The Quest for an Efficient LTL Model-Checking

E. Renault

Friday, May 18th

What is Model-Checking? (Trebuchet Example)

What is Model-Checking? (Trebuchet Example)

Finally Pivot Bar released?

What is Model-Checking? (Trebuchet Example)

Finally Pivot Bar released?

Model-Checking

What is Model-Checking? (Trebuchet Example)

Finally Pivot Bar released?

Model-Checking

Verified

Violated

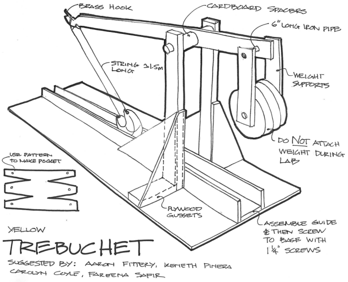
What is Model-Checking? (Trebuchet Example)

Finally Pivot Bar released?

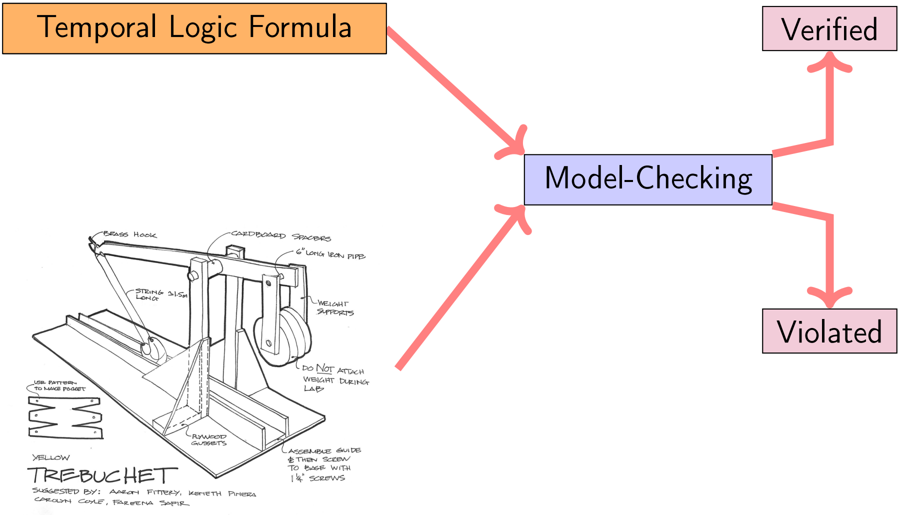
Model-Checking

Verified

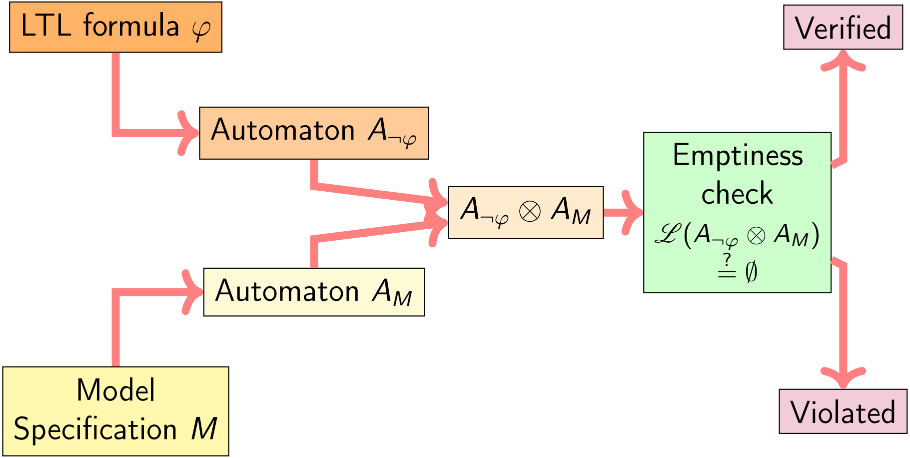
Violated



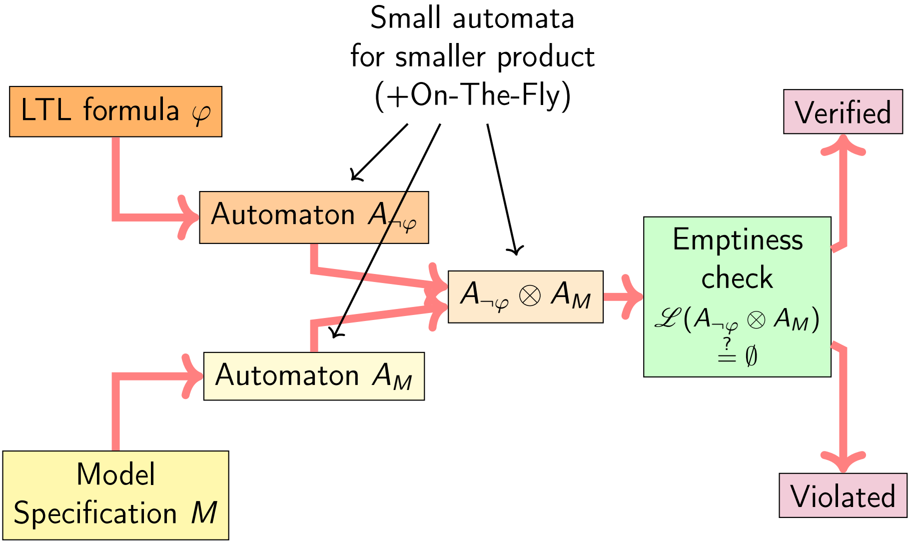
What is Model-Checking? (Trebuchet Example)



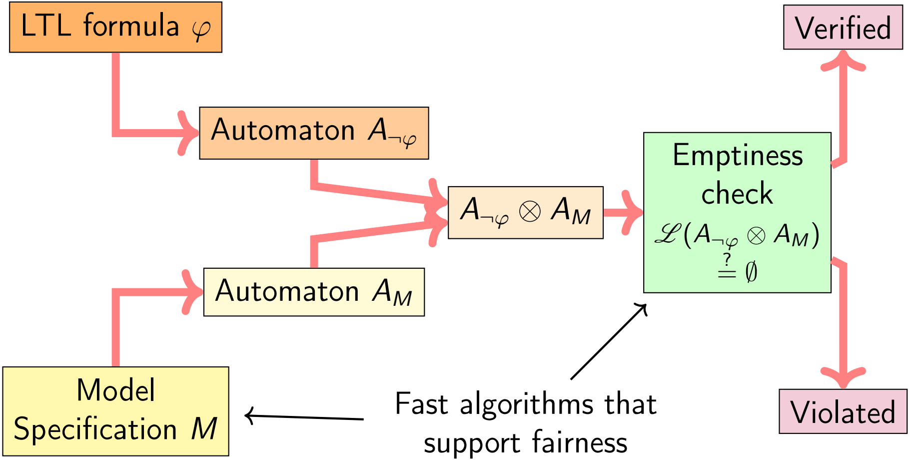
Automata-Theoretic Approach to Model Checking



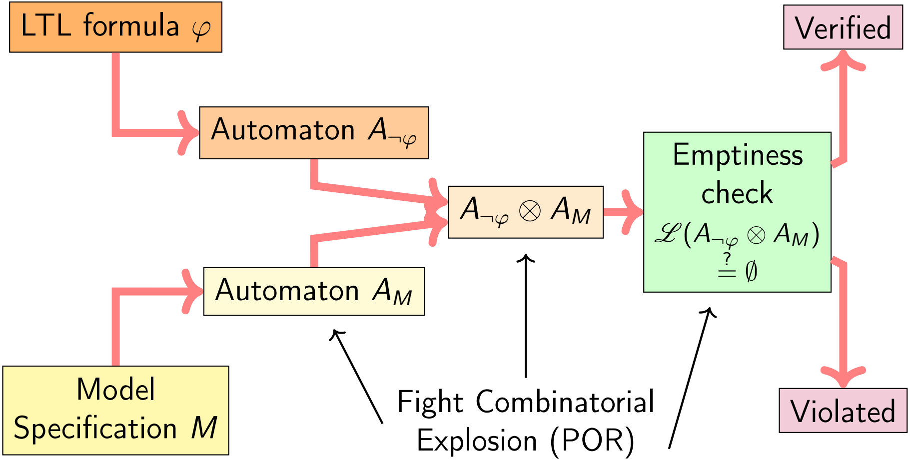
Automata-Theoretic Approach to Model Checking



Automata-Theoretic Approach to Model Checking



Automata-Theoretic Approach to Model Checking




Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
Symbolic techniques Mountains
ω-automata villages
The forest of the Emptiness
Dijkstra Temple
SCC HILLS
Bitstate haching
State space caching
Tarjan Temple
The one hundred bridges of the proviso
On the fly
The dead Forest of the Union-Find
Decomposition Tower
Renault
UFSCC
Map
Stubborn
Lakes of the POR
Conflicting
Timed Automata
The Outpost of the Parellism
bledge
owcty
Persistent
Linear Topology
swarming
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands
1
2
3
4
5

ω-Automata Villages


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
ω-automata villages
1
The forest of the Emptiness
Dijkstra Temple
SCC hills
Tarjan Temple
Symbolic techniques Mountains
Bitstate haching
State space caching
On the fly
The one hundred bridges of the proviso
Decomposition Tower
The dead Forest of the Union-Find
Renault
UFSCC
Map
Lakes of the POR
Stubborn
Conflicting
Persistent
Timed Automata
The outpost of the parellism
bledge
nege
owcty
swarming
Linear Topology
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands

Many automata . . .

- Büchi, Co-Büchi, Streett, Rabin, Parity, Muller, other?
- Generalized or not?
- Transition-based or state-based?
- Support fairness (weak or strong)

The HOA format support all these variations. HOA is supported by many tools: Spot ltl3ba, Rabinizer3, ltl3dra

Many automata . . .

- Büchi, Co-Büchi, Streett, Rabin, Parity, Muller, other?
- Generalized or not?
- Transition-based or state-based?
- Support fairness (weak or strong)

Transition-based Generalized Büchi Automata (TGBA) seems to be a good compromise:

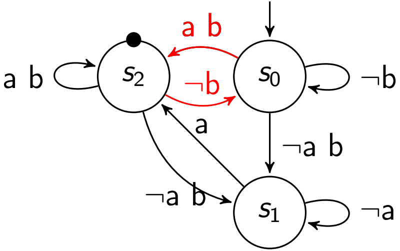
- Support for weak fairness
- Emptiness checks may be linear regardless the acceptance condition

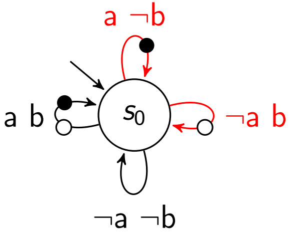
---

The HOA format support all these variations. HOA is supported by many tools: Spot ltl3ba, Rabinizer3, ltl3dra

Fight Combinatorial Explosion

Büchi Automata (BA) $\mathcal{F}$={●}	Transition-based Generalized Büchi Automata (TGBA) $\mathcal{F}$={●, ○}





Infinite runs are accepting if they visit each acceptance set infinitely often. If there is such a run: $\mathscr{L}(A) \neq \emptyset$.

Two equivalent and minimal automata for the LTL formula GF a ∧ GF b

Support Fairness

Weak fairness can be expressed using the LTL property:

$$\bigwedge_{i \in Processes} \text{GF progress}_i$$

	Min. det. BA		Min. det. TGBA	
Nb. Processes	states	transitions	states	transitions
1	2	4	1	2
2	3	12	1	4
4	5	80	1	16
8	9	2 304	1	256
n	$(n+1)$	$(n+1).2^n$	1	2^n

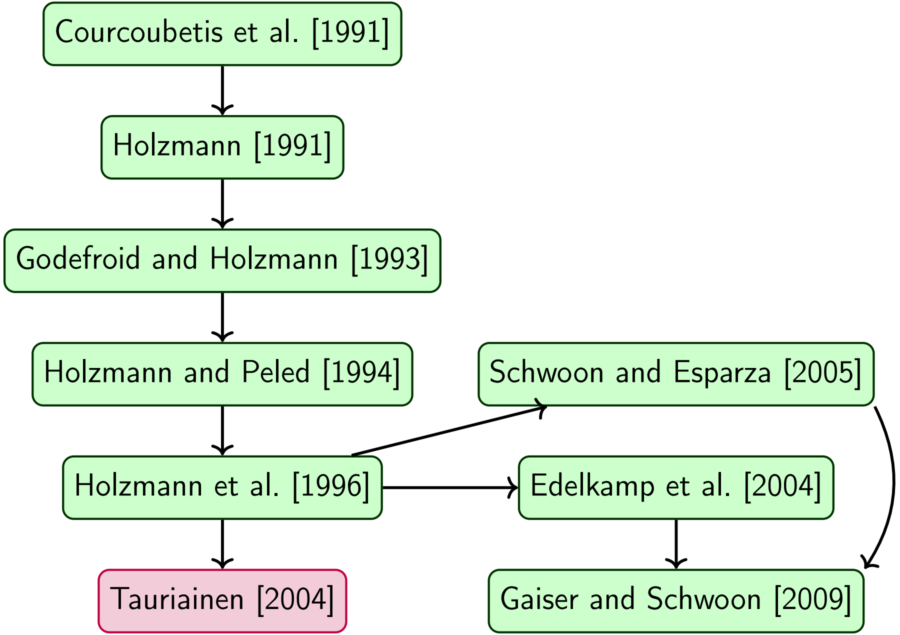
TGBA are never worst than BA!

The Forest of the Emptiness and the SCC Hills


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
Dijkstra Temple
SCC hills
Tarjan Temple
Symbolic techniques Mountains
ω-automata villages
The forest of the Emptiness
Bitstate haching
State space caching
On the fly
The one hundred bridges of the proviso
Decomposition Tower
The dead Forest of the Union-Find
Renault
UFSCC
Map
Lakes of the POR
Stubborn
Conflicting
Persistent
Timed Automata
The outpost of the parellism
bledge
owcty
ncgc
swarming
Linear Topology
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands
1
2

Sequential Emptiness Checks

- **NDFS-based**: look for accepting runs of the automaton using a second interleaved DFS


Courcoubetis et al. [1991]
Holzmann [1991]
Godefroid and Holzmann [1993]
Holzmann and Peled [1994]
Schwoon and Esparza [2005]
Holzmann et al. [1996]
Edelkamp et al. [2004]
Tauriainen [2004]
Gaiser and Schwoon [2009]

Sequential Emptiness Checks

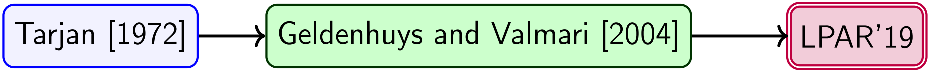
- **NDFS-based**: look for accepting runs of the automaton using a second interleaved DFS

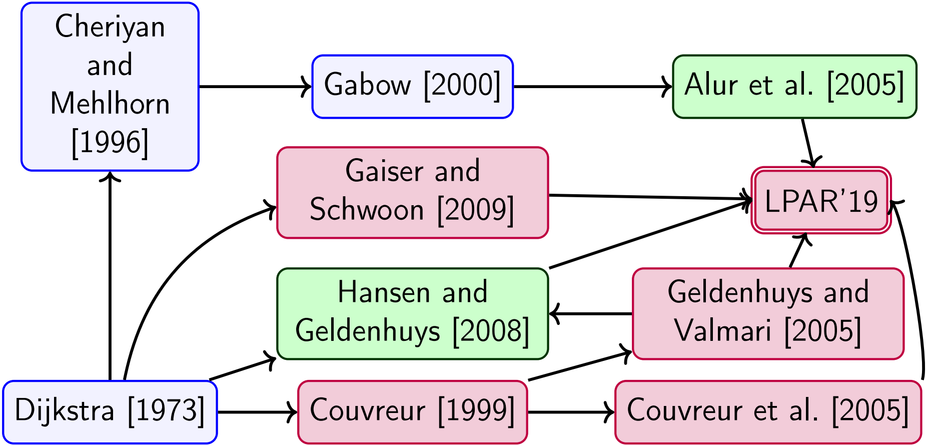
	NDFS-based	
Memory requirements	2 extra bits per state	
Closing edge detect.	easy only on DFS stack	
On-the-fly	✓	
Bit state hashing	✓	
State space caching	✓	
Generalization	Proportionnal to $\mid \mathcal{F} \mid$	

Sequential Emptiness Checks

- **NDFS-based**: look for accepting runs of the automaton using a second interleaved DFS
- **SCC-based**: compute SCC of the automaton and look for accepting SCC using only one DFS

	NDFS-based	
Memory requirements	2 extra bits per state	
Closing edge detect.	easy only on DFS stack	
On-the-fly	✓	
Bit state hashing	✓	
State space caching	✓	
Generalization	Proportionnal to $\mid \mathcal{F} \mid$	

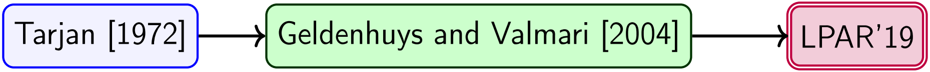

Tarjan [1972]
Geldenhuys and Valmari [2004]
LPAR'19

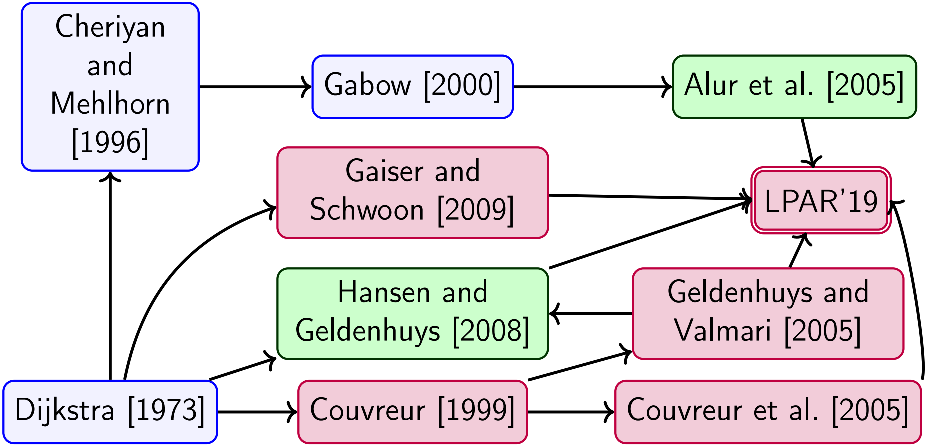

Cheriyan and Mehlhorn [1996]
Gabow [2000]
Alur et al. [2005]
Gaiser and Schwoon [2009]
LPAR'19
Hansen and Geldenhuys [2008]
Geldenhuys and Valmari [2005]
Dijkstra [1973]
Couvreur [1999]
Couvreur et al. [2005]

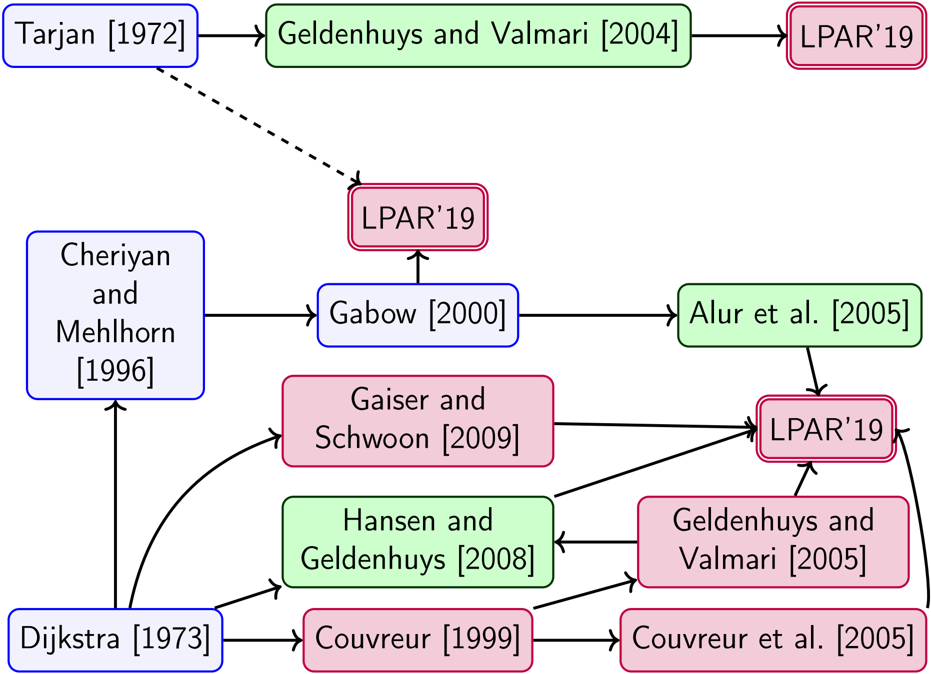
Sequential Emptiness Checks

- **NDFS-based**: look for accepting runs of the automaton using a second interleaved DFS
- **SCC-based**: compute SCC of the automaton and look for accepting SCC using only one DFS

	NDFS-based	SCC-based
Memory requirements	2 extra bits per state	1 or 2 int per state
Closing edge detect.	easy only on DFS stack	easy
On-the-fly	✓	✓
Bit state hashing	✓	✓
State space caching	✓	✓
Generalization	Proportionnal to $\mid \mathcal{F} \mid$	Independant to $\mid \mathcal{F} \mid$


Tarjan [1972]
Geldenhuys and Valmari [2004]
LPAR'19

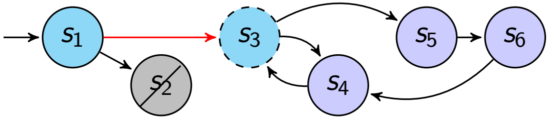

Cheriyan and Mehlhorn [1996]
Gabow [2000]
Alur et al. [2005]
Gaiser and Schwoon [2009]
LPAR'19
Hansen and Geldenhuys [2008]
Geldenhuys and Valmari [2005]
Dijkstra [1973]
Couvreur [1999]
Couvreur et al. [2005]

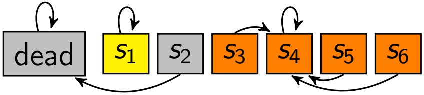

Tarjan [1972]
Geldenhuys and Valmari [2004]
LPAR'19
LPAR'19
Cheriyan and Mehlhorn [1996]
Gabow [2000]
Alur et al. [2005]
Gaiser and Schwoon [2009]
LPAR'19
Hansen and Geldenhuys [2008]
Geldenhuys and Valmari [2005]
Dijkstra [1973]
Couvreur [1999]
Couvreur et al. [2005]

Using Union-Find for Emptiness Check

Main Idea

- Store state's SCC-membership in a Union-Find
- Marking an SCC of size S as *Dead* in $O(Ack^{-1}(S))$ (quasi-constant) rather that in $O(S)$
- Independant from the underlying algorithm (Tarjan/Dijkstra)

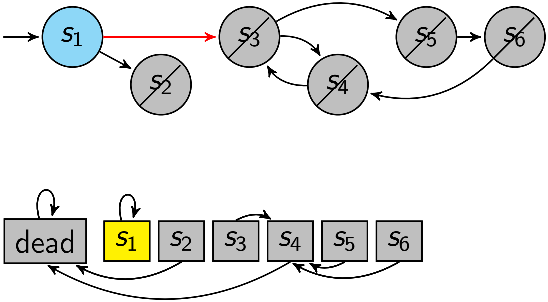




Using Union-Find for Emptiness Check

Main Idea

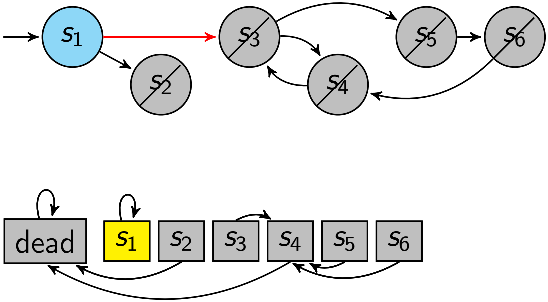
- Store state's SCC-membership in a Union-Find
- Marking an SCC of size S as *Dead* in $O(Ack^{-1}(S))$ (quasi-constant) rather that in $O(S)$
- Independant from the underlying algorithm (Tarjan/Dijkstra)



Using Union-Find for Emptiness Check

Main Idea

- Store state's SCC-membership in a Union-Find
- Marking an SCC of size S as *Dead* in $O(Ack^{-1}(S))$ (quasi-constant) rather that in $O(S)$
- Independant from the underlying algorithm (Tarjan/Dijkstra)
- Easy to parallelize (later on this talk!)



The outpost of the parallelism

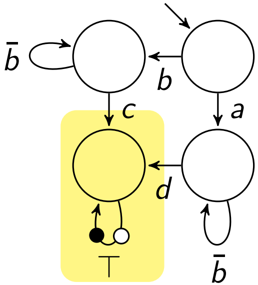
The Decomposition Tower


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
ω-automata villages
The forest of the Emptiness
Dijkstra Temple
SCC hills
Symbolic techniques Mountains
Bitstate haching
State space caching
Tarjan Temple
On the fly
The one hundred bridges of the proviso
The dead Forest of the Union-Find
Decomposition Tower
Renault
UFSCC
Map
Stubborn
Lakes of the POR
Conflicting
bledge
Timed Automata
The outpost of the parellism
Persistent
nege
owcty
Linear Topology
swarming
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands
1
2
3

Strength of $A_{\neg\varphi}$ & Emptiness Check of $A_{\neg\varphi} \otimes A_{Sys}$ [Bloem al., 1999]

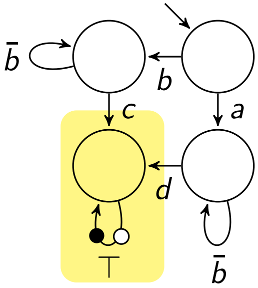
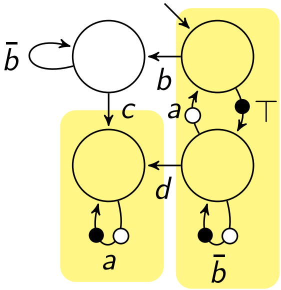
Terminal Automaton	Weak Automaton	Strong Automaton
Accepting SCC are complete and contain only accepting cycles	*Accepting SCC contain only accepting cycles*	*Accepting SCC can mix accepting cycles and non accepting cycles*
Reachability Assumption on A_{Sys}: no deadlock.	Simple cycle search	NDFS-based or SCC-based

Strength of $A_{\neg\varphi}$ & Emptiness Check of $A_{\neg\varphi} \otimes A_{Sys}$ [Bloem al., 1999]

Terminal Automaton	Weak Automaton	Strong Automaton
	*Accepting SCC contain only accepting cycles*	*Accepting SCC can mix accepting cycles and non accepting cycles*
Reachability Assumption on A_{Sys}: no deadlock.	Simple cycle search	NDFS-based or SCC-based

Strength of $A_{\neg\varphi}$ & Emptiness Check of $A_{\neg\varphi} \otimes A_{Sys}$

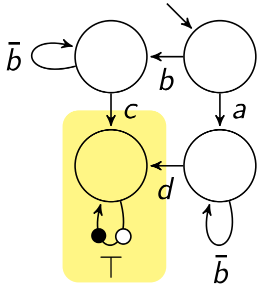
[Bloem al., 1999]

Terminal Automaton	Weak Automaton	Strong Automaton
		*Accepting SCC can mix accepting cycles and non accepting cycles*
Reachability Assumption on A_{Sys}: no deadlock.	Simple cycle search	NDFS-based or SCC-based

Strength of $A_{\neg\varphi}$ & Emptiness Check of $A_{\neg\varphi} \otimes A_{Sys}$

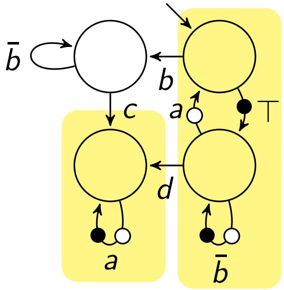
[Bloem al., 1999]

Terminal Automaton



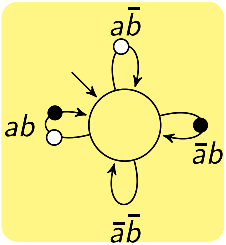
Reachability
Assumption on A_{Sys}: no deadlock.

Weak Automaton



Simple cycle search

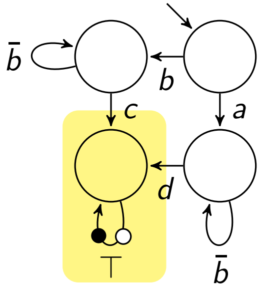
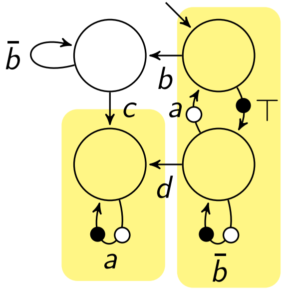
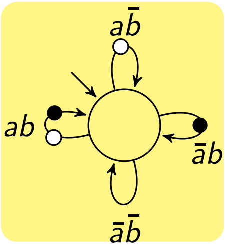
Strong Automaton



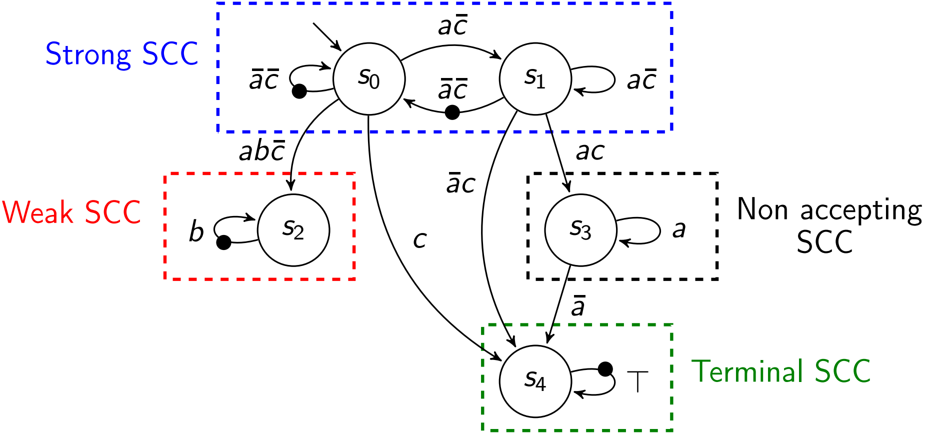
NDFS-based or SCC-based

Strength of $A_{\neg\varphi}$ & Emptiness Check of $A_{\neg\varphi} \otimes A_{Sys}$

[Bloem al., 1999]

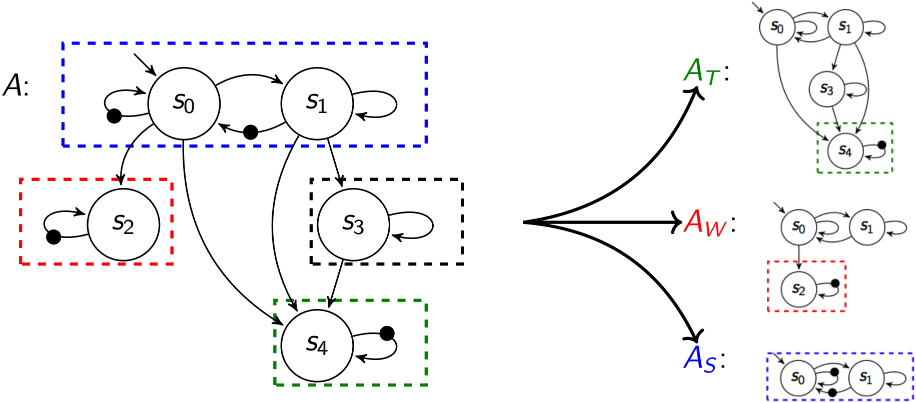
Terminal Automaton	$\subset$	Weak Automaton	$\subset$	Strong Automaton
				
Reachability Assumption on A_{Sys}: no deadlock.	$<$	Simple cycle search	$<$	NDFS-based or SCC-based

Strong Automaton with Multiple SCC Strengths
[Edelkamp et al., 2004]



$A_{\neg\varphi}$ for $\neg\varphi = (\mathsf{G}\, a \rightarrow \mathsf{G}\, b)\, \mathsf{W}\, c$

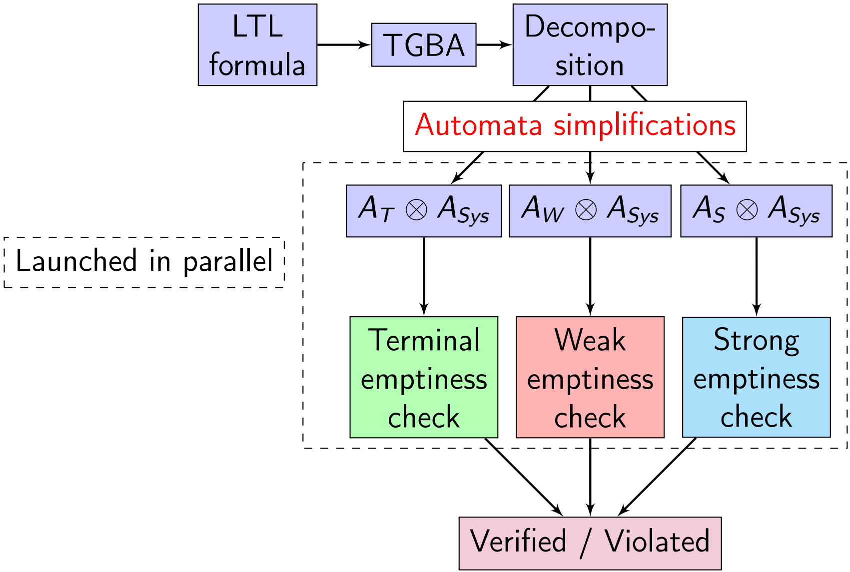
Decomposing the Property Automaton



$$\mathscr{L}(A) = \mathscr{L}(A_T) \cup \mathscr{L}(A_W) \cup \mathscr{L}(A_S).$$

- A_T: captures the terminal behaviors of A
- A_W: captures the weak behaviors of A
- A_S: captures the strong behaviors of A

Decomposition Canevas



Note: emptiness-check agnostic.

Results

On 10 models from BEEM and 3 268 random formula

	No simpl.			With simpl.		
	A_T	A_W	A_S	A_T	A_W	A_S
States Reduction (%)	20	27	54	47	40	60
Transitions Reduction (%)	25	35	67	50	42	67

After simplifications

- Reduction of 86% of states for $A_{sys} \otimes A_T$
- Reduction of 39% of states for $A_{sys} \otimes A_W$
- Reduction of 42% of states for $A_{sys} \otimes A_S$

Average Speedup

- 15% for empty products,
- 70% for non-empty products.

The outpost of the parallelism

The Dead forest of the Union-Find (UFSCC & CNDFS)


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
Symbolic techniques
Mountains
2
Dijkstra Temple
SCC hills
ω-automata
villages
The forest of the Emptiness
Bitstate haching
1
State space caching
Tarjan Temple
On the fly
The one hundred bridges of the proviso
3
The dead Forest of the
Union-Find
Decomposition Tower
Renault
UFSCC
Map
Stubborn
Lakes of the POR
Conflicting
bledge
Timed Automata
The outpost of the parellism
owcty
negc
Persistent
4
swarming
Linear Topology
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands

Problem Statement

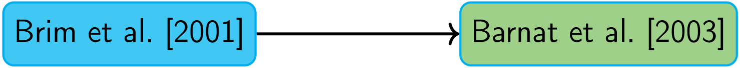
Reif [1985]

Depth-First Search is Inherently Sequential

Brim et al. [2001]

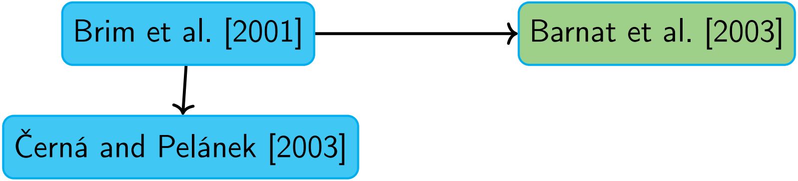
Brim et al. [2001]

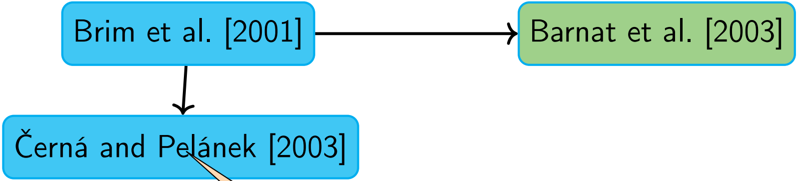
- Detects *negative cycles*
- Transitions are tagged 0 except the one from an accepting state (tagged -1
- Maintains shortest distance from the initial state
- If negative distance, a counterexample is reported


Brim et al. [2001]
Barnat et al. [2003]

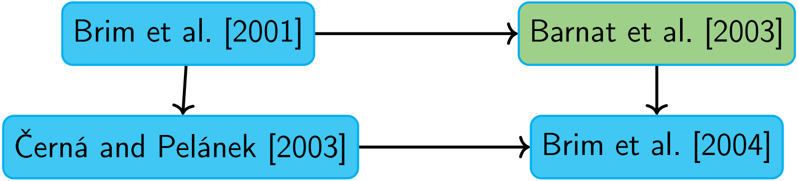
Brim et al. [2001] → Barnat et al. [2003]

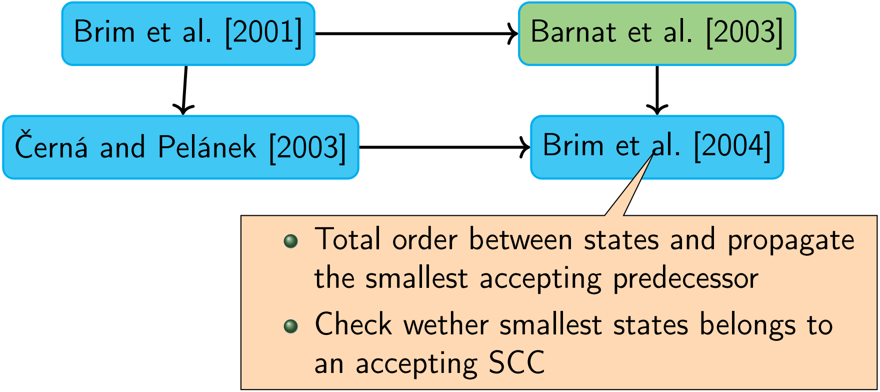
- Track BFS detph of each state
- When a transition goes to an highest state: launch a sequential DFS

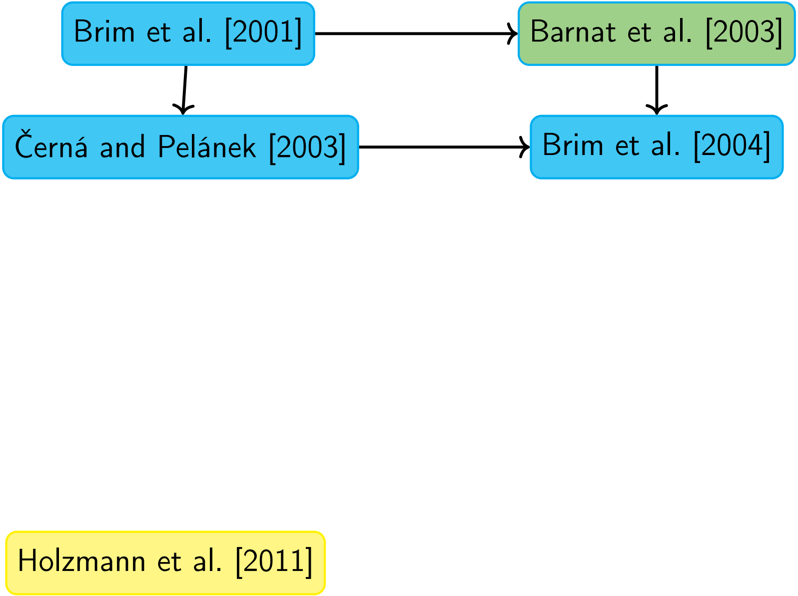

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]

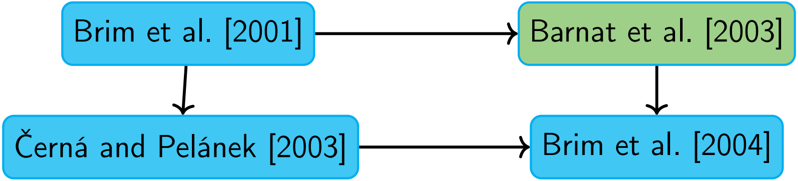


- Explicit OWCTY
- Compute SCCs with accepting states
- If such an SCC, a counterexample exists


Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]

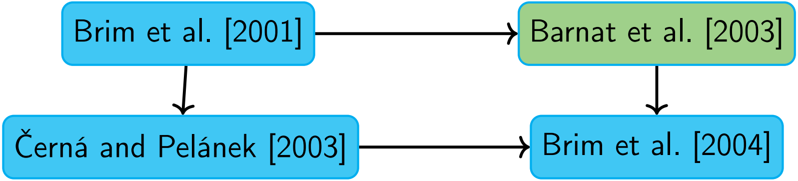

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Total order between states and propagate the smallest accepting predecessor
Check wether smallest states belongs to an accepting SCC

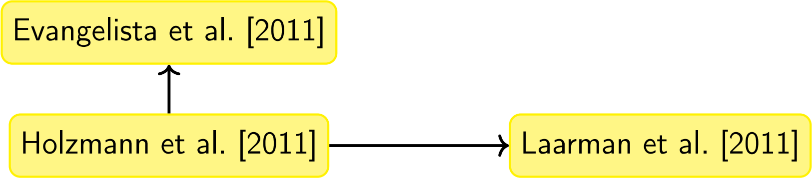

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Holzmann et al. [2011]

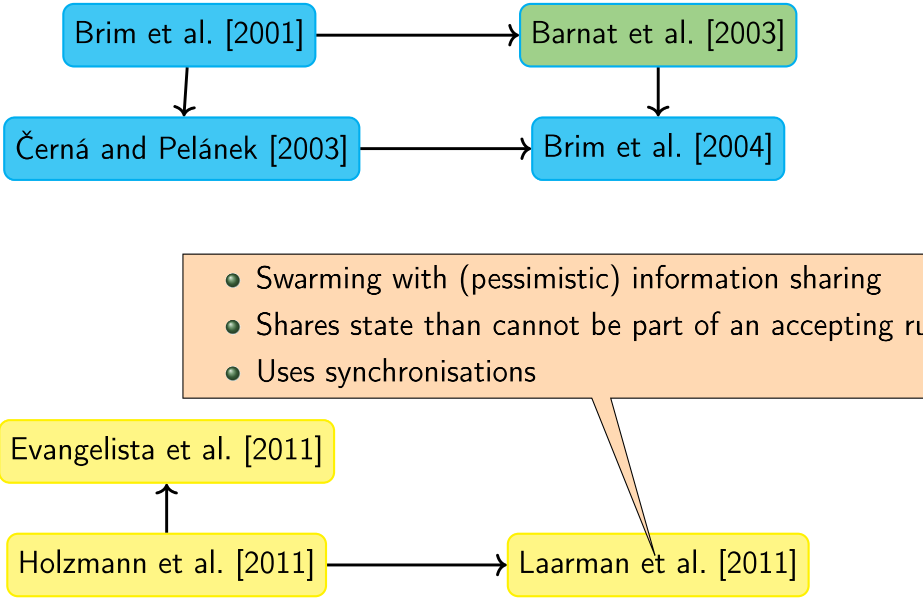


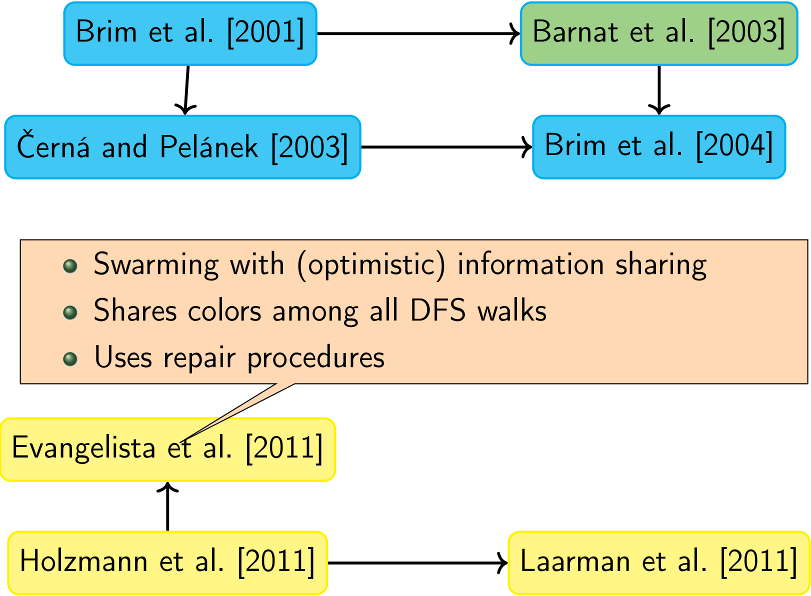
- Run multiple independant emptiness check in parallel
- Each thead has it own transition order

Holzmann et al. [2011]

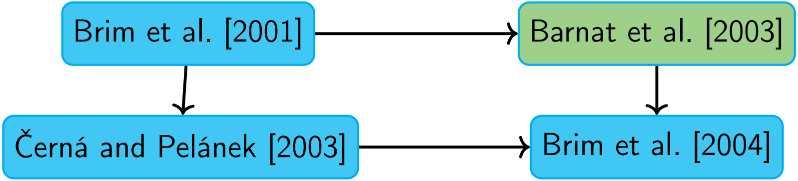

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]

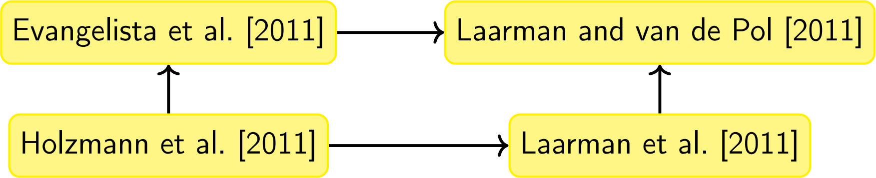

Evangelista et al. [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

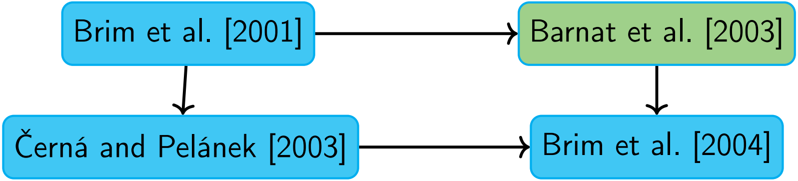

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Swarming with (pessimistic) information sharing
Shares state than cannot be part of an accepting ru
Uses synchronisations
Evangelista et al. [2011]
Holzmann et al. [2011]
Laarman et al. [2011]



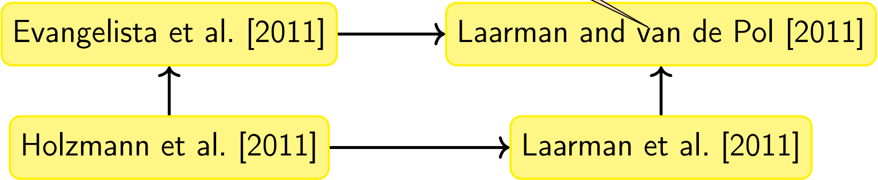
- Swarming with (optimistic) information sharing
- Shares colors among all DFS walks
- Uses repair procedures

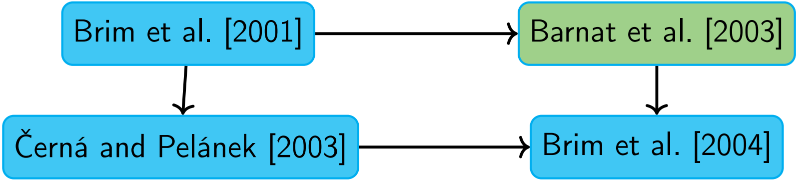

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]

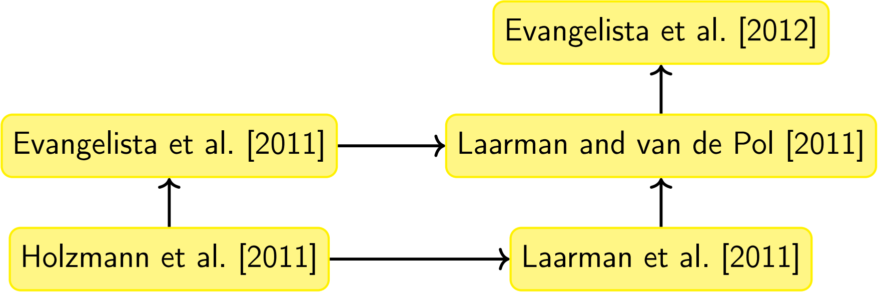

Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

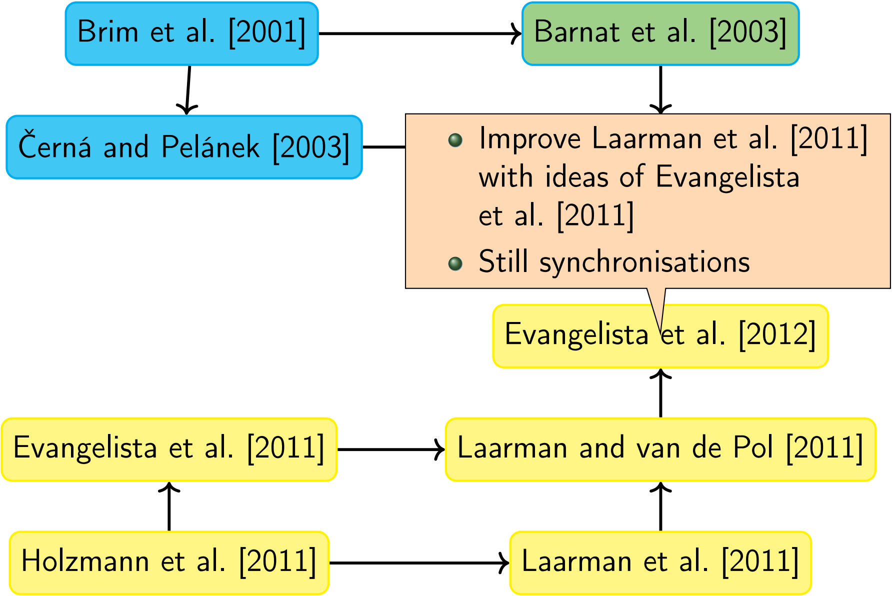


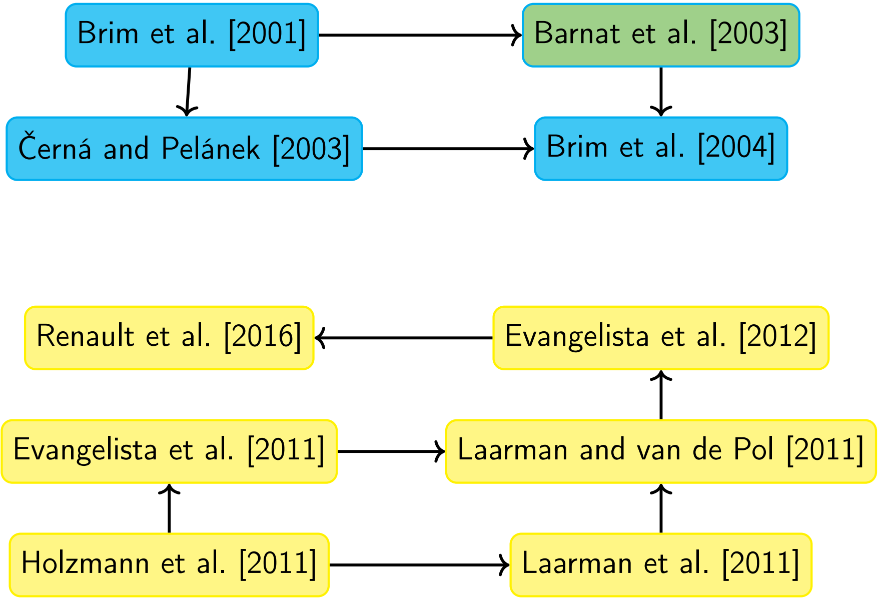
- Mix the 2 previous algorithms: Laarman et al. [2011] is used as a repair procedure



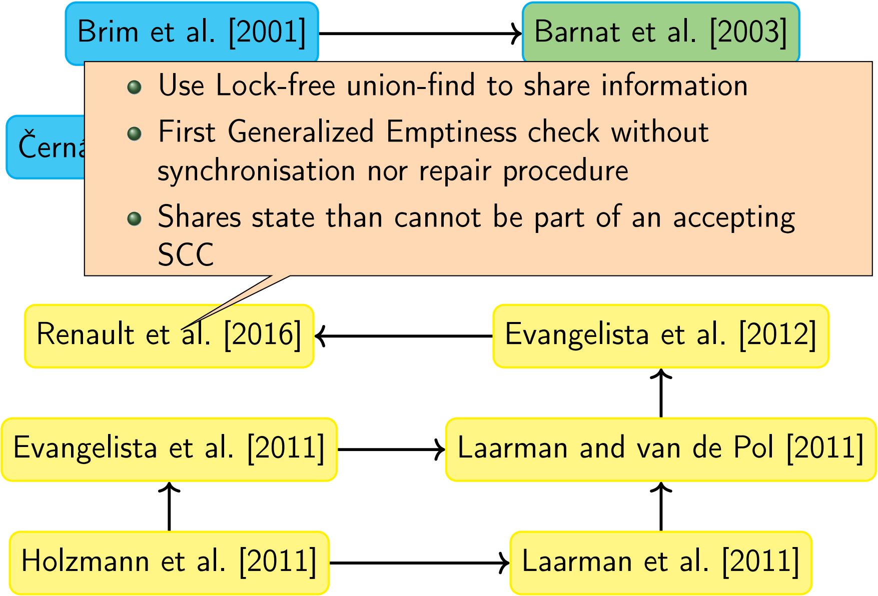

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]

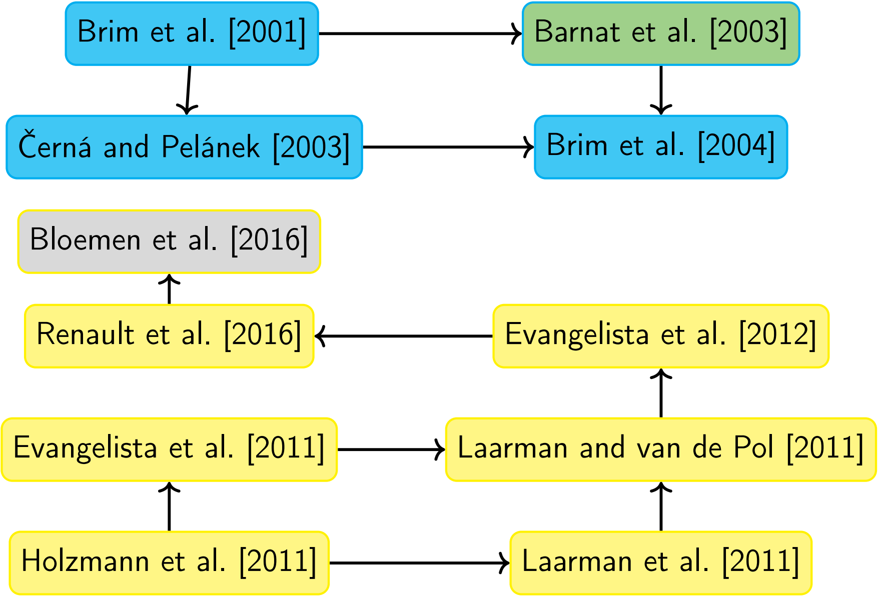

Evangelista et al. [2012]
Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]


Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Improve Laarman et al. [2011] with ideas of Evangelista et al. [2011]
Still synchronisations
Evangelista et al. [2012]
Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

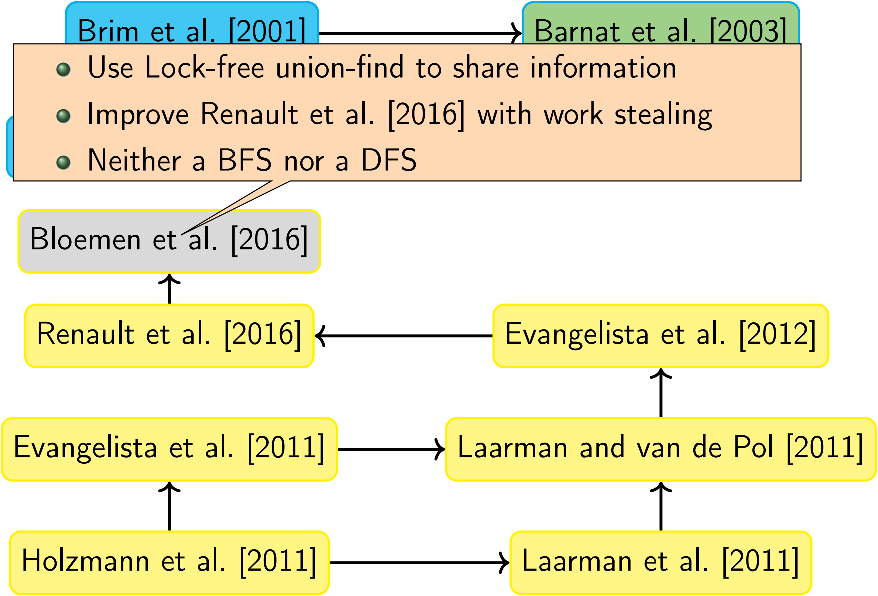

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Renault et al. [2016]
Evangelista et al. [2012]
Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

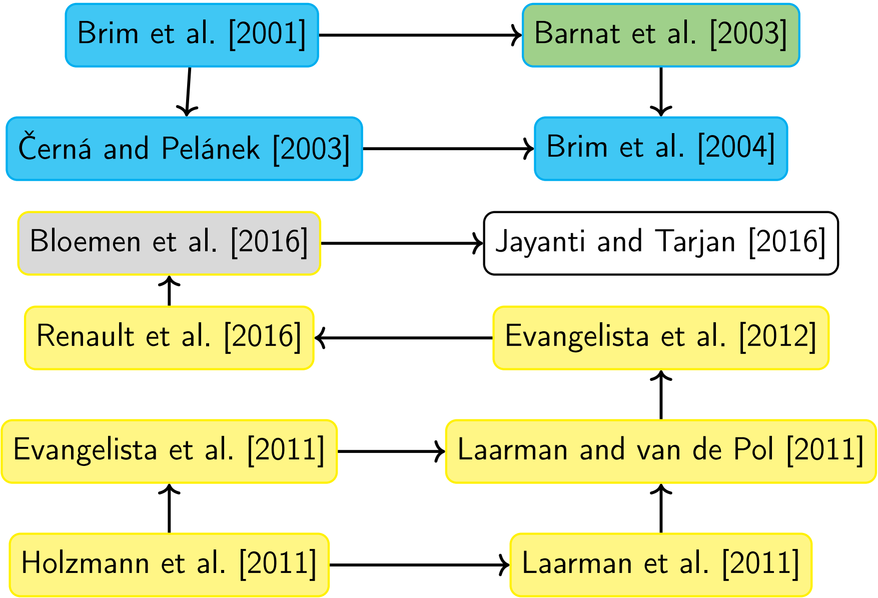
- Use Lock-free union-find to share information
- First Generalized Emptiness check without synchronisation nor repair procedure
- Shares state than cannot be part of an accepting SCC




Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Bloemen et al. [2016]
Renault et al. [2016]
Evangelista et al. [2012]
Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

- Use Lock-free union-find to share information
- Improve Renault et al. [2016] with work stealing
- Neither a BFS nor a DFS

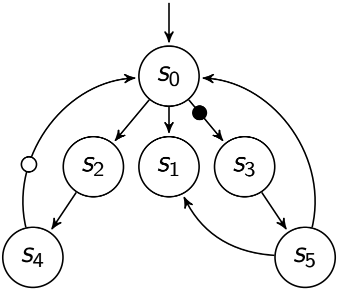


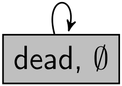

Brim et al. [2001]
Barnat et al. [2003]
Černá and Pelánek [2003]
Brim et al. [2004]
Bloemen et al. [2016]
Jayanti and Tarjan [2016]
Renault et al. [2016]
Evangelista et al. [2012]
Evangelista et al. [2011]
Laarman and van de Pol [2011]
Holzmann et al. [2011]
Laarman et al. [2011]

Example for Bloemen et al.

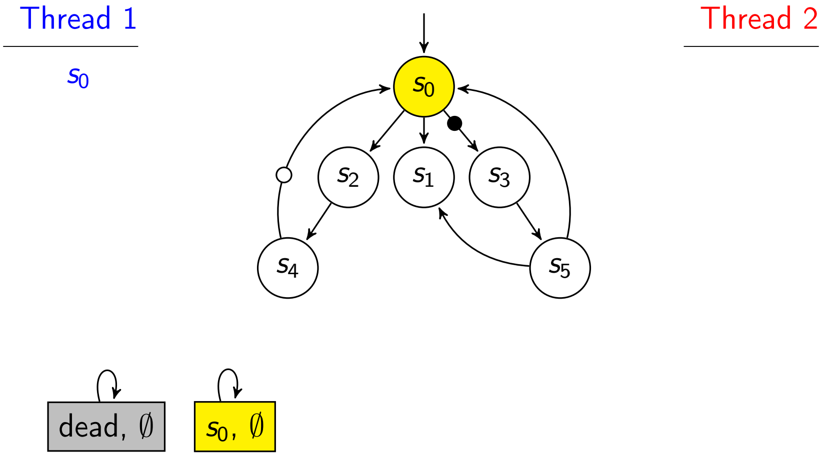
Thread 1

Thread 2





Example for Bloemen et al.



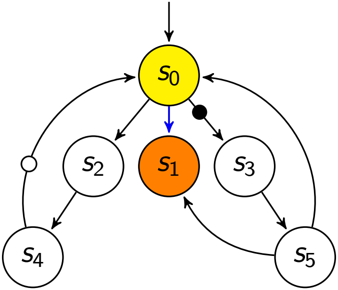
Example for Bloemen et al.

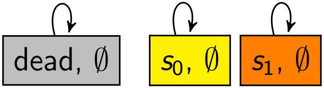
Thread 1

s_0

s_1

Thread 2



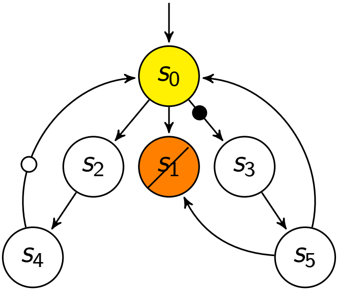


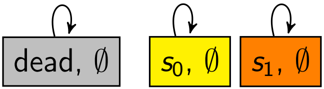
Example for Bloemen et al.

Thread 1

s_0
$s_1[s_1]$

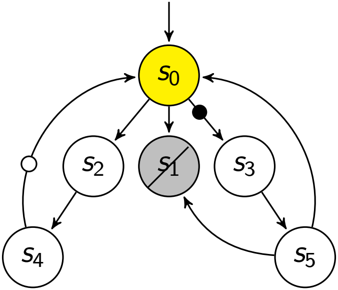
Thread 2

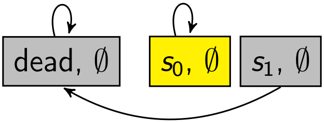




Example for Bloemen et al.

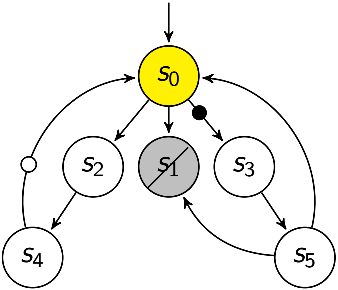
Thread 1	Thread 2
s_0	

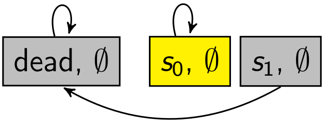




Example for Bloemen et al.

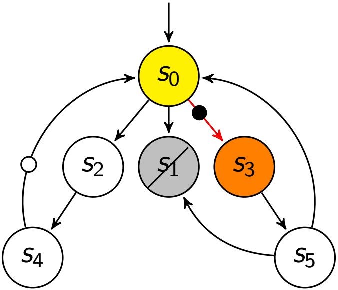
Thread 1	Thread 2
s_0	s_0



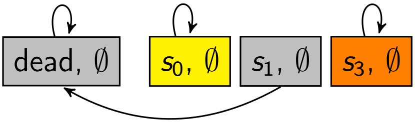


Example for Bloemen et al.

Thread 1
s_0

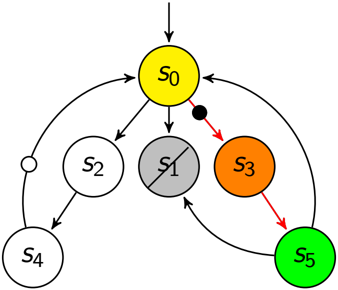


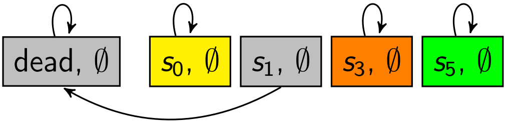
Thread 2
s_0
s_3



Example for Bloemen et al.

Thread 1
s_0





Thread 2
s_0
s_3
s_5

Example for Bloemen et al.

Thread 1

s_0

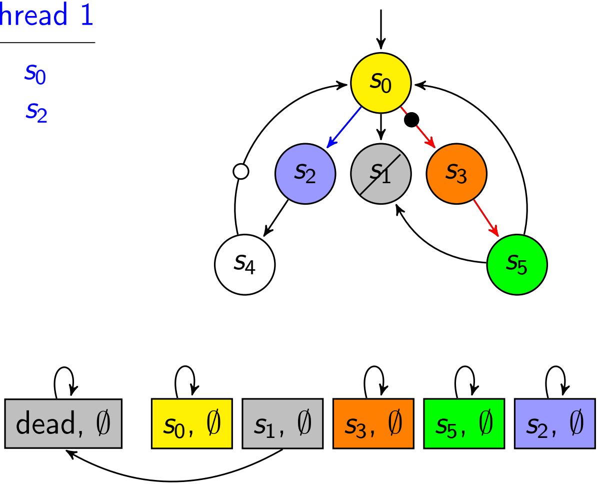
s_2

Thread 2

s_0

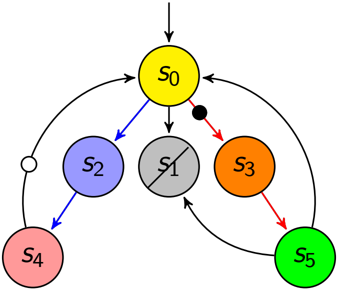
s_3

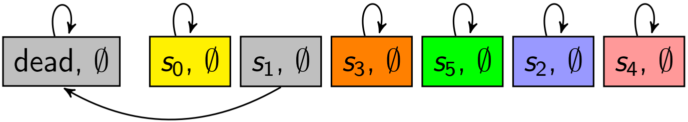
s_5



Example for Bloemen et al.

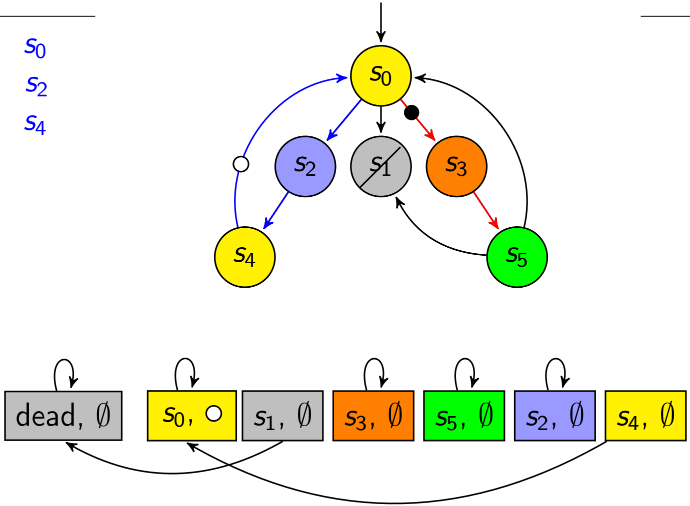
Thread 1	Thread 2
s_0	s_0
s_2	s_3
s_4	s_5



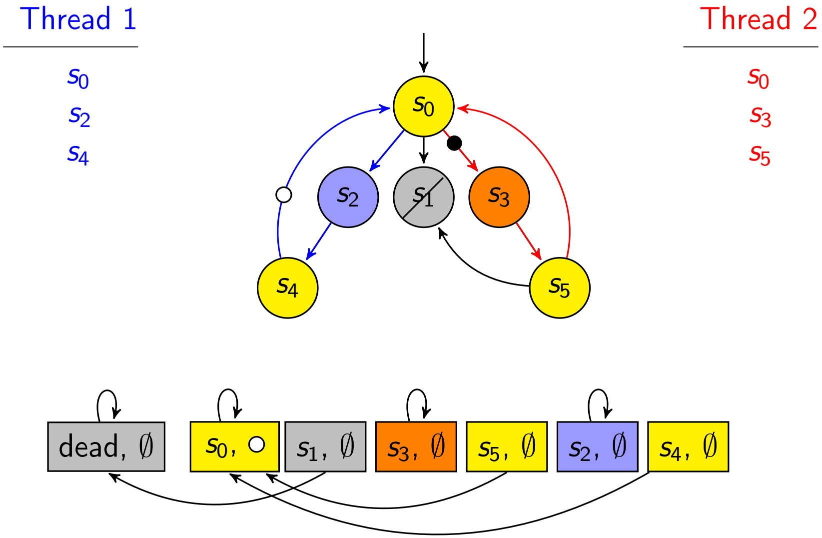


Example for Bloemen et al.

Thread 1	Thread 2
s_0	s_0
s_2	s_3
s_4	s_5

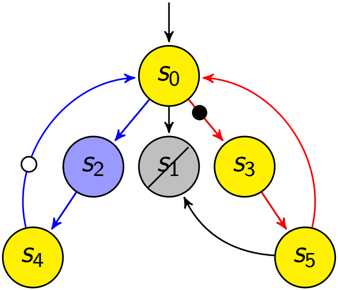


Example for Bloemen et al.

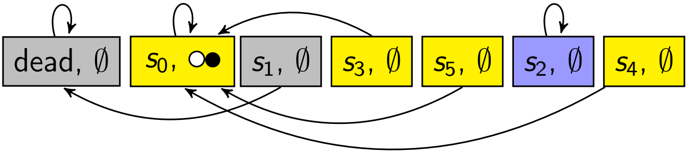


Example for Bloemen et al.

Thread 1
s_0
s_2
s_4



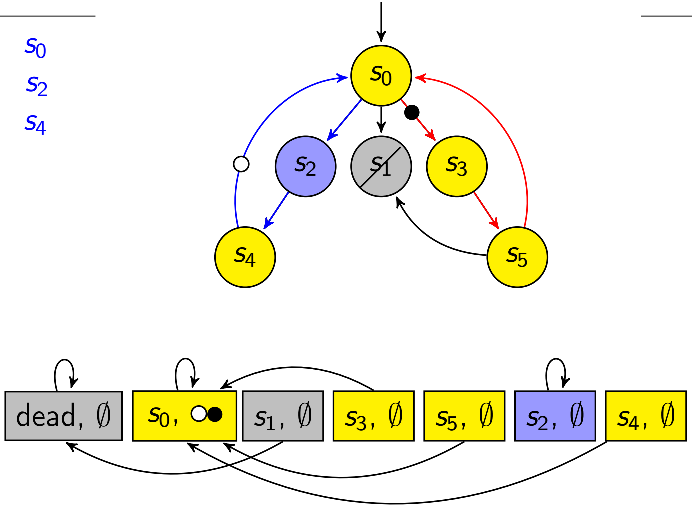
Thread 2
s_0
s_3
s_5



If $\mathcal{F} = \{\bullet, \circ\}$ then report counterexampe otherwise continue!

Example for Bloemen et al.

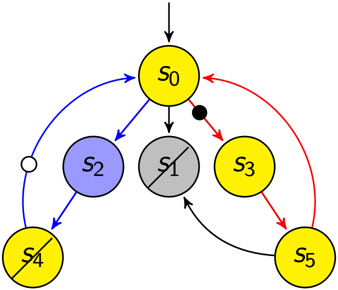
Thread 1
s_0
s_2
s_4



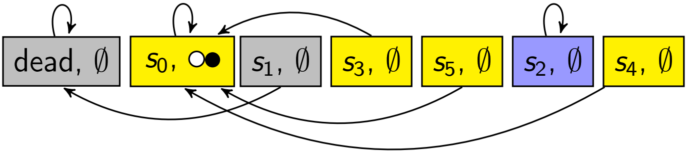
Thread 2
s_0
s_3
s_5

Example for Bloemen et al.

Thread 1
s_0
s_2
s_4



Thread 2
s_0
s_3
$s_5[s_4]$



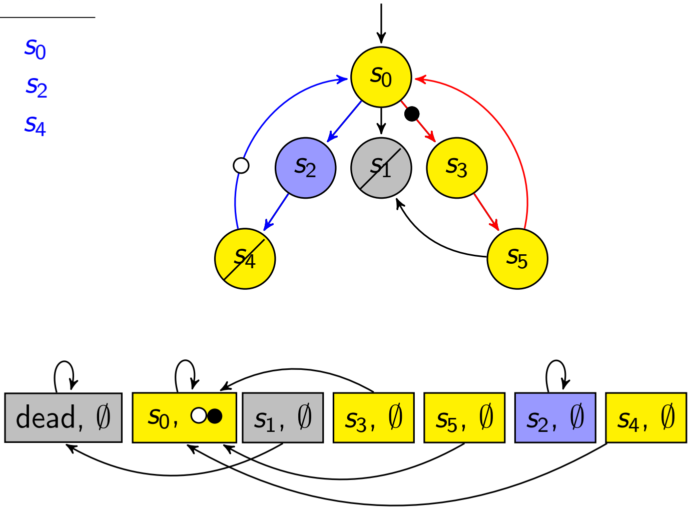
Example for Bloemen et al.

Thread 1

s_0
s_2
s_4

Thread 2

s_0
s_3
$s_5[s_0]$
s_2



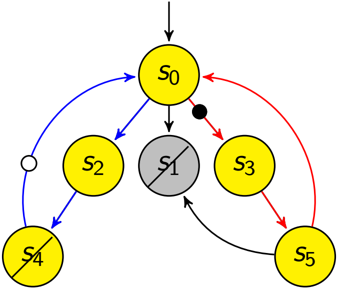
Example for Bloemen et al.

Thread 1

s_0

s_2

s_4



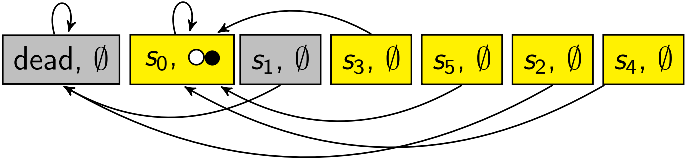
Thread 2

s_0

s_3

$s_5[s_0]$

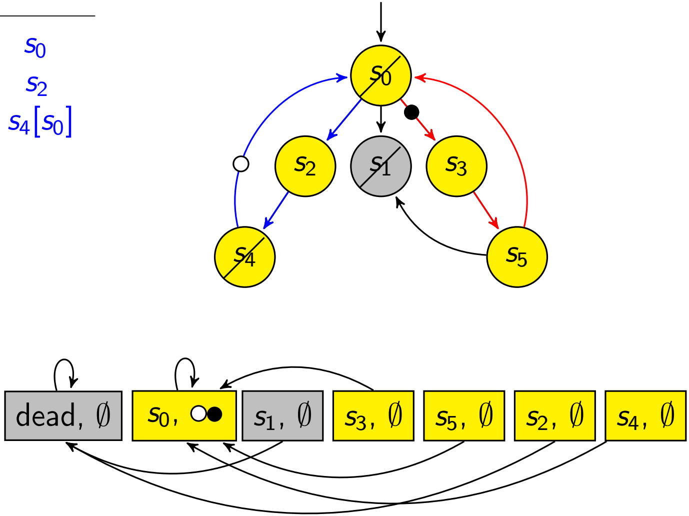
s_2



Example for Bloemen et al.

Thread 1

s_0
s_2
$s_4[s_0]$



Thread 2

s_0
s_3
$s_5[s_0]$
s_2

Example for Bloemen et al.

Thread 1

s_0

s_2

$s_4[s_3]$

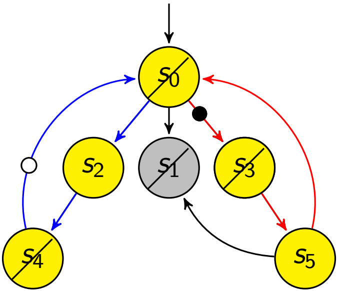
Thread 2

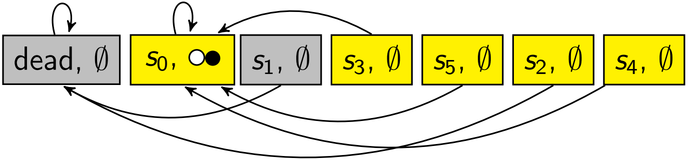
s_0

s_3

$s_5[s_0]$

s_2





Example for Bloemen et al.

Thread 1

s_0

s_2

$s_4[s_3]$

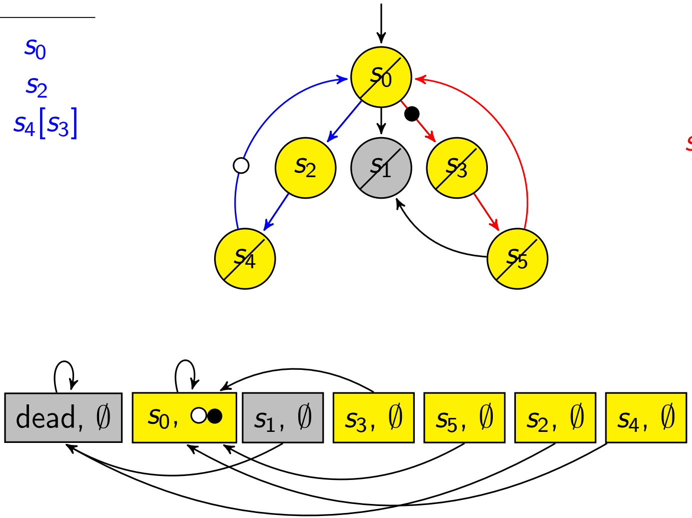
Thread 2

s_0

s_3

$s_5[s_0]$

$s_2[s_5]$



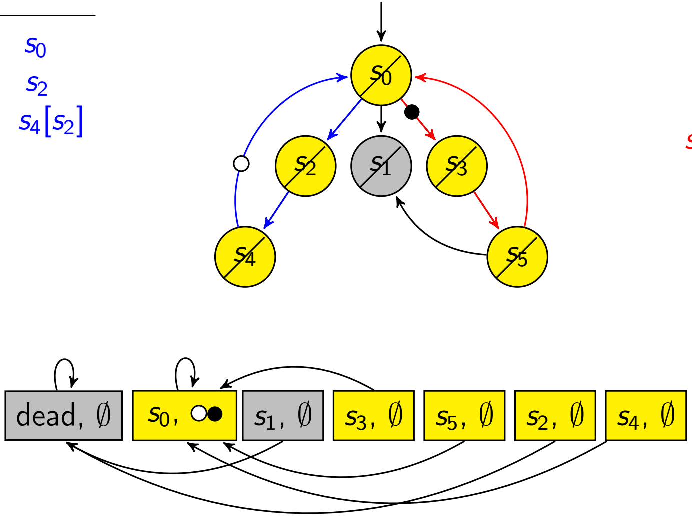
Example for Bloemen et al.

Thread 1

s_0

s_2

$s_4[s_2]$



Thread 2

s_0

s_3

$s_5[s_0]$

$s_2[s_5]$

Example for Bloemen et al.

Thread 1

s_0

s_2

$s_4[s_2]$

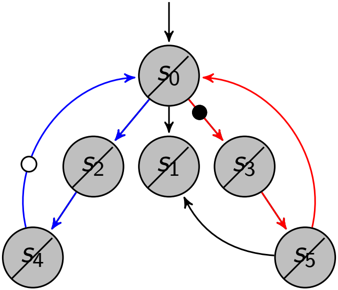
Thread 2

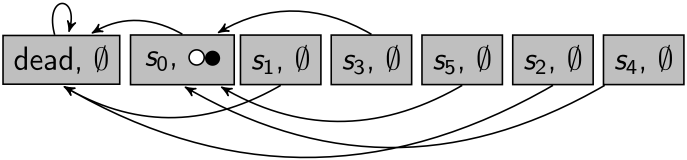
s_0

s_3

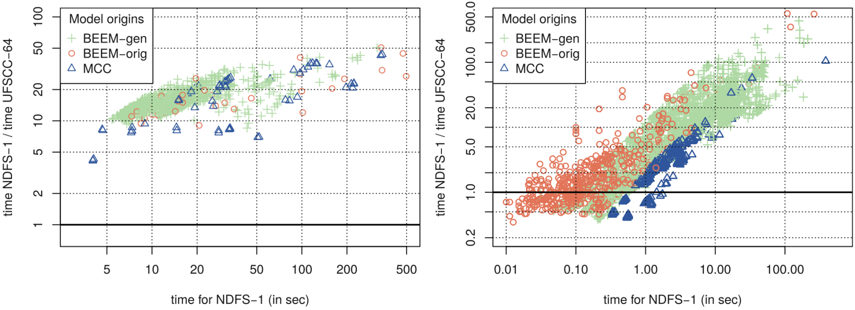
$s_5[s_0]$

$s_2[s_5]$



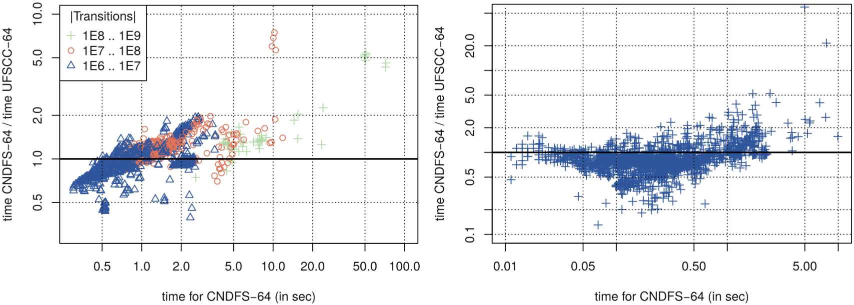


Results 1/2 (Bloemen and van de Pol [2016])



(a) Without counterexamples (b) With counterexamples

Results 1/2 (Bloemen and van de Pol [2016])



(a) Without counterexamples (b) With counterexamples

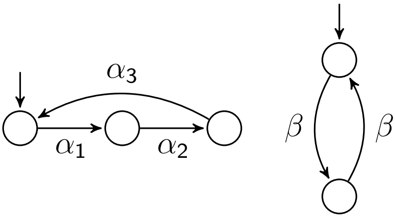
The lakes of the POR & the one hundred bridges of the proviso


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
Symbolic techniques
Mountains
ω-automata
villages
The forest of the Emptiness
Dijkstra Temple
SCC hills
Bitstate haching
State space caching
Tarjan Temple
On the fly
The one hundred bridges of the proviso
The dead Forest of the
Union-Find
Decomposition Tower
Renault
UFSCC
Map
Stubborn
Lakes of the POR
Conflicting
bledge
The outpost of the parellism
Persistent
owcty
swarming
Linear Topology
SCC+POR
Timed Automata
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands
1
2
3
4
5

State Space Explosion

- Two concurrent processes
- β independent of α_1, α_2, and α_3

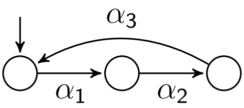
Process 1	Process 2



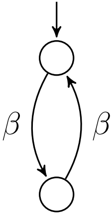
State Space Explosion

- Two concurrent processes
- β independent of α_1, α_2, and α_3

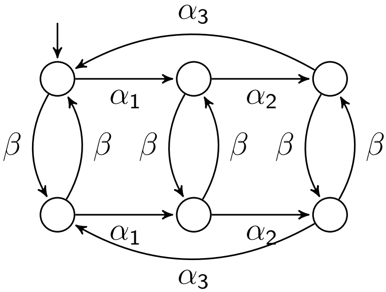
Process 1



Process 2



State Space

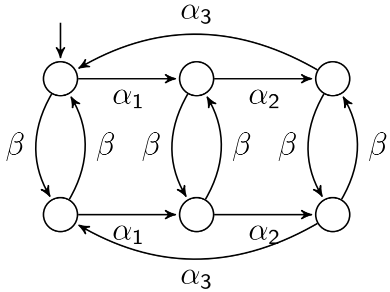


Process interleavings are one of the main sources of state-space explosion for explicit model checkers

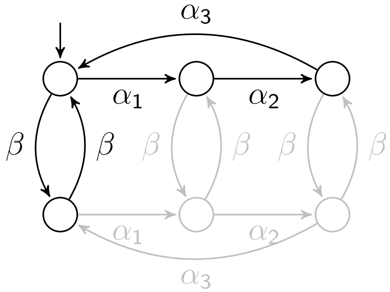
Partial Order Reductions (POR)

- Build a reduced state space
- For each state only consider a **reduced** subset of actions

State Space



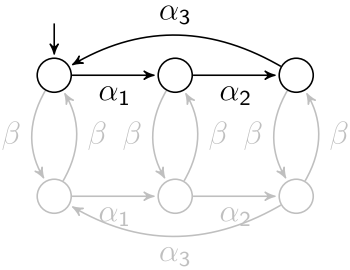
Possible Reduced State Space



POR work only iff the property to check belongs to LTL\X

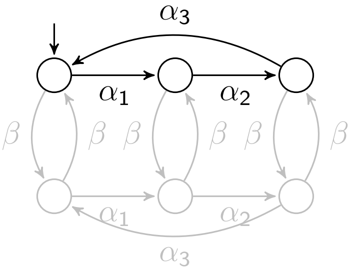
The Ignoring Problem for Liveness Properties

- If the same actions are consistently ignored along a cycle, they may never be executed (below β is never executed)



The Ignoring Problem for Liveness Properties

- If the same actions are consistently ignored along a cycle, they may never be executed (below β is never executed)



Requires an extra condition: the proviso

A proviso[a] ensures that every cycle in the reduced graph contains at least one **expanded state**, i.e, a state where all actions are considered.

[a]More simpler provisos can be applied for safety properties Evangelista and Pajault [2010]

Model Checking LTL\X with POR

Use classical DFS-based emptiness checks

During DFS:

- how to detect cycles without expanded states?
- which state to expand in a cycle?

Objectives:

- Choose states to expand states in order to have the smallest reduced state space

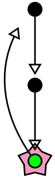
Variations on SPIN's proviso

SOURCE [Peled, 1994]

Expanded state  Not expanded state ● Already visited edge →

Variations on SPIN's proviso

SOURCE [Peled, 1994]

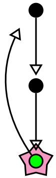


Systematically expands the source of a backedge

Expanded state  Not expanded state ● Already visited edge →

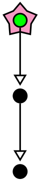
Variations on SPIN's proviso

SOURCE [Peled, 1994]



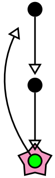
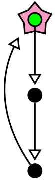
Systematically expands the source of a backedge

CONDSOURCE



Expanded state · Not expanded state ● · Already visited edge →

Variations on SPIN's proviso

SOURCE [Peled, 1994]	CONDSOURCE
	
Systematically expands the source of a backedge	Expands the source of backedge iff destination is not expanded

Expanded state ✩ Not expanded state ● Already visited edge →

Evaluation

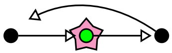
- 38 models from the BEEM benchmark
- *reduced* implements the stubborn-set method from Valmari
- Each model is run 100 times with different transition order

	states (10^6)		transitions (10^6)		st/ms
Full	784.45	100.00%	2,677.73	100.00%	17.90
SOURCE [Peled, 1994]	303.21	38.65%	679.16	25.36%	12.33
CONDSOURCE	252.83	32.23%	518.80	19.37%	11.85
None	57.58	7.34%	97.65	3.65%	22.65

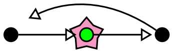
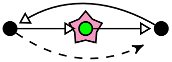
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CONDSOURCE

Deconstructing Evangelista and Pajault [2010] proviso

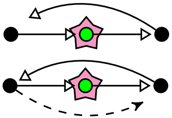
- Based on CONDSOURCE
- Try to reduce useless expansions: 

Deconstructing Evangelista and Pajault [2010] proviso

- Based on CONDSOURCE
- Try to reduce useless expansions: 
- Must consider all closing-edges: 

Deconstructing Evangelista and Pajault [2010] proviso

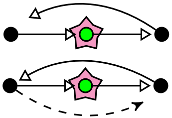
- Based on CONDSOURCE
- Try to reduce useless expansions:
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded

Deconstructing Evangelista and Pajault [2010] proviso

- Based on CONDSOURCE
- Try to reduce useless expansions:
- Must consider all closing-edges:

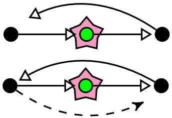


- Colors: safe, dangerous, on-dfs & not expanded

WEIGHTED	SCAN	KNOWN

Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:
- Must consider all closing-edges:



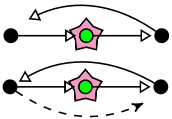
- Colors: safe, dangerous, on-dfs & not expanded

Weighted	Scan	Known

Keep track of exp-
-anded states on DFS

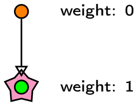
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded

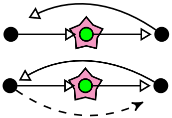
Weighted	Scan	Known



Keep track of exp-
-anded states on DFS

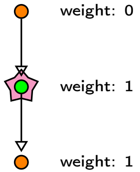
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded

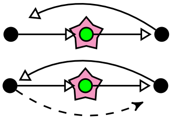
Weighted	Scan	Known



Keep track of exp-
-anded states on DFS

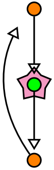
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CONDSOURCE
- Try to reduce useless expansions:
- Must consider all closing-edges:



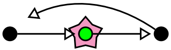
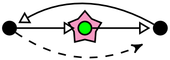
- Colors: safe, dangerous, on-dfs & not expanded

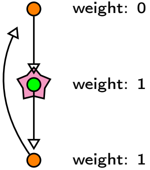
WEIGHTED	SCAN	KNOWN



Keep track of exp-
-anded states on DFS

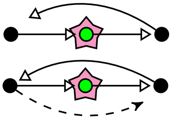
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions: 
- Must consider all closing-edges: 
- Colors: safe, dangerous, on-dfs & not expanded

Weighted	Scan	Known
		
Keep track of exp- -anded states on DFS	Early tag "safe" states	

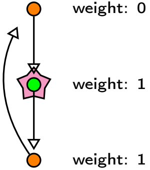
Deconstructing Evangelista and Pajault [2010] proviso

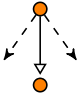
- Based on CONDSOURCE
- Try to reduce useless expansions:
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded

WEIGHTED	SCAN	KNOWN



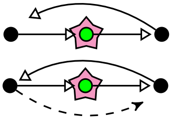


Keep track of exp- -anded states on DFS

Early tag "safe" states

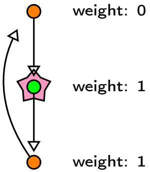
Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:
- Must consider all closing-edges:

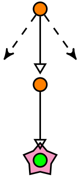


- Colors: safe, dangerous, on-dfs & not expanded

Weighted	Scan	Known



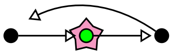
Keep track of exp- -anded states on DFS



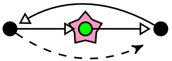
Early tag "safe" states

Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:

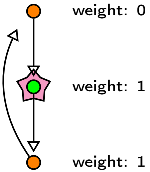


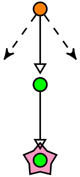
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded

Weighted	Scan	Known



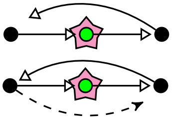


Keep track of exp- -anded states on DFS

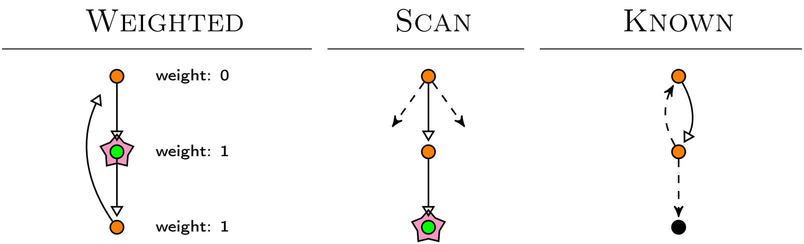
Early tag "safe" states

Deconstructing Evangelista and Pajault [2010] proviso

- Based on CondSource
- Try to reduce useless expansions:
- Must consider all closing-edges:



- Colors: safe, dangerous, on-dfs & not expanded



Weighted	Scan	Known
Keep track of exp- -anded states on DFS	Early tag "safe" states	Prioritizing known successors

Evaluation of each optimization

	states (10^6)		transitions (10^6)		st/ms
Full	784.45	100.00%	2,677.73	100.00%	17.90
Source [Peled, 1994]	303.21	38.65%	679.16	25.36%	12.33
WeightedSource	263.43	33.58%	537.56	20.08%	11.68
WeightedSourceKnown[1]	262.63	33.48%	534.35	19.96%	11.77
CondSource	252.83	32.23%	518.80	19.37%	11.85
CondSourceKnown	251.05	32.00%	510.91	19.08%	11.89
WeightedSourceScan	250.49	31.93%	505.98	18.90%	11.67
WeightedSourceKnownScan[1]	248.11	31.63%	498.68	18.62%	11.70
None	57.58	7.34%	97.65	3.65%	22.65

- SOURCE have the best throughput
- Most of the improvement comes from COND
- Evangelista's provisos outperforms SOURCE

[1] [Evangelista and Pajault, 2010]

Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

SOURCE

DEST

Systematically expands the source of a backegde

Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

SOURCE	DEST

Systematically expands the source of a backegde

Provisos Based on Destination Expansion

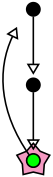
- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

Systematically expands the source of a backegde

Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

SOURCE	DEST

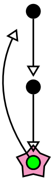


Systematically expands the source of a backegde

Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

SOURCE	DEST

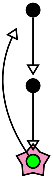


Systematically expands the source of a backegde

Systematically expands the destination of a backedge

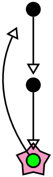
Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

Source	Dest
	
Systematically expands the source of a backegde	Systematically expands the destination of a backedge

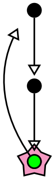
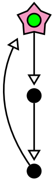
Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

SOURCE	DEST
	
Systematically expands the source of a backegde	Systematically expands the destination of a backedge

Provisos Based on Destination Expansion

- Proposed by Nalumasu and Gopalakrishnan [2002] in a narrower context

Source	Dest
	
Systematically expands the source of a backegde	Systematically expands the destination of a backedge

Optimizations for these new provisos

- Compatible with: COND, WEIGHTED, KNOWN

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Optimizations for these new provisos

- Compatible with: Cond, Weighted, Known

Colored	Unknown	Deepest

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Optimizations for these new provisos

- Compatible with: Cond, Weighted, Known

Colored	Unknown	Deepest

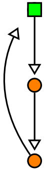
Reuse colors
Mark for expansion
Expand iff necessary

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Optimizations for these new provisos

- Compatible with: COND, WEIGHTED, KNOWN

COLORED	UNKNOWN	DEEPEST

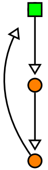
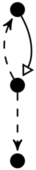


Reuse colors
Mark for expansion
Expand iff necessary

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Optimizations for these new provisos

- Compatible with: COND, WEIGHTED, KNOWN

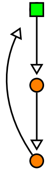
COLORED	UNKNOWN	DEEPEST
		
Reuse colors Mark for expansion Expand iff necessary	Prioritizing unknown successsors	

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Optimizations for these new provisos

- Compatible with: Cond, Weighted, Known

Colored



Reuse colors
Mark for expansion
Expand iff necessary

Unknown

Prioritizing
unknown
successsors

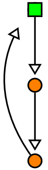
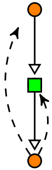
Deepest

Only mark the deepest
dest. for expansion

Mark for expansion ■ Already visited edge → Not yet visited edge - ->

Optimizations for these new provisos

- Compatible with: COND, WEIGHTED, KNOWN

COLORED	UNKNOWN	DEEPEST
		
Reuse colors Mark for expansion Expand iff necessary	Prioritizing unknown successsors	Only mark the deepest dest. for expansion

Mark for expansion ■ Already visited edge → Not yet visited edge ⇢

Evaluation

	states (10^6)		transitions (10^6)		st/ms
DeepestDestUnknown	276.51	35.25%	570.52	21.31%	11.81
DeepestDest	275.31	35.10%	566.63	21.16%	11.87
WeightedDestUnknown	273.94	34.92%	563.61	21.05%	11.83
Dest	272.79	34.77%	508.17	18.98%	14.48
WeightedDest	272.68	34.76%	559.73	20.90%	11.80
WeightedSourceKnownScan	248.11	31.63%	498.68	18.62%	11.70
CondDest	213.98	27.28%	413.15	15.43%	12.57
CondDestUnknown	213.92	27.27%	412.75	15.41%	12.52
ColoredDest	213.92	27.27%	412.93	15.42%	12.54
ColoredDestUnknown	213.83	27.26%	412.27	15.40%	12.46

- CondDest outperforms state-of-the-art provisos
- Weighted and Deepest variants are disappointing

Improving Provisos With SCCs information

- When destination is red, an expansion is required:
  - Until now, the source was expanded

Improving Provisos With SCCs information

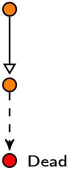
- When destination is red, an expansion is required:
  - Until now, the source was expanded

DEAD	HIGHLINKS

Improving Provisos With SCCs information

- When destination is red, an expansion is required:
  - Until now, the source was expanded

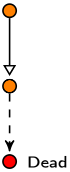
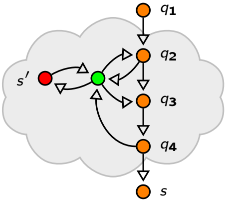
Dead	Highlinks



Avoid expansions when dest. is dead, i.e. in a fully visited SCC

Improving Provisos With SCCs information

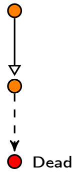
- When destination is red, an expansion is required:
  - Until now, the source was expanded

Dead	Highlinks
	
Avoid expansions when dest. is dead, i.e. in a fully visited SCC	Adaptation of Deepest when dest. is not on the DFS and not dead

Improving Provisos With SCCs information

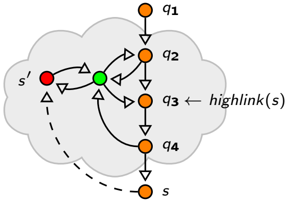
- When destination is red, an expansion is required:
  - Until now, the source was expanded

Dead



Avoid expansions when dest. is dead, i.e. in a fully visited SCC

Highlinks



Adaptation of Deepest when dest. is not on the DFS and not dead

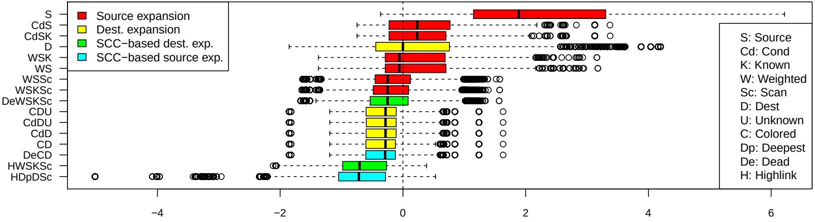
DEAD and HIGHLINKS are compatibles with both source and destination expansion-based provisos.

Evaluation 1/2

	states (10^6)		transitions (10^6)	
DeepestDest	275.31	35.10%	566.63	21.16%
DeadDeepestDest	269.10	34.30%	543.64	20.30%
WeightedDest	272.68	34.76%	559.73	20.90%
DeadWeightedDest	270.62	34.50%	554.91	20.72%
DeadWeightedSourceKnownScan	247.68	31.57%	497.79	18.59%
ColoredDest	213.92	27.27%	412.93	15.42%
DeadColoredDest	213.87	27.26%	412.80	15.42%
HighlinkWeightedDest	207.41	26.44%	393.22	14.68%
HighlinkWeightedDestScan	206.23	26.29%	391.05	14.60%
HighlinkWeightedSourceKnown	203.20	25.90%	386.84	14.45%
HighlinkWeightedSourceKnownScan	203.08	25.89%	386.60	14.44%
HighlinkDeepestDest	192.84	24.58%	349.89	13.07%
HighlinkDeepestDestScan	191.78	24.45%	347.95	12.99%

Evaluation 2/2

- Standard score for selected provisos
  - take the set of 1600 runs generated
  - compute a mean number μ_M for each model M
  - compute a standard deviation σ_M for each model M
  - standard score for a run r is then $\frac{states(r) - \mu_M}{\sigma_M}$
- Boxplot standard score



Results

- Overview of state-of-the-art provisos for checking liveness properties
- New heuristics: Colored, Deepest, Dead, Highlink
- Combination with existing heuristics
- Intensive evaluation
- Independant of the reduction technique: ample set, sttuborn set, etc. (see [Laarman et al., 2014] for survey)

Our recommended provisos:

- CondDest in NDFS-based emptiness-checks
- HighlinkWeightedSourceKnown in SCC-based emptiness checks (no scan required)

Explore new Lands . . .

Perspectives

- Parallel Algorithms
  - **Exploit Topology**:
    - *If the automaton to check is linear, parallel algorithms can't help to speed up computation*
  - **Mix UFSCC with POR**:
    - *CNDFS has been successfuly mixed with POR and can benefits from all previous techniques.*
  - **Improve classical ω-automata algorithms**
- Distributed Algorithms
  - **Improve existing algorithms**
  - **Build message-passing algorithms rather than shared memory-one**


Explicit Techniques Valley
strong
terminal
weak
NDFS
SCC
Symbolic techniques Mountains
ω-automata villages
The forest of the Emptiness
Dijkstra Temple
SCC hills
Tarjan Temple
Bitstate haching
State space caching
On the fly
The one hundred bridges of the proviso
Decomposition Tower
The dead Forest of the Union-Find
Renault
UFSCC
Map
Stubborn
Lakes of the POR
Conflicting
Persistent
Timed Automata
The outpost of the parellism
bledge
nege
owcty
swarming
Linear Topology
SCC+POR
endfs
lndfs
ω-automata parallel emptiness checks
CNDFS
mc-ndfs
Satisfiability
Distributed islands

Bibliography I

Alur, R., Chaudhuri, S., Etessami, K., and Madhusudan, P. (2005). On-the-fly reachability and cycle detection for recursive state machines. In Halbwachs, N. and Zuck, L., editors, Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), volume 3440 of Lecture Notes in Computer Science, pages 61–76. Springer Berlin Heidelberg.

Barnat, J., Brim, L., and Chaloupka, J. (2003). Parallel breadth-first search LTL model-checking. In Proceedings of the 18th IEEE International Conference On Automated Software Engineering (ASE'03), pages 106–115. IEEE Computer Society.

Bloemen, V., Laarman, A., and van de Pol, J. (2016). Multi-core On-the-fly SCC Decomposition. In Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'16). ACM.

Bloemen, V. and van de Pol, J. (2016). Multi-core scc-based ltl model checking. Hardware and Software: Verification and Testing: 12th International Haifa Verification Conference (HVC'16), pages 18–33.

Brim, L., Černá, I., Krcal, P., and Pelánek, R. (2001). Distributed LTL model checking based on negative cycle detection. In Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'01), pages 96–107.

Bibliography II

Brim, L., Černá, I., Moravec, P., and Šimša, J. (2004). Accepting predecessors are better than back edges in distributed LTL model-checking. In Hu, A. J. and Martin, A. K., editors, Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04), volume 3312 of Lecture Notes in Computer Science, pages 352–366. Springer.

Černá, I. and Pelánek, R. (2003). Distributed explicit fair cycle detection (set based approach). In Ball, T. and Rajamani, S., editors, Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN'03), volume 2648 of Lecture Notes in Computer Science, pages 49–73. Springer Berlin Heidelberg.

Cheriyan, J. and Mehlhorn, K. (1996). Algorithms for dense graphs and networks on the random access computer. Algorithmica, 15(6):521–549.

Courcoubetis, C., Vardi, M. Y., Wolper, P., and Yannakakis, M. (1991). Memory-efficient algorithm for the verification of temporal properties. In Clarke, E. M. and Kurshan, R. P., editors, Proceedings of the 2nd international workshop on Computer Aided Verification (CAV'90), volume 531 of Lecture Notes in Computer Science, pages 233–242. Springer-Verlag.

Couvreur, J.-M. (1999). On-the-fly verification of temporal logic. In Wing, J. M., Woodcock, J., and Davies, J., editors, Proceedings of the World Congress on Formal Methods in the Development of Computing Systems (FM'99), volume 1708 of Lecture Notes in Computer Science, pages 253–271, Toulouse, France. Springer-Verlag.

Bibliography III

Couvreur, J.-M., Duret-Lutz, A., and Poitrenaud, D. (2005). On-the-fly emptiness checks for generalized Büchi automata. In Godefroid, P., editor, Proceedings of the 12th International SPIN Workshop on Model Checking of Software (SPIN'05), volume 3639 of Lecture Notes in Computer Science, pages 143–158. Springer.

Dijkstra, E. W. (1973). EWD 376: Finding the maximum strong components in a directed graph. `http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD376.PDF`.

Edelkamp, S., Leue, S., and Lluch-Lafuente, A. (2004). Directed explicit-state model checking in the validation of communication protocols. STTT, 5(2–3):247–267.

Evangelista, S., Laarman, A., Petrucci, L., and van de Pol, J. (2012). Improved multi-core nested depth-first search. In Proceedings of the 10th international conference on Automated technology for verification and analysis (ATVA'12), volume 7561 of Lecture Notes in Computer Science, pages 269–283. Springer-Verlag.

Evangelista, S. and Pajault, C. (2010). Solving the ignoring problem for partial order reduction. STTT, 12(2):155–170.

Evangelista, S., Petrucci, L., and Youcef, S. (2011). Parallel nested depth-first searches for LTL model checking. In Proceedings of the 9th international conference on Automated technology for verification and analysis (ATVA'11), volume 6996 of Lecture Notes in Computer Science, pages 381–396. Springer-Verlag.

Gabow, H. N. (2000). Path-based depth-first search for strong and biconnected components. Information Processing Letters, 74(3-4):107–114.

Bibliography IV

Gaiser, A. and Schwoon, S. (2009). Comparison of algorithms for checking emptiness on Büchi automata. In Hlinený, P., Matyás, V., and Vojnar, T., editors, Procedings of Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'09), volume 13 of OASICS. Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, Germany.

Geldenhuys, J. and Valmari, A. (2004). Tarjan's algorithm makes on-the-fly LTL verification more efficient. In Jensen, K. and Podelski, A., editors, Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04), volume 2988 of Lecture Notes in Computer Science, pages 205–219. Springer.

Geldenhuys, J. and Valmari, A. (2005). More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoretical Computer Science, 345(1):60–82.

Godefroid, P. and Holzmann, G. J. (1993). On the verification of temporal properties. In Danthine, A. A. S., Leduc, G., and Wolper, P., editors, Proceedings of the 13th IFIP TC6/WG6.1 International Symposium on Protocol Specification, Testing, and Verification (PSTV'93), volume C-16 of IFIP Transactions, pages 109–124, Liege, Belgium. North-Holland.

Hansen, H. and Geldenhuys, J. (2008). Cheap and small counterexamples. In Cerone, A. and Gruner, S., editors, Proceedings of the 6th IEEE International Conference on Software Engineering and Formal Methods (SEFM'08), pages 53–62. IEEE Computer Society.

Holzmann, G. J. (1991). Design and Validation of computer protocols, volume 07632 of Prentice Hall Software Series. Brian W. Kernighan.

Bibliography V

Holzmann, G. J., Joshi, R., and Groce, A. (2011). Swarm verification techniques. IEEE Transaction on Software Engineering, 37(6):845–857.

Holzmann, G. J. and Peled, D. (1994). An improvement in formal verification. In Proceeding of the 7th IFIP WG 6.1 International Conference on Formal Description Techniques (FORTE'94), volume 6 of IFIP Conference Proceedings, pages 109–124, Berne, Switzerland. Chapman & Hall.

Holzmann, G. J., Peled, D. A., and Yannakakis, M. (1996). On nested depth first search. In Grégoire, J.-C., Holzmann, G. J., and Peled, D. A., editors, Proceedings of the 2nd Spin Workshop, volume 32 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

Jayanti, S. V. and Tarjan, R. E. (2016). A randomized concurrent algorithm for disjoint set union. pages 75–82.

Laarman, A., Langerak, R., van de Pol, J., Weber, M., and Wijs, A. (2011). Multi-core nested depth-first search. In Bultan, T. and Hsiung, P.-A., editors, Proceedings of the Automated Technology for Verification and Analysis, 9th International Symposium (ATVA'11), volume 6996 of Lecture Notes in Computer Science, pages 321–335, Taipei, Taiwan. Springer.

Laarman, A., Pater, E., Pol, J., and Hansen, H. (2014). Guard-based partial-order reduction. STTT, pages 1–22.

Laarman, A. and van de Pol, J. (2011). Variations on multi-core nested depth-first search. In PDMC, pages 13–28.

Bibliography VI

Nalumasu, R. and Gopalakrishnan, G. (2002). An efficient partial order reduction algorithm with an alternative proviso implementation. FMSD, 20(1):231–247.

Nuutila, E. and Soisalon-Soininen, E. (1994). On finding the strongly connected components in a directed graph. Information Processing Letters, 49(1):9–14.

Peled, D. (1994). Combining partial order reductions with on-the-fly model-checking. In Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94), volume 818 of Lecture Notes in Computer Science, pages 377–390. Springer-Verlag.

Reif, J. H. (1985). Depth-first search is inherently sequential. Information Processing Letters, 20:229–234.

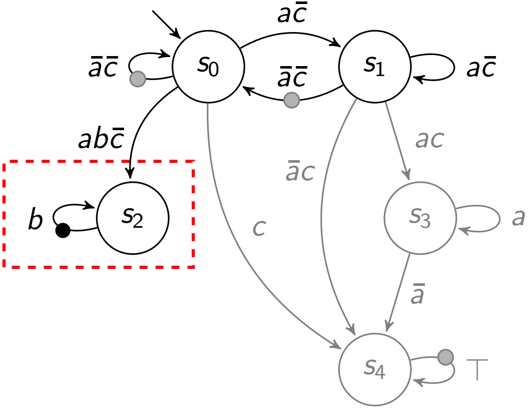
Renault, E., Duret-Lutz, A., Kordon, F., and Poitrenaud, D. (2016). Variations on parallel explicit model checking for generalized Büchi automata. International Journal on Software Tools for Technology Transfer (STTT), ??(??):??–??

Schwoon, S. and Esparza, J. (2005). A note on on-the-fly verification algorithms. In Halbwachs, N. and Zuck, L., editors, Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05), volume 3440 of Lecture Notes in Computer Science, Edinburgh, Scotland. Springer.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1(2):146–160.

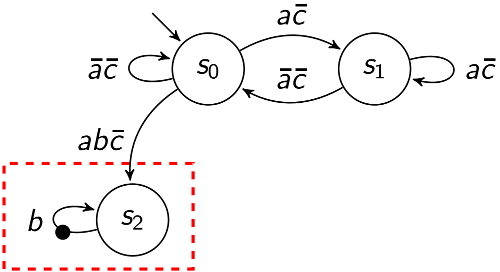
Tauriainen, H. (2004). Nested emptiness search for generalized Büchi automata. In Proceedings of the 4th International Conference on Application of Concurrency to System Design (ACSD'04), pages 165–174. IEEE Computer Society.

Construction of A_W



All acceptance sets are removed and
a single acceptance set labels all transitions of *weak* SCC.

Construction of A_W



All acceptance sets are removed and
a single acceptance set labels all transitions of *weak* SCC.