Static Program Analysis

Lecture 17: Abstract Interpretation VII (Limits & Improvements of CEGAR)

Winter Semester 2016/17

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ws-1617/spa/

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

Schedule of Lectures

Jan 17/19: Interprocedural DFA

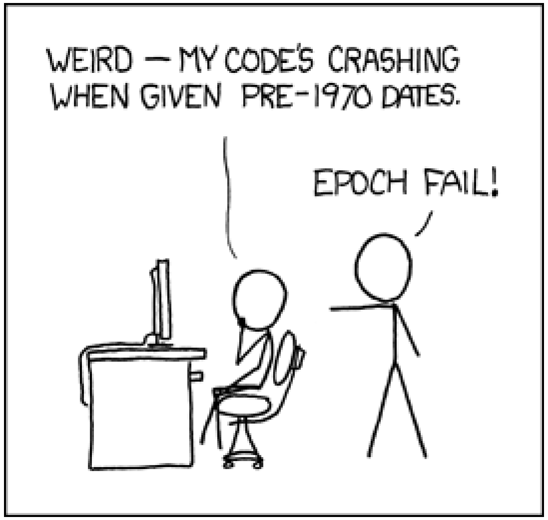
Jan 24/26: [no lectures/exercise classes]

Jan 31/Feb 2: Pointer/shape analysis

Feb 7: [no lecture]

Feb 9: Exam preparation

Seminar *Verification and Static Analysis of Software* (SS 2017)



https://xkcd.com/376

Topics

- Pointer and shape analysis
- Advanced model checking techniques
- Analysis of probabilistic programs
- ...

More information

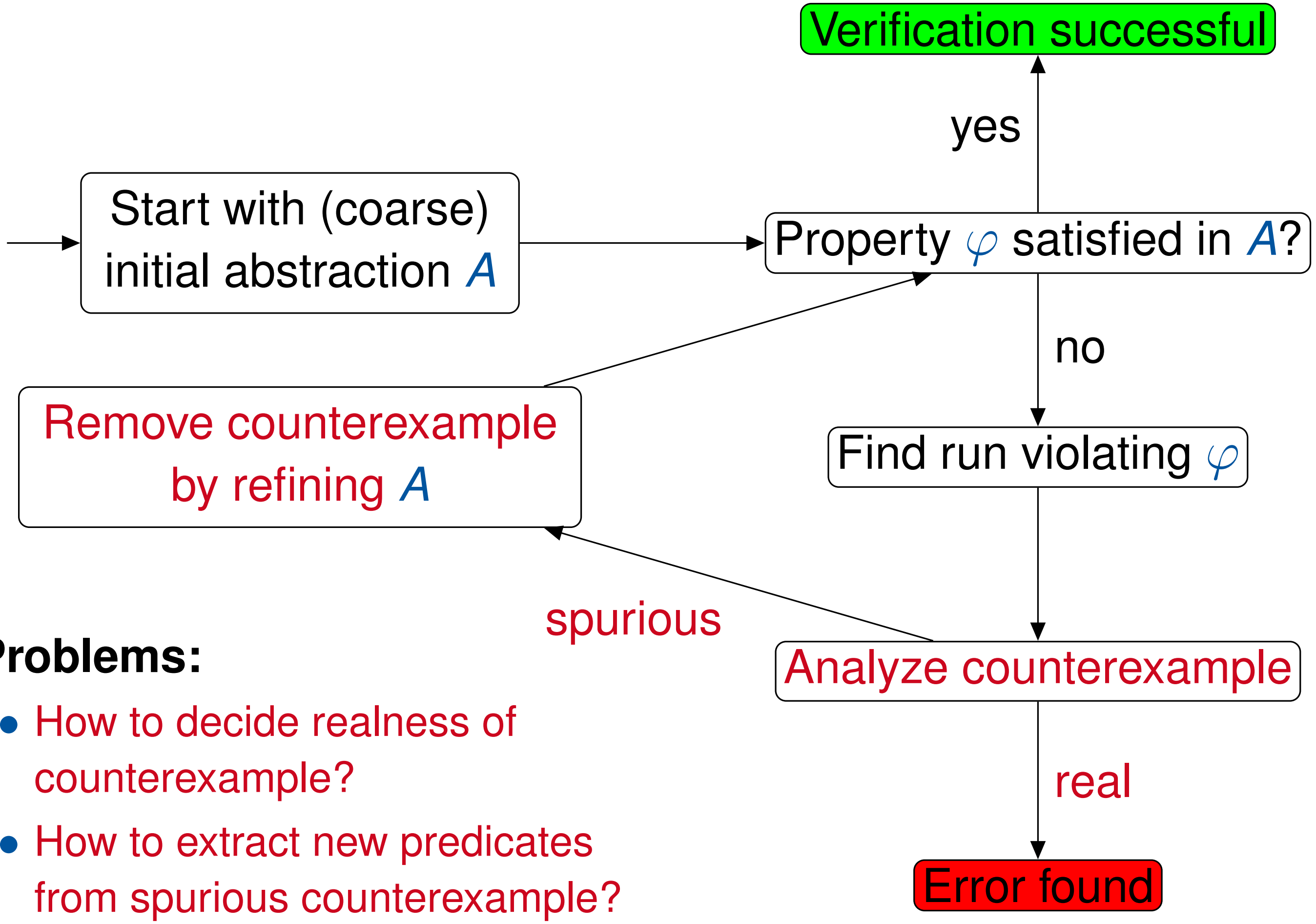
`https://moves.rwth-aachen.de/teaching/ss-17/vsas/`

Registration

between January 13 and 29 via
`https://www.graphics.rwth-aachen.de/apse/`

Recap: Counterexample-Guided Abstraction Refinement (CEGAR)

Reminder: CEGAR



Problems:

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

Abstract Semantics for Predicate Abstraction

Definition (Execution relation for predicate abstraction)

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

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

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

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

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

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

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

Recap: Counterexample-Guided Abstraction Refinement (CEGAR)

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

$\Rightarrow$ All representable as (non-)reachability of "bad locations"

$\Rightarrow$ Counterexample = path to bad locations

Definition (Counterexample)

- A counterexample is a sequence of $k \geq 1$ 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
  - $c_0, \ldots, c_k \in Cmd$ (or $c_k = \downarrow$)
  - $Q_1, \ldots, Q_k \in Abs(P)$ 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
$$\forall i \in \{1, \ldots, k\} : \sigma_i \models Q_i \quad \text{and} \quad \langle c_{i-1}, \sigma_{i-1} \rangle \rightarrow \langle c_i, \sigma_i \rangle$$
- Otherwise it is called spurious.

Elimination of Spurious Counterexamples

Lemma

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

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

Proof (idea).

Inductive definition of b_i as strongest postconditions:

1. $b_0 := \mathrm{true}$
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) $b_i := \exists x'.(b_{i-1}[x \mapsto x'] \wedge x = a[x \mapsto x'])$
  (for $x := a$; 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 $b_k \equiv \mathrm{false}$; by induction on k) □

Recap: Counterexample-Guided Abstraction Refinement (CEGAR)

Abstraction Refinement

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

Lemma

After refinement, the spurious counterexample

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

with $Q_k \not\equiv \text{false}$ *does not exist anymore.*

Proof.

omitted □

Where CEGAR Fails

Example 17.1

- $c :=$ `[x := a]`^{0}`;`

```
[y := b]^1;
while [¬(x = 0)]^2 do
  [x := x - 1]^3;
  [y := y - 1]^4
end;
if [a = b ∧ ¬(y = 0)]^5 then
  [skip]^6
else
  [skip]^7
end
```

- Interesting property: label 6 unreachable

- Initial abstraction: $P = \emptyset$ ($\implies Abs(P) = \{\text{true}, \text{false}\}$)
- Abstraction refinement: on the board
- Observation: iteration yields predicates of the form

$$\texttt{x = a-}k \quad \text{and} \quad \texttt{y = b-}k$$

for all $k \in \mathbb{N}$

- Actually required: loop invariant

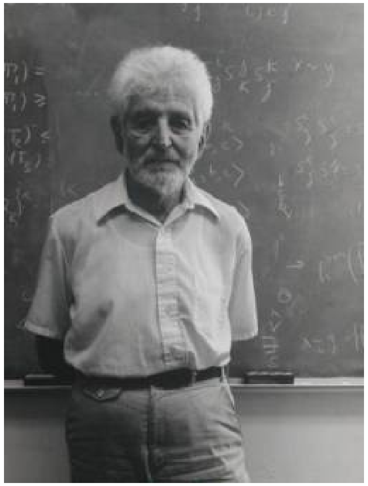
$$\texttt{a = b} \implies \texttt{x = y}$$

but predicate `x = y` not generated in CEGAR loop

Craig Interpolation

Craig Interpolation

- **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 (1918–2016)

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 16.1)
2. Construct strongest postconditions $s_0, \ldots, s_k$ with $s_0 \equiv \text{true}$, $s_k \equiv \text{false}$ (according to Lemma 16.2)
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
   i. $w_k := \text{false}$
   ii. for $i = 0, \ldots, k-1$: definition of w_i depending on w_{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 $b_1, \ldots, b_{k-1}$

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

Using Craig Interpolants II

Example 17.3 (cf. Example 16.3)

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

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 (cf. Example 16.3):
$$\begin{aligned} s_0 &= \mathsf{true} \\ s_1 &= (\mathtt{x} = \mathtt{z}) \\ s_2 &= (\mathtt{x} + 1 = \mathtt{z}) \\ s_3 &= (\mathtt{x} + 1 = \mathtt{z} \wedge \mathtt{y} = \mathtt{z}) \\ s_4 &= \mathsf{false} \end{aligned}$$
3. Weakest preconditions w_i: on the board
4. Craig interpolants b_i: on the board

SLAM Tool

- was: Software, Languages, Analysis, and Modeling

  "SLAM originally was an acronym but we found it too cumbersome to explain. We now prefer to think of 'slamming' the bugs in a program."

  (T. Ball,, B. Cook, V. LevinS.K. Rajamani, Sriram K.: SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft, IFM 2004)
- First implementation of CEGAR for C programs
- Checks behavioural requirements of software interfaces
  - e.g., "a thread may not acquire a lock it has already acquired, or release a lock it does not hold"
- Supports recursive procedures, pointers, and memory allocation
- Sub-tools:
  - `C2bp`: C program $\times$ Predicates $\longrightarrow$ Boolean program (Boolean variables = predicates)
  - BEPOP: symbolic (BDD-based) model checker for (recursive) Boolean programs
  - `newton`: abstraction refinement
- Developed into commercial product (Static Driver Verifier – SDV; part of Windows Driver Foundation development kit)
  - 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/`

CPAchecker Tool

- CPA: “Configurable Program Analysis”
- Java re-implementation of Berkeley Lazy Abstraction Software Verification Tool (BLAST)
- Software model checker for C programs
- Uses CEGAR with Craig interpolation and lazy abstraction
  - abstraction is constructed on-the-fly
  - model locally refined on demand
  - enables use of different predicates at different program points

  $\Rightarrow$ “abstract reachability tree”
- 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/`

Practical Experiences

(cf. V. D'Silva, D. Kroening, G. Weissenbacher: *A Survey of Automated Techniques for Formal Software Verification*, IEEE Trans. on CAD of Integrated Circuits and Systems 27(7), 2008, 1165–1178)

- Predicate abstraction & CEGAR suitable for checking control-flow-related safety properties
  - predicates good for representation of control flow
  - safety ("Nothing bad is going to happen.") goes well with over-approximation
  - liveness ("Eventually something good will happen.") requires under-approximation
- Does not work well with complex heap-based data structures or arrays ($\Rightarrow$ Pointer/Shape Analysis)
- (Real) counterexamples often more useful than correctness proof
- Abstraction refinement cycle may not terminate
- Main application field: safety properties of device drivers and systems code up to 50 kLOCs