Introduction to Software Verification

Orna Grumberg

Lectures Material
winter 2017-18

Lecture 13

16.1.18

Other solutions to the state-explosion problem

Small models replace the full, concrete model:

- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry

Abstraction preserving ACTL/ACTL*

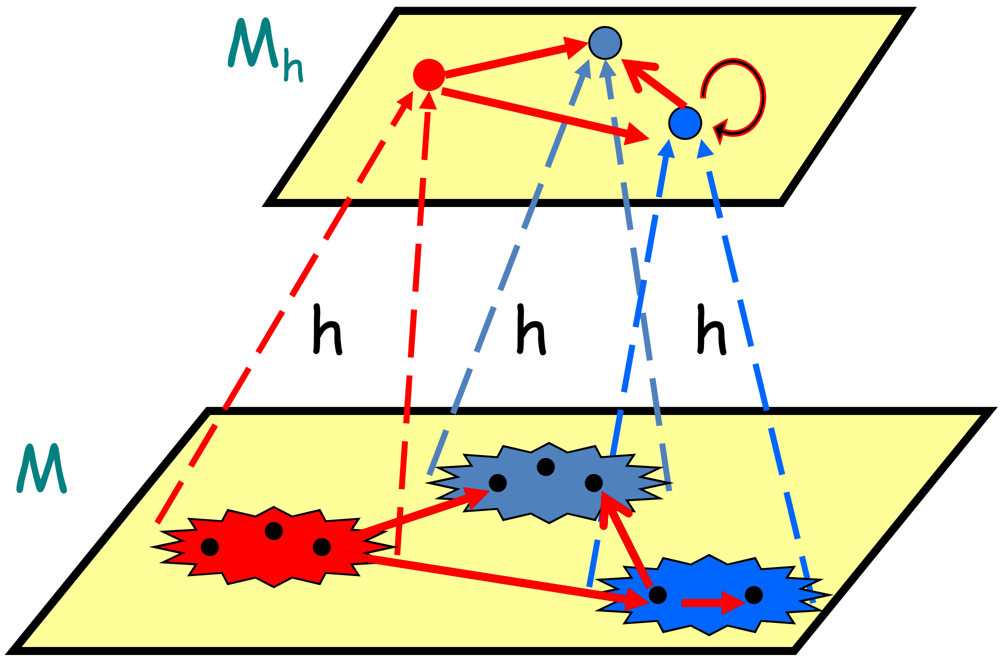
We use **Existential Abstraction** in which the abstract model is an **over-approximation** of the concrete model:

- The abstract model has **more behaviors**
- But no concrete behavior is lost

- Every ACTL/ACTL* property true in the abstract model is also true in the concrete model

Existential Abstraction

Given an abstraction function $h : S \rightarrow S_h$, the concrete states are grouped and mapped into abstract states :



$M \leq M_h$

How to define an abstract model:

Given M and φ, **choose**

- **S_h** - a set of abstract states
- ***AP*** – a set of atomic propositions that label concrete and abstract states
- **h** : $S \rightarrow S_h$ - a mapping from S on S_h that satisfies:

$$h(s) = h(t) \text{ only if } L(s)=L(t)$$

- h is called appropriate w.r.t. *AP*

The abstract model
$M_h = (S_h, I_h, R_h, L_h)$

- $s_h \in I_h \Leftrightarrow \exists \mathbf{s} \in I : h(\mathbf{s}) = s_h$
- $(s_h, t_h) \in R_h \Leftrightarrow$
  $\exists \mathbf{s}, \mathbf{t}\ [\ h(\mathbf{s}) = s_h \wedge h(\mathbf{t}) = t_h \wedge (\mathbf{s}, \mathbf{t}) \in R\ ]$
- $L_h(s_h) = L(s)$ for some s where $h(s) = s_h$

This is an exact abstraction

An approximated abstraction (an approximation)

- $s_h \in I_h \Leftarrow \exists s \in I : h(s) = s_h$
- $(s_h, t_h) \in R_h \Leftarrow$
  $\exists s,t \; [\, h(s) = s_h \wedge h(t) = t_h \wedge (s,t) \in R \,]$
- L_h is as before

Notation:

M_r - reduced (exact) M_h - approximated

Depending on h and the size of M, M_h (I.e. I_h, R_h) can be built using:

- BDDs or
- SAT solver or
- Theorem prover (SMT)

We later demonstrate such constructions for specific types of abstractions

Predicate Abstraction

- Given a program over variables **V**
- **Predicate** P_i is a first-order atomic formula over V

  Examples: $x+y < z^2$, $x=5$

- Choose: ***AP* = { $P_1,\ldots,P_k$ }** that includes
  - the atomic formulas in the property φ and
  - conditions in **if**, **while** statements of the program

Predicate Abstraction - Example

```
while (x≤1) {
        ......
        if (y=2) { .... }
        ......
}
```

$\varphi = AFG(x>y)$

$AP = \{x>y, x\leq 1, y=2\}$

Predicate Abstraction

- Labeling of concrete states:

  $L(s) = \{ P_i \mid s \models P_i \}$

Example (concrete model)

Program over natural variables x, y

$S = N \times N$

$AP = \{ P_1, P_2, P_3 \}$ where

$P_1 = x \leq 1$, $P_2 = x > y$, $P_3 = y = 2$

$AP = \{ x \leq 1 , x > y , y = 2 \}$

$L((0,0)) = L((1,1)) = L(0,1)) = \{ P_1 \}$

$L((0,2)) = L((1,2)) = \{ P_1, P_3 \}$

$L((2,3)) = \emptyset$

Abstract model - Definition

- Abstract states are defined over Boolean variables $\{ B_1,\ldots,B_k \}$:
  $S_h \subseteq \{ 0,1 \}^k$
- $h(s) = s_h \Leftrightarrow$
  for all $1 \le j \le k : [s \models P_j \Leftrightarrow s_h \models B_j]$
- $L_h(s_h) = \{ P_j \mid s_h \models B_j \}$
- Is h appropriate for AP?

Example (concrete model)

Program over natural variables x, y

$S = N \times N$

$AP = \{ P_1, P_2, P_3 \}$ where

$P_1 = x \leq 1$, $P_2 = x > y$, $P_3 = y = 2$

$AP = \{ x \leq 1 , x > y , y = 2 \}$

$L((0,0)) = L((1,1)) = L(0,1)) = \{ P_1 \}$

$L((0,2)) = L((1,2)) = \{ P_1, P_3 \}$

$L((2,3)) = \emptyset$

Example - (abstract model)

$AP=\{P_1=(x\leq 1),P_2=(x>y),P_3=(y=2)\}$

$S_h \subseteq \{0,1\}^3$

$h((0,0)) = h((1,1)) = h(0,1)) = (1,0,0)$

$h((0,2)) = h((1,2)) = (1,0,1)$

No concrete state is mapped to (1,1,1)

$L_h((1,0,0)) = \{ P_1 \}$

$L_h((1,0,1)) = \{ P_1, P_3 \}$

The concrete state and its abstract state are labeled identically

Computing R_h (same example)

$(s_h, t_h) \in R_h \Leftrightarrow$

$$\exists \mathbf{s}, \mathbf{t}\ [\ h(\mathbf{s}) = s_h \wedge h(\mathbf{t}) = t_h \wedge (\mathbf{s},\mathbf{t}) \in R\ ]$$

Computing R_h (same example)

Program with one statement: x := x+1

$$\big(\overbrace{(b_1,b_2,b_3)}^{s_h},\overbrace{(b'_1,b'_2,b'_3)}^{t_h}\big) \in R_h \Leftrightarrow$$

$$\exists \overbrace{xy}^{s}\overbrace{x'y'}^{t}\ [\ \left.\begin{array}{l} P_1(x,y) \Leftrightarrow b_1 \wedge \\ P_2(x,y) \Leftrightarrow b_2 \wedge \\ P_3(x,y) \Leftrightarrow b_3 \wedge \end{array}\right\} h(s)=s_h$$

$$\left.x'=x+1 \wedge y'=y \wedge\right\} R(s,t)$$

$$\left.\begin{array}{l} P_1(x',y') \Leftrightarrow b'_1 \wedge \\ P_2(x',y') \Leftrightarrow b'_2 \wedge \\ P_3(x',y') \Leftrightarrow b'_3 \end{array}\right\} h(t)=t_h\ ]$$

Depending on h and the size of M, M_h (I.e. I_h, R_h) can be built using:

- BDDs, if S is finite and not too big
- SAT solver, if S is finite and possibly big
- Theorem prover (SMT), S might be infinite

Logic preservation Theorem

- **Theorem** If φ is an ACTL/ACTL* specification over AP, then

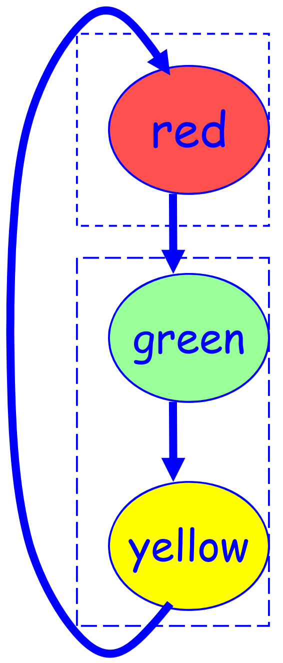
$$M_h \models \varphi \Rightarrow M \models \varphi$$

- However, the reverse may not be valid.

Traffic Light Example

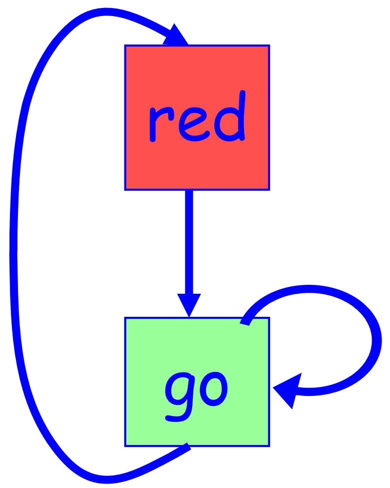
Property:
$\varphi = \mathbf{AG\ AF} \neg (state=red)$

Abstraction function h maps green, yellow to go.



M

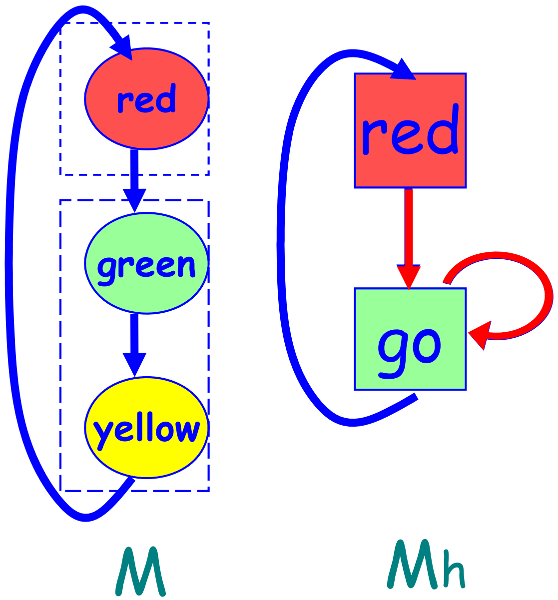
$M \models \varphi \Leftarrow M_h \models \varphi$



M_h

Traffic Light Example (Cont)

If the abstract model invalidates a specification, the actual model may still satisfy the specification.



- Property:

  φ = **AG AF** (state=red)

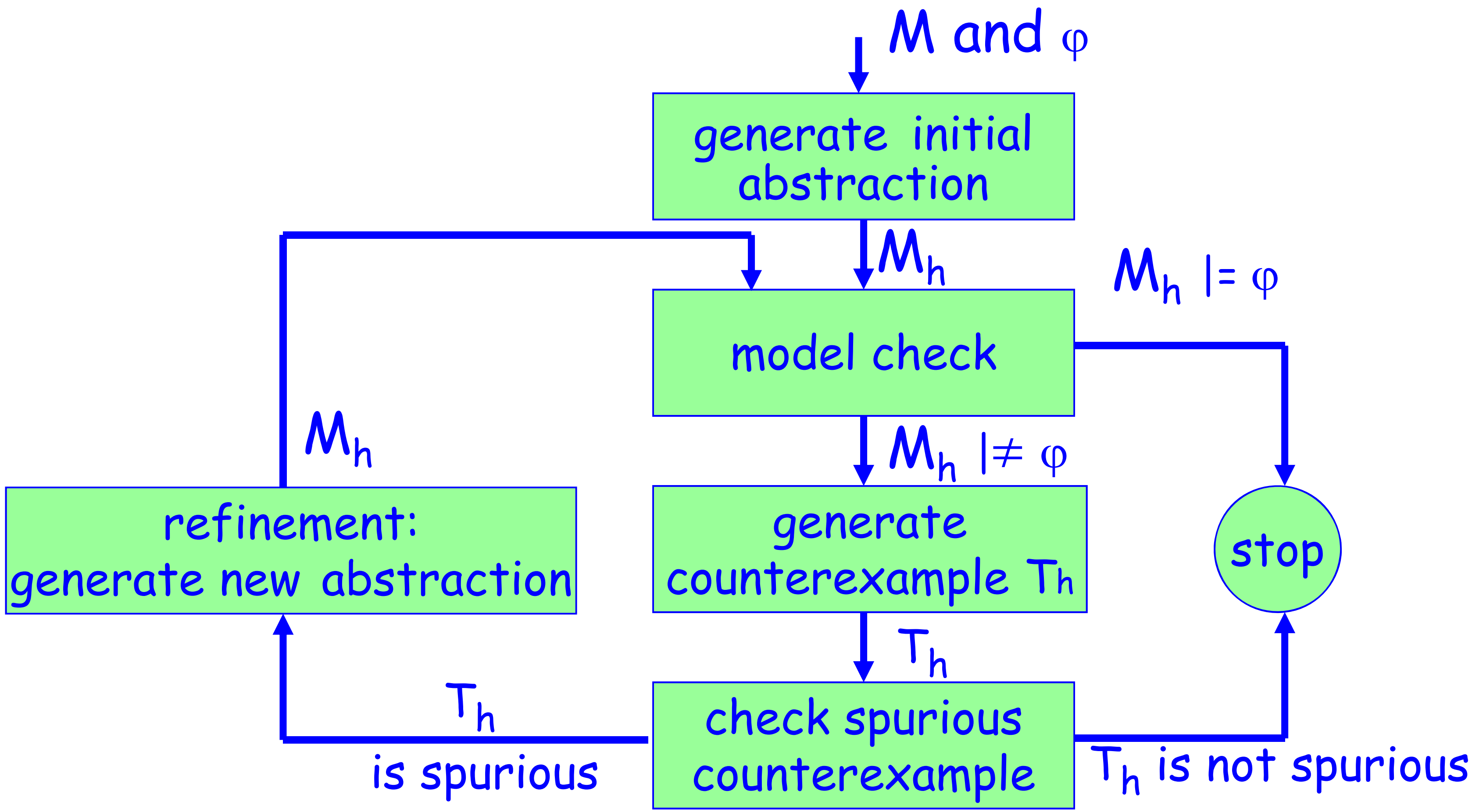
- $M \models \varphi$ but $M_h \not\models \varphi$

- Spurious Counterexample:

  $\langle$red,go,go, ...$\rangle$

CounterExample-Guided Abstraction-Refinement (CEGAR)

The CEGAR Methodology



Generating the Initial Abstraction

- If we use predicate abstraction then predicates are extracted from the program's control flow and the checked property
- If we use localization reduction then the un-abstracted variables are those appearing in the predicates above

Predicate Abstraction - Example

```
while (true) {
        if (reset == 1) { x=y=0; }
        else if (x<y) { x=x+1; }
        else if (x==y && !(y==2)) { y=y+1; }
        else if (x==y) { x=y=0; }
}
```

$\varphi = AF(x==y)$

AP={reset==1, x<y, x==y, y==2}

Model Check The Abstract Model

Given the abstract model M_h

- If $M_h \not\models \varphi$, then the model checker generates a counterexample trace (T_h)
- Most current model checkers generate paths or loops
- Question : is T_h spurious?

Counterexamples

- For AGp it is a path to a state satisfying $\neg p$
- For AFp it is a infinite path represented by a path+loop, where all states satisfy $\neg p$

On the other hand

- For EFp we need to return the whole computation tree (the whole model)

- For $AX(AGp \vee AGq)$ we need to return a computation tree demonstrating $EX(EF\neg p \wedge EF\neg q)$

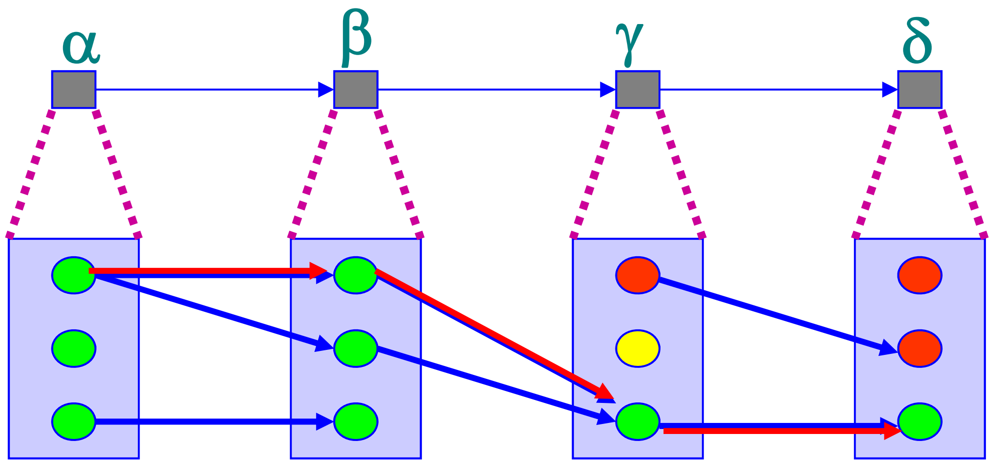
Path Counterexample

Assume that we have four abstract states

$\{1,2,3\} \leftrightarrow \alpha \quad \{4,5,6\} \leftrightarrow \beta$

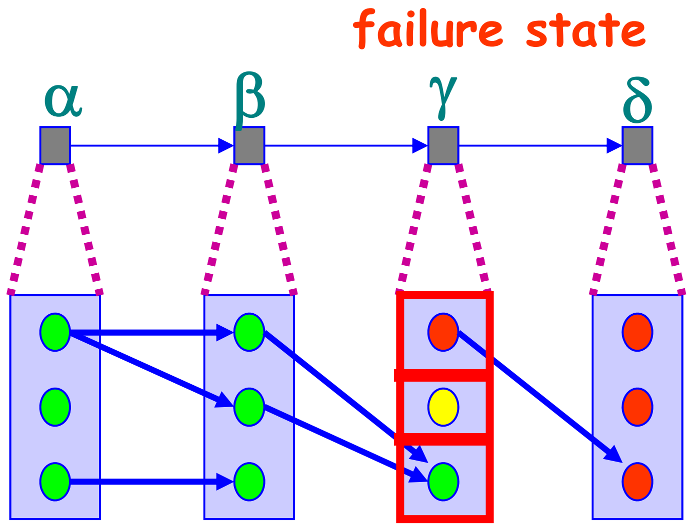
$\{7,8,9\} \leftrightarrow \gamma \quad \{10,11,12\} \leftrightarrow \delta$

Abstract counterexample $T_h = \langle \alpha, \beta, \gamma, \delta \rangle$



T_h is not spurious, therefore, $M \not\models \varphi$

Spurious Path Counterexample



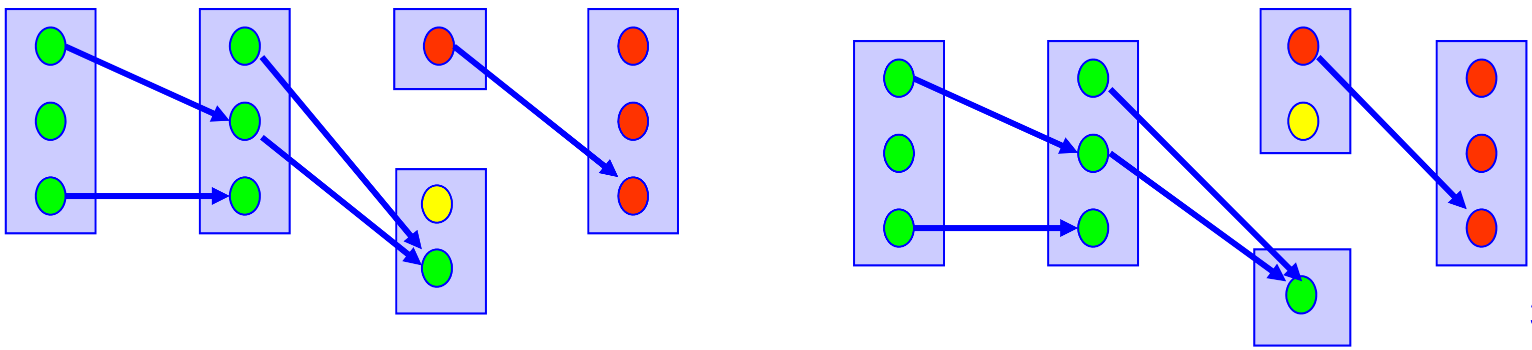
T_h is spurious

The concrete states mapped to the failure state are partitioned into 3 sets

states	dead-end	bad	irrelevant
reachable	yes	no	no
out edges	no	yes	no

Refining The Abstraction

- Goal : refine h so that the dead-end states and bad states do not belong to the same abstract state.
- For this example, two possible solutions.



Refining the abstraction

- Refinement separates dead-end states from bad states, thus, eliminating the spurious transition from S_{i-1} to S_i
- This can be done, for instance, by adding a new predicate to the abstract model and building a new, refined abstract model

Completeness of CEGAR

If M is finite

- Our methodology refines the abstraction until either the property is proved or a real counterexample is found

- Theorem Given a finite model M and an ACTL* specification ϕ whose counterexample is either path or loop, our algorithm will find a model M_a such that

$$M_a \models \phi \Leftrightarrow M \models \phi$$

Conclusion

We presented a framework for Counterexample Guided Abstraction Refinement (CEGAR) that

- Automatically constructs an initial abstraction, based on the checked property and the system
- If the abstract system contains a spurious counterexample then the abstraction is automatically refined in order to eliminate the counterexample