Scalability in Model Checking

CS60030 FORMAL SYSTEMS

PALLAB DASGUPTA,
FNAE, FASc,
A K Singh Distinguished Professor in AI,
Dept of Computer Science & Engineering
Indian Institute of Technology Kharagpur
Email: pallab@cse.iitkgp.ac.in
Web: http://cse.iitkgp.ac.in/~pallab

FMSAFE
FORMAL METHODS FOR SAFETY CRITICAL SYSTEMS

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Handling Large State Spaces

State Explosion

- If M has *k* state variables, then it has 2^k states
- Not all these states are reachable
- Not all state variables are relevant for a property we wish to prove

Representation

- Symbolic – we will never actually generate the explicit state space
- Reduced – throw out those state variables that are inconsequential

Decision strategies

- Proving the property on an abstraction of M may be sufficient
- Proving the property assuming all states are reachable may be sufficient

More on scalability

- BDD, SAT, SMT
  - Not good enough for many of the state spaces where we wish to use formal methods

<u>OPTIONS</u> (we will elaborate each of these)

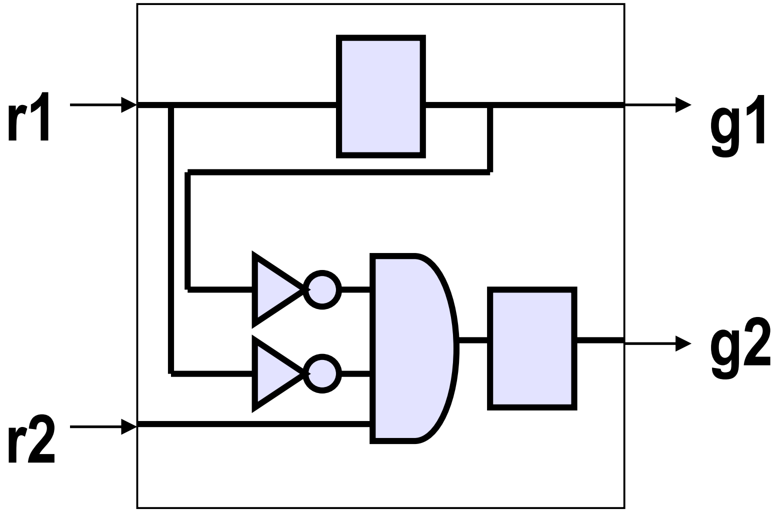
- BOUNDED SEARCH
  - In many cases we may know an upper-bound on the length of potential counter-examples
  - We can unfold only up to that depth
- INDUCTION
  - We can inductively prove certain properties with limited unfolding
- ABSTRACTION – REFINEMENT
  - We reduce the complexity of the STS by dropping some of its variables and prove that the abstraction is safe

BOUNDED MODEL CHECKING

Bounded Model Checking

- Represent sets of states and the transition relation as Boolean logic formulas
- Instead of computing the fixpoints, unroll the transition relation up to certain fixed bound and search for violations of the property within that bound
- Transform this search to a Boolean satisfiability problem and solve it using a SAT solver

Example: *Bound*=2



Is there a witness of length=2?

Clauses from Transition Relation:

C_1^1: $r2^0 \wedge \neg r1^0 \wedge \neg g1^0 \Rightarrow g2^1$

C_2^1: $r1^0 \Rightarrow g1^1$

Clauses from Initial State:

I: $g2^0 \wedge \neg g1^0$

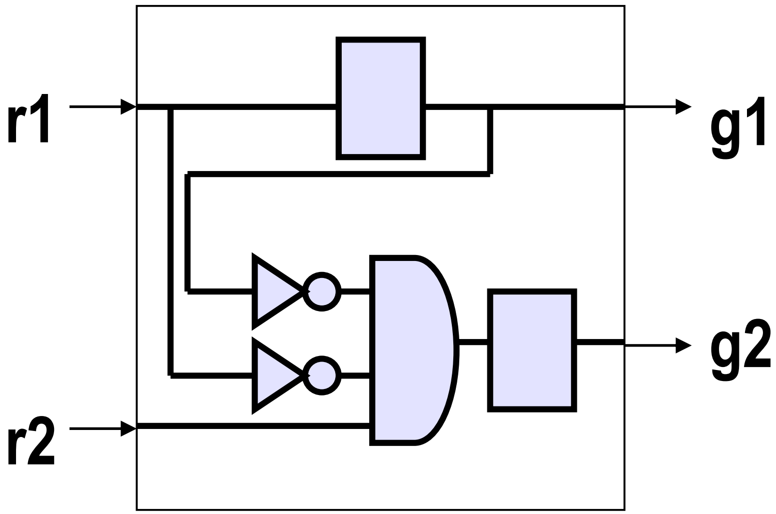
Clauses from Property: $F(r1 \wedge (\neg Xg1 \vee \neg XXg1))$

Z^1: $r1^0 \wedge \neg g1^1$

SAT Check: Is $Z^1 \wedge I \wedge C_1^1 \wedge C_2^1$ satisfiable?

Answer: No, since Z^1 conflicts with C_2^1

Example: *Bound*=3



Is there a witness of length=3?

Clauses from Transition Relation:

C_1^1, C_2^1: from previous iteration

C_1^2: $r2^1 \wedge \neg r1^1 \wedge \neg g1^1 \Rightarrow g2^2$

C_2^2: $r1^1 \Rightarrow g1^2$

Clauses from Initial State:

I: $g2^0 \wedge \neg g1^0$

Clauses from Property: F(r1 $\wedge$ ($\neg$Xg1 $\vee$ $\neg$XXg1))

Z^2: $(r1^0 \wedge (\neg g1^1 \vee \neg g1^2)) \vee (r1^1 \wedge \neg g_1^{\,2})$

SAT Check: Is $Z^2 \wedge I \wedge C_1^1 \wedge C_2^1 \wedge C_1^2 \wedge C_2^2$ satisfiable?

Yes: Witness: $r1^0 = 1$, $r1^1 = 0$, $g1^1 = 1$, $g1^2 = 0$, rest are don't cares

Conclusion: We have found a bug!!

What Can We Guarantee?

Note that we are checking only for bounded paths (paths which have at most k+1 distinct states)

- So if the property is violated by only paths with more than k+1 distinct states, we would not find a counter-example using bounded model checking
- Hence if we do not find a counter-example using bounded model checking we are not sure that the property holds

However, if we find a counter-example, then we are sure that the property is violated since the generated counter-example is never spurious (i.e., it is always a concrete counter-example)

Proving Correctness

If we can find a way to figure out when we should stop then we would be able to provide guarantee of correctness.

There is a way to define a *diameter* of a transition system so that a property holds for the transition system if and only if it is not violated on a path bounded by the diameter.

So if we do bounded model checking using the diameter of the system as our bound, then we can guarantee correctness if no counter-example is found.

Formal Methodology

Bound on path length k

Clauses describing the system M :

- Initial state : $I(s_1)$
- Unrolled transition relation : $\bigwedge_{i=1..k-1} T(s_i, s_{i+1})$

Loop clause $loop_k = \bigvee_{i=1..k} T(s_k, s_i)$

$[f]_{i,k}$ means that temporal property f is true at runs starting from s_i and provable in k BMC iterations.

For the property f to hold on the system $M \wedge [f]_{1,k}$ must be valid.

Translation of LTL to SAT

Xf is true at state s_i , iff f is provable starting from s_{i+1}

$$[\,X f\,]_{i,k} = (i < k) \wedge [\,f\,]_{i+1,k}$$

Ff is true in state s_i , iff f is provable within k iterations from some future state s_j

$$[\,F f\,]_{ik} = \vee_{j=i..k} [\,f\,]_{j,k}$$

Gf is true in state s_i , iff f is true at all states reachable in k iterations and all paths loop

$$[\,G f\,]_{i,k} = \wedge_{j=i..k} [\,f\,]_{j,k} \wedge loop_k$$

f U g is true at s_i , iff g is provable from some state reachable within k iterations and f is provable from all preceding states within k iterations

$$[\,f\,U\,g\,]_{i,k} = \vee_{j=i..k}(\,[\,g\,]_{j,k} \wedge \wedge_{n=i..j-1}[\,f\,]_{n,k})$$

ABSTRACTION REFINEMENT

Cone-of-influence reduction

Two state variables

- b and d

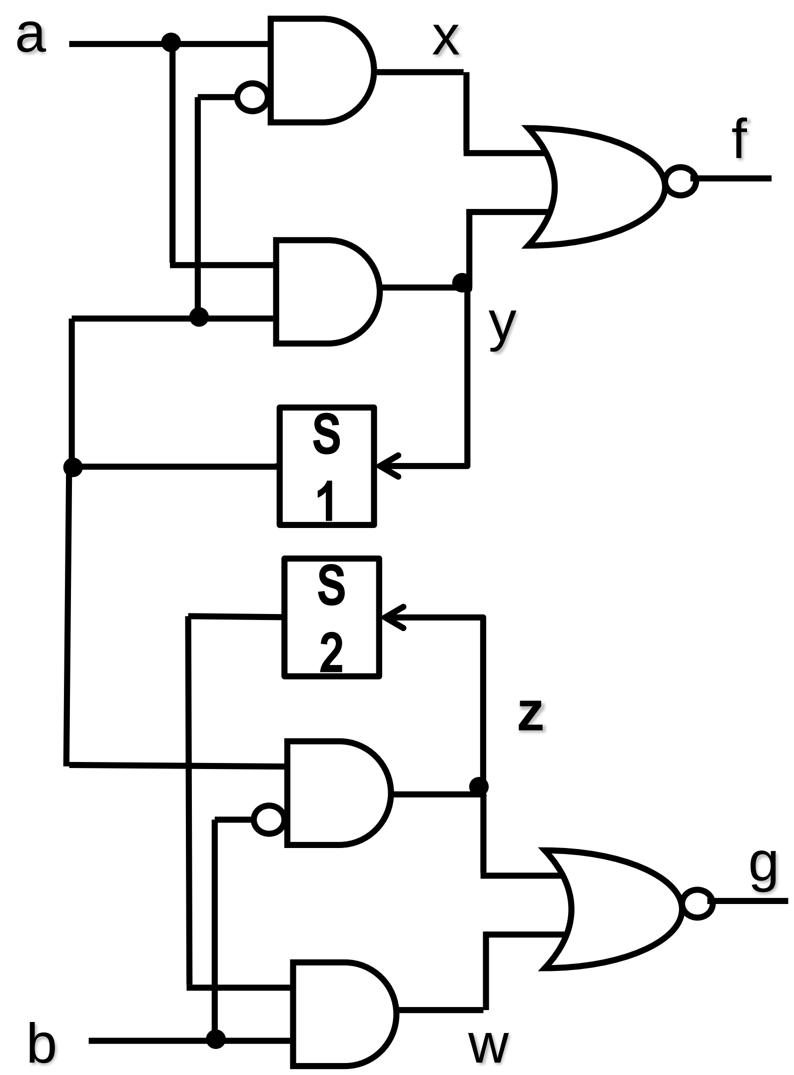
The value of *f* is influenced by:

- Input *a*
- State S1

The value of *g* is influenced by:

- Input *b*
- State S2
- State S1, because S2 is influenced by it
- Input *a*, because S1 is influenced by it

Computable using static analysis



Abstraction

Cone-of-influence reduction with respect to a property does not loose any relevant information

- The problem is that quite often COI is not enough

Abstractions further reduce the size of the state machine

What kind of abstractions do we want?

- Bugs must not escape detection.
- This is guaranteed by the following constraint:
  - Any run which exists in the original state machine must also exist in the abstract state machine
- This is achieved by *existential abstraction* of the transition relation

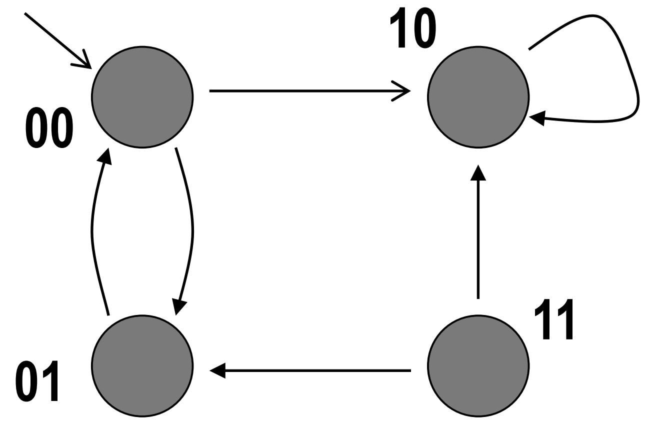
Existential Abstraction

In this example, we eliminate x_2

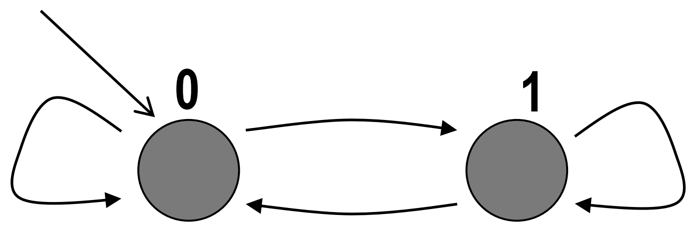
- Let h(s) denote the abstract state corresponding to a state s in the original machine
- Existential abstraction:

  $T(s_a, s_b) \Rightarrow T'(\, h(s_a), h(s_b)\,)$
- In other words:

  $T'(\, s_i, s_j\,) \Rightarrow \exists\, s_a, s_b\ T(s_a, s_b)$ such that $h(s_a) = s_i$ and $h(s_b) = s_j$



$T(x_1, x_2)$



$T'(x_1)$

Existential Abstraction

Corresponding to every run in the original state machine, we have a run in the abstract state machine

- Therefore counterexamples in the original machine (if any) are preserved in the abstract state machine
- If a property holds on the abstract state machine, then it also holds in the original state machine

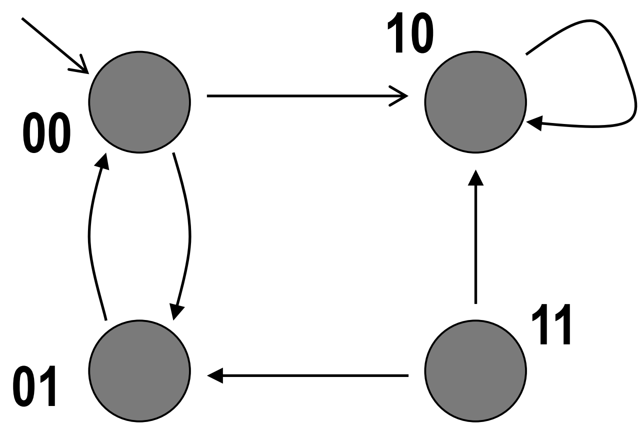
Problem: A counterexample found in the abstract state machine is not necessarily real

False counterexample

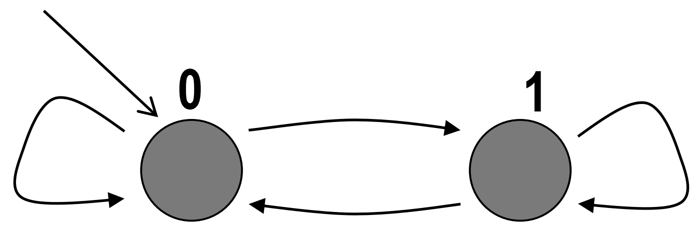
Consider the property

$G(x_1 \Rightarrow G (x_1))$

- Whenever x_1 goes high, it stays high
- This is true in the original state machine
  (look at the reachable states only)
- *But it is false in the abstract state m/c*

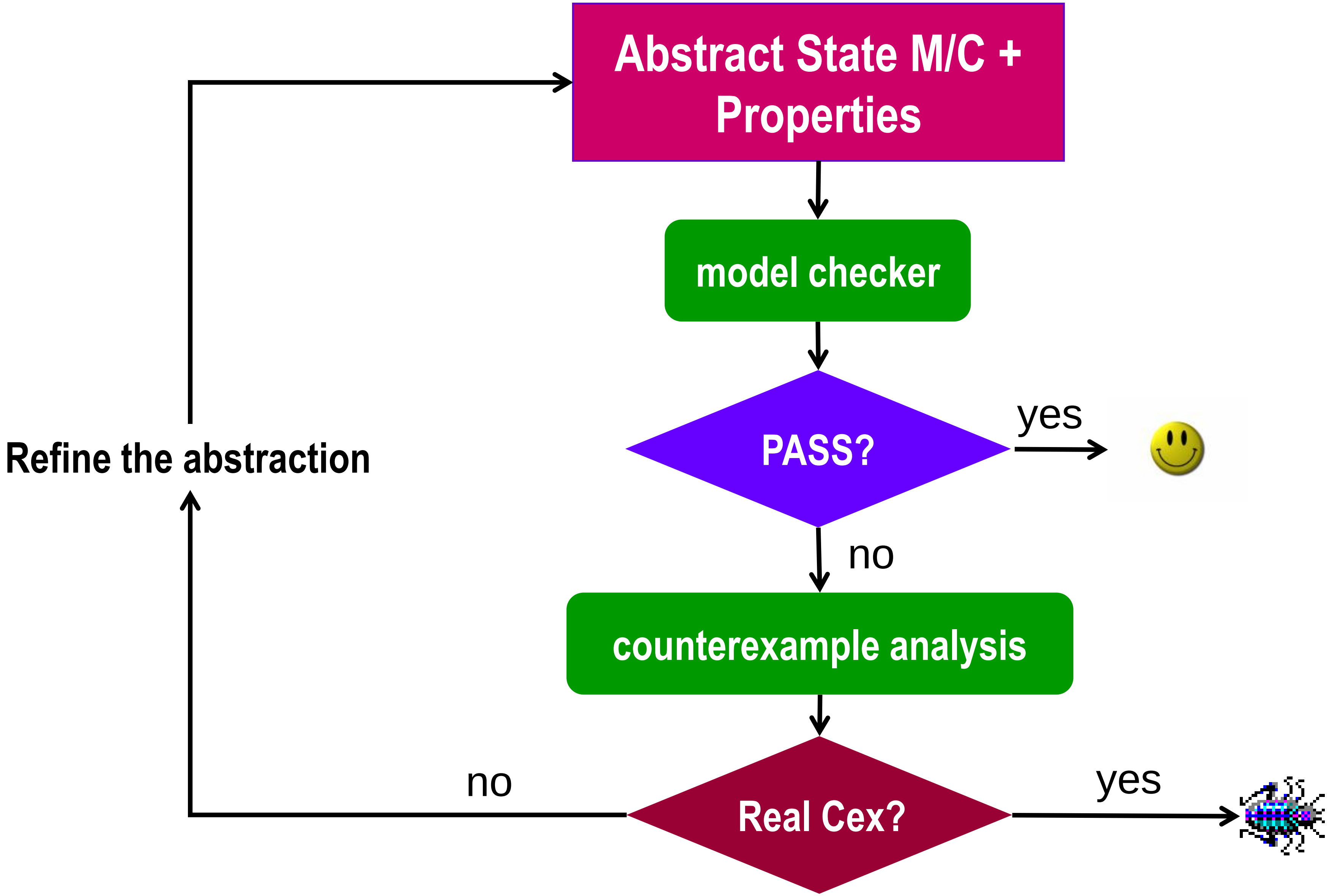


$T(x_1, x_2)$



$T'(x_1)$

Abstraction Refinement



Checking the Counterexample

Counterexample: $(c_1, \ldots, c_m)$

- **Each c_i is an assignment to the set of remaining state variables.**

Concrete traces corresponding to the counterexample:

$$\varphi = I(s_1) \wedge \left(\bigwedge_{i=1}^{m-1} R(s_i, s_{i+1}) \right) \wedge \left(\bigwedge_{i=1}^{m} h(s_i) = c_i \right)$$

Initial State

Unrolled Transition Relation

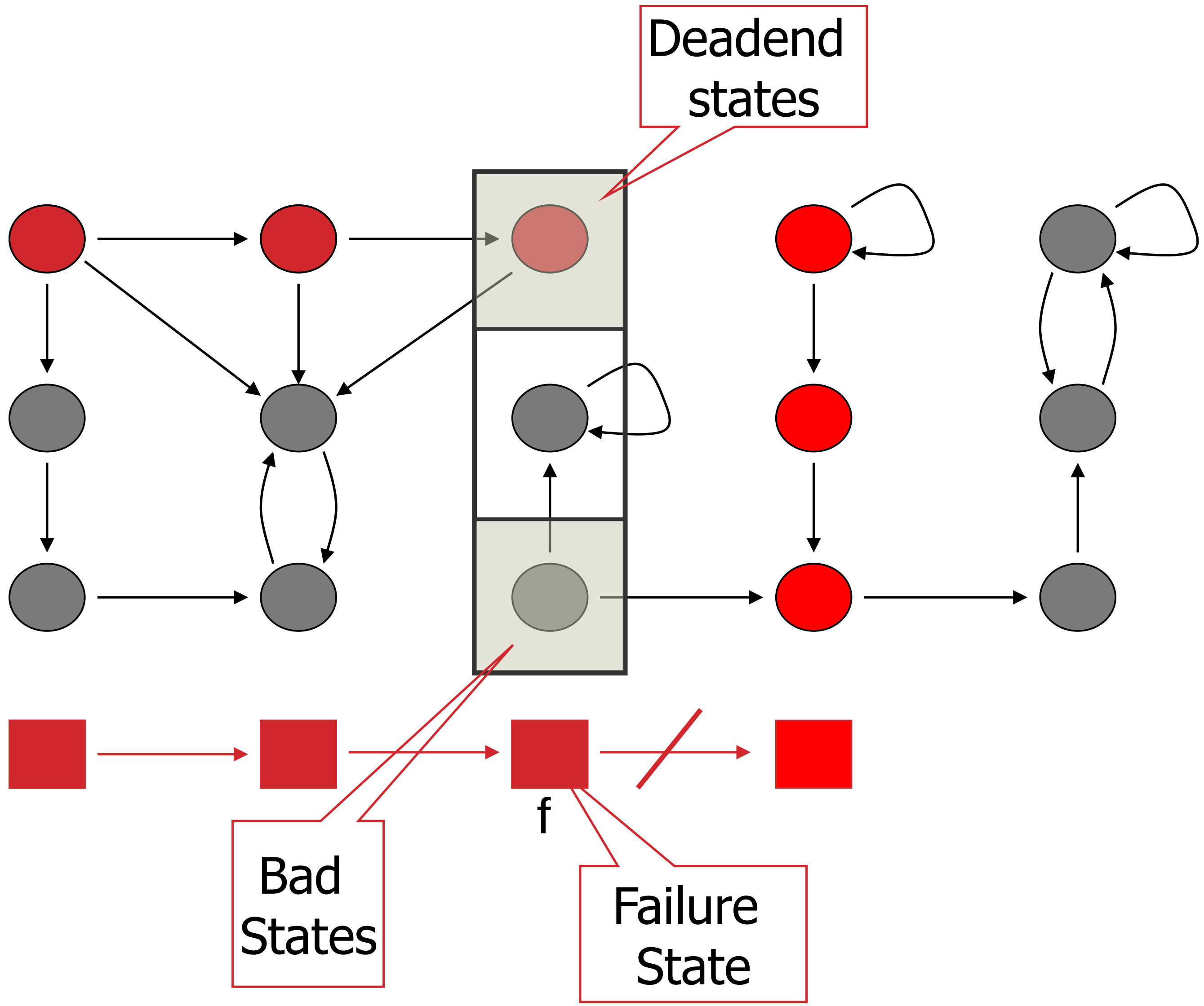
Compliance with counterexample

Abstraction/Refinement with conflict analysis

- **Simulate counterexample on concrete model with SAT**
- **If the instance is unsatisfiable, analyze conflict**
- **Make visible one of the variables in the clauses that lead to the conflict**

Source: Chauhan, Clarke, Kukula, Sapra, Veith, Wang, FMCAD 2002

Why do we get spurious counterexample?

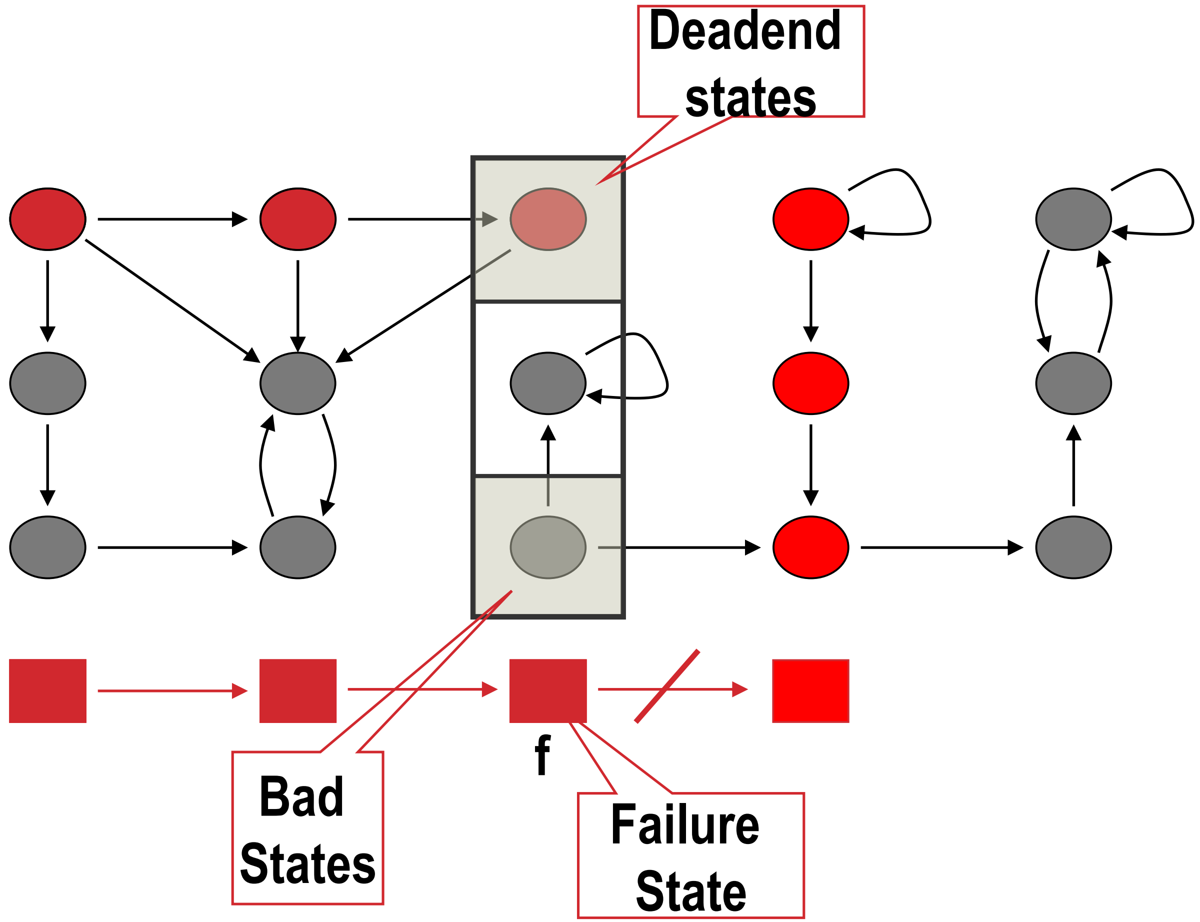


Refinement

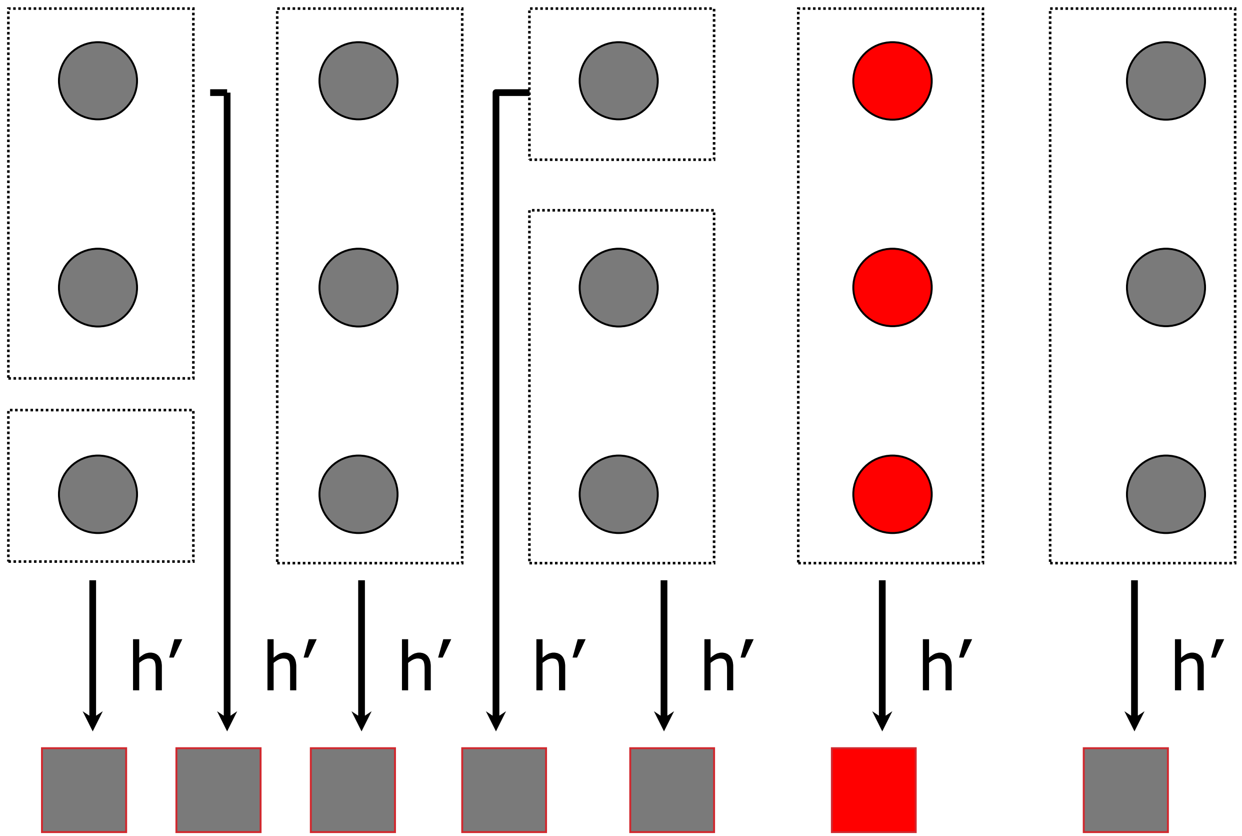
Problem: Deadend and Bad States are in the same abstract state.

Solution: Refine abstraction function.

The sets of Deadend and Bad states should be separated into different abstract states.

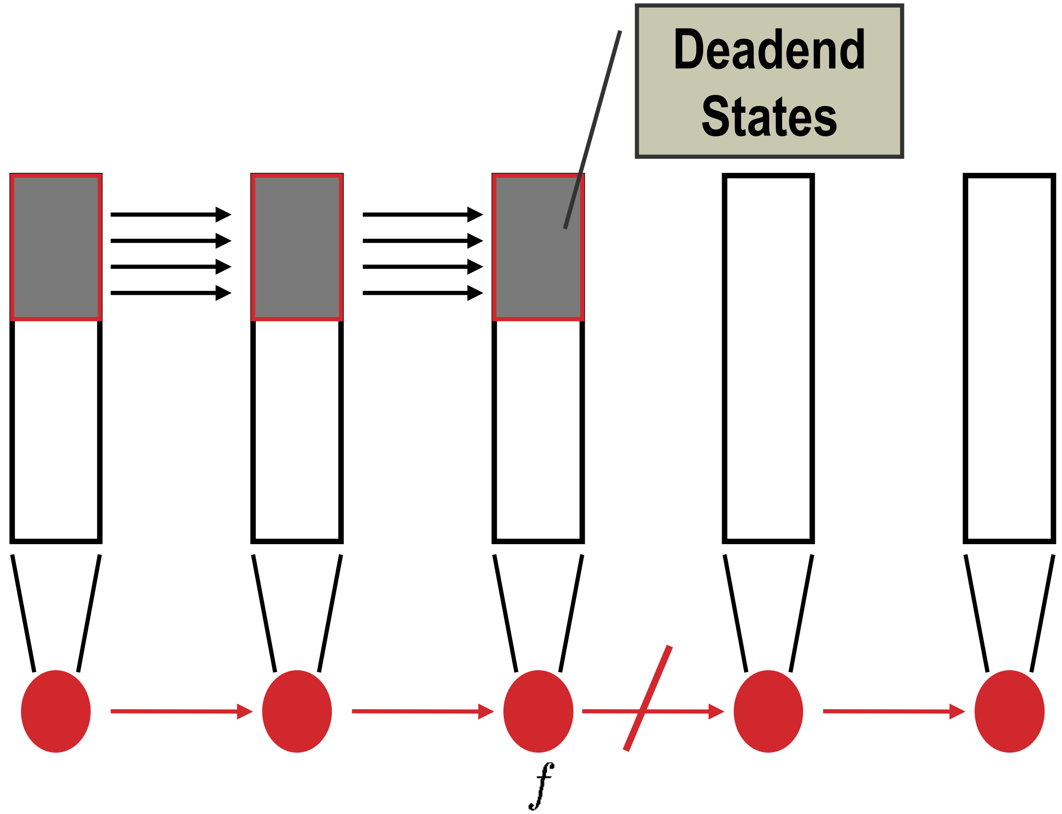


Refinement



Refinement : h’

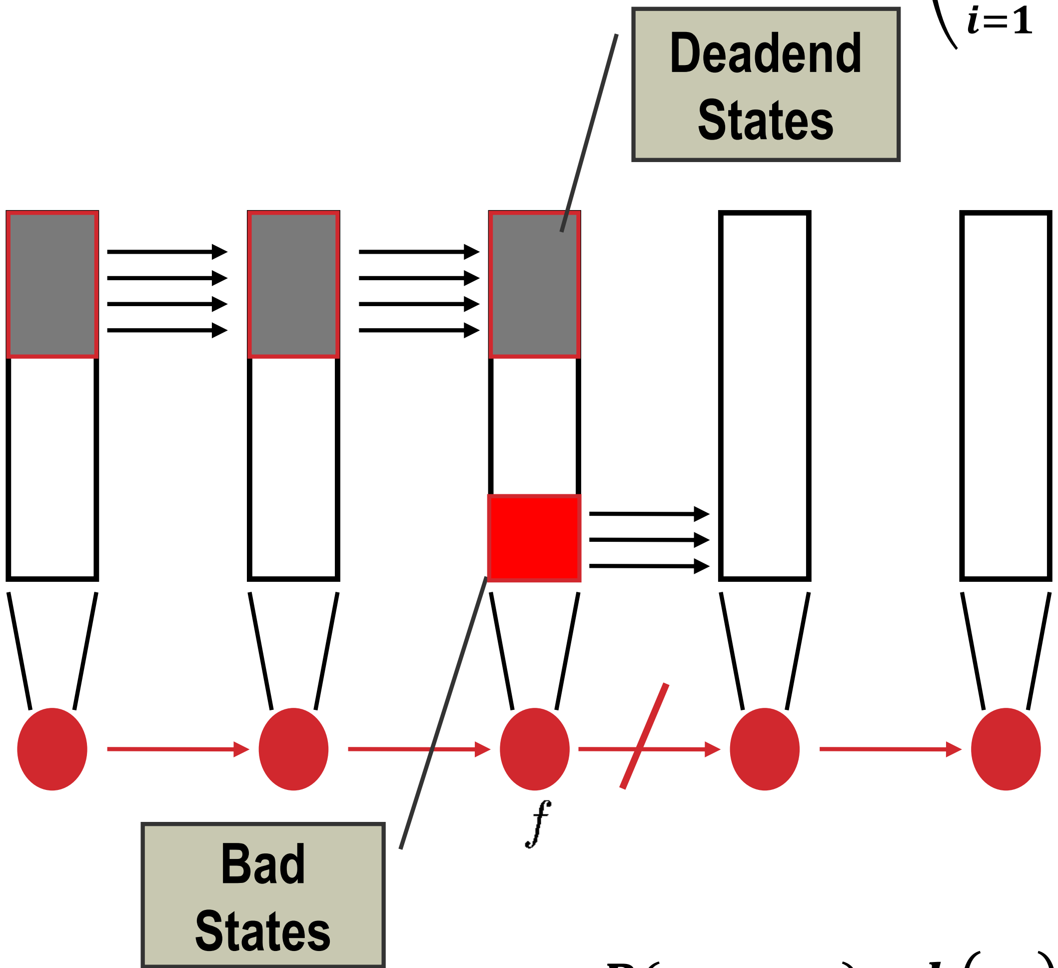
Refinement



$$\varphi_D = \boldsymbol{I}(\boldsymbol{s}_1) \wedge \left(\bigwedge_{i=1}^{f-1} \boldsymbol{R}(\boldsymbol{s}_i, \boldsymbol{s}_{i+1}) \right) \wedge \left(\bigwedge_{i=1}^{f} \boldsymbol{h}(\boldsymbol{s}_i) = \boldsymbol{c}_i \right)$$

Refinement

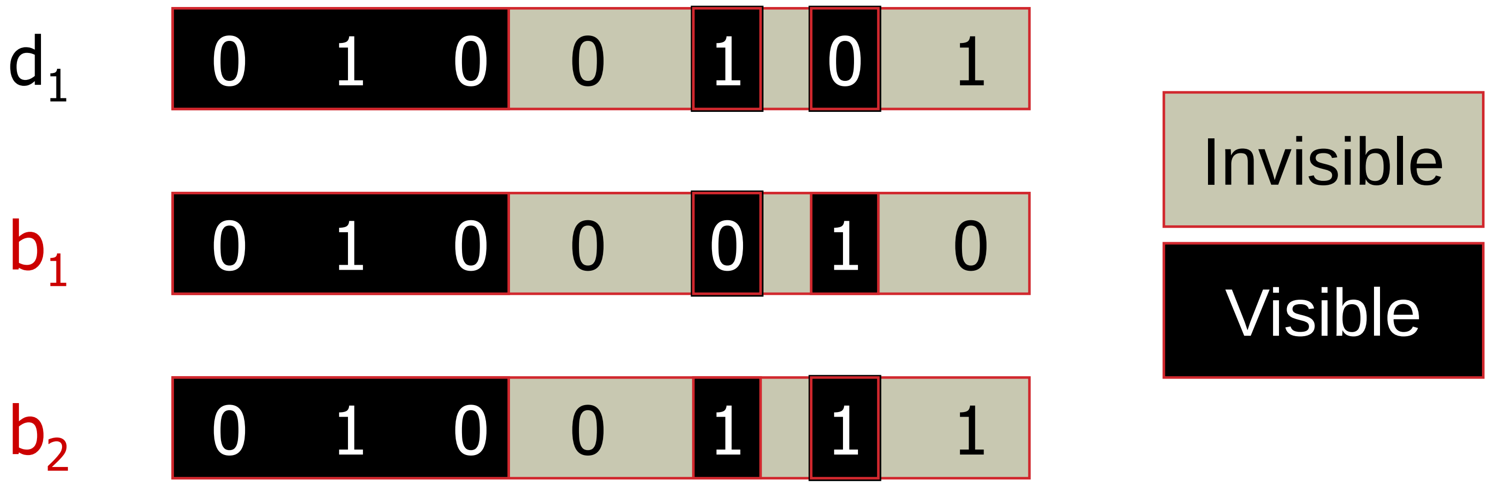
$$\varphi_D = I(s_1) \wedge \left(\bigwedge_{i=1}^{f-1} R(s_i, s_{i+1}) \right) \wedge \left(\bigwedge_{i=1}^{f} h(s_i) = c_i \right)$$



Is $\varphi_D \wedge \varphi_B = \Phi$?

$$\varphi_B = R(s_f, s_{f+1}) \wedge h(s_f) = c_f \wedge h(s_{f+1}) = c_{f+1}$$

Refinement as Separation



<u>Refinement</u>: Find subset U of I that separates between all pairs of dead-end and bad states. Make them visible.

U must be minimal !

Refinement as Separation

d_1	0	1	0	0	1	0	1
b_1	0	1	0	0	0	1	0
b_2	0	1	0	0	1	1	1

I

V

<u>Refinement</u>: Find subset U of I that separates between all pairs of dead-end and bad states. Make them visible.

U must be minimal !

Refinement as Separation

<u>The state separation problem</u>

Input: Sets D, B

Output: Minimal U $\in$ I s.t.:

$$\forall\, d \in D,\ \forall\, b \in B,\ \exists u \in U.\quad d(u) \neq b(u)$$

The refinement h' is obtained by adding U to V.

Separation methods

ILP-based separation

- Minimal separating set.
- Computationally expensive.

Decision Tree Learning based separation.

- Not optimal.
- Polynomial.