Program verification 2019

Lecture 5: Abstract Model checking

Instructor: Ashutosh Gupta

IITB, India

Compile date: 2019-01-15
Where are we?

- SMT solving
- Lattice theory
- Abstract interpretation
Lecture plan

- Model checking with finite abstraction
- Predicate abstraction
- Abstract reachability graph
- Model checking vs. Abstract interpretation
- Abstract counterexample
- Abstraction refinement
- Counter example guided abstraction refinement
Topic 5.1

Model checking
Abstract program

Definition 5.1
Let us consider a finite abstraction $D$ and a program $P = (V, L, l_0, l_e, E)$. An abstract program $P\# = \text{Abstract}(P, D)$ is $(V, L, l_0, l_e, E\#)$ where $E\#$ is defined as follows. If $(\ell, \rho, \ell') \in E$ then $(\ell, \rho\#, \ell') \in E\#$, where

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

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

Theorem 5.1
$$\forall d \in D \exists d' \in D. \text{sp}(\gamma(d), \rho\#) = \gamma(d')$$

Theorem 5.2
If $P\#$ is safe then $P$ is safe.
Example: abstract program

Example 5.1

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

Program:

```
x := 1
x = x + 2
x < 0
```

Abstract program:

```
\rho_1^\# = \{(-, +), (0, +), (+, +), (\top, +)\}
\rho_2^\# = \{(-, +), (-, 0), (-, +), (0, +), (+, +)
\quad (\top, \top)\}
\rho_3^\# = \{(-, -), (\top, -)\}
```

We have only listed pairs that do not have \(\bot\) as second component.
Abstract reachability graph

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

Definition 5.2

Abstract reachability graph (ARG) $(reach, R)$ is the smallest directed graph such that

- $reach \subseteq L \times D$
- $(\ell_0, \top) \in reach$
- If $((\ell, d), (\ell', d')) \in R$ iff $\exists (\ell, \rho^\#, \ell') \in E^\#$. $d' = sp(d, \rho^\#)$

Theorem 5.3

If $\forall d. d \neq \text{bot} \land (l_e, d) \not\in reach$ then $P^\#$ is safe.
Example: abstract reachability graph

Abstract program:

Abstract reachability graph:

\[ \ell_0 \]
\[ \ell_1 \]
\[ \ell_e \]
\[ \rho_1^\# = \{(-,+),(0,+),(+,+), (\top,+)\} \]
\[ \rho_2^\# = \{(−,+), (−,0), (−,+), (0,+), (+,+), (\top,\top)\} \]
\[ \rho_3^\# = \{(−,−), (\top,\top)\} \]

We are not showing abstract states with \( \bot \).
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 5.1: AbstractMC**($P^# = (V, L, \ell_0, \ell_e, E^#), D = (\subseteq, \top, \bot)$)

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

\[
\text{worklist} := \{((\ell_0, \top))\}; \text{reach} := \emptyset; \text{covered} := \emptyset; \\
\text{parent} : \text{reach} \cup \text{worklist} \rightarrow \text{reach} \cup \text{worklist} := \{(((\ell_0, \top), (\ell_0, \top))\}; \\
\text{path} : \text{reach} \cup \text{worklist} \rightarrow (\text{sequences of } E^#) := \{(((\ell_0, \top), (\ell_0, \top))\}; \\
\text{while } \text{worklist} \neq \emptyset \text{ do} \\
\text{choose } (\ell, d) \in \text{worklist}; \text{worklist} := \text{worklist} \setminus \{(\ell, d)\}; \\
\text{if } d = \bot \text{ or } \exists s \in \text{parent}^*(((\ell, d)). (\_, s) \in \text{covered} \text{ then continue}; \\
\text{if } \ell = \ell_e \text{ then return } \text{Counterexample}(\text{path}(\ell, d)) ; \\
\text{reach} := \text{reach} \cup \{(\ell, d)\}; \\
\text{if } \exists(\ell, d') \in \text{reach} \setminus \text{range(covered)} \text{. } d \subseteq d' \text{ then} \\
\text{covered} := \text{covered} \cup \{((\ell, d'), (\ell, d))\} \\
\text{else} \\
\text{if } \exists(\ell, d') \in \text{reach} \setminus \text{range(covered)} \text{. } d' \subseteq d \text{ then} \\
\text{covered} := \text{covered} \cup \{((\ell, d), (\ell, d'))\} \\
\text{foreach } (\ell, \rho^#, \ell') \in E^# \text{ do} \\
\text{d'} := \text{sp}(d, \rho^#); \text{worklist} := \text{worklist} \cup \{((\ell', d'))\}; \\
\text{parent}((\ell', d')) = (\ell, d); \text{path}((\ell', d')) = \text{path}((\ell, d)).(\ell, \rho^#, \ell'); \\
\text{return Correct}
\]
On the fly abstraction

In AbstractMC, we only access $P^\#$ to compute post operator over $d$. This suggests, AbstractMC can be implemented in the following two ways.

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

Exercise 5.1

*Discuss benefits of both the approaches*
Finite abstraction

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

\( Prefs = \{p_1, \ldots, p_n\} \)

\( C = p(\mathcal{Q}^{|V|}) \)

\( D = \bot \cup p(Prefs) \)

\( \bot \subseteq S_1 \subseteq S_2 \) if \( S_2 \subseteq S_1 \)

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

\( \gamma(S) = \wedge S \)

Example 5.2

\( V = \{x, y\} \)

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

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

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

We represent abstract state as bit vectors, e.g., [101] represents \( x \leq 1 \land y \leq 5 \)
Example: ARG with Cartesian predicate abstraction

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

Program:

\begin{align*}
\ell_0 & \quad \rho_1: x := 0 \\
& \quad \quad y := 0 \\
\rho_2 & \quad x := x + 1 \\
& \quad \quad y := y + 1 \\
\rho_3 & \quad \text{skip;} \\
\rho_4 & \quad x := x - 1 \\
& \quad \quad y := y - 1 \\
\rho_5 & \quad x < 0 \land y > 0 \\
\ell_e &
\end{align*}

Exercise 5.2

Complete the ARG
Spurious counterexample

AbstractMC($P^\#$, $D$) may fail to prove $P^\#$ correct and return a path $e_1^\# \ldots e_m^\#$, which is called abstract counterexample.

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

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

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

Example 5.3

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 5.3
Consider abstractions

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

\(D_2\) refines \(D_1\) if

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

Exercise 5.3
\(\gamma_1 \circ \alpha_2\) is order embedding.
Abstraction refinement

**Theorem 5.4**
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 built 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

Program \rightarrow \text{Abstract Model} \rightarrow \text{Model checker} \rightarrow \text{feasibility check}

- Initial abstraction
- Refined abstraction
- Refinement
- Spurious counterexample
- Counterexample
- No bug found
- Property holds
- Successful bug found
Computing refinement

In order to automate CEGAR, we need an effective method for computing new predicates that result in the desired refinement.

Here, we discuss the following two methods for refinement of a predicate abstraction.

▶ Syntax based refinement
▶ Interpolation based refinement

Let us first define(remind) some notations.
Path constraints for spurious counterexample (reminder)

$V_i$ be the vector of variables obtained by adding subscript $i$ after each variable in $V$.

**Definition 5.4**

*For a spurious counterexample $e_1 \ldots e_n$, path constraints* $\text{pathCons}(e_1 \ldots e_n)$ *is*

$$\bigwedge_{i \in 1 \ldots n} e_i(V_{i-1}, V_i)$$

A path is **feasible** if corresponding path constraints is satisfiable.

**Note:** Path constraints are also known as “SSA formulas”.
Syntax based refinement

\[
\text{core} = \text{unsatCore}(\ pathCons(e_1 \ldots e_n) ).
\]

\[
\text{preds} = \text{atoms of core after erasing subscripts in its variables}
\]

Add \textit{preds} in the predicate domain to obtain the refined abstract domain
Interpolation

Definition 5.5

Let $A$ and $B$ be formulas such that $A \land B$ is unsat. An interpolant $I$ between $A$ and $B$ is a formula such that

- $A \Rightarrow I$
- $B \land I \Rightarrow \bot$
- $\text{vars}(I) \subseteq \text{vars}(A) \cap \text{vars}(B)$

Theorem 5.5 (Craig interpolation theorem)

Interpolant always exists.

Example 5.4

Consider:

$A = x_1 + x_2 \leq 2 \land x_3 - x_2 \leq 0$

$B = 6x_4 - 2x_1 \leq -8 \land -3x_4 - x_3 \leq 0$

$\text{vars}(A) = \{x_1, x_2, x_3\}$ \quad $\text{vars}(B) = \{x_1, x_3, x_4\}$ \quad $\text{vars}(I) \subseteq \{x_1, x_3\}$

$I = x_1 + x_3 \leq 2$
Interpolation chain

We can extend the definition of interpolant to our setting

Definition 5.6

Consider unsat formula $\bigwedge_{i \in 1..m} e_i(V_{i-1}, V_i)$. An interpolant chain is a sequence of formulas such that $I_0 \ldots I_m$ such that

- $I_0 = \top$
- $\forall i \in 1..m \ I_{i-1} \land e_i(V_{i-1}, V_i) \Rightarrow I_i$
- $I_m = \bot$
- $\text{vars}(I_i) \subseteq V_i$
Interpolation for refinement

We compute interpolation chain $I_0 \ldots I_m$ for \texttt{pathCons(cons)}

$preds = \text{atoms in } I_0 \ldots I_m \text{ after erasing subscripts in its variables}$

Add $preds$ in the predicate domain to obtain the refined abstract domain

\textbf{Theorem 5.6}
\textit{The new abstract domain eliminates spurious counterexample $e_1 \ldots e_n$}
Example: interpolation for refinement

Program:

ρ₁: x := 0, y := 0
ρ₂: x := x + 1, y := y + 1
ρ₃: skip;
ρ₄: x := x - 1, y := y - 1
ρ₅: x < 0 ∧ y > 0

Spurious counterexample: ρ₁ρ₂ρ₃ρ₄ρ₅₁.

⊤
ρ₁(x₀, y₀, x₁, y₁) = (x₁ = 0 ∧ y₁ = 0)
I₁ = y₁ ≤ 0
ρ₂(x₁, y₁, x₂, y₂) = (x₂ = x₁ + 1 ∧ y₂ = y₁ + 1)
I₂ = y₂ ≤ 1 ← New predicate
ρ₃(x₂, y₂, x₃, y₃) = (x₃ = x₂ ∧ y₃ = y₂)
I₃ = y₃ ≤ 0
ρ₄(x₃, y₃, x₄, y₄) = (x₄ = x₃ - 1 ∧ y₄ = y₃ - 1)
I₄ = y₄ ≤ 0
ρ₄(x₄, y₄, x₅, y₅) = (x₅ = x₄ - 1 ∧ y₅ = y₄ - 1)
I₅ = y₅ ≤ 0
ρ₅(x₅, y₅, x₆, y₆) = (x₅ < 0 ∧ y₅ > 0)
⊥
Example: refined reachability graph

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

Program:

\[
\begin{align*}
\rho_1 &: \ x := 0 \\
& \quad \quad \quad \ y := 0 \\
\rho_2 &: \ x := x + 1 \\
& \quad \quad \quad \ y := y + 1 \\
\rho_3 &: \ skip; \\
\rho_4 &: \ x := x - 1 \\
& \quad \quad \quad \ y := y - 1 \\
\rho_5 &: \ x < 0 \land y > 0
\end{align*}
\]

Exercise 5.4

Complete the ARG
Example: good refinement

Consider the earlier spurious counterexample again: $\rho_1\rho_2\rho_3\rho_4\rho_5\rho_5$.

$\top$

$\rho_1(x_0, y_0, x_1, y_1) = (x_1 = 0 \land y_1 = 0)$

$\mathcal{I}_1 = y_1 \leq x_1$

$\rho_2(x_1, y_1, x_2, y_2) = (x_2 = x_1 + 1 \land y_2 = y_1 + 1)$

$\mathcal{I}_2 = y_2 \leq x_2 \quad \leftarrow \text{New predicate}$

$\rho_3(x_2, y_2, x_3, y_3) = (x_3 = x_2 \land y_3 = y_2)$

$\mathcal{I}_3 = y_3 \leq x_3$

$\rho_4(x_3, y_3, x_4, y_4) = (x_4 = x_3 - 1 \land y_4 = y_3 - 1)$

$\mathcal{I}_4 = y_4 \leq x_4$

$\rho_5(x_4, y_4, x_5, y_5) = (x_5 = x_4 - 1 \land y_5 = y_4 - 1)$

$\mathcal{I}_5 = y_5 \leq x_5$

$\rho_5(x_5, y_5, x_6, y_6) = (x_5 < 0 \land y_5 > 0)$

$\bot$
Example: ARG without spurious counterexample

\[ \text{Preds} = \{ x \geq 0, y \leq 0, x \geq 1, y \leq x \} \]

Program:

\[
\begin{align*}
\rho_1 &: x := 0 \\
y &: y := 0 \\
\rho_2 &: x := x + 1 \\
y &: y := y + 1 \\
\rho_3 &: \text{skip}; \\
\rho_4 &: x := x - 1 \\
y &: y := y - 1 \\
\rho_5 &: x < 0 \land y > 0
\end{align*}
\]

Exercise 5.5

Complete the ARG
Observation on CEGAR

- Bad predicates waste our time
- We may reuse the ARG computed in last iteration
- Potential exponential blowup
Topic 5.2

Problems
Abstract reachability graph

Exercise 5.6

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

\[ \rho_1 : x := 0 \]
\[ y := 1 \]

\[ \rho_2 : x := x + 2 \]
\[ y := y + 1 \]

\( \rho_3 : \text{skip} \)

\[ \rho_4 : x := x - 2 \]
\[ y := y - 1 \]

\[ \rho_5 : x < 0 \land y > 10 \]
CPAchecker

Exercise 5.7

*Download CPAchecker: https://cpachecker.sosy-lab.org/*

Apply the tool on the following example and report the generated ARG.

```c
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 Bübhi

Exercise 5.8

Convert the following LTL formula into a Büchi automaton

\( \square \Diamond a \land \Diamond \square b \)
End of Lecture 5