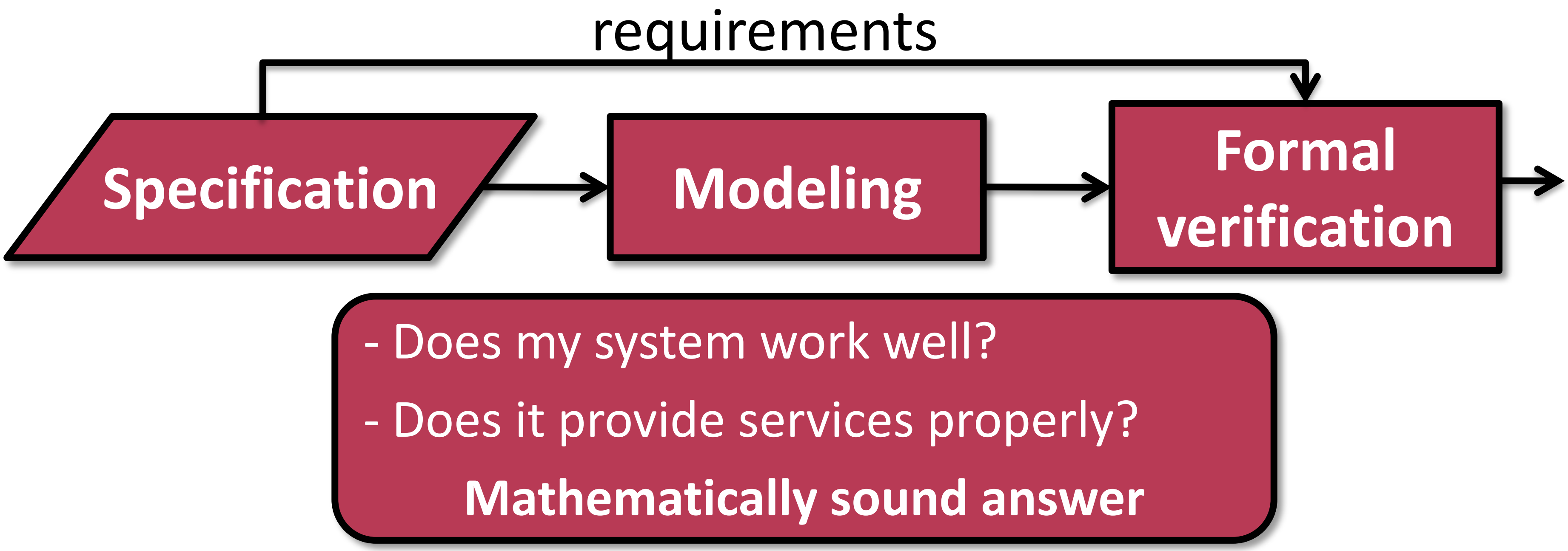
Saturation

András Vörös

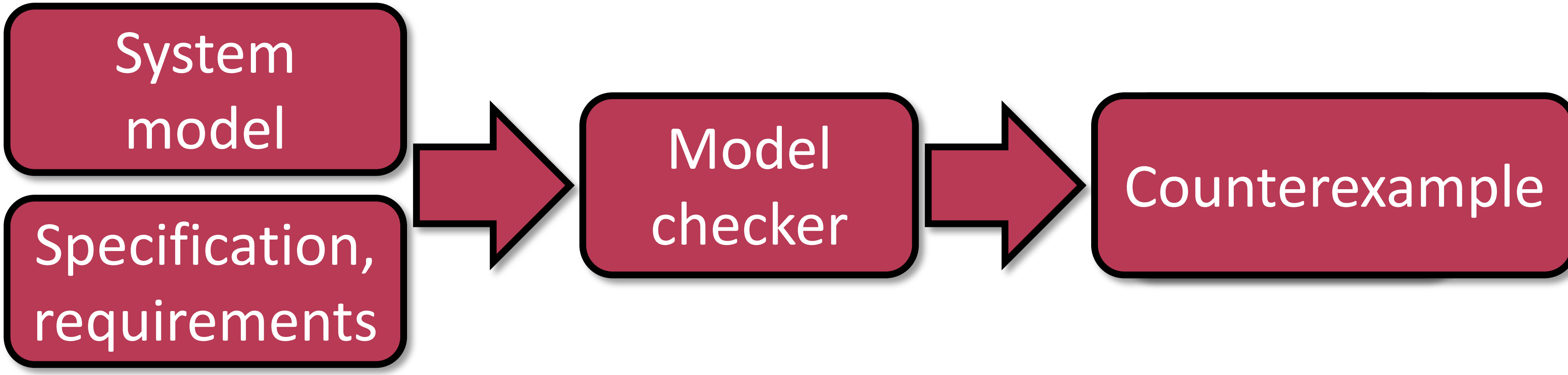
Formal methods

- Safety critical and embedded systems
  - Railway, automotive industry, air transportation
  - Reliability is an important issue
- Design time analysis
  - These models can be used for implementation



Model checking

- Automatic verification method



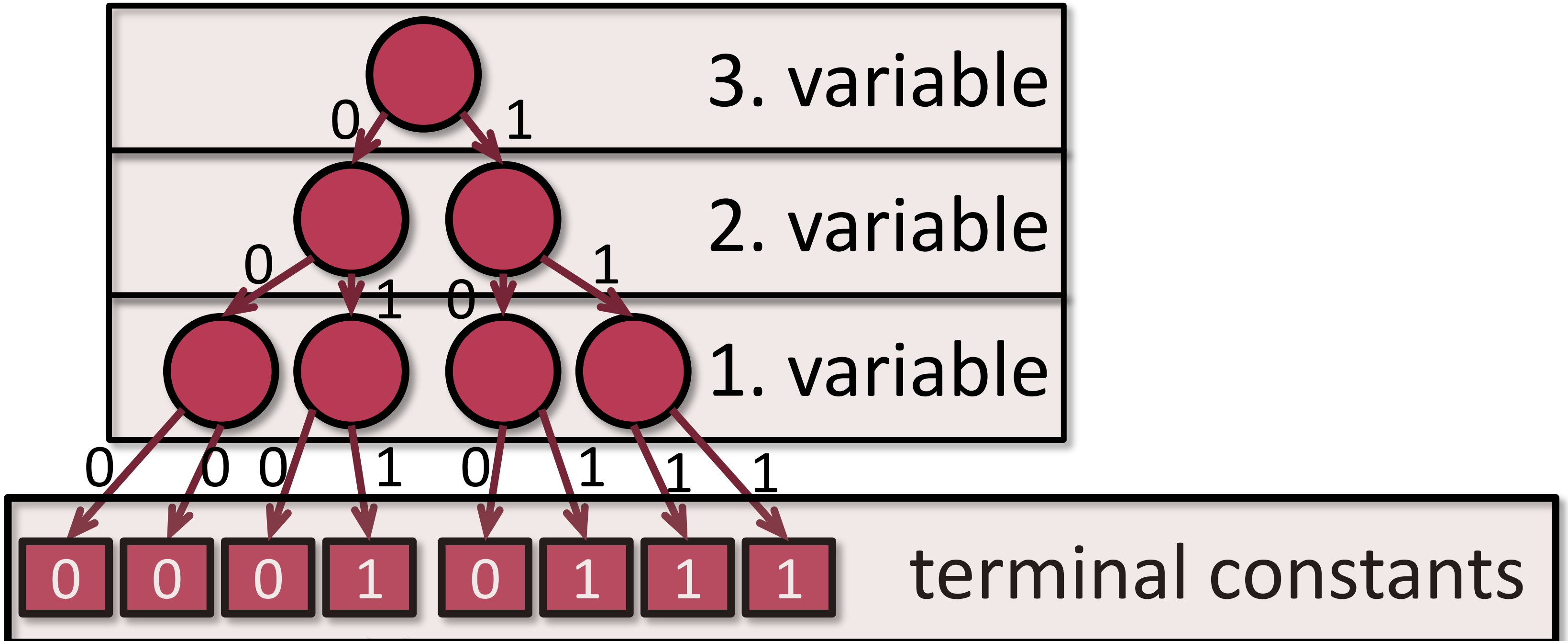
- Prerequisite:
  - Exploring and representing the reachable states
- Problem:
  - State space explosion
  - Time and space requirements

Saturation algorithm

- Efficient solution for:
  - State space generation
  - Model checking
- Symbolic algorithm
  - Encoding of states
  - Special underlying data structures
    - Multi Valued Decision Diagrams (MDD-s)
- Special iteration strategy
  - Efficient for asynchronous models

Multi Valued Decision Diagrams

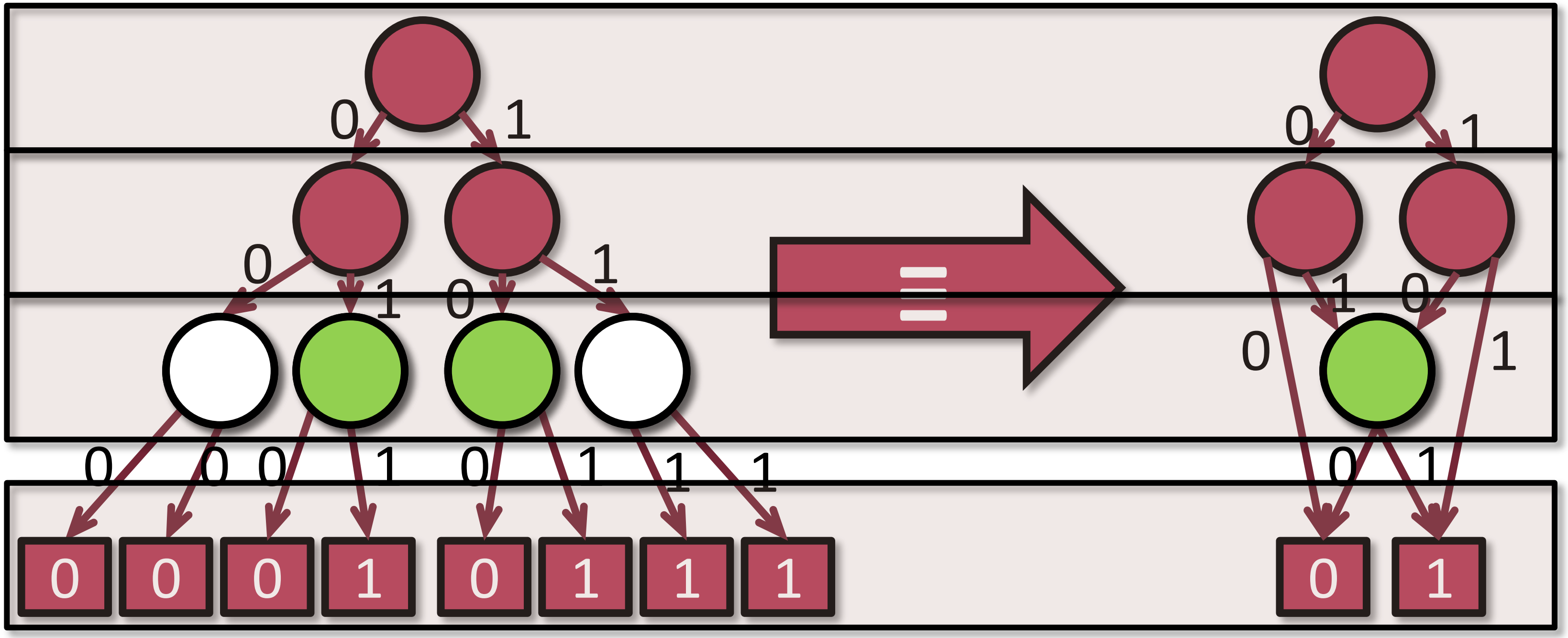
- Derived from decision trees
  - variables are ordered into levels
- Example:
  - only binary variables



Decision tree

Multi Valued Decision Diagrams

- Derived from decision trees
  - variables are ordered into levels
- Special reduction rules
  - in a bottom-up fashion, applying reduction from level-to-levels
- Compact representation of multi valued functions

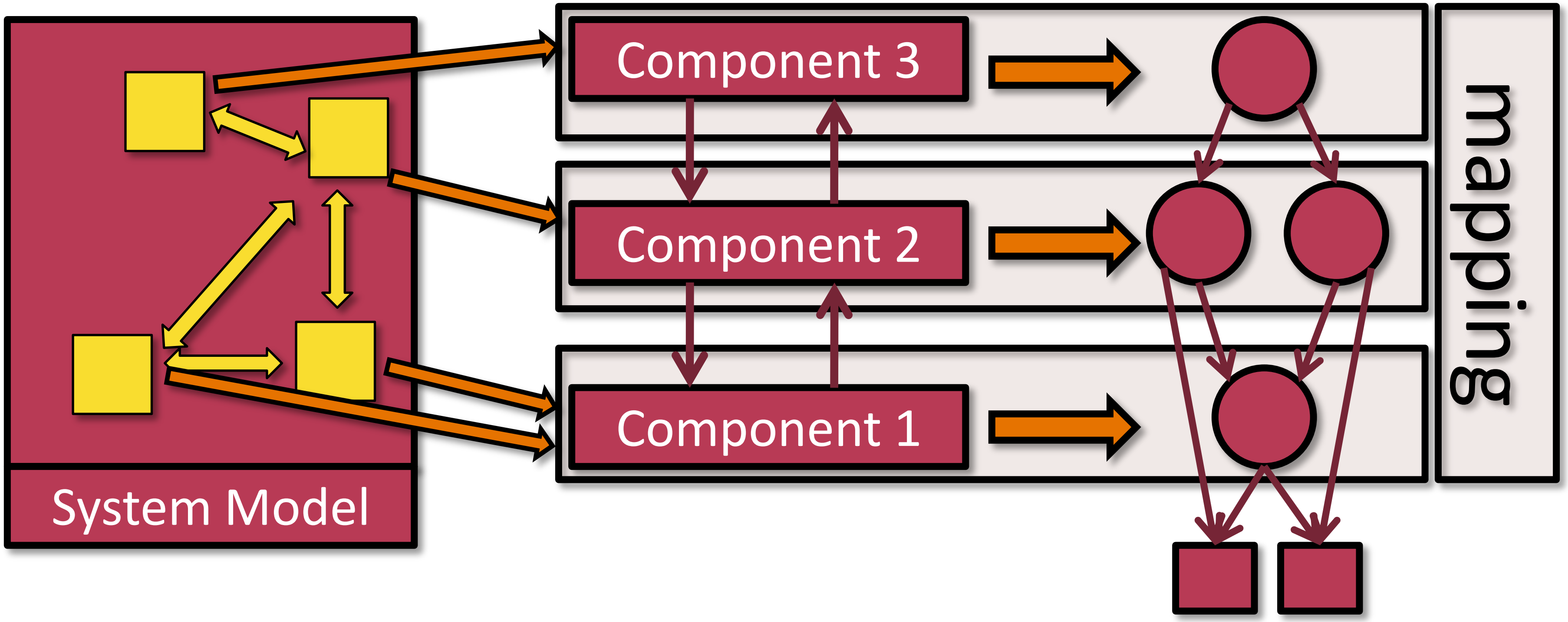


Decision tree

MDD

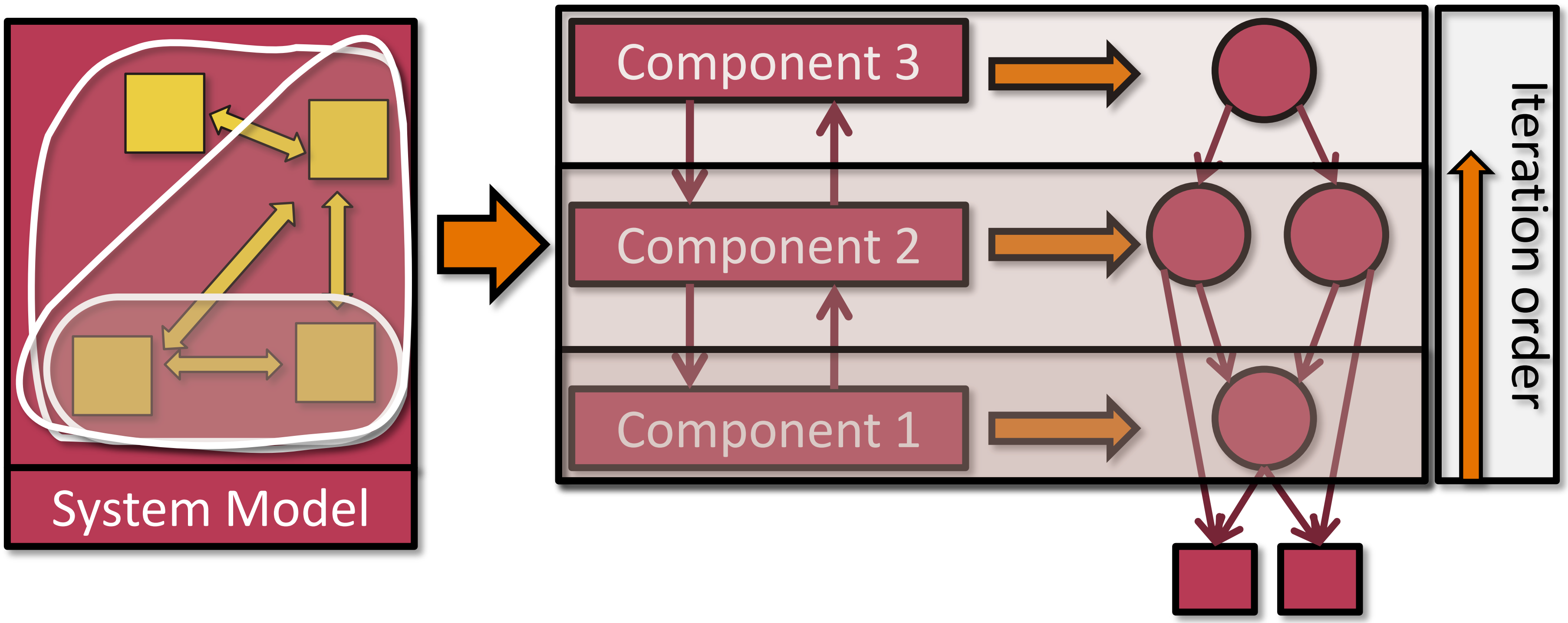
Symbolic algorithm

- Symbolic encoding instead of explicit state representation
  - Decomposition is needed
- Saturation uses component wise encoding

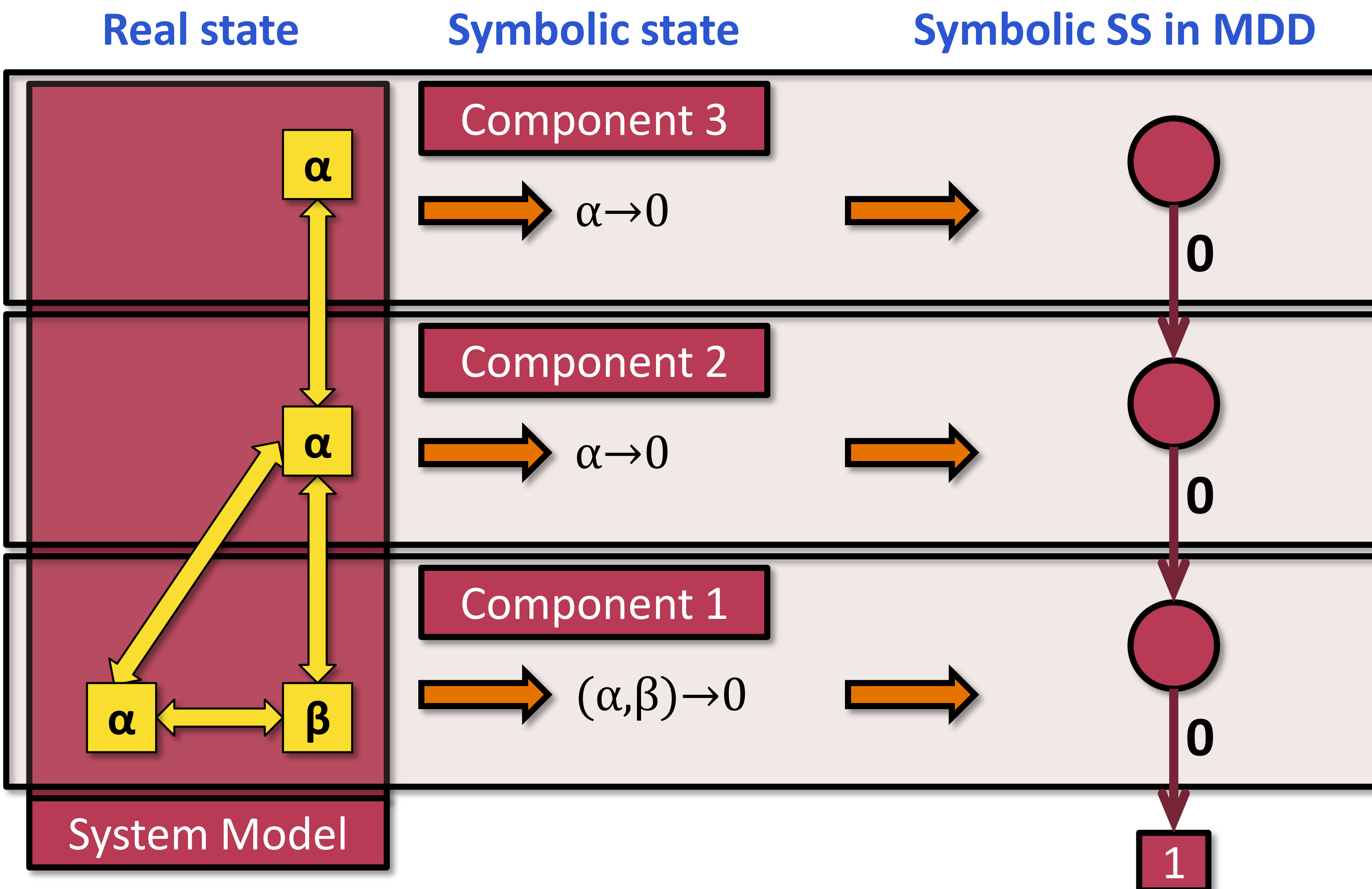


Special iteration

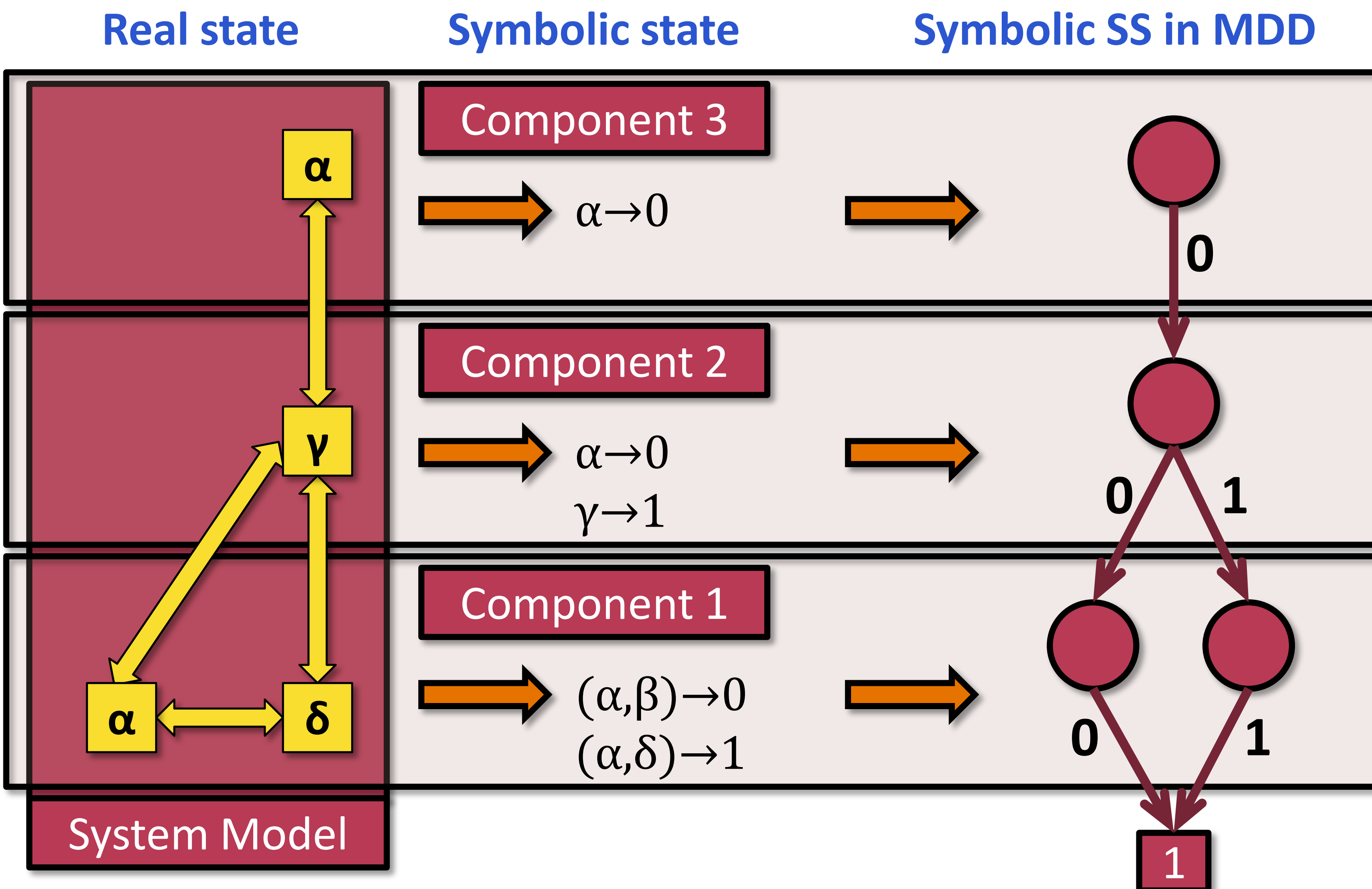
- Local exploration in a greedy manner
- Exploring global synchronization events if needed
- Uses the primarily defined order of the decision diagram variable encoding
- Efficient for Globally Asynchronous, Locally Synchronous models (GALS)



State space representation with MDDs

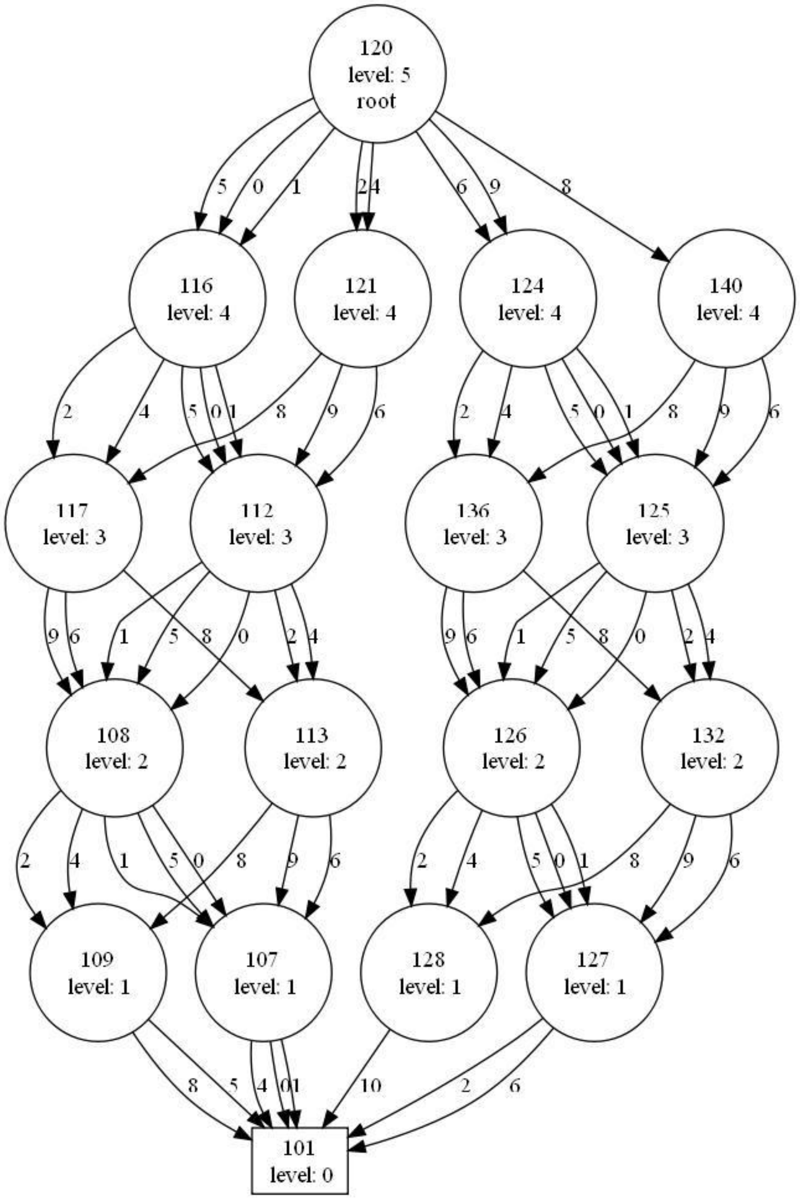


State space representation with MDDs



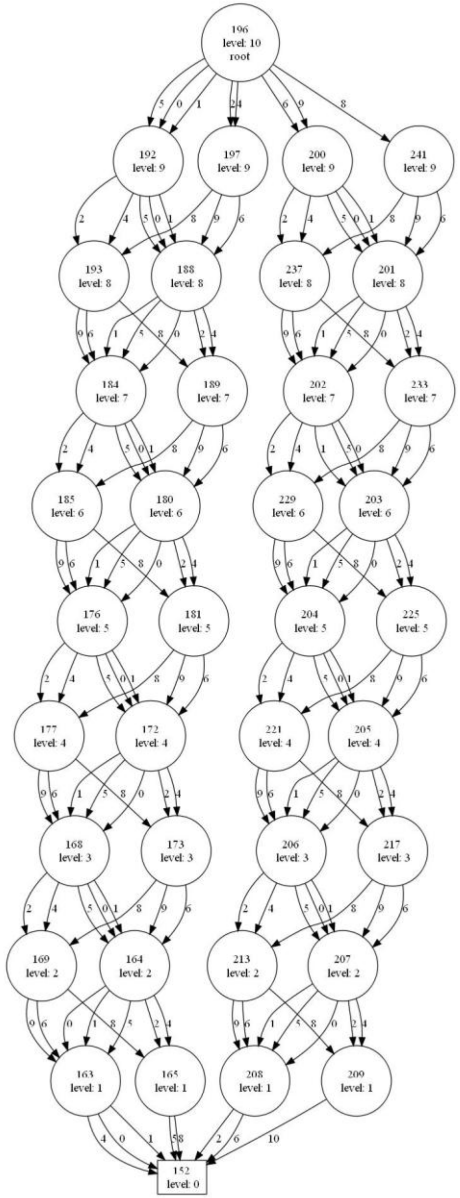
Experiments

- Dining philosophers
  - 5 philosophers
- State space representation
  - 1364 states
  - 19 nodes



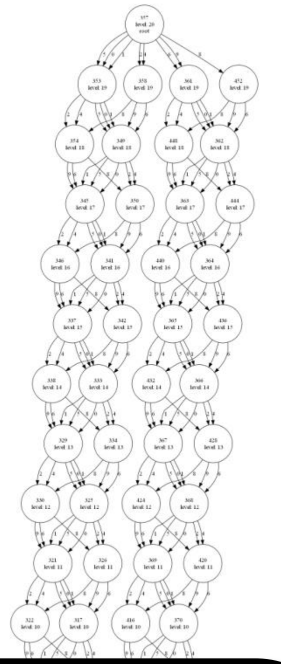
Experiments

- Dining philosophers
  - 10 philosophers
- State space representation
  - 1,860,498 states
  - 40 nodes



Experiments

- Dining philosophers
  - 20 philosophers
- State space representation
  - 3,461,452,808,002 states
  - 80 nodes



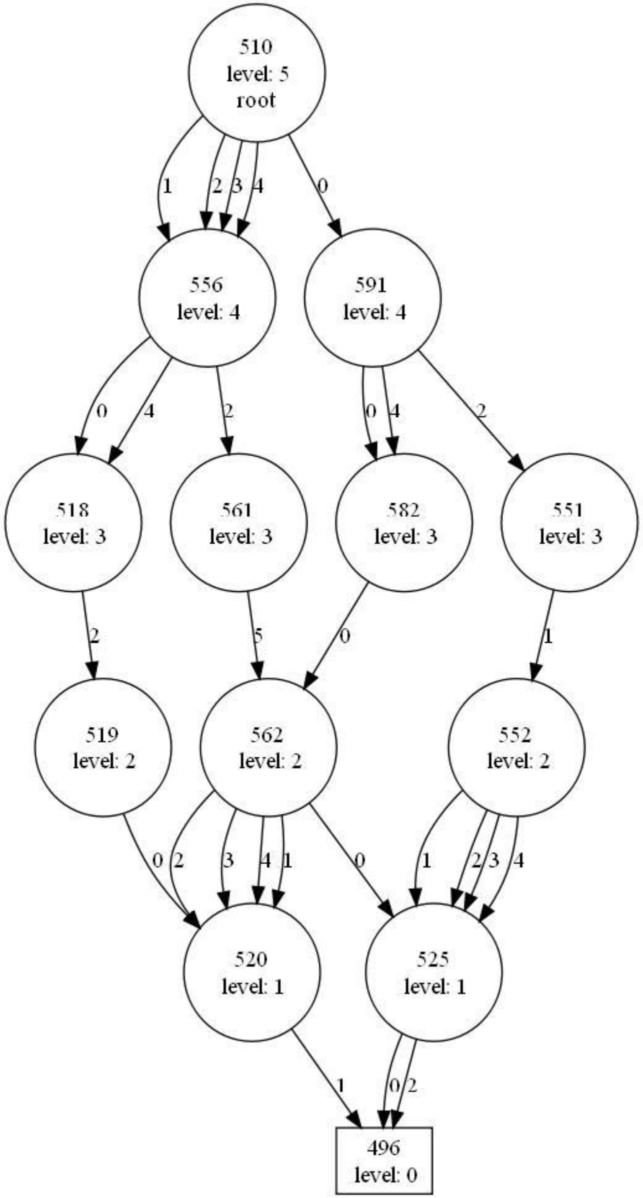
Exponential growth in the state space

⇕

Linear growth in the state space representation

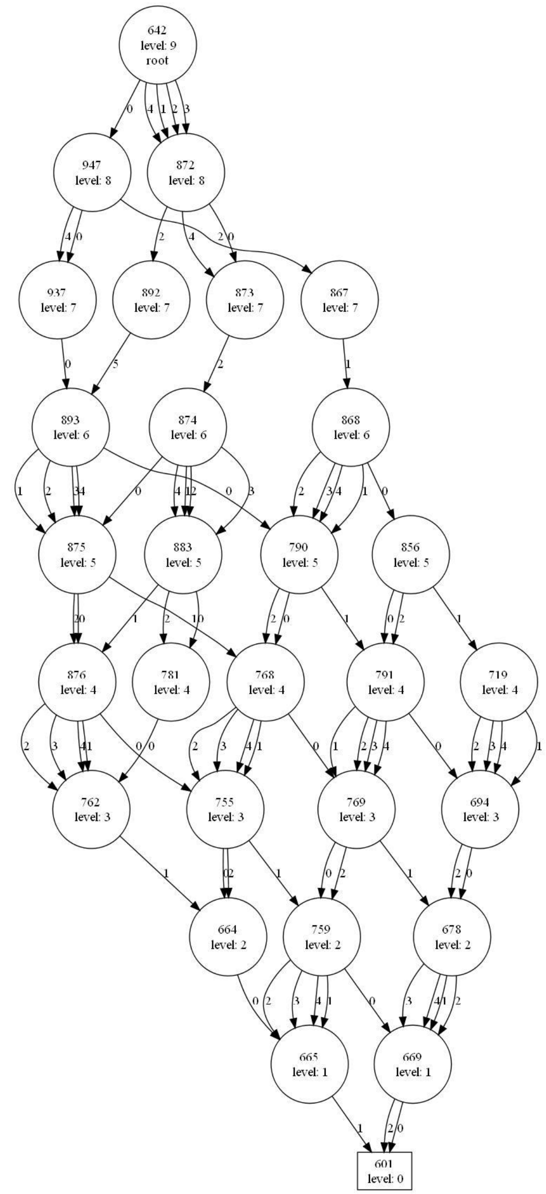
Experiments

- Slotted Ring communication protocol
  - 2 slots
- State space representation
  - 52 states
  - 14 nodes



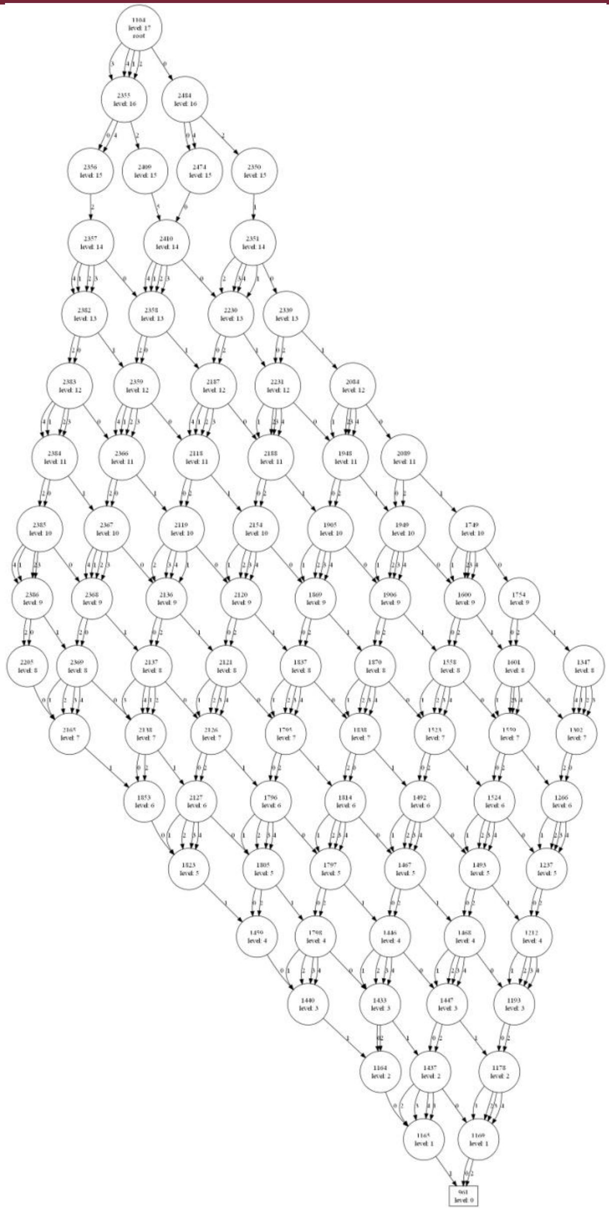
Experiments

- Slotted Ring communication protocol
  - 4 slots
- State space representation
  - 5136 states
  - 30 nodes



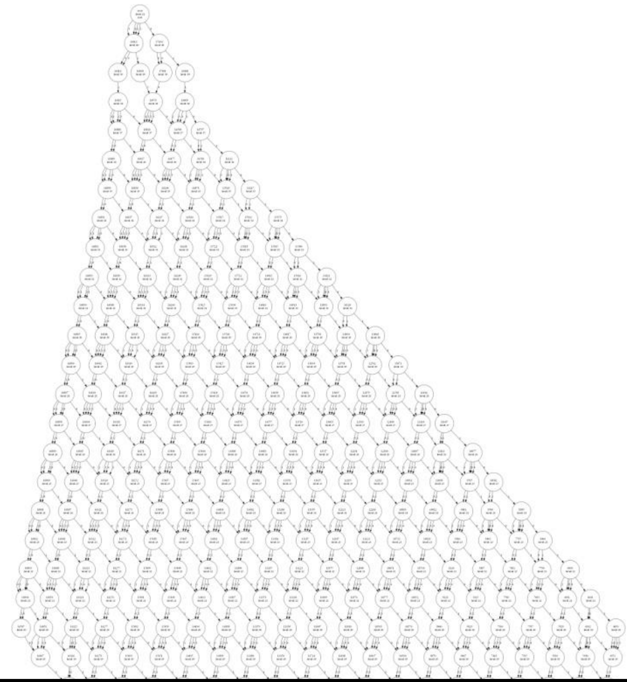
Experiments

- Slotted Ring communication protocol
  - 8 slots
- State space representation
  - 68,026,624 states
  - 103 nodes



Experiments

- Slotted Ring communication protocol
  - 20 slots
- State space representation
  - 10^{20} states
  - 487 nod

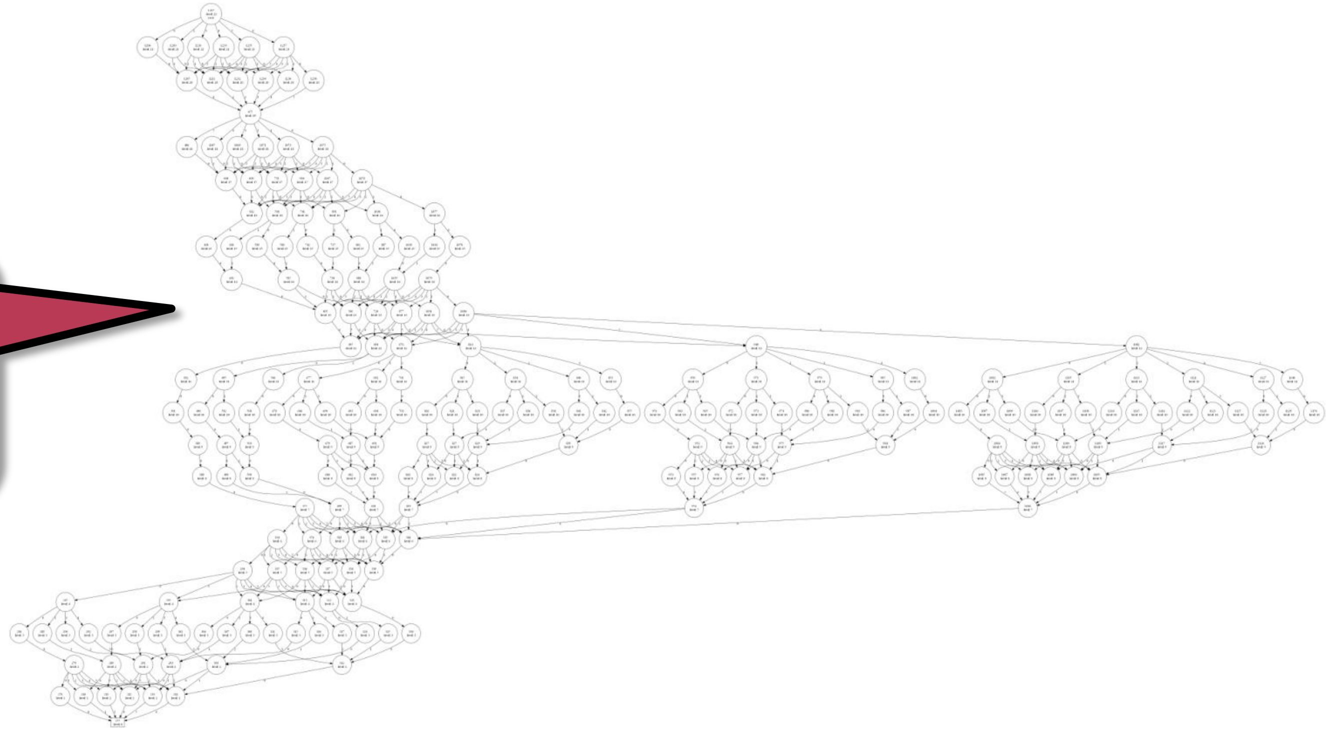


Scales up to about 200 slots in the ring and about 10^{200} states

(60902 nodes in the state space MDD, the full state space generation lasted 222 seconds long)

Experiments

- Flexible manufacturing system
  - 5 item
- State space representation
  - about 2,900,000 states
  - 248 nodes



Not so nice, but still efficient ☺

Experiments

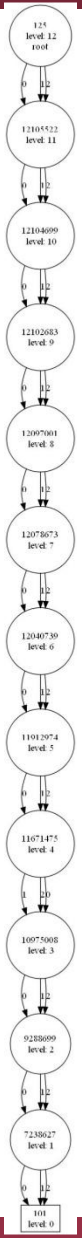
- Tower of Hanoi game
  - It consists of three rods, and a number of disks of different sizes which can slide onto any rod.
  - Rules:
    - Only one disk may be moved at a time
    - Each move consists of taking the upper disk from one of the rods and sliding it onto another rod, on top of the other
    - No disk may be placed on top of a smaller disk

Synchronous model:

- at most 4 transitions are enabled from each state

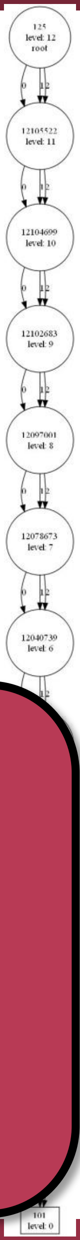
Experiments

- Tower of Hanoi game
  - 12 disks
- State space representation
  - 531 441 (3^{12}) states
  - 12 nodes
- **Unfortunately**:
  - during the exploration we construct more nodes
  - the state space generation took 58 seconds
  - huge number of transitions in the model



Experiments

- Tower of Hanoi game
  - 12 disks
- State space representation
  - 531 441 (3^{12}) states
  - 12 nodes
- **Unfortunately:**
  - d
    n
  - th
  - h



Conclusion:

Efficient state space representation

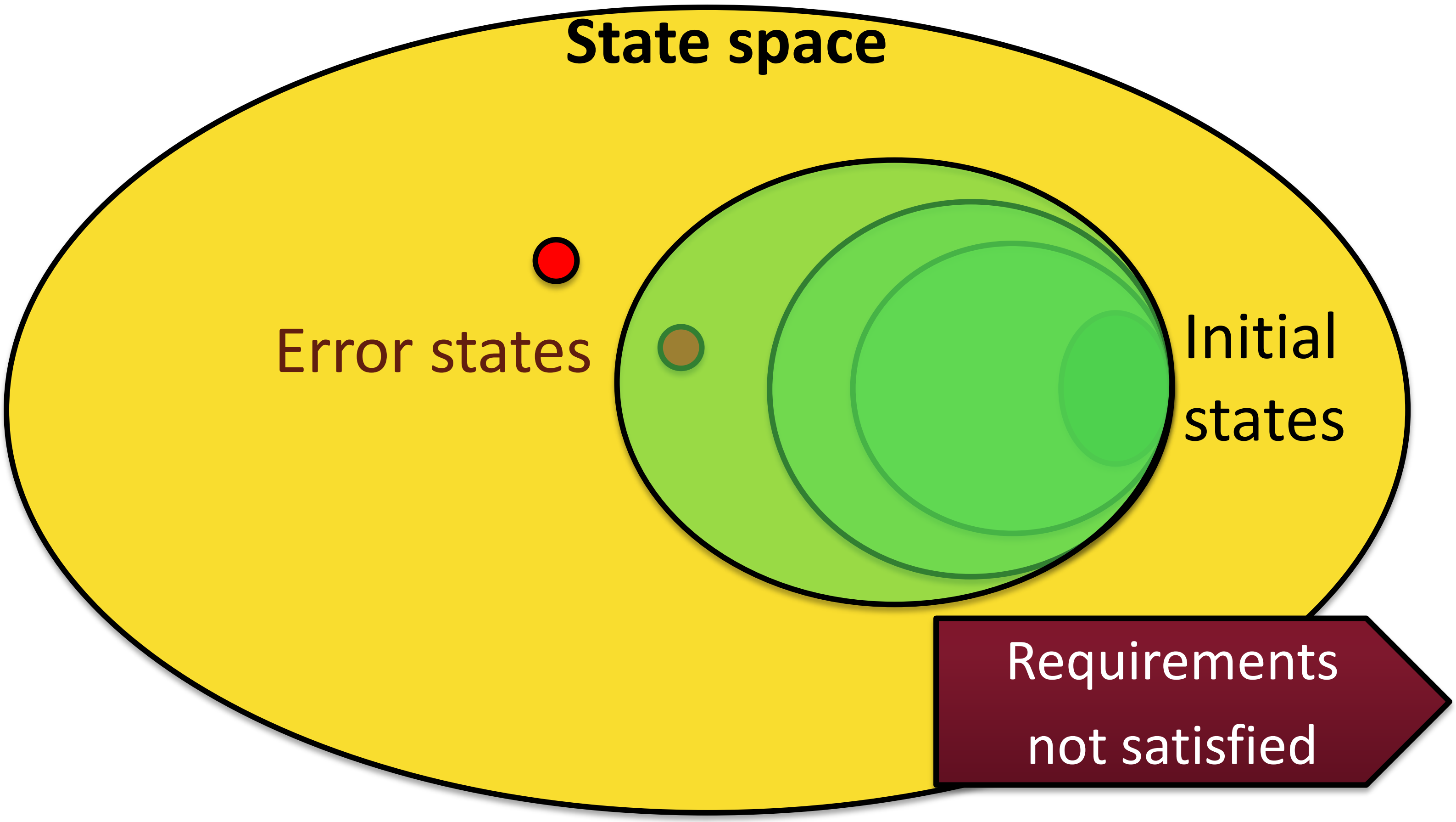
Efficient iteration

- For asynchronous models -

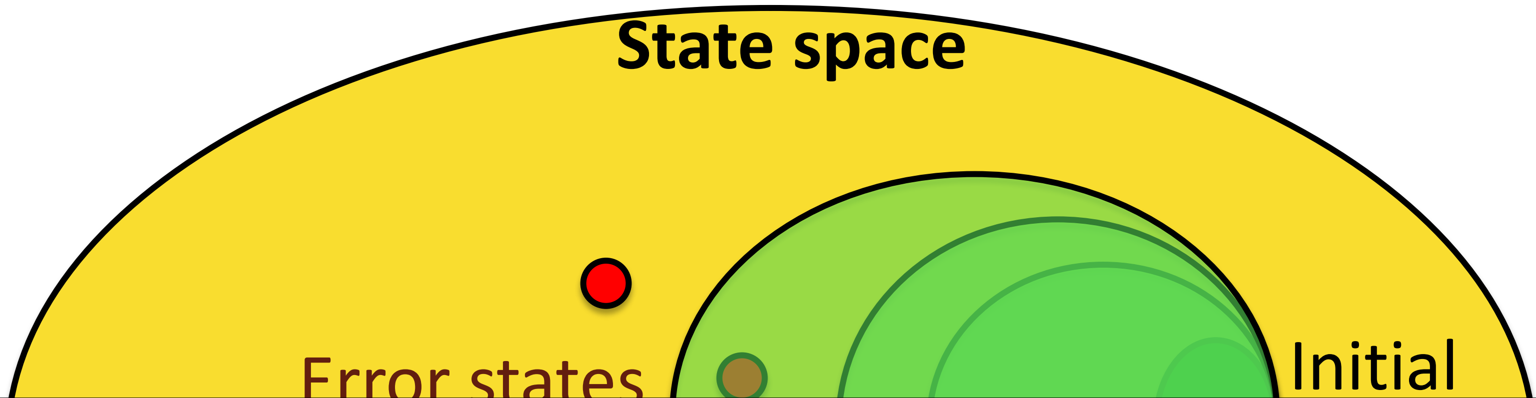
Problems

- Efficiency of the algorithm highly depends:
  - Decomposition
  - Variable ordering
    - Bottleneck of symbolic methods
- Best performance if this information is provided manually

Motivation for bounded model checking



Motivation for bounded model checking



Bounded model checking (BMC)

- explores a *k*-bounded part of the state space (usually in a breadth first manner)
- examines the specification on this smaller part

Bounded Saturation

- Bounded model checking
  - explores a k-bounded part of the state space
    - usually in a breadth first manner
  - examines the specification on this smaller part
- Saturation
  - Explores the state space in an irregular recursive order
  - Difficult to bound the exploration
  - There is no distance information in the MDD-s

Bounded Saturation

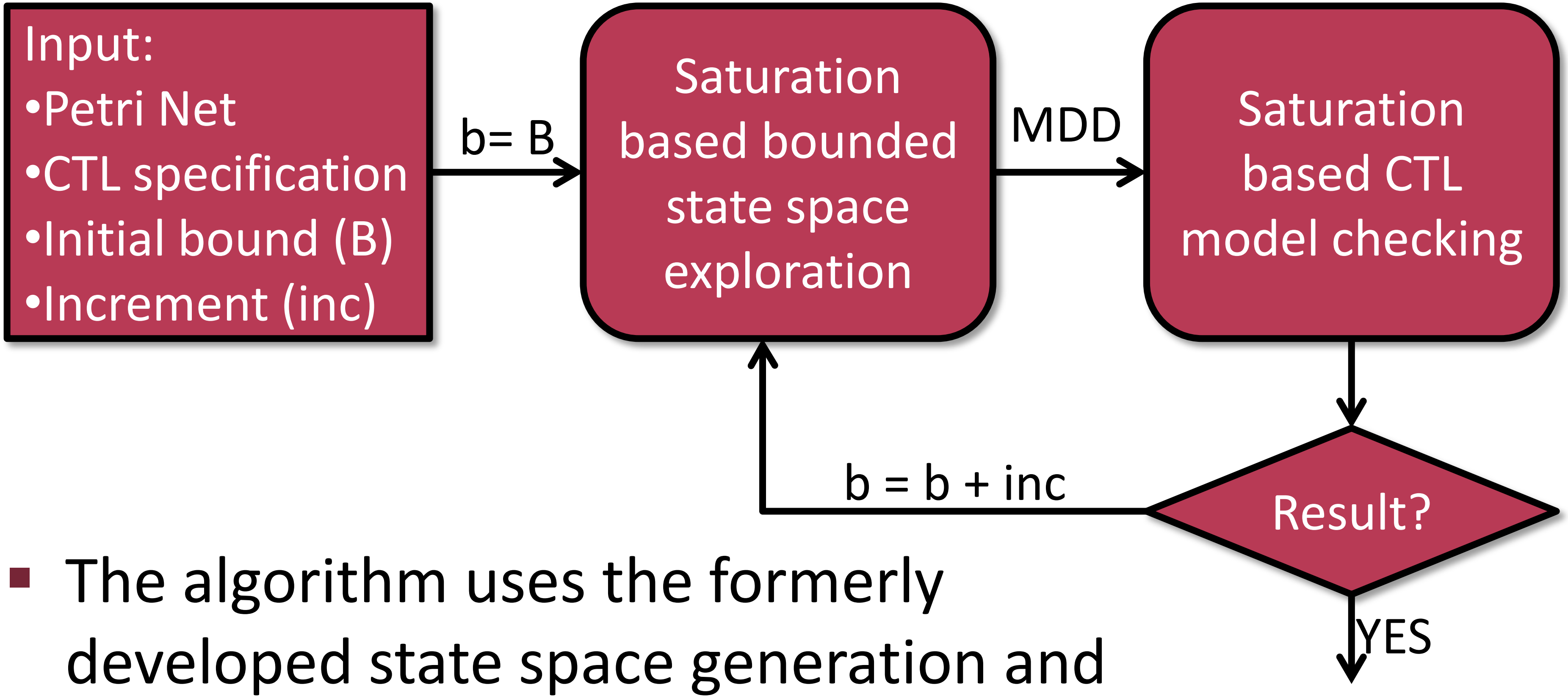
- Bounded model checking
  - explores a k-bounded part of the state space
    - usually in a breadth first manner
  - examines the specification on this smaller part
- Saturation
  - Explores the state space in an irregular recursive order
  - D
  - T

New data structure:

Edge Valued Decision Diagrams (EDDs)

- MDD based data structure enriched with distance information

Saturation Based bounded model checking



- The algorithm uses the formerly developed state space generation and CTL model checking algorithms
- Efficient for some models and specifications

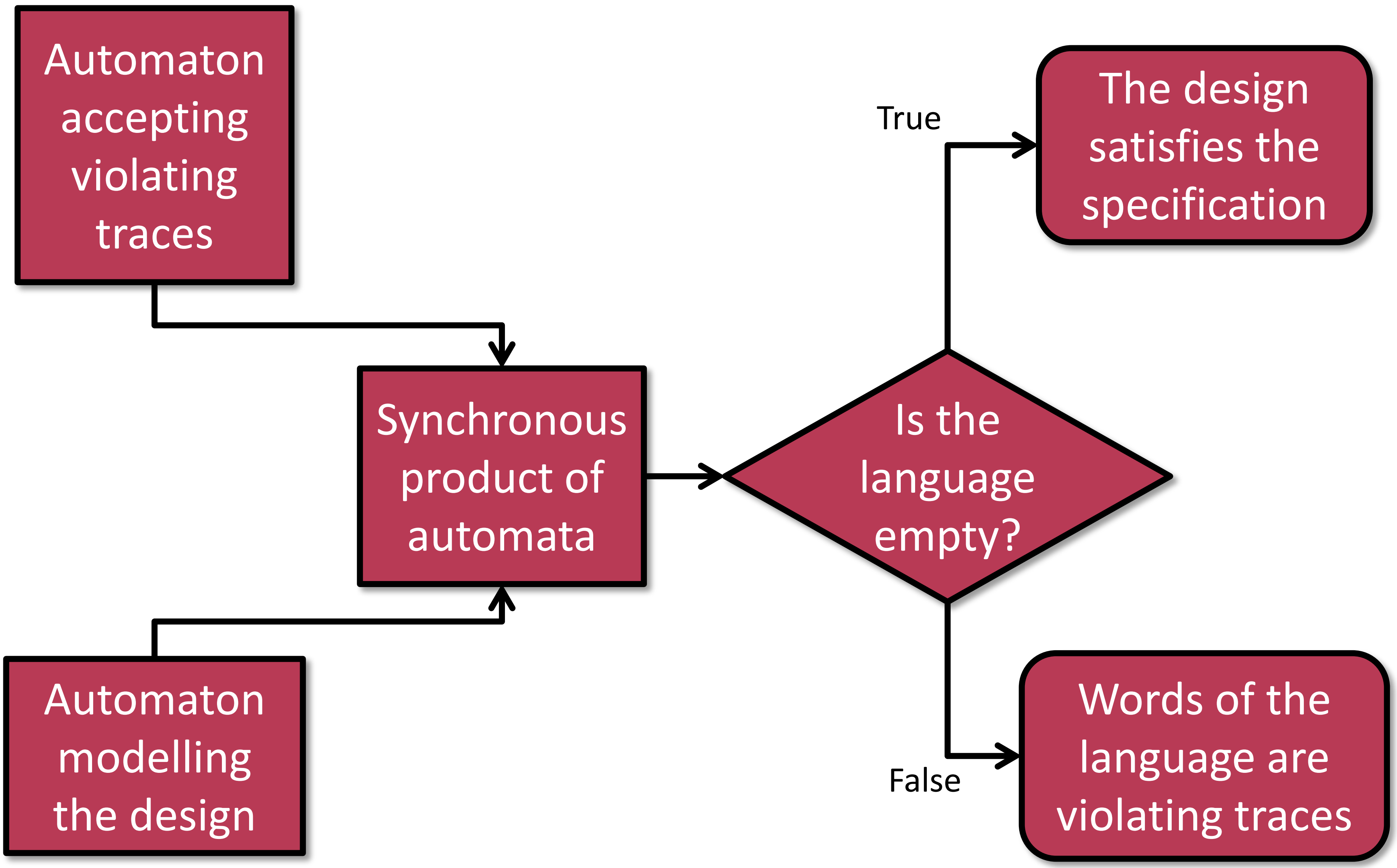
References

- G. Ciardo, R. Marmorstein, and R. Siminiceanu. The saturation algorithm for symbolic state-space exploration. Int. Journal Software Tools for Technology Transfer, 8(1):4–25, 2006.
- G. Ciardo and R. Siminiceanu. Structural symbolic CTL model checking of asynchronous systems. In Computer Aided Verification (CAV'03), LNCS 2725, pages 40–53. Springer-Verlag, 2003.
- A. Yu, G. Ciardo, and G. Lüttgen. Decision-diagram-based techniques for bounded reachability checking of asynchronous systems. Int. J. Softw. Tools Technol. Transf., 11:117–131, February 2009.
- Y. Zhao and G. Ciardo. Symbolic CTL model checking of asynchronous systems using constrained saturation. In Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis, ATVA '09, pages 368–381 2009. Springer-Verlag.
- A. Vörös, T. Bartha, D. Darvas, T. Szabó, A. Jámbor, and Á . Horváth. Parallel saturation based model checking. In ISPDC, Cluj Napoca, 2011. IEEE Computer Society

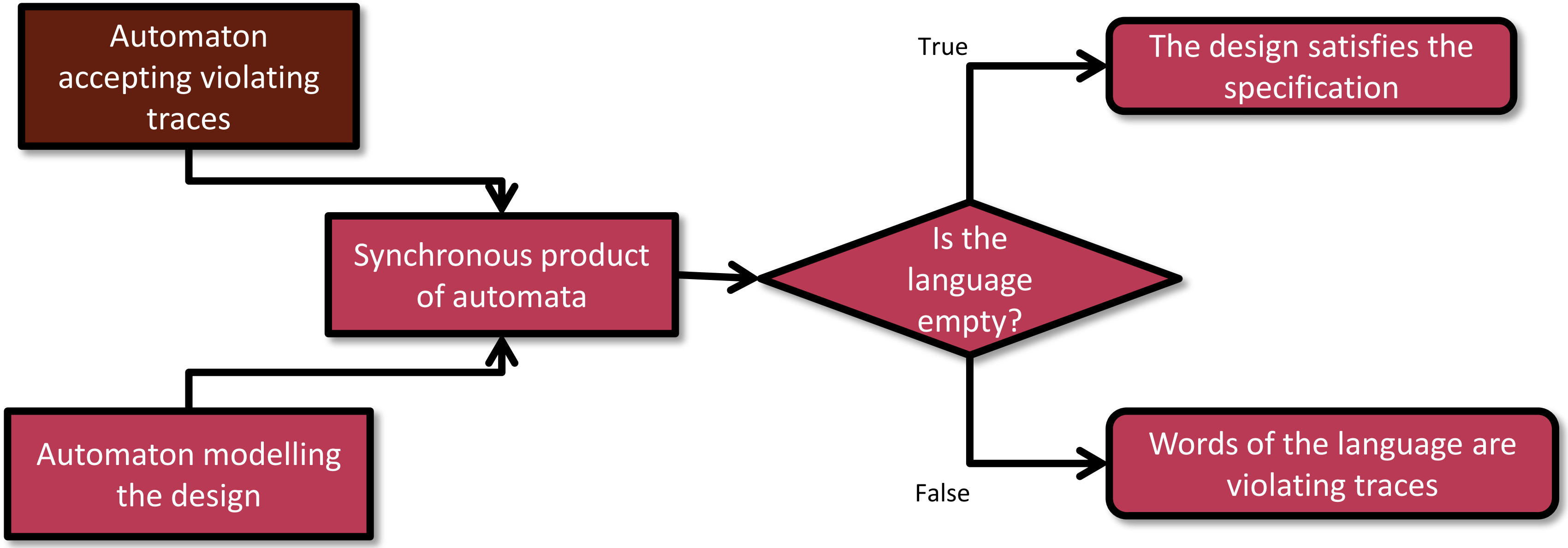
Automata theoretic model checking

Verifying ω-regular properties

Traditional approach

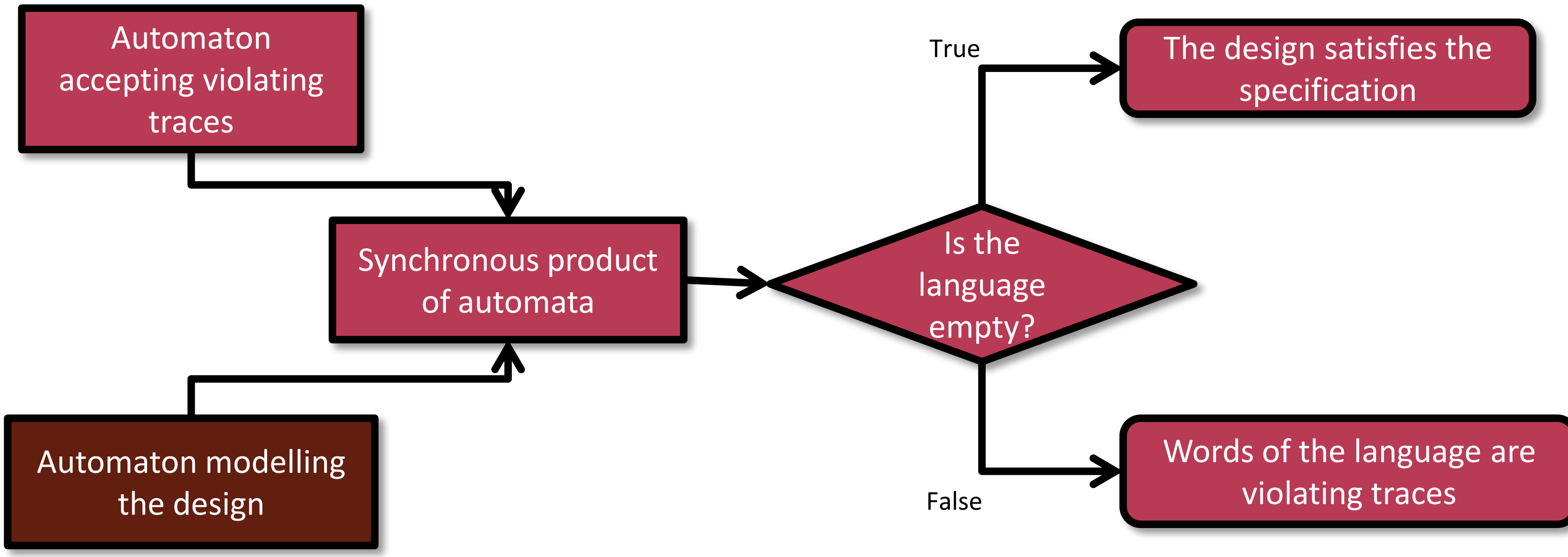


Detailed steps



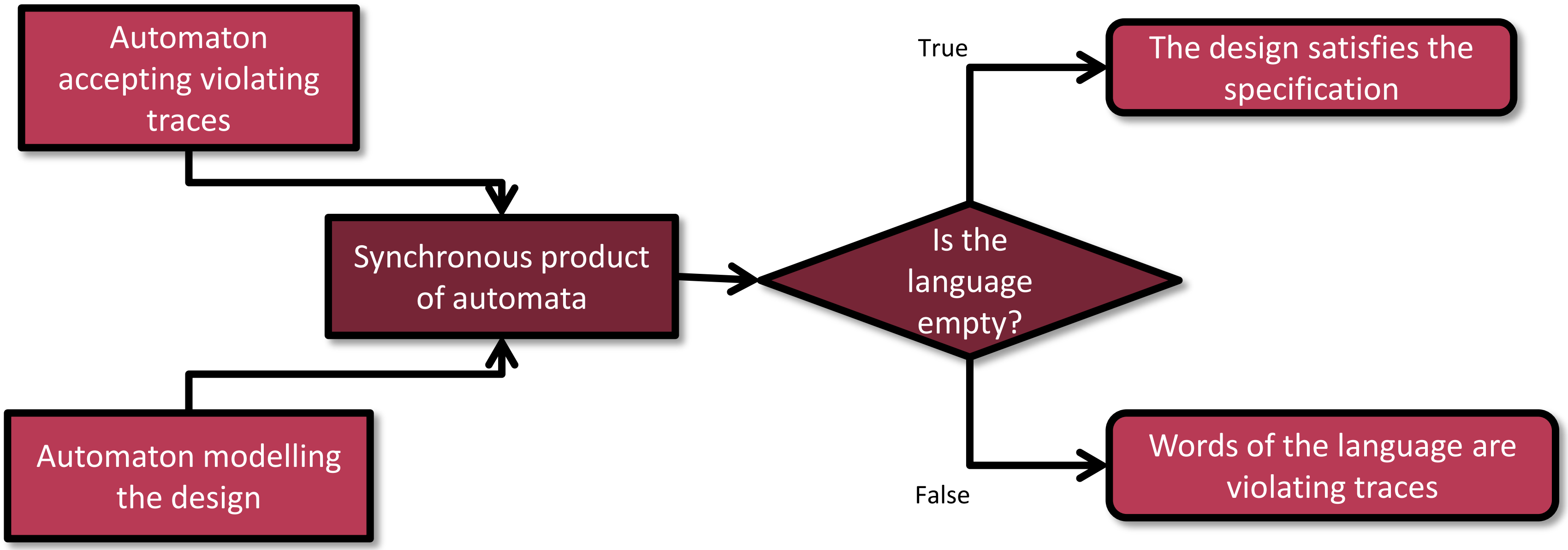
- Can be obtained by:
  - Complementing the Büchi-automaton describing the specification (not always possible)
  - **Compiling the negation of the specification described in linear temporal logic**
- Compilation algorithm and optimization is crucial
  - It is the „exponential part" of the model checking process
  - Product computation complexity is linear in the size of the automaton

Detailed steps



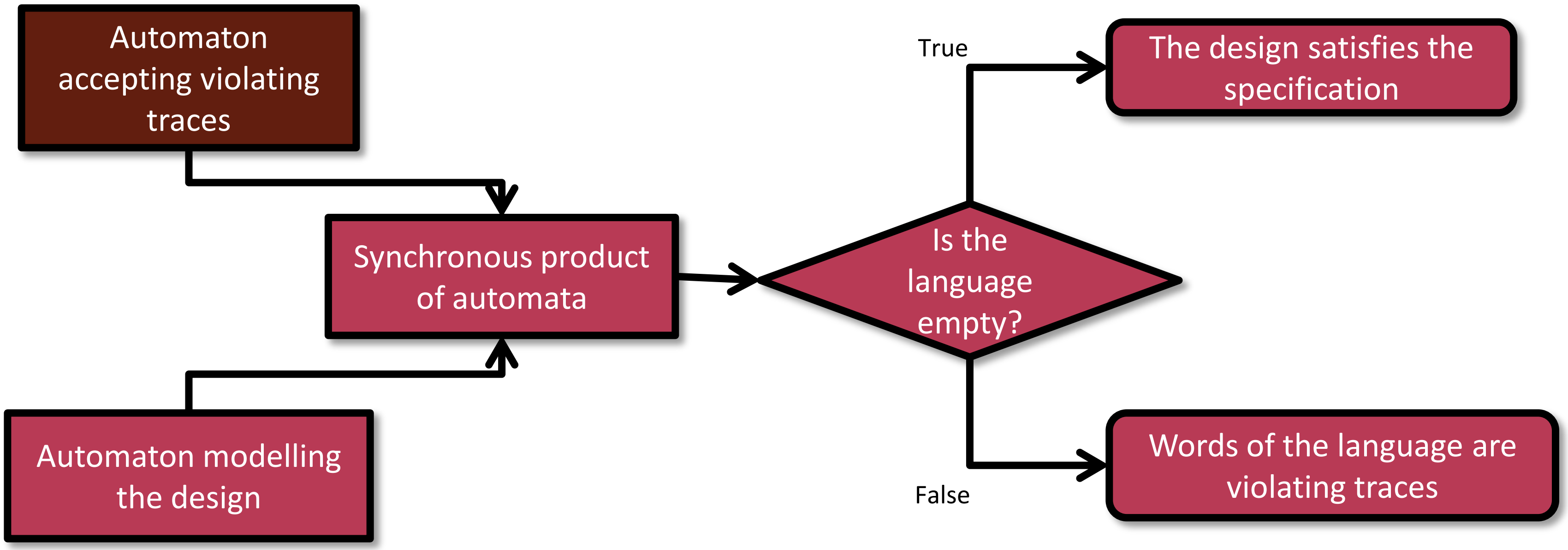
- Can be obtained from higher level models by the means of state-space generation
  - State space can be huge, especially in asynchronous systems
- Symbolic representation can solve storage problems
  - But it complicates the generation process
- Saturation algorithm provides efficient symbolic state space exploration
  - Especially in asynchronous systems…

Detailed steps



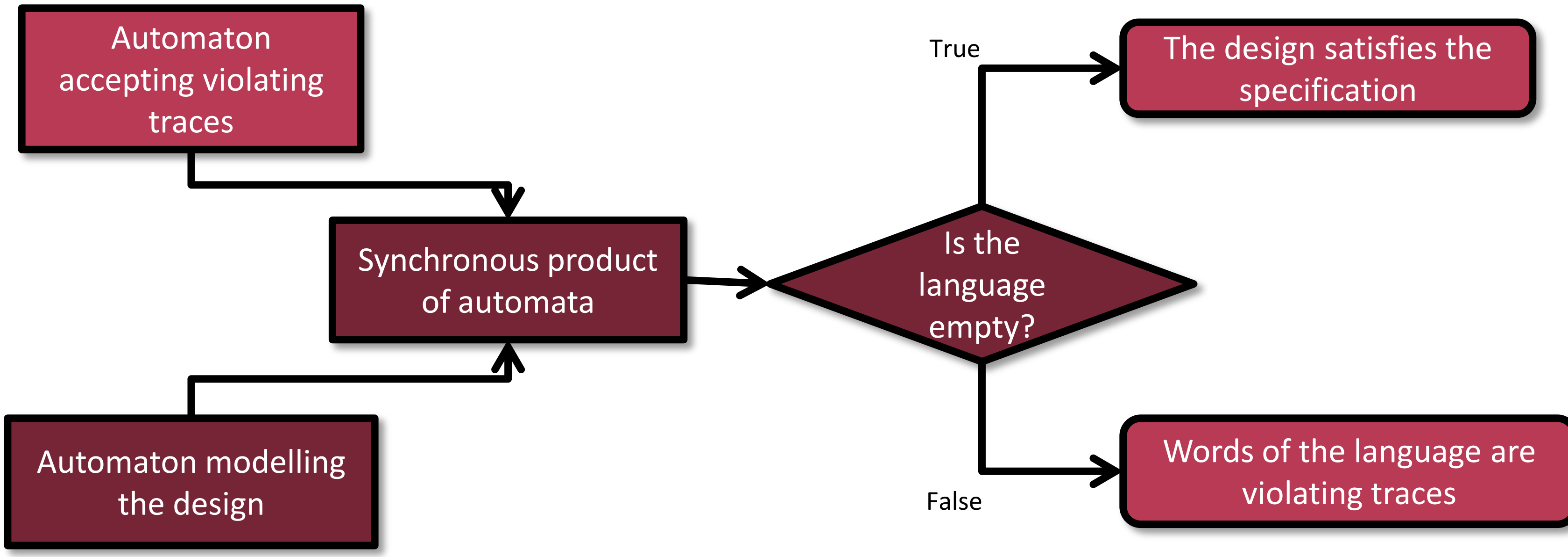
- Product state space is the Cartesian product of the sets of states of the automata
  - But only a part is reachable from the initial state
- A word of the product language is an accepting trace of the product automaton
  - An accepting trace is a reachable loop that contains accepting states
- Task: find such a loop or prove that there does not exist one
  - Efficient on-the-fly algorithms on explicit graph representations
  - Symbolic approaches are cumbersome

Detailed steps



- Compilation algorithm handling linear temporal logics (LTL)
  - Past and future time linear temporal expressions
- Heavy optimization and minimization of the specification automaton
  - Reduction of the temporal expressions
  - Removing redundancy from the automaton
- **A compiled and optimized automaton can be used multiple times!**

Detailed steps



- Product state space is generated together with the system state space
  - Using a modified version of the saturation algorithm
- Emptiness checking is performed simultaneously during the generation process
  - Unlike traditional symbolic approaches, accepting loops are being detected incrementally
  - State space generation halts immediately when a loop is found
- **This gives us an incremental, on-the-fly LTL model checking process**
  - Supporting LTL, PLTL and any ω-regular specification properties