Static Program Analysis

Lecture 17: Abstract Interpretation VII (Final Remarks on CEGAR)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://moves.rwth-aachen.de/teaching/ws-1415/spa/

Winter Semester 2014/15

Outline

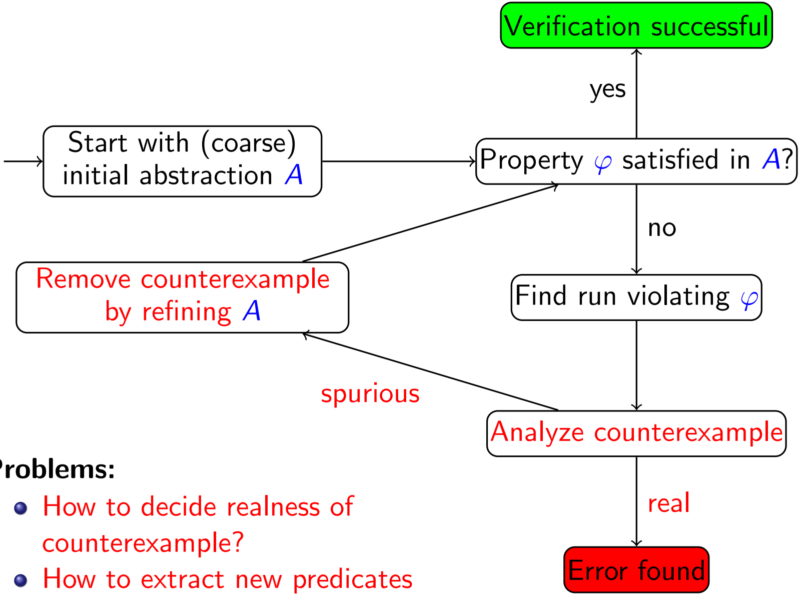
1 Recap: Counterexample-Guided Abstraction Refinement

2 Where CEGAR Fails

3 Craig Interpolation

4 CEGAR Tools

Reminder: CEGAR



Problems:

- How to decide realness of counterexample?
- How to extract new predicates from spurious counterexample?

Abstract Semantics for Predicate Abstraction I

Definition (Execution relation for predicate abstraction)

If $c \in Cmd$ and $Q \in Abs(p_1, \ldots, p_n)$, then $\langle c, Q \rangle$ is called an abstract configuration. The execution relation for predicate abstraction is defined by the following rules:

$$\text{(skip)} \frac{}{\langle \texttt{skip}, Q \rangle \Rightarrow \langle \downarrow, Q \rangle} \quad \text{(asgn)} \frac{}{\langle x \texttt{ := } a, Q \rangle \Rightarrow \langle \downarrow, \bigsqcup \{ Q_{\sigma[x \mapsto val_\sigma(a)]} \mid \sigma \models Q \} \rangle}$$

$$\text{(seq1)} \frac{\langle c_1, Q \rangle \Rightarrow \langle c_1', Q' \rangle \quad c_1' \neq \downarrow}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_1' ; c_2, Q' \rangle} \qquad \text{(seq2)} \frac{\langle c_1, Q \rangle \Rightarrow \langle \downarrow, Q' \rangle}{\langle c_1 ; c_2, Q \rangle \Rightarrow \langle c_2, Q' \rangle}$$

$$\text{(if1)} \frac{}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, Q \rangle \Rightarrow \langle c_1, \overline{Q \wedge b} \rangle}$$

$$\text{(if2)} \frac{}{\langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, Q \rangle \Rightarrow \langle c_2, \overline{Q \wedge \neg b} \rangle}$$

$$\text{(wh1)} \frac{}{\langle \texttt{while } b \texttt{ do } c, Q \rangle \Rightarrow \langle c ; \texttt{while } b \texttt{ do } c, \overline{Q \wedge b} \rangle}$$

$$\text{(wh2)} \frac{}{\langle \texttt{while } b \texttt{ do } c, Q \rangle \Rightarrow \langle \downarrow, \overline{Q \wedge \neg b} \rangle}$$

Counterexamples

Typical properties of interest:

- a certain program location is not reachable (dead code)
- division by zero is excluded
- the value of `x` never becomes negative
- after program termination, the value of `y` is even

Definition (Counterexample)

- A counterexample is a sequence of abstract transitions of the form

$$\langle c_0, \text{true}\rangle \Rightarrow \langle c_1, Q_1\rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k\rangle$$

  where

  - $k \geq 1$
  - $c_0, \ldots, c_k \in Cmd$ (or $c_k = \downarrow$)
  - $Q_1, \ldots, Q_k \in Abs(p_1, \ldots, p_n)$ with $Q_k \not\equiv \text{false}$

- It is called real if there exist concrete states $\sigma_0, \ldots, \sigma_k \in \Sigma$ such that

$$\forall i \in \{1, \ldots, k\} : \sigma_i \models Q_i \text{ and } \langle c_{i-1}, \sigma_{i-1}\rangle \rightarrow \langle c_i, \sigma_i\rangle$$

- Otherwise it is called spurious.

Elimination of Spurious Counterexamples

Lemma

If $\langle c_0, \mathsf{true}\rangle \Rightarrow \langle c_1, Q_1\rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k\rangle$ *is a spurious counterexample, there exist Boolean expressions* $b_0, \ldots, b_k$ *with* $b_0 \equiv \mathsf{true}$, $b_k \equiv \mathsf{false}$, *and*

$$\forall i \in \{1, \ldots, k\}, \sigma, \sigma' \in \Sigma : \sigma \models b_{i-1}, \langle c_{i-1}, \sigma\rangle \to \langle c_i, \sigma'\rangle \implies \sigma' \models b_i$$

Proof (idea).

Inductive definition of b_i as strongest postconditions:

1. $b_0 := \mathsf{true}$
2. for $i = 1, \ldots, k$: definition of b_i depending on b_{i-1} and on (axiom) transition rule applied in $\langle c_{i-1}, .\rangle \Rightarrow \langle c_i, .\rangle$:

- (skip) $b_i := b_{i-1}$
- (asgn) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
  (x' = previous value of x)
- (if1) $b_i := b_{i-1} \wedge b$
- (if2) $b_i := b_{i-1} \wedge \neg b$
- (wh1) $b_i := b_{i-1} \wedge b$
- (wh2) $b_i := b_{i-1} \wedge \neg b$

(yields $p_k \equiv \mathsf{false}$; by induction on k) □

Abstraction Refinement

Abstraction refinement step:

- Using $b_1, \ldots, k_{k-1}$ as computed before, let $P' := P \cup \{p_1, \ldots, p_n\}$ where $p_1, \ldots, p_n$ are the atomic conjuncts occurring in $b_1, \ldots, k_{k-1}$
- Refine $Abs(P)$ to $Abs(P')$

Lemma

After refinement, the spurious counterexample

$$\langle c_0, \mathsf{true}\rangle \Rightarrow \langle c_1, Q_1\rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k\rangle$$

with $Q_k \neq \mathsf{false}$ *does not exist anymore.*

Proof.

omitted □

Outline

1 Recap: Counterexample-Guided Abstraction Refinement

2 Where CEGAR Fails

3 Craig Interpolation

4 CEGAR Tools

Where CEGAR Fails

Example 17.1

- Let $c_0 :=$
```
[x := a]^0;
[y := b]^1;
while [¬(x = 0)]^2 do
  [x := x - 1]^3;
  [y := y - 1]^4;
if [a = b ∧ ¬(y = 0)]^5 then
  [skip]^6;
else
  [skip]^7;
```
- Interesting property: label 6 unreachable
- Initial abstraction: $P = \emptyset$ ($\Longrightarrow$ $Abs(P) = \{\text{true}, \text{false}\}$)
- Abstraction refinement: on the board
- Observation: iteration yields predicates of the form $x = a{-}k$ and $y = b{-}k$ for all $k \in \mathbb{N}$
- Actually required: loop invariant $a = b \Longrightarrow x = y$, but $x = y$ not generated in CEGAR loop

Outline

1 Recap: Counterexample-Guided Abstraction Refinement

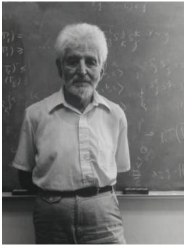
2 Where CEGAR Fails

3 Craig Interpolation

4 CEGAR Tools

- **Problem:** predicates often unnecessarily complex and involving "irrelevant" variables
- **Idea:** consider only variables that are relevant for previous and future part of execution

William Craig (* 1918)



Definition 17.2 (Craig interpolant)

Let $b_1, b_2 \in BExp$ where $b_1 \models b_2$. A Craig interpolant of b_1 and b_2 is a formula $b_3 \in BExp$ with $b_1 \models b_3$, $b_3 \models b_2$, and $Var_{b_3} \subseteq Var_{b_1} \cap Var_{b_2}$.

Using Craig Interpolants I

1. Begin with spurious counterexample $\langle c_0, \text{true}\rangle \Rightarrow \langle c_1, Q_1\rangle \Rightarrow \ldots \Rightarrow \langle c_k, Q_k\rangle$ (according to Definition 16.3)
2. Construct strongest postconditions $s_0, \ldots, s_k$ with $s_0 \equiv \text{true}$, $s_k \equiv \text{false}$ (according to Lemma 16.4)
3. Analogously it is possible to construct weakest preconditions $w_0, \ldots, w_k$ with $w_0 \equiv \text{true}$, $w_k \equiv \text{false}$ starting from w_k
   1. $w_k := \text{false}$
   2. for $i = 0, \ldots, k-1$: definition of b_i depending on b_{i+1} and on (axiom) transition rule applied in $\langle c_i, .\rangle \Rightarrow \langle c_{i+1}, .\rangle$:
      - (skip) $w_i := w_{i+1}$
      - (asgn) $w_i := w_{i+1}[x \mapsto a]$
      - (if1) $w_i := (w_{i+1} \wedge b) \vee \neg b \equiv w_{i+1} \vee \neg b$
      - (if2) $w_i := w_{i+1} \vee b$
      - (wh1) $w_i := w_{i+1} \vee \neg b$
      - (wh2) $w_i := w_{i+1} \vee b$
4. Possible to show: $s_i \models w_i$ for each $i \in \{0, \ldots, k\}$
5. For each $i \in \{0, \ldots, k\}$, choose Craig interpolant b_i of s_i and w_i
6. Refine abstraction by atomic conjuncts occurring in $b_1, \ldots, k_{k-1}$

Remark: Craig interpolants always exist for first-order formulae (but are not necessarily unique)

Using Craig Interpolants II

Example 17.3 (cf. Example 16.5)

Let $c_0 :=$ `[x := z]`^{0}`;[z := z + 1]`^{1}`;[y := z]`^{2}`;`
`if [x = y]`^{3} `then [skip]`^{4} `else [skip]`^{5}

1. Spurious counterexample:

$$\langle 0, \mathsf{true}\rangle \Rightarrow \langle 1, \mathsf{true}\rangle \Rightarrow \langle 2, \mathsf{true}\rangle \Rightarrow \langle 3, \mathsf{true}\rangle \Rightarrow \langle 4, \mathsf{true}\rangle$$

2. Strongest postconditions:

$$\begin{aligned} s_0 &= \mathsf{true} \\ s_1 &= (x = z) \\ s_2 &= (x + 1 = z) \\ s_3 &= (x + 1 = z \wedge y = z) \\ s_4 &= \mathsf{false} \end{aligned}$$

3. Weakest preconditions w_i: on the board
4. Craig interpolants b_i: on the board

Outline

1 Recap: Counterexample-Guided Abstraction Refinement

2 Where CEGAR Fails

3 Craig Interpolation

4 CEGAR Tools

CPAchecker

- CPA: "Configurable Program Analysis"
- Java re-implementation of Berkeley Lazy Abstraction Software Verification Tool (BLAST)
- Software model checker for C programs
- Verifies that software satisfies behavioural requirements of associated interfaces
- Uses CEGAR with Craig interpolation and lazy abstraction
  - abstraction is constructed on-the-fly
  - model locally refined on demand
- Sucessfully applied to C programs with $> 130,000$ LOC
  - D. Beyer, M.E. Keremoglu: *CPAchecker: A Tool for Configurable Software Verification*. Proc. CAV, 2011, 184–190
- WWW: http://cpachecker.sosy-lab.org/

SLAM

- was: Software, Languages, Analysis, and Modeling
- First implementation of CEGAR for C programs
- Also verifies that software satisfies behavioural requirements of associated interfaces
- Supports pointers, memory allocation, and BDD-based model checking
- Sub-tools:
  - `c2bp`: C program $\times$ Predicates $\rightarrow$ Boolean program
  - Bepop: symbolic model checker for (recursive) Boolean programs
  - `newton`: abstraction refinement
- Developed into commercial product (Static Driver Verifier, SDV)
  - T. Ball, V. Levin, S.K. Rajamani: *A Decade of Software Model Checking with SLAM*. Comm. ACM 54(7), 2011, 68–76
- WWW: http://research.microsoft.com/en-us/projects/slam/