Multi-variable Quantification of BDDs in External Memory using Nested Sweeping

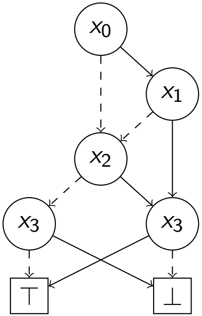
Steffan Christ Sølvsten, Jaco van de Pol

TACAS 2025

AARHUS UNIVERSITY

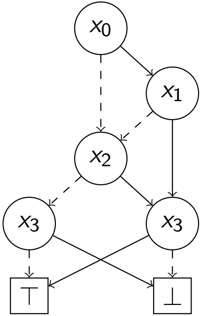
1986

Binary Decision Diagrams



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

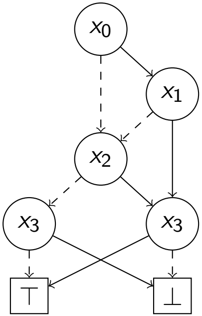
Binary Decision Diagrams



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

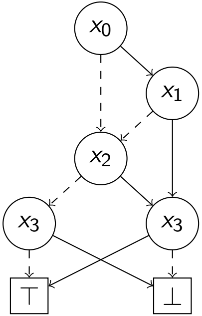
Theorem (Bryant '86)
Given a fixed variable order, a (Reduced Ordered) Binary Decision Diagram is a unique canonical representation of a Boolean function.

Binary Decision Diagrams



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Binary Decision Diagrams



(a) $(x_0 \wedge x_1 \wedge x_3) \vee (x_2 \oplus x_3)$

Theorem (Bryant '86)
Given BDDs ϕ and ψ, $\phi \odot \psi$ is computible in $\mathcal{O}(|\phi| \cdot |\psi|)$ time.

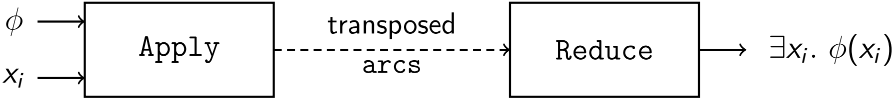
Theorem (Bryant '86)
Given BDD ϕ and Boolean b, $\phi[x_i \mapsto b]$ is computible in $\mathcal{O}(|\phi|)$ time.

Corollary
Given BDD ϕ, $\exists x_i.\ \phi(x)$ requires $\mathcal{O}(|\phi|^2)$ time.

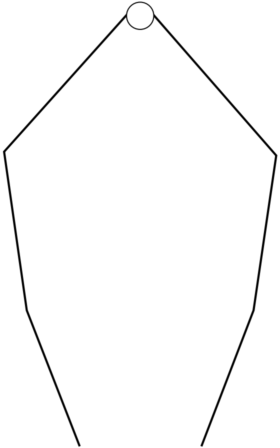
Proof.
$\exists x_i.\phi(x_i) \equiv \phi[x_i \mapsto \bot] \vee \phi[x_i \mapsto \top]$ $\square$

1996

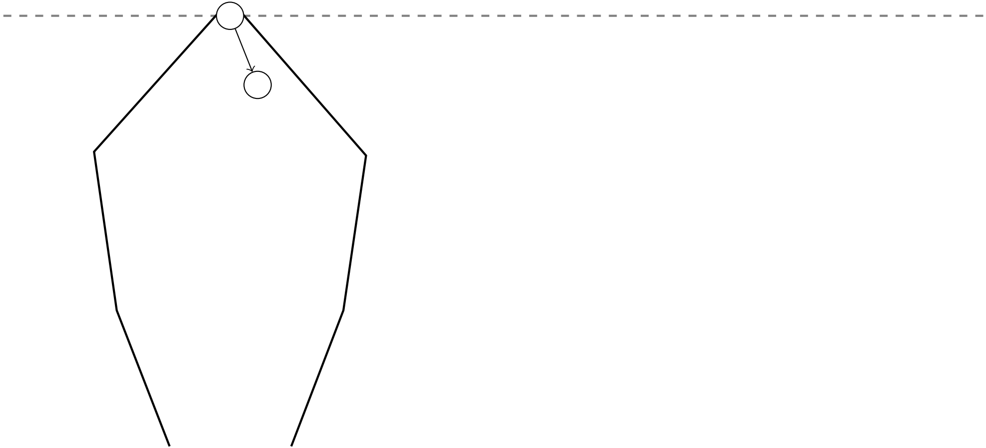
$\exists x_i.\ \phi(x_i)$



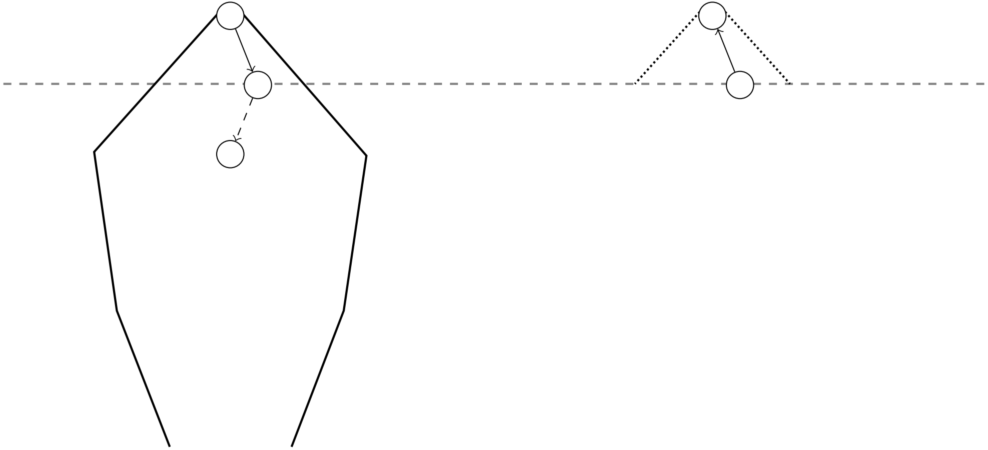
$\exists x_i.\ \phi(x_i)$: `Apply`



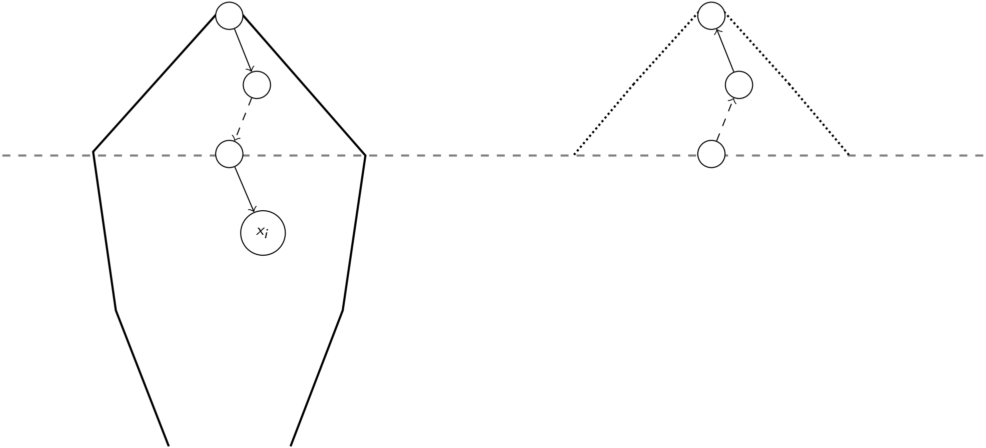
$\exists x_i.\ \phi(x_i)$: `Apply`



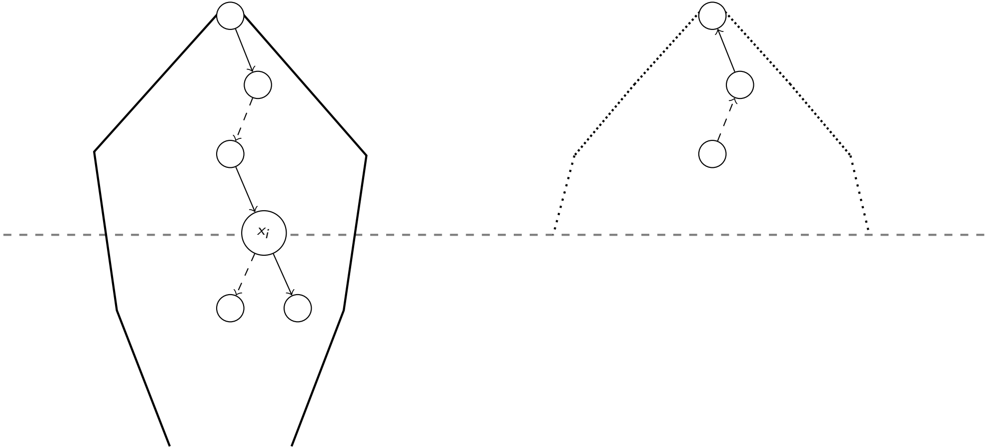
$\exists x_i.\ \phi(x_i)$: `Apply`



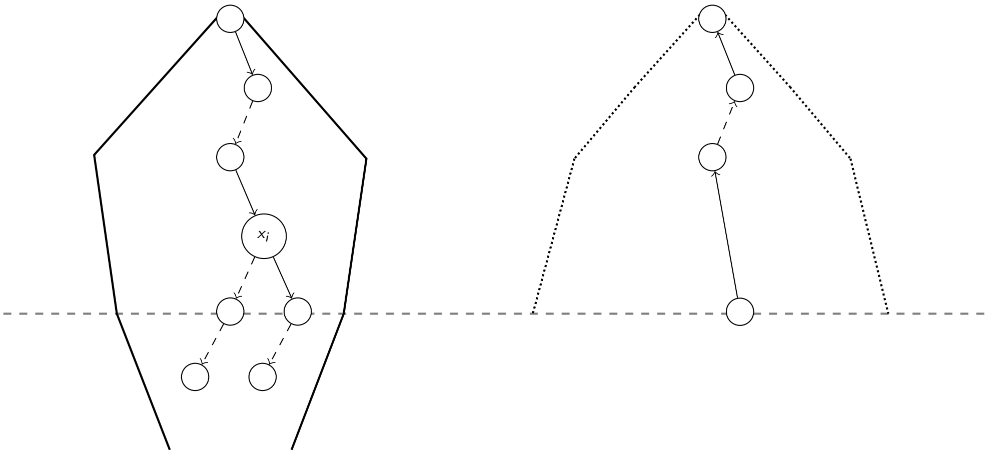
$\exists x_i.\ \phi(x_i)$: `Apply`



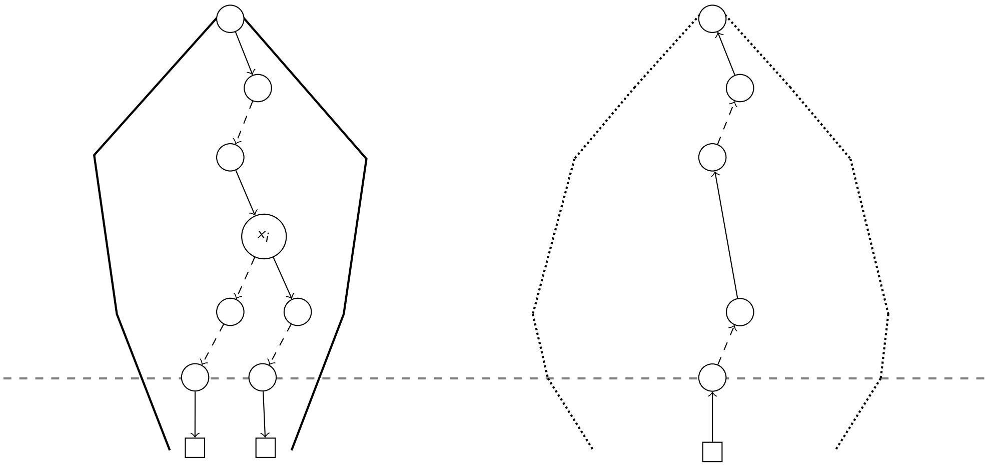
$\exists x_i.\ \phi(x_i)$: `Apply`



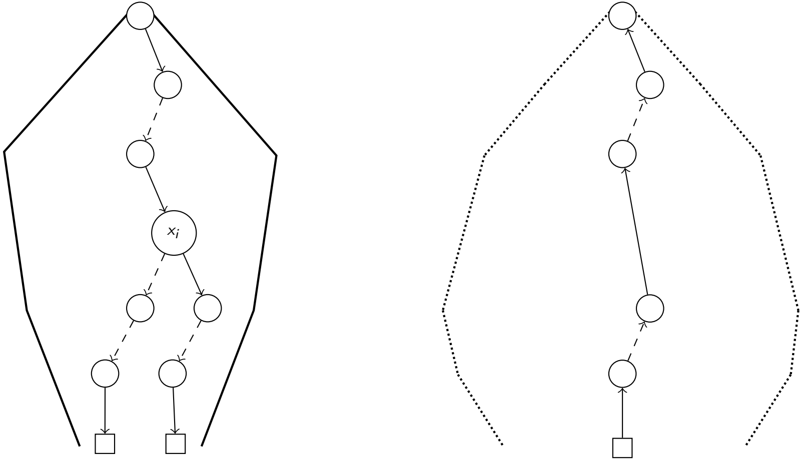
$\exists x_i.\ \phi(x_i)$: `Apply`



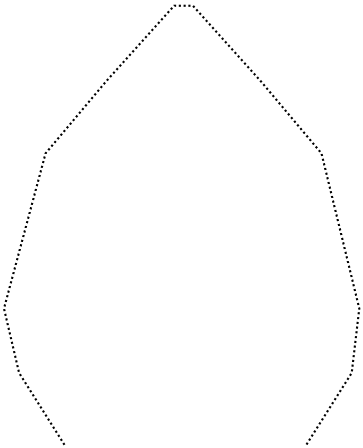
$\exists x_i.\ \phi(x_i)$: `Apply`



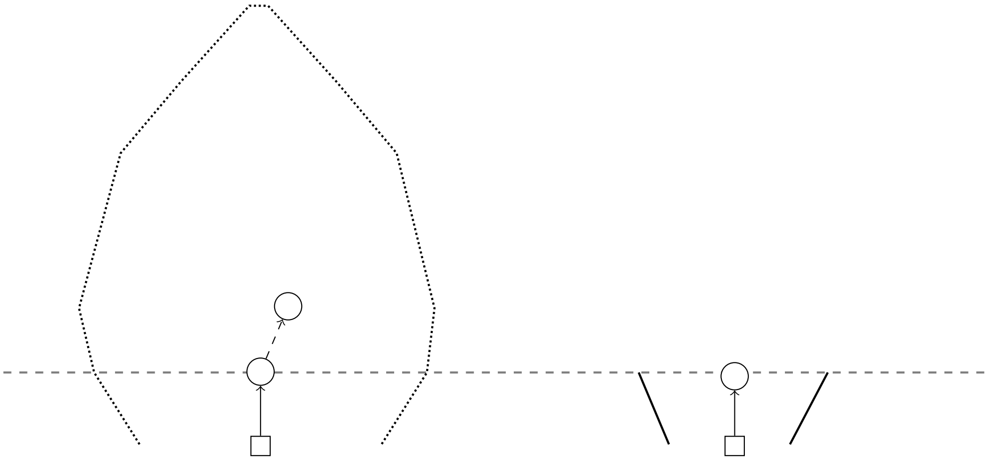
$\exists x_i.\ \phi(x_i)$: `Apply`



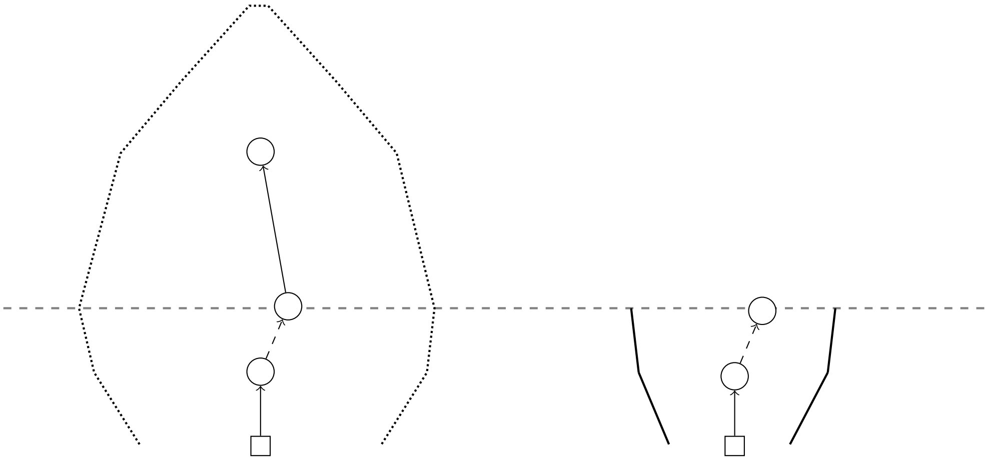
$\exists x_i.\ \phi(x_i)$: `Reduce`



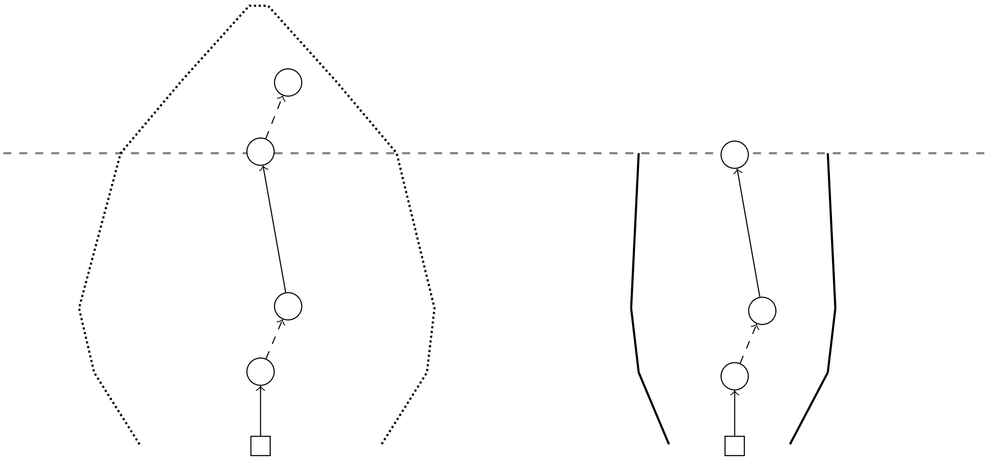
$\exists x_i.\ \phi(x_i)$: `Reduce`



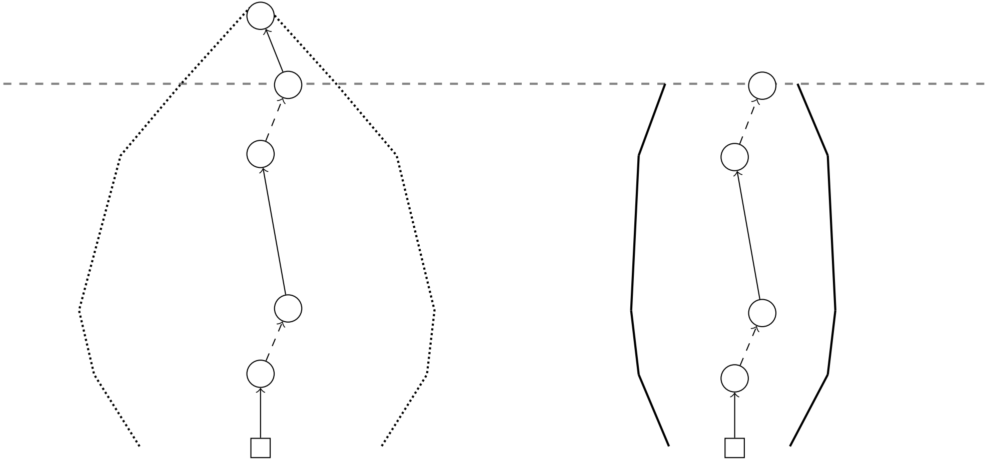
$\exists x_i.\ \phi(x_i)$: `Reduce`



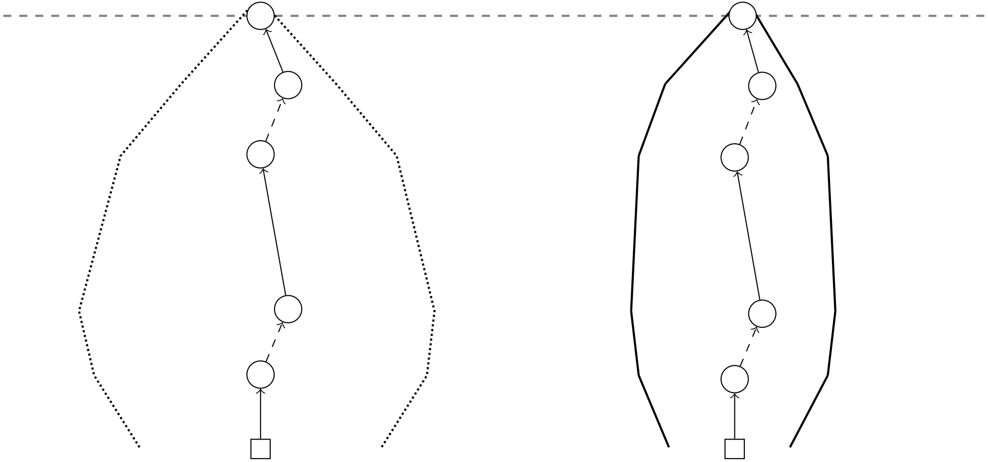
$\exists x_i.\ \phi(x_i)$: `Reduce`



$\exists x_i.\ \phi(x_i)$: `Reduce`



$\exists x_i.\ \phi(x_i)$: `Reduce`



$\exists x_i .\ \phi(x_i)$

Theorem (Lars Arge '96)
Given BDDs ϕ and ψ, $\phi \odot \psi$ is computible in $\mathcal{O}(sort(|\phi| \cdot |\psi|))$ time and I/Os.

Theorem (Sølvsten et al. '22)
Given BDD ϕ and Boolean b, $\phi[x_i \mapsto b]$ is computible in $\mathcal{O}(sort(|\phi|))$ time and I/Os.

$\exists x_i.\ \phi(x_i)$

Theorem (Lars Arge '96)
Given BDDs ϕ and ψ, $\phi \odot \psi$ is computible in $\mathcal{O}(sort(|\phi| \cdot |\psi|))$ time and I/Os.

Theorem (Sølvsten et al. '22)
Given BDD ϕ and Boolean b, $\phi[x_i \mapsto b]$ is computible in $\mathcal{O}(sort(|\phi|))$ time and I/Os.

Corollary (Søelvsten et al. '22)
Given BDD ϕ, the time and I/O complexity of quantification is

- $\mathcal{O}(sort(|\phi|^2))$ *for a single variable.*
- $\mathcal{O}(sort(|\phi|^{2^k}))$ *for k variables.*

2022

Adiar

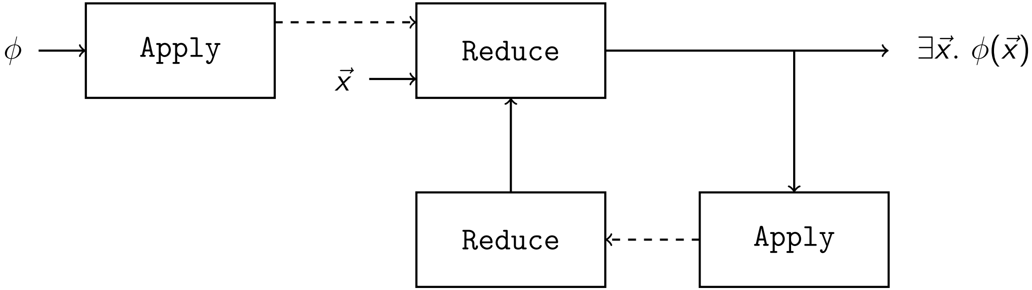
I/O-efficient Decision Diagrams

---

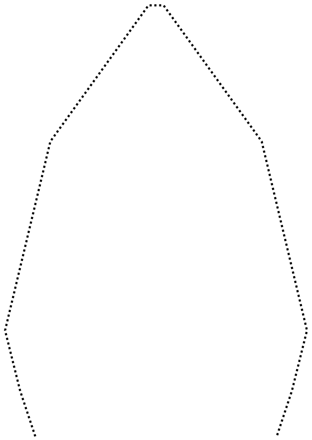
github.com/ssoelvsten/adiar

2025

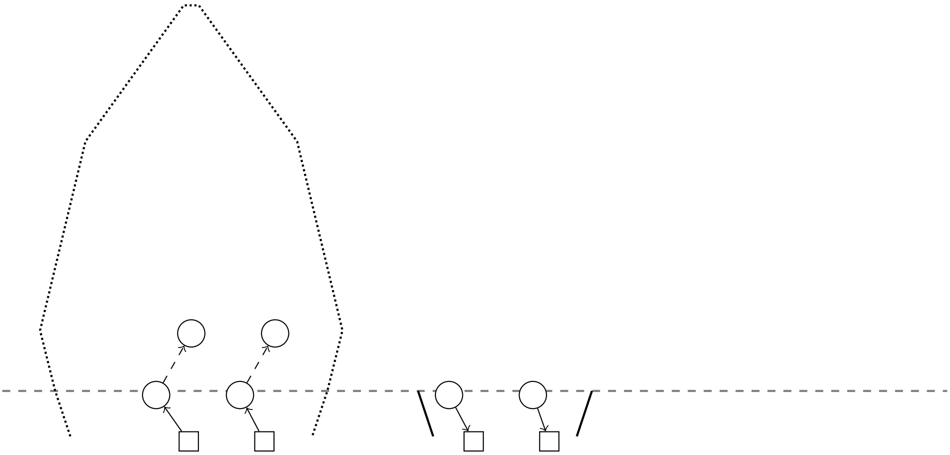
$\exists \vec{x}.\ \phi(\vec{x})$



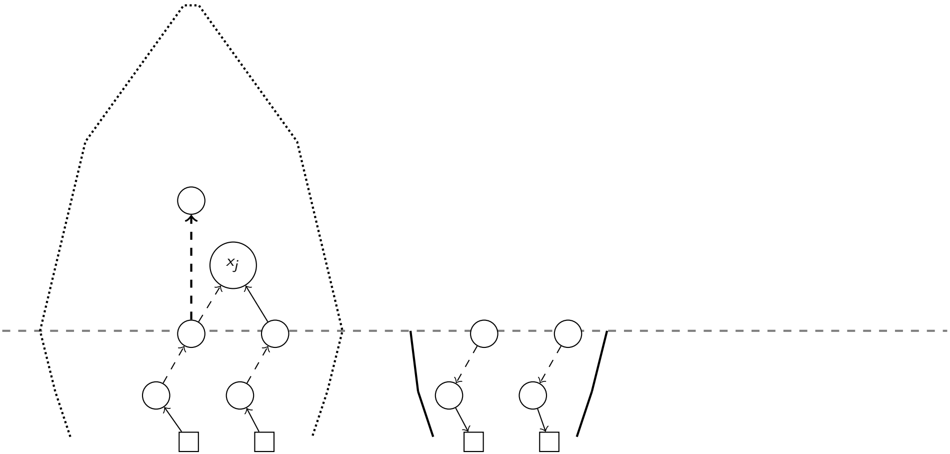
$\exists \vec{x}.\ \phi(\vec{x})$



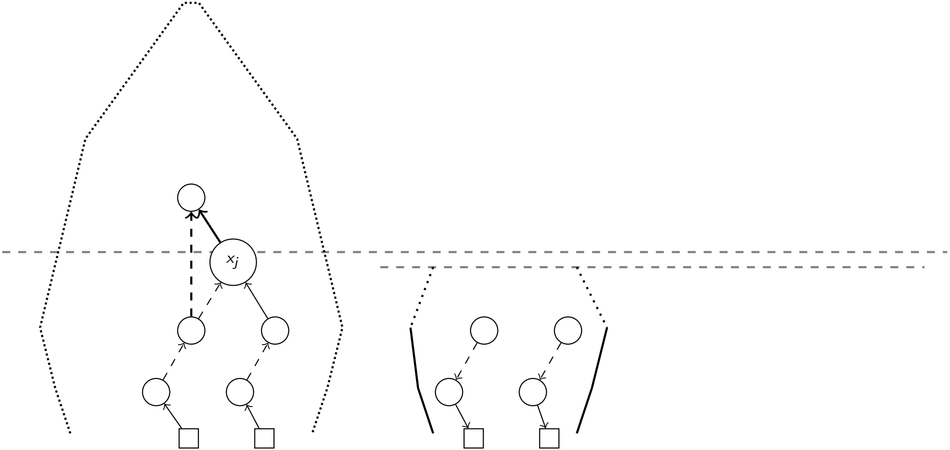
$\exists \vec{x}.\ \phi(\vec{x})$



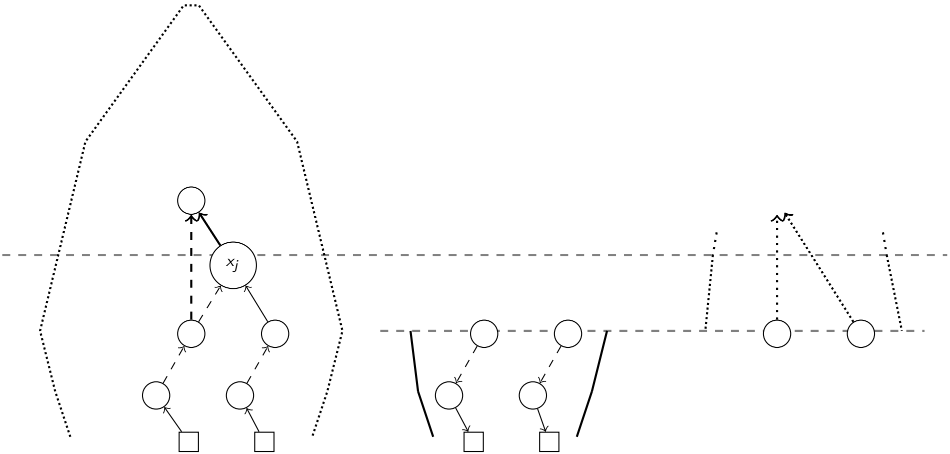
$\exists \vec{x}.\ \phi(\vec{x})$



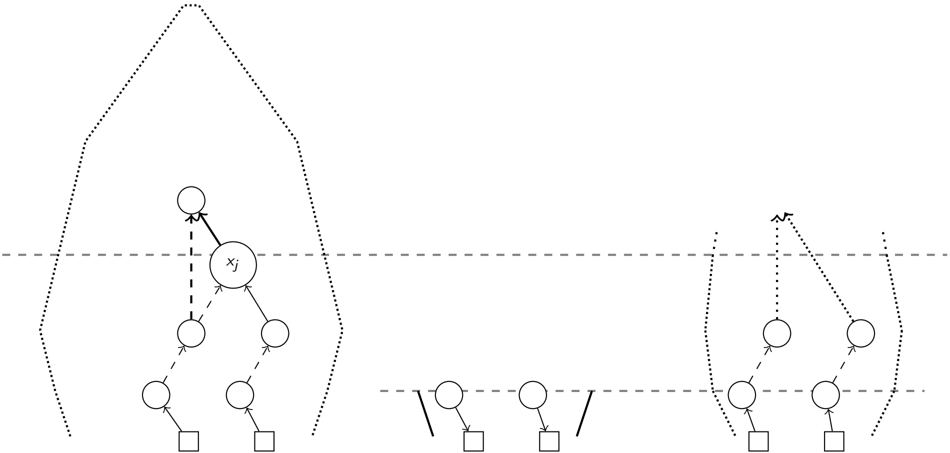
$\exists\vec{x}.\ \phi(\vec{x})$



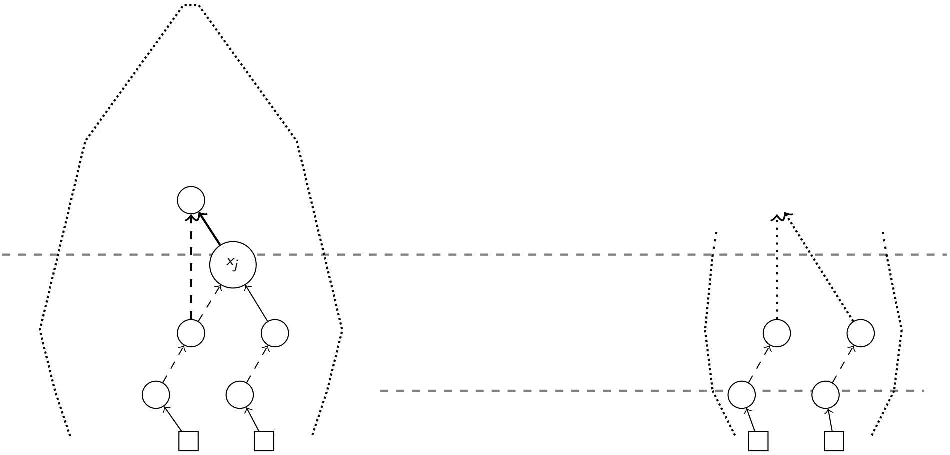
$\exists \vec{x}.\ \phi(\vec{x})$



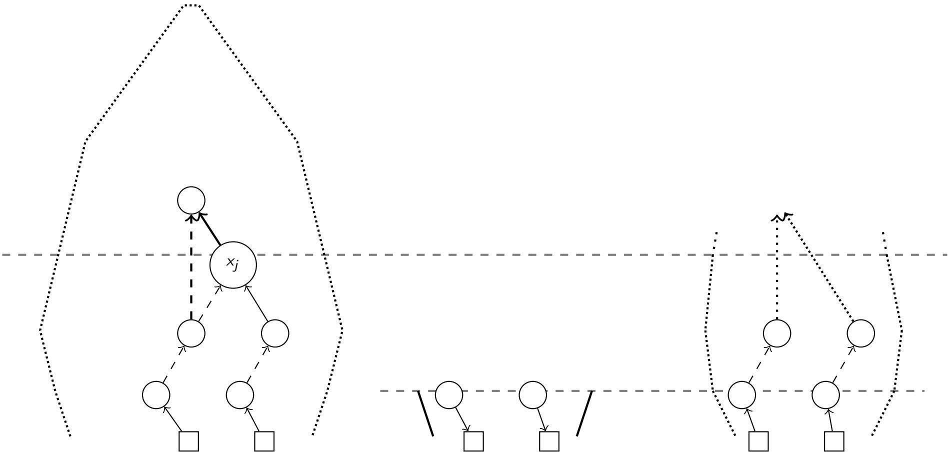
$\exists \vec{x}.\ \phi(\vec{x})$



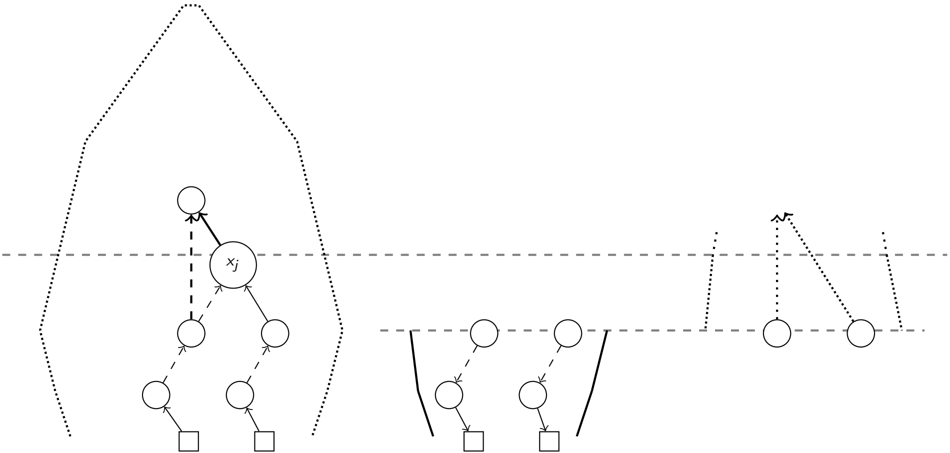
$\exists \vec{x}.\ \phi(\vec{x})$



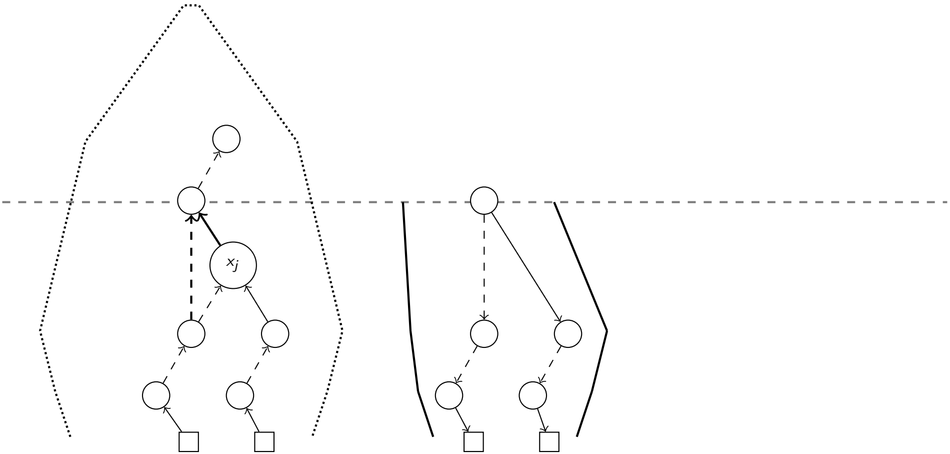
$\exists \vec{x}.\ \phi(\vec{x})$



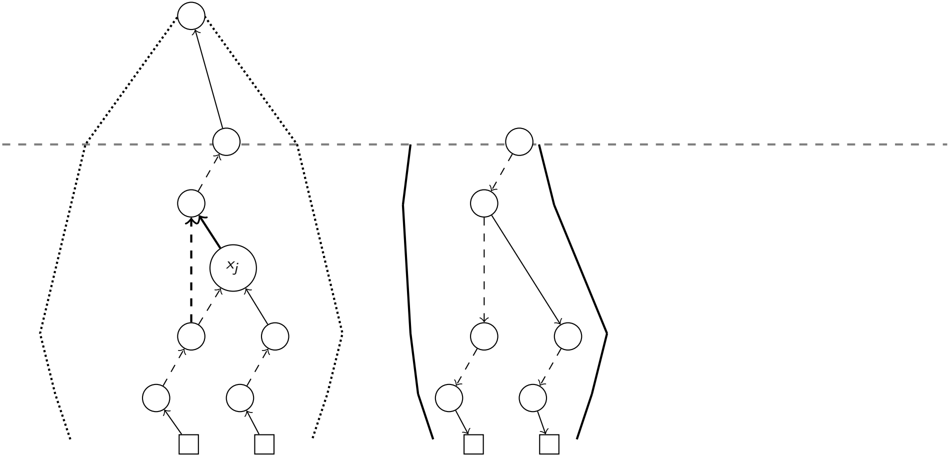
$\exists \vec{x}.\ \phi(\vec{x})$



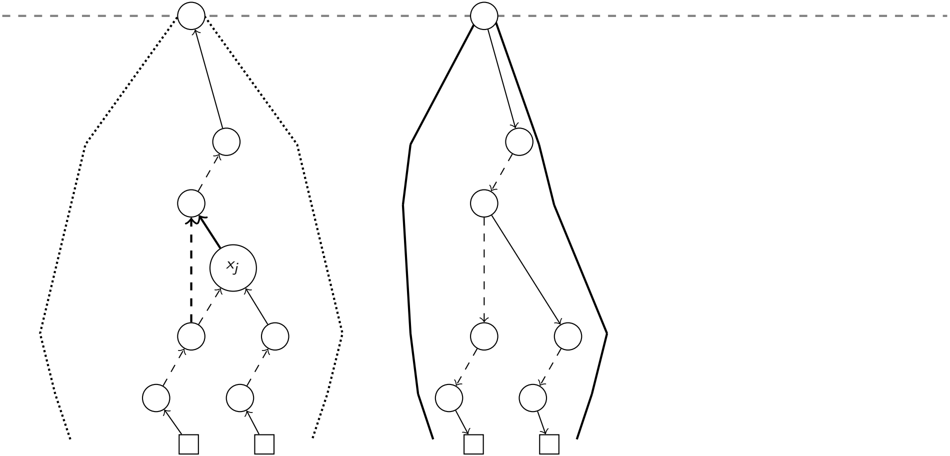
$\exists \vec{x}.\ \phi(\vec{x})$



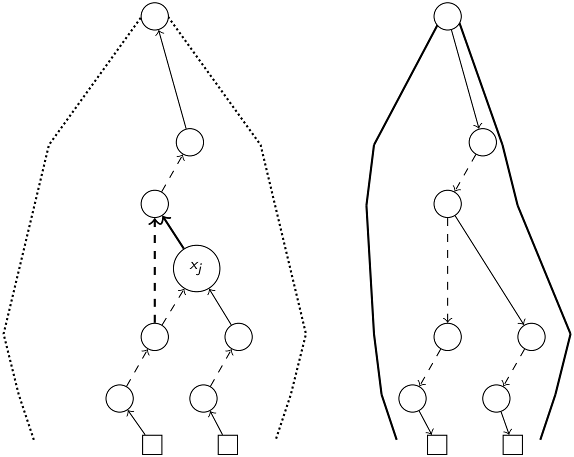
$\exists \vec{x}.\ \phi(\vec{x})$



$\exists \vec{x}.\ \phi(\vec{x})$



$\exists \vec{x}.\ \phi(\vec{x})$



$\exists \vec{x}.\ \phi(\vec{x})$

Theorem (Sølvsten et al. '25)

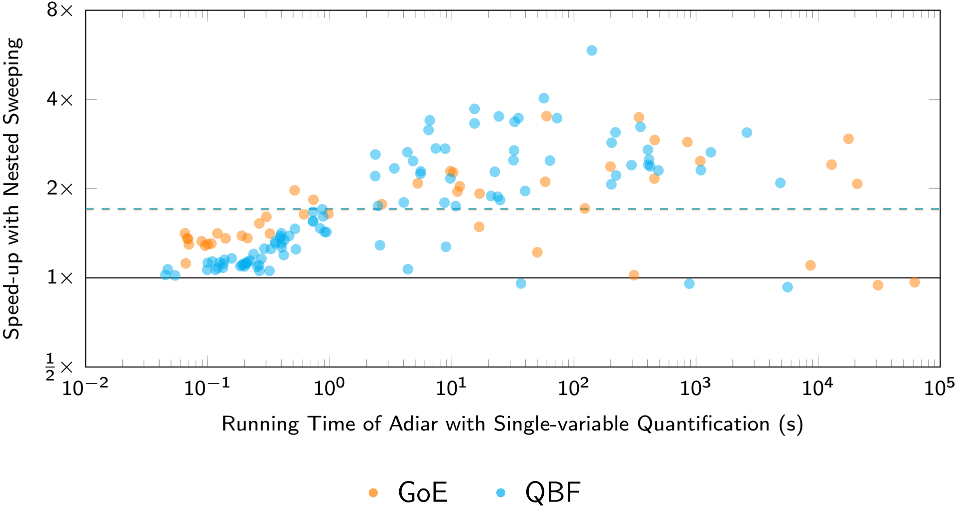
Given BDD ϕ, the quantification of k variables, $\exists \vec{x}.\ \phi(\vec{x})$, is computible in $\mathcal{O}(\text{sort}(|\phi|^{2^k}))$ time and I/Os.

Benchmarks

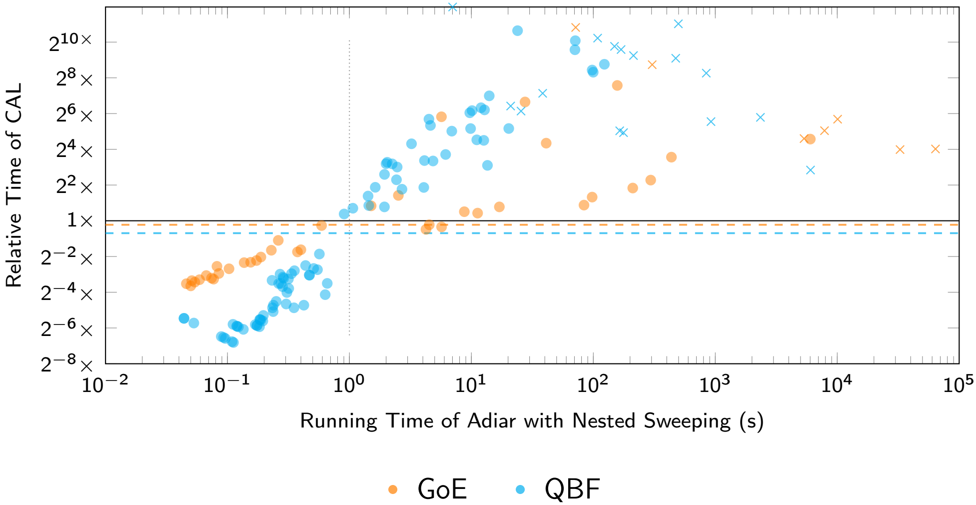
Garden-of-Eden
Given dimensions $N_1, N_2 \in \mathbb{N}$, determine whether there exists in Conway's *Game of Life* an initial state of size $N_1 \times N_2$ that is a *Garden of Eden*, i.e. is otherwise unreachable.

Quantified Boolean Formula
Determine whether a Boolean formula $\exists \vec{x_1} \forall \vec{x_2} \ldots \exists \vec{x_k}.\ \phi(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_k})$ (or any order of quantifiers) evaluates to $\top$ or $\bot$. For inputs, we use the two-player games from:
Irfansha Shaik and Jaco van de Pol: "*Concise QBF encodings for games on a grid (extended version)*". arXiv (2023).

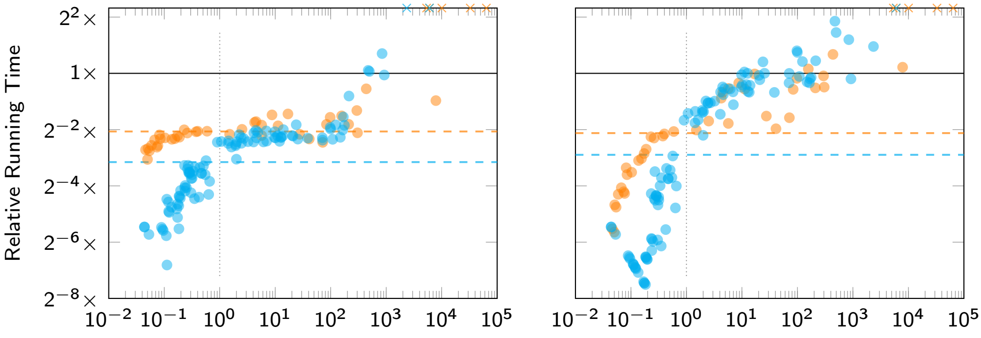
Benchmarks : Single vs. Nested Quantification



Benchmarks : Comparison to CAL



Benchmarks : Comparison to BuDDy and CUDD



(a) BuDDy

(b) CUDD

Steffan Christ Sølvsten

`soelvsten@cs.au.dk`

ssoelvsten.github.io

Adiar

github.com/ssoelvsten/adiar

ssoelvsten.github.io/adiar

AARHUS UNIVERSITY

Nested Sweeping Framework

New BDD algorithms:

- [x] Multi-variable Quantification
- [x] Relational Product
- [ ] Functional Composition
- [ ] Variable Reordering

Other Decision Diagrams:

- [ ] Quantum Multi-valued Decision Diagrams
- [ ] Polynomial Boolean Rings

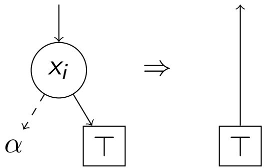
Better Transposition: Deepest Variable Quantification

$$bdd\_exists(f, \max(\vec{x}))$$

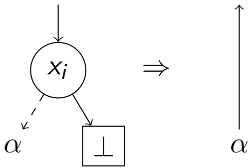
Motivation
Includes the first nested sweep inside of the transposition step.

In Practice
Slows down computation time on average by 4.7%.

Better Transposition: Pruning $\top$ Siblings



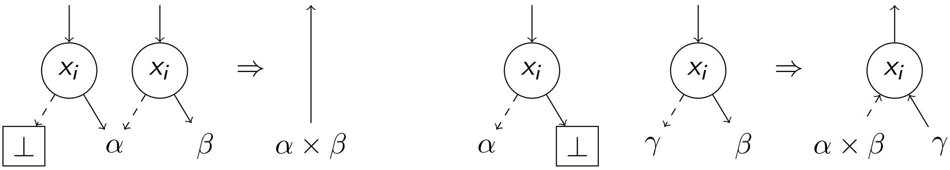
(a) Pruning due to $\top$.



(b) Pruning due to $\perp$.

In Practice
If it prunes subtrees, running time can improve up to 21%. Otherwise, it adds an overhead of up to 2%.

Better Transposition: Partial Quantification



(a) Fully quantified pair of nodes. **(b)** Partially quantified pair of nodes.

In Practice

In some instances, it improves performance by a factor of $\sim 2\times$.

In others, it slows down by $\sim 2\times$.

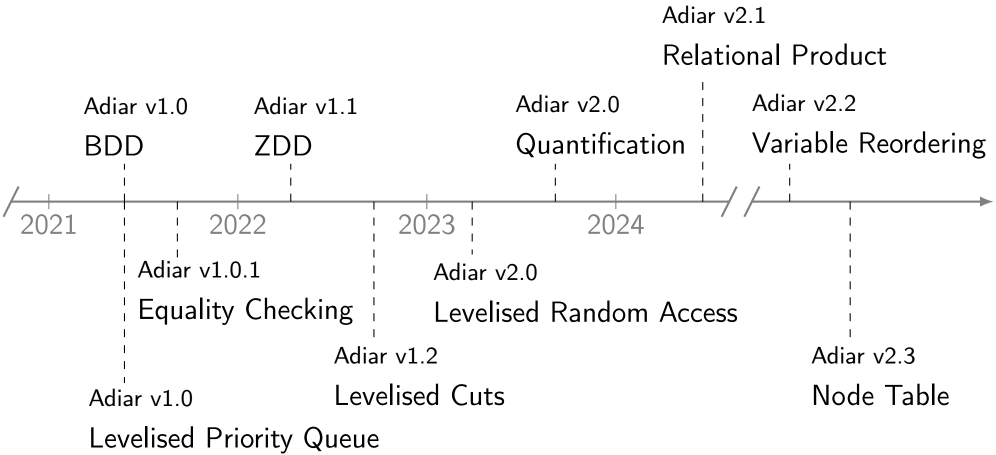
Observation

Instances improved by $\top$ *Pruning* were disjoint from *Partial Quantification*.

Optimisations for Nested Sweeping

- **Leave terminal arcs out of nested sweeps.**
  Required to satisfy invariants in nested algorithms from [TACAS 22].
- **Bail-out of Inner Sweep if all edges are subtree-preserving.**
  In practice, 75.6% of all levels are skippable (median of 81.9% for each benchmark).
- **Use a sorted list as a bridge from the outer to the inner sweep.**
  Postpones initialising data structures for the nested sweep until it is invoked.
  - More memory available for the outer `Reduce` sweep.
  - Sorting once and then merging on-the-fly with a priority queue can be faster than maintaining a larger priority queue.
  - This enables the *levelised priority queues* [TACAS* 22], *levelised cuts* [ATVA 23], and *levelised random access* [SPIN 24] optimisations.

	Single Quantification		Nested Sweeping	
	LOC	# Tests	LOC	# Tests
`nested_sweeping.h`	–	–	1287	104
`quantify`	548	84	1152	152
`core/...`	326	–	904	–
`bdd/...`	122	64	157	108
`zdd/...`	100	20	20	44


Adiar v2.1
Relational Product
Adiar v1.0
BDD
Adiar v1.1
ZDD
Adiar v2.0
Quantification
Adiar v2.2
Variable Reordering
2021
2022
2023
2024
Adiar v1.0.1
Equality Checking
Adiar v2.0
Levelised Random Access
Adiar v1.2
Levelised Cuts
Adiar v2.3
Node Table
Adiar v1.0
Levelised Priority Queue