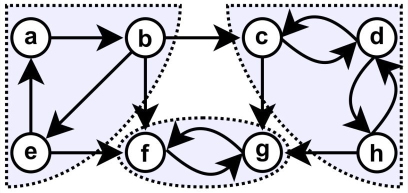
Fast Symbolic Computation of Bottom SCCs

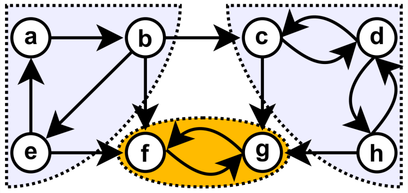
Anna Blume Jakobsen   Rasmus Skibdahl Melanchton Jørgensen
Jaco van de Pol   **Andreas Pavlogiannis**

AARHUS UNIVERSITY

Bottom Strongly Connected Components (BSCCs)



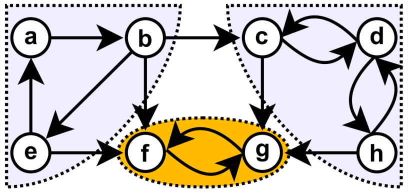
Bottom Strongly Connected Components (BSCCs)



Bottom (or terminal) SCCs

- SCCs that cannot be escaped
- Attractors in dynamical systems
- Define long-run properties

Bottom Strongly Connected Components (BSCCs)



Bottom (or terminal) SCCs

- SCCs that cannot be escaped
- Attractors in dynamical systems
- Define long-run properties

How do we compute BSCCs efficiently?

Symbolic Representations

Applications yield huge graphs

- State-space explosion
- 10^9 nodes and above

Symbolic Representations

Applications yield huge graphs

- State-space explosion
- 10^9 nodes and above

- Inputs are not that large
- Symbolically represented

Symbolic Representations

Applications yield huge graphs

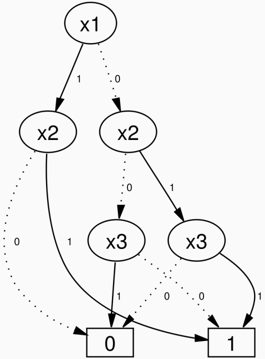
- State-space explosion
- 10^9 nodes and above

- Inputs are not that large
- Symbolically represented

- Algorithms need also be symbolic

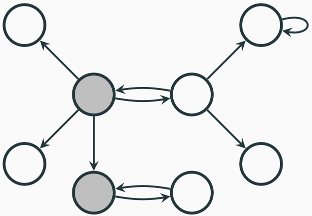
Binary Decision Diagrams

- A graph representation of boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$
- Succinct
- A data structure for symbolic computation



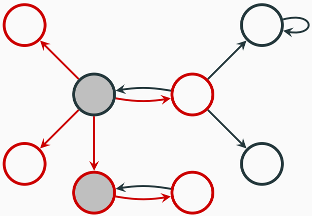
Symbolic Computation

- Symbolic representation gives coarse-grained access to $G = (V, E)$
- V, E represented via BDDs
- Subsets $X \subseteq V$ represented via BDDs



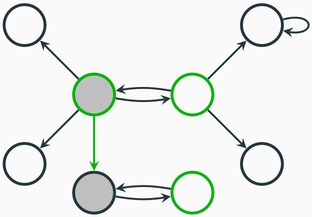
Symbolic Computation

- Symbolic representation gives coarse-grained access to $G = (V, E)$
- V, E represented via BDDs
- Subsets $X \subseteq V$ represented via BDDs
  - *Post*(X)



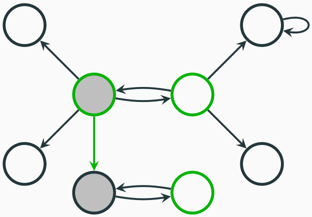
Symbolic Computation

- Symbolic representation gives coarse-grained access to $G = (V, E)$
- V, E represented via BDDs
- Subsets $X \subseteq V$ represented via BDDs
  - $Post(X)$, $Pre(X)$



Symbolic Computation

- Symbolic representation gives coarse-grained access to $G = (V, E)$
- V, E represented via BDDs
- Subsets $X \subseteq V$ represented via BDDs
  - $\mathit{Post}(X)$, $\mathit{Pre}(X)$
- Set operations: $X \cup Y$, $X \cap Y$, $X \setminus Y$



Complexity measure: # of symbolic steps

Symbolic (B)SCC Decomposition

How do we compute (B)SCCs symbolically?

Algorithms for general SCCs

- [TCAD '00] BwdFwd in $O(n^2)$
- [FMSD '06] LockStep in $O(n \cdot \log n)$
- [SODA '03] Skeleton in $O(n)$
- [TACAS '23] Chain in $O(n)$

Symbolic (B)SCC Decomposition

How do we compute (B)SCCs symbolically?

Algorithms for general SCCs

- [TCAD '00] BwdFwd in $O(n^2)$
- [FMSD '06] LockStep in $O(n \cdot \log n)$
- [SODA '03] Skeleton in $O(n)$
- [TACAS '23] Chain in $O(n)$

For BSCCs, a variant of BwdFwd is efficient ($O(n)$ time) and often practical

This Paper

1. Pendant: A new symbolic algorithm for BSCCs

- $O(n)$ symbolic complexity
- Fast(er) in practice

This Paper

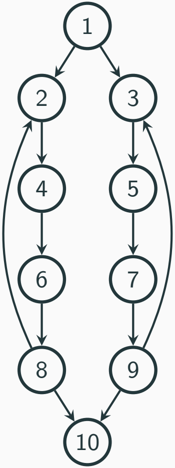
1. Pendant: A new symbolic algorithm for BSCCs

- $O(n)$ symbolic complexity
- Fast(er) in practice

2. Deadlock detection

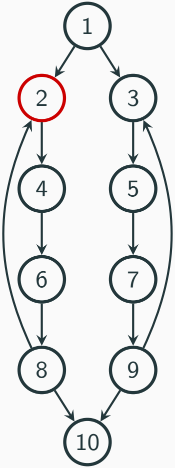
- Deadlocks are special BSCCs
- Essentially for free

Running BwdFwd



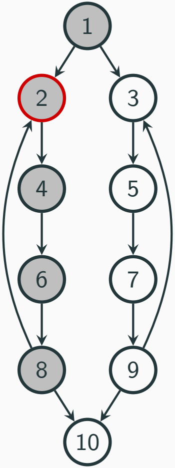
Running BwdFwd

- Pick an arbitrary pivot v



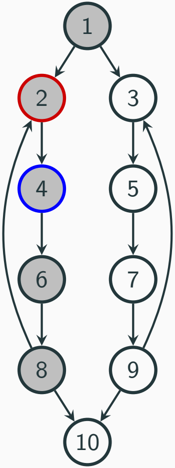
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)



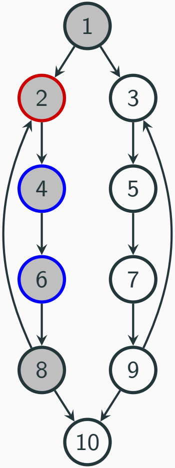
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)



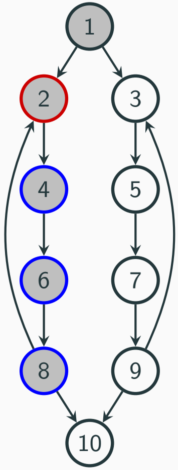
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)



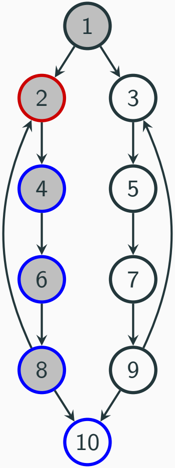
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)



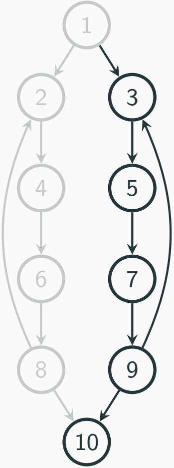
Running BwdFwd

- Pick an arbitrary pivot v
- Compute $\mathrm{Bwd}(v)$
- Compute $\mathrm{Fwd}(v)$
- If $\mathrm{Fwd}(v) \not\subseteq \mathrm{Bwd}(v)$, $\mathrm{SCC}(v)$ is not a BSCC



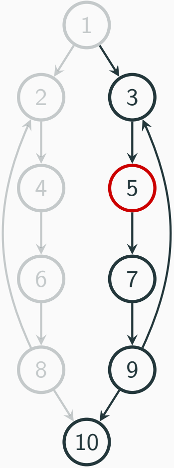
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)
- If Fwd(v) $\not\subseteq$ Bwd(v), SCC(v) is not a BSCC
- Remove Bwd(v) and repeat



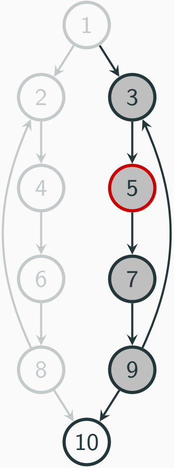
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)
- If Fwd(v) $\not\subseteq$ Bwd(v), SCC(v) is not a BSCC
- Remove Bwd(v) and repeat



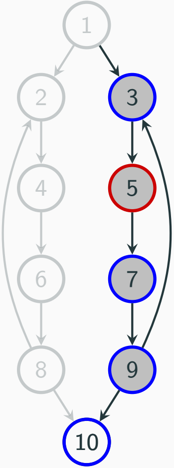
Running BwdFwd

- Pick an arbitrary pivot v
- Compute $\text{Bwd}(v)$
- Compute $\text{Fwd}(v)$
- If $\text{Fwd}(v) \not\subseteq \text{Bwd}(v)$, $\text{SCC}(v)$ is not a BSCC
- Remove $\text{Bwd}(v)$ and repeat



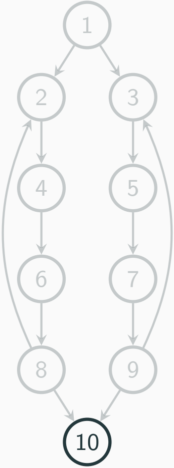
Running BwdFwd

- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)
- If Fwd(v) $\not\subseteq$ Bwd(v), SCC(v) is not a BSCC
- Remove Bwd(v) and repeat

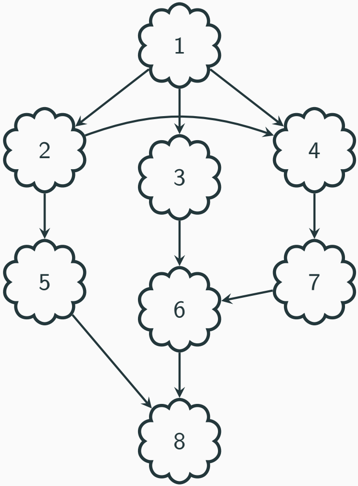


Running BwdFwd

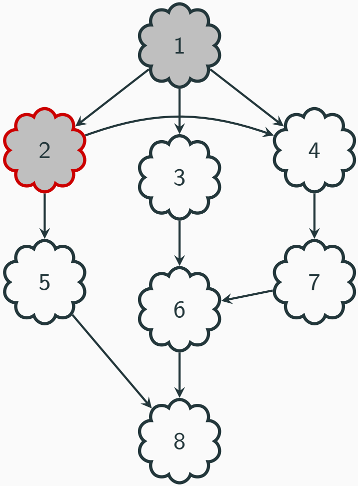
- Pick an arbitrary pivot v
- Compute Bwd(v)
- Compute Fwd(v)
- If Fwd(v) $\not\subseteq$ Bwd(v), SCC(v) is not a BSCC
- Remove Bwd(v) and repeat



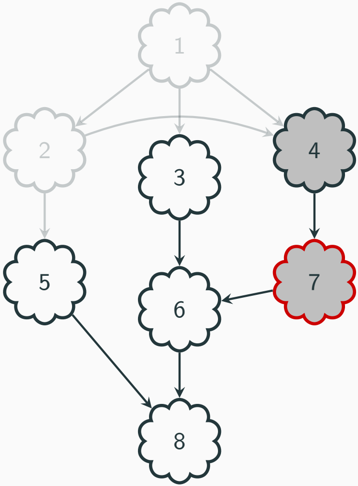
BwdFwd On The Quotient Graph



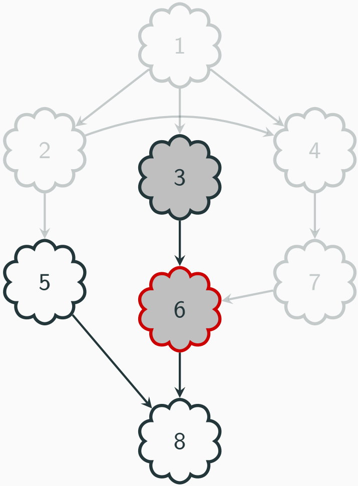
BwdFwd On The Quotient Graph



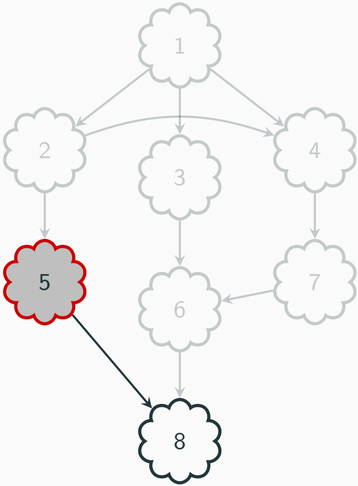
BwdFwd On The Quotient Graph



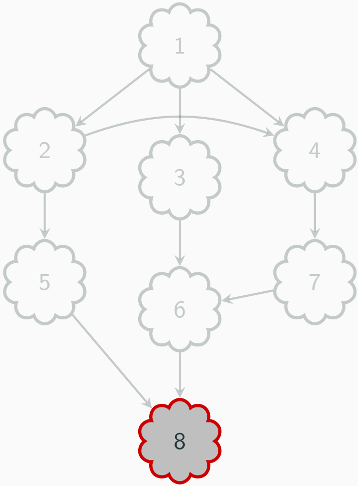
BwdFwd On The Quotient Graph



BwdFwd On The Quotient Graph



BwdFwd On The Quotient Graph



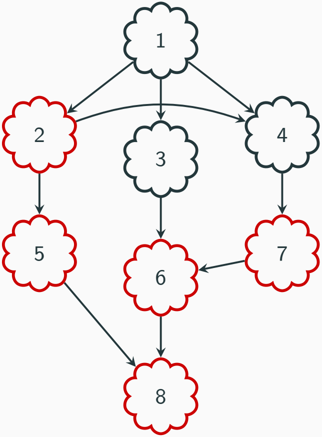
Overhead = 4 SCCs

Motivation

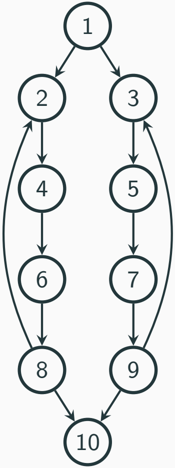
Is there an algorithm that

- Chooses SCCs in a smarter way: towards the bottom of the quotient graph
- Retains the linear-time worst-case complexity
- Less SCC computation overhead in practice

Yes!

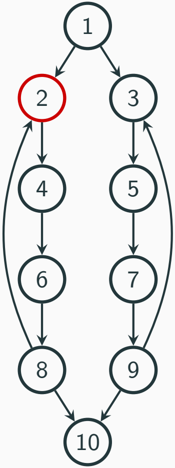


Running Pendant



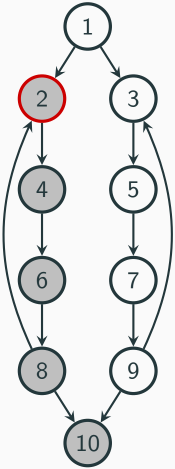
Running Pendant

- Pick an arbitrary pivot v
  - Fwd(v) always contains a BSCC!



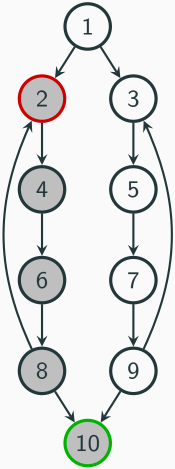
Running Pendant

- Pick an arbitrary pivot v
  - Fwd(v) always contains a BSCC!
- Compute Fwd(v)



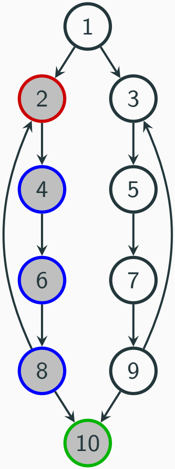
Running Pendant

- Pick an arbitrary pivot v
  - Fwd(v) always contains a BSCC!
- Compute Fwd(v)
- Remember the last layer



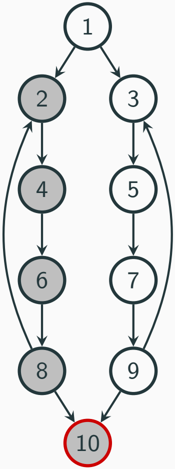
Running Pendant

- Pick an arbitrary pivot v
  - $\mathrm{Fwd}(v)$ always contains a BSCC!
- Compute $\mathrm{Fwd}(v)$
- Remember the last layer
- Compute $\mathrm{SCC}(v) = \mathrm{Bwd}(v) \cap \mathrm{Fwd}(v)$



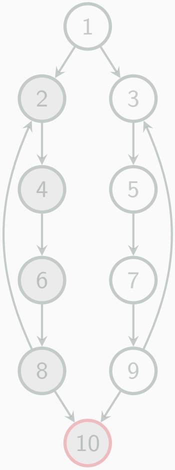
Running Pendant

- Pick an arbitrary pivot v
  - $\mathrm{Fwd}(v)$ always contains a BSCC!
- Compute $\mathrm{Fwd}(v)$
- Remember the last layer
- Compute $\mathrm{SCC}(v) = \mathrm{Bwd}(v) \cap \mathrm{Fwd}(v)$
- Next pivot from the last layer

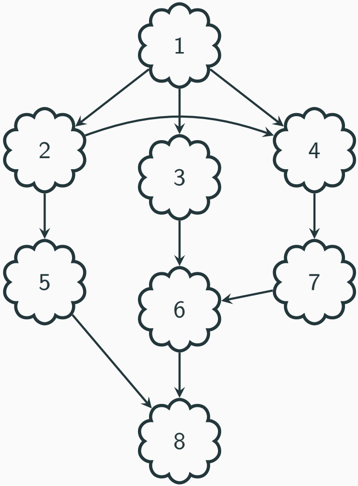


Running Pendant

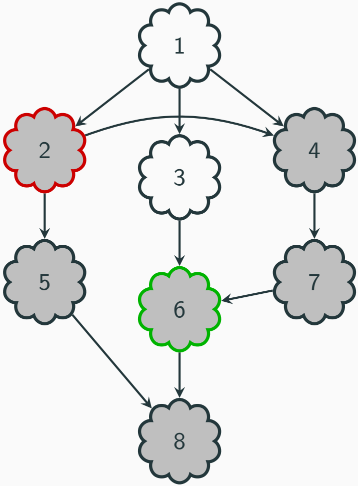
- Pick an arbitrary pivot v
  - $\text{Fwd}(v)$ always contains a BSCC!
- Compute $\text{Fwd}(v)$
- Remember the last layer
- Compute $\text{SCC}(v) = \text{Bwd}(v) \cap \text{Fwd}(v)$
- Next pivot from the last layer



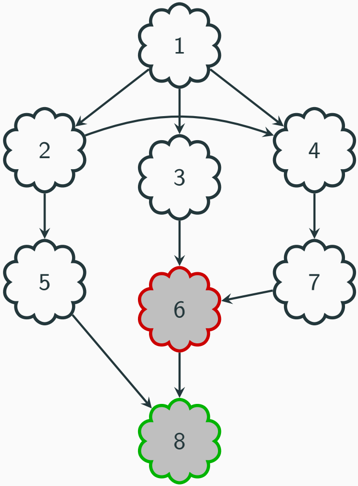
Pendant On The Quotient Graph



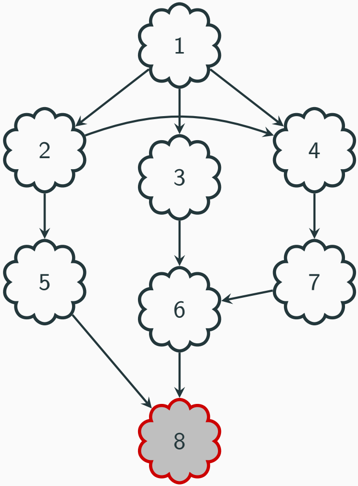
Pendant On The Quotient Graph



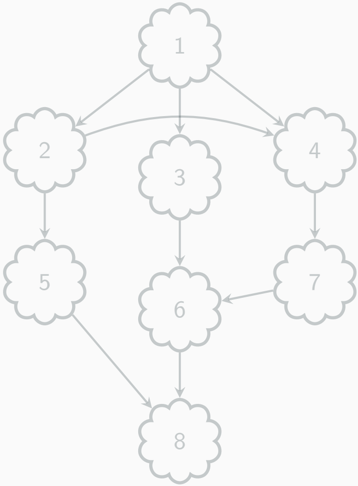
Pendant On The Quotient Graph



Pendant On The Quotient Graph



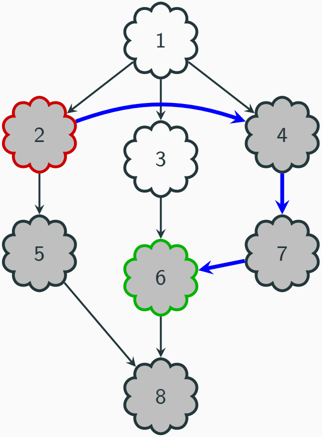
Pendant On The Quotient Graph



Overhead = 2 SCCs

Time Complexity

- Same nodes visited among different recursive steps
- $O(n)$ running time is not obvious

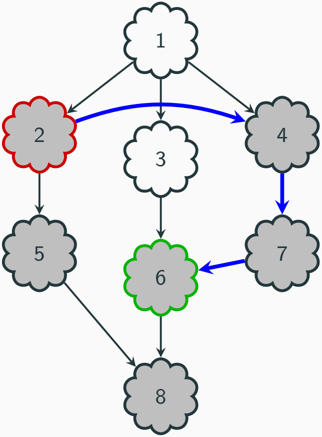


Time Complexity

- Same nodes visited among different recursive steps
- $O(n)$ running time is not obvious

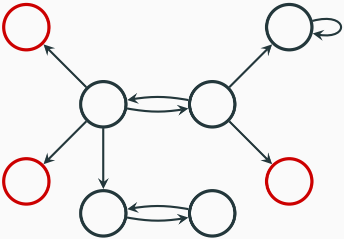
Insights

- Symbolic: the whole cost for $\mathrm{Fwd}(v)$ can be attributed to the longest path
- No node in this path will be touched again



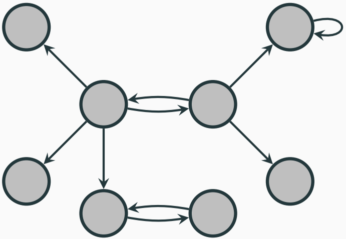
Deadlocks

A **deadlock** is a trivial BSCC



Deadlocks

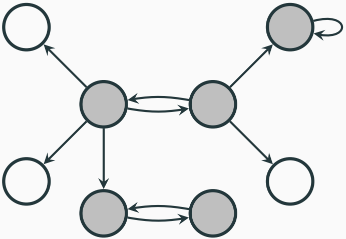
A **deadlock** is a trivial BSCC



Deadlocks

A **deadlock** is a trivial BSCC

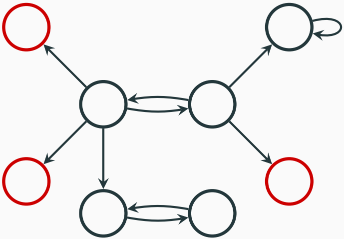
- $H \leftarrow Pre(V, G)$



Deadlocks

A **deadlock** is a trivial BSCC

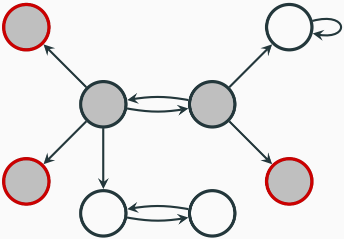
- $H \leftarrow Pre(V, G)$
- $D \leftarrow V \setminus H$



Deadlocks

A **deadlock** is a trivial BSCC

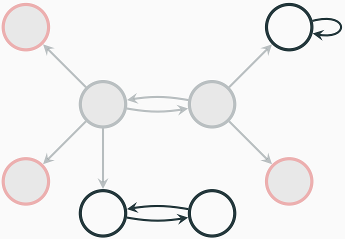
- $H \leftarrow Pre(V, G)$
- $D \leftarrow V \setminus H$
- $B \leftarrow \textsc{Bwd}(D, G)$



Deadlocks

A **deadlock** is a trivial BSCC

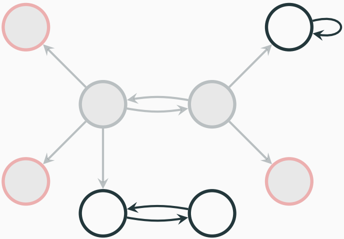
- $H \leftarrow Pre(V, G)$
- $D \leftarrow V \setminus H$
- $B \leftarrow \textsc{Bwd}(D, G)$
- Return $G[V \setminus B]$



Deadlocks

A **deadlock** is a trivial BSCC

- $H \leftarrow Pre(V, G)$
- $D \leftarrow V \setminus H$
- $B \leftarrow \textsc{Bwd}(D, G)$
- Return $G[V \setminus B]$



All deadlocks at the cost of **one** *Pre*

Experimental Setup

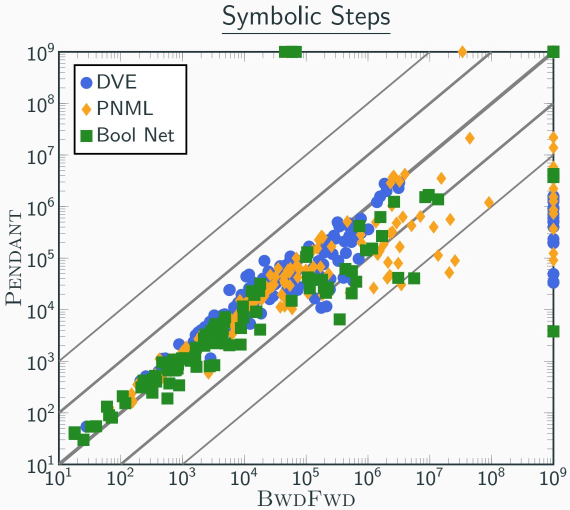
Methods

- PENDANT vs BWDFWD
- Deadlock Detection
- ITGR (Interleaved Transition Guided Reduction)

Benchmarks

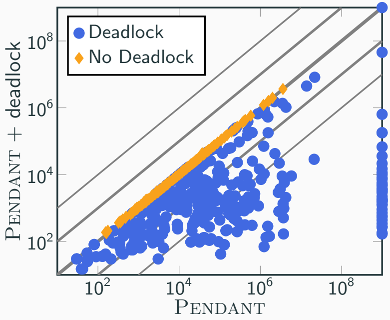
- PetriNet models from the Model Checking Competition
- DiVinE models from the Benchmark of Explicit Models
- Asynchronous Boolean Networks

Pendant vs BwdFwd

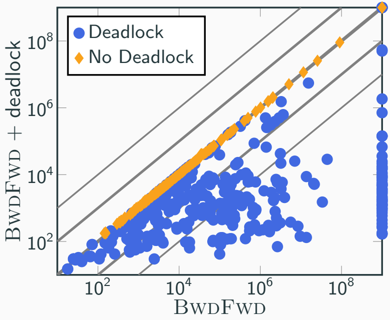


Deadlocks vs No Deadlocks

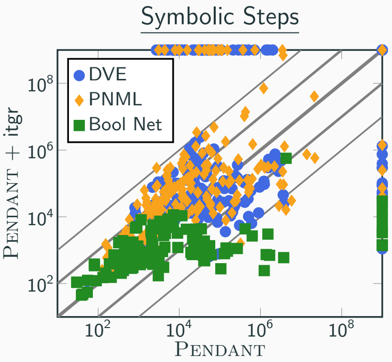
Symbolic Steps

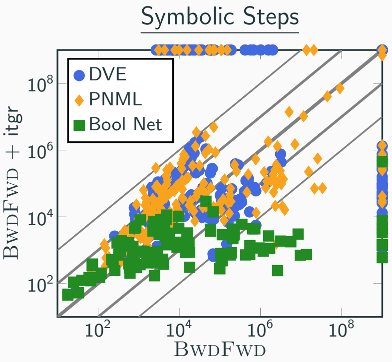


Symbolic Steps

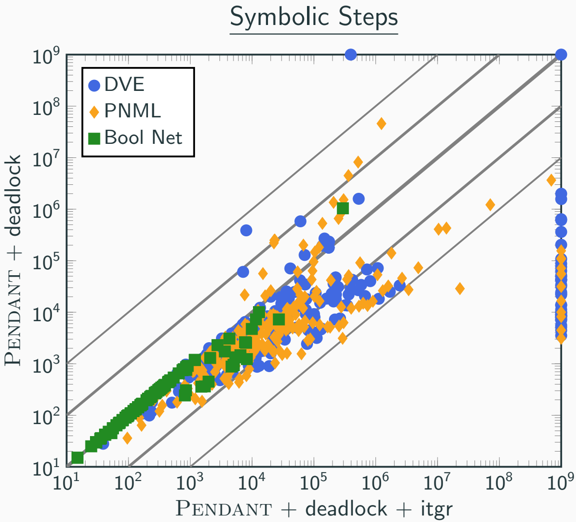


ITGR vs No ITGR

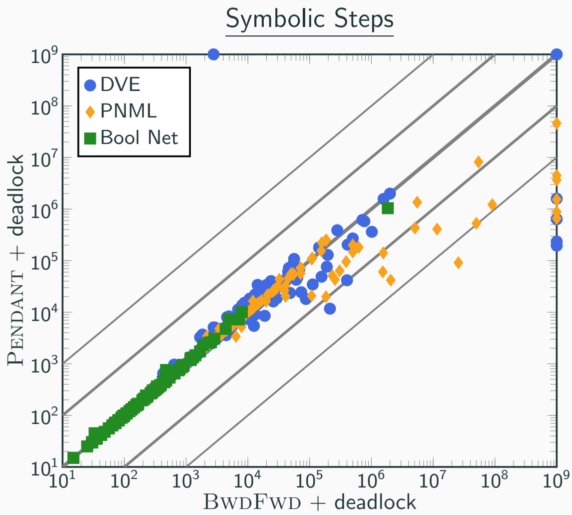




Pendant+Deadlocks vs Pendant+Deadlocks+ITGR



Pendant+Deadlocks vs BwdFwd+Deadlocks



Conclusion

- Pendant: A new, symbolic algorithm for Bottom SCCs
- $O(n)$ (symbolic) time
- Faster in practice
- Deadlock detection speeds up the computation further

Conclusion

- PENDANT: A new, symbolic algorithm for Bottom SCCs
- $O(n)$ (symbolic) time
- Faster in practice
- Deadlock detection speeds up the computation further

Thank you!