [2] Van Dijk: *Sylvan – Multi-core Decision Diagrams*. (2016)
  - [3] Van Dijk et al.: *Multi-core on-the-fly saturation*. (2019)
  - [4] Brand et al.: *A Decision Diagram Operation for Reachability*. (2023).
  - [5] Marmorstein & Siminiceanu: *The Saturation algorithm for Symbolic State-space Exploration*. (2006).

## Conclusions and Future Work

- To improve the I/O-efficient  $Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$ , focus on *AndExists*.
  - Factor of  $\sim 2\times$  by using a *AndExists* instead of *And* and *Exists* for conventional depth-first implementations [1]. This may explain the sudden performance gap.
  - For larger instances, less than  $\frac{1}{10}$ th of the time is spent on the *And*.
- Deal with small BDDs using Depth-first Recursion.
- Apply ideas from recent and more advanced BDD algorithms.

For example the ones in [2], [3], [4], and [5] .

- 
- [1] Van Dijk et al.: *A Comparative Study of BDD packages for Probabilistic Symbolic Model Checking*. (2015)
  - [2] Van Dijk: *Sylvan – Multi-core Decision Diagrams*. (2016)
  - [3] Van Dijk et al.: *Multi-core on-the-fly saturation*. (2019)
  - [4] Brand et al.: *A Decision Diagram Operation for Reachability*. (2023).
  - [5] Marmorstein & Siminiceanu: *The Saturation algorithm for Symbolic State-space Exploration*. (2006).

## Conclusions and Future Work

- To improve the I/O-efficient  $Next(S_{\vec{x}}, T_{\vec{x}, \vec{x}'})$ , focus on *AndExists*.
  - Factor of  $\sim 2\times$  by using a *AndExists* instead of *And* and *Exists* for conventional depth-first implementations [1]. This may explain the sudden performance gap.
  - For larger instances, less than  $\frac{1}{10}$ th of the time is spent on the *And*.
- Deal with small BDDs using Depth-first Recursion.
- Apply ideas from recent and more advanced BDD algorithms.  
For example the ones in [2], [3], [4], and [5] .
- Design a  $Replace(\pi)$  for Non-monotone Variable Substitutions.

- 
- [1] Van Dijk et al.: *A Comparative Study of BDD packages for Probabilistic Symbolic Model Checking*. (2015)
  - [2] Van Dijk: *Sylvan – Multi-core Decision Diagrams*. (2016)
  - [3] Van Dijk et al.: *Multi-core on-the-fly saturation*. (2019)
  - [4] Brand et al.: *A Decision Diagram Operation for Reachability*. (2023).
  - [5] Marmorstein & Siminiceanu: *The Saturation algorithm for Symbolic State-space Exploration*. (2006).

# Steffan Christ Sølvsten

---

✉ [soelvsten@cs.au.dk](mailto:soelvsten@cs.au.dk)

🌐 [ssoelvsten.github.io](https://ssoelvsten.github.io)

## Adiar

---

🔗 [github.com/ssoelvsten/adiar](https://github.com/ssoelvsten/adiar)

📖 [ssoelvsten.github.io/adiar](https://ssoelvsten.github.io/adiar)

