Software Model Checking with Abstraction-Based Methods

Ákos Hajdu

hajdua@mit.bme.hu

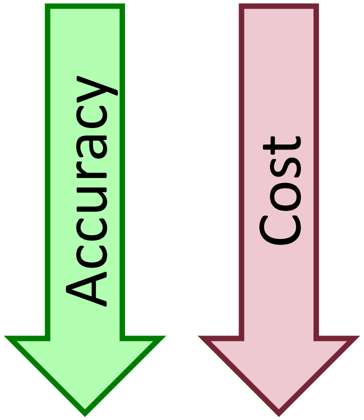
Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

INTRODUCTION

Introduction

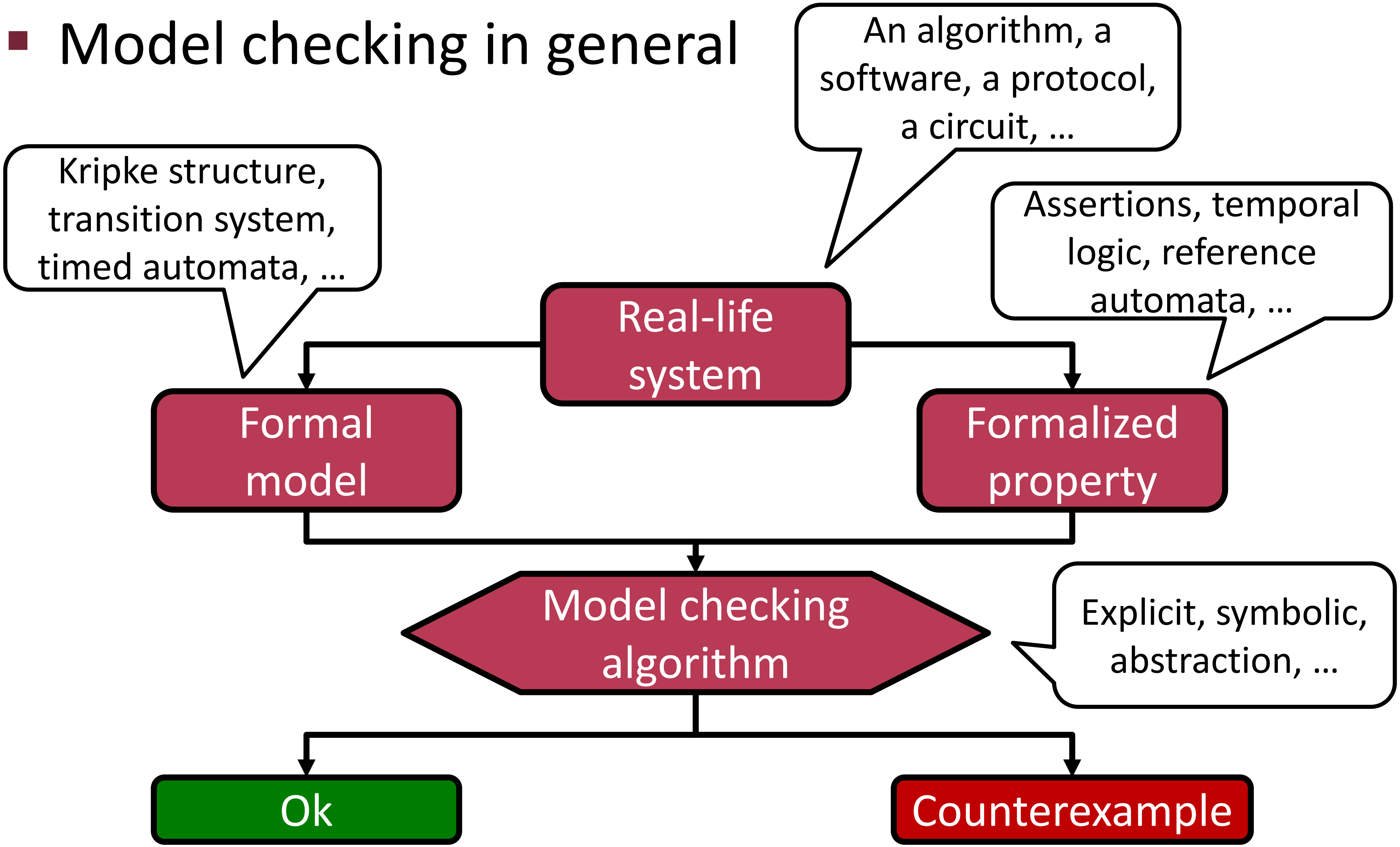
- Motivation
  - Checking the source code directly
  - Should work by “pushing a button”
    - No deep background knowledge should be required

- Typical software verification techniques
  - Static analysis
    - Error patterns
    - Abstract interpretation
  - Model checking



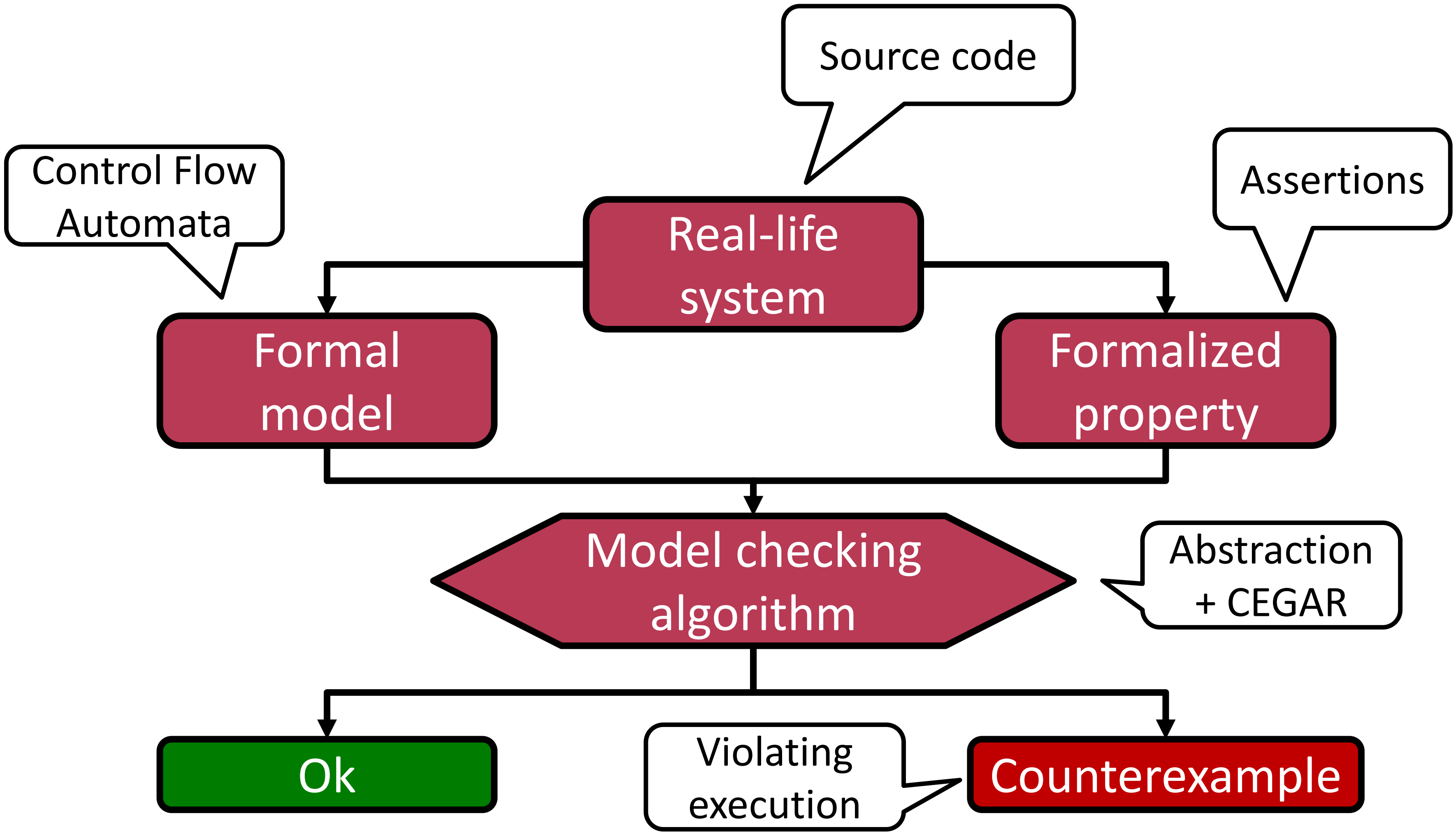
Introduction – Model Checking

- Model checking in general



Introduction – Model Checking

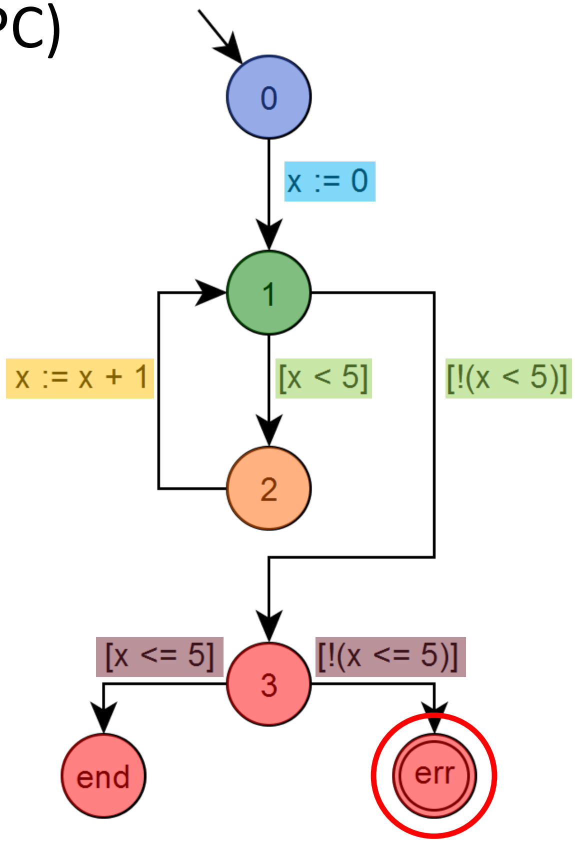
- This lecture: focus on software and abstraction



Introduction – Model and Property

- Control-Flow Automata (CFA)
  - **L** = (l_0, l_1, …): set of control locations (PC)
  - **G**: set of edges with operations over a set of variables
    - E.g., guard, assignment …

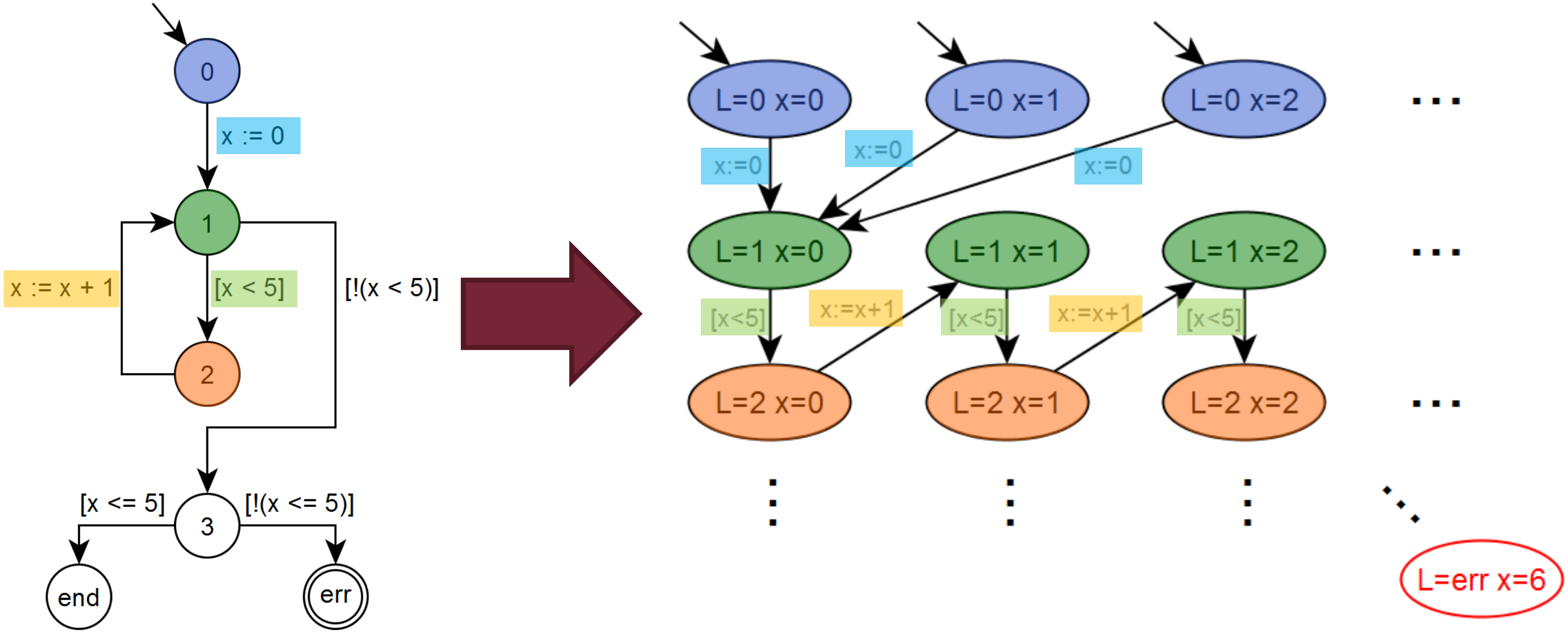
```
   x : int
0: x = 0
1: while (x < 5) {
2:    x = x + 1
   }
3: assert (x <= 5)
```



- Typical property: "error" location (l_E) should not be reachable

Introduction – 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



Introduction – Mathematical Logic

- Propositional logic (PL)
  - Boolean variables and operators
  - SAT problem: is the formula satisfiable
    - Example: bounded model checking
  - Expressive power sometimes not enough

$$\neg p \land (p \lor q)$$

- First order logic (FOL)
  - Functions, predicates, quantifiers
  - Not decidable in general

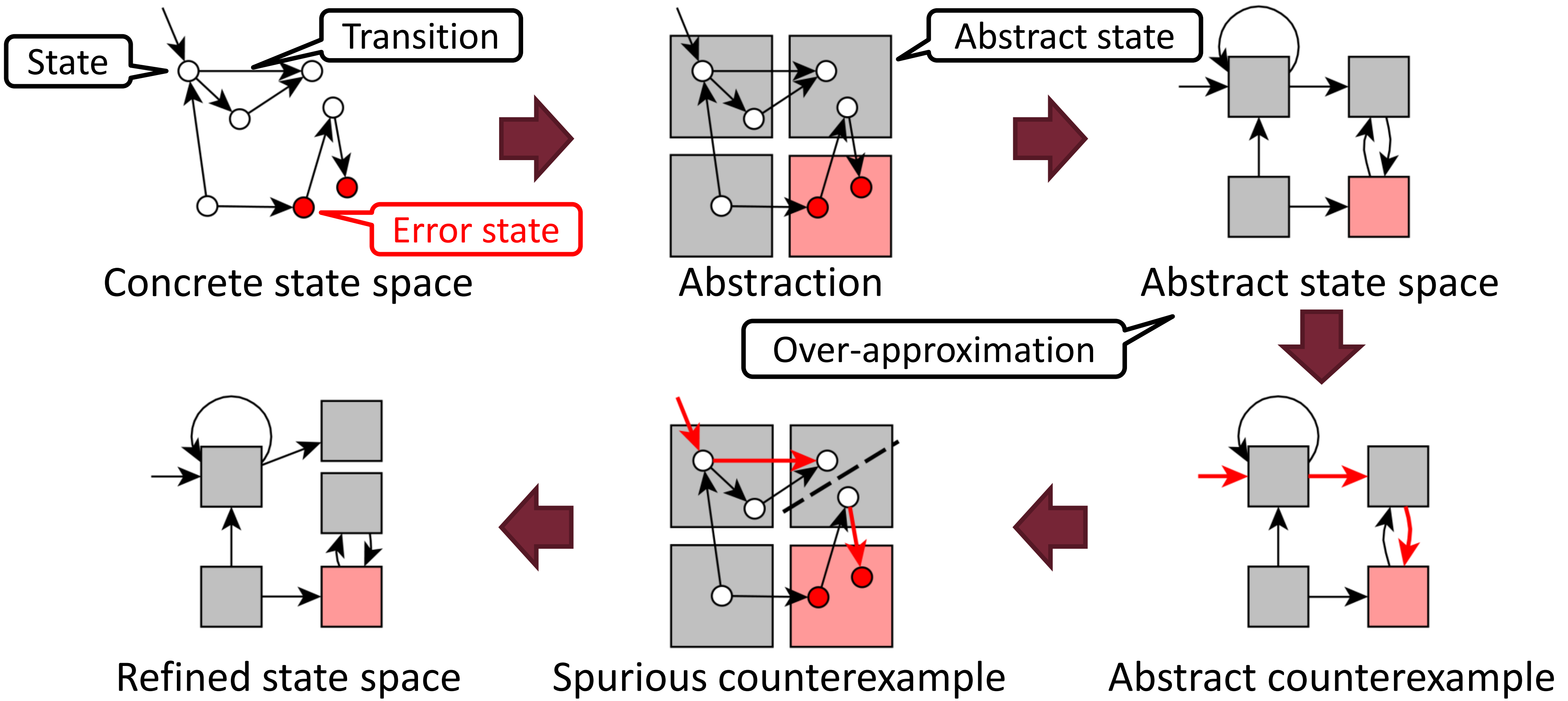
$$\forall x, y\ \exists z: p(f(x,y), g(z))$$

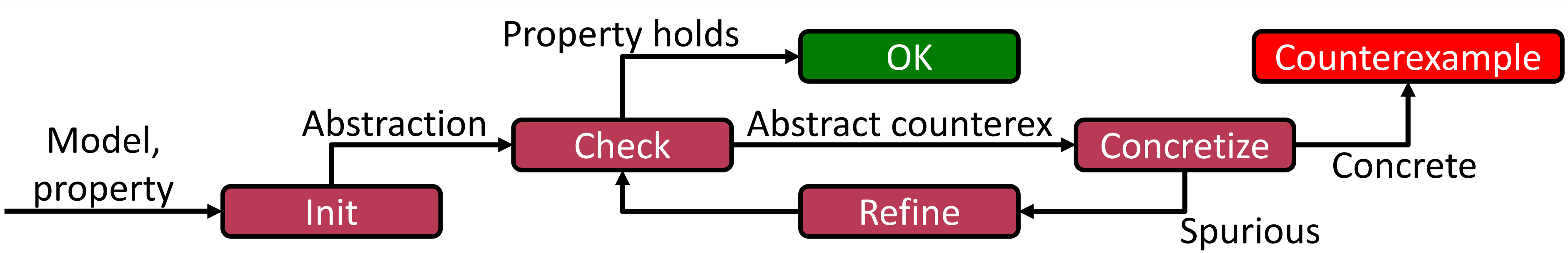
- Satisfiability Modulo Theories (SMT)
  - FOL formulas
  - Interpreted symbols, decidable fragments
    - E.g., integer arithmetic

$$(x \leq y + 1) \land (y \geq 3)$$

COUNTEREXAMPLE-GUIDED ABSTRACTION REFINEMENT (CEGAR)

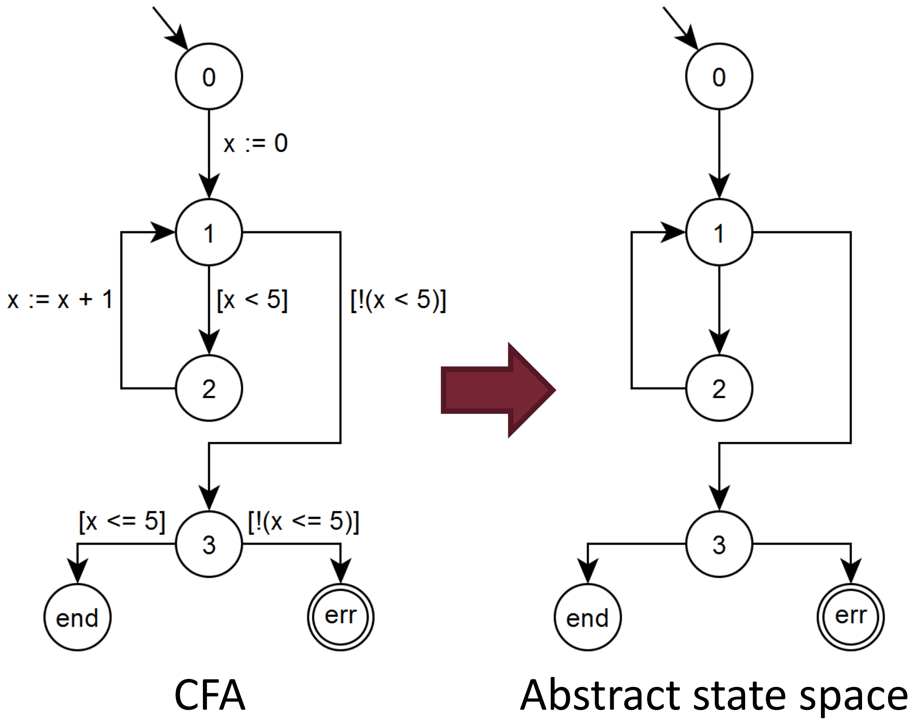
CEGAR – Introduction





Abstraction – Introduction

- Abstraction
  - General mathematical concept
  - Goal: hide details
  - Result: easier problem
- Example
  - Location abstraction
    - $(l, x_1, x_2, \ldots, x_n) \rightarrow (l)$
  - Usually not enough alone
    - Error location trivially reachable
    - Extension with predicate abstraction



Predicate Abstraction

- Predicate abstraction
  - Keep track of predicates instead of concrete values for variables
  - Abstract state: concrete states corresponding to the same location, satisfying the same predicates
- Performing abstraction
  - Enumerate and join concrete states
    - State space explosion ☹
  - 3x3 concrete states in the example → 5 abstract states

Variables:
$x, y; D_x = D_y = \{0,1,2\}$
Predicates:
$(x = y), (x < y), (y = 2)$

$y \backslash x$	0	1	2
0	$(x = y)$		
1	$(x < y)$	$(x = y)$	
2	$(x < y)$ $(y = 2)$	$(x < y)$ $(y = 2)$	$(x = y)$ $(y = 2)$

Predicate Abstraction

- Performing abstraction (differently)
  - Enumerate abstract states only: which one is feasible?
  - Predicate set P → $|L| \cdot 2^{|P|}$ possible abstract states

- Example
  - 3 predicates → 8 possible abstract states (for each location)
  - Some of them are not feasible
    - Use SMT solver to filter
    - E.g. $(x = y) \wedge (x < y) \wedge \neg(y = 2)$ cannot hold at the same time

	$x = y$	$x < y$	$y = 2$
1	X	X	X
~~2~~	~~X~~	~~X~~	~~✓~~
3	X	✓	X
4	X	✓	✓
5	✓	X	X
6	✓	X	✓
~~7~~	~~✓~~	~~✓~~	~~X~~
~~8~~	~~✓~~	~~✓~~	~~✓~~

Predicate Abstraction

- Abstract states with Boolean variables
  - Concrete → Abstract: $(l, x_1, \dots, x_n) \rightarrow (l, b_1, \dots, b_m)$
  - b_i: Boolean variable: ith predicate holds or not
  - Notation: $p(b_i) = \begin{cases} p_i & \text{if } b_i \text{ is true} \\ \neg p_i & \text{otherwise} \end{cases}$
- Example

Variables:

$x, y;\ D_x = D_y = \{0,1,2\}$

Predicates:

$(x = y), (x < y), (y = 2)$

$l\ x\ y$

$(0,0,0) \rightarrow (0, T, F, F)$

$(6,1,2) \rightarrow (6, F, T, T)$

$l\ (x = y)(x < y)(y = 2)$

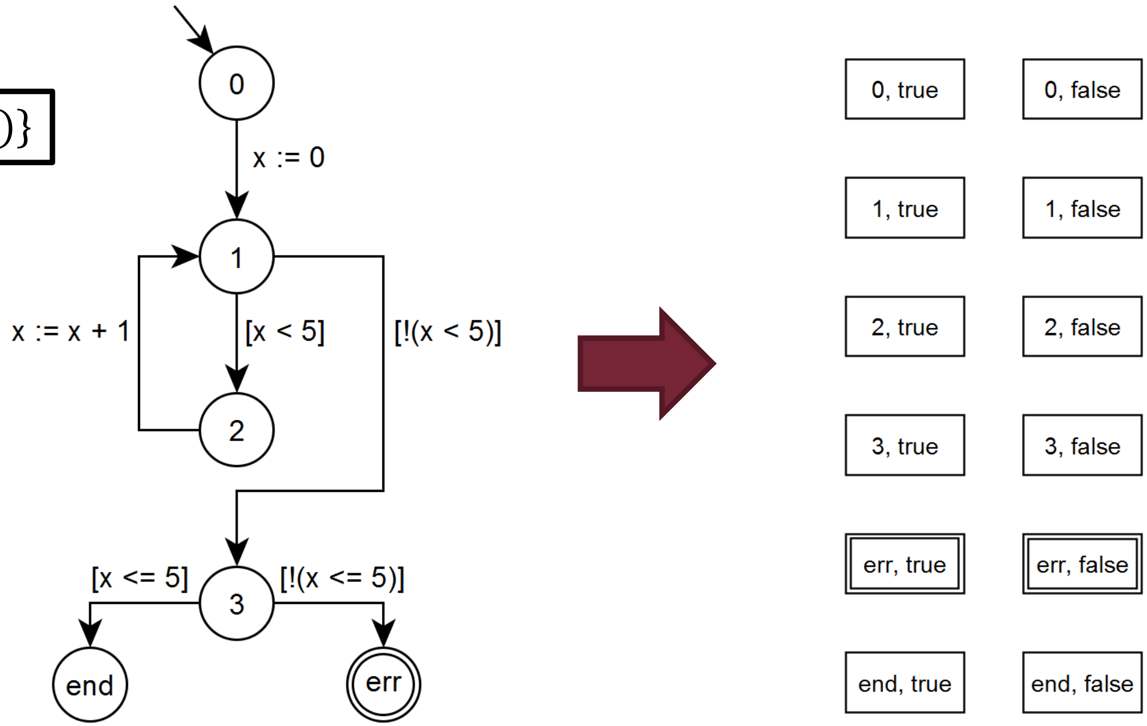
Predicate Abstraction

- Abstract initial states, error states
  - Abstract initial state: $(l, b_1, \dots, b_m)$, where $l = l_0$
  - Abstract error state: $(l, b_1, \dots, b_m)$, where $l = l_E$
- Abstract transitions
  - Abstract transition: at least one concrete transition exists between contained concrete states (existential property of the abstraction)
    - Calculate with SMT solver (without enumerating concrete states)
    - For $(l, b_1, \dots, b_m)$ and $(l', b'_1, \dots, b'_m)$:
      - $\exists op: (l, op, l') \in G$, i.e. there is an edge with operation *op* between locations in the CFA, and
      - $p(b_1) \wedge \dots \wedge p(b_m) \wedge op \wedge p(b'_1) \wedge \dots \wedge p(b'_m)$ is satisfiable

Predicate Abstraction

- Example

$$P = \{(x \leq 5)\}$$

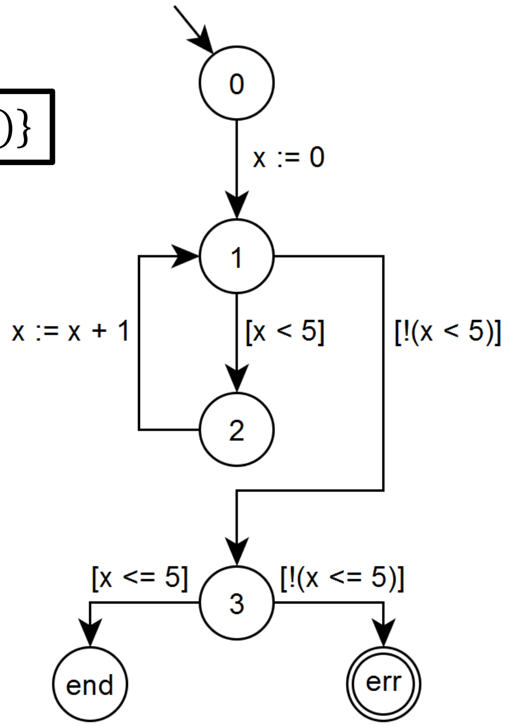


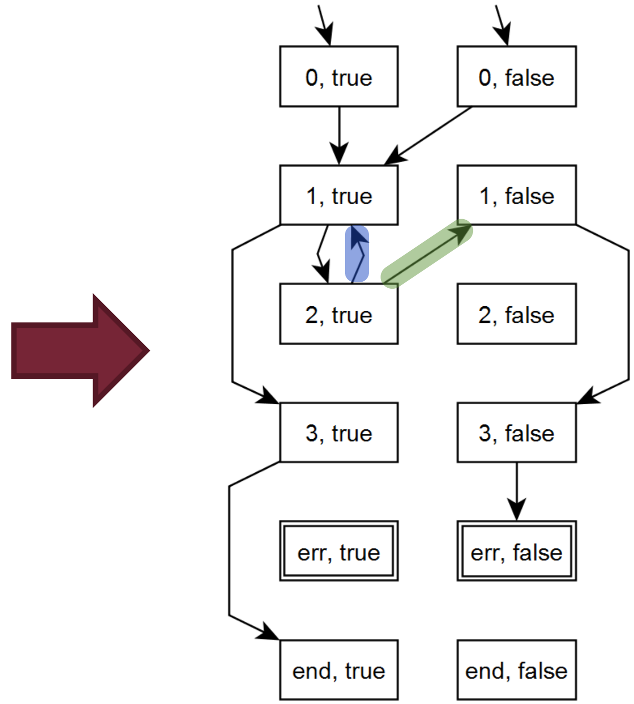
- 6 locations, 1 predicate → 12 abstract states

Predicate Abstraction

- Example

$P = \{(x \leq 5)\}$

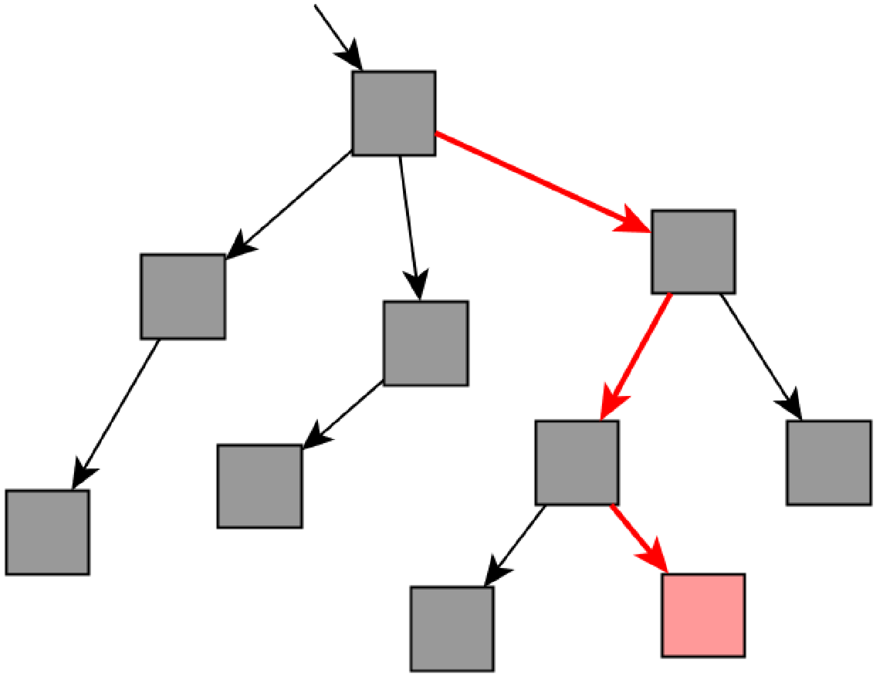




- Transition examples
  - $(2, \text{true}) \rightarrow (1, \text{true})$
    - $(2, x := x + 1, 1) \in G$ and $(x \leq 5) \wedge (x' = x + 1) \wedge (x' \leq 5)$ is satisfiable: $x = 0, x' = 1$
  - $(2, \text{true}) \rightarrow (1, \text{false})$
    - $(2, x := x + 1, 1) \in G$ and $(x \leq 5) \wedge (x' = x + 1) \wedge \neg(x' \leq 5)$ is satisfiable: $x = 5, x' = 6$

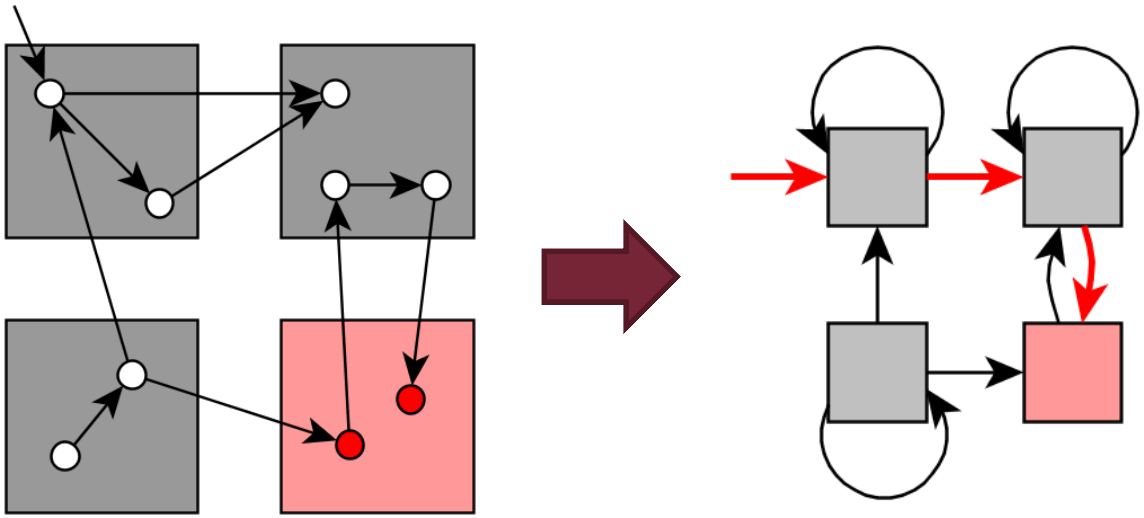
Model Checking

- Traverse abstract state space
  - With some search strategy, e.g., DFS, BFS
  - Search for error state
- Optimizations
  - On-the-fly
    - Calculate abstract states during the search
  - Incremental
    - Do not explore unchanged parts after refinement



Model Checking

- Properties of existential abstraction
  - Over-approximates the original model
    - There is a corresponding abstract path for each concrete path
    - If no abstract path exists to error state: concrete does not exist either
    - If abstract path exists to error state: concrete path may exist
  - Therefore, abstract counterexamples must be checked
    - Does it have a corresponding concrete path?

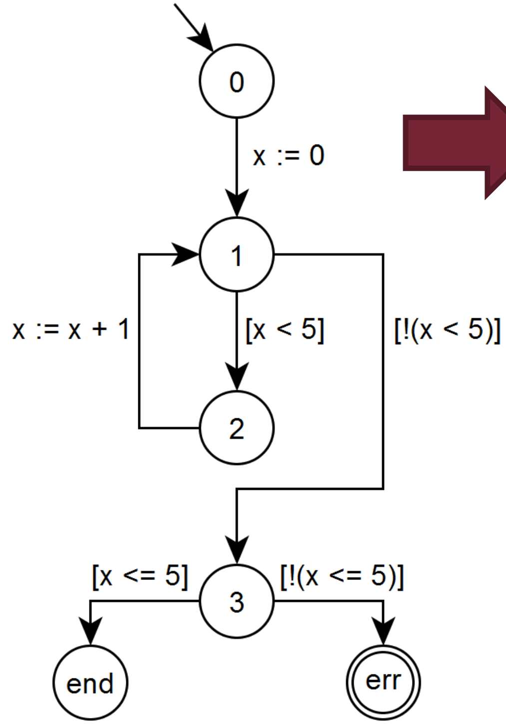


Abstract Counterexample

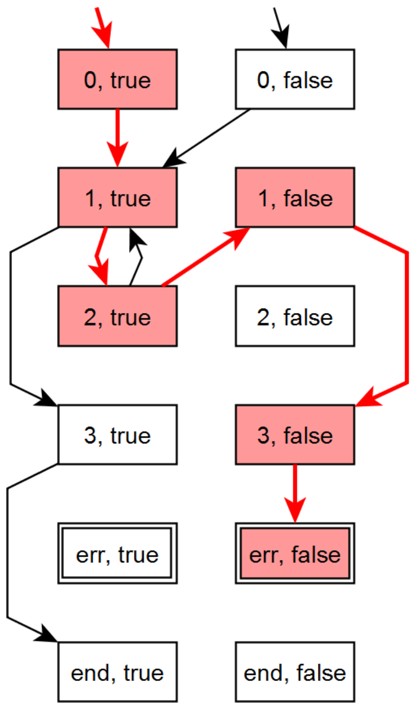
- Form of abstract counterexample
  - Sequence of locations and predicates
  - $(l_1, b_{1,1}, \dots, b_{1,m}), (l_2, b_{2,1}, \dots, b_{2,m}), \dots, (l_n, b_{n,1}, \dots, b_{n,m})$
- Finding a concrete path → traverse a part of the concrete state space
  - Guided by abstract counterexample (what should be explored)
  - SMT solver can be used
    - Similarly to bounded model checking (BMC)
    - Generalize the method presented at existential abstraction for n steps
- Concrete path exists → concrete model is faulty
- Concrete path does not exist → spurious counterexample

Abstract Counterexample

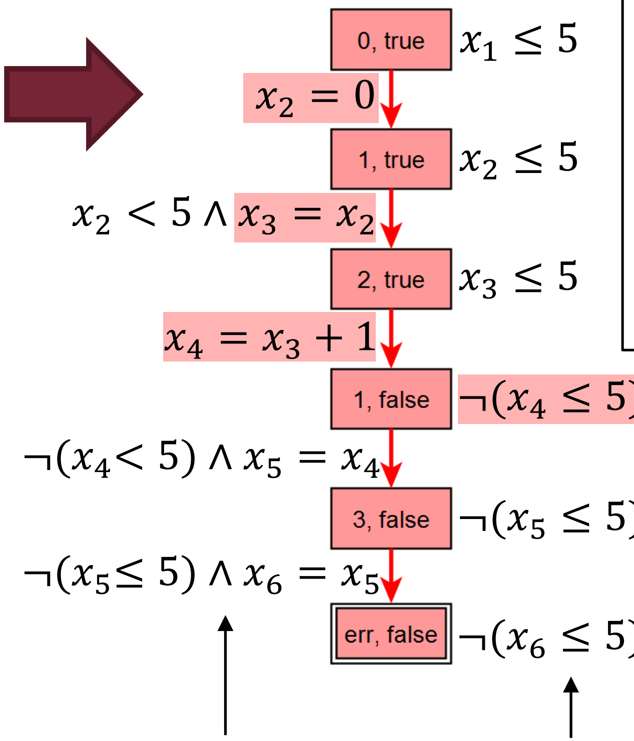
- Example



$P = \{(x \leq 5)\}$



Abstract counterexample (6 states)



Operations on transitions

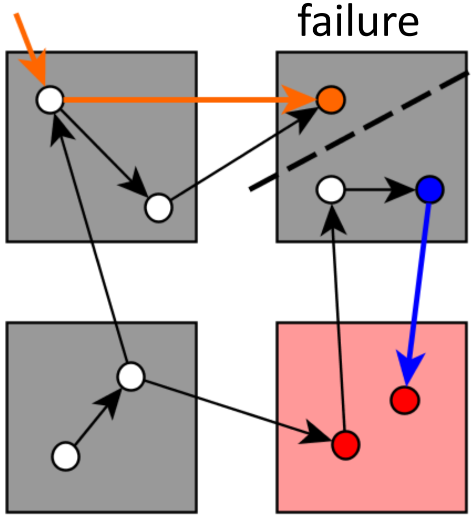
Predicates of states

$(x_1 = 3)$
$(x_2 = 0)$
$(x_3 = 0)$
$(x_4 = ?)$

Not satisfiable

Spurious Counterexample

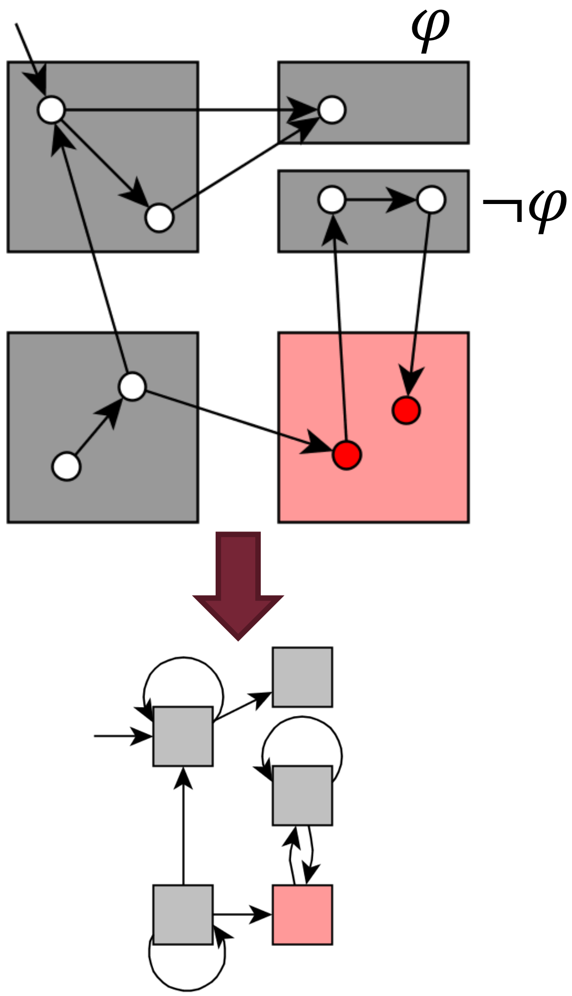
- "Failure" state: A concrete path exists until a state and after, but these are separate in the concrete model
- Group concrete states mapped to "failure" state
  - D = "Dead-end": reachable
  - B = "Bad": transition to next state
  - IR = "Irrelevant": others



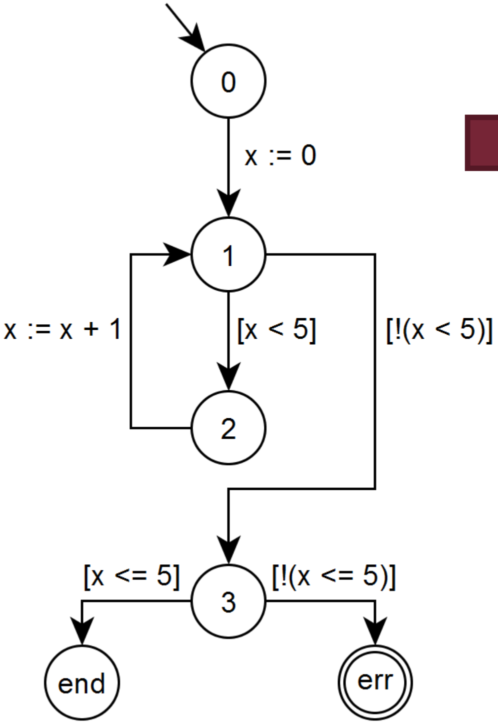
- Reason for spurious counterexample
  - Set of predicates does not distinguish D and B

Abstraction Refinement

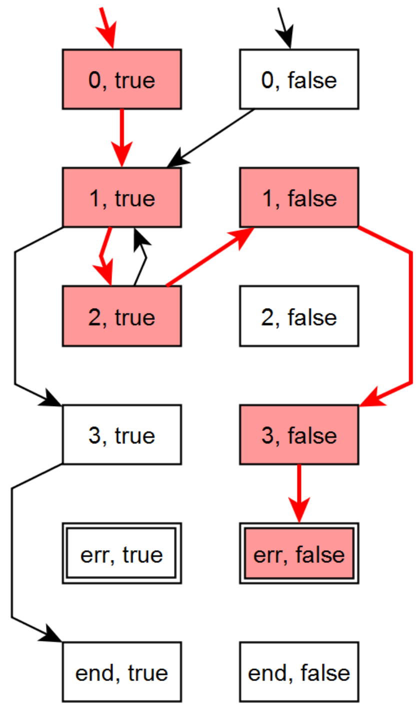
- Eliminating the spurious counterexample
  - More predicates (finer abstraction)
  - Separate D and B
    - Without enumerating concrete states
    - Describe D and B with formulas
    - SMT solver can generate a formula φ that separates (holds for D but not for B)
    - This method is called *interpolation*
  - The set $P \cup \{\varphi\}$ will eliminate this spurious counterexample
    - Moreover it is enough to split only the failure state *(lazy abstraction)*
- Additional spurious counterexamples
  - More predicates



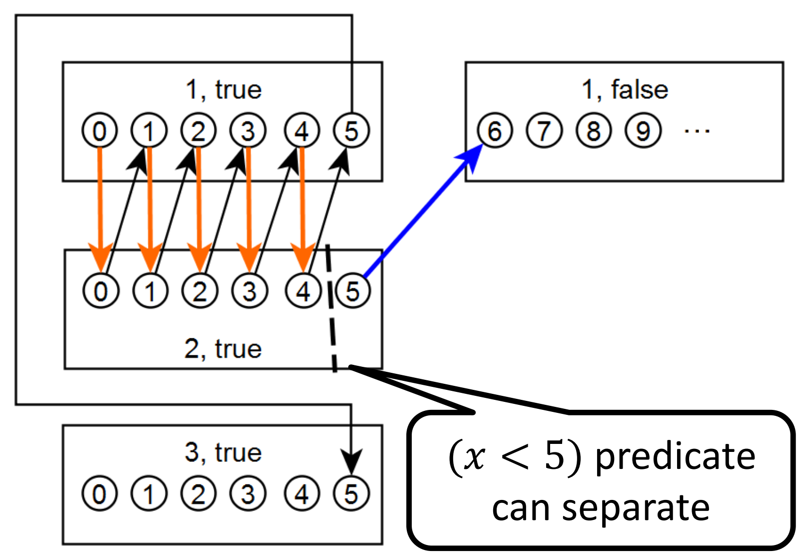
- Example



$P = \{(x \leq 5)\}$

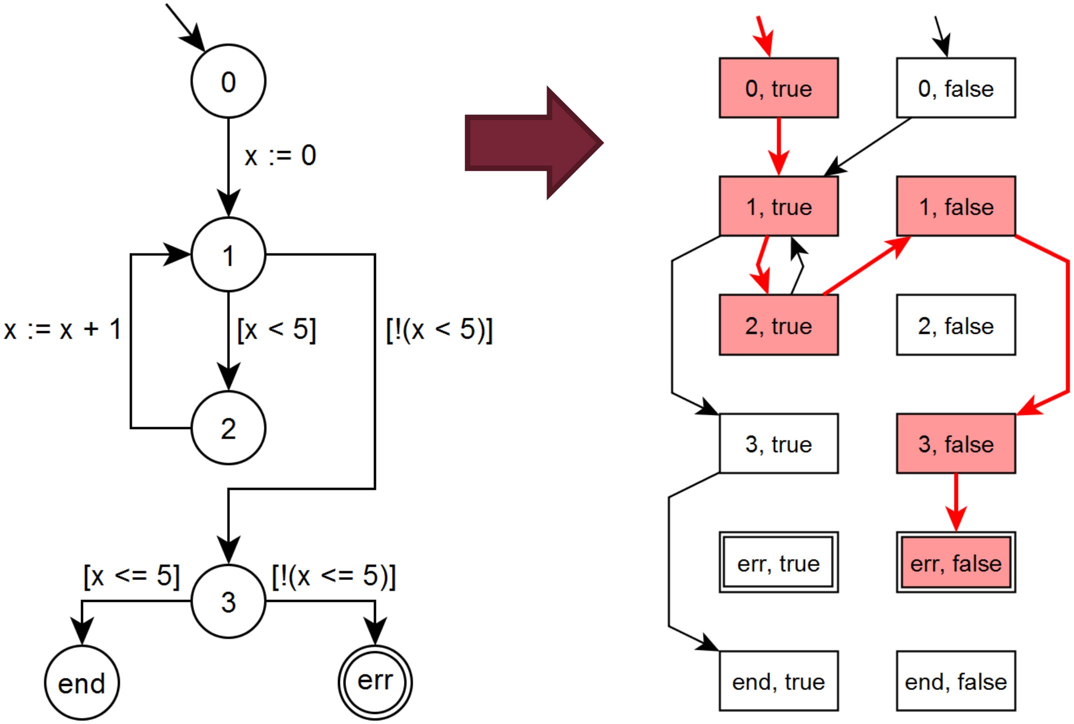


$(x_1 = 3)$
$(x_2 = 0)$
$(x_3 = 0)$
$(x_4 = ?)$



Abstraction Refinement

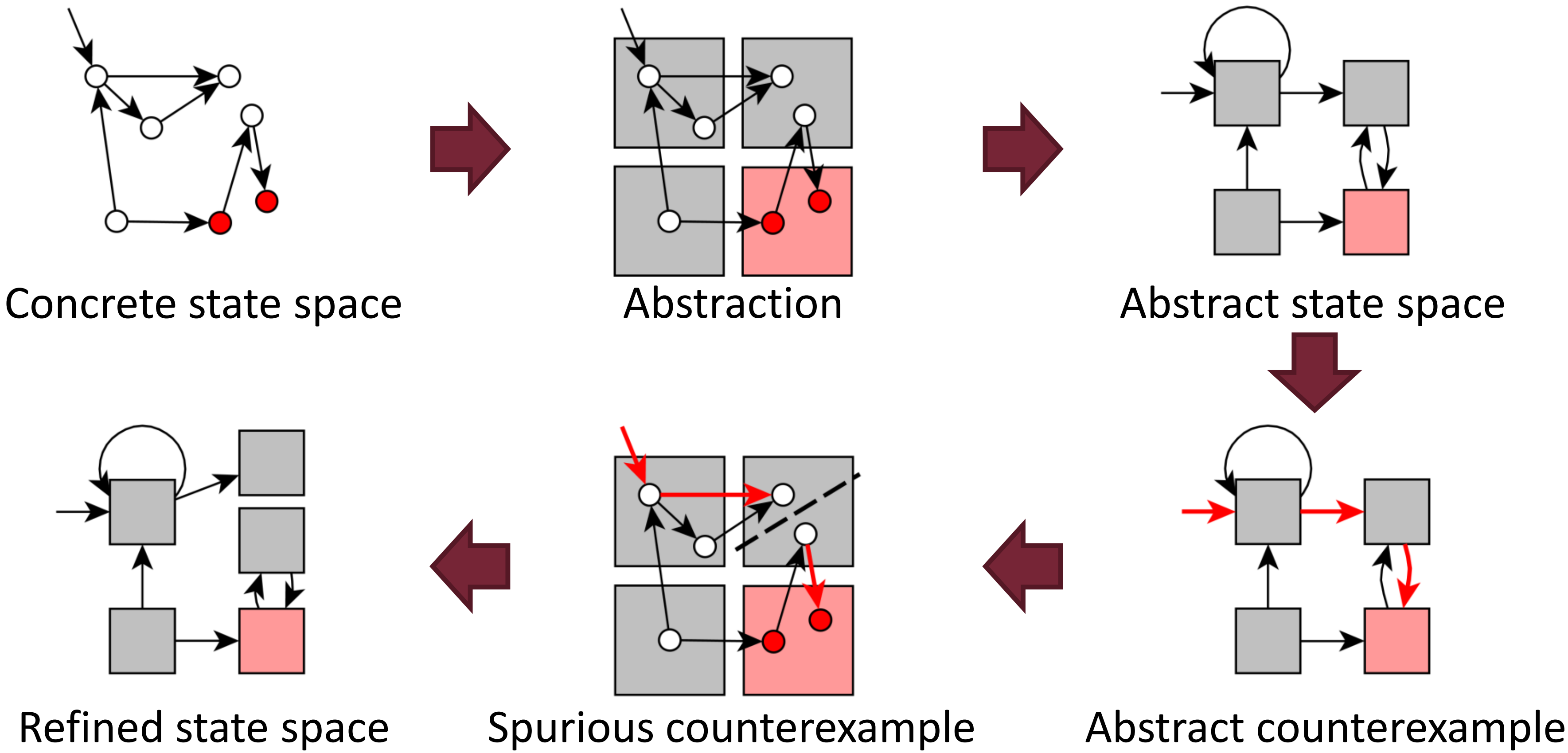
- Example

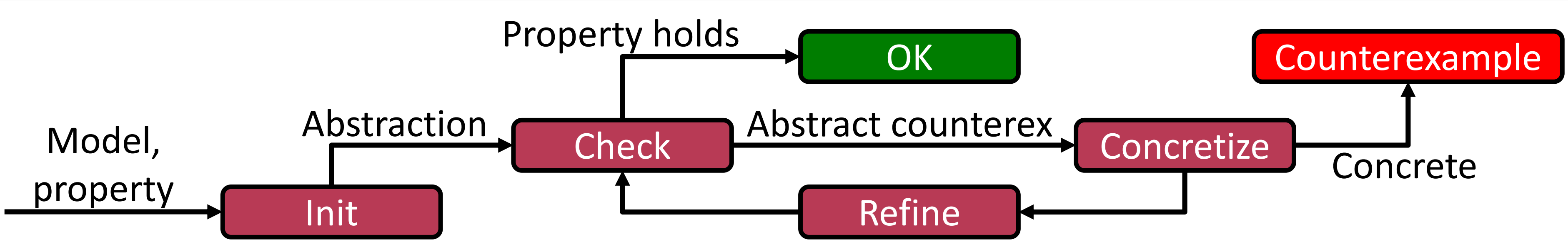


$P = \{(x \leq 5)\}$

$P = \{(x \leq 5), (x < 5)\}$

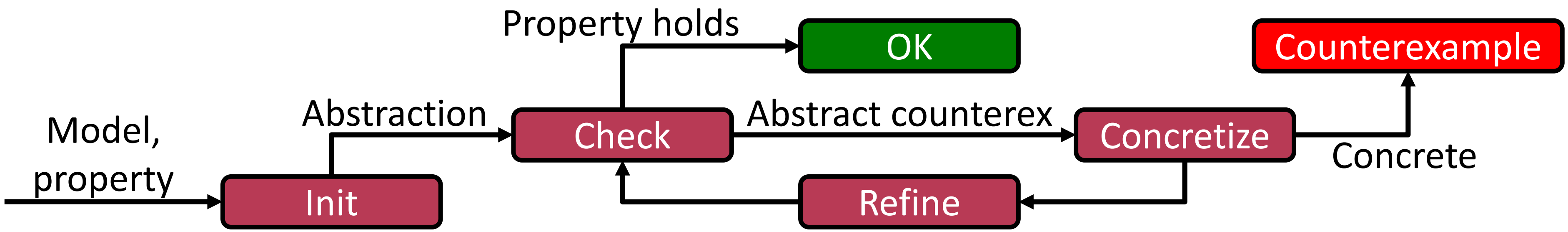
CEGAR – Summary





The algorithm

- Counterexample-Guided Abstraction Refinement (CEGAR)
  - Automatic method
    - Each step is automatic
    - Deep knowledge of formal methods is not required
  - How about the initial set of predicates?
    - It can be an empty set (the algorithm will split states)
    - It can come from conditional statements in the software
    - Other heuristics



TOOLS

Tools

- SLAM2
  - Part of Static Driver Verifier Research Platform (SDVRP)
  - Structure
    - Driver C code: analyzed component
    - Platform model: describe environment
    - Analysis: adherence to API usage rules
  - Algorithms
    - Create Boolean program with predicate abstraction
    - Symbolic model checking: BEBOP tool
    - CEGAR loop
  - research.microsoft.com/en-us/projects/slam/

Tools

- BLAST
  - Berkeley Lazy Abstraction Software Verification Tool
  - Input: C program + requirement (BLAST Query Language)
  - Predicate abstraction
    - Building abstract reachability tree (ART)
  - Refinement: new predicate with interpolation
    - Lazy abstraction: apply new predicate locally
  - Limitations: multiplication, bit operations, overflow
  - mtc.epfl.ch/software-tools/blast/index-epfl.php

Tools

- CPAchecker
  - The Configurable Software-Verification Platform
  - Input: C program + specification
    - Assertion, error label, deadlock, null dereference, …
  - Highly configurable
    - Different kinds of abstractions (not only predicate)
    - Can consider multiple prefixes of a counterexample
      - Chooses from different refinements (refinement strategy)
  - cpachecker.sosy-lab.org/

Tools

- Theta
  - Generic, modular, configurable model checking framework
  - Developed at BME-MIT
  - Generic: various kinds of formal models
    - Transition systems, control flow automata, timed automata
  - Modular: reusable and combinable modules
  - Configurable: different algorithms and strategies
  - github.com/FTSRG/theta

Tools

- Competition on Software Verification 2017 (SV-COMP)
  - sv-comp.sosy-lab.org/2017/
  - 32 tools, 8908 input tasks (program + requirement)
  - Categories
    - Arrays (ArraysReach, ArraysMemSafety)
    - Bit Vectors (BitVectorsReach, Overflows)
    - Heap Data Structures (HeapReach, HeapMemSafety)
    - Floats
    - Integers and Control Flow (ControlFlow, Simple, ECA, Loops, Recursive, ProductLines, Sequentialized)
    - Termination
    - Concurrency
    - Software Systems (DeviceDriversLinux64, BusyBox)

SUMMARY

Summary

- Software model checking
  - Verification by “pushing a button”
    - Problem to be solved: state space explosion
  - Solution: abstraction
    - Location + predicates
    - Properties of existential abstraction
  - CEGAR: automatically obtain proper abstraction
    1. Initial abstraction
    2. Model checking
    3. Examining the counterexample
    4. Refining the abstraction
  - Tools