Software Verification with Abstraction-Based Methods

Ákos Hajdu

PhD student

Department of Measurement and Information Systems, Budapest University of Technology and Economics

MTA-BME Lendület Cyber-Physical Systems Research Group

Electrical and Computer Engineering Department, McGill University

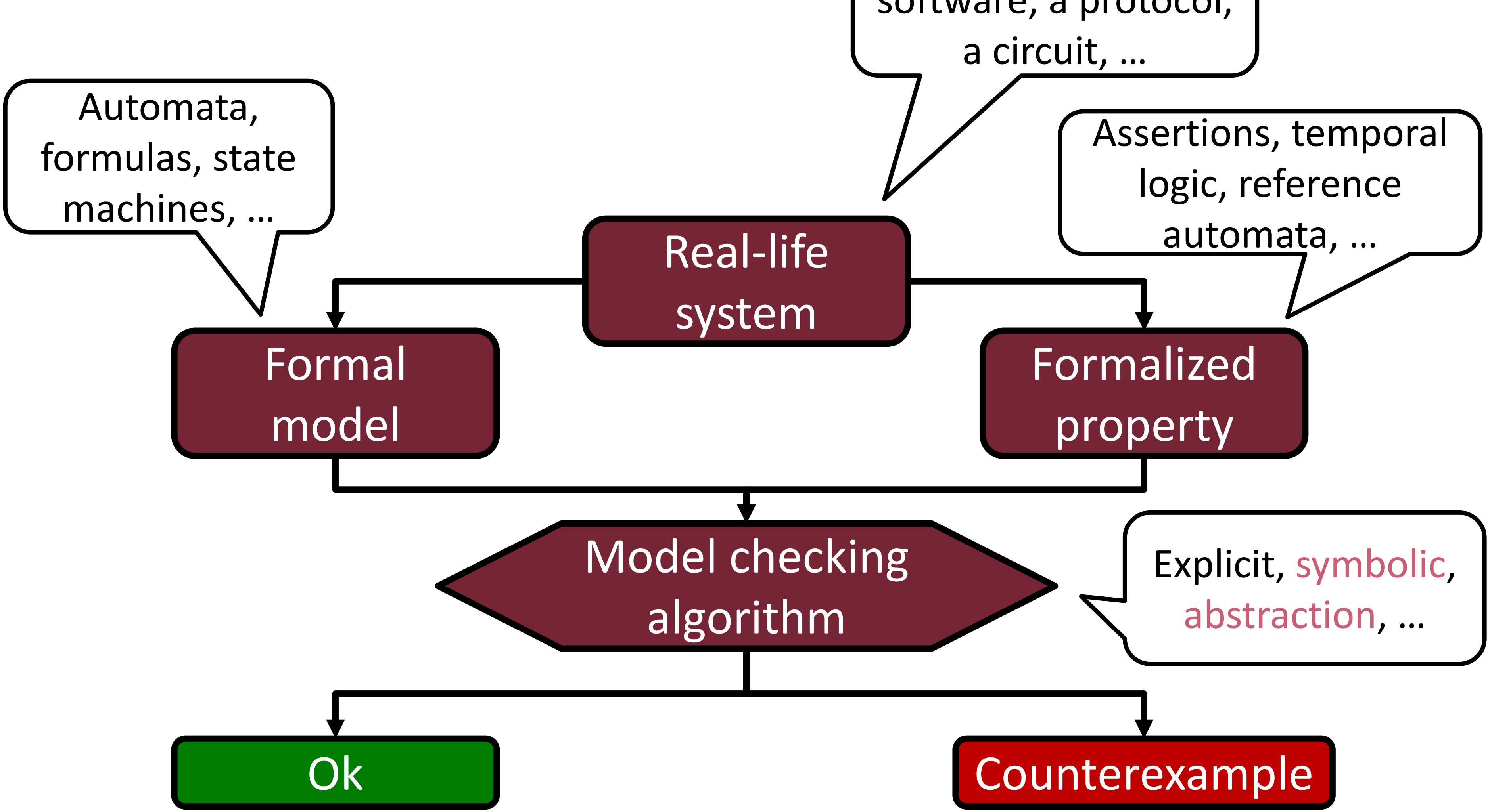
Background

Background – Formal Verification

- Formal verification
  - Prove or disprove the correctness of a system with respect to a formal property (specification) relying on sound mathematical basis
- Model checking
  - Exhaustively enumerate the possible states and transitions (the state space) of the system and check if it meets the property

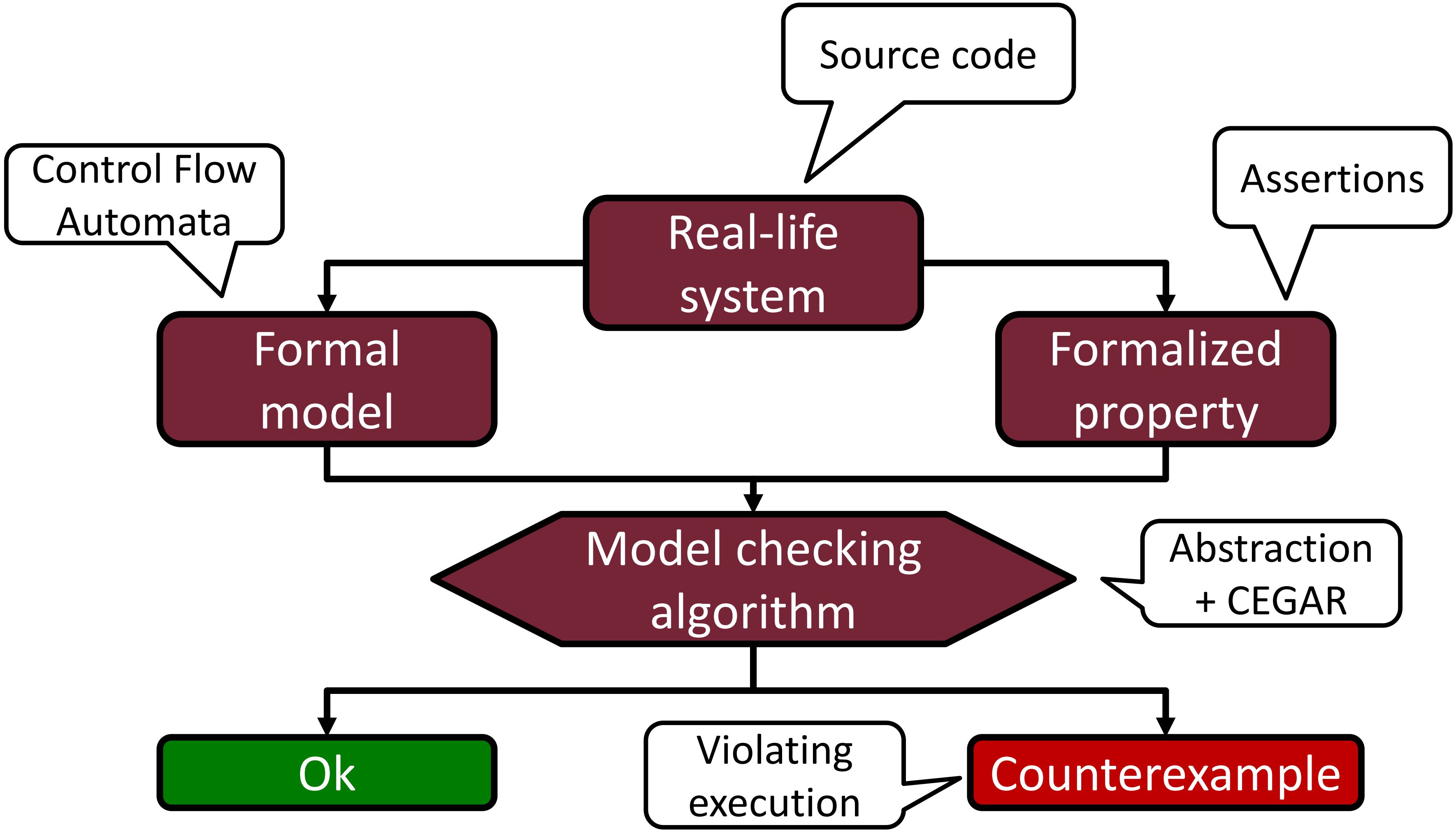
Background – Model Checking

- Model checking in general



Background – Model Checking

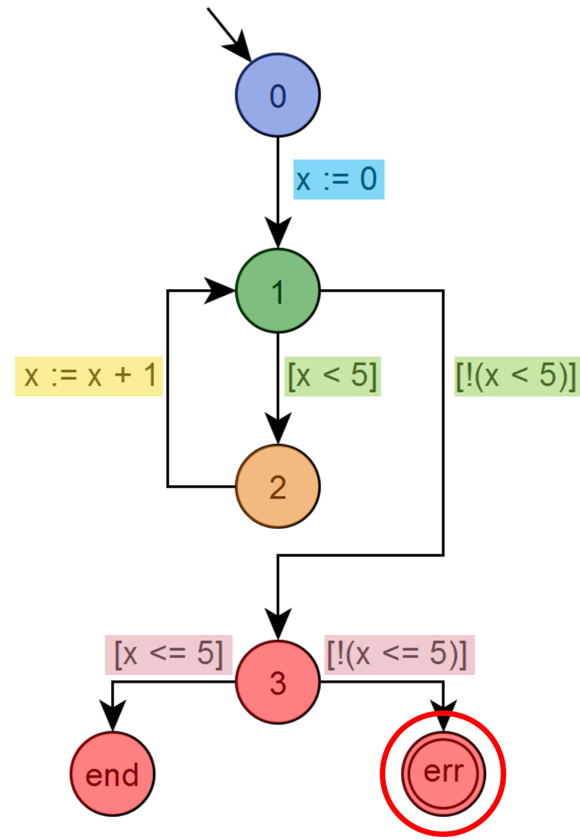
- This talk: focus on software and abstraction



Background – Model and Property

- Control-Flow Automaton
  - Set of control locations (PC)
  - Set of edges with operations over a set of variables
    - E.g., guard, assignment …

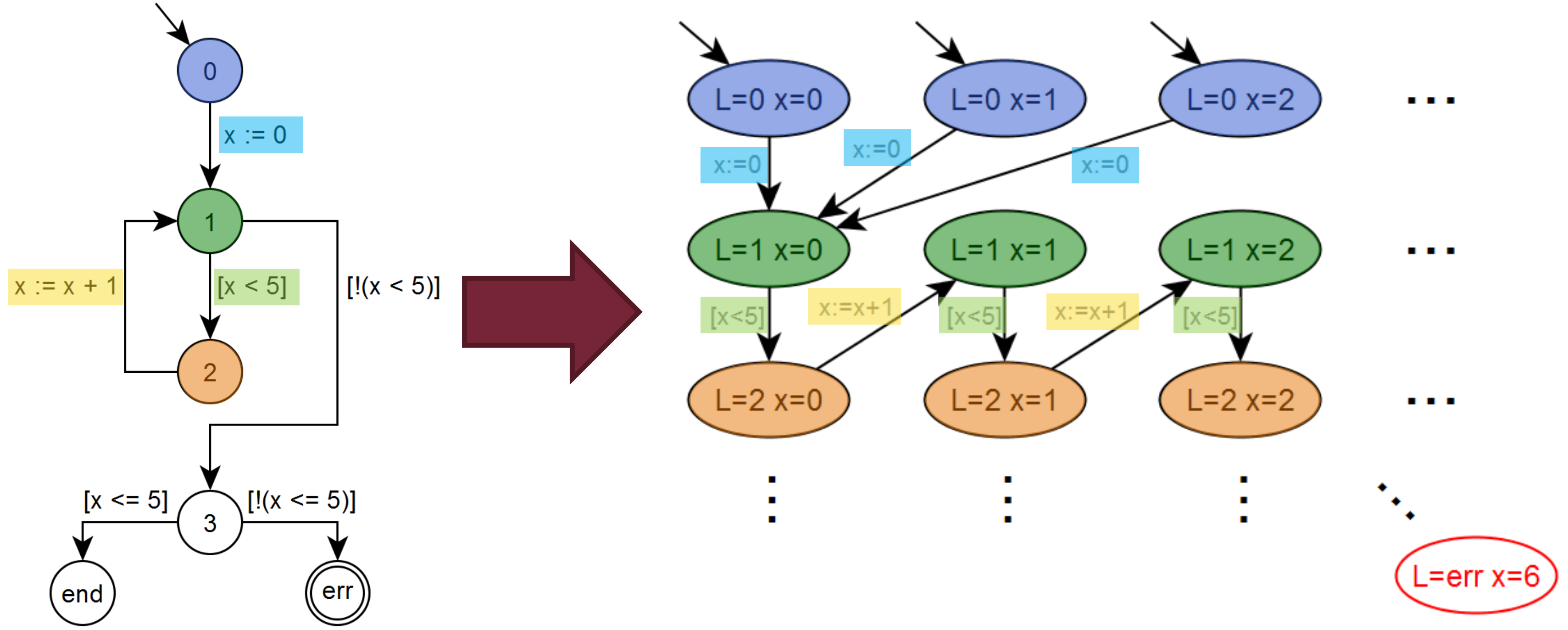
```
   x : int
0: x = 0
1: while (x < 5) {
2:     x = x + 1
   }
3: assert (x <= 5)
```



- Typical property: "error" location should not be reachable

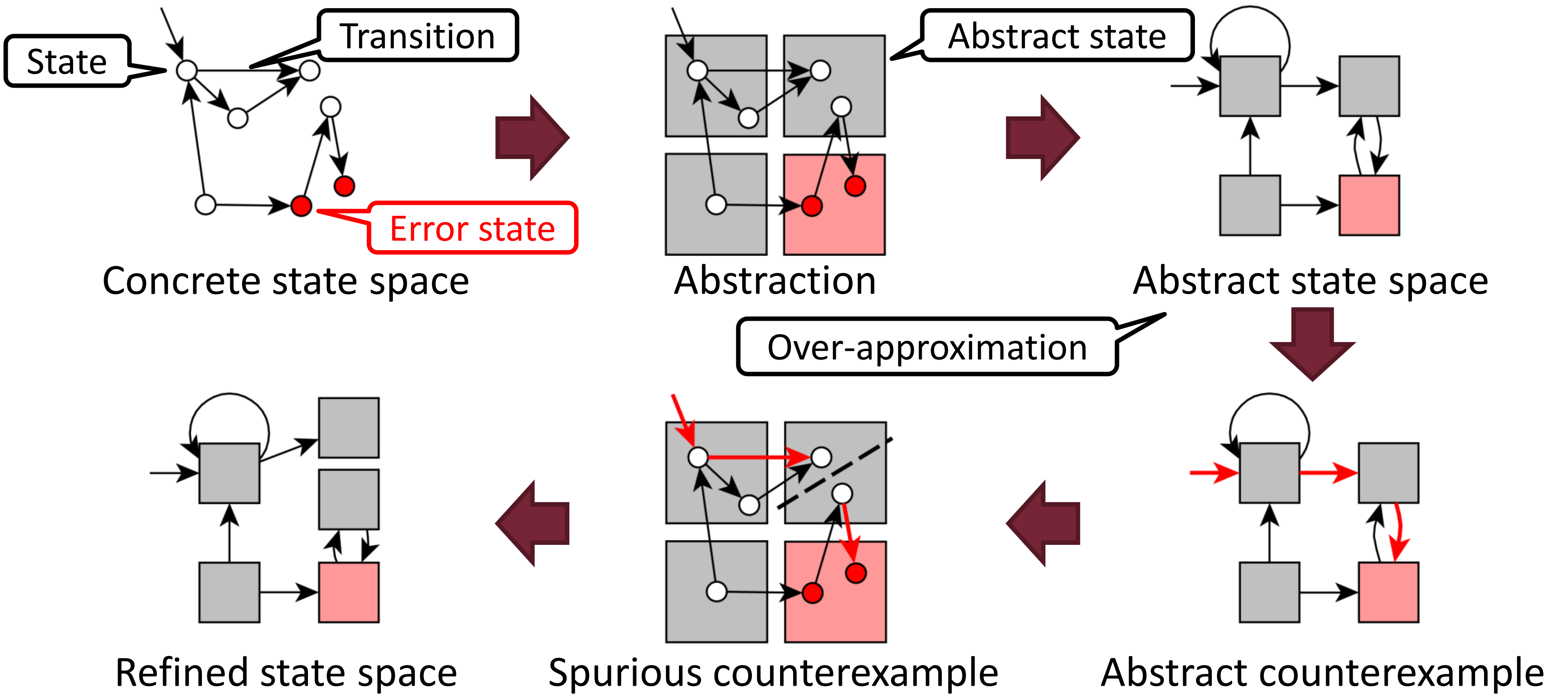
Background – States and Transitions

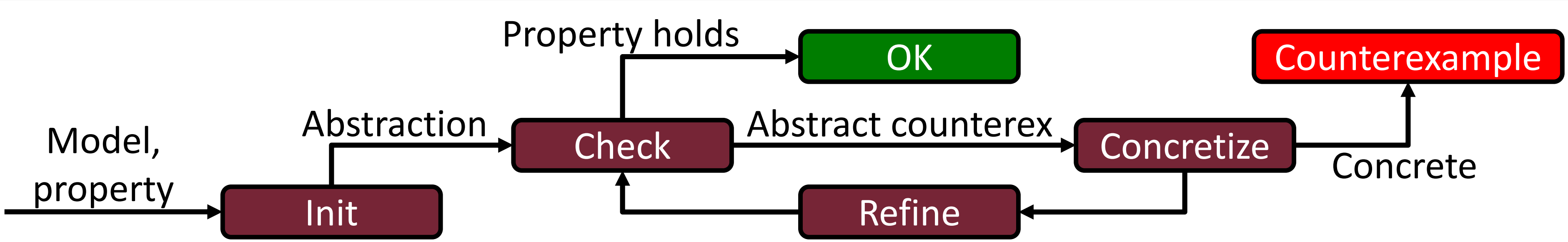
- State: location + valuation of variables (L, x_1, x_2, …, x_n)
- Transition: operations
- Problem: state space explosion caused by data variables
  - E.g., 10 locations and 2 integers: $10 \cdot 2^{32} \cdot 2^{32}$ possible states
- Goal: reduce the state space representation by abstraction



Counterexample-Guided Abstraction Refinement (CEGAR)

CEGAR – Introduction



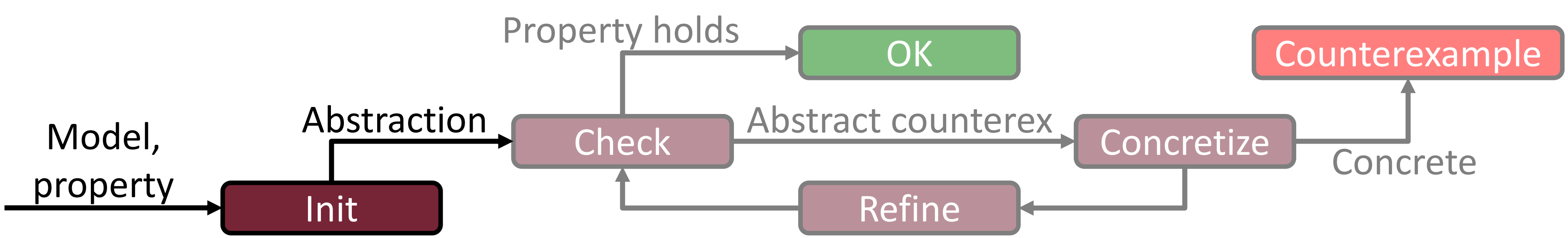


CEGAR – Initial Abstraction

- Predicate abstraction
  - Track predicates instead of concrete values
  - |P| predicates → $2^{|P|}$ possible abstract states
  - Label of a state: predicates, e.g. ¬(x > y) ∧ (y = 3)

Variables:
x, y; $D_x = D_y = \{1, 2, 3\}$
Predicates:
(x > y), (y = 3)

	(x > y)	¬(x > y)
(y = 3)		(x=1, y=3) (x=2, y=3) (x=3, y=3)
¬(y = 3)	(x=2, y=1) (x=3, y=1) (x=3, y=2)	(x=1, y=1) (x=1, y=2) (x=2, y=2)

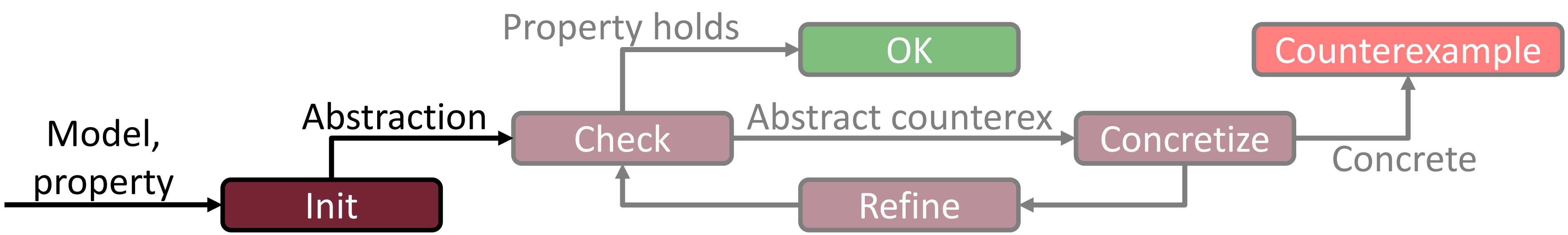


CEGAR – Initial Abstraction

- Explicit value abstraction
  - Partition variables: visible / invisible
  - Track values for visible variables only
  - Label of a state: assignment, e.g. $(x = 1) \wedge (y = 2)$

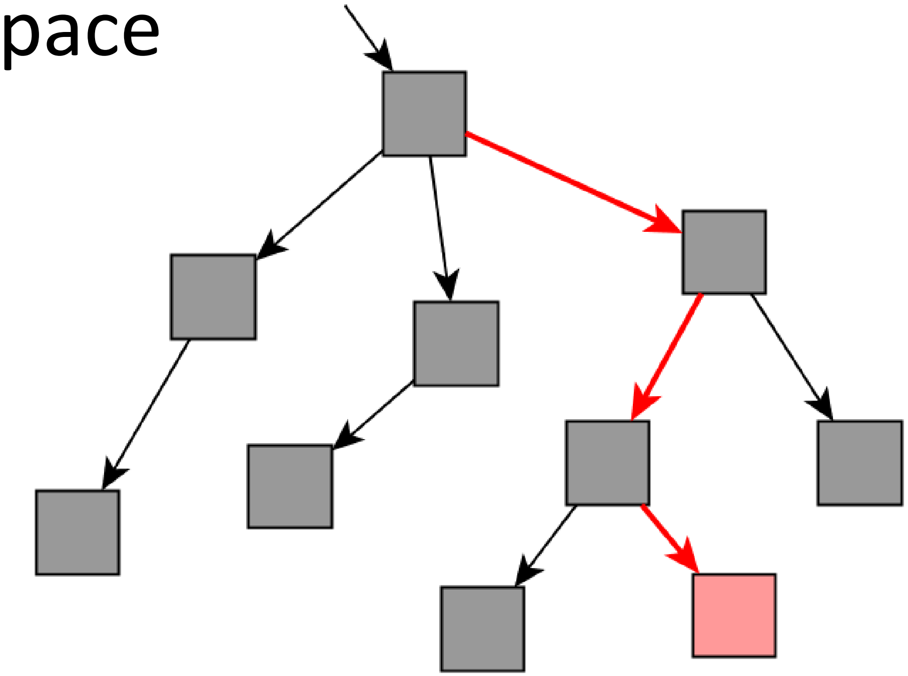
Variables: x, y, z
$D_x = \{0, 1\}$, $D_y = \{0, 1, 2\}$, $D_z = \{0, 1\}$
Visible = {x, y}

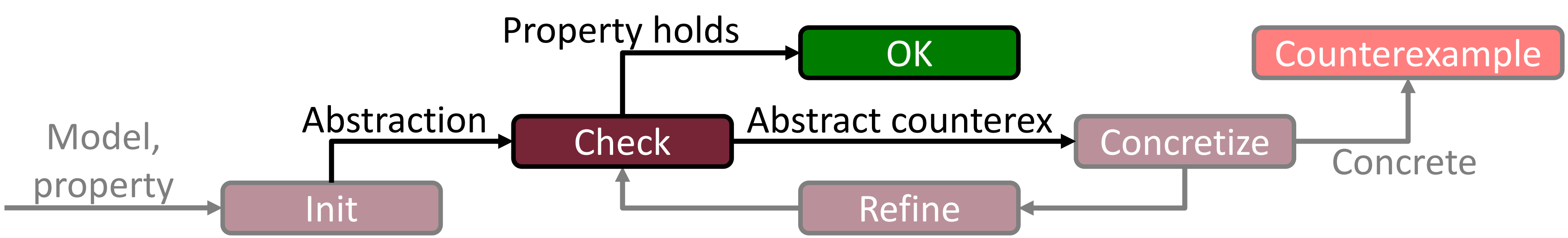
	x=0	**x=1**
y=0	(x=0, y=0, **z=0**) (x=0, y=0, **z=1**)	(x=1, y=0, **z=0**) (x=1, y=0, **z=1**)
y=1	(x=0, y=1, **z=0**) (x=0, y=1, **z=1**)	(x=1, y=1, **z=0**) (x=1, y=1, **z=1**)
y=2	(x=0, y=2, **z=0**) (x=0, y=2, **z=1**)	(x=1, y=2, **z=0**) (x=1, y=2, **z=1**)



CEGAR – Model Checking

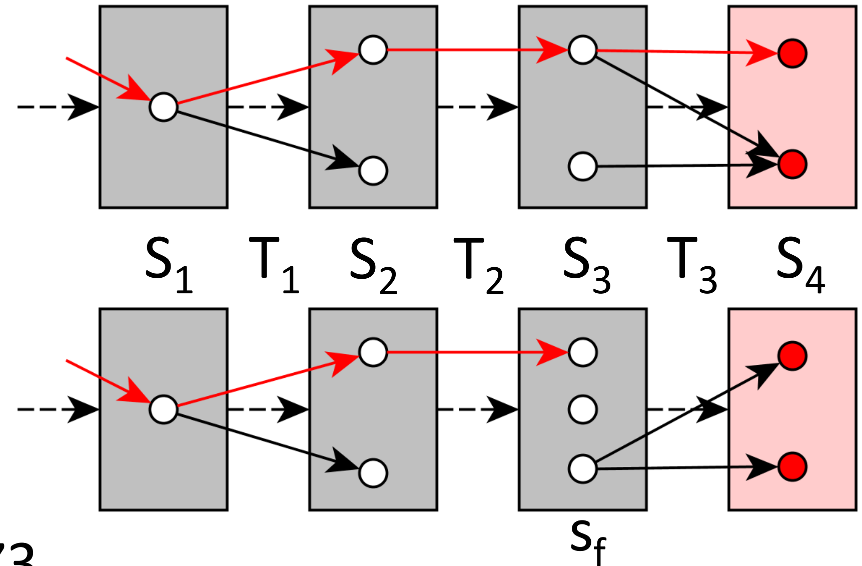
- Traverse abstract state space
  - Search strategy
- Search for error state
- Optimizations
  - On-the-fly
  - Incremental

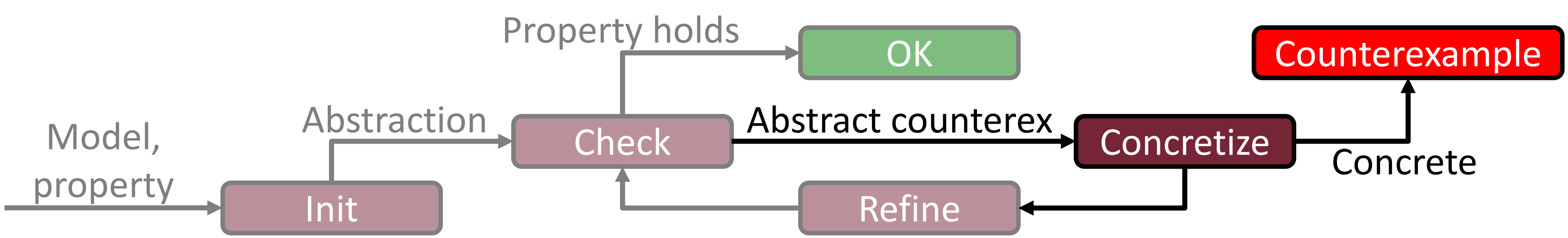




CEGAR – Concretization

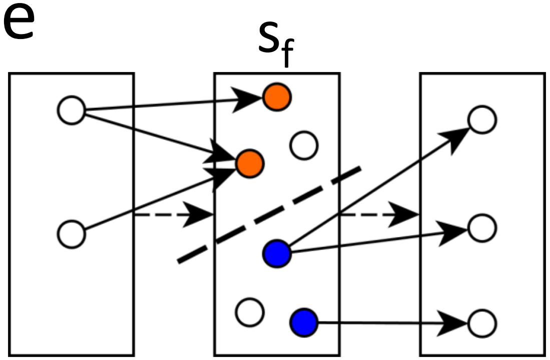
- Traverse subset of concrete state space
  - Concretizable counterexample
  - Spurious counterexample
    - Failure state (S_f)
  - Use SMT solver, e.g. Microsoft Z3
    - $S_1 \land T_1 \land S_2 \land T_2 \land \ldots \land T_{n-1} \land S_n$



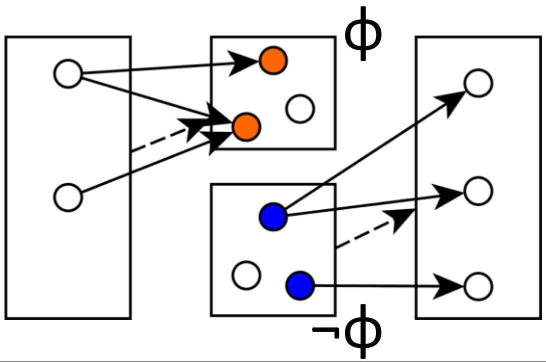


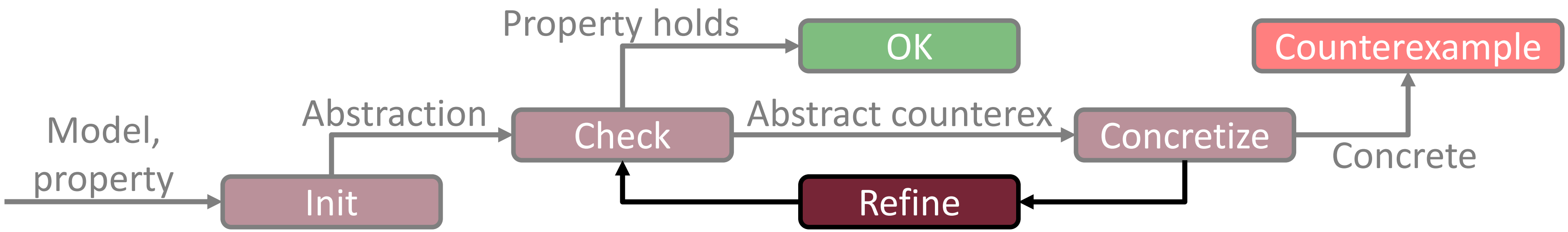
CEGAR – Abstraction Refinement

- Classify states mapped to the failure state
  - D = Dead-end: reachable
  - B = Bad: transition to next state
  - IR = Irrelevant: others



- Goal: finer abstraction mapping D and B to separate abstract states
  - SMT solver: interpolation formula ϕ
  - Use ϕ as predicate or extract its variables

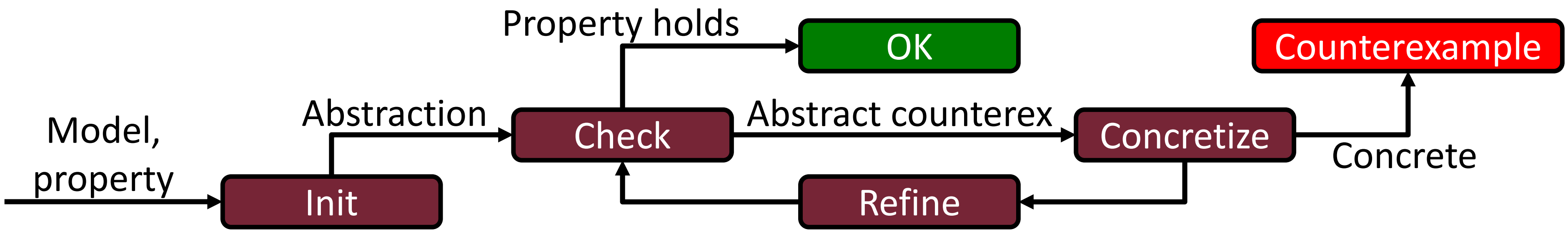




CEGAR – Summary

- CEGAR is a general concept
  - Explore abstract state space
  - Refine abstraction if needed

- Many variants exist (for various formal models)
  - Abstract domains*, e.g., predicates, explicit values, zones*
  - Refinement strategies*, e.g., interpolation, unsat cores*
  - Exploration strategies*, e.g., BFS, DFS*



Research Questions

- Integrate variants into common framework? Combine?
  - Theta Verification Framework. http://theta.inf.mit.bme.hu
  - A configurable CEGAR framework with interpolation-based refinements. Ákos Hajdu, Tamás Tóth, András Vörös, and István Majzik. FORTE 2016, vol. 9688 of LNCS.
- Which variants perform well for given verification tasks?
  - Exploratory analysis of the performance of a configurable CEGAR framework. Ákos Hajdu and Zoltán Micskei. PhD Mini-Symposium 2017, BME DMIS.
  - Towards evaluating size reduction techniques for software model checking. Gyula Sallai, Ákos Hajdu, Tamás Tóth, and Zoltán Micskei. VPT 2017. (Accepted)
- Domain specific CEGAR variants?
  - Exploiting hierarchy in the abstraction-based verification of statecharts using SMT solvers. Bence Czipó, Ákos Hajdu, Tamás Tóth, and István Majzik. FESCA 2017, vol. 245 of EPTCS.
  - New search strategies for the Petri net CEGAR approach. Ákos Hajdu, András Vörös, and Tamás Bartha. ICATPN 2015, vol. 9115 of LNCS.

http://home.mit.bme.hu/~hajdua/publications

Theta Verification Framework

Theta Verification Framework

Θ

Theta

Generic

Various kinds of formal models

Configurable

Different algorithms and strategies

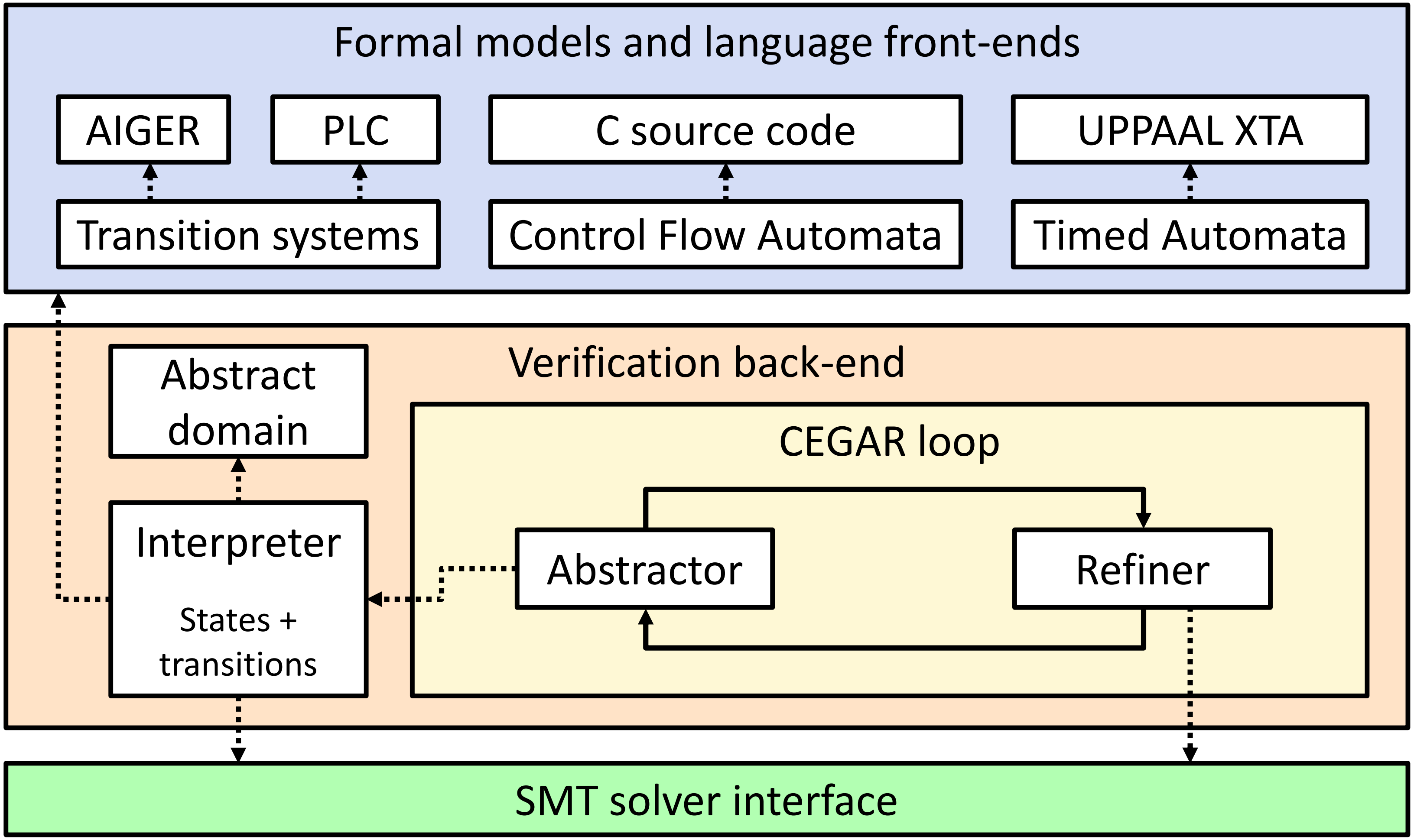
Modular

Reusable and combinable modules

http://theta.inf.mit.bme.hu

Theta Verification Framework

- Architecture



Theta Verification Framework

- Configurability

Abstract domain
- Predicate
- Explicit value
- Zone
- Location
- Composition

Refinement strategy
- Binary interp. forw.
- Binary interp. backw.
- Sequence interp.
- Unsat core

Search strategy
- BFS
- DFS

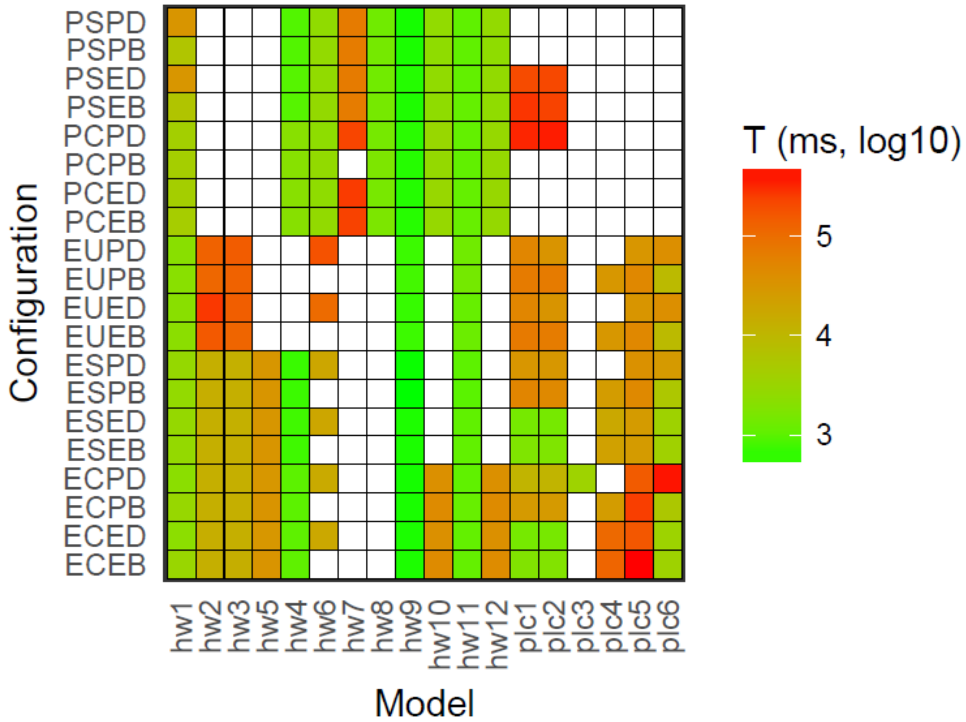
Initial precision
- Empty
- Property-based

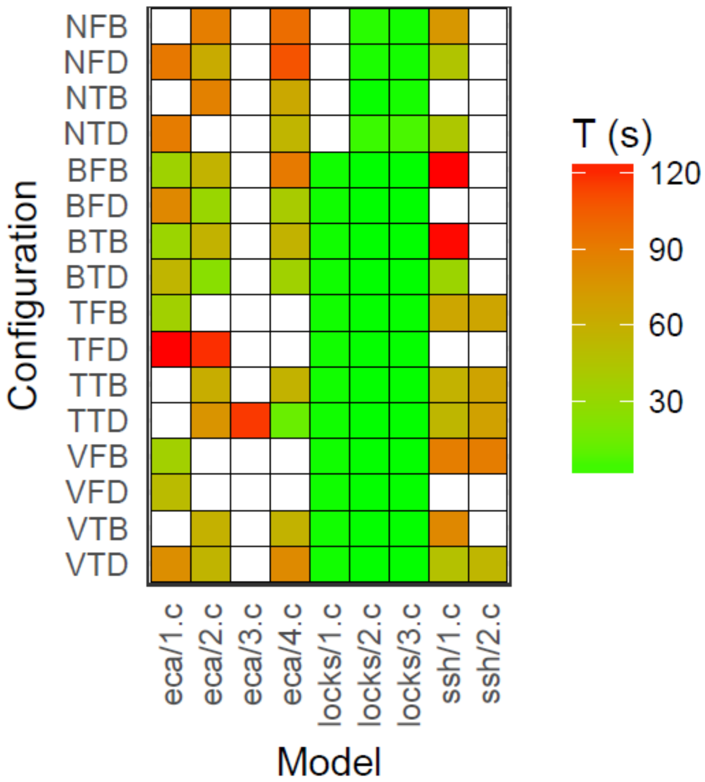
Precision granularity
- Constant
- Location-based

Predicate split
- Atoms
- Conjuncts
- Whole

Theta Verification Framework

- Evaluation
  - Really diverse results
  - Current research: data analysis & heuristics

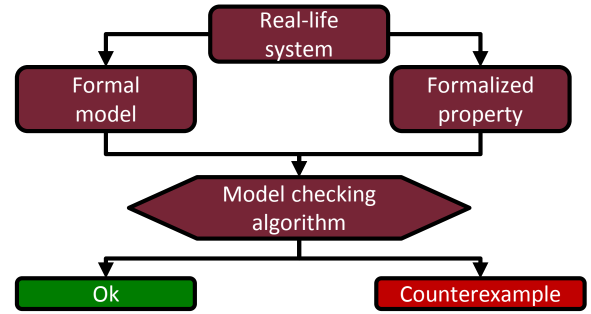


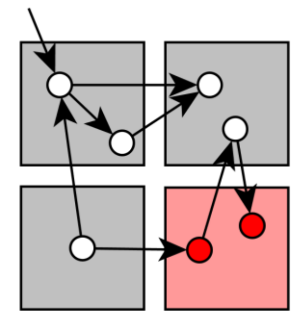


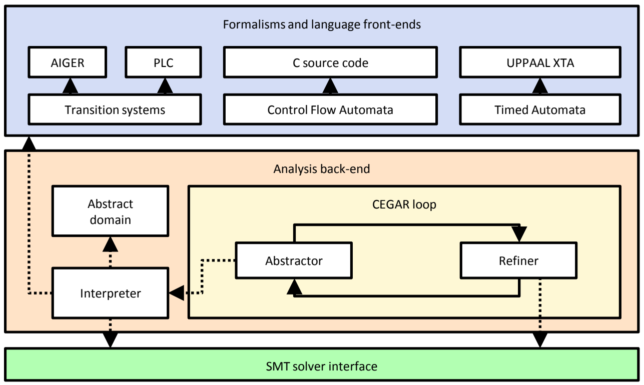
Conclusions

Conclusions

- Formal verification
  - Formal model + property
  - Model checking
- Abstraction-based methods
  - CEGAR
- Theta Framework
  - Generic, modular, configurable
  - Evaluation







hajdua@mit.bme.hu
inf.mit.bme.hu/en/members/hajdua