Bounded Saturation Based CTL Model Checking

András Vörös, **Dániel Darvas**

Budapest University of Technology and Economics, Budapest, Hungary

Fault Tolerant Systems Research Group

Tamás Bartha

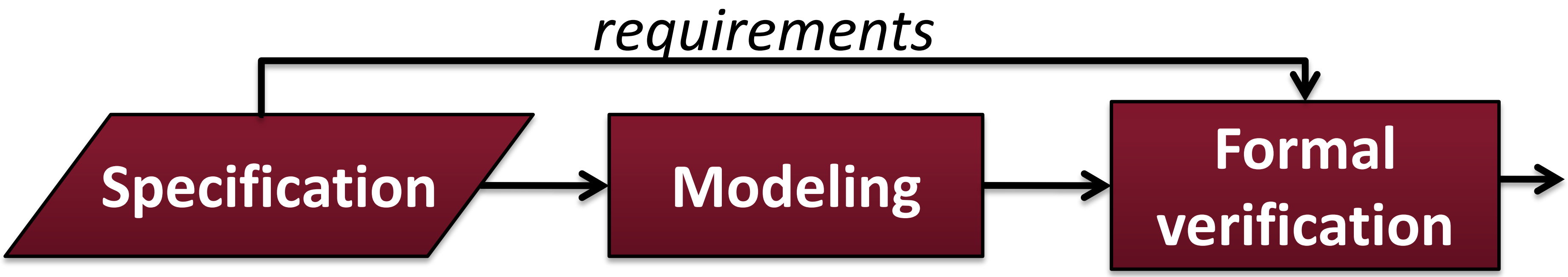
MTA SZTAKI, Budapest, Hungary

Computer and Automation Research Institute

Contents

Formal methods

- Safety critical and embedded systems
  - Railway, automotive industry, air transportation
  - Reliability is an important issue
- Design time analysis:



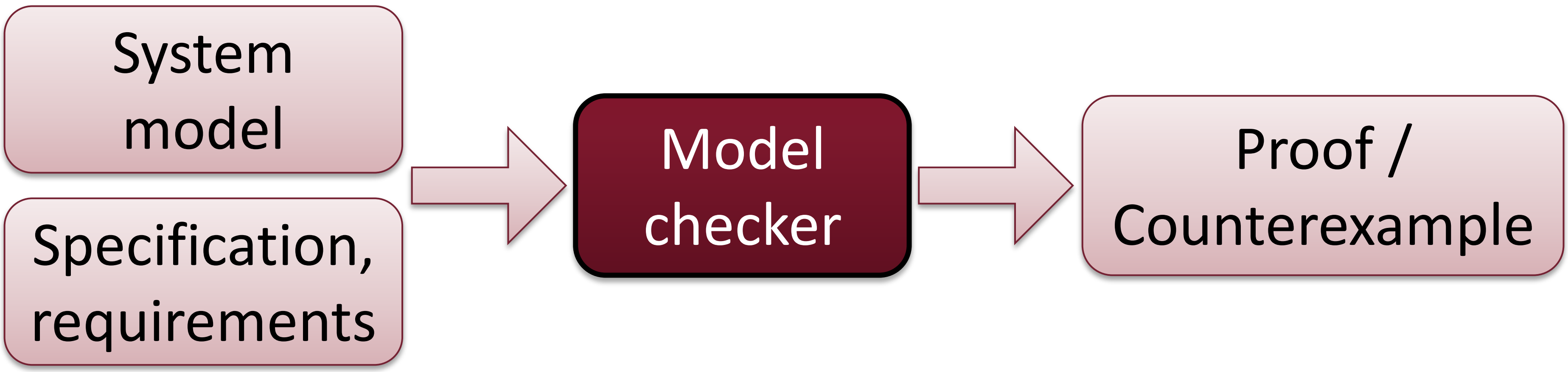
– Does my system work well?

– Does it provide services properly?

Mathematically sound answer

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 (MDDs)
- Special **iteration strategy**
  - Efficient for asynchronous models

Overview of the saturation workflow

Model of the system

Decomposition and ordering

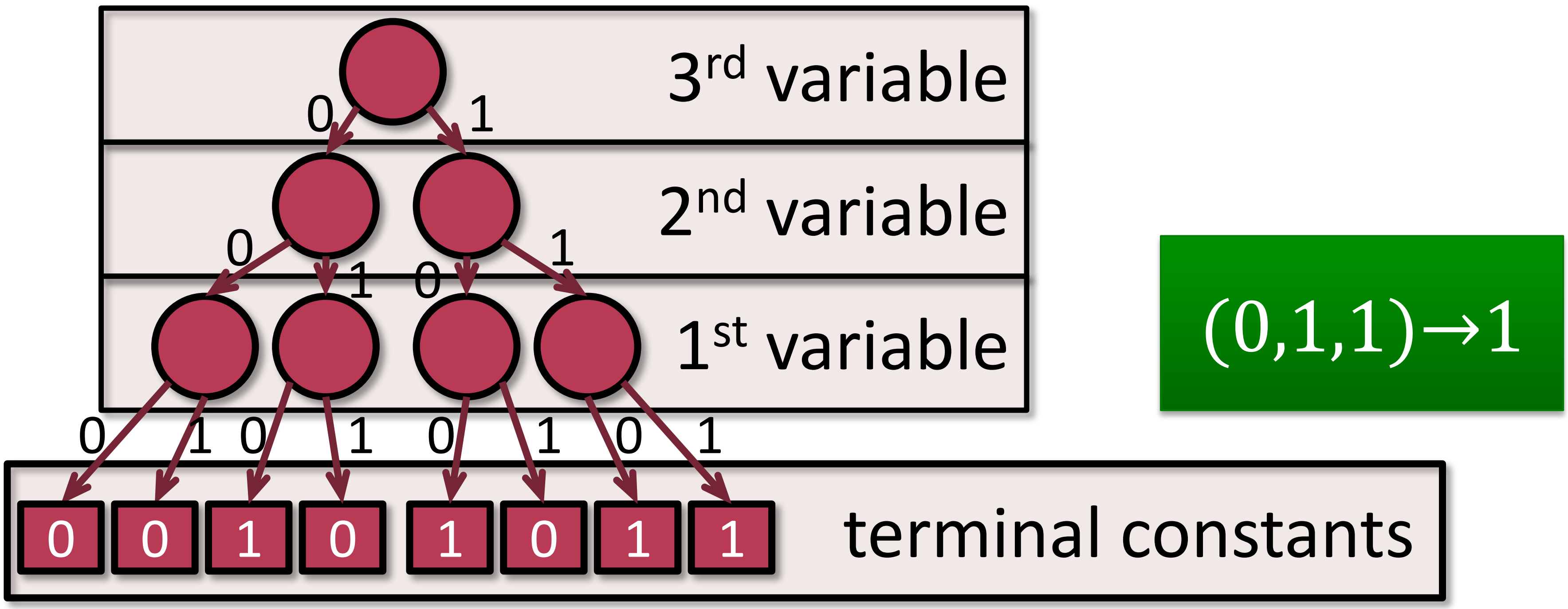
Saturation

State space exploring

Model checking

Multi-valued Decision Diagrams

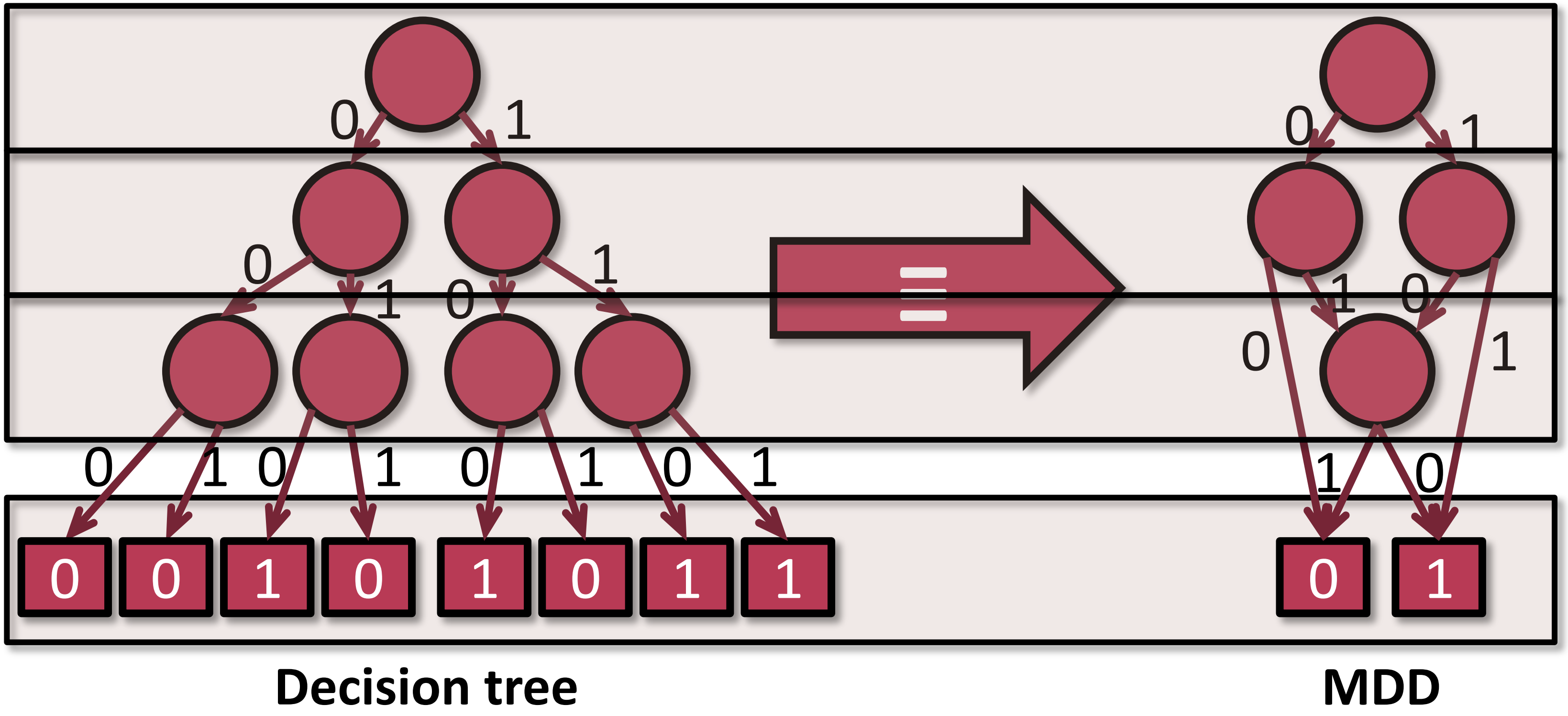
- Derived from **decision trees**
  - variables are ordered into levels



Decision tree

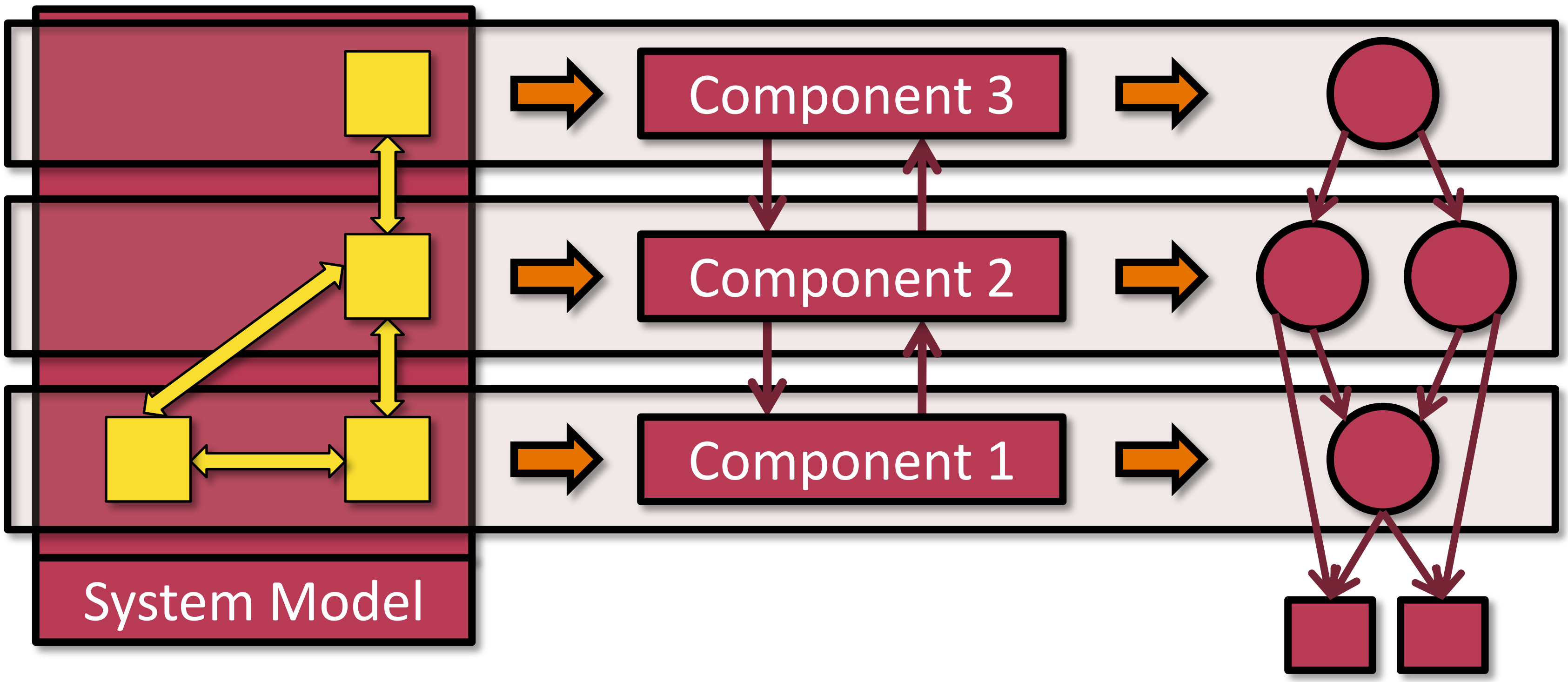
Multi-valued Decision Diagrams

- 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



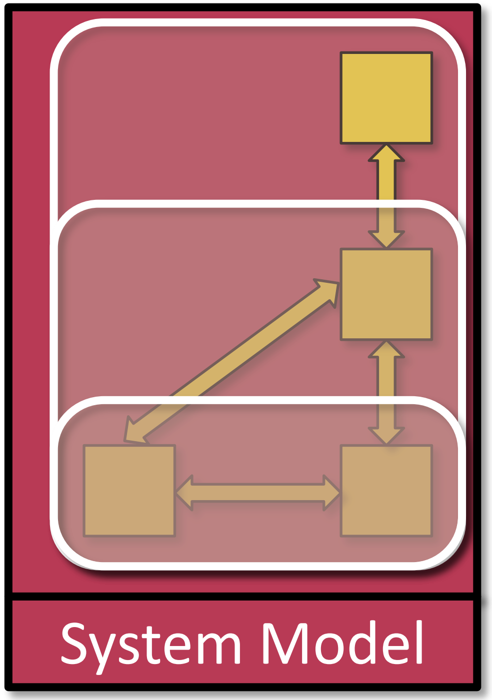
Symbolic algorithm

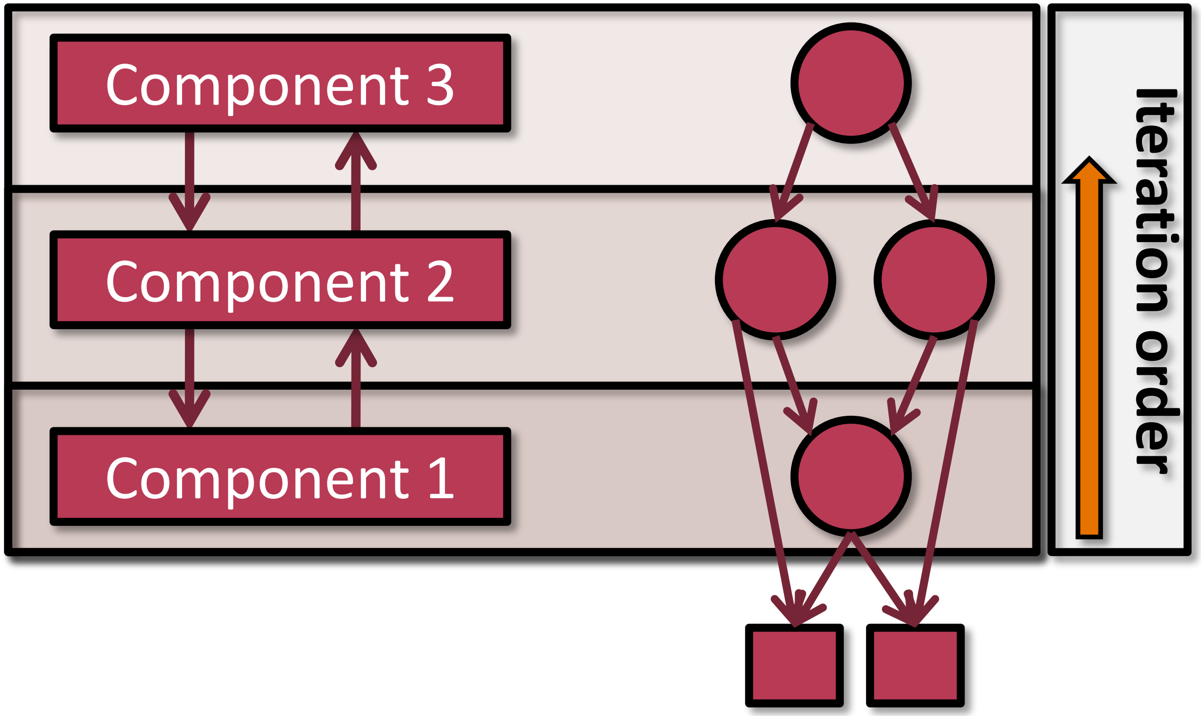
- Symbolic encoding instead of explicit state representation
  - Decomposition is needed
- Saturation uses componentwise encoding



Special iteration

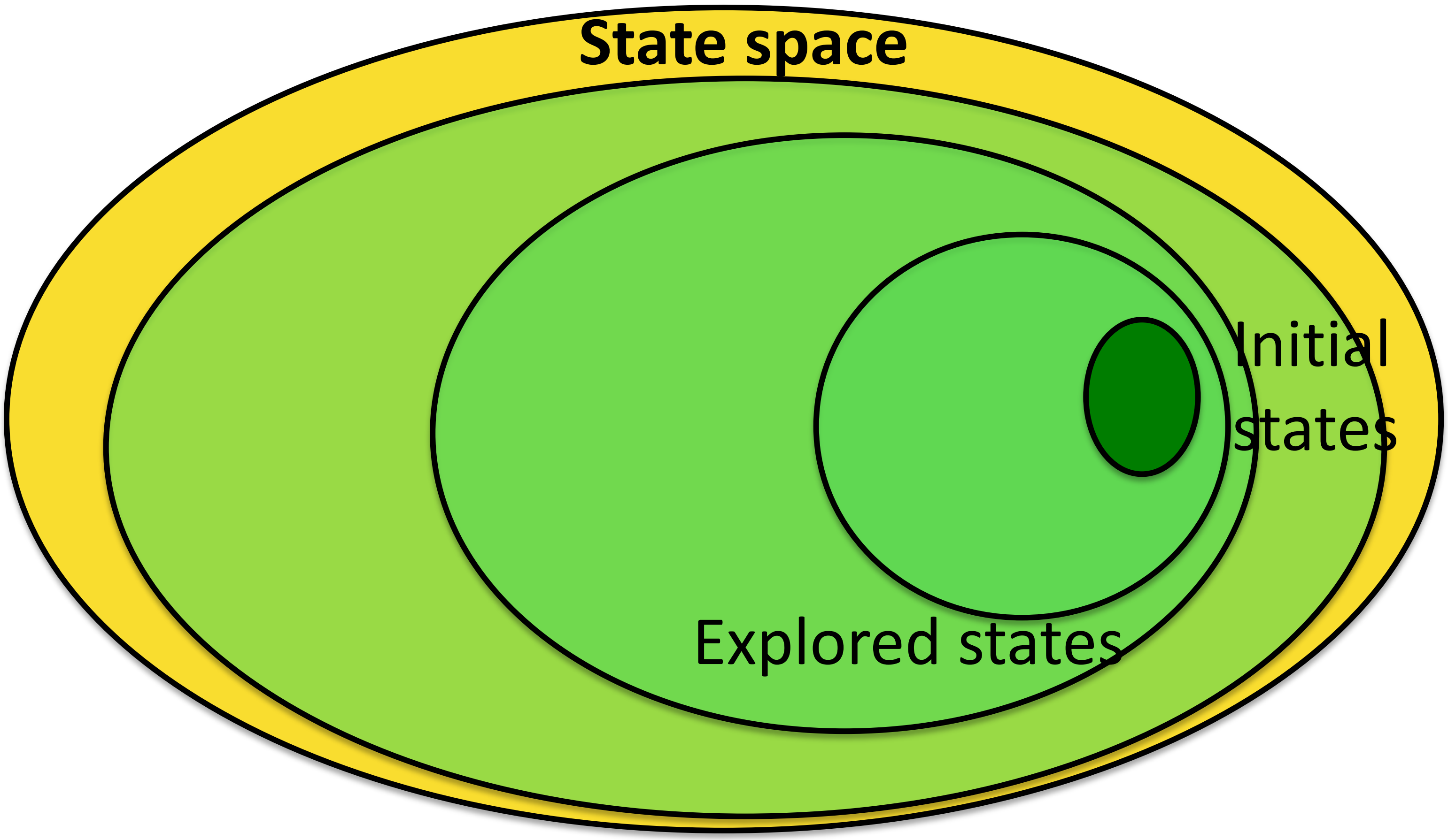
- Local exploration in a greedy manner
- Exploring global synchronization events if needed
- Uses the primarily defined order of the decision diagram variable encoding



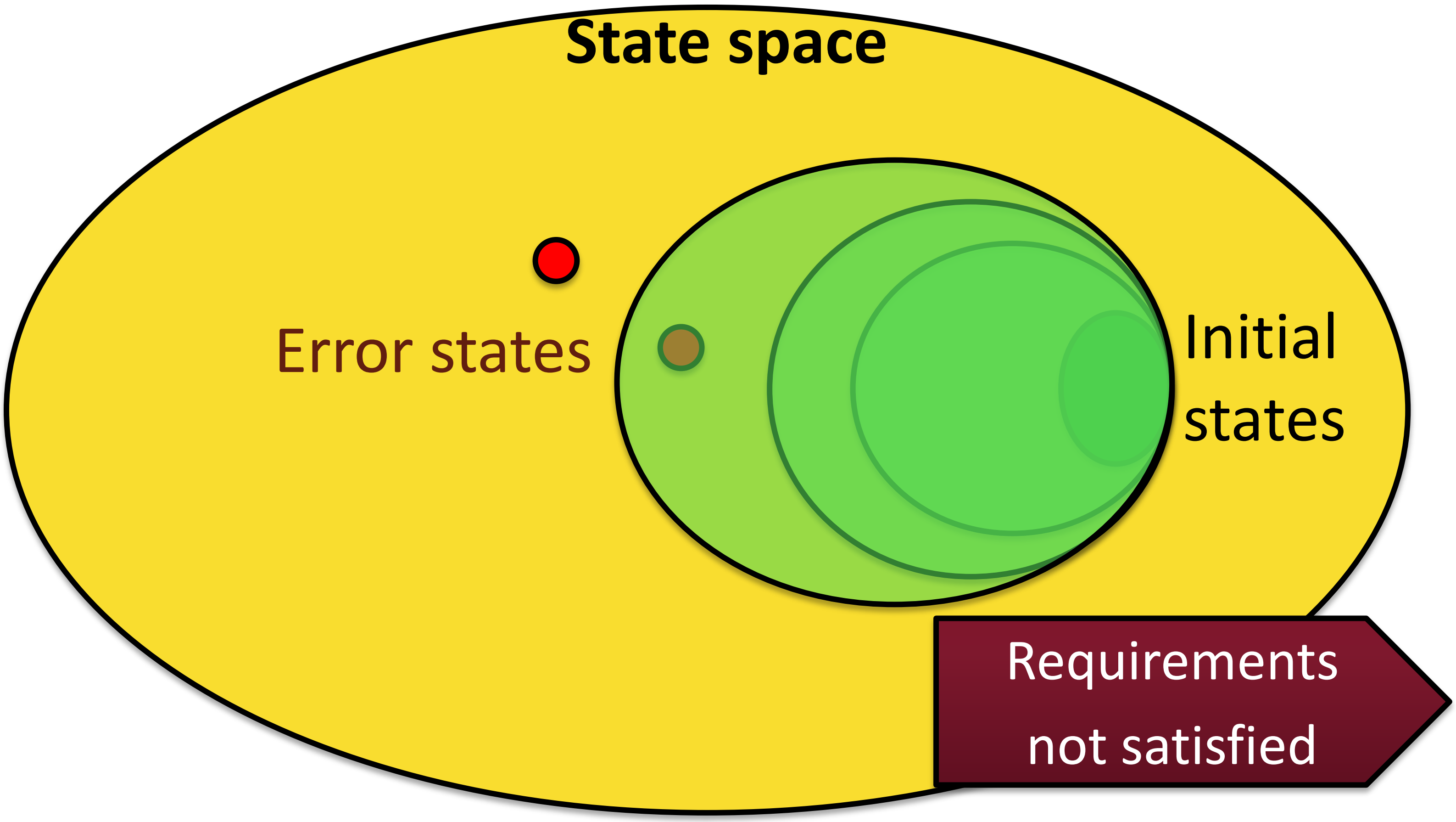


Bounded saturation

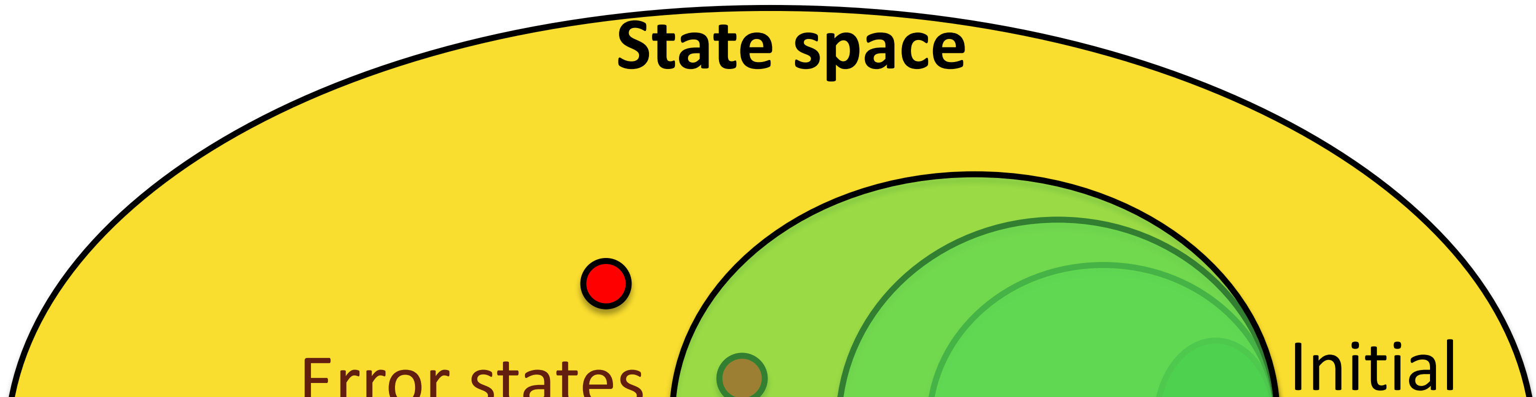
Motivation for bounded model checking



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

Problems with Bounded Saturation

- Main problems with bounded saturation:
  - Saturation explores the state space in an irregular recursive order
    - ⇒ **Difficult to limit** the exploration
  - There is **no distance information** in the MDDs
    - ⇒ New data structure is needed

Problems with Bounded Saturation

- Main problems with bounded saturation:

Bounded number of iterations

Truncating the state space representation

New data structure:

Edge-valued decision diagrams (EDDs)

MDD based data structure enriched with distance information

Open questions before our work

- How can we ***implement*** a bounded state space exploration (BSSE) module?
  - In theory it needs *information about the state space*, but this is not available a priori.
- Can BSSE and a saturation based model checker ***work together***?
- Is there an efficient way for using ***exact bounds***?
- How can we use this method for model checking ***in practice***?

Truncating methods

- Two approach:

Exact truncating

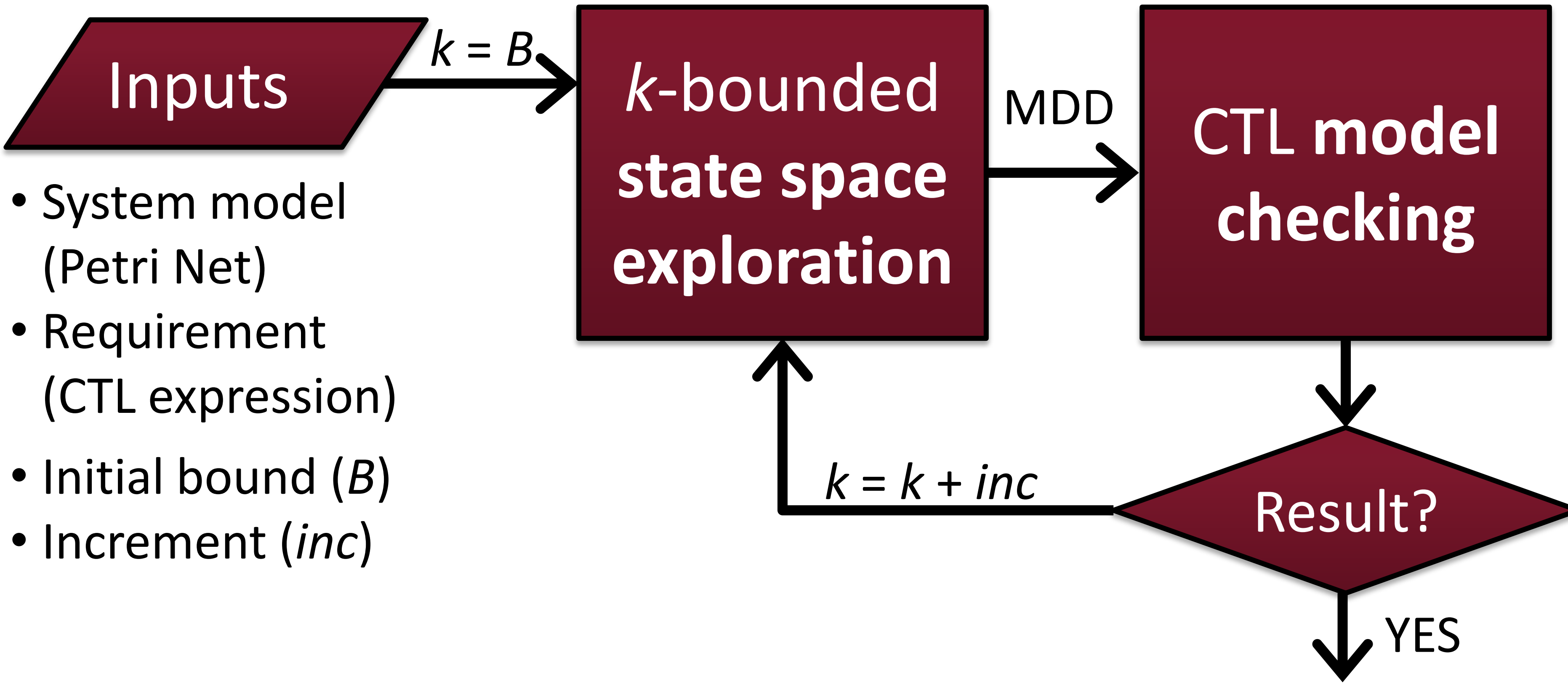
- exact *k*-bound
- less efficient originally
- in our work: **competitive (using caches)**

Approximate truncating

- only $k..k \cdot C$-bound
  (C: number of components)
- more efficient
  (as stated earlier)

Iterative BMC

- The necessary *k*-bound is not known a priori
- We use an **iterative algorithm**:



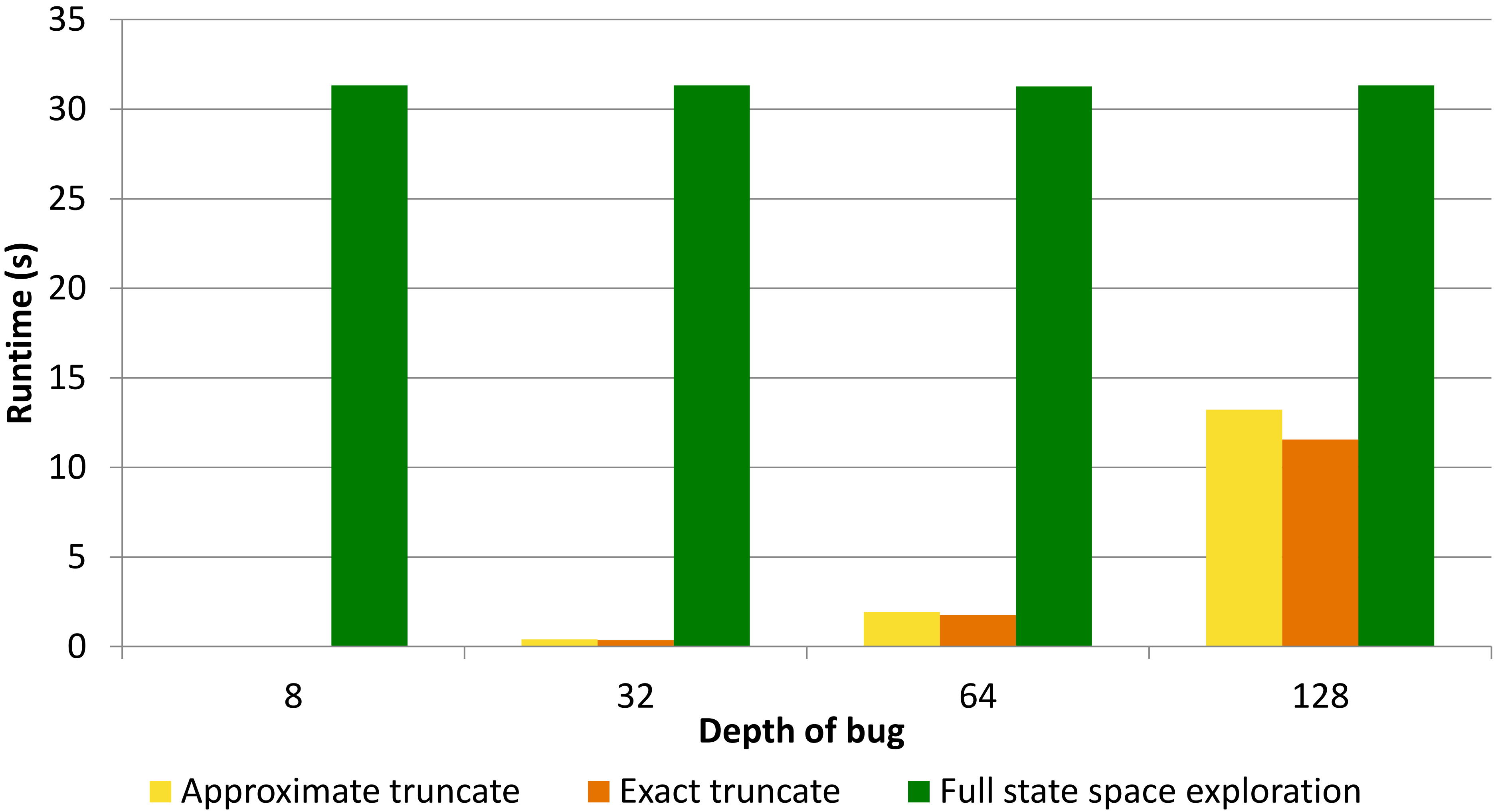
Efficiency problems

- Classical saturation based MC may explore and check some unreachable states needlessly
  - The answer is correct but could be more efficient
  - Work in progress (constrained saturation)
- Incremental building of the state space
  - The algorithm restarts state space exploration from the initial state
  - Future work

Measurements and conclusion

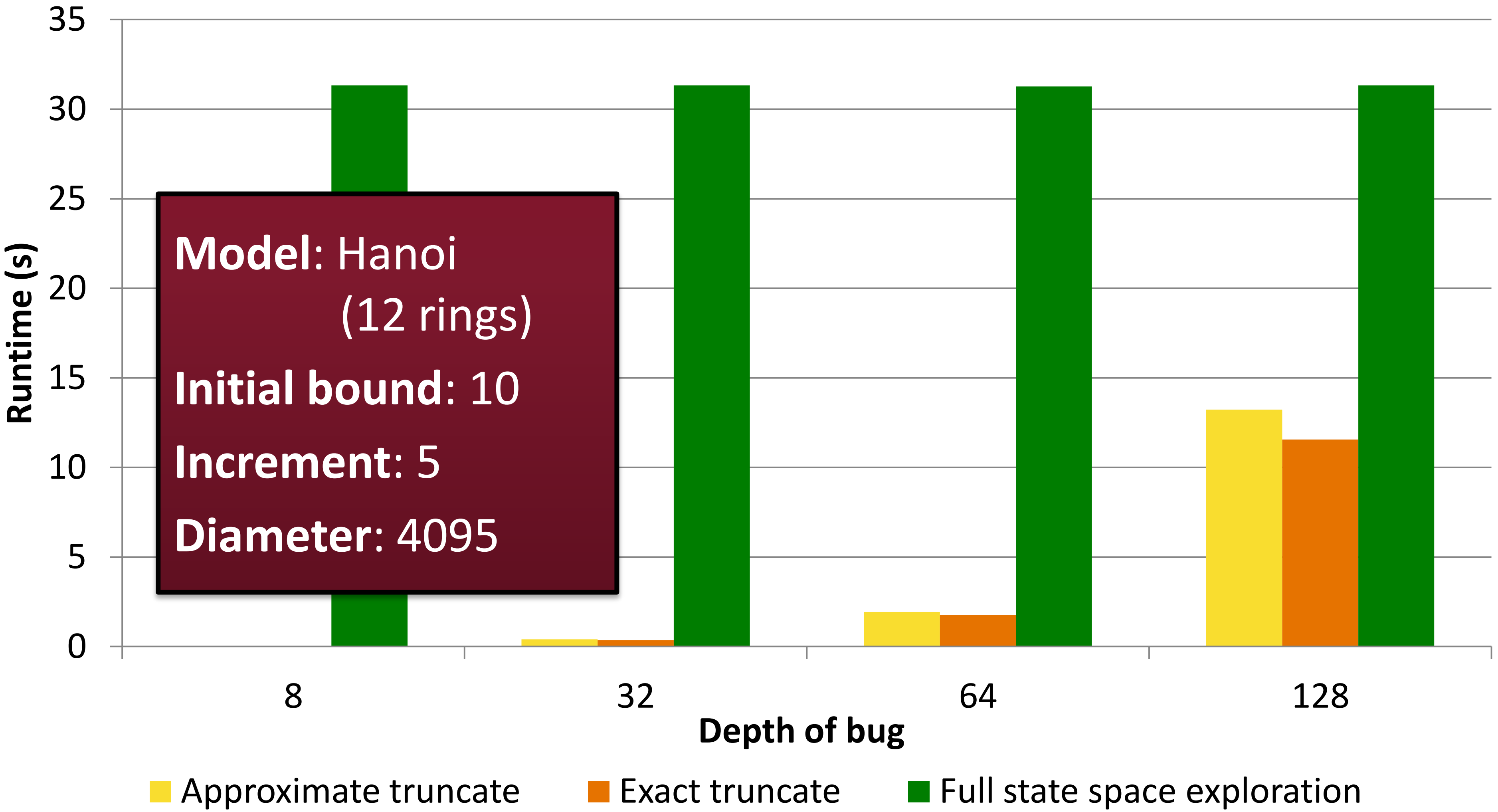
Measurements

- Scaling with **depth of „error state”**



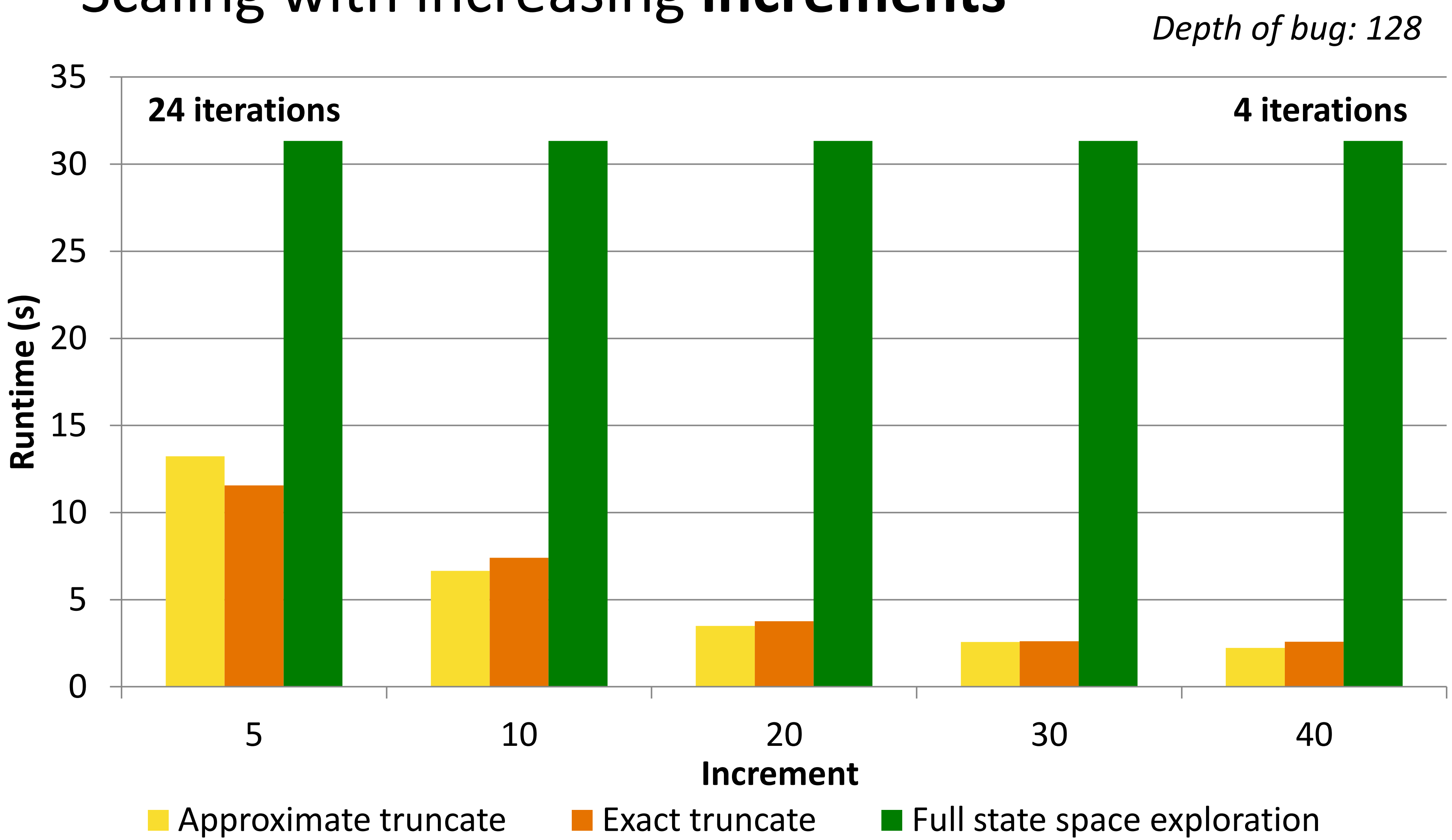
Measurements

- Scaling with **depth of „error state"**



Measurements

- Scaling with increasing **increments**



Conclusion

- We have *implemented* a bounded model checker in our tool
  - Theoretical and practical improvements in the original bounded algorithm
- Saturation based bounded state space generation and classic MC *could work together*
- Efficient for ”*shallow bugs*”
- Efficiency is highly depending on the chosen *parameters*

Thank you for your attention!