Platform Dependent Verification

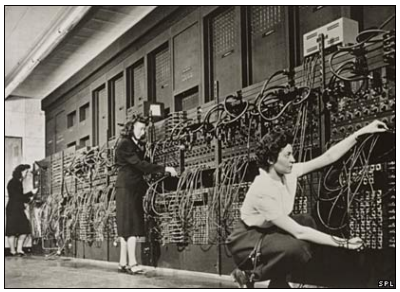
Engineering Verification Tools for 21st Century

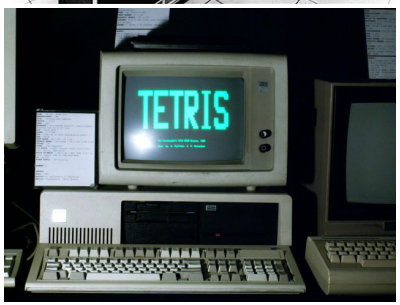
Luboš Brim and Jiří Barnat

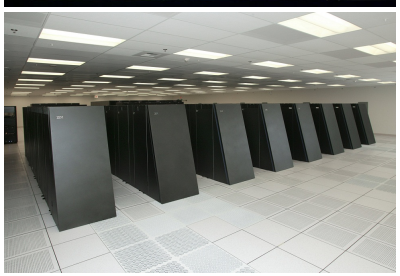
ParaDiSe
Parallel & Distributed Systems Laboratory

Masaryk University
Czech Republic

Motivation



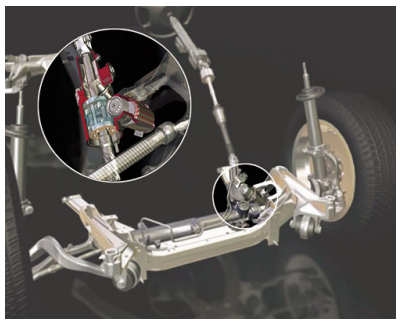




Complexity of developed systems is growing

- Numerical computation & text processing
- Database systems, computer games
- Simulations of physical systems
- Virtual reality
- ...

Parallelism and communication are essential

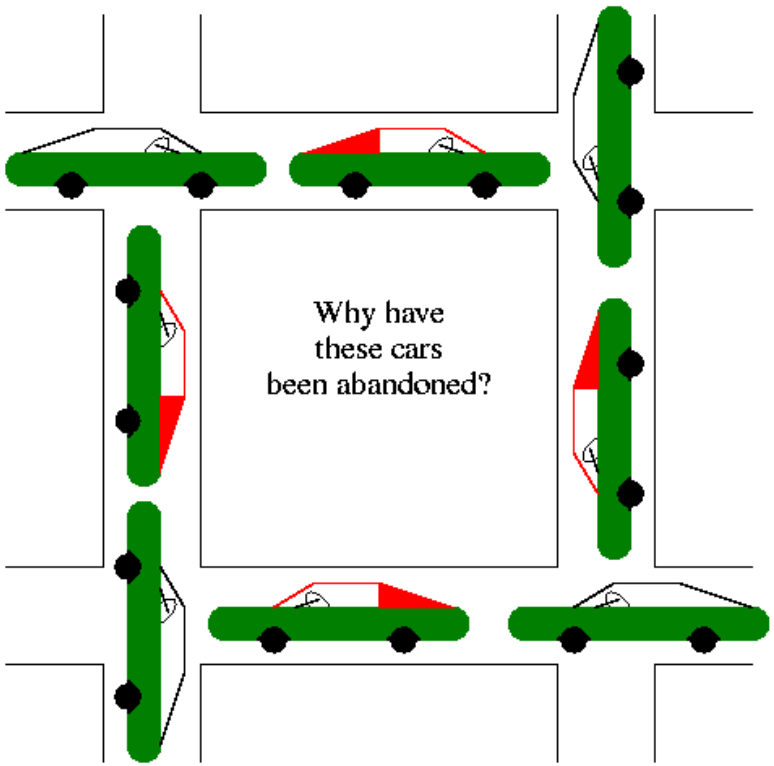


Correctness and reliability are critical measures.

Flaws of concurrent systems

Concurrent systems are everywhere

- Network applications, data communication protocols
- Client-server systems
- Multi-threaded code
- ...



Errors Related to Concurrency

- Deadlocks
- Livelocks
- Underspecification
- Overspecification
- Assumption about execution speed

Formal Verification

Verification Methods

- Simulations, Prototyping, Testing, **Formal Verification**

Formal Verification Benefits

- Early integration in design process.
- More effective (higher coverage).
- Reduction in verification time.
- Reduction in development costs and time-to-market.

Press-the-button!

... a brief look at model-checking

Model Checking

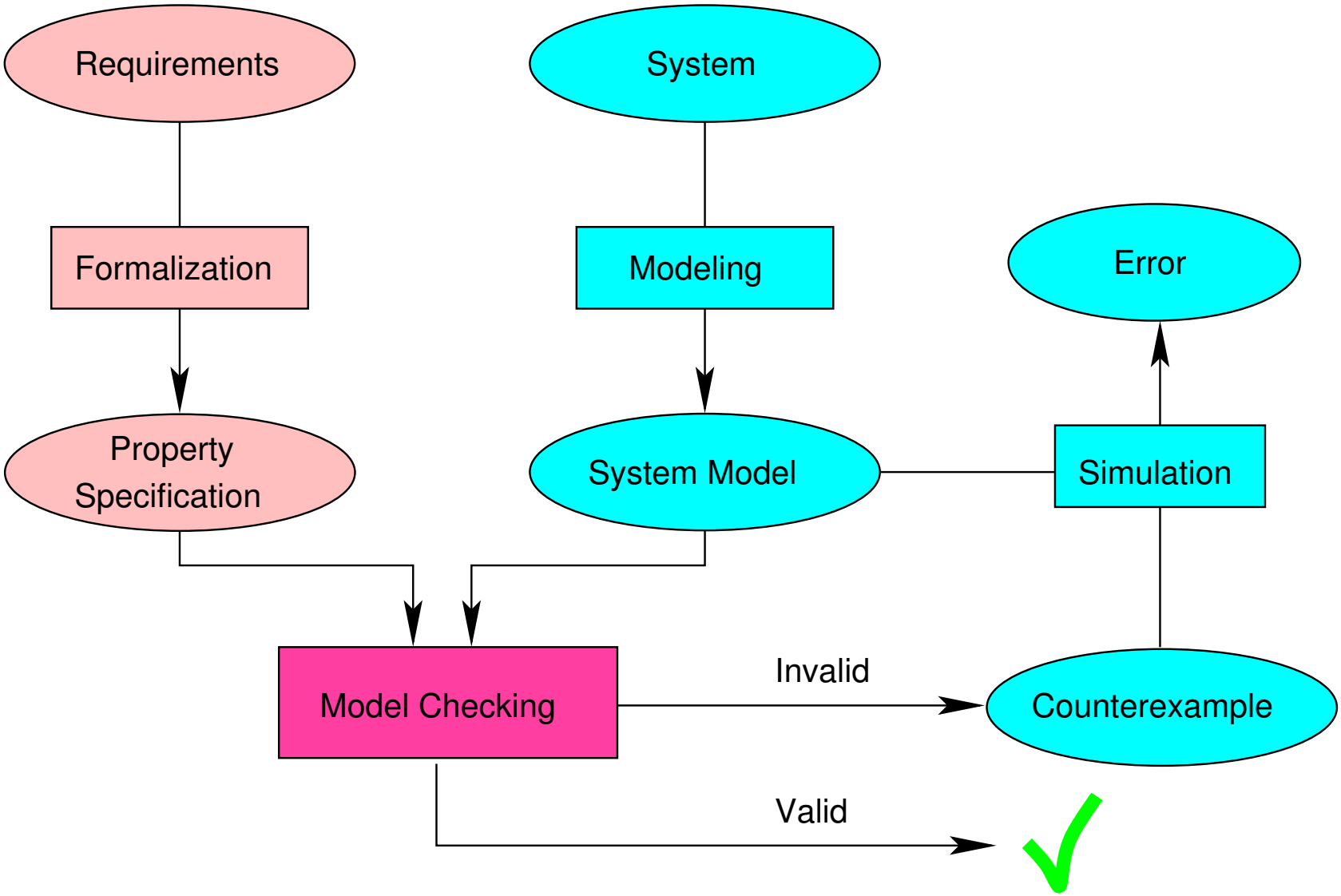
ACM Turing Award 2007

- Edmund M. Clarke jr. (CMU, USA)
- Allen E. Emerson (Texas at Austin, USA)
- Joseph Sifakis (IMAG Grenoble, France)

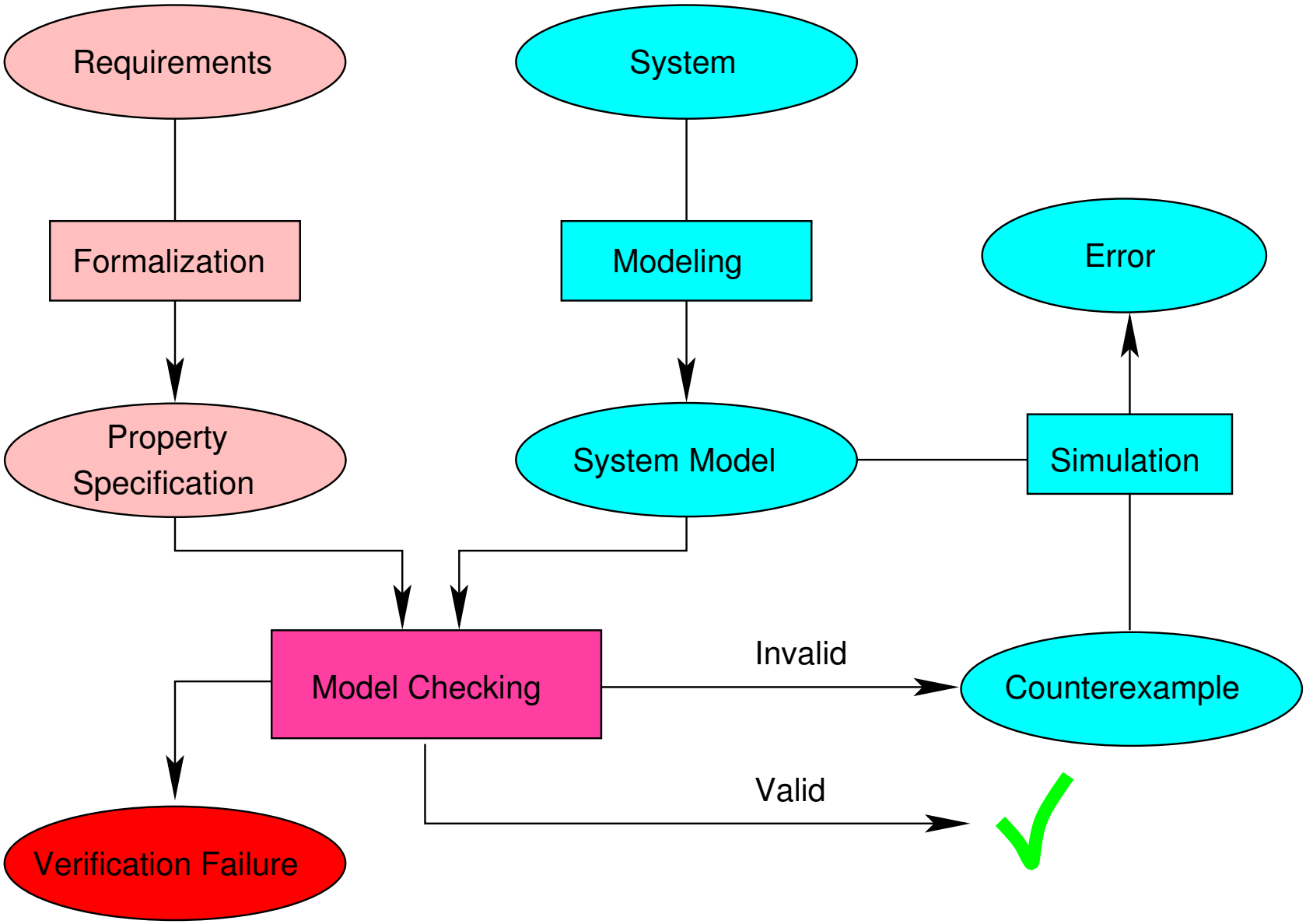
Jury justification

"For their roles in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries."

Model Checking and State Space Explosion Problem



Model Checking and State Space Explosion Problem



Fight the state explosion problem by squeezing all the power out of the contemporary hardware

What is needed ...

Contemporary computing platforms

- Distributed memory systems
- Shared memory parallelism
- GP GPU many-cores
- External memory
- . . . and combinations.

Efficient utilization requires

- Parallel algorithms
- Algorithm engineering

The DiVinE Story!

... platform dependent model checking with DiVinE

The DiVinE Story: DiVinE

What is DiVinE?

- Explicit-state automata-based LTL model checker.
- Can exploit aggregate computing power of multiple network-interconnected multi-cored workstations.

Where is it?

`http://divine.fi.muni.cz`

The DiVinE Story!

... parallel algorithms

The DiVinE Story: Need for Parallel Algorithms

Automata-based LTL Model Checking

- Accepting cycle detection in a directed graph.

Optimal Serial Algorithms

- Rely on **difficult-to-parallelize** DFS-postorder.
- Inefficient on parallel/distributed HW architectures and non-flat memory hierarchies.

Parallel scalable algorithms needed!

The DiVinE Story: Parallel Scalable Algorithms

	Complexity	Optimality	On-The-Fly
Nested DFS	O(N+M)	Yes	Yes
OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	No
weak LTL properties	O(N+M)	Yes	No
MAP Algorithm	O(N.N.(N+M))	No	Yes
MAP-OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	Yes
weak LTL properties	O(N+M)	Yes	Yes
BLEDGE Algorithm	O(N.N.(N+M))	No	Yes
NEGC Algorithm	O(N.N.(N+M))	No	Yes

N – number of states

M – number of transitions

The DiVinE Story: Parallel Scalable Algorithms

	Complexity	Optimality	On-The-Fly
Nested DFS	O(N+M)	Yes	Yes
OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	No
weak LTL properties	O(N+M)	Yes	No
MAP Algorithm	O(N.N.(N+M))	No	Yes
MAP-OWCTY Algorithm			
general LTL properties	O(N.(N+M))	No	Yes
weak LTL properties	O(N+M)	Yes	Yes
BLEDGE Algorithm	O(N.N.(N+M))	No	Yes
NEGC Algorithm	O(N.N.(N+M))	No	Yes

N – number of states

M – number of transitions

The DiVinE Story: Partial Order Reduction

Observations

- Distributed memory processing should not stay isolated.
- It must combine with other techniques.

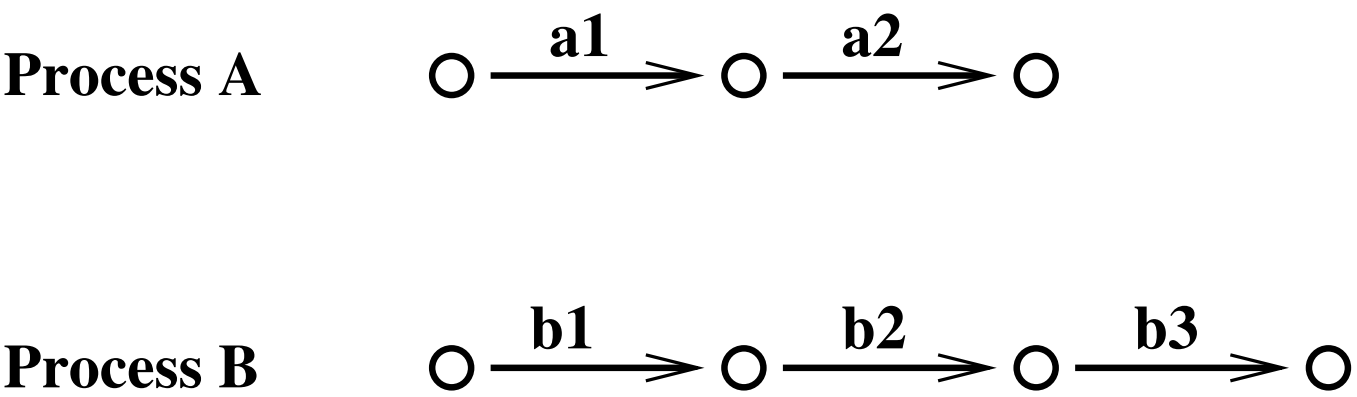
Partial Order Reduction (POR)

- Key technique to fight state space explosion in explicit state model checking.

POR Principle

Orders execution of independent actions among processes of the distributed system.

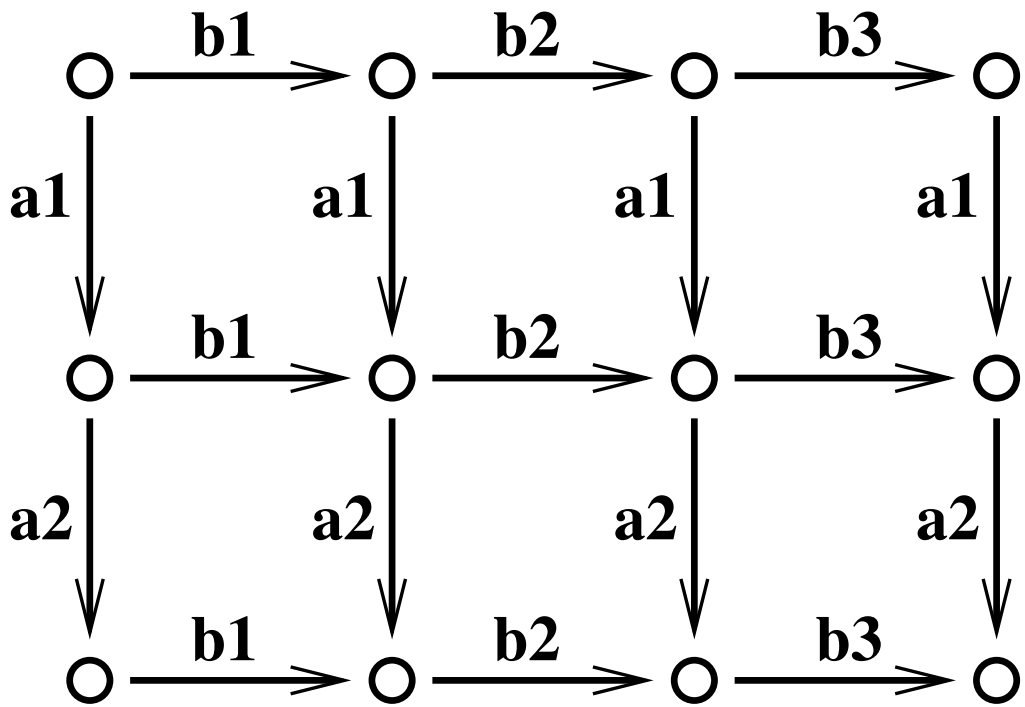
Distributed system definition



POR Principle

Orders execution of independent actions among processes of the distributed system.

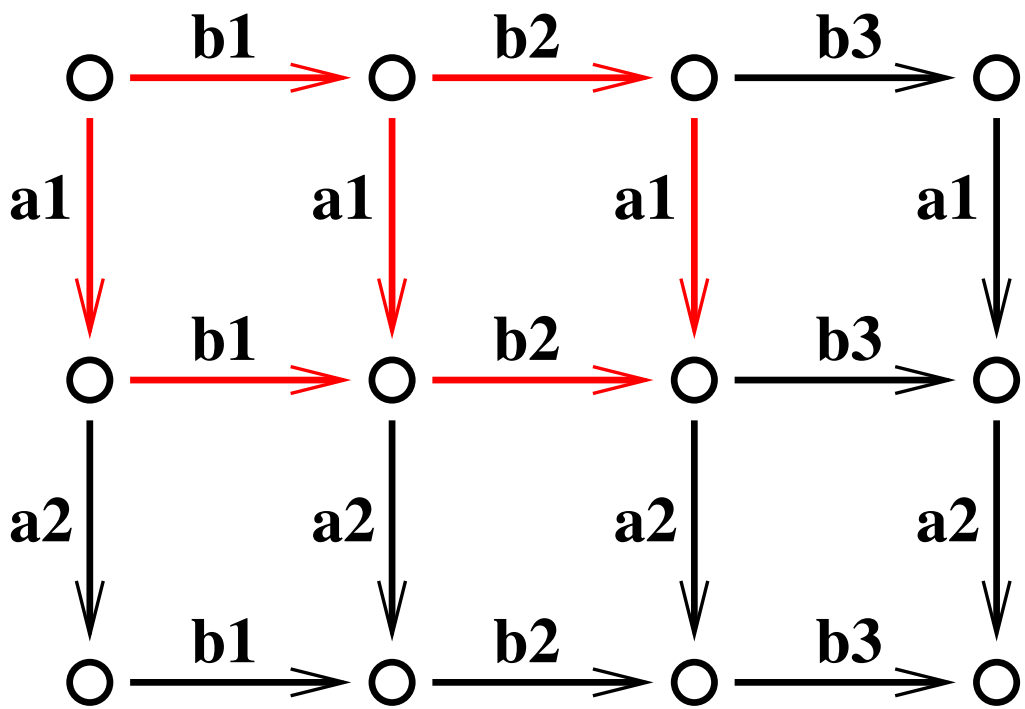
Unreduced State Space Graph



POR Principle

Suppose actions **a1**, **b1** and **b2** are independent.

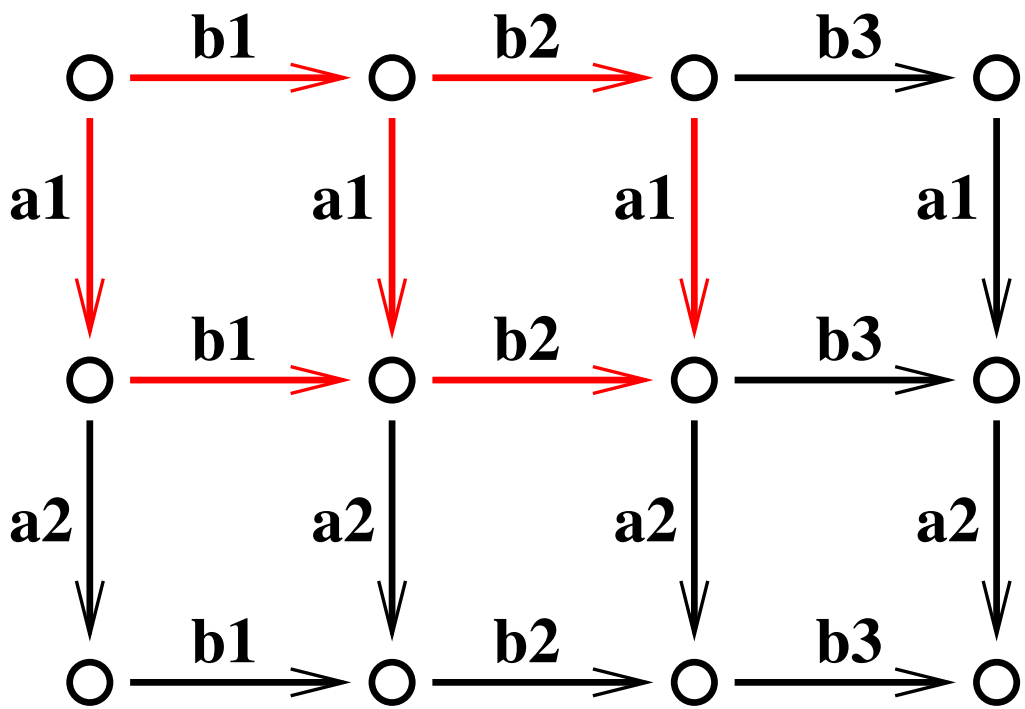
Unreduced State Space Graph



POR Principle

By postponing execution of selected independent action (e.g. **a1**) the state space graph is reduced.

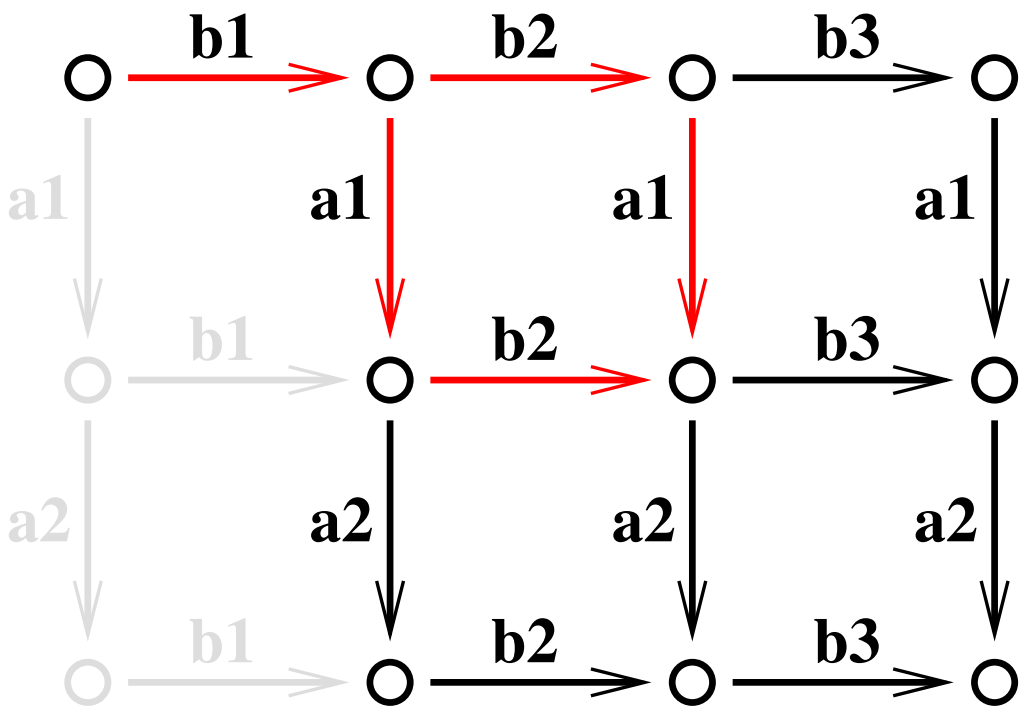
Unreduced State Space Graph



POR Principle

By postponing execution of selected independent action (e.g. **a1**) the state space graph is reduced.

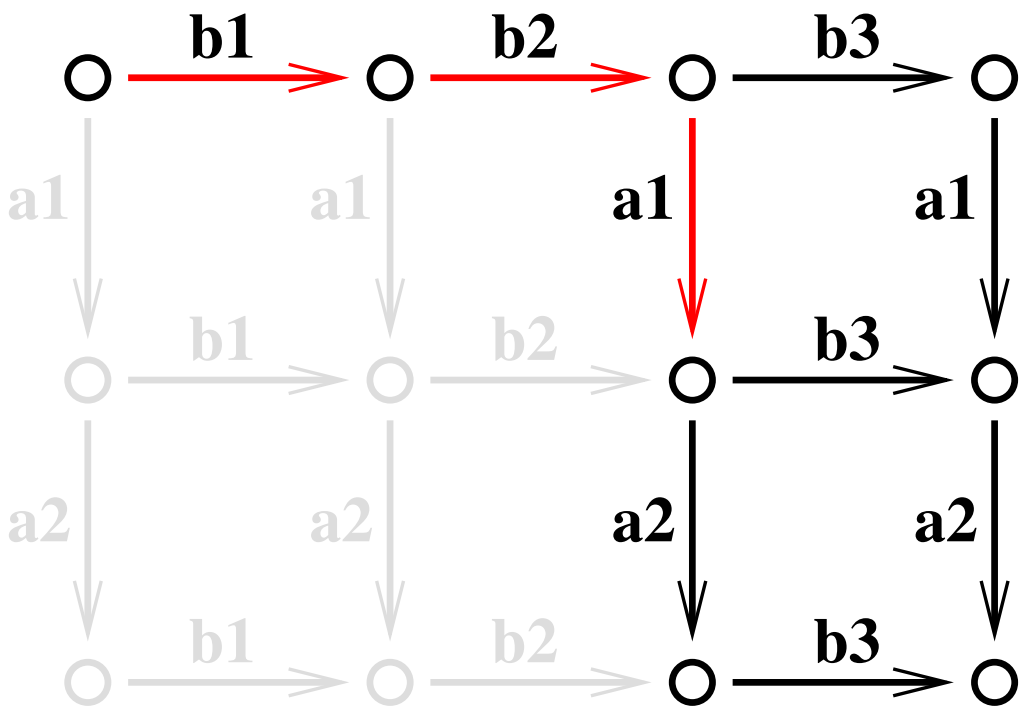
Reduced State Space Graph



POR Principle

By postponing execution of selected independent action (e.g. **a1**) the state space graph is reduced.

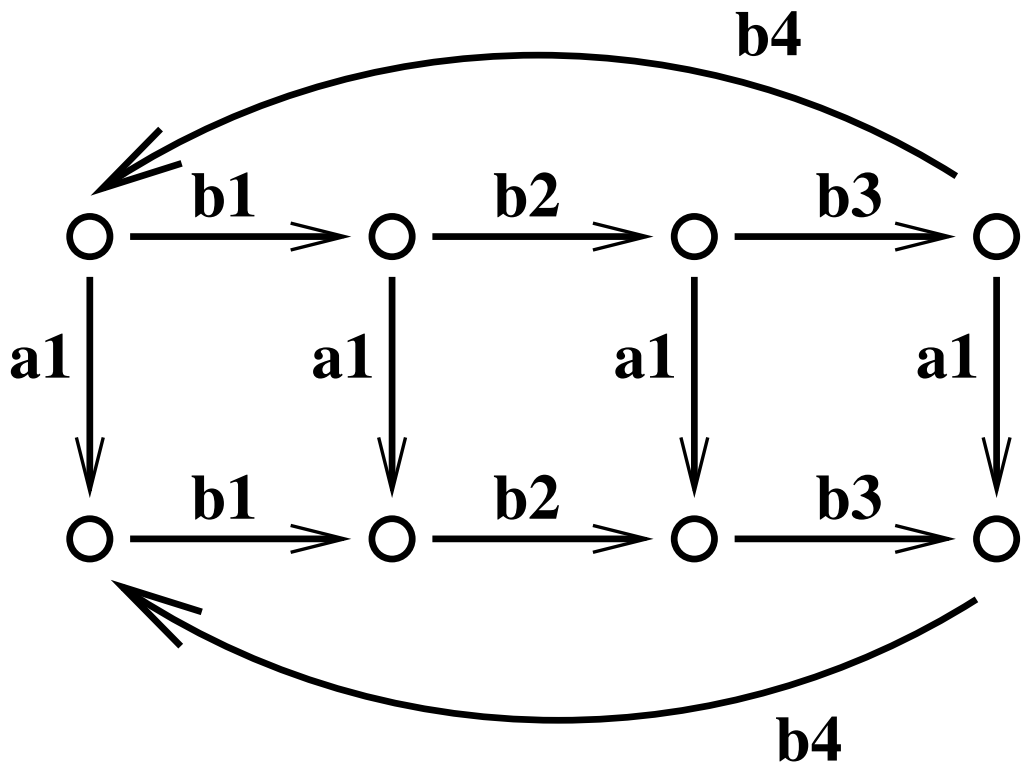
Reduced State Space Graph



POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

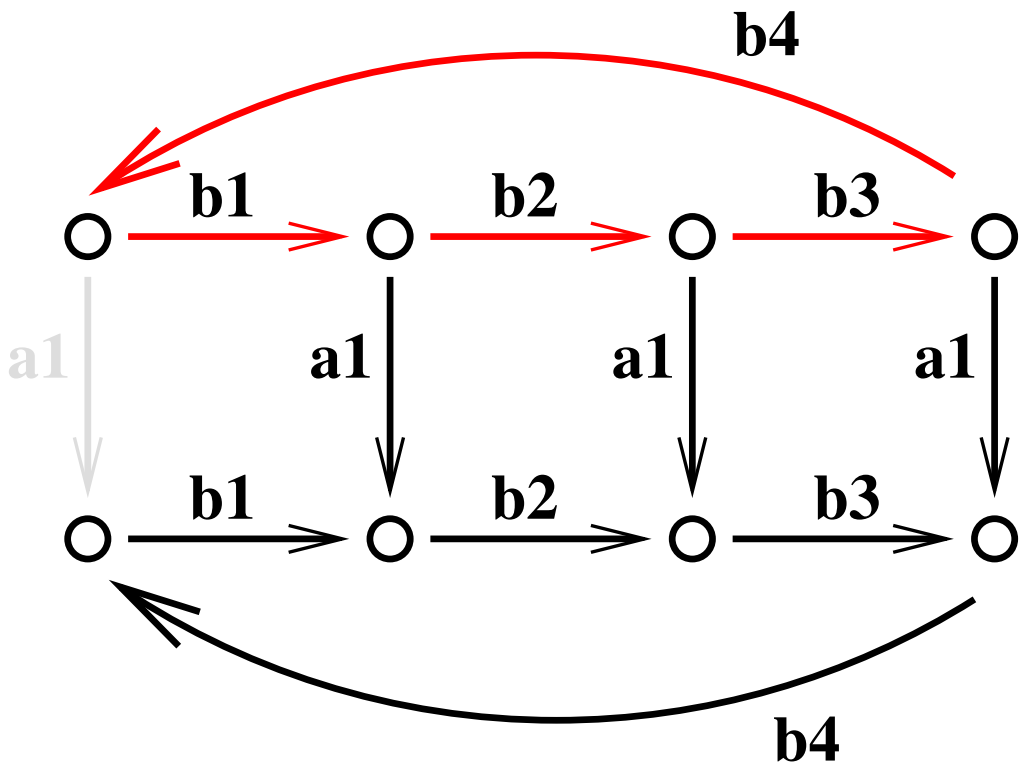
Unreduced State Space Graph



POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

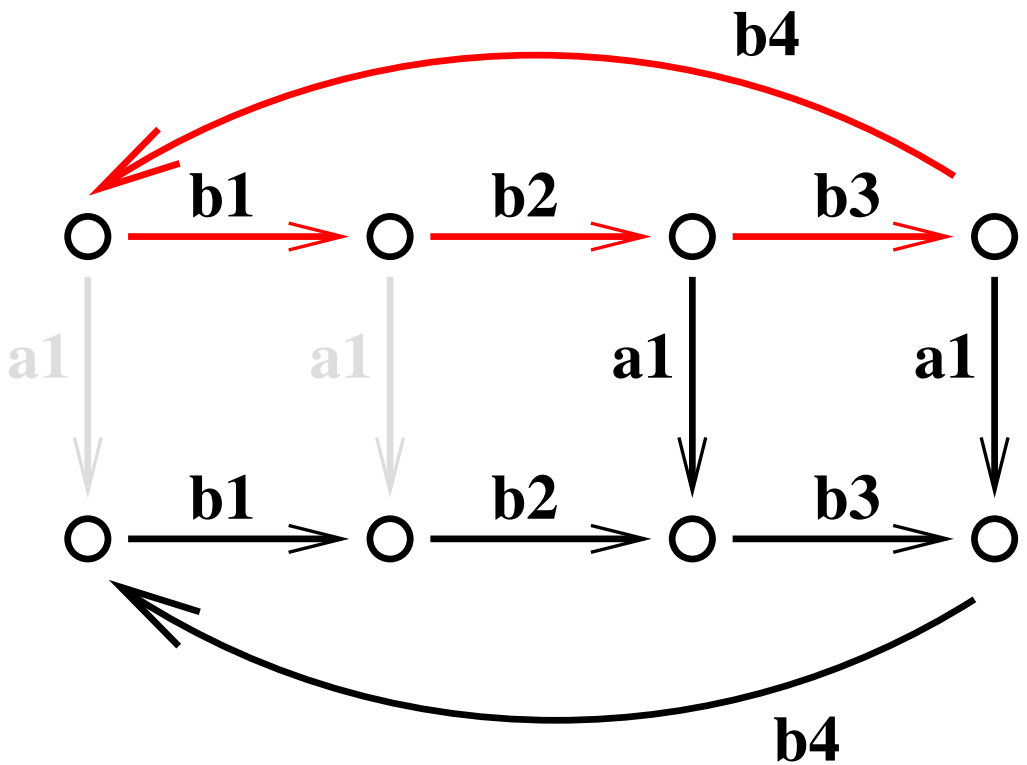
Permanent Postponing Problem



POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

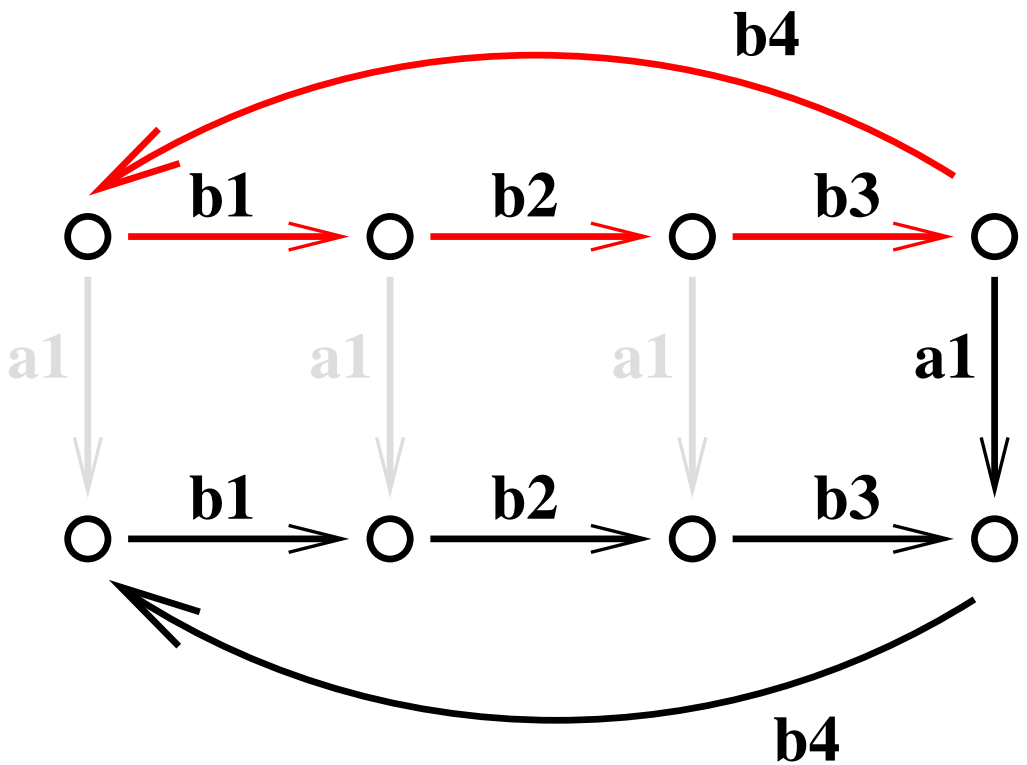
Permanent Postponing Problem



POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

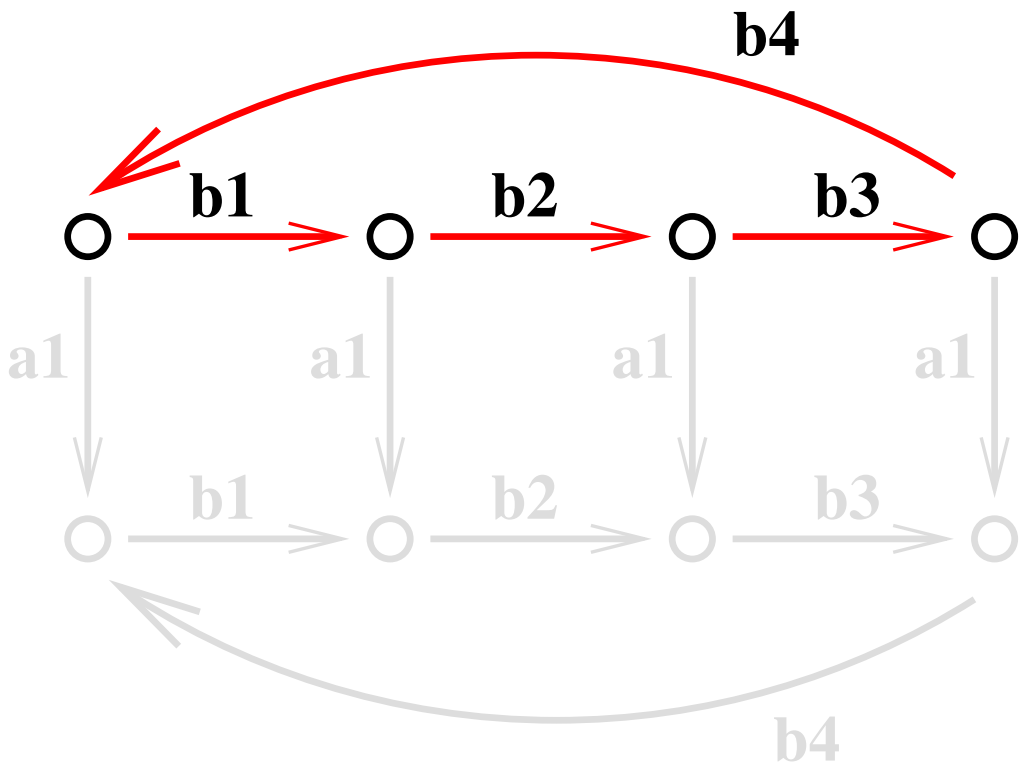
Permanent Postponing Problem



POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

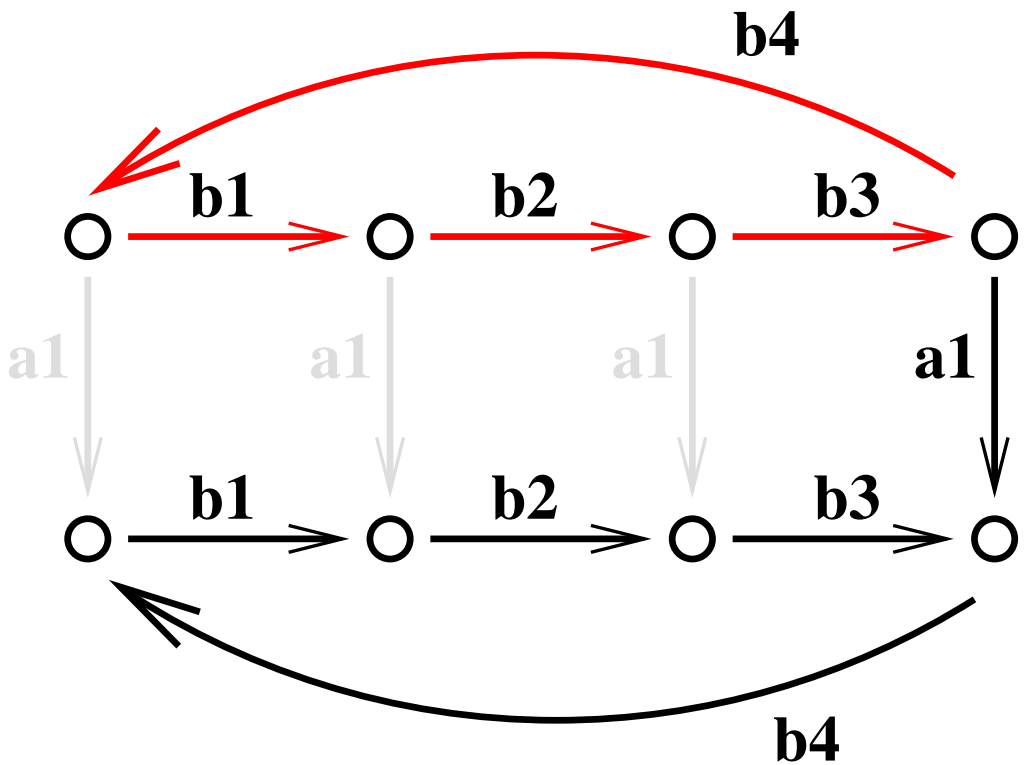
Permanent Postponing Problem



POR Principle

Infinite postponing is avoided if every cycle contains at least one fully expanded state ⇒ **cycle proviso**.

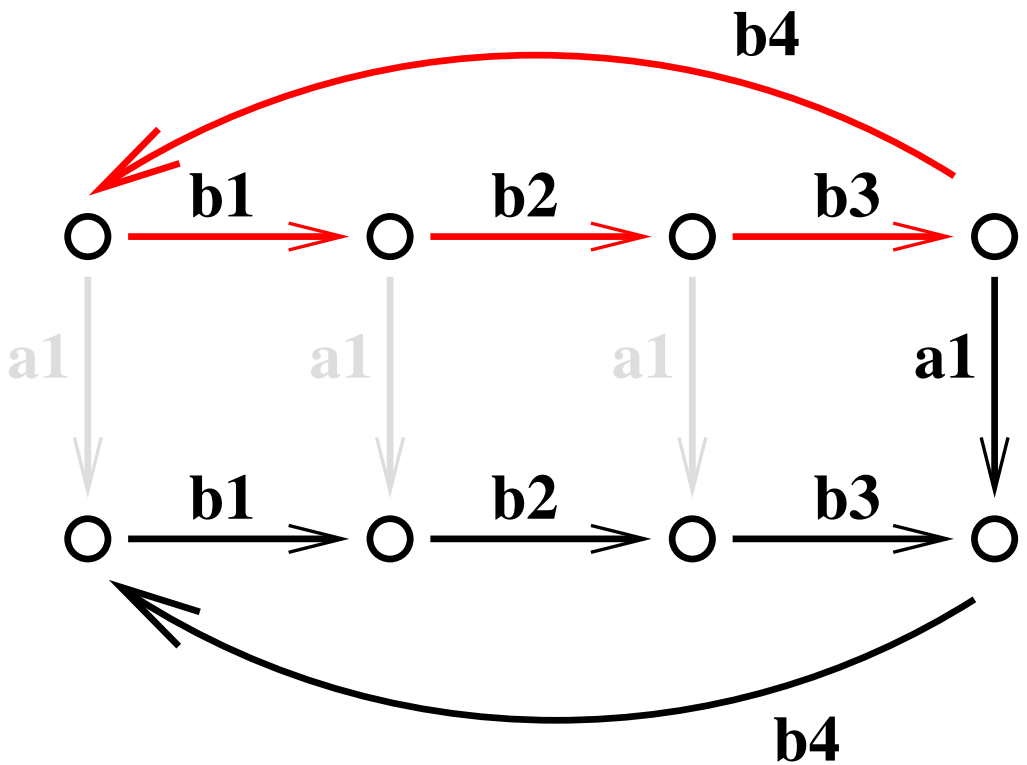
Cycle with Fully Expanded State



POR Principle

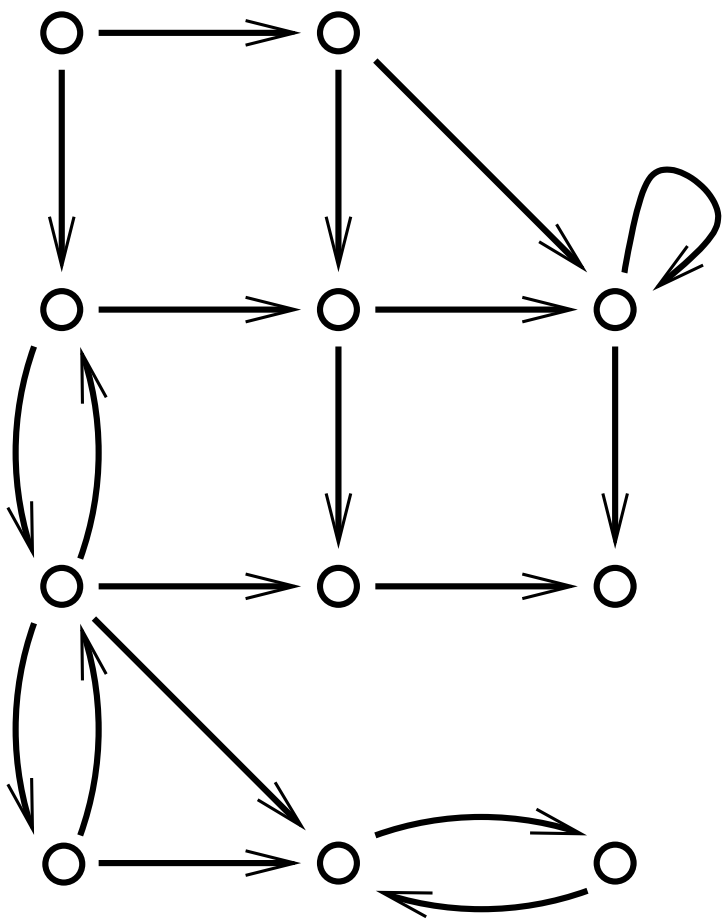
To detect the cycle, **depth-first search stack** is used in the standard sequential approach.

Cycle with Fully Expanded State



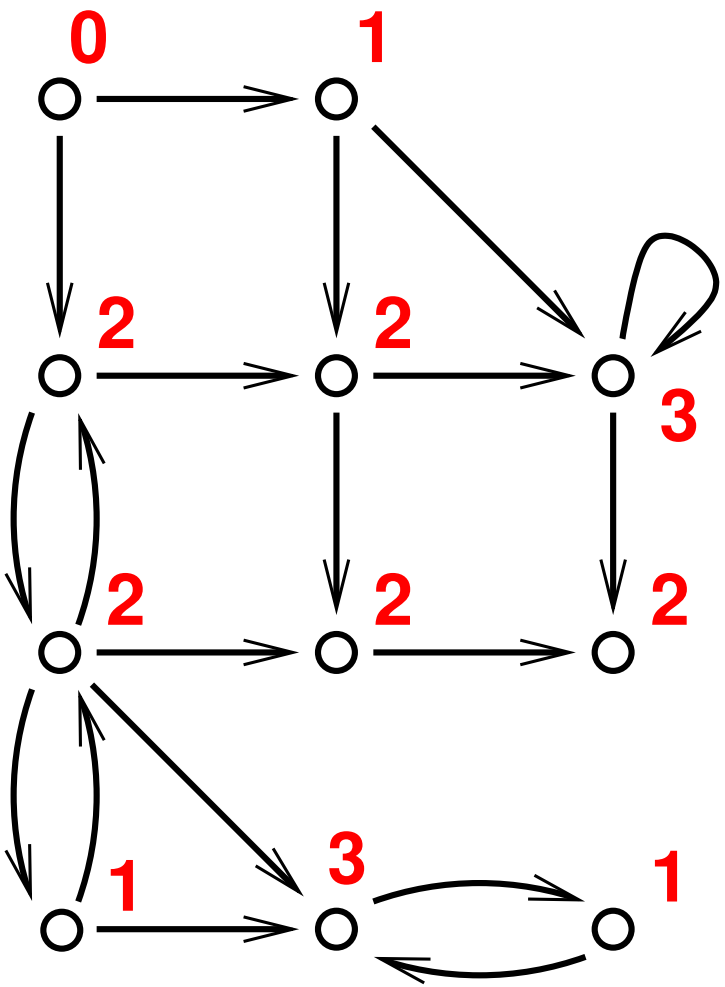
The DiVinE Story: Topological Sort Proviso Demo

Suppose a directed graph.



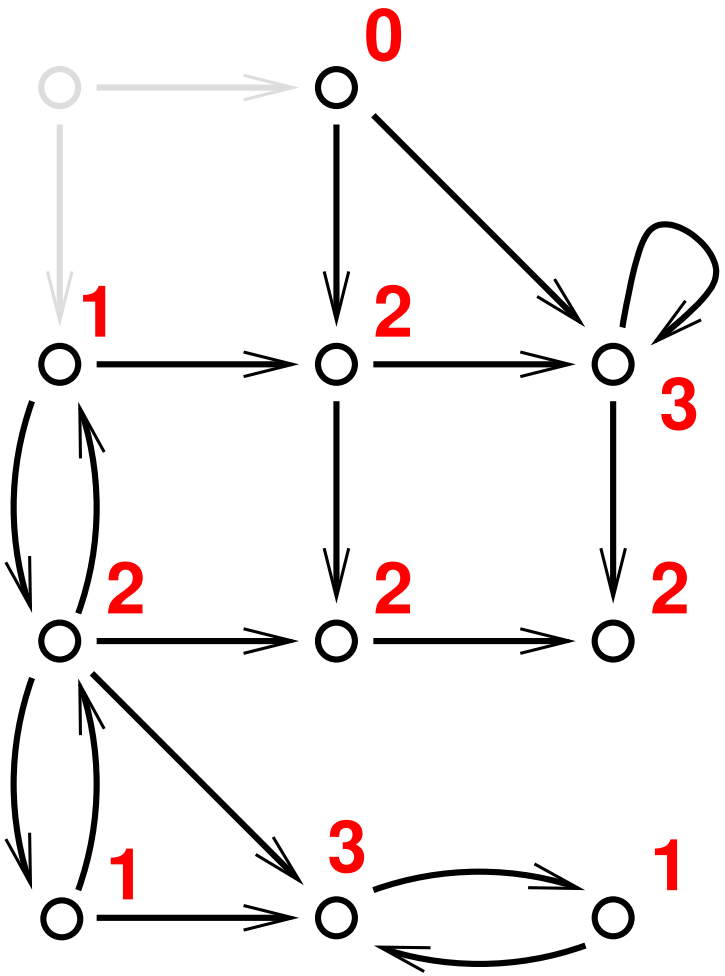
The DiVinE Story: Topological Sort Proviso Demo

First, indegree is computed for every vertex.



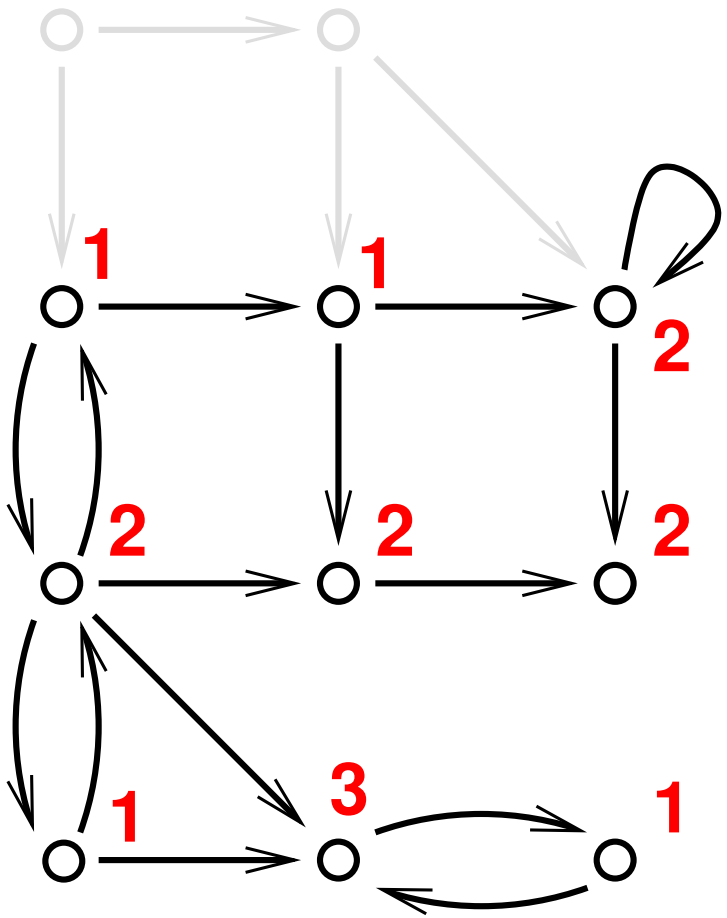
The DiVinE Story: Topological Sort Proviso Demo

Vertices with zero indegree are "eliminated".



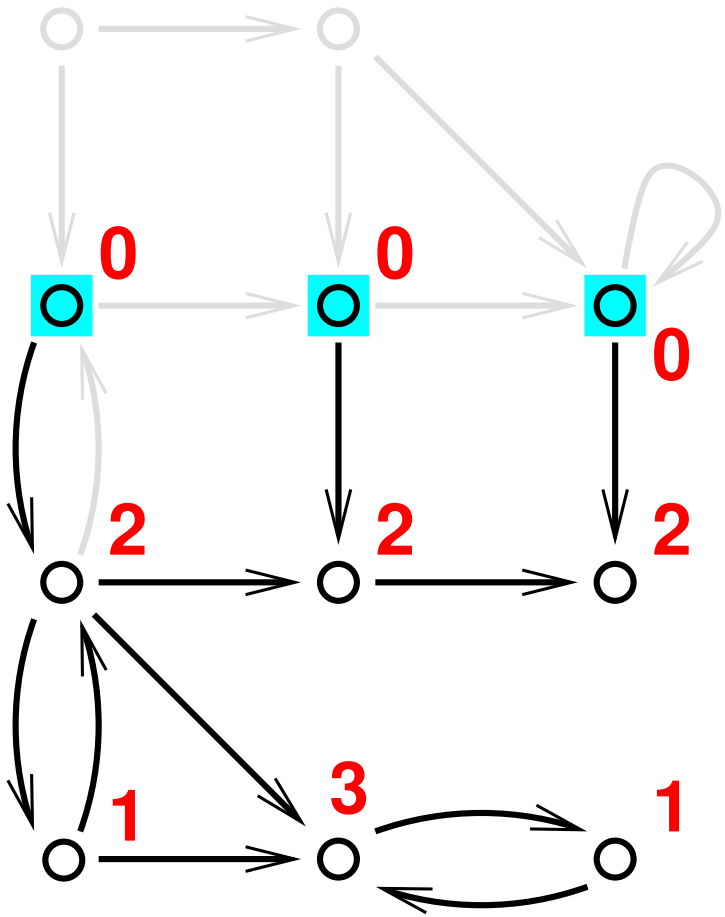
The DiVinE Story: Topological Sort Proviso Demo

Again, vertices with zero indegree are “eliminated”.



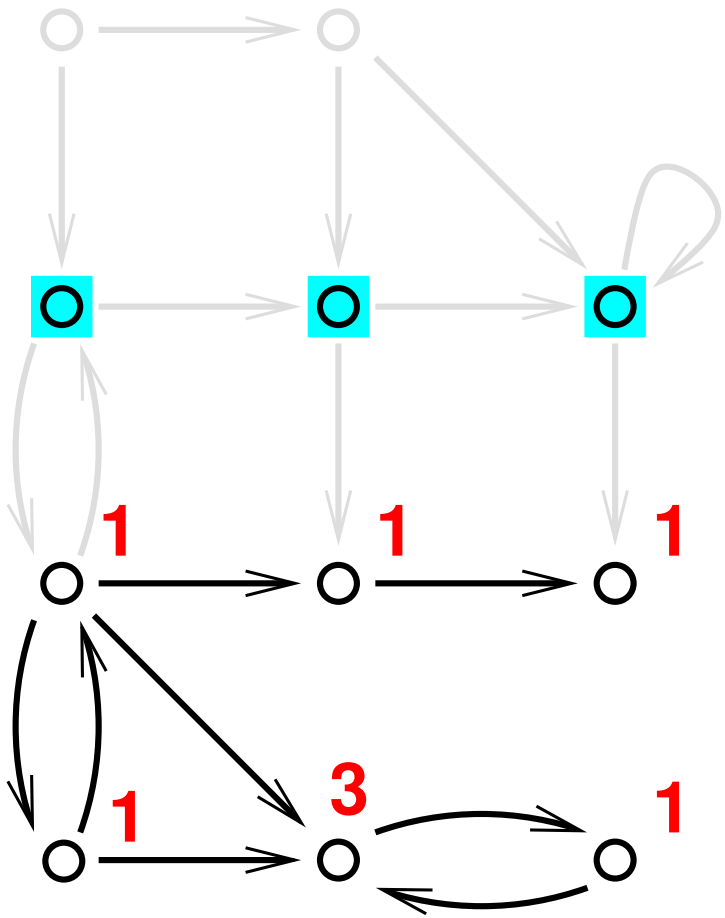
The DiVinE Story: Topological Sort Proviso Demo

Vertices whose indigree has decreased, are marked and removed.



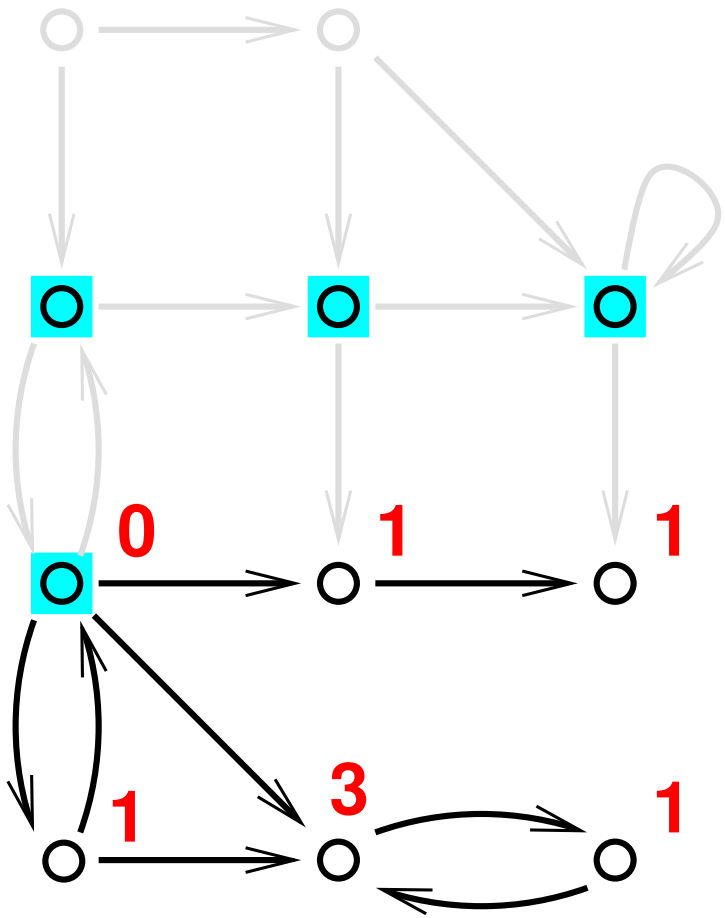
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



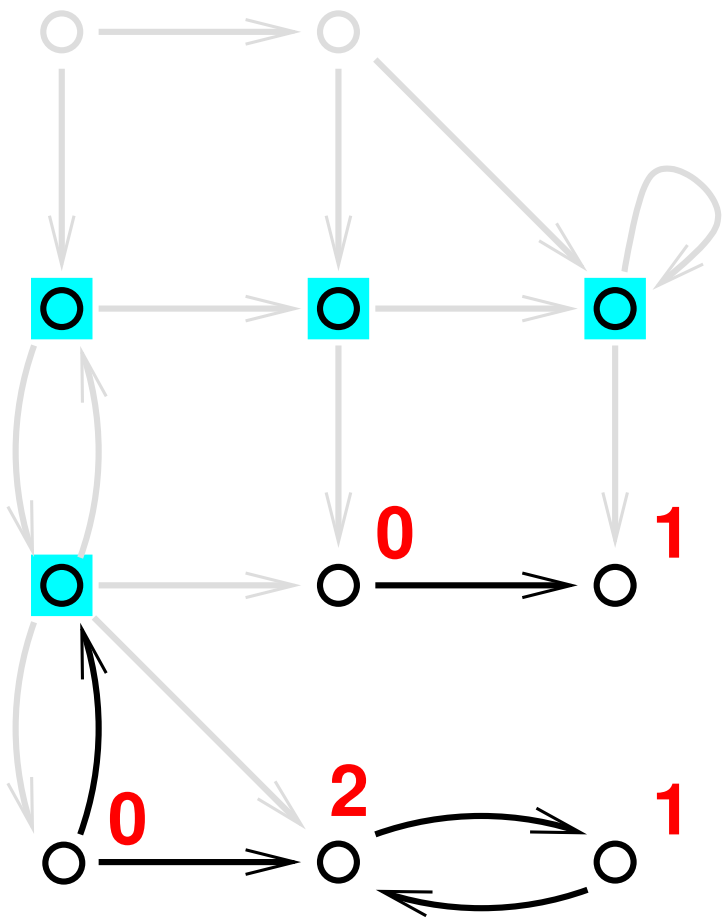
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



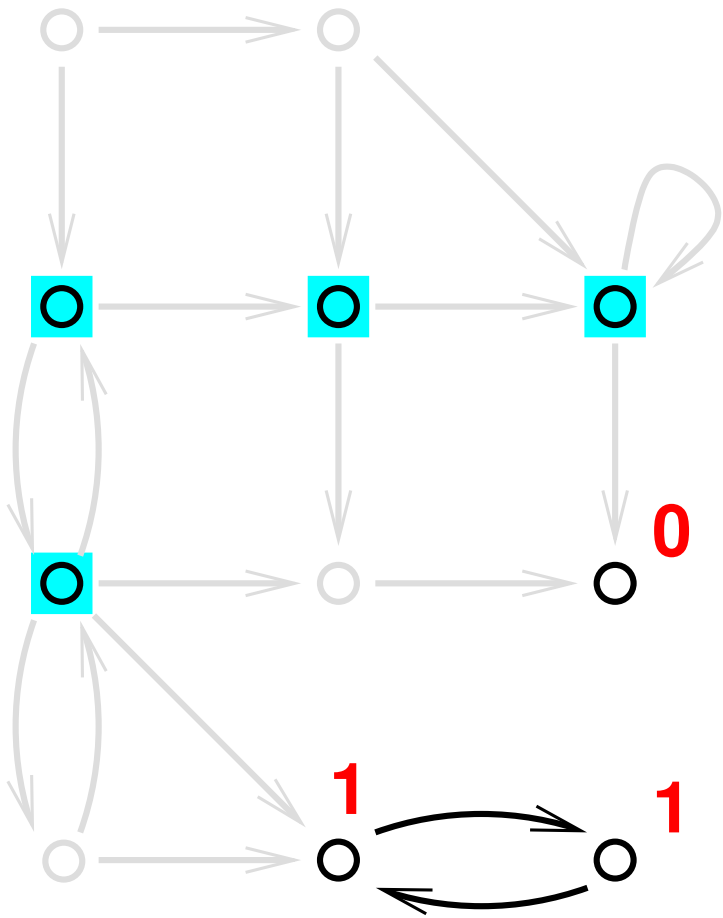
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



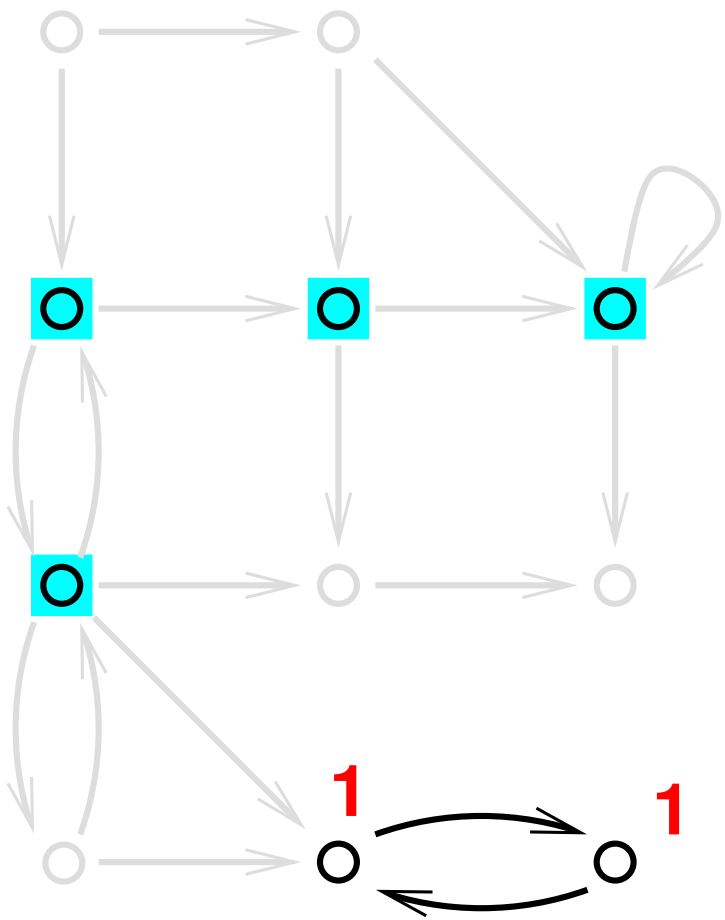
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



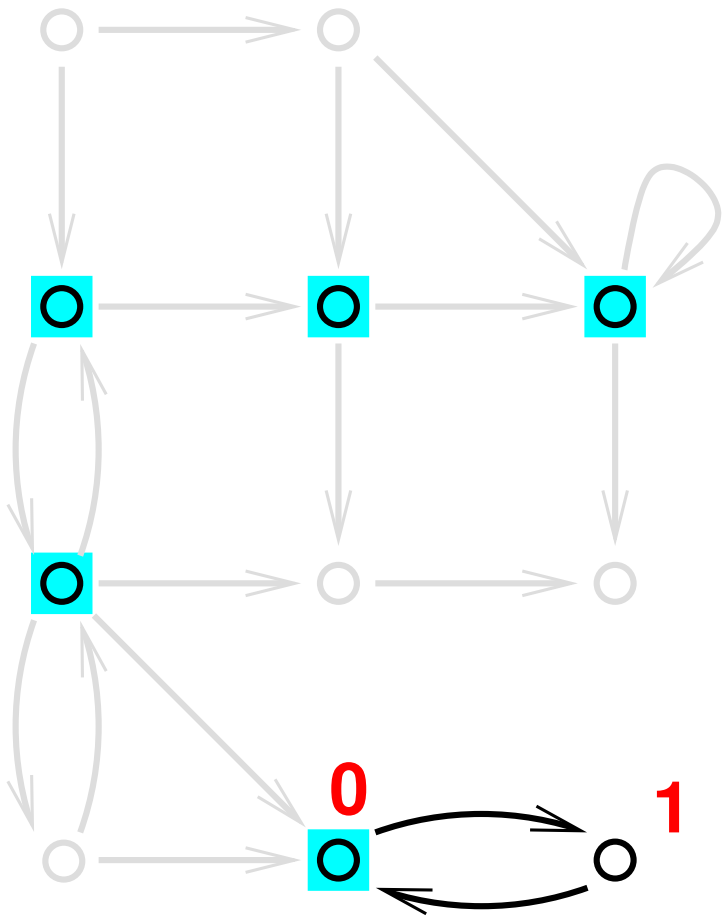
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



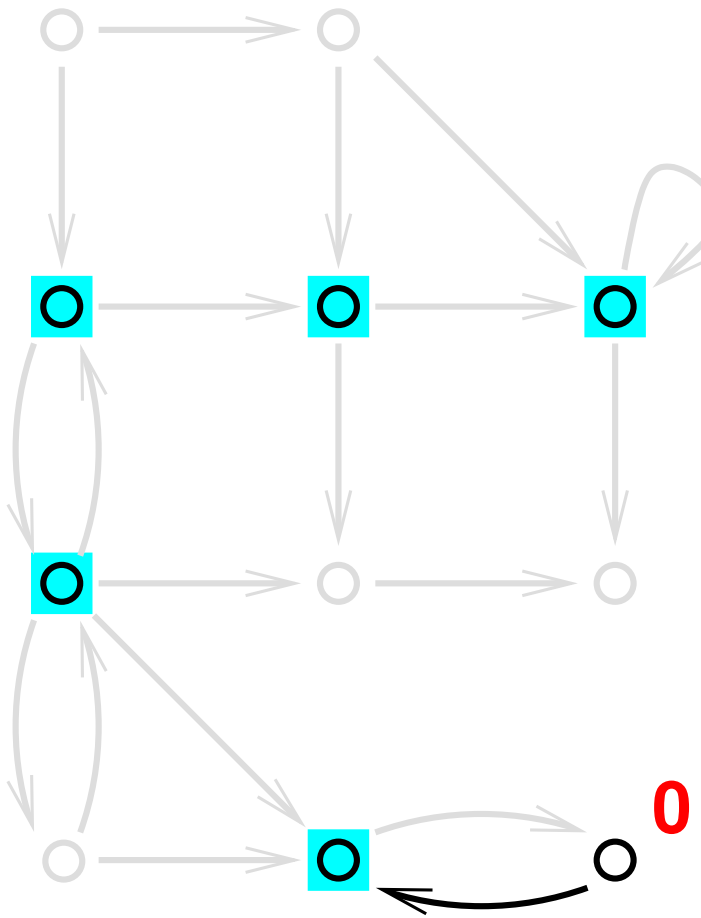
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



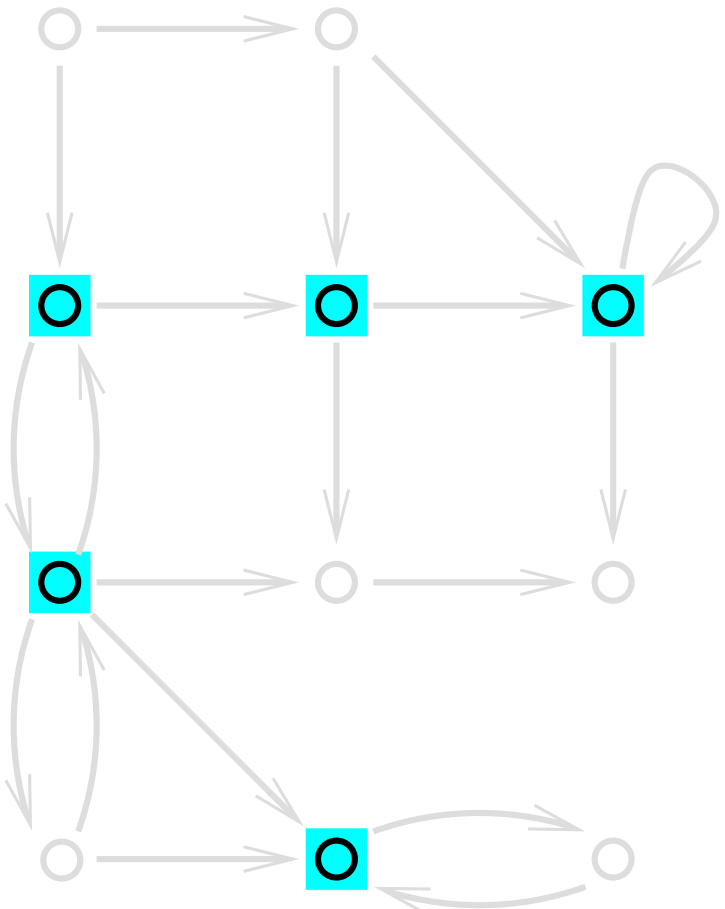
The DiVinE Story: Topological Sort Proviso Demo

Repeat until all of the graph is processed.



The DiVinE Story: Topological Sort Proviso Demo

All cycles are covered.

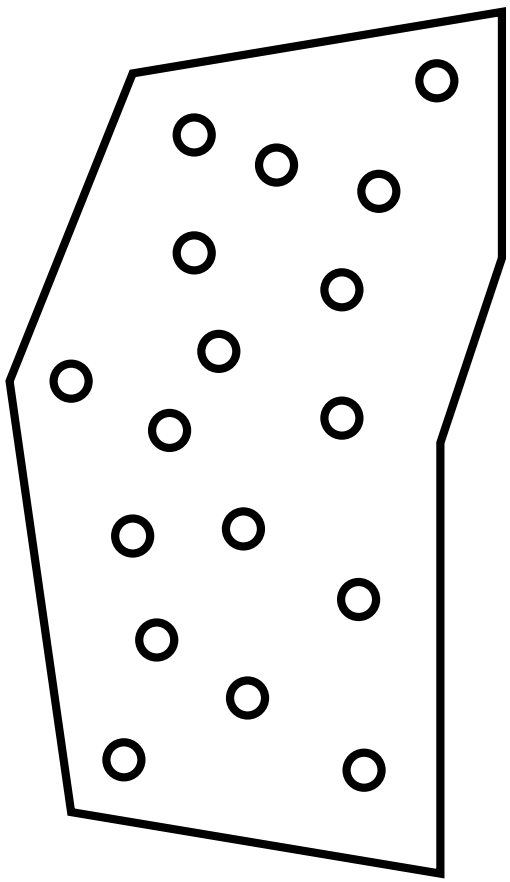


State Space Generation with POR – Demo

Generation starts from an initial state.

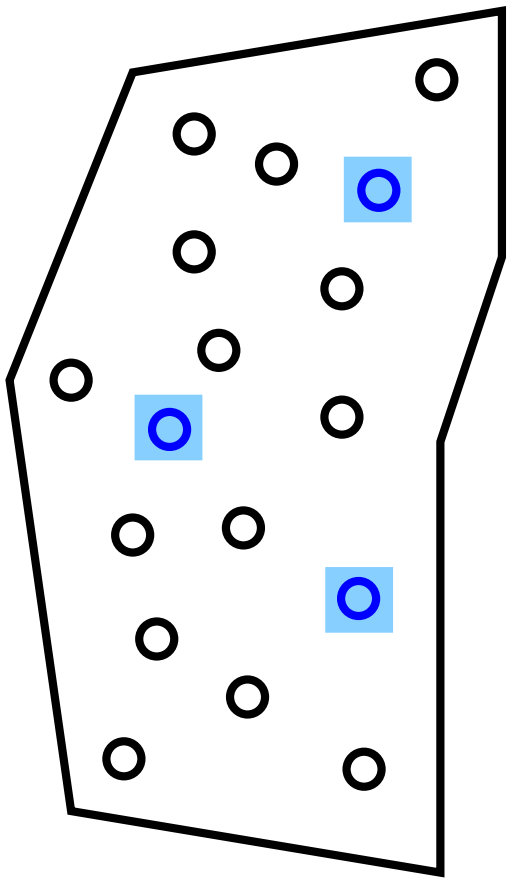
State Space Generation with POR – Demo

Initial part of the state space is generated using action-postponing principle.



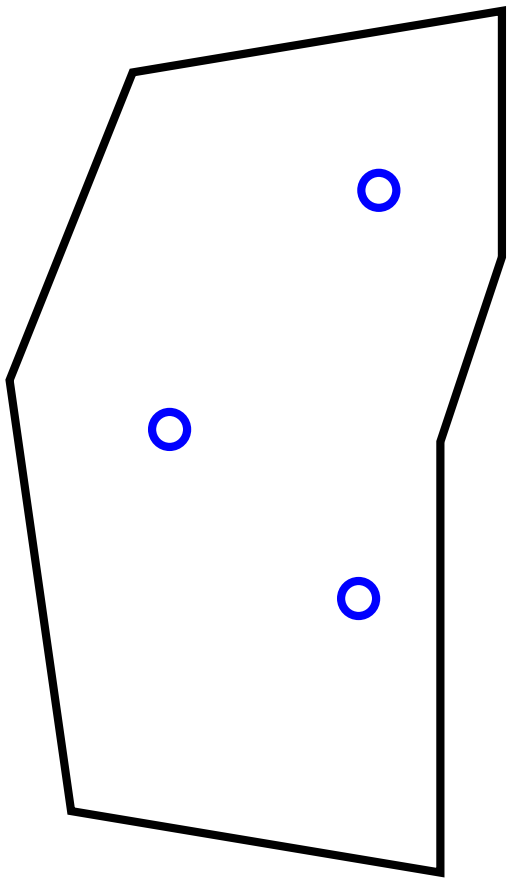
State Space Generation with POR – Demo

Topological sort proviso is applied to mark at least one state on every cycle.



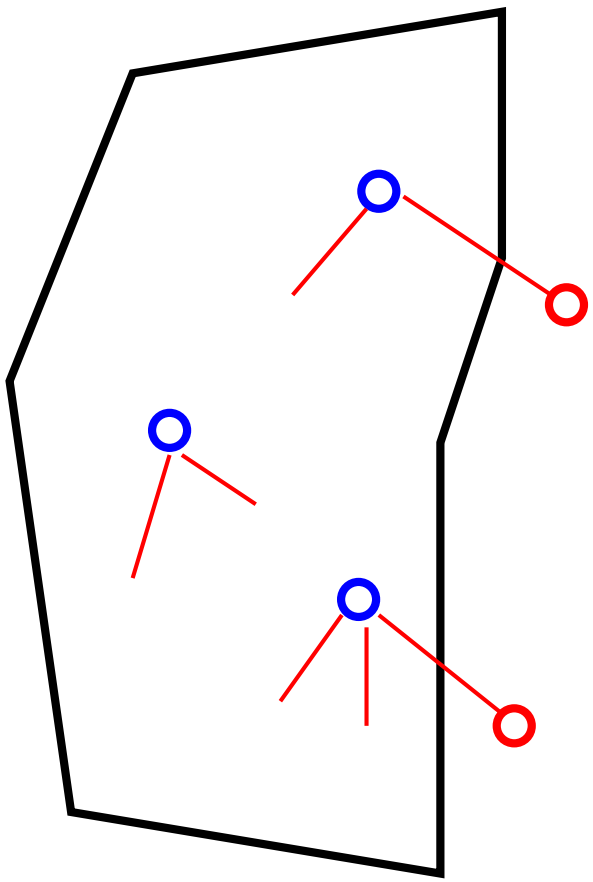
State Space Generation with POR – Demo

Topological sort proviso is applied to mark at least one state on every cycle.



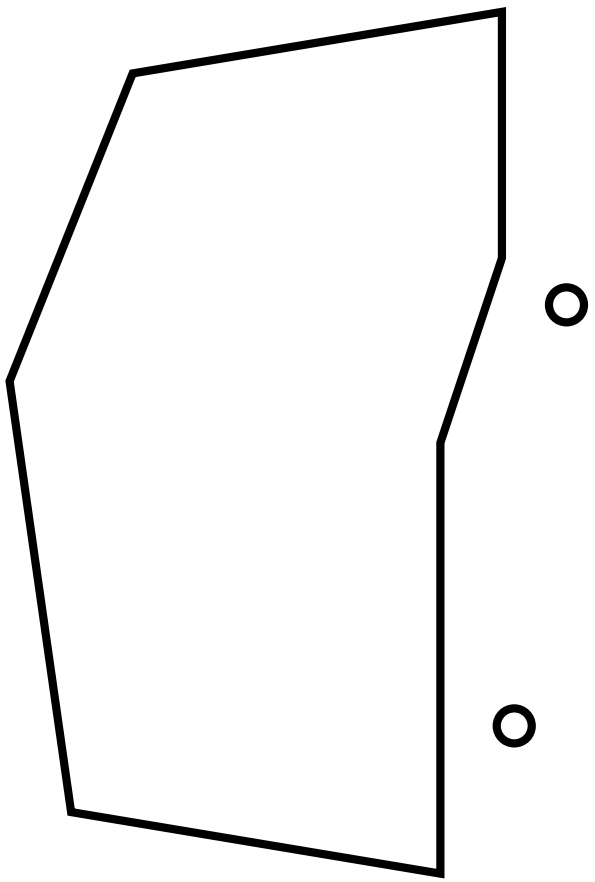
State Space Generation with POR – Demo

Marked states are fully re-expanded.



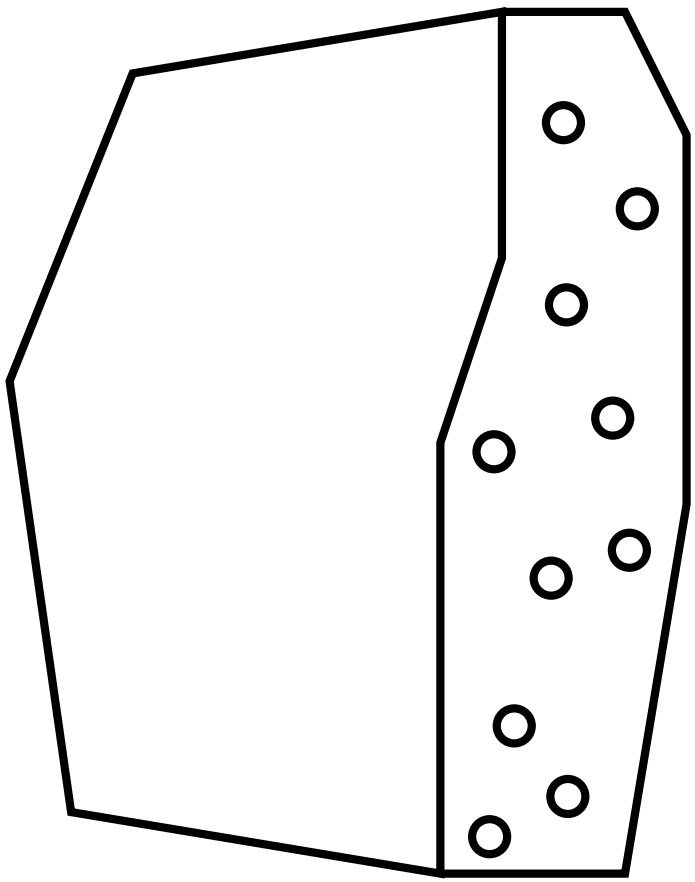
State Space Generation with POR – Demo

New states are discovered outside the previously generated part of the state space.



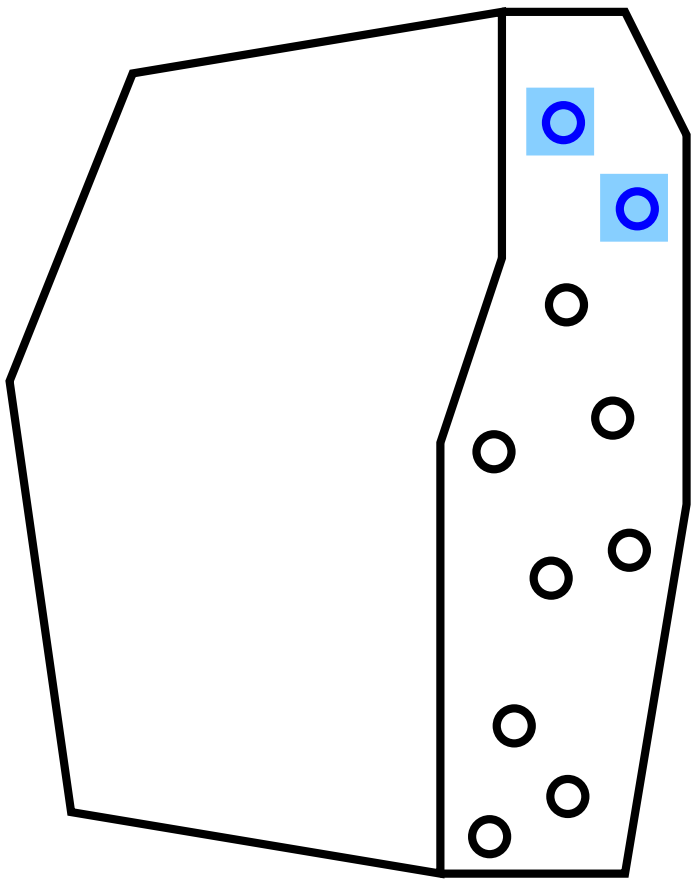
State Space Generation with POR – Demo

New part of the reduced state space is generated using action-postponing principle.



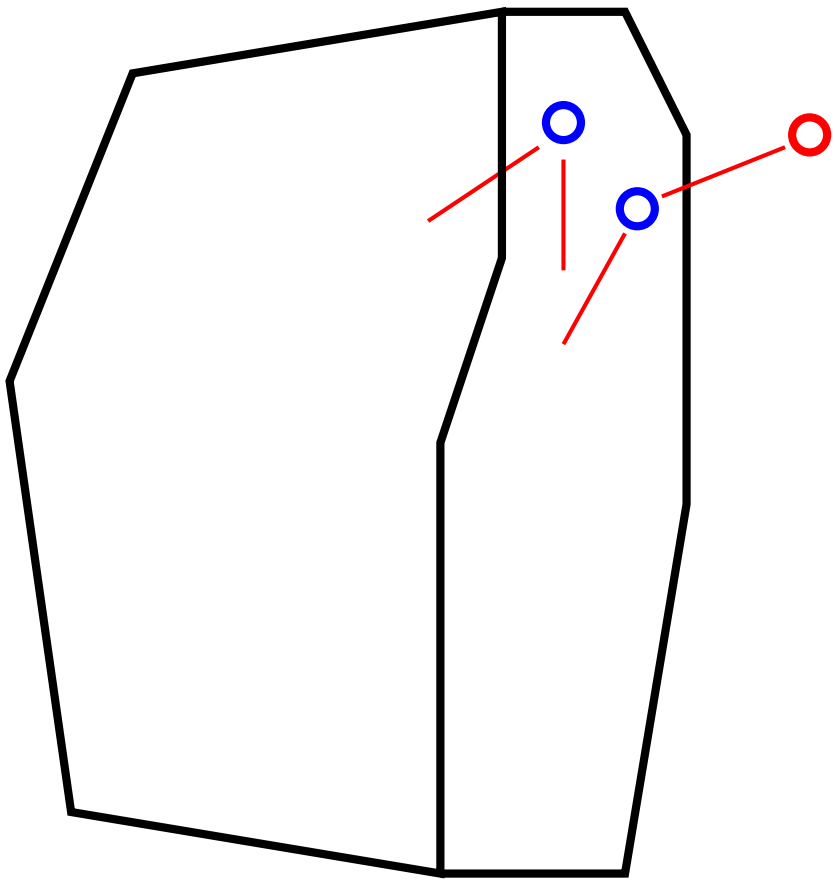
State Space Generation with POR – Demo

Topological sort proviso is applied to mark at least one state on every cycle.



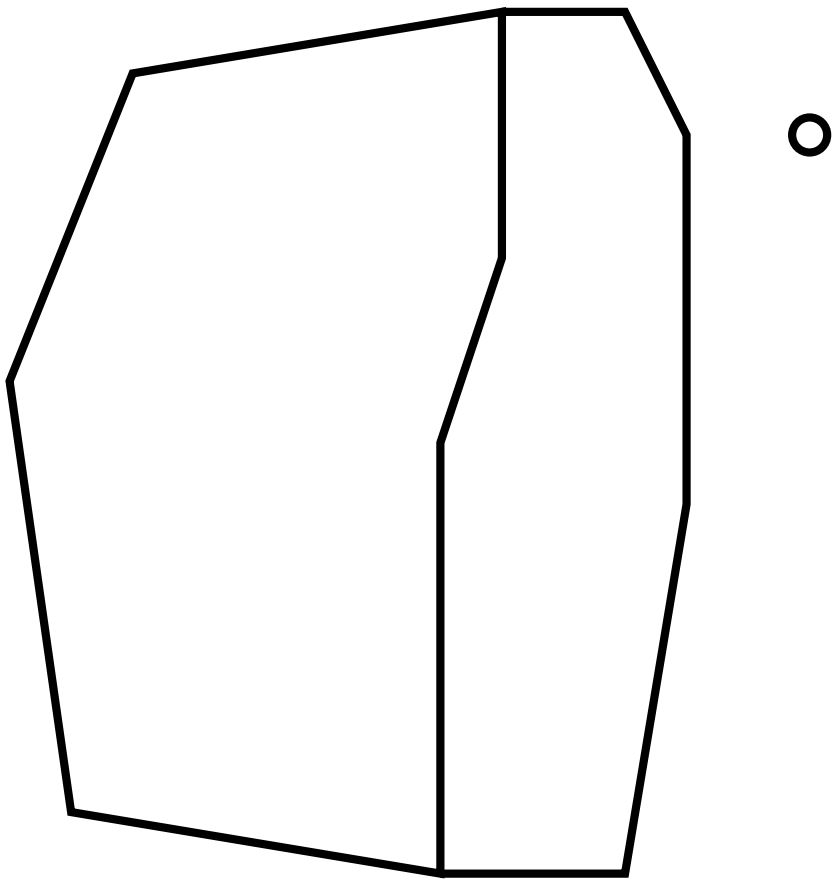
State Space Generation with POR – Demo

Marked states are fully re-expanded.



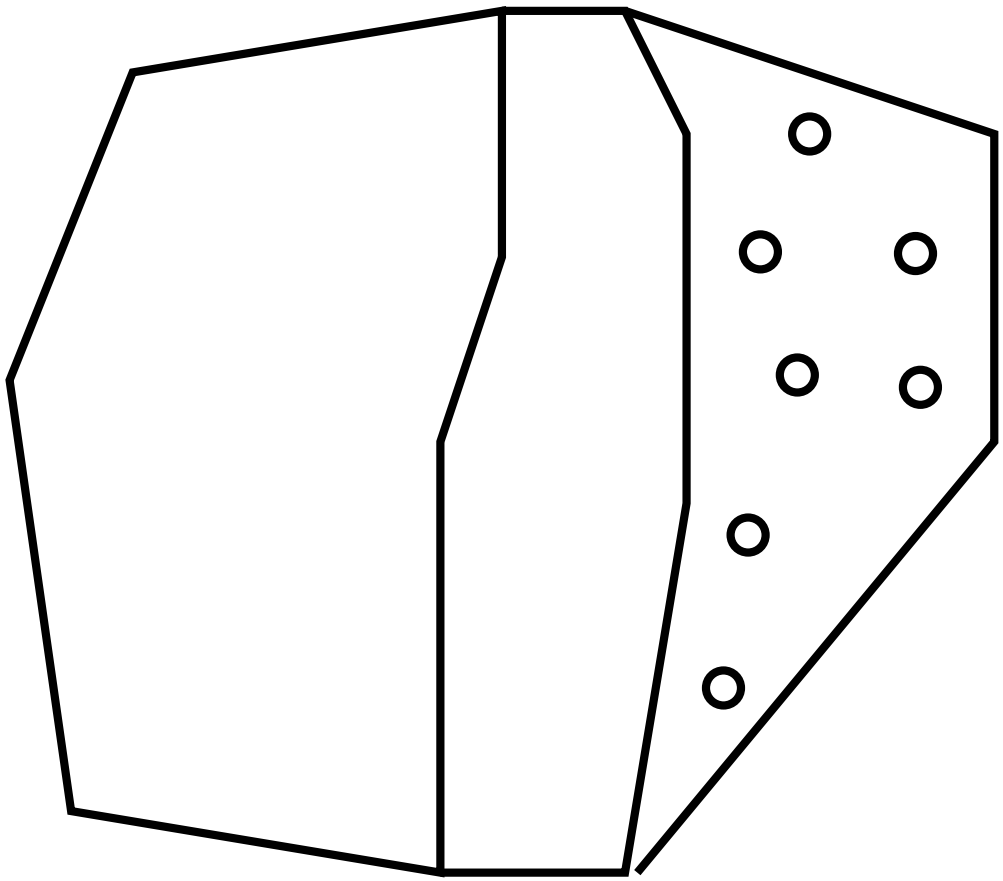
State Space Generation with POR – Demo

New state is discovered outside the previously generated parts of the state space.



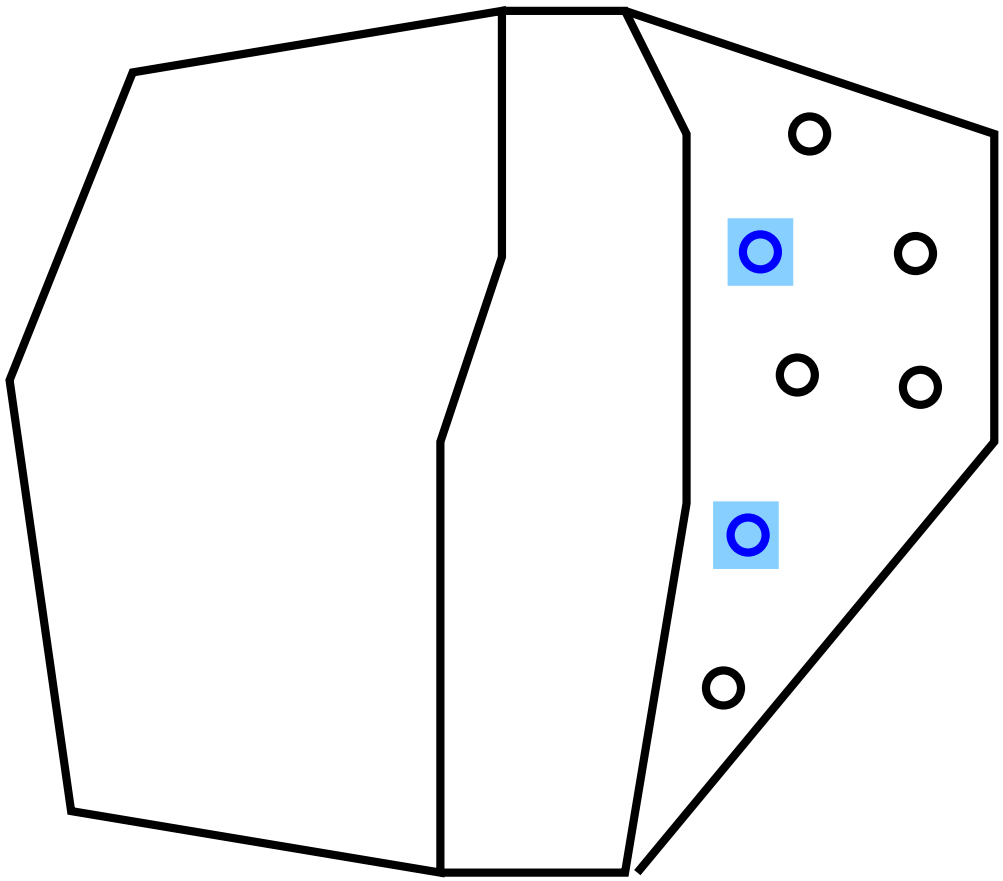
State Space Generation with POR – Demo

Additional part of the reduced state space is generated using action-postponing principle.



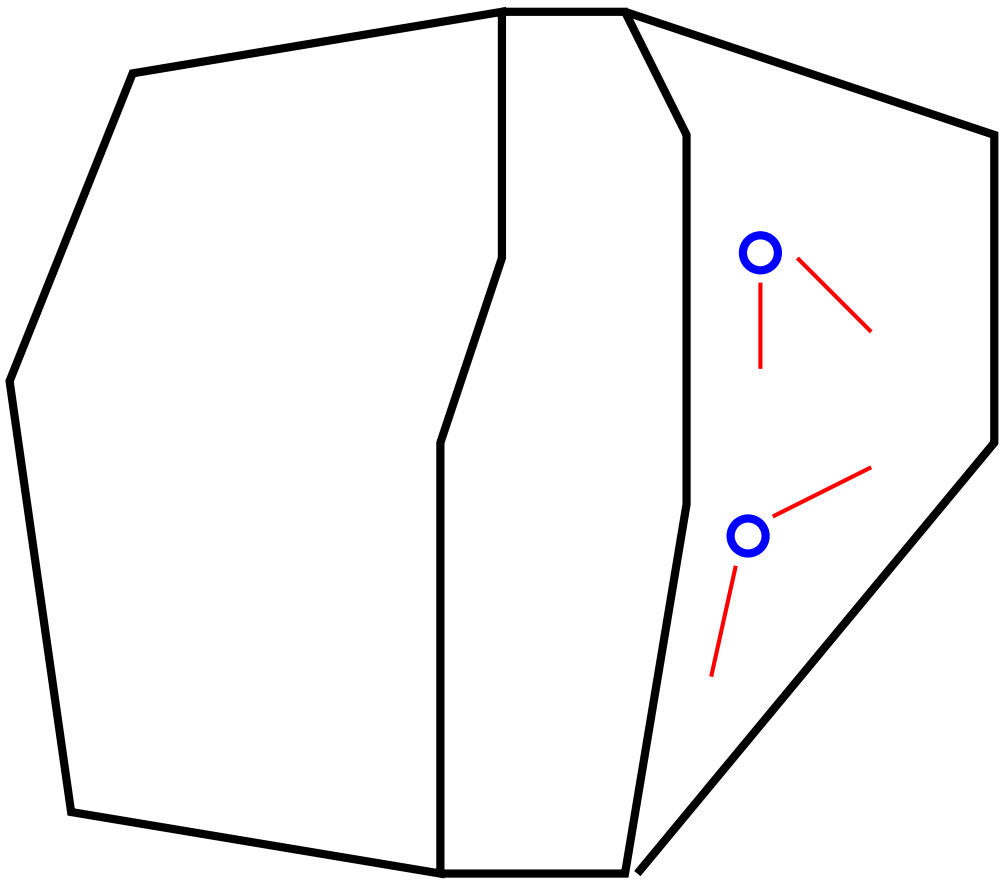
State Space Generation with POR – Demo

Topological sort proviso is applied to mark at least one state on every cycle.



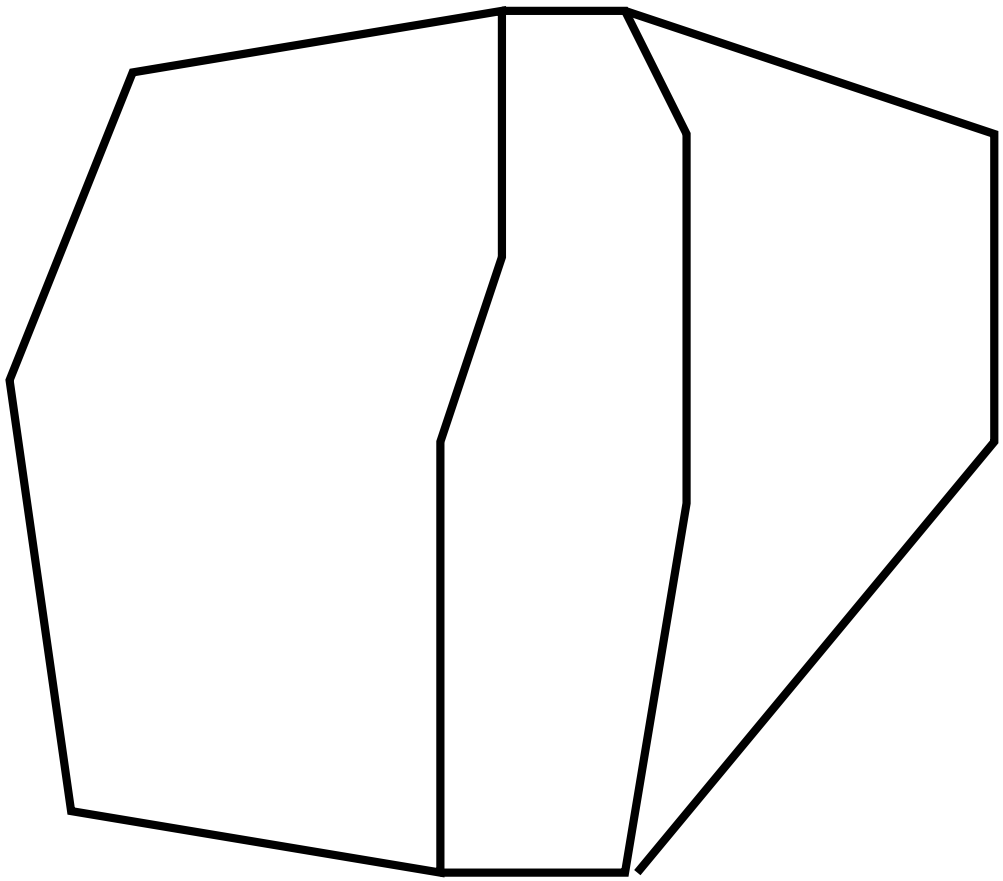
State Space Generation with POR – Demo

Marked states are fully re-expanded.



State Space Generation with POR – Demo

No new states are generated, hence the reduced state space has been fully constructed.



The DiVinE Story!

... algorithm engineering

The DiVinE Story: Algorithm Engineering

Algorithm engineering

"Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice."

[Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science]

The DiVinE Story: Algorithm Engineering

Algorithm engineering

"Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice."

[Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science]

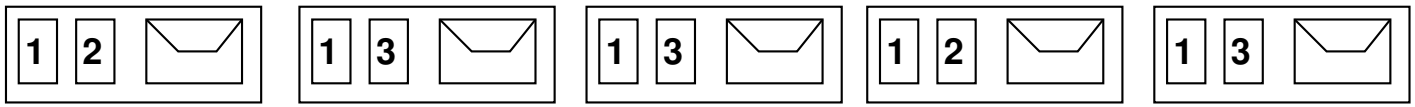
In other words ...

... implementation matters.

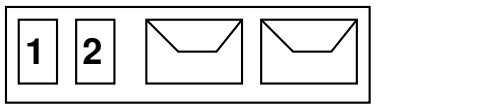
The DiVinE Story: Message Aggregation

Principle

- Individual messages



  are sorted according the target and combined



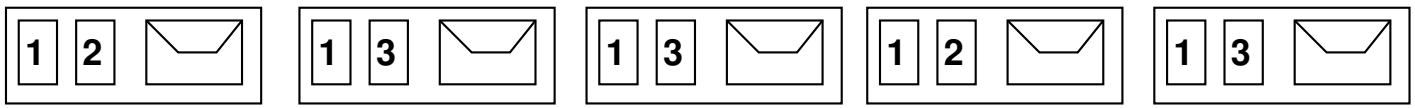
Buffers are flushed when

- explicitly requested by underlying algorithm, or
- maximal number of messages in buffer is reached, or
- node is otherwise idle (empty queue), or
- messages in the buffer are too old.

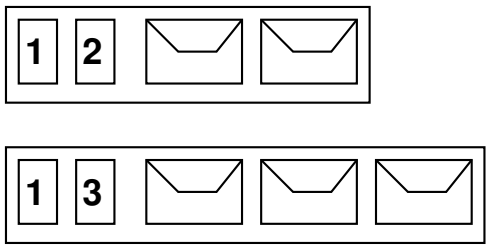
The DiVinE Story: Message Aggregation

Principle

- Individual messages



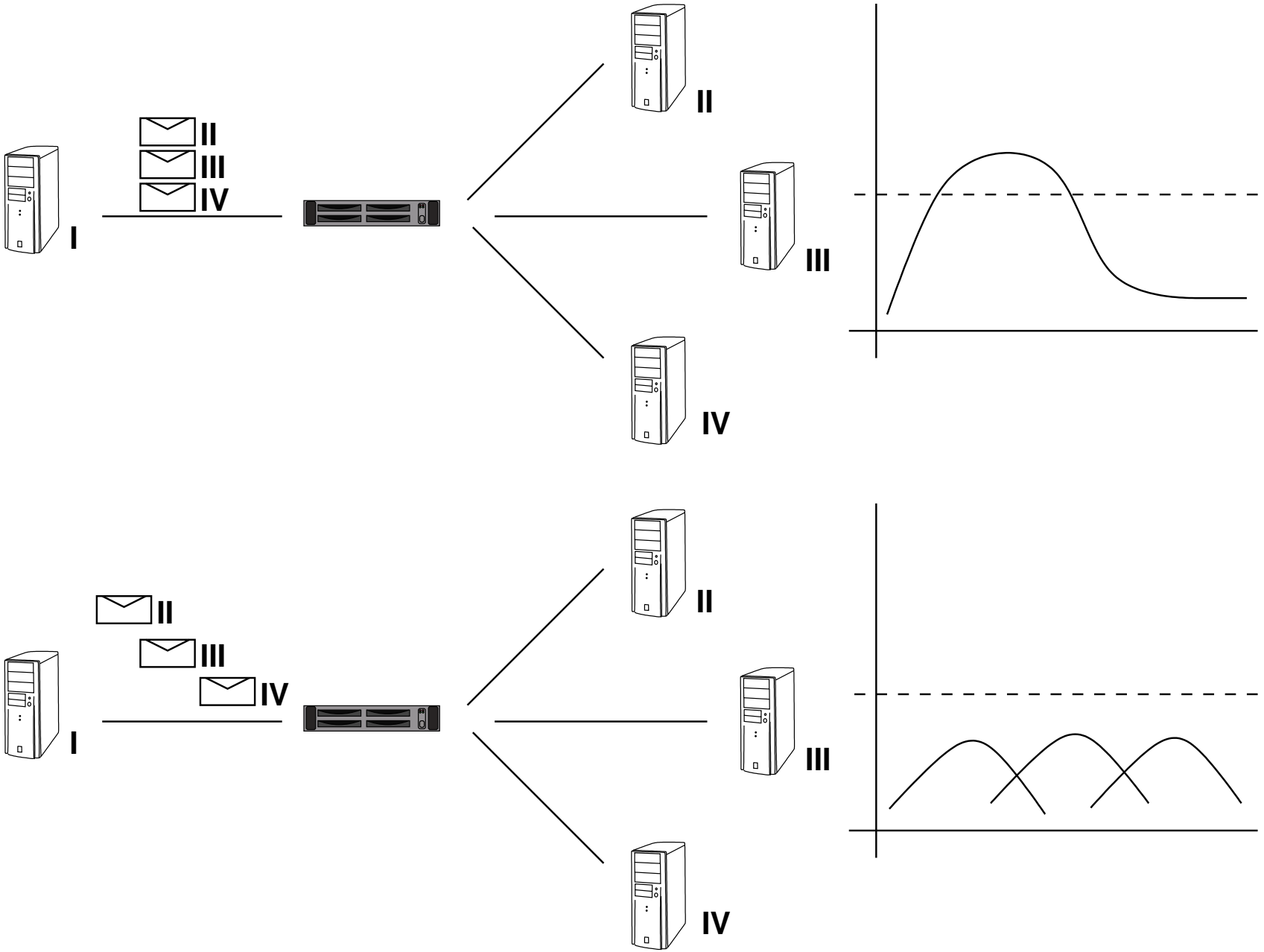
  are sorted according the target and combined



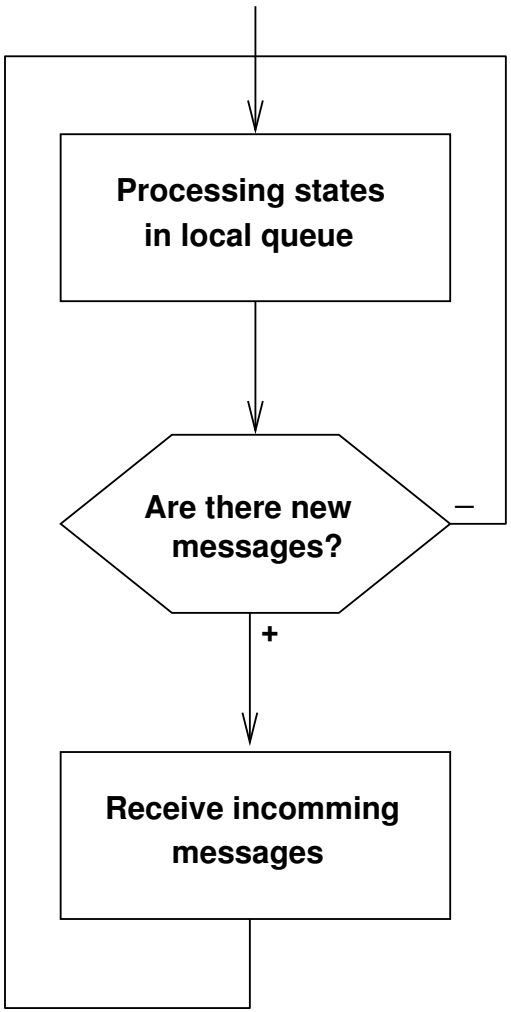
Buffers are flushed when

- explicitly requested by underlying algorithm, or
- maximal number of messages in buffer is reached, or
- node is otherwise idle (empty queue), or
- messages in the buffer are too old. $\Leftarrow$ **expensive and useless**

The DiVinE Story: Flushing Buffers



The DiVinE Story: Polling for incoming messages

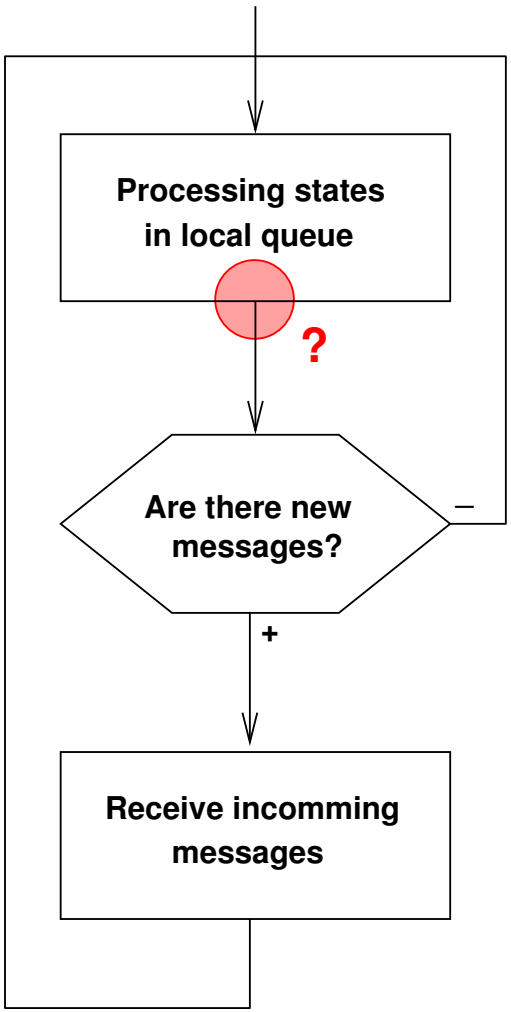


Graph Exploration Loop

- States to be explored locally are taken from a local queue.
- Queue contains locally generated states and states received from network.

When to process incoming messages?

The DiVinE Story: Polling for incoming messages



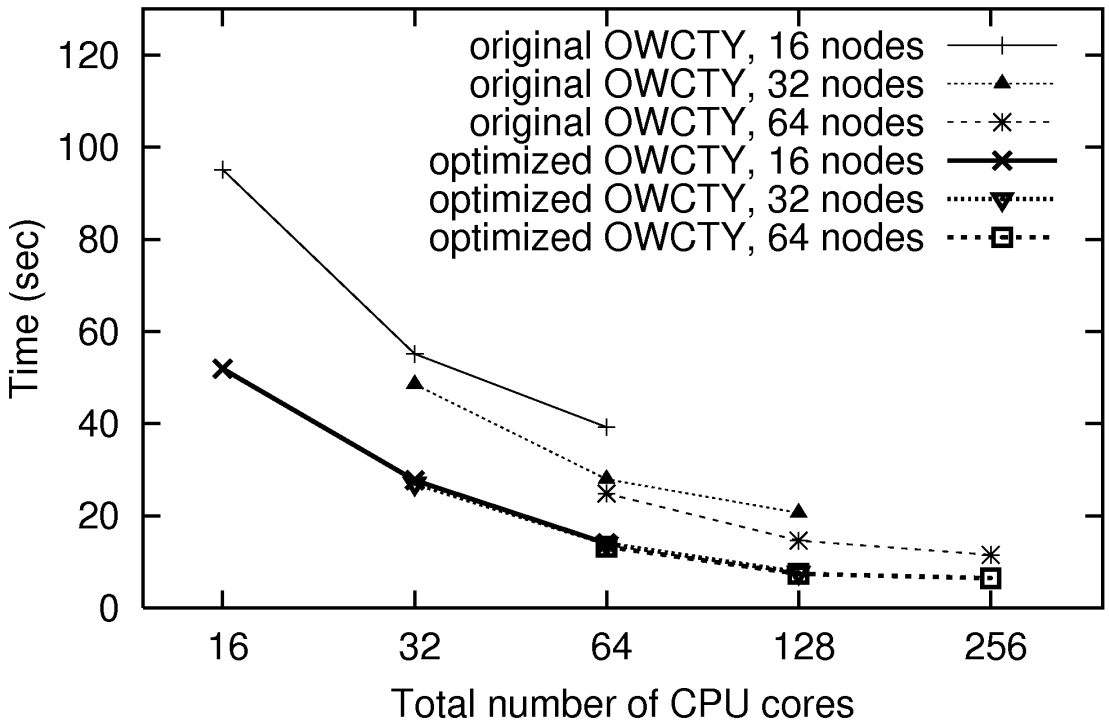
Graph Exploration Loop

- States to be explored locally are taken from a local queue.
- Queue contains locally generated states and states received from network.

When to process incoming messages?

- High rate $\rightarrow$ CPU load.
- Low rate $\rightarrow$ increased memory demands.

The DiVinE Story: Performance of OWCTY Algorithm



In cooperation
with VU Amsterdam.

Cores	Runtime (sec)	Efficiency
1	631.7	100%
64	13.3	74%
128	7.4	67%
256	5.0	49%

The DiVinE Story: Engineering on Other Platforms

Threading in Shared-Memory

- Avoid false sharing.
- Efficient memory allocation and deallocation.
- Lock-free communication data structures.

Shared-Memory Memory Access

- Memory access optimized hash tables.
- Compact data representation.

External Memory (disks)

- Delayed duplicate detection.
- Recompute rather than communicate.

GP GPU Many Cores

- Avoid hash-based graph partitioning.
- LTL model checking as matrix-vector multiplication.

What's Next

... some conclusions

Conclusions

General observation

- Gap between pseudo-code and implementation is widening.
- Implementations need to be tuned for individual platforms.
- We should learn to appreciate engineering solutions.

Explicit state verification future

- Combination with static analysis and symbolic approaches.
- Increased importance of techniques that trade space for time.
- Platform dependent state space reductions.

The End

... thank you for your attention