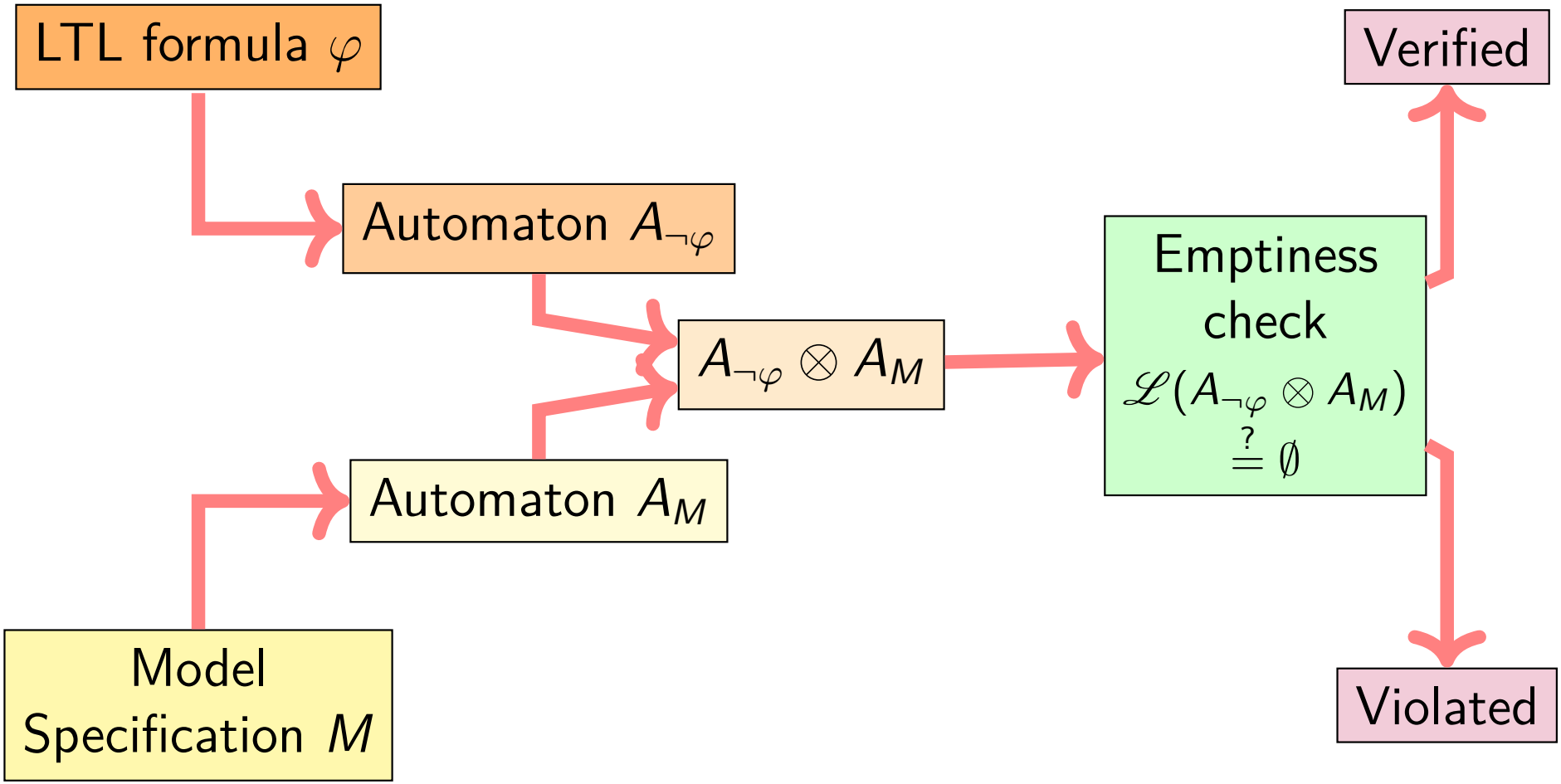
Combining Parallel Emptiness Checks with Partial Order Reductions

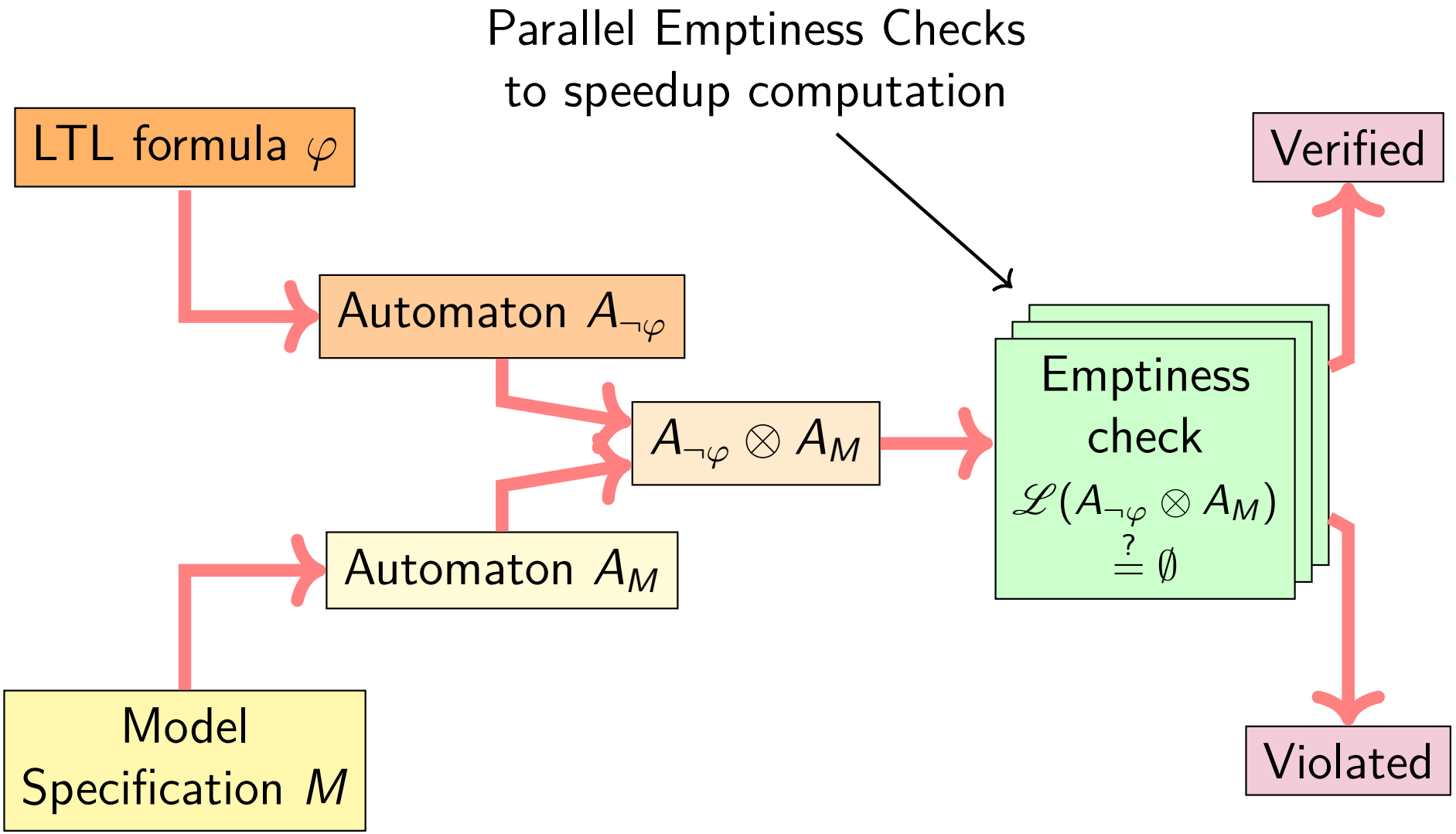
D. Poitrenaud, E. Renault

Friday November 8th

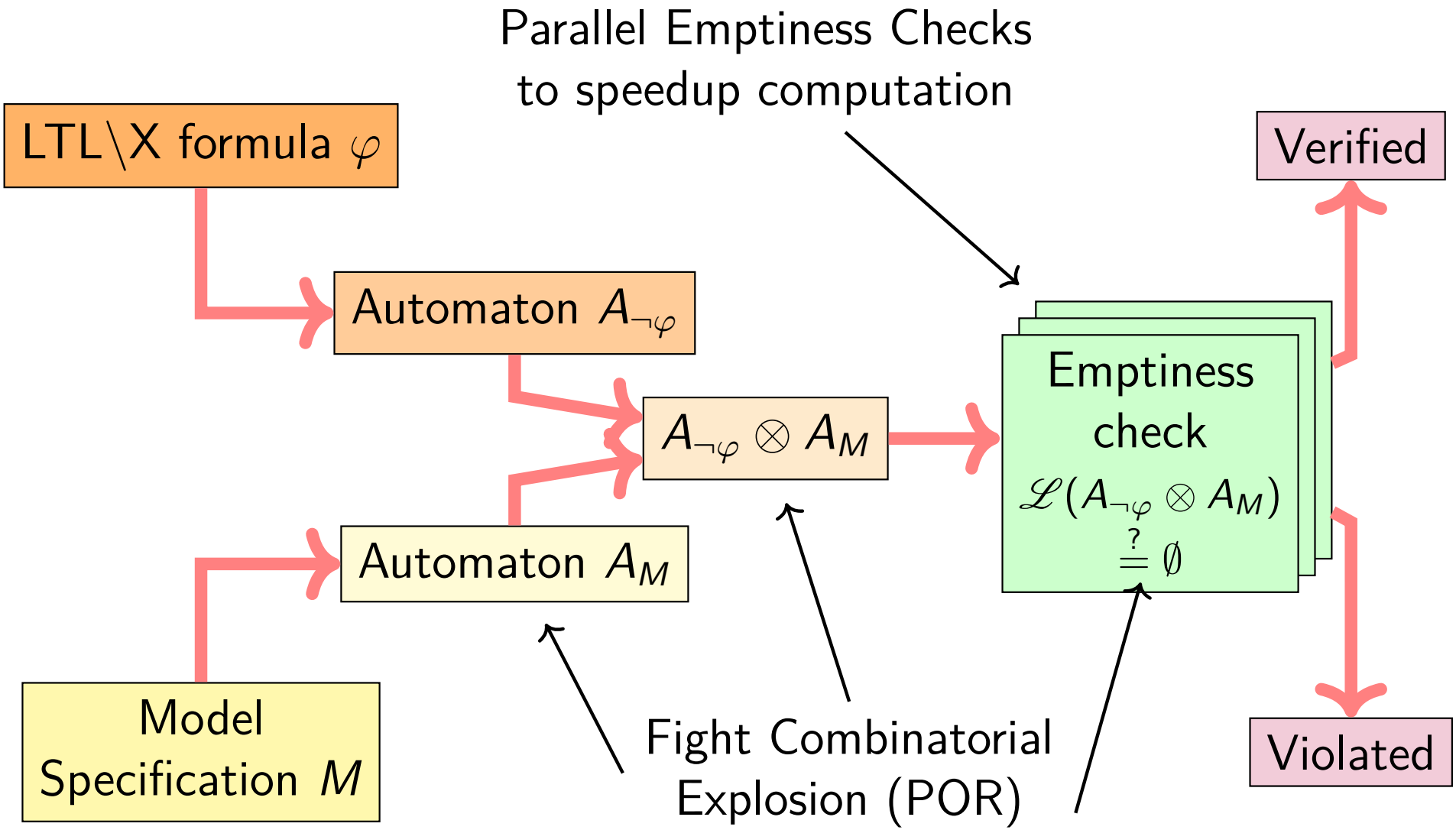
Automata-Theoretic Approach to Model Checking



Automata-Theoretic Approach to Model Checking



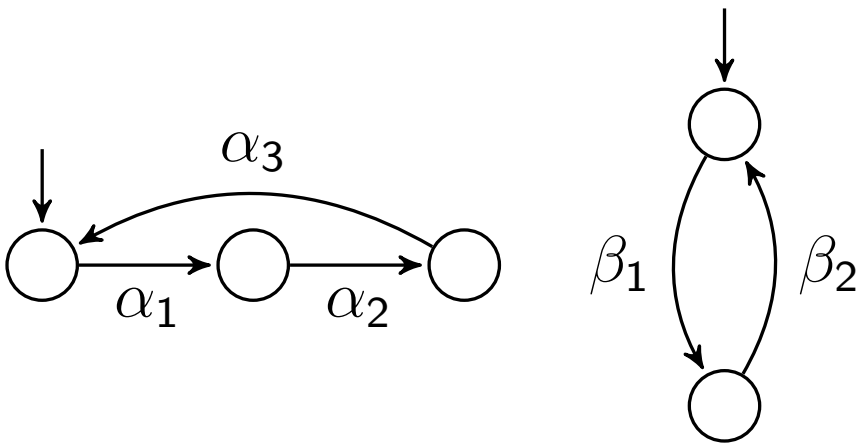
Automata-Theoretic Approach to Model Checking



State Space Explosion

- Two concurrent processes
- β_1, β_2 independent of α_1, α_2, and α_3

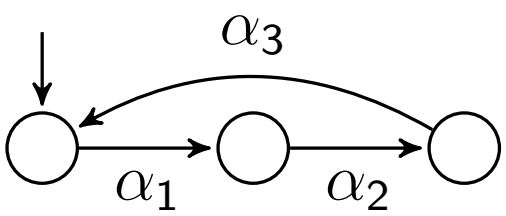
Process 1 | Process 2



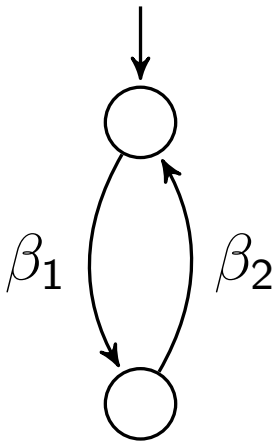
State Space Explosion

- Two concurrent processes
- β_1, β_2 independent of α_1, α_2, and α_3

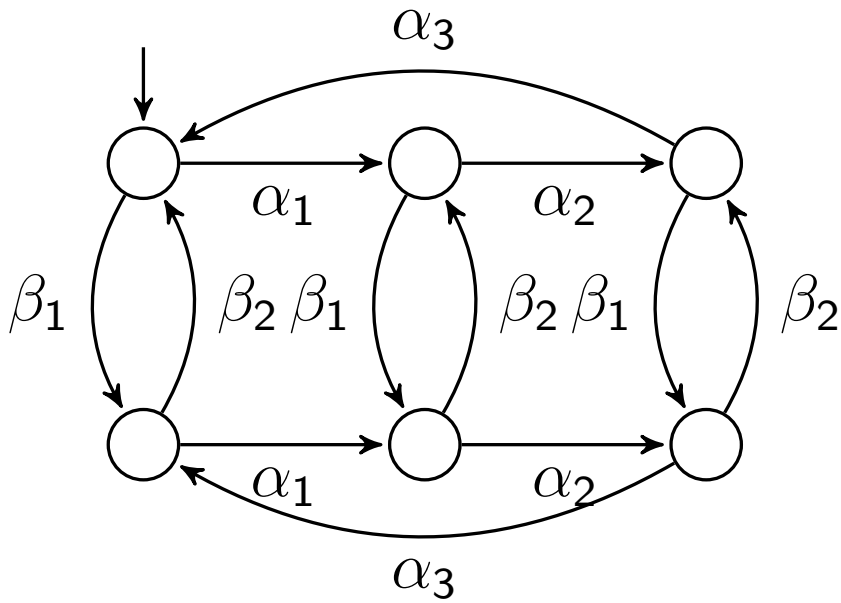
Process 1



Process 2



State Space

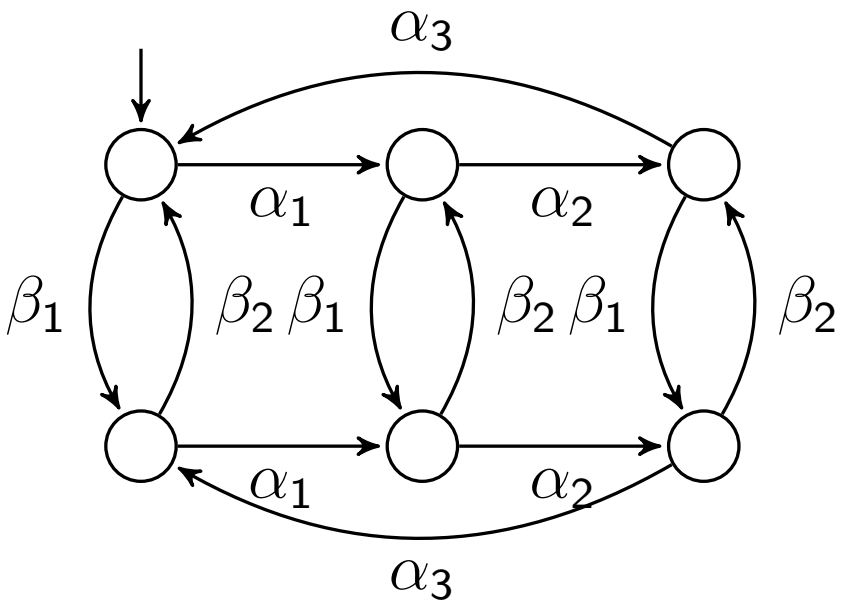


Process interleavings are one of the main sources of state-space explosion for explicit model checkers

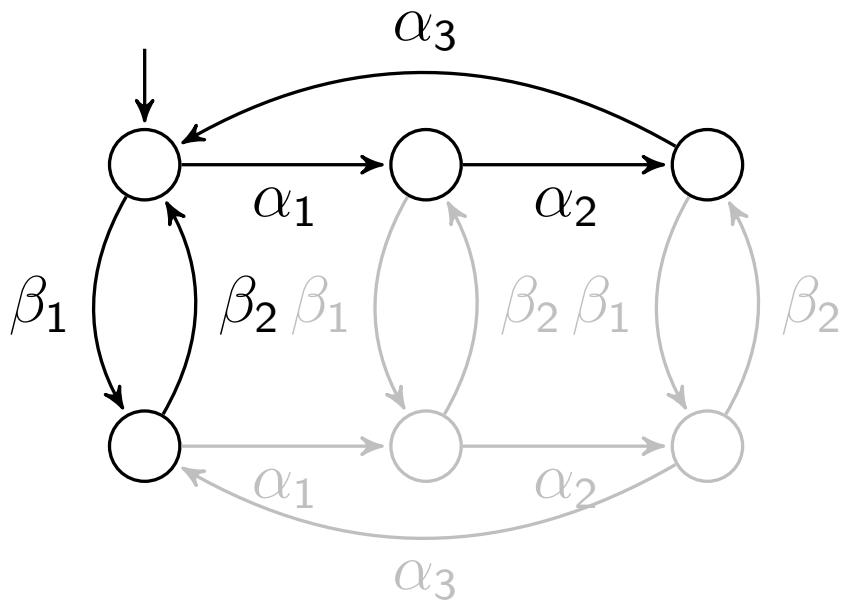
Partial Order Reductions (POR)

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

State Space



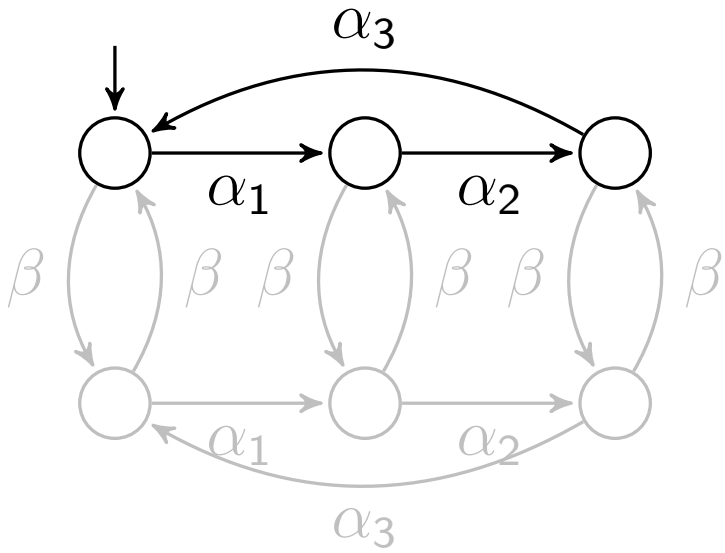
Possible Reduced State Space



POR work if and only if the property to check is stuttering invariant

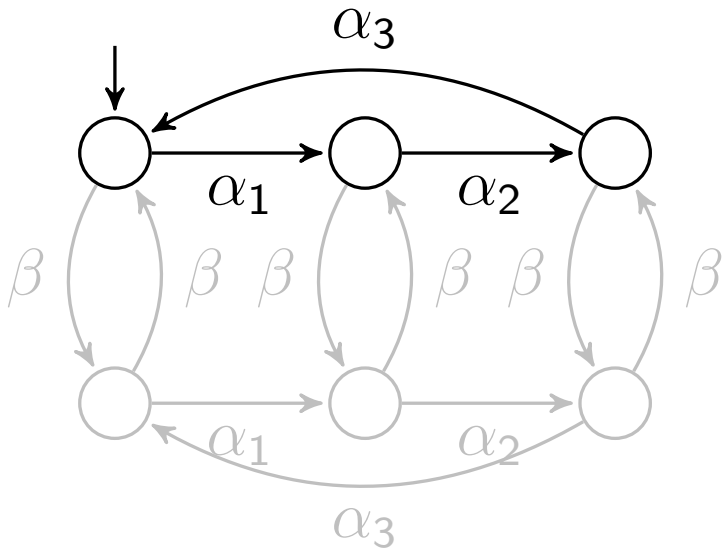
The Ignoring Problem for Liveness Properties

- 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)

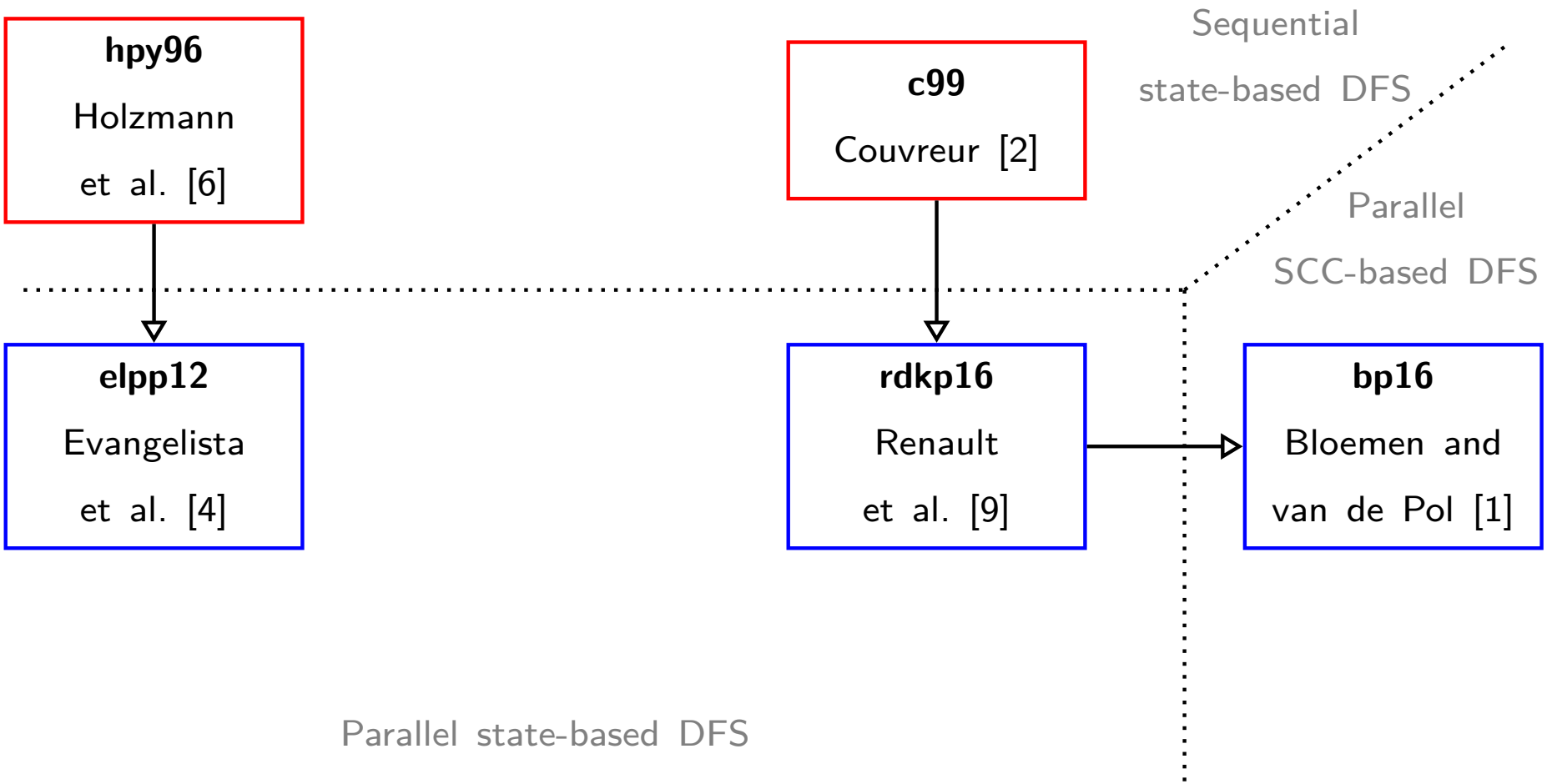


Requires an extra condition: the proviso

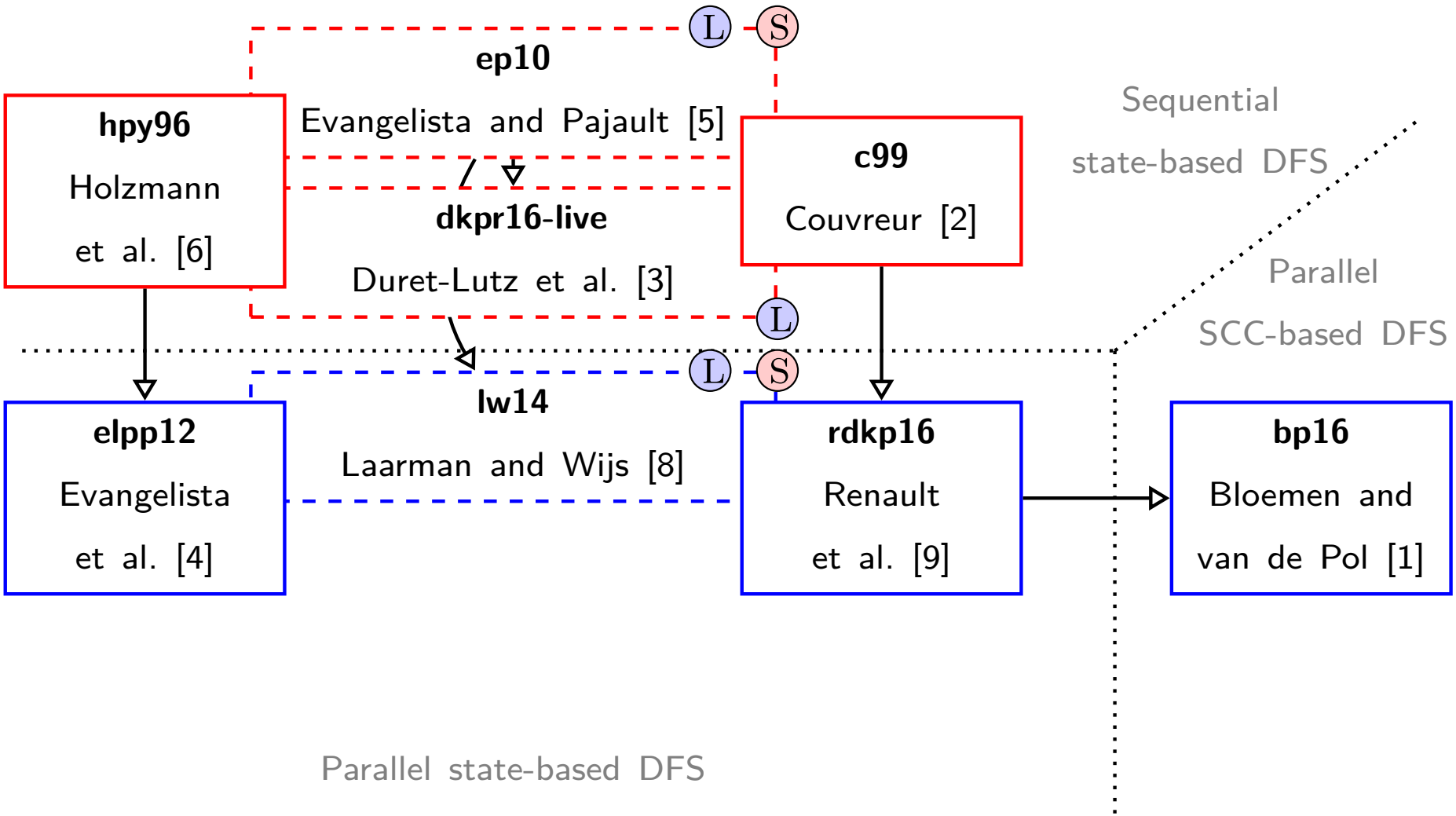
A proviso[a] ensures that every cycle in the reduced graph contains at least one **expanded state**, i.e, a state where all actions are considered.

[a] More simpler provisos can be applied for safety properties Evangelista and Pajault [5]

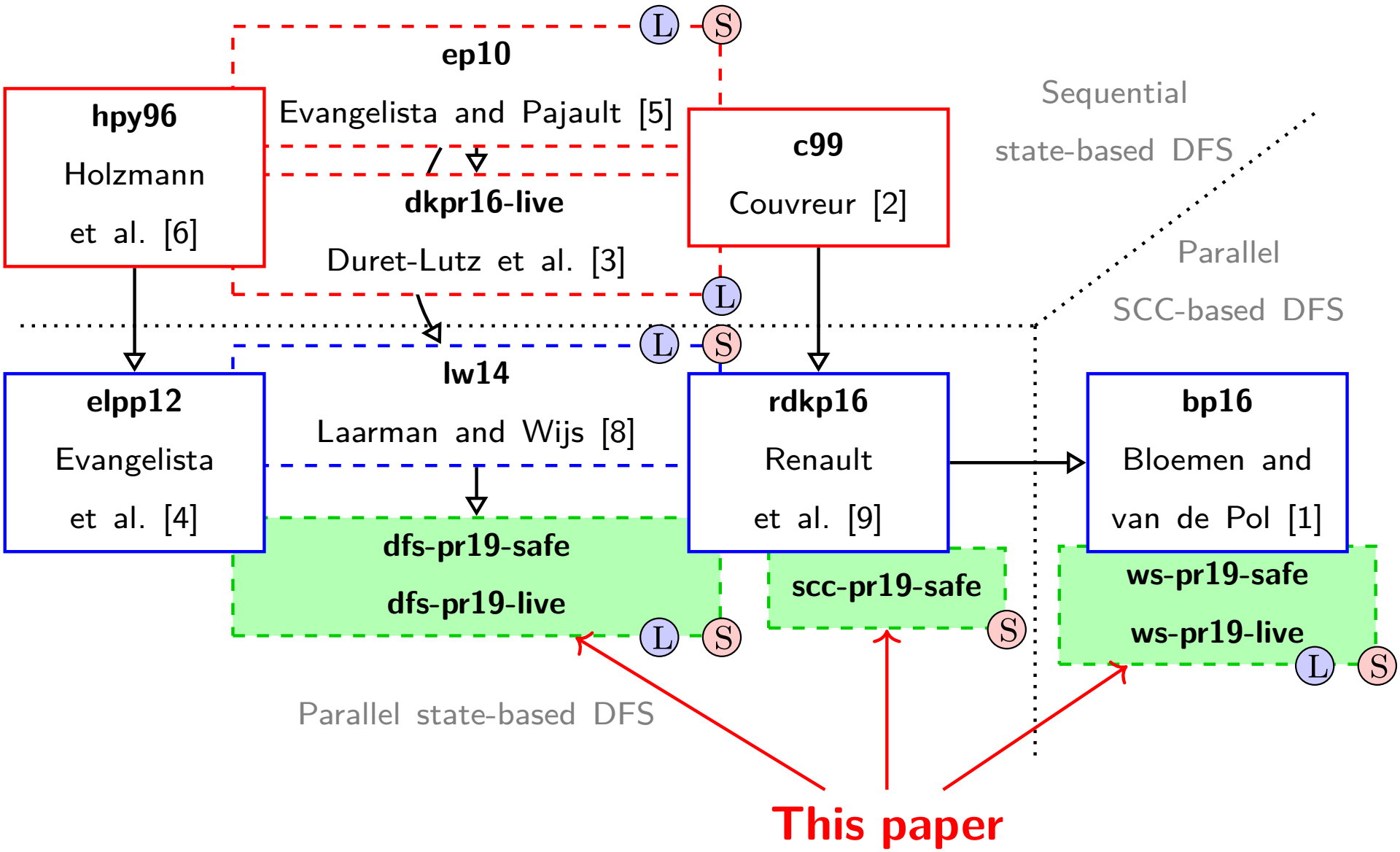
State-Of-The-Art: Emptiness Cheks & POR



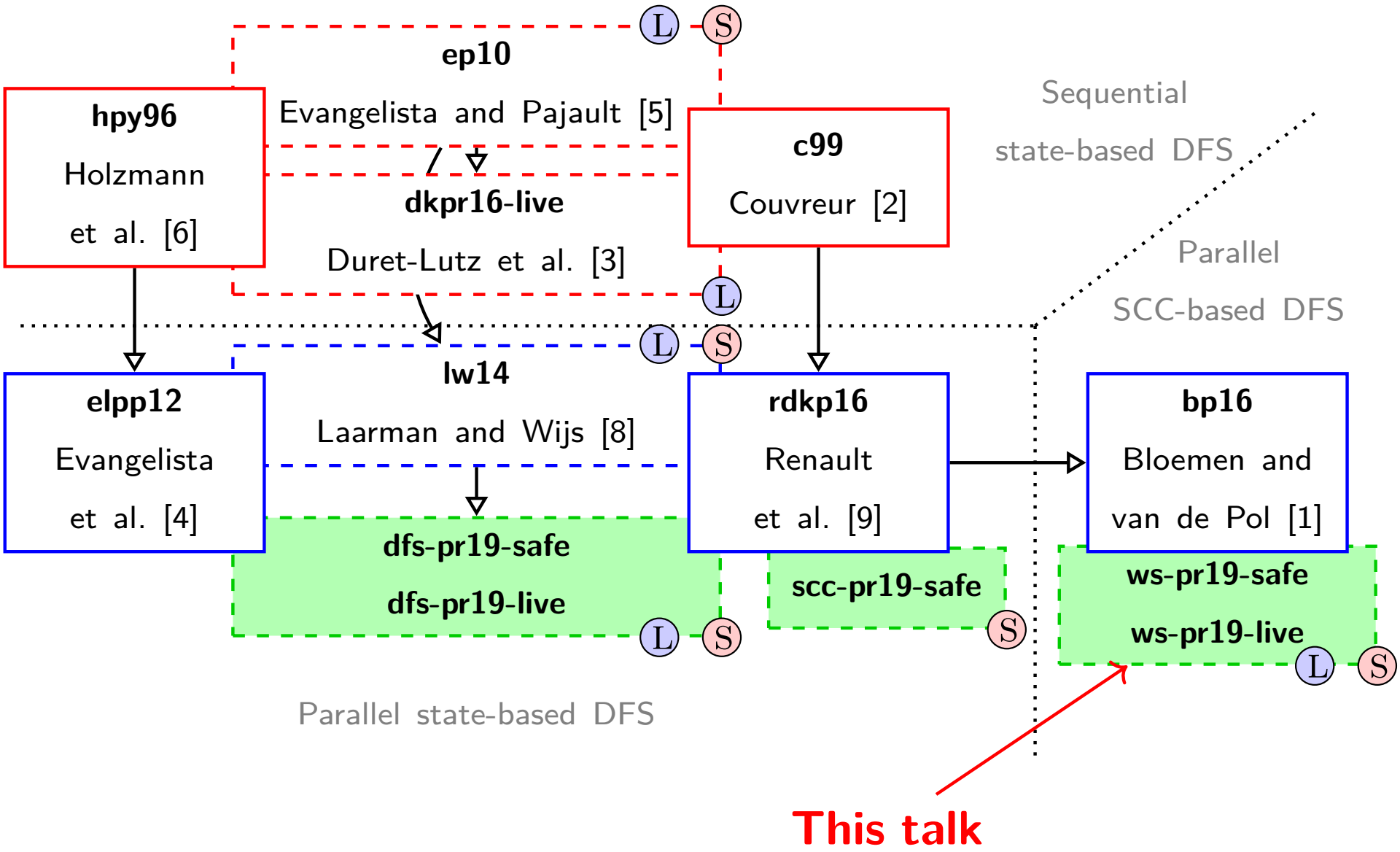
State-Of-The-Art: Emptiness Cheks & POR



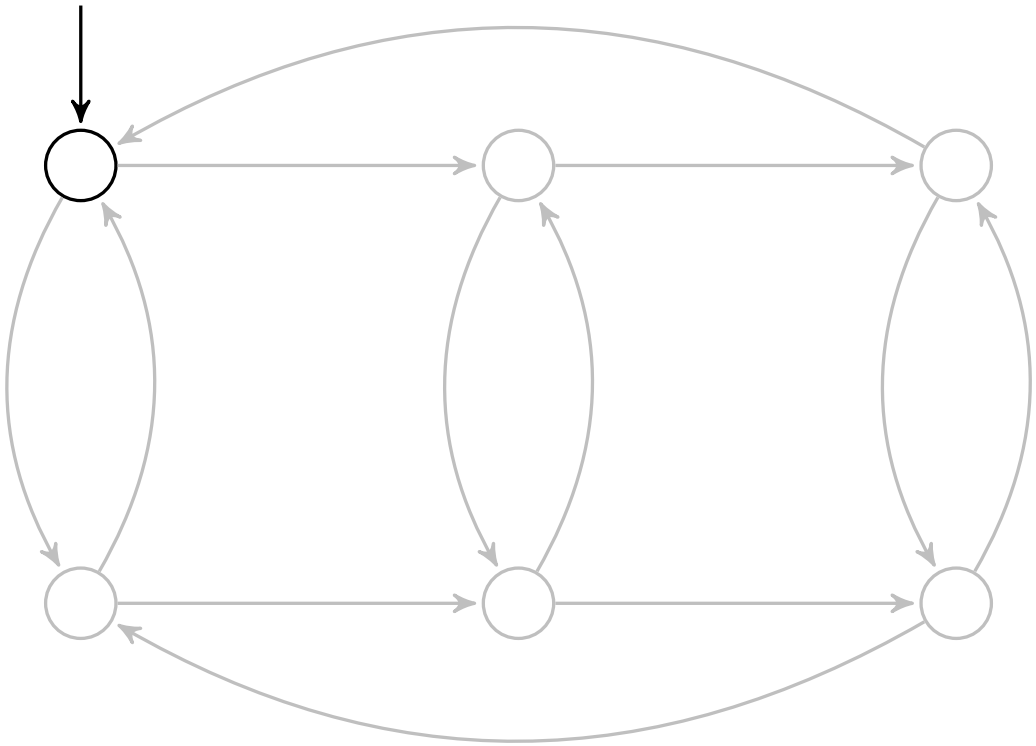
State-Of-The-Art: Emptiness Cheks & POR



State-Of-The-Art: Emptiness Cheks & POR

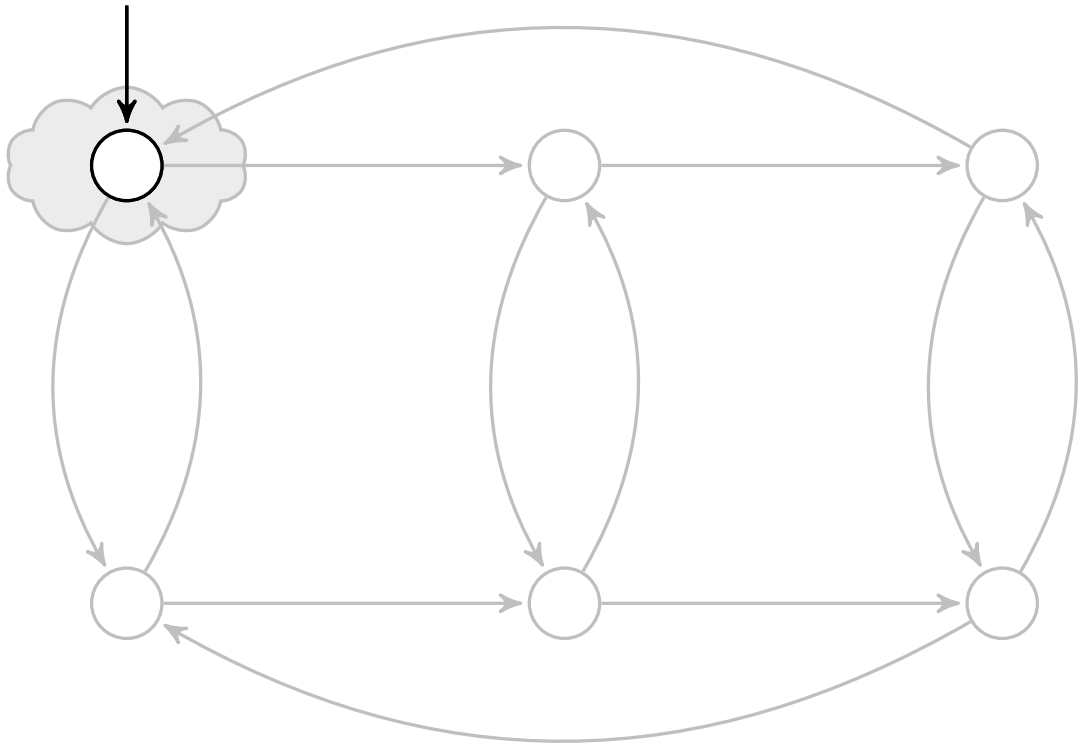


Bloemen's Emptiness Check explained



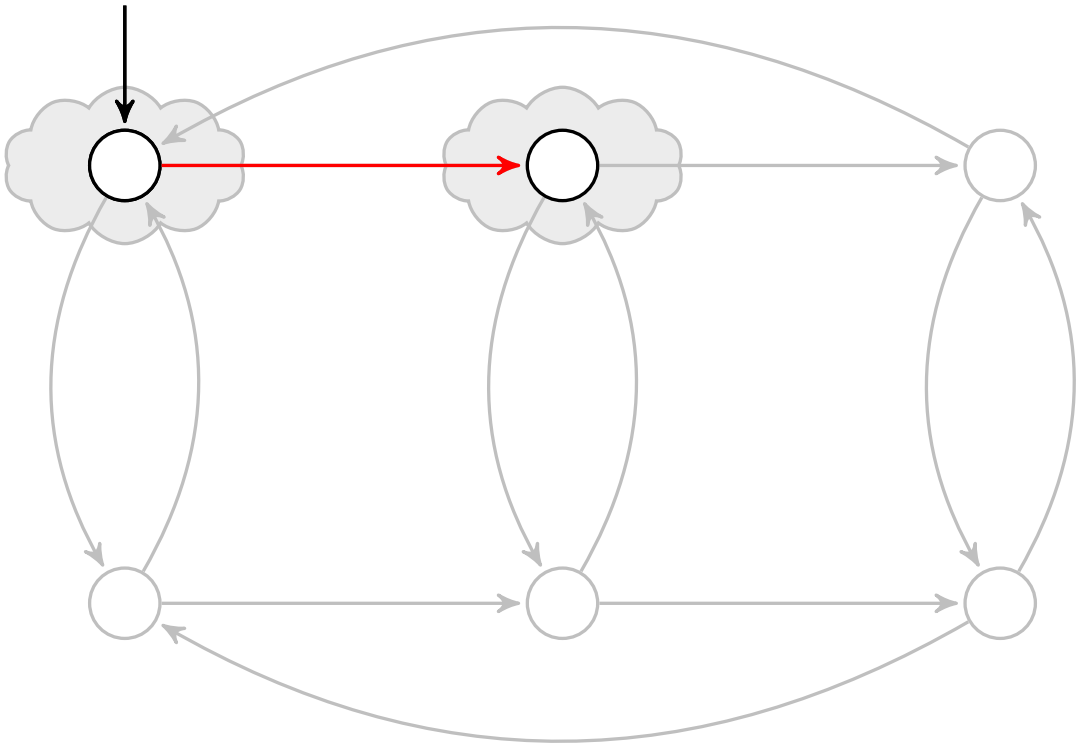
Start by the initial state (on-the-fly compatibility)

Bloemen's Emptiness Check explained



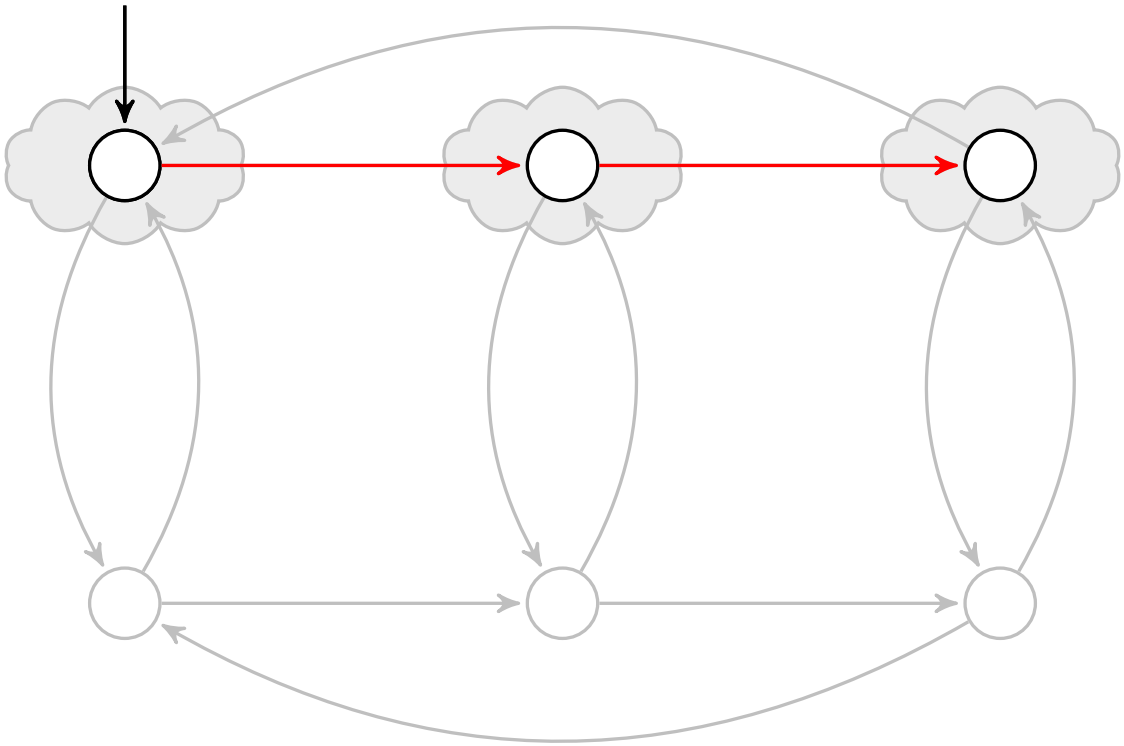
Clouds represent the actual knwoledge of SCCs

Bloemen's Emptiness Check explained



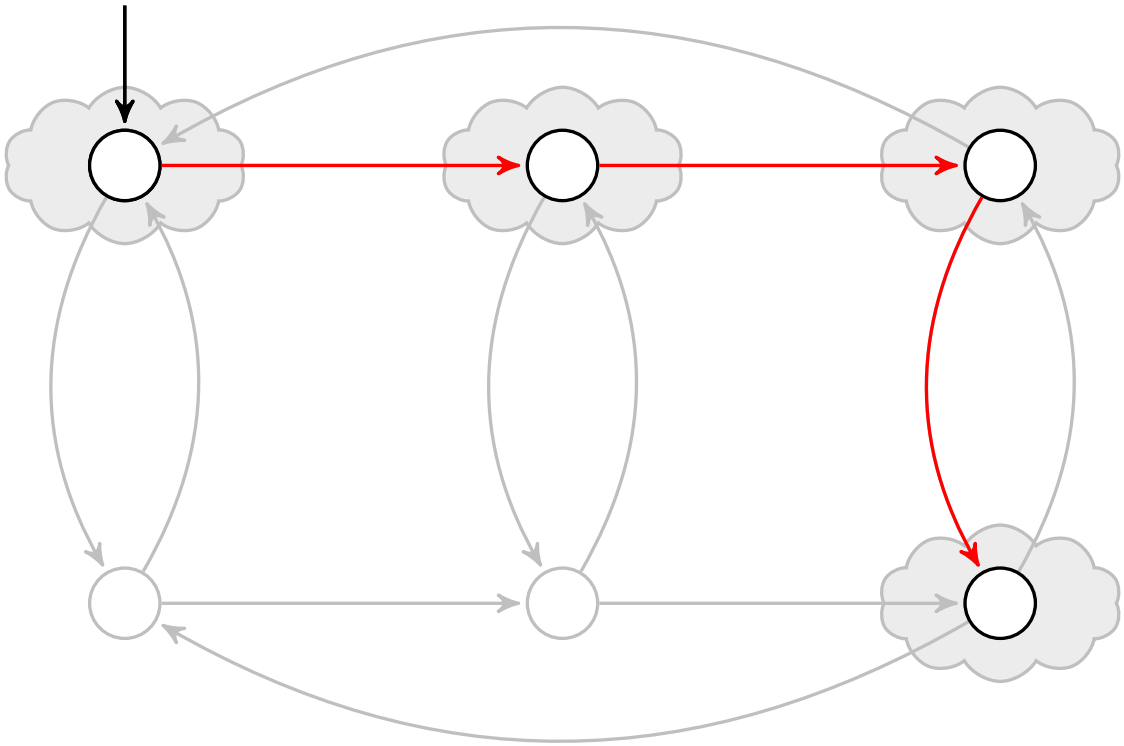
Red edges represent the DFS stack

Bloemen's Emptiness Check explained



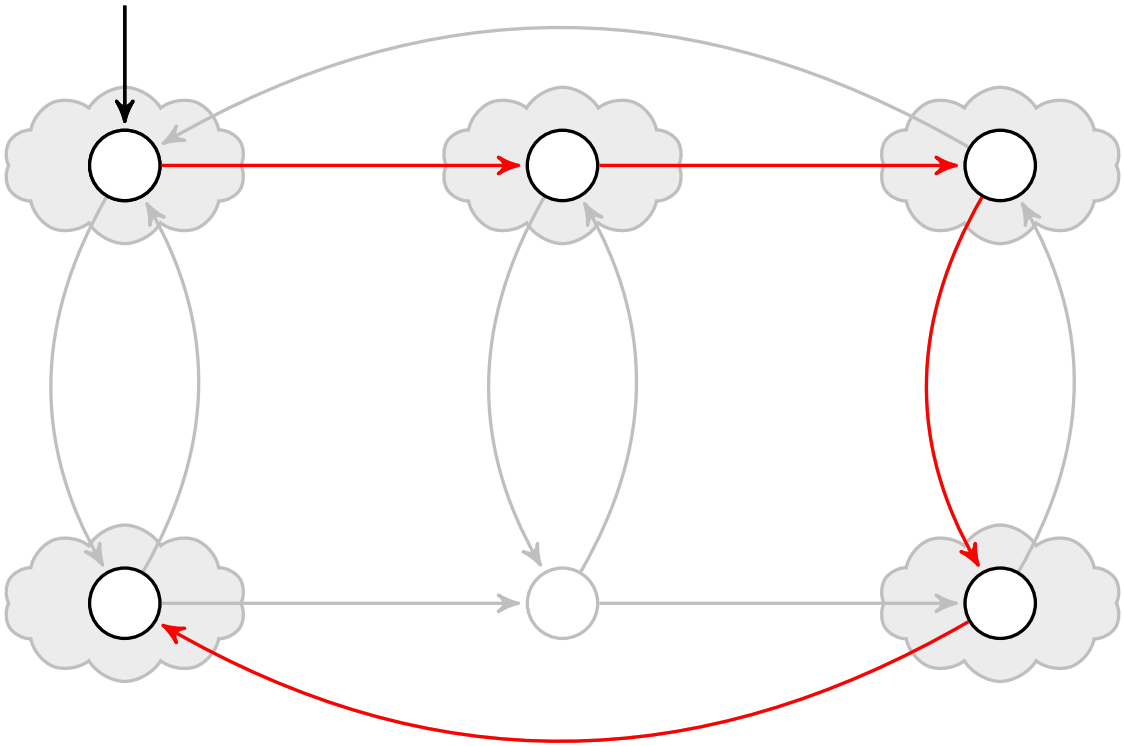
Red edges represent the DFS stack

Bloemen's Emptiness Check explained



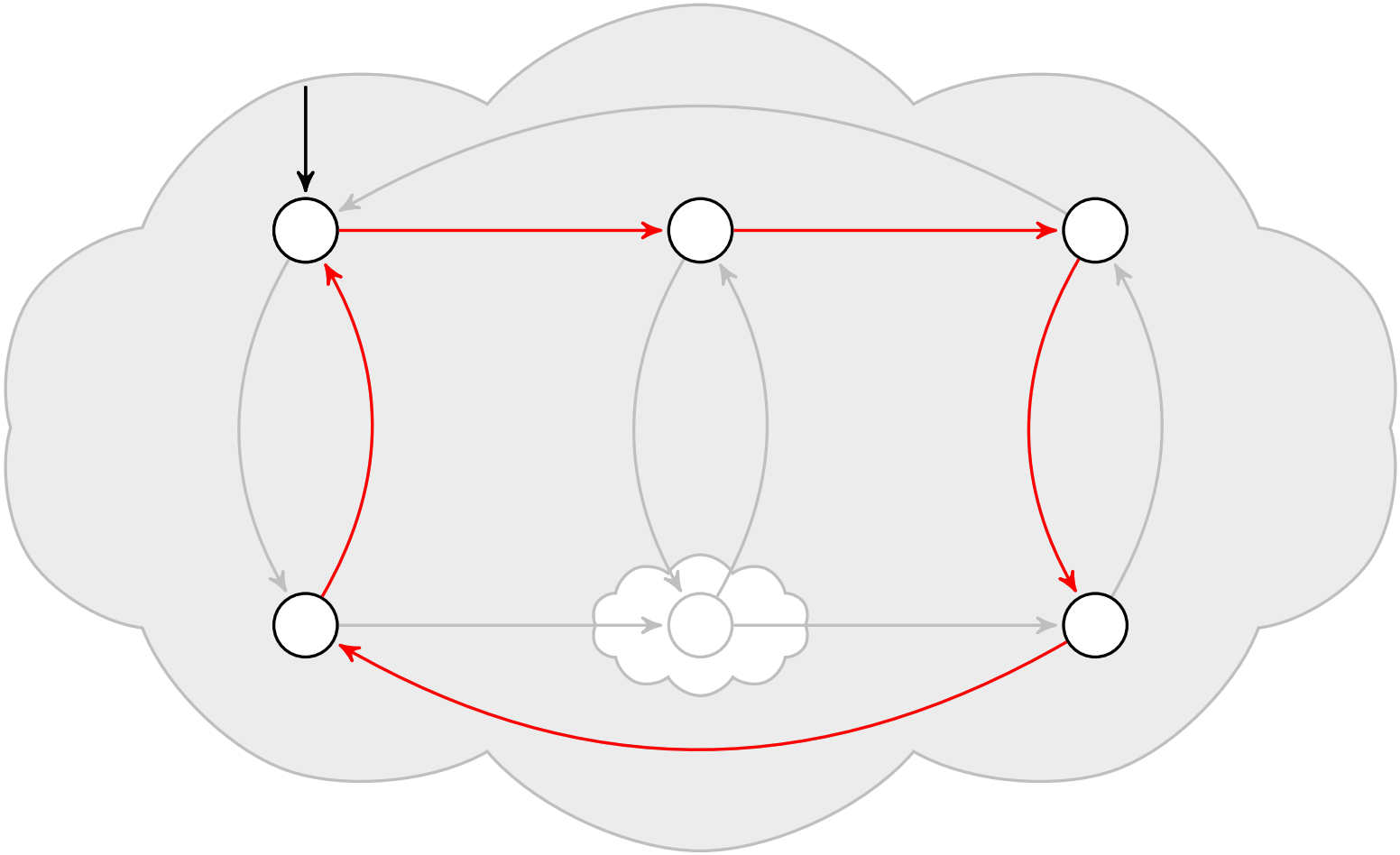
Red edges represent the DFS stack

Bloemen's Emptiness Check explained



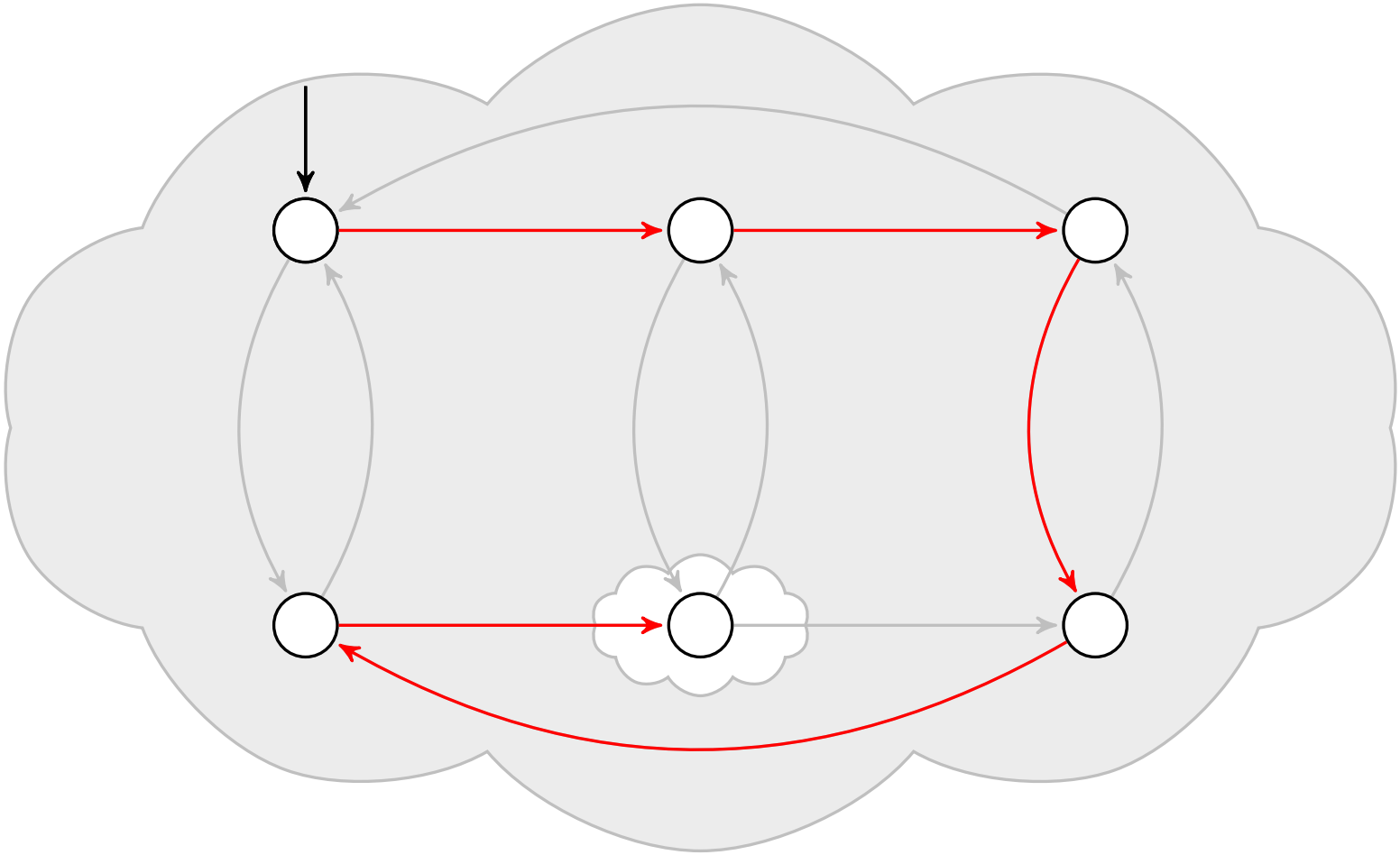
Red edges represent the DFS stack

Bloemen's Emptiness Check explained



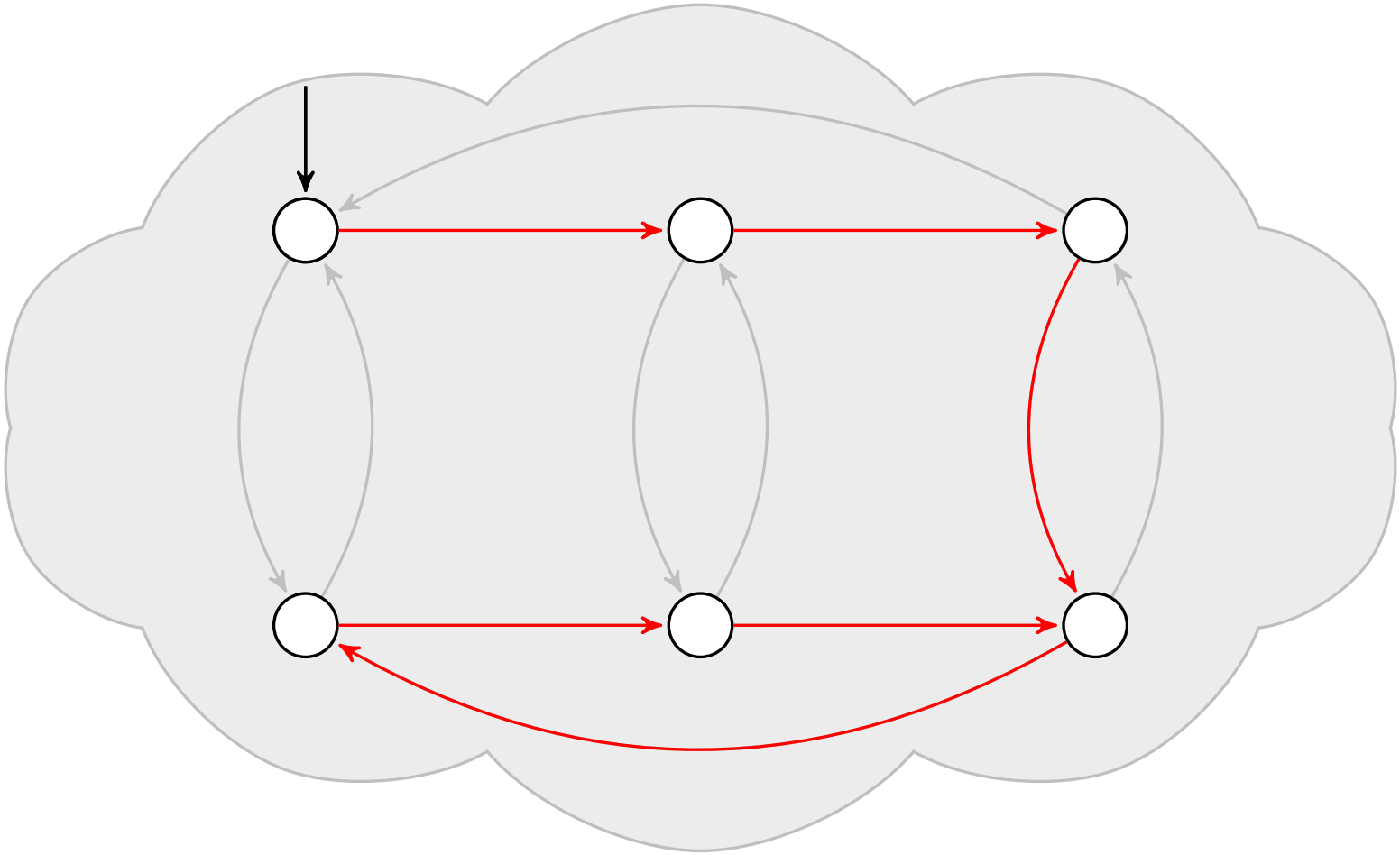
All (except one) states are discovered to belong to the same SCC

Bloemen's Emptiness Check explained



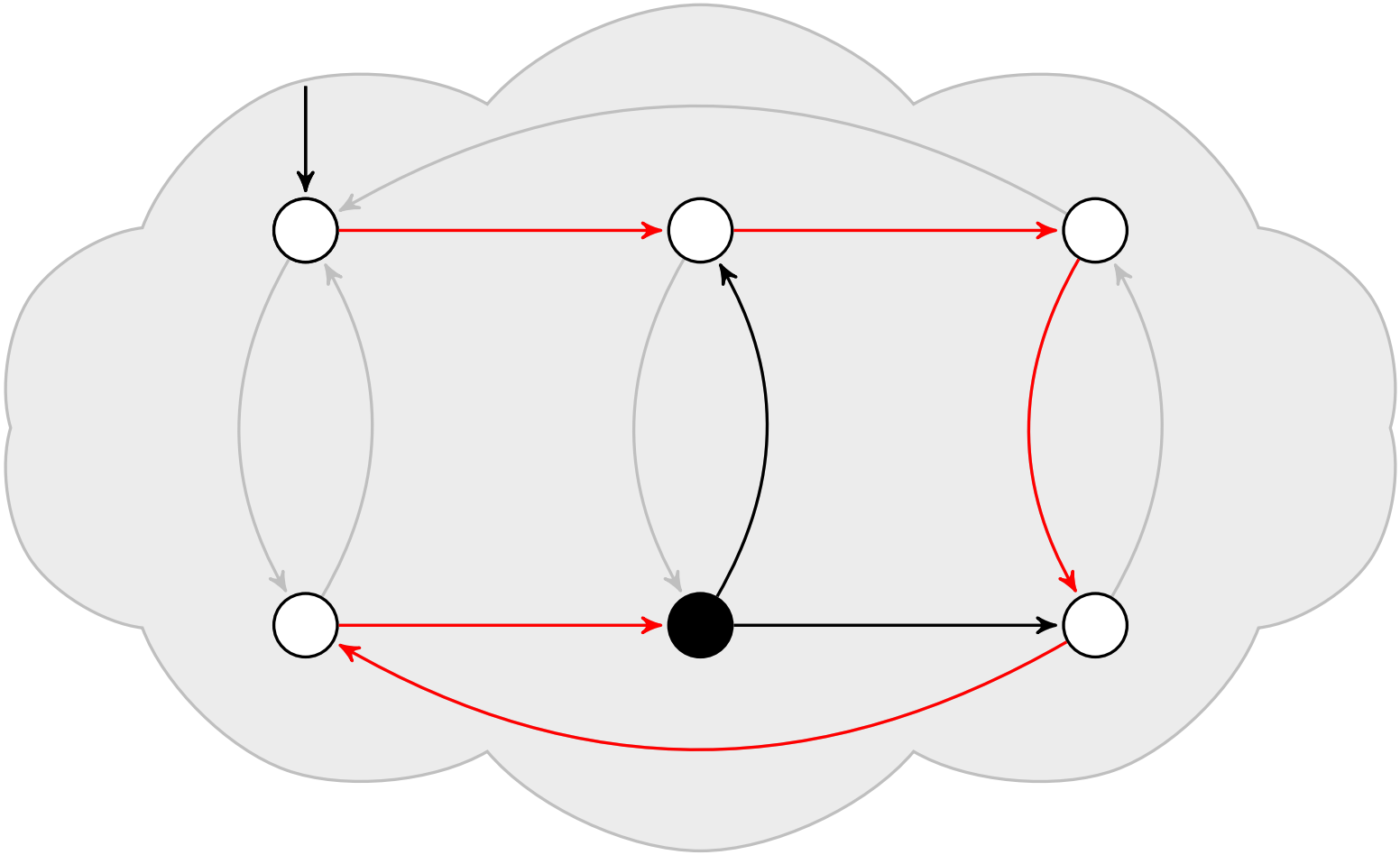
Continue DFS towards new successors

Bloemen's Emptiness Check explained



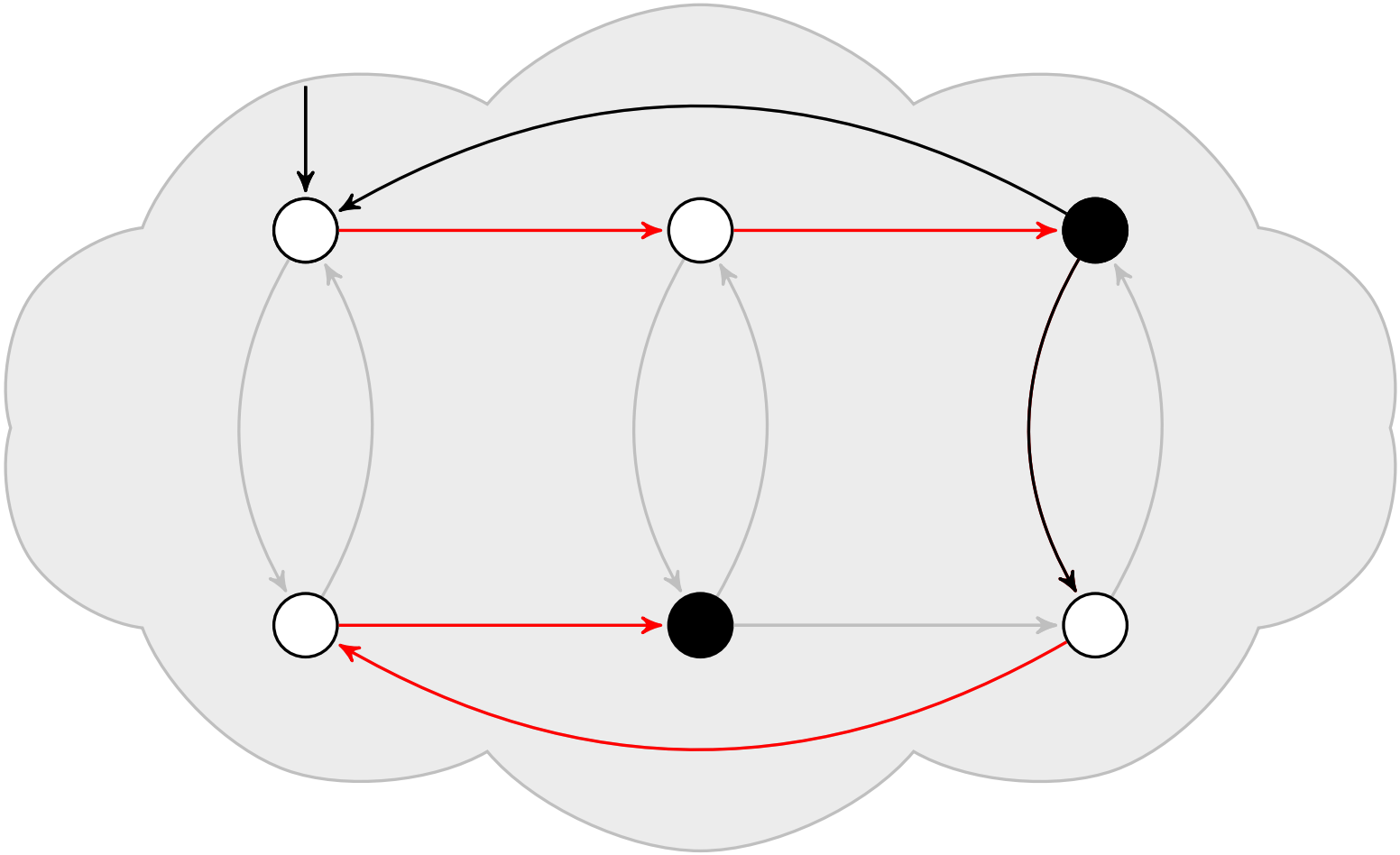
The state is detected to belong to the (only) SCC

Bloemen's Emptiness Check explained



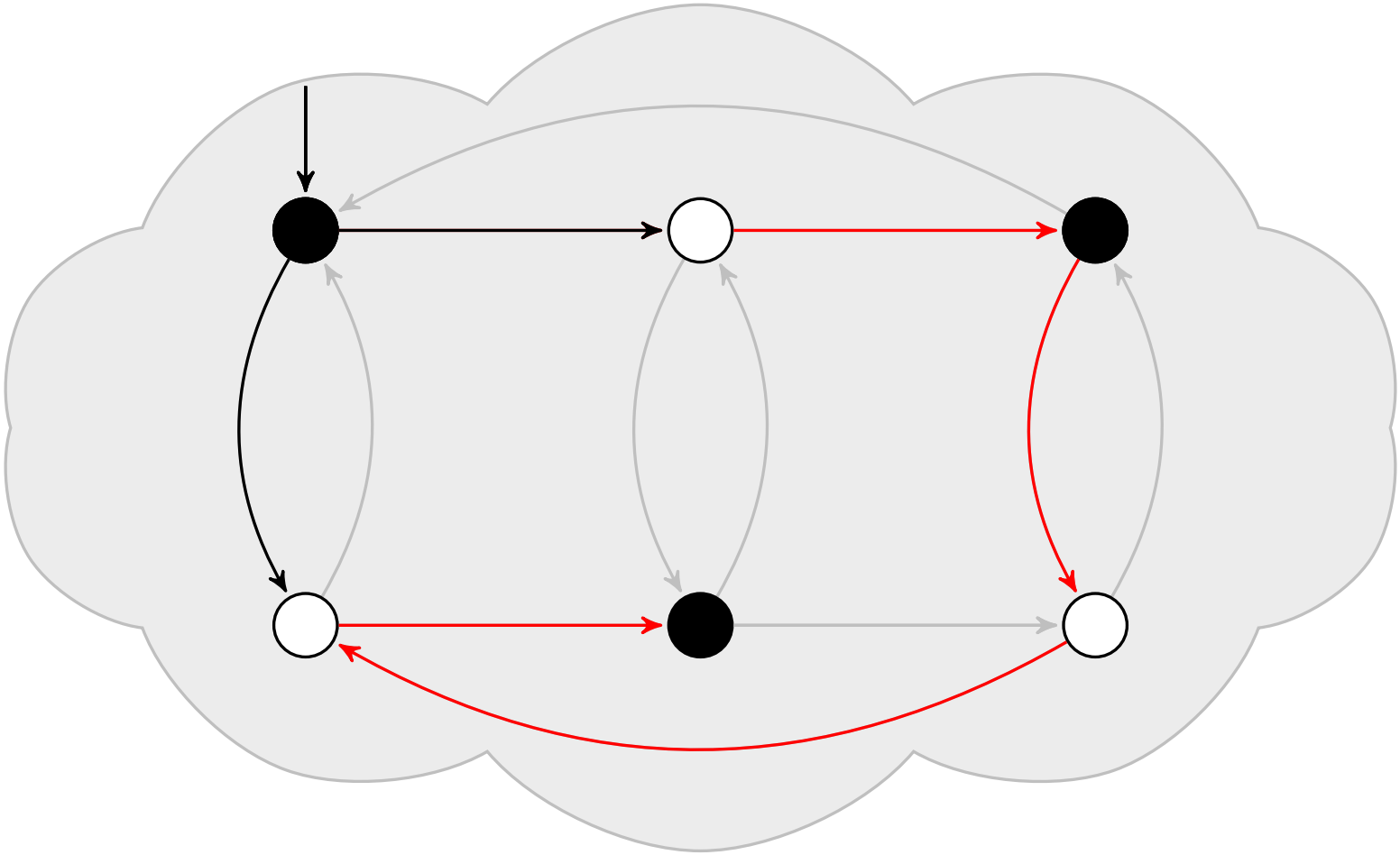
Its successors already belong to this SCC, the state is tagged DONE

Bloemen's Emptiness Check explained



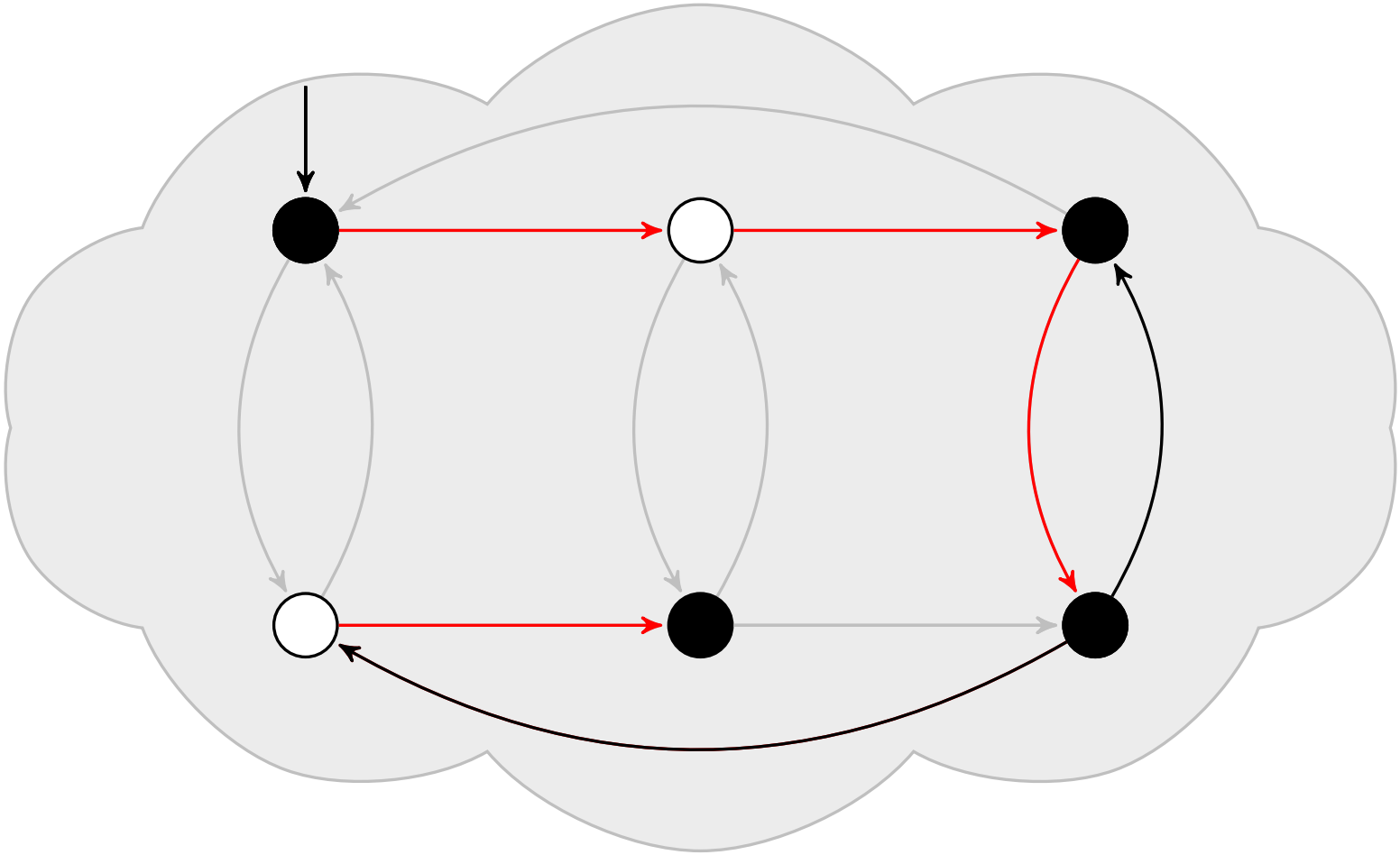
Pick states randomly and tag them DONE

Bloemen's Emptiness Check explained



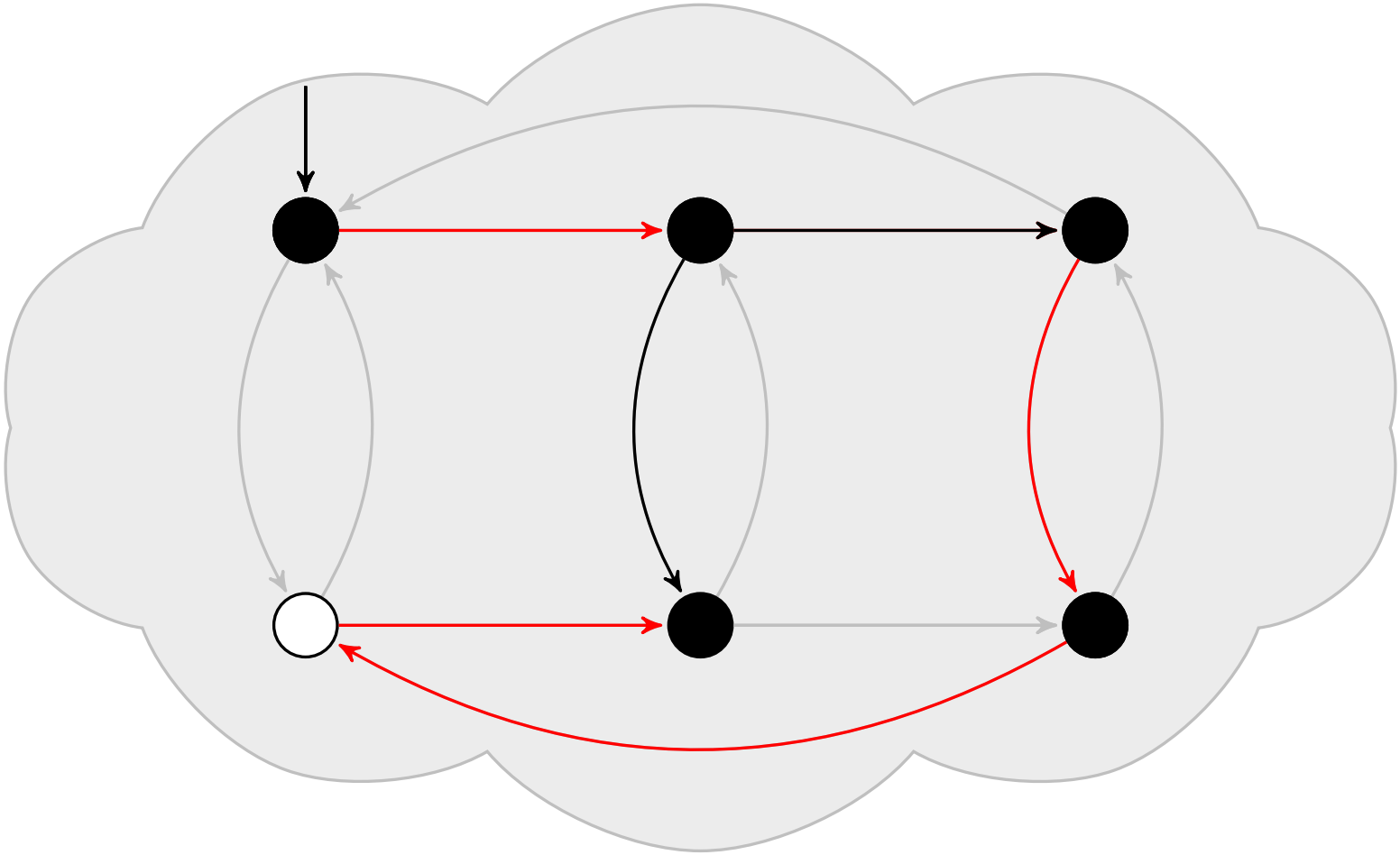
Pick states randomly and tag them DONE

Bloemen's Emptiness Check explained



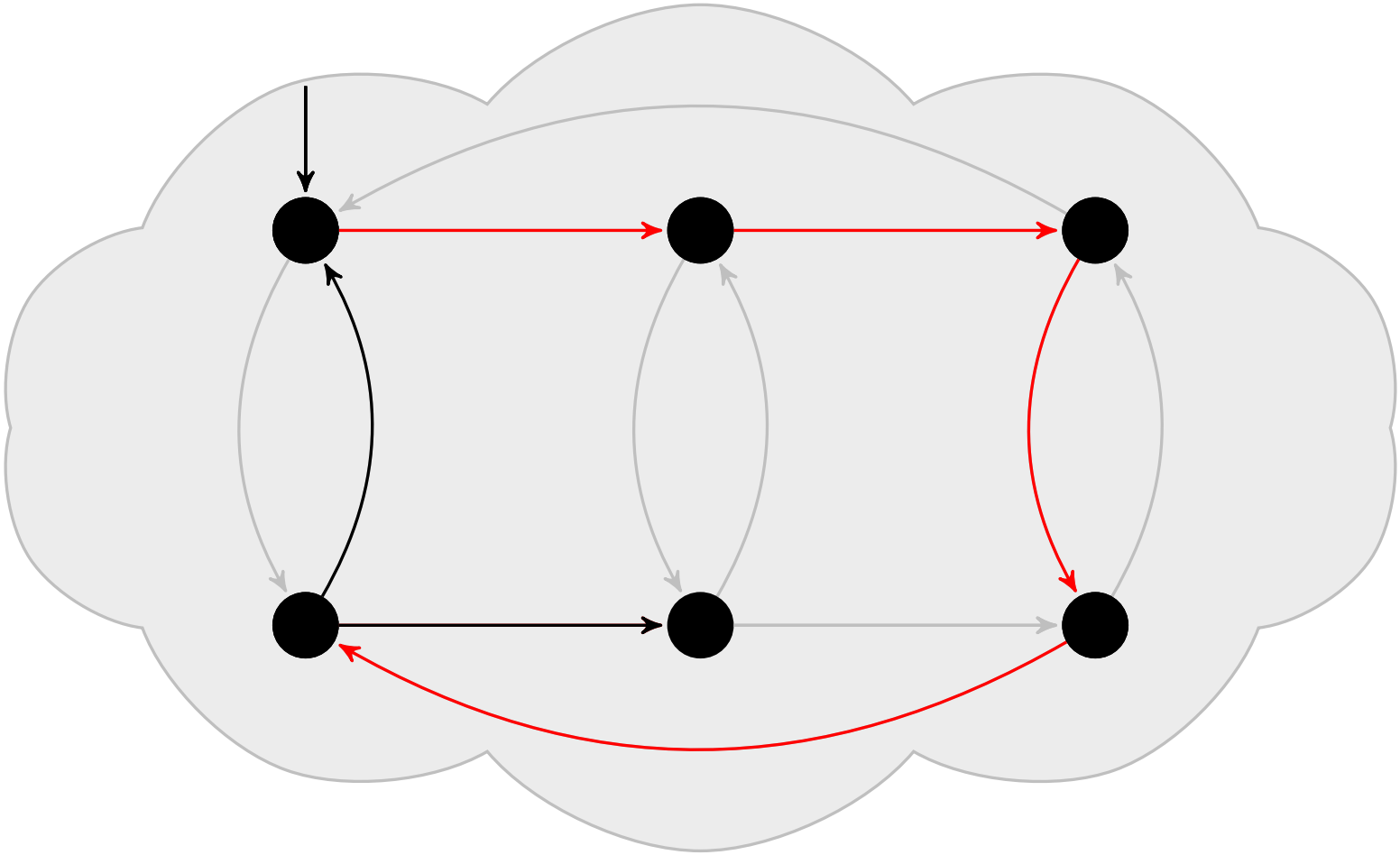
Pick states randomly and tag them DONE

Bloemen's Emptiness Check explained



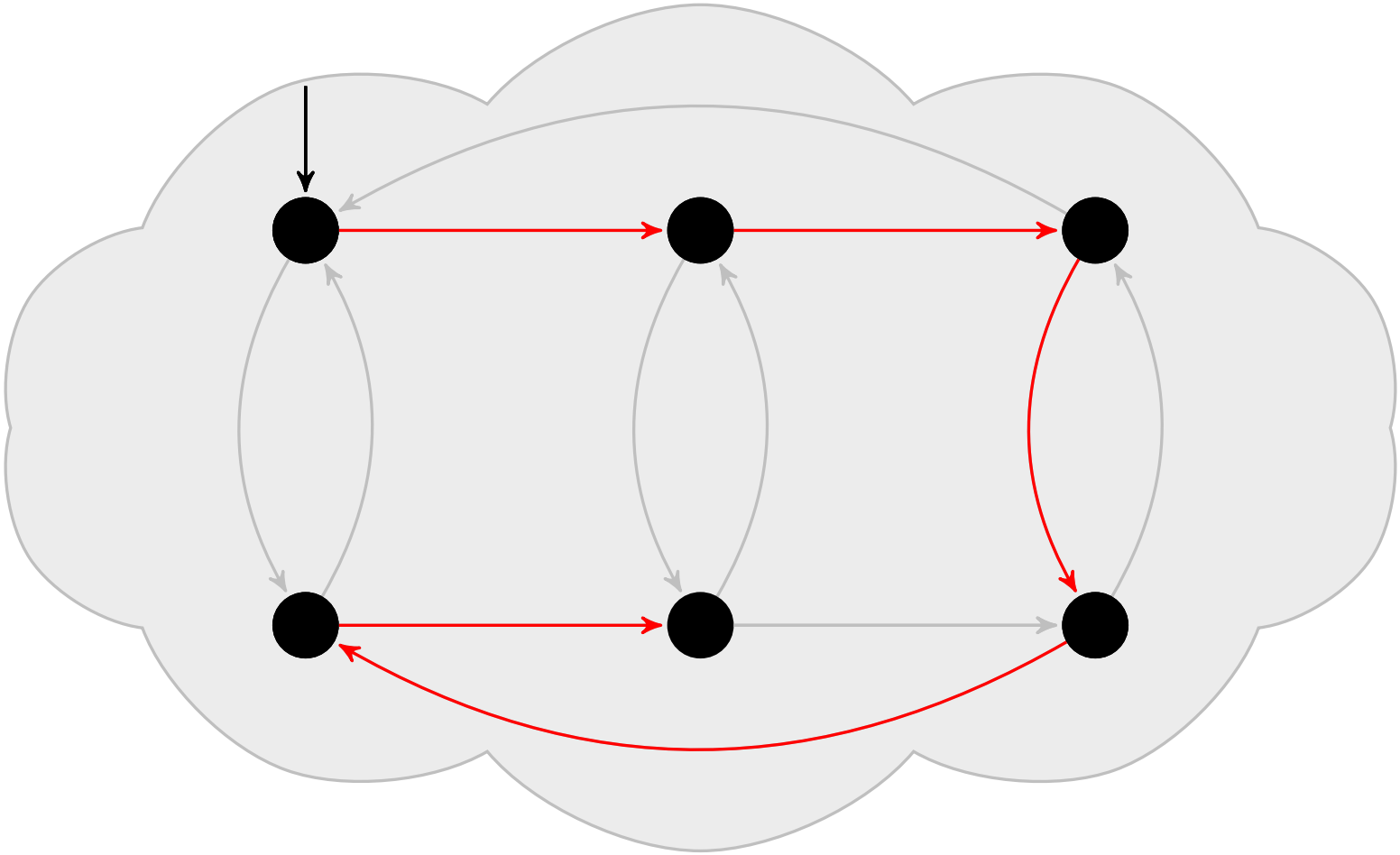
Pick states randomly and tag them DONE

Bloemen's Emptiness Check explained



Pick states randomly and tag them DONE

Bloemen's Emptiness Check explained



The SCC has been explored, backtrack!

Problematic

Problem's description

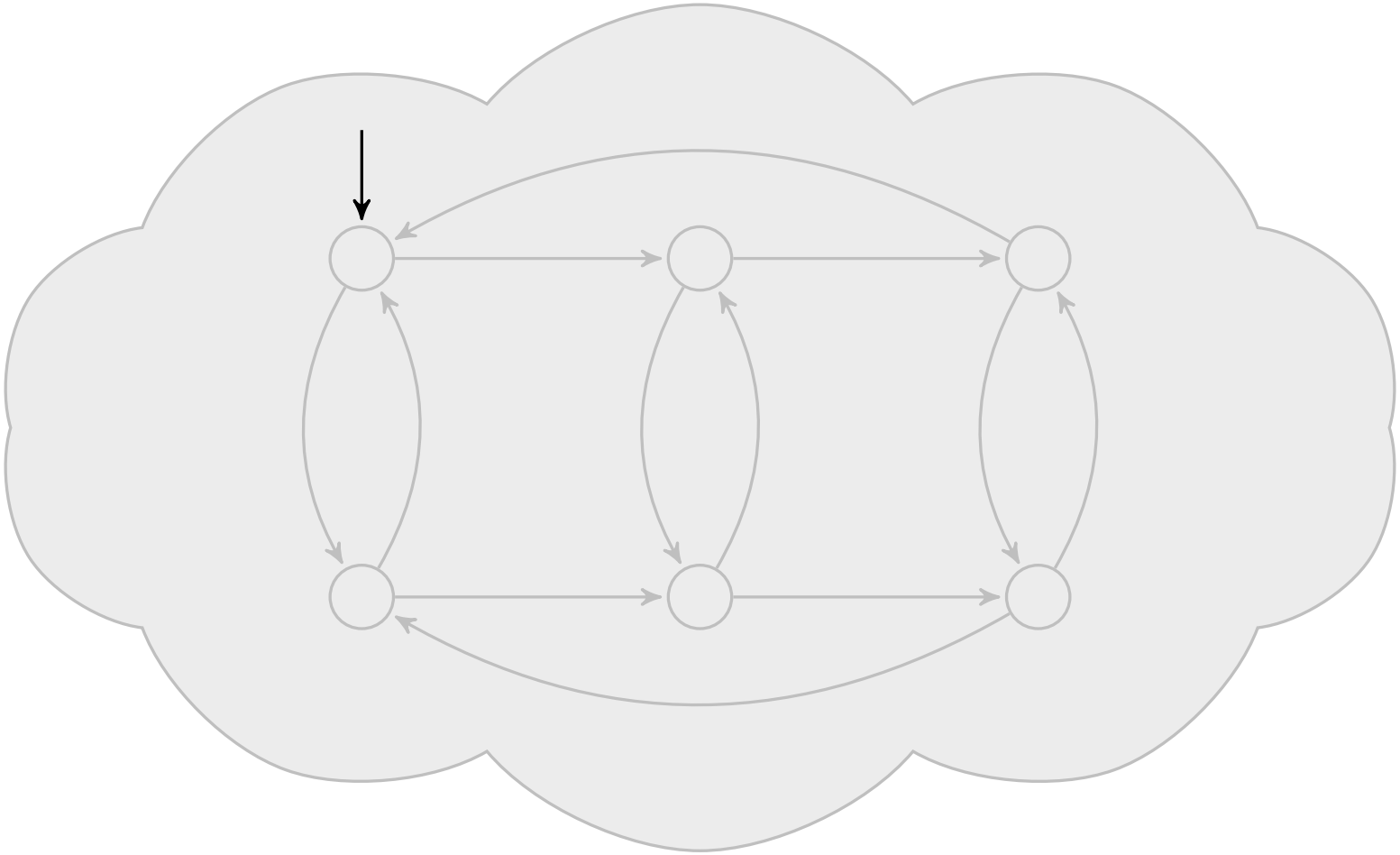
How to ensure that each cycle contains
(at least) one expanded cycle?

Rewording

Given a set of states (that belong to the same SCC), how can you decide wether an expansion is required only by considering the DONE status of its successors?

Sequential Solution: A Pessimistic Approach

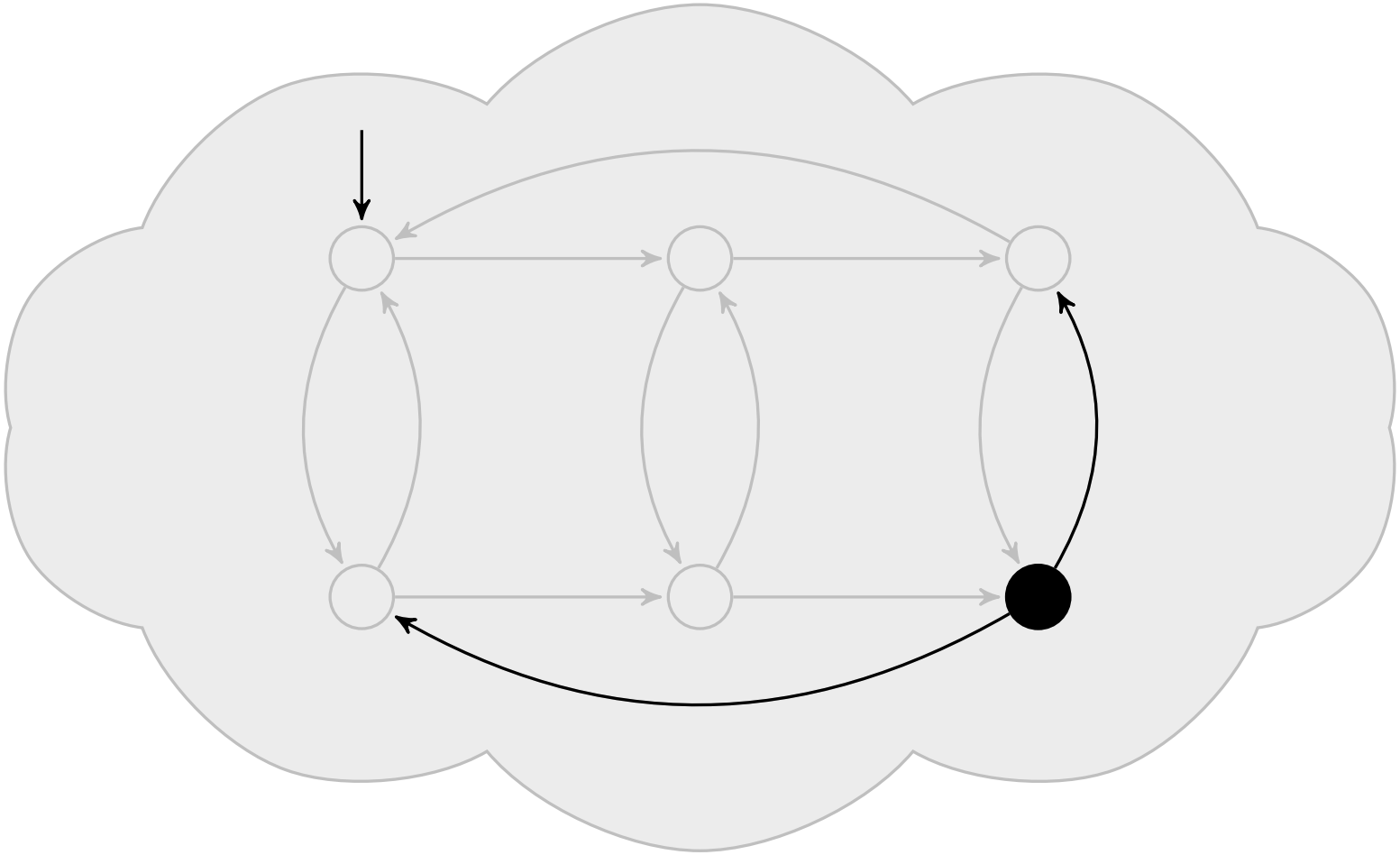
Idea: expand states with one successor DONE



Pick randomly one state and mark it DONE

Sequential Solution: A Pessimistic Approach

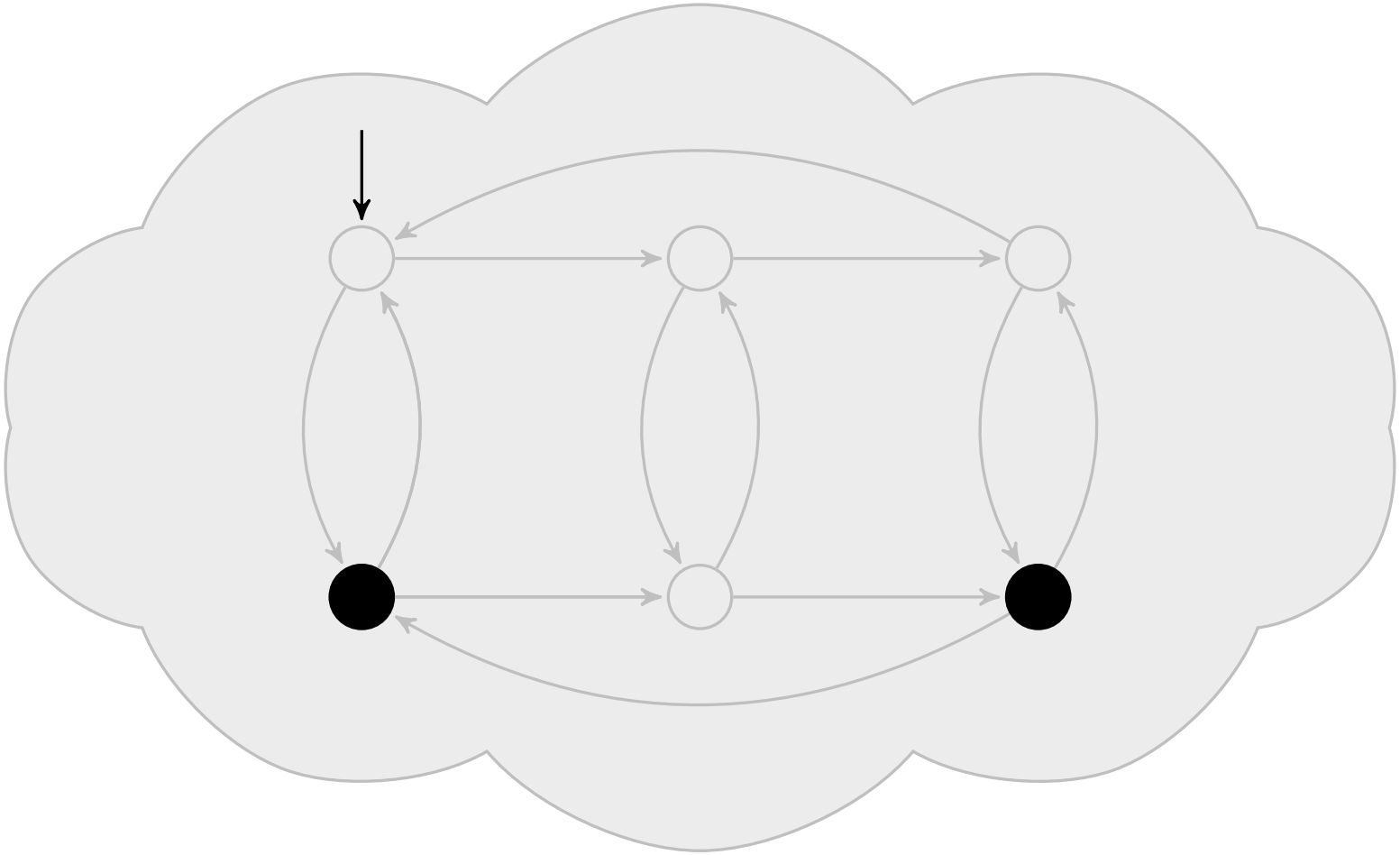
Idea: expand states with one successor DONE



Pick randomly one state and mark it DONE

Sequential Solution: A Pessimistic Approach

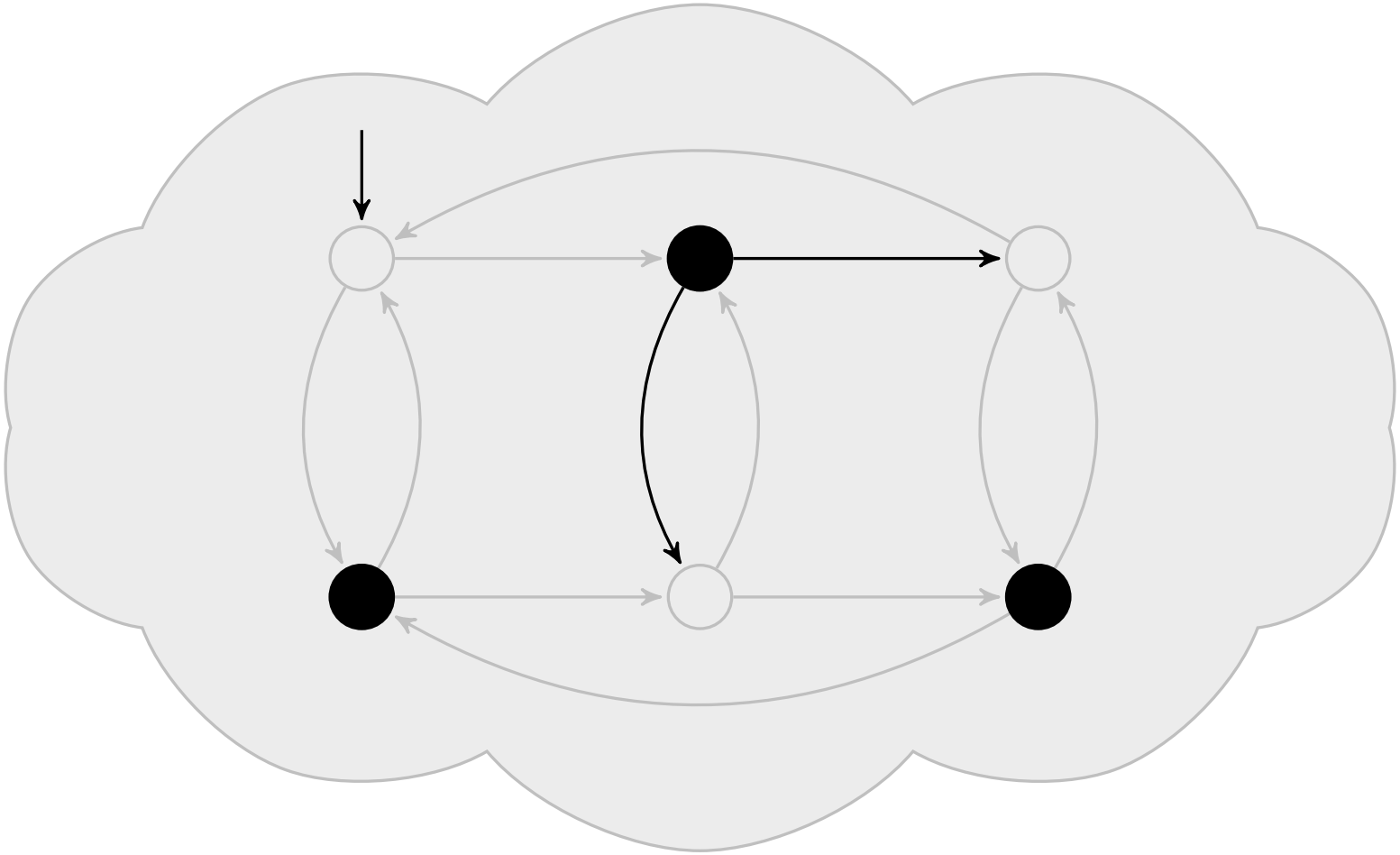
Idea: expand states with one successor DONE



Pick randomly one state and mark it DONE

Sequential Solution: A Pessimistic Approach

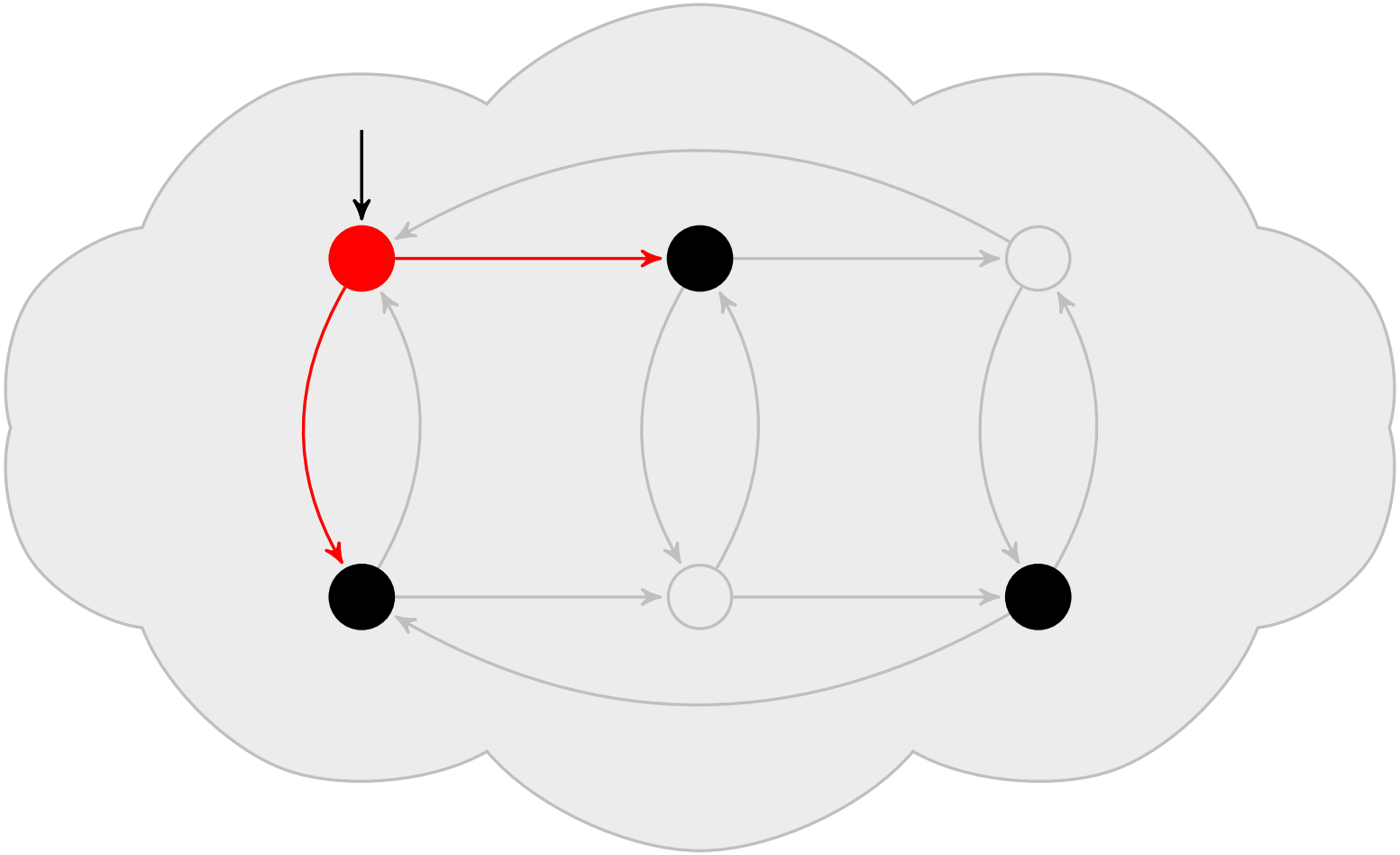
Idea: expand states with one successor DONE



Pick randomly one state and mark it DONE

Sequential Solution: A Pessimistic Approach

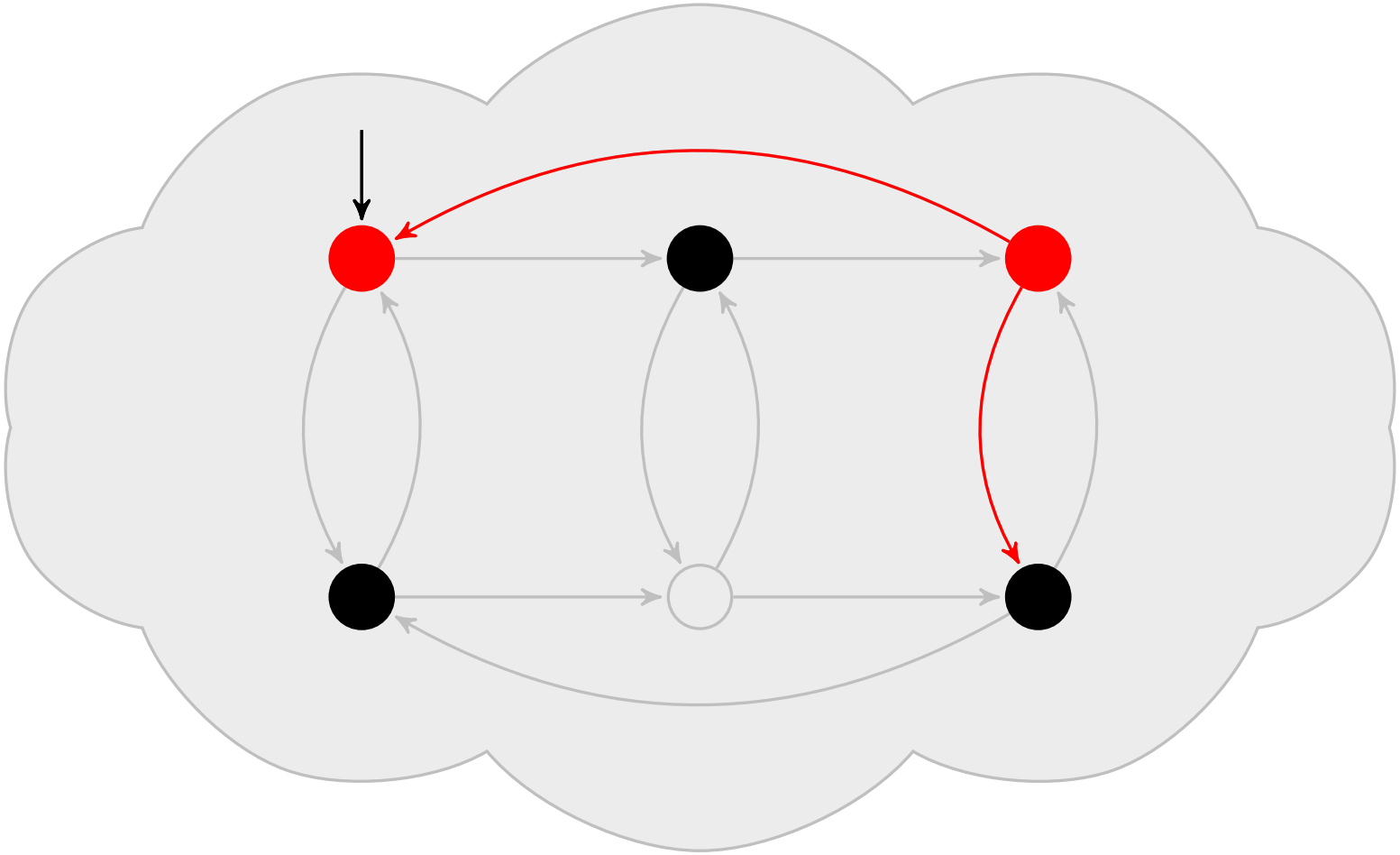
Idea: expand states with one successor DONE



Pick randomly one state and mark it DONE

Sequential Solution: A Pessimistic Approach

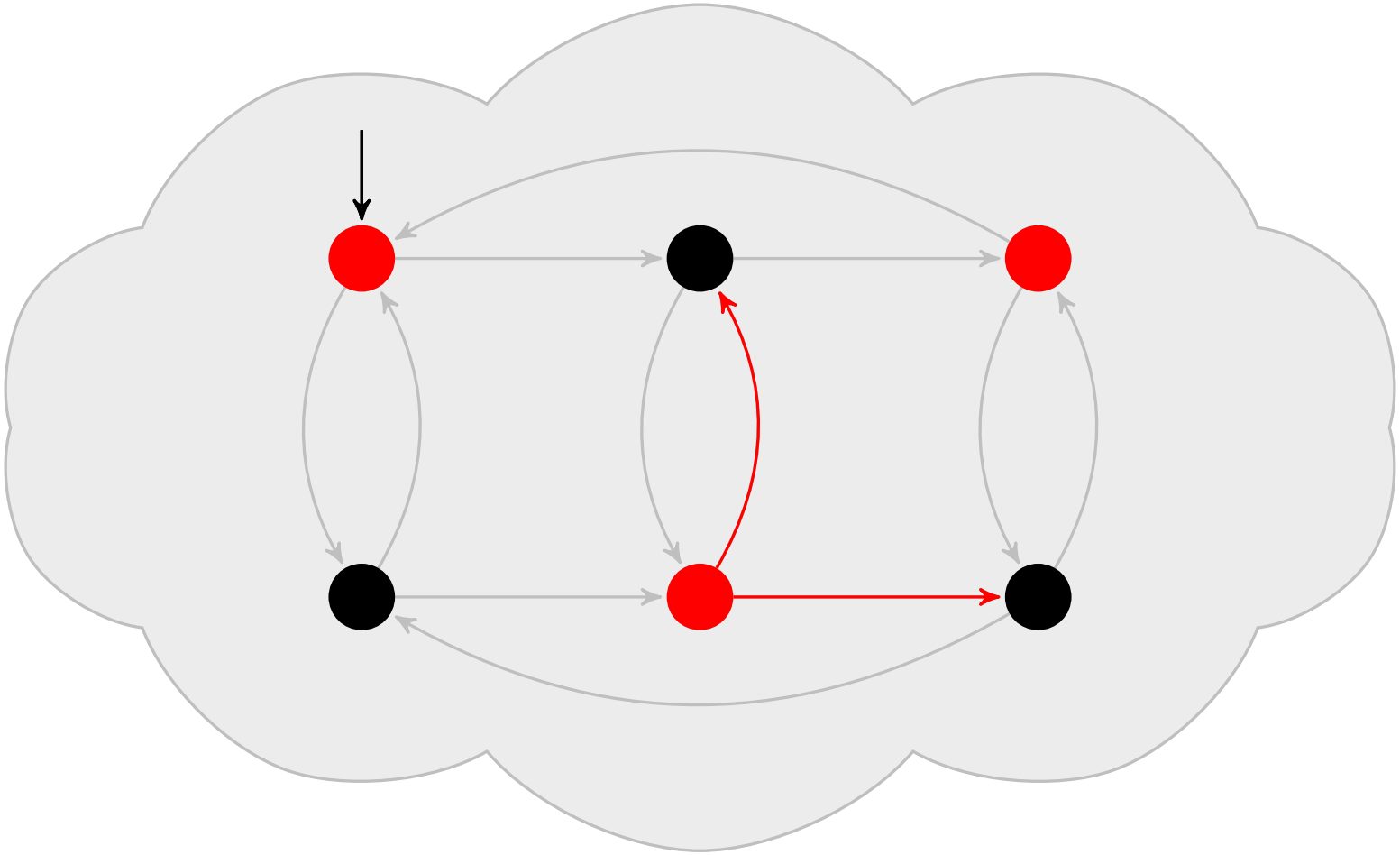
Idea: expand states with one successor DONE



State has one DONE successor: expand it!

Sequential Solution: A Pessimistic Approach

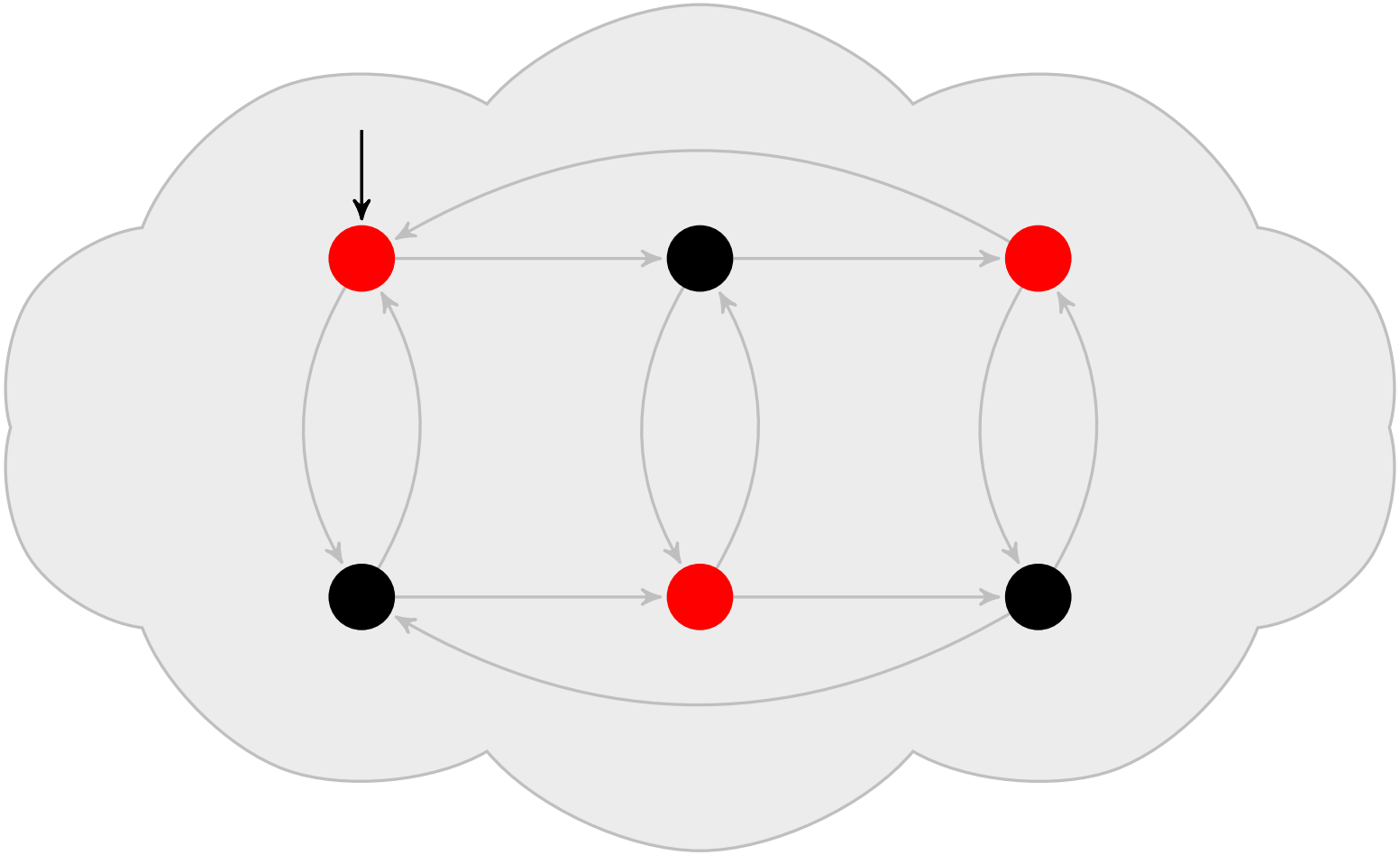
Idea: expand states with one successor DONE



State has one DONE successor: expand it!

Sequential Solution: A Pessimistic Approach

Idea: expand states with one successor DONE

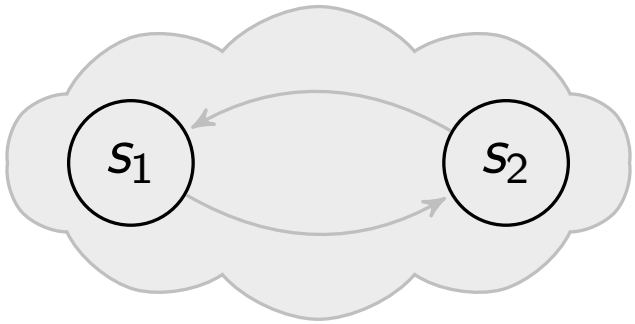


State has one DONE successor: expand it!

Parallelisation: Problem Statement

This idea does not work in parallel

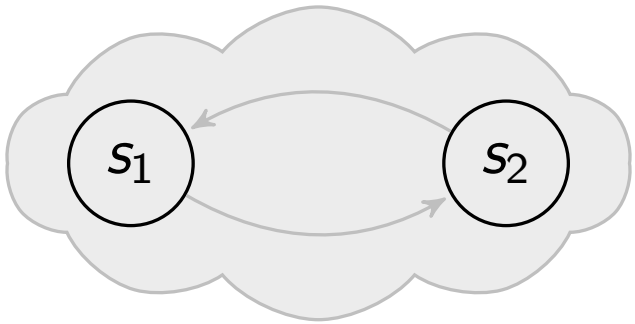
- s_1 and s_2 are known to be in the same SCC
- Thread 1 process s_1
- Thread 2 process s_2
- Thread 1 checks s_2: not expanded
- Thread 2 checks s_1: not expanded
- Both s_1 and s_2 are tagged DONE



Parallelisation: Pessimistic Solution

A state currently processed is tagged WIP. If a state s has a successor tagged WIP, s is expanded.

- s_1 is tagged WIP
- s_2 is tagged WIP
- Thread 1 check successors: s_2 is WIP, state will be expanded
- Thread 2 check successors: s_1 is WIP, state will be expanded



Others Results and Remarks

The WS-PR19-LIVE presented above is a pessimistic approach:

- Sequential: $N/2$ expansions in average, for an SCC of size N
- Parallel: ?

The paper also suggests:

- An adaptation of Bloemen's algorithm for safety WS-PR19-SAFE
- An adaptation of other parallel emptiness check with provisos for safety and liveness: DFS-PR19-SAFE, DFS-PR19-LIVE, and SCC-PR19-SAFE

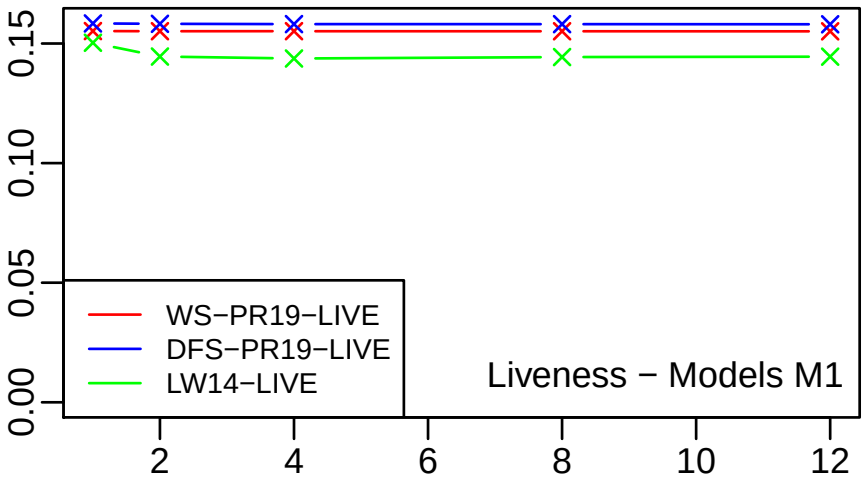
All the approaches presented in this paper are:

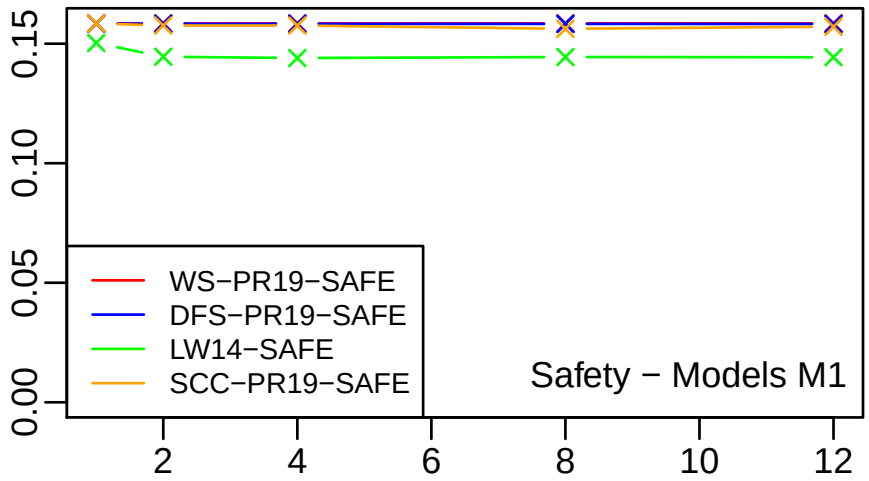
- Compatible with *Persistent sets*, *Stubborn set* and *Ample set*
- **Compatible with on-the-fly exploration technique**

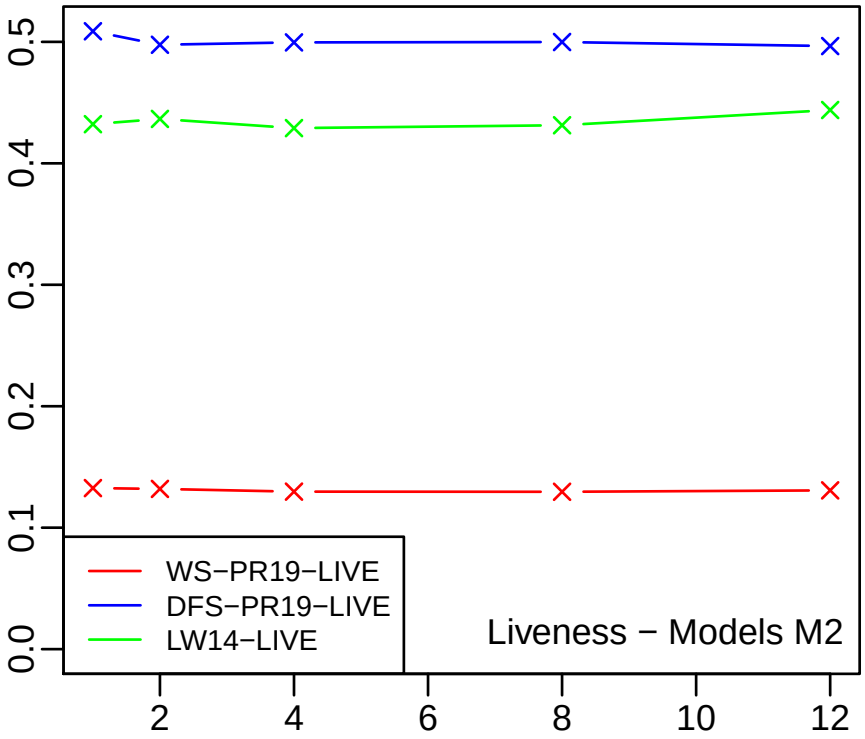
Evaluation

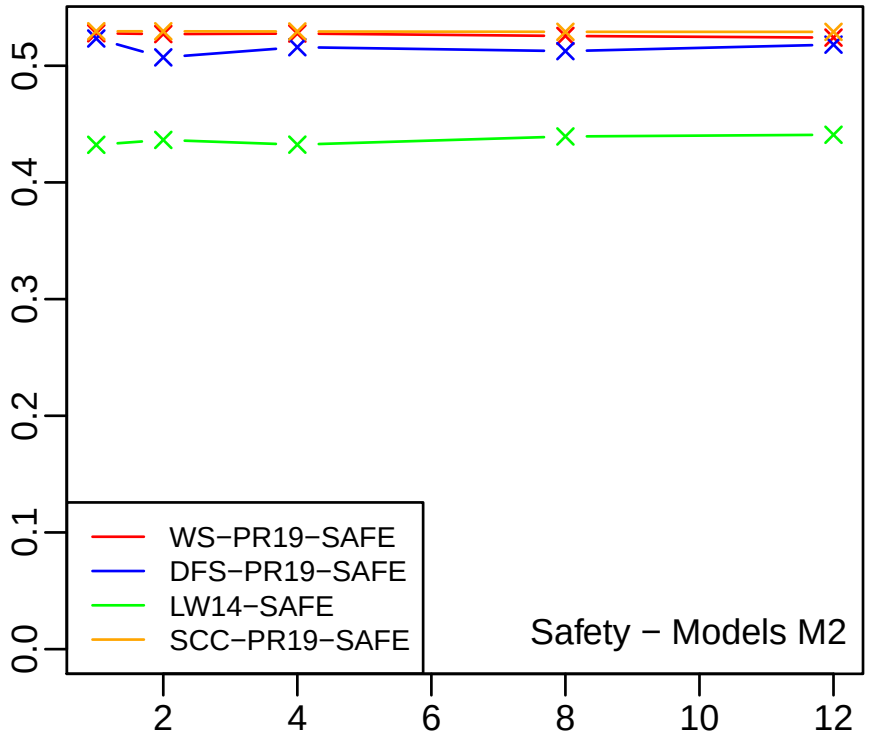
- 21 models from the BEEM benchmark divided into two categories:
  - $\mathcal{M}_1$: models with short cycles and many small SCCs
  - $\mathcal{M}_2$: models with long cycles and a small number of large SCCs
- Reductions are implemented by the way of the stubborn-set method from Valmari
- Maximum running time 40 minutes (in sequential)
- Up to 12 threads (the maximum we can test)
- Compare the 5 algorithms we propose against state of the art algorithm (**lw14**)

Reduction

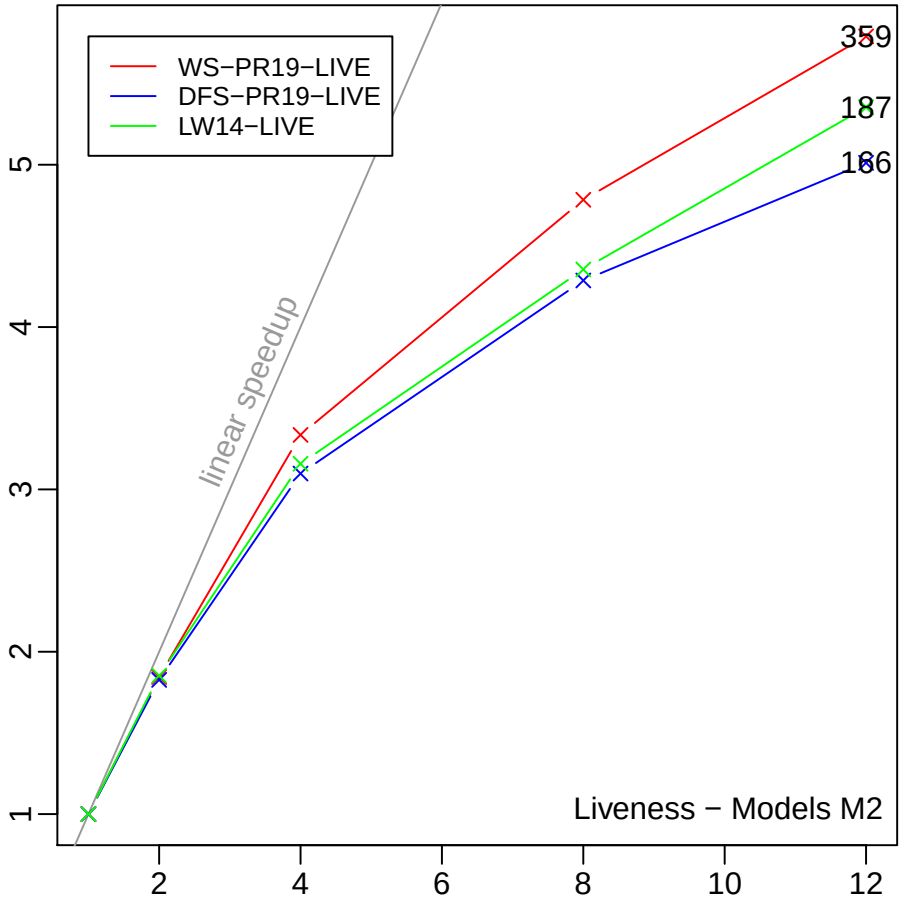

0.00
0.05
0.10
0.15
2
4
6
8
10
12
WS−PR19−LIVE
DFS−PR19−LIVE
LW14−LIVE
Liveness − Models M1

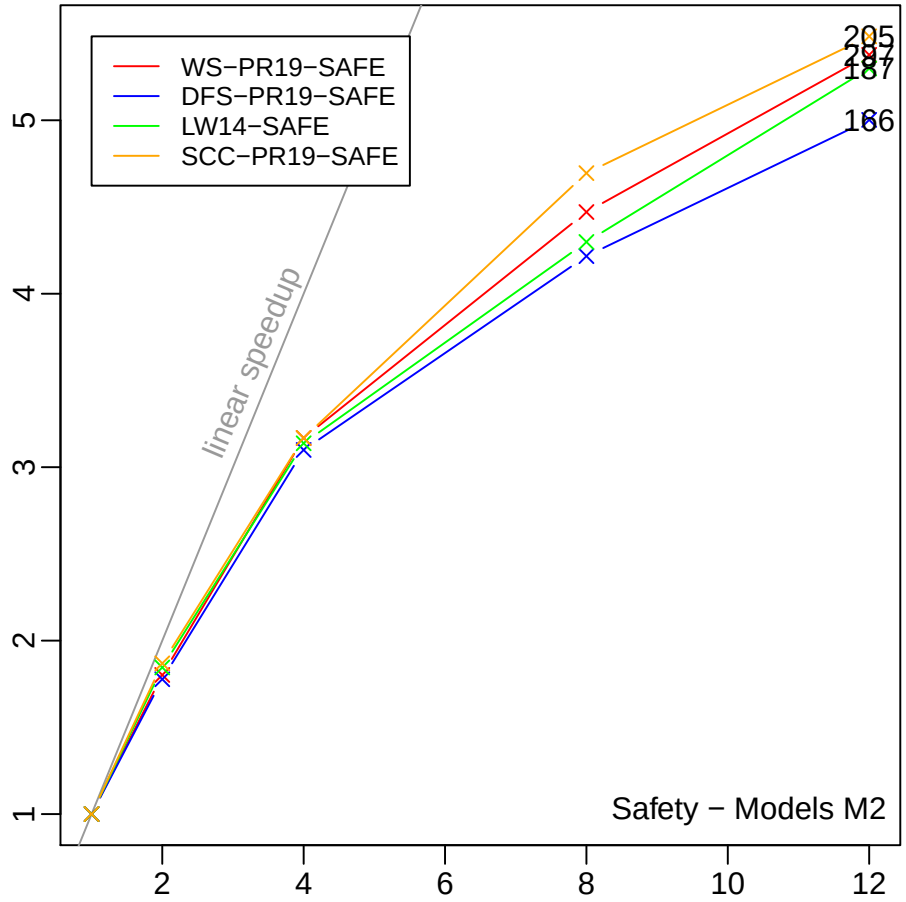

0.00
0.05
0.10
0.15
2
4
6
8
10
12
WS−PR19−SAFE
DFS−PR19−SAFE
LW14−SAFE
SCC−PR19−SAFE
Safety − Models M1


0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8
10
12
WS−PR19−LIVE
DFS−PR19−LIVE
LW14−LIVE
Liveness − Models M2


0.0
0.1
0.2
0.3
0.4
0.5
2
4
6
8
10
12
WS−PR19−SAFE
DFS−PR19−SAFE
LW14−SAFE
SCC−PR19−SAFE
Safety − Models M2

Speedup


WS−PR19−LIVE
DFS−PR19−LIVE
LW14−LIVE
linear speedup
359
187
166
Liveness − Models M2
1
2
3
4
5
2
4
6
8
10
12


WS−PR19−SAFE
DFS−PR19−SAFE
LW14−SAFE
SCC−PR19−SAFE
linear speedup
205
187
166
Safety − Models M2
1
2
3
4
5
2
4
6
8
10
12

Conclusion & Perspective

- Combine POR with SOTAs emptiness check for both liveness and safety properites
- Intensive evaluation
- Independant of the reduction technique: ample set, sttuborn set, etc. (see Laarman et al. [7] for survey)

Perspectives

Can we build an non-pessimistic algorithm for the combination between Bloemen's emptiness check and POR?

Bibliography I

[1] Bloemen, V. and van de Pol, J. (2016). Multi-core scc-based ltl model checking. In Bloem, R. and Arbel, E., editors, Proceedings of the 12th International Haifa Verification Conference (HVC'16), Lecture Notes in Computer Science, pages 18–33. Springer International Publishing.

[2] 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.

[3] Duret-Lutz, A., Kordon, F., Poitrenaud, D., and Renault, E. (2016). Heuristics for checking liveness properties with partial order reductions. In Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16), volume 9938 of Lecture Notes in Computer Science, pages 340–356. Springer.

[4] 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.

[5] Evangelista, S. and Pajault, C. (2010). Solving the ignoring problem for partial order reduction. International Journal on Software Tools for Technology Transfer, 12(2):155–170.

Bibliography II

[6] 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 (SPIN'96), volume 32 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society.

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

[8] Laarman, A. W. and Wijs, A. J. (2014). Partial-order reduction for multi-core ltl model checking. In Yahav, E., editor, HVC 2014, volume 8855 of LNCS, pages 267–283. Springer.

[9] 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), pages 1–21.