

# CS766: Analysis of concurrent programs 2020

## Lecture 20: Counterexample guided abstraction refinement (CEGAR)

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2020-02-04

# Limitations of symbolic model checking

- ▶ Too precise
- ▶ Often does not scale!
- ▶ Approximations like BMC or concolic testing have severe limitations

Let us bring back abstraction!

## Topic 20.1

Recall: abstract domain and abstract post

## Recall: Abstract domain

### Definition 20.1

*Concrete* objects of analysis or domain —  $C = \mathcal{P}(\mathbb{Q}^V)$

- ▶ *not all sets are concisely representable in computer*
- ▶ *too (infinitely) many of them*

### Definition 20.2

*Abstract* domain — only simple to represent sets  $D \subseteq C$

- ▶ *D should allow efficient algorithms for desired operations*
- ▶ *far fewer, but possibly infinitely many*
- ▶ *Sets in  $C \setminus D$  are **not precisely** representable in D*

### Definition 20.3

An *abstraction function*  $\alpha : C \rightarrow D$  maps each set  $c \in C$  to  $\alpha(c)$ .

### Definition 20.4

A *concretization function*  $\gamma : D \rightarrow C$  maps each set  $d \in D$  to  $d$ .

## Recall: Example: abstraction – intervals

### Example 20.1

Let us assume  $V = \{x\}$

Consider  $D = \{\perp, \top\} \cup \{[a, b] \mid a, b \in \mathbb{Q}\}$ .

Ordering among elements of  $D$  are defined as follows:

$\perp \sqsubseteq [a, b] \sqsubseteq \top$  and  $[a_1, b_1] \sqsubseteq [a_2, b_2] \Leftrightarrow a_2 \leq a_1 \wedge b_1 \leq b_2$

Let  $\alpha(c) \triangleq [\inf(c), \sup(c)]$  and  $\gamma([a, b]) \triangleq [a, b]$

### Exercise 20.1

Give the following value

►  $\alpha(\{0, 3, 5\}) =$

►  $\alpha([0, 3] \cup [5, 6]) =$

►  $\alpha((0, 3)) =$

►  $\alpha(\{1/x \mid x \geq 1\}) =$

# Abstract operations

Let us suppose we have the following abstract domain

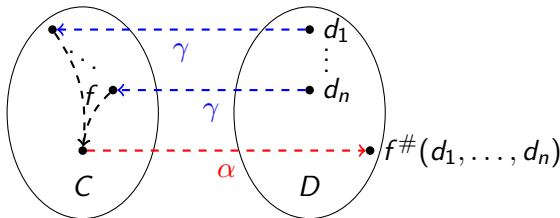
$$(C, \subseteq) \xrightleftharpoons[\alpha]{\gamma} (D, \sqsubseteq).$$

Let us suppose we also have a function  $f : C^n \rightarrow C$  in concrete domain  $C$ .

## Definition 20.5

We define an *abstract operation*  $f^\# : D^n \rightarrow D$  as follows

$$f^\#(d_1, \dots, d_n) = \alpha \circ f(\gamma(d_1), \dots, \gamma(d_n))$$



## Example: abstract operation

We use  $f$ ,  $\alpha$ , and  $\gamma$  to implement  $f^\#$ . For example,

- ▶ We may implement  $\sqcup$  as follows

$$x \sqcup y = \alpha(\gamma(x) \cup \gamma(y))$$

- ▶ We may implement  $\sqcap$  as follows

$$x \sqcap y = \alpha(\gamma(x) \cap \gamma(y))$$

### Example 20.2

*Consider interval domain. Let us compute  $[0, 3] \sqcup [8, 11]$ .*

- ▶  $[0, 3] \sqcup [8, 11] = \alpha(\gamma([0, 3]) \cup \gamma([8, 11])) = \alpha([0, 3] \cup [8, 11]) = [0, 11]$

**Commentary:** The  $\sqcup$  computation may look a simple thing made complex. However, the above captures the idea that the function calculation

## Abstract strongest post

Recall from earlier lecture, we discussed abstract post. Now we have the formal definition.

$$sp^\#(d, \rho) = \alpha \circ sp(\gamma(d), \rho)$$

### Example 20.3 (Reminder)

*Recall the following abstraction function*

$$wideOne(X) = \{n + 1, n \mid n \in X\}$$

*We defined the following abstract post*

$$sp^\#(F, \rho) = \underbrace{wideOne}_{\alpha}(sp(\underbrace{F}_{\gamma \text{ is identity}}, \rho))$$



## Topic 20.2

### Abstract model checking

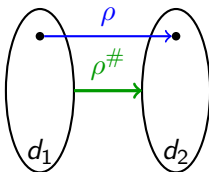
# Abstract program

## Definition 20.6

Let us consider a finite abstraction  $D$  and a program  $P = (V, L, \ell_0, \ell_e, E)$ . An **abstract program**  $P^\# = \text{ABSTRACT}(P, D)$  is  $(V, L, \ell_0, \ell_e, E^\#)$  where  $E^\#$  is defined as follows.

If  $(\ell, \rho, \ell') \in E$  then  $(\ell, \rho^\#, \ell') \in E^\#$ , where

$$\rho^\# = \{\gamma(d) \times \gamma(d') \mid d' = sp^\#(d, \rho)\}.$$



We assume  $D$  and  $P$  allow  $\rho^\#$  to be easily representable in a computer.

# Properties of abstract programs

## Theorem 20.1

$$\forall d \in D \exists d' \in D. sp(\gamma(d), \rho^\#) = \gamma(d')$$

In other words, the reachable states of the abstract programs are representable in  $D$ .

## Theorem 20.2

*If  $P^\#$  is safe then  $P$  is safe.*

Just analyze the abstract program.

## Example : abstract edges

### Example 20.4

Consider the following edge and sign abstraction  $D = \{\top, -, 0, +, \perp\}$ .

$$\rho_1 = (x' = 1)$$

Let us build abstract edge.

►  $sp^\#(+, \rho_1) = +$

►  $sp^\#(0, \rho_1) = +$

►  $sp^\#(-, \rho_1) = +$

►  $sp^\#(\top, \rho_1) = +$

►  $sp^\#(\perp, \rho_1) = \perp$

No need to record pairs  
that start with  $\perp$

$$\rho_1^\# = \{(-, +), (0, +), (+, +), (\top, +)\}$$

### Exercise 20.2

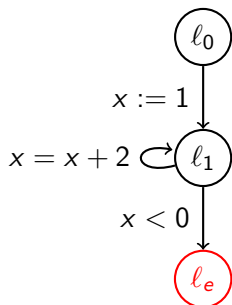
Give abstraction of  $\rho_2 = (x' = x + 1)$

# Example: abstract program

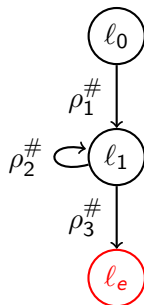
## Example 20.5

Consider the following program and sign abstraction  $D = \{\top, -, 0, +, \perp\}$ .

Program:



Abstract  
program:



$$\rho_1^\# = \{(-, +), (0, +), (+, +), (\top, +)\}$$

$$\rho_2^\# = \{(-, \top), (0, +), (+, +), (\top, \top)\}$$

$$\rho_3^\# = \{(-, -), (\top, -)\}$$

We have only listed pairs that do not have  $\perp$  as second component.

# Abstract reachability graph

Since  $D = (\sqsubseteq, \top, \perp)$  is finite, symbolic execution of  $P^\# = \text{ABSTRACT}(P, D)$  will produce finitely many symbolic states, which are called **abstract states**.

## Definition 20.7

**Abstract reachability graph (ARG)**  $(\text{reach}, R)$  is the smallest directed graph such that

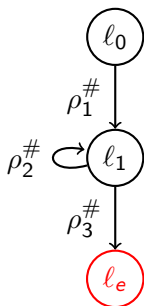
- ▶  $\text{reach} \subseteq L \times D$
- ▶  $(\ell_0, \top) \in \text{reach}$
- ▶  $((\ell, d), (\ell', d')) \in R$  if  $\exists (\ell, \rho^\#, \ell') \in E^\#$ .  $d' = \text{sp}(d, \rho^\#)$

## Theorem 20.3

If  $\forall d. d \neq \text{bot} \wedge (l_e, d) \notin \text{reach}$  then  $P^\#$  is safe.

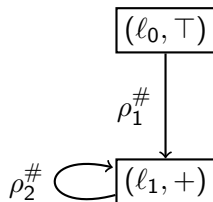
# Example: abstract reachability graph

Abstract program:



$$\begin{aligned}\rho_1^\# &= \{(-, +), (0, +), (+, +), (\top, +)\} \\ \rho_2^\# &= \{(-, +), (-, 0), (-, +), (0, +), (+, +), \\ &\quad (\top, \top)\} \\ \rho_3^\# &= \{(-, -), (\top, -)\}\end{aligned}$$

Abstract reachability graph:



We are not showing abstract states with  $\perp$ .

## Exercise 20.3

Draw the rest of ARG with  $\perp$

# Model checking

The word **model checking** originated from the area of modal logic, where finding a model that satisfies a formula is called model checking.

In our situation, we have a logical statement  $P^\#$  **is not safe**

We search for a model of the statement, i.e., a path in the abstract reachability graph that reaches to error location.

If no model found, then  $P^\#$  is safe.

Abstract reachability graph may be large.

In contrast, abstract interpretation **does not** construct large objects.



# Abstract model checking

**Algorithm 20.1:**  $\text{ABSTMC}(P^\# = (V, L, \ell_0, \ell_e, E^\#), D = (\sqsubseteq, \top, \perp))$

**Output:** CORRECT if  $P^\#$  is safe, abstract counterexample otherwise

$worklist := \{(\ell_0, \top)\}$ ;  $reach := \emptyset$ ;  $covered := \emptyset$ ;

$parent : reach \cup worklist \rightarrow reach \cup worklist := \{((\ell_0, \top), (\ell_0, \top))\}$ ;

$path : reach \cup worklist \rightarrow (\text{sequences of } E^\#) := \{((\ell_0, \top), \epsilon)\}$ ;

**while**  $worklist \neq \emptyset$  **do**

    choose  $(\ell, d) \in worklist$ ;  $worklist := worklist \setminus \{(\ell, d)\}$ ;

**if**  $d = \perp$  or  $\exists s \in parent^*((\ell, d)). s \in covered$  **then continue**;

**if**  $\ell = \ell_e$  **then return** COUNTEREXAMPLE( $path(\ell, d)$ ) ;

$reach := reach \cup \{(\ell, d)\}$ ;

**if**  $\exists (\ell, d') \in reach - range(covered). d \sqsubseteq d'$  **then**

$covered := covered \cup \{((\ell, d'), (\ell, d))\}$

**else**

**if**  $\exists (\ell, d') \in reach - range(covered). d' \sqsubseteq d$  **then**

$covered := covered \cup \{((\ell, d), (\ell, d'))\}$

$P^\#$  accessed  
only once

**foreach**  $(\ell, \rho^\#, \ell') \in E^\#$  **do**

$d' := sp(d, \rho^\#)$ ;  $worklist := worklist \cup \{(\ell', d')\}$ ;

$parent((\ell', d')) = (\ell, d)$ ;  $path((\ell', d')) = path((\ell, d)).(\ell, \rho^\#, \ell')$ ;

**return** CORRECT

# On the fly abstraction

In ABSTMC, we only access  $P^\#$  to compute post operator over  $d$ .

This suggests, ABSTMC can be implemented in the following two ways.

- ▶ **Precompute**  $P^\#$  and run ABSTMC as presented.
- ▶ **On the fly construction** of  $P^\#$ . We construct transitions of  $P^\#$  as we need them

## Exercise 20.4

*Discuss benefits of both the approaches*

# Finite abstractions

The following abstractions are widely used in modelcheckers

- ▶ Cartesian predicate abstraction
- ▶ Boolean predicate abstraction

## Finite abstraction example : Cartesian predicate abstraction

Cartesian predicate abstraction is defined by a set of predicates

$$Preds = \{p_1, \dots, p_n\}$$

$$C = \mathbf{p}(\mathbb{Q}^{|V|})$$

$$D = \perp \cup \mathbf{p}(Preds)$$

$$\perp \sqsubseteq S_1 \sqsubseteq S_2 \text{ if } S_2 \subseteq S_1$$

$$\alpha(c) = \{p \in P \mid c \Rightarrow p\}$$

$$\gamma(S) = \bigwedge S$$

//  $\emptyset$  represents  $\top$

### Example 20.6

$$V = \{x, y\}$$

$$P = \{x \leq 1, -x - y \leq -1, y \leq 5\}$$

$$\alpha(\{(0, 0)\}) = \{x \leq 1, y \leq 5\}$$

$$\alpha((x-1)^2 + (y-3)^2 = 1) = \{-x - y \leq -1, y \leq 5\}$$

# Representing predicate domain

We represent abstract state as bit vectors.

## Example 20.7

Consider  $V = \{x, y\}$  and  $P = \{x \leq 1, -x - y \leq -1, y \leq 5\}$

Let  $[101]$  represent  $x \leq 1 \wedge y \leq 5$

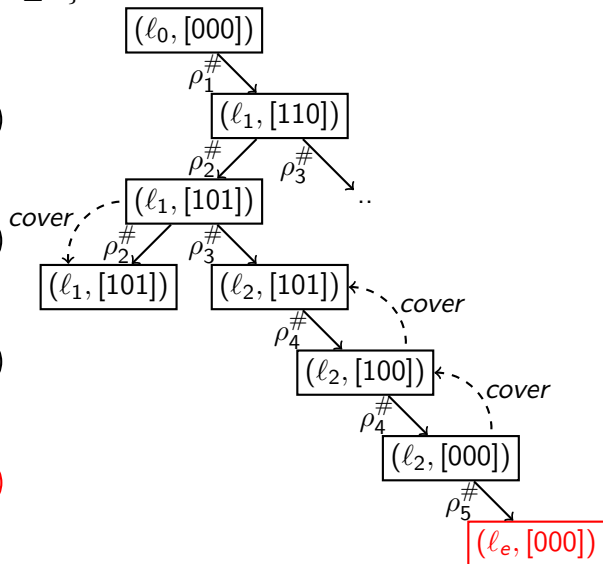
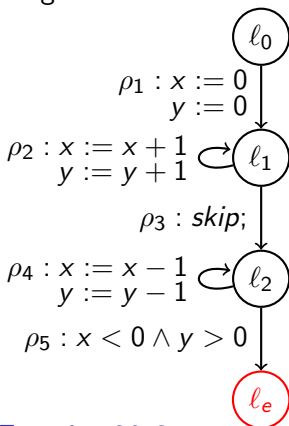
## Exercise 20.5

- ▶  $[100]$  represents ...
- ▶  $[000]$  represent ...
- ▶ Is  $[100] \sqsubseteq [000]$ ?
- ▶ Is  $[100] \sqsubseteq [001]$ ?
- ▶ Is  $[101] \sqsubseteq [001]$ ?
- ▶ Can we represent false in predicate domain without using special symbol  $\perp$ ?

# Example: ARG with Cartesian predicate abstraction

$Preds = \{x \geq 0, y \leq 0, x \geq 1\}$ .

Program:



Exercise 20.6

Complete the ARG

## Spurious counterexample

$\text{ABSTMC}(P^\#, D)$  may fail to prove  $P^\#$  correct and return a path  $e_1^\# \dots e_m^\#$ , which is called **abstract counterexample**.

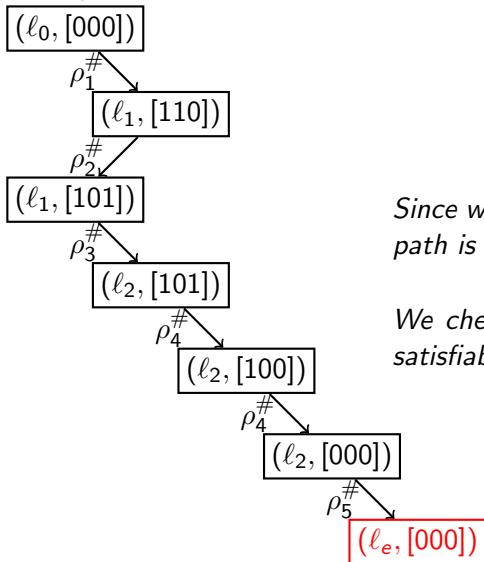
Let  $e_1 \dots e_m$  be the corresponding path in  $P$ . Now we have two possibilities.

- ▶  $e_1 \dots e_m$  is feasible. Then, we have found a bug
- ▶  $e_1 \dots e_m$  is not feasible. Then, we call  $e_1 \dots e_m$  as **spurious counterexample**.

We need to fix our abstraction such that we do not get the spurious counterexample.

# Example : spurious counterexample

## Example 20.8



Since we cannot execute  $\rho_1\rho_2\rho_3\rho_4\rho_4\rho_5$ , the path is a *spurious counterexample*.

We check the feasibility of the path using satisfiability of path constraints.



# Refinement relation

## Definition 20.8

*Consider abstractions*

$$(C, \subseteq) \xleftrightarrow[\alpha_1]{\gamma_1} (D_1, \sqsubseteq_1) \quad \text{and} \quad (C, \subseteq) \xleftrightarrow[\alpha_1]{\gamma_2} (D_2, \sqsubseteq_2).$$

$D_2$  *refines*  $D_1$  if

$$\forall c \in C. \gamma_1(\alpha_1(c)) \subseteq \gamma_2(\alpha_2(c))$$

## Exercise 20.7

$\gamma_1 \circ \alpha_2$  is order embedding.

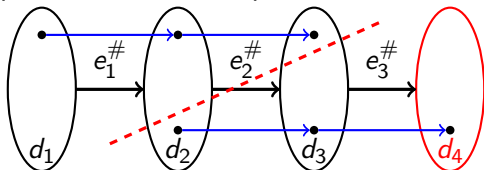
# Abstraction refinement

## Theorem 20.4

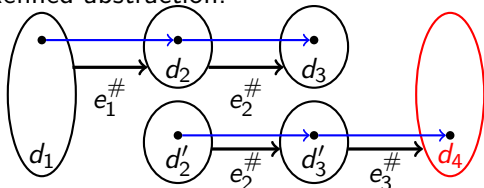
If  $\text{ABSTRACT}(P, D_1)$  exhibits a spurious counterexample then there is an abstraction  $D_2$  such that  $D_2$  refines  $D_1$  and  $\text{ABSTRACT}(P, D_2)$  does not exhibit the same counter example.

## Proof sketch.

Spurious counterexample:



Refined abstraction:



We say the refinement to  $D_2$  from  $D_1$  **ensures progress**, i.e., counterexamples are not repeated if ARG is build again with  $D_2$

# Refinement Strategy for predicate abstraction

## **General refinement strategy**

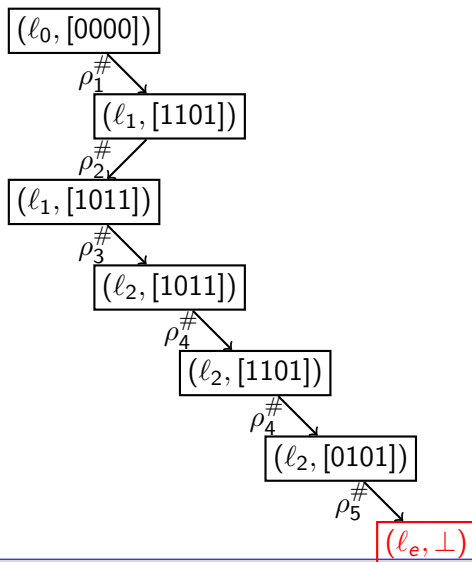
Split abstract states such that the spurious counterexample is disconnected.

In predicate abstraction, we only need to add more predicates. The new abstraction will certainly be refinement.

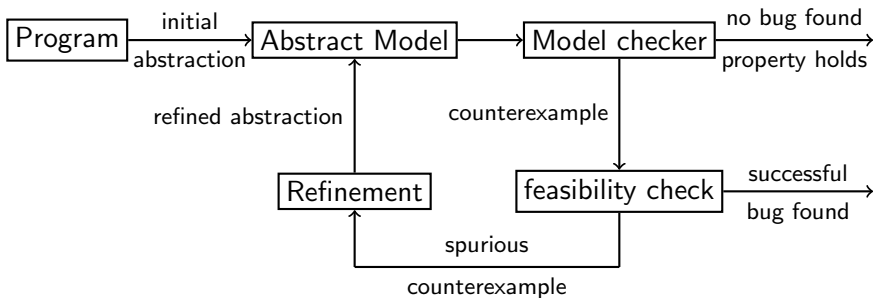
## Example: refinement

Adding predicate  $y \leq -1$  will remove the spurious counterexample.

$Preds = \{x \geq 0, y \leq 0, x \geq 1, y \leq 1\}$



# CEGAR: CounterExample Guided Abstraction Refinement



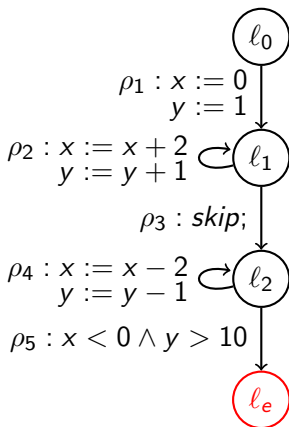
## Topic 20.3

### Problems

# Abstract reachability graph

## Exercise 20.8

Choose a set of predicates that will prove the following program correct and show the ARG of the program using the predicates.



# CPAchecker

## Exercise 20.9

Download CPAchecker: <https://cpachecker.sosy-lab.org/>  
Apply the tool on the following example and report the generated ARG.

```
int x=0; y=0; z=0; w=0;
while( * )) {
    if( * ) {
        x = x+1;
        y = y+100;
    }else if ( * ) {
        if (x >= 4) {
            x = x+1;
            y = y+1;
        }
    }else if (y > 10*w && z >= 100*x) {
        y = -y;
    }
    w = w+1;
    z = z+10;
}
if (x >= 4 && y <= 2)
    error();
```



# LTL to Büchi

## Exercise 20.10

*Convert the following LTL formula into a Büchi automaton*

$$\Box \Diamond a \wedge \Diamond \Box b$$

End of Lecture 20