Recap: Counterexample-Guided Abstraction Refinement

Program $c$, property $\varphi$

Compute $c_{Abs}$

$c_{Abs} \models \varphi$?

- Yes
- No

Refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

Is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

- Yes
- No

Obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$
CEGAR program $c$, property $\varphi$

compute $c_{Abs}$

$c_{Abs} \models \varphi$?

yes

no

refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

no

yes

obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$
Counterexamples for safety properties

Consider a program $c$ and safety property defined by a program $c_{bad}$. Consider the abstract semantics of $c$ with respect to predicates $P$.

An abstract counterexample is a sequence of abstract transitions

$$(c_0, true) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)$$

such that

- $c_0 = c$ and $c_k = c_{bad}$
- $q_i \not\models false$ for all $1 \leq i \leq k$
Example: abstract counterexample

\[
\begin{align*}
&[x := z]^1; \\
&[z := z + 1]^2; \\
&[y := z]^3; \\
&\text{if } [y = x]^4 \text{ then } \\&\quad \{ \\
&\quad \quad [\text{skip}]^5 \\
&\quad \}\text{ else } \\
&\quad \{ \\
&\quad \quad [\text{skip}]^6 \\
&\quad \}
\end{align*}
\]

property: block 5 is never reached

Initially \( P = \emptyset \)

\( \text{Cubes}(P) = \{ true, false \} \)

abstract counterexample

\[(1, true) \Rightarrow (2, true) \Rightarrow (3, true) \Rightarrow (4, true) \Rightarrow (5, true) \]
CEGAR

program $c$, property $\varphi$

compute $c_{Abs}$

$c_{Abs} \models \varphi$?

no

refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

yes

obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$
Genuine and spurious counterexamples

Consider a program $c$ and safety property defined by a program $c_{bad}$. Consider the abstract semantics of $c$ with respect to predicates $P$.

An abstract counterexample $(c, q_0) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_{bad}, q_k)$ is **genuine** (or **concretizable**) if there exists a sequence of states

$$\sigma_0, \sigma_1, \ldots, \sigma_k$$

such that $\sigma_i \models q_i$ for all $0 \leq i \leq k$ and

$$(c, \sigma_0) \rightarrow (c_1, \sigma_1) \rightarrow \ldots \rightarrow (c_{bad}, \sigma_k)$$

An abstract counterexample that is not genuine is called **spurious**.
Example: spurious counterexample

\[
\begin{align*}
[x := z] & ; \\
[z := z + 1] & ; \\
[y := z] & ; \\
\text{if } [y = x] & \text{ then } \{ \\
\quad [\text{skip}] & \\
\} \\
\text{else } & \{ \\
\quad [\text{skip}] & \\
\}
\end{align*}
\]

property: block 5 is never reached

Initially \( P = \emptyset \)

\[
Cubes(P) = \{ \text{true, false} \}
\]

The abstract counterexample

\[
(1, \text{true}) \Rightarrow (2, \text{true}) \Rightarrow (3, \text{true}) \Rightarrow (4, \text{true}) \Rightarrow (5, \text{true})
\]

is spurious, as there is no execution \( \sigma_0, \ldots, \sigma_4 \) reaching block 5.
Executions as sequences of elementary commands

A sequence of programs $c_0, c_1, \ldots, c_k$ such that

$$(c_0, \ldots) \Rightarrow (c_1, \ldots) \Rightarrow \ldots \Rightarrow (c_k, \ldots)$$

determines a sequence $r_1, \ldots, r_k$ of elementary commands that are executed by the respective transitions in the sequence.

The conditions of if and while are mapped to assert statements depending on which branch is being taken.

Thus, the sequence $r_1, \ldots, r_k$ consists of statements of the form

- skip
- $x := a$
- read($x$)
- assert($b$)
- print($a$)
Consider the sequence of abstract transitions:

\[(x := x + 1; \textbf{if } \neg(y = 0) \textbf{ then } \{x := x - y\} \textbf{ else } \{y := 2; \text{skip}\}, ... ) \Rightarrow \]

\[(\textbf{if } \neg(y = 0) \textbf{ then } \{x := x - y\} \textbf{ else } \{y := 2; \text{skip}\}, ... ) \Rightarrow \]

\[(y := 2; \text{skip}, ...) \Rightarrow \]

\[(\text{skip}, ...) \Rightarrow \]

The corresponding sequence of commands is

\[x := x + 1; \textbf{assert}(y = 0); y := 2\]

Characterization of spurious counterexamples

Lemma
An abstract counterexample \((c_0, q_0) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)\) is spurious if and only if there exist predicates \(p_0, \ldots, p_k\) such that:

\[\begin{align*}
&\quad p_0 = true \text{ and } p_k = false \\
&\text{for all } 1 \leq i \leq k, \{ p_{i-1} \} r_i \{ p_i \} \text{ is a valid Hoare triple (for all } \sigma, \sigma' \in States, \text{ if } \sigma \models p_{i-1} \text{ and } (r_i, \sigma) \rightarrow \sigma', \text{ then, } \sigma' \models p_i). \end{align*}\]

Corollary
An abstract counterexample \((c_0, q_0) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)\) is spurious if and only if \{true\}r{false} is a valid Hoare triple, where we define the acyclic program \(r\) as \(r = r_1; r_2; \ldots; r_k\). This is equivalent to \(sp(true, r) \models false\) and to \(wp(r, false) \models true.\)
Proof of the Lemma

Lemma

An abstract counterexample \((c_0, q_0) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)\) is spurious if and only if there exist predicates \(p_0, \ldots, p_k\) such that:

- \(p_0 = \text{true} \) and \(p_k = \text{false}\)
- for all \(1 \leq i \leq k\), \(\{p_{i-1}\}r_i\{p_i\}\) is a valid Hoare triple (for all \(\sigma, \sigma' \in \text{States}\), if \(\sigma \models p_{i-1}\) and \((r_i, \sigma) \rightarrow \sigma'\), then, \(\sigma' \models p_i\)).

Proof Define \(p_0 := \text{true} \) and \(p_i := sp(p_{i-1}, r_i)\) for \(i = 1, \ldots, k\).

\[
\begin{align*}
sp(p, \text{skip}) & \models p \\
sp(p, \text{print}(a)) & \models p \\
sp(p, x := a) & \models \exists x'(p[x'/x] \land x = (a[x'/x])) \\
sp(p, \text{read}(x)) & \models \exists x'(p[x'/x]) \\
sp(p, \text{assert}(b)) & \models p \land b
\end{align*}
\]
Proof of the Lemma (continued)

Clearly, \( \{p_{i-1}\} r_i \{p_i\} \) is a valid Hoare triple for all \( 1 \leq i \leq k \).

Since the formula \( sp(p_{i-1}, r_i) \) characterizes the set of states that are successors of states in \( \{\sigma \in States \mid \sigma \models p_{i-1}\} \) with respect to the command \( r_i \), it is easy to see that \( p_k \models \text{false} \) if and only if the abstract counterexample is spurious.
Example: spurious counterexample

\[x := z;\]
\[z := z + 1;\]
\[y := z;\]
\[\text{if } [y = x] \text{ then } \{\]
\[\text{[skip]}\]
\} \]
\text{else } \{ 
\[\text{[skip]}\] 
\} 

abstract counterexample

\[(1, true) \Rightarrow (2, true) \Rightarrow (3, true) \Rightarrow (4, true) \Rightarrow (5, true)\]

strongest postconditions

\[p_0 := true\]
\[p_1 := \exists x'(true \land x = z)\]
\[\models x = z\]
\[p_2 := \exists z'(x = z' \land z = z' + 1)\]
\[\models z = x + 1\]
\[p_3 := \exists y'(z = x + 1 \land y = z)\]
\[\models z = x + 1 \land y = z\]
\[p_4 := z = x + 1 \land y = z \land y = x\]
\[\models false\]
CEGAR program $c$, property $\varphi$

compute $c_{Abs}$

$\text{Abs}\vdash \varphi$?

is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$
Extraction of refinement predicates

Let \((c, true) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)\) be a spurious abstract counterexample and define the predicates \(p_0, p_1, \ldots, p_k\) where

- \(p_0 := true\),
- \(p_i := sp(p_{i-1}, r_i)\) for \(1 \leq i \leq k\).

We take \(P' := P \cup \{p_1, \ldots, p_n\}\) as the refined set of predicates.

**Lemma**

In the abstract semantics of \(c\) with respect to predicates \(P'\), if

\[(c, true) \Rightarrow (c_1, q'_1) \Rightarrow \ldots \Rightarrow (c_k, q'_k)\]

is a sequence with the same programs \(c, c_1, \ldots, c_k\) as the spurious counterexample, then it holds that \(q'_k = false\). That is, the spurious counterexample was eliminated by the refinement.
Proof of the lemma

Consider a sequence

\[(c, \text{true}) \Rightarrow (c_1, q'_1) \Rightarrow \ldots \Rightarrow (c_k, q'_k)\]

We will show that \(q'_k \models false\) by establishing \(q'_i \models p_i\) for all \(i \in \{0, \ldots, k\}\). The latter we will show by induction.

For \(i = 0\) we have \(q'_i = p'_i = \text{true}\).

Induction hypothesis: \(q'_i \models p_i\) for some \(i\).

Since \(sp\) is monotonic with respect to \(\models\), by the induction hypothesis we have \(sp(q'_i, r_{i+1}) \models sp(p_i, r_{i+1})\). Thus,

\[
sp(q'_i, r_{i+1}) \models p_{i+1} \quad \text{（definition of } p_{i+1})
\]

\[
\Rightarrow sp(q'_i, r_{i+1}) \models \overline{p_{i+1}} \quad \text{（homework）}
\]

\[
q'_{i+1} \models \overline{p_{i+1}} \quad \text{（definition of } \Rightarrow) \]

\[
q'_{i+1} \models p_{i+1} \quad (p_{i+1} \in P')
\]
Example: abstraction refinement

\[ x := z \]
\[ z := z + 1 \]
\[ y := z \]
\[ \text{if } y = x \text{ then } \{
    \text{[skip]}
\} \]
\[ \text{else } \{
    \text{[skip]}
\} \]

refined abstraction with
\[ P' = P \cup \{p_1, p_2, p_3\} \]
where
\[ p_1 := (x = z) \]
\[ p_2 := (z = x + 1) \]
\[ p_3 := (z = x + 1 \land y = z) \]

\[ (1, \text{true}) \Rightarrow (2, p_1 \land \neg p_2 \land \neg p_3) \]
\[ \Rightarrow (3, \neg p_1 \land p_2) \]
\[ \Rightarrow (4, \neg p_1 \land p_2 \land p_3) \]
\[ \Rightarrow (5, \neg p_1 \land p_2 \land p_3 \land x = y) \]

\[ \neg p_1 \land p_2 \land p_3 \land x = y \models false \]

Remark: Often, the atomic formulas occurring in the strongest postconditions are used as refinement predicates. That is, we could take \( \{x = z, z = x + 1, y = x\} \). Note however, that our abstract domain does not allow us to express disjunctions over \( \text{Cubes}(P) \).
Counterexample-Guided Abstraction Refinement

Program $c$, property $\varphi$

Compute $c_{Abs}$

$c_{Abs} \models \varphi$?

refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution

is $r_{Abs}$ concretizable to an execution $r$ of $c$ such that $r \not\models \varphi$?

obtain abstract execution $r_{Abs}$ of $c_{Abs}$ such that $r_{Abs} \not\models \varphi$
Example: non-terminating refinement

\[
[x := a]; \quad [y := b]
\]

while \([\neg(x = 0)]\) do 
\{
\[
x := x - 1; \quad y := y - 1
\]
\}

if \([a = b \land \neg(y = 0)]\) then 
\{
  [skip]
\}

else 
\{
  [skip]
\}

Block 7 is never reached.

CEGAR produces an infinite sequence of spurious counterexamples, for \(n \in \mathbb{N}\), with predicates

\[
x = a - n \quad y = b - n
\]

The described refinement does not discover the loop invariant

\[a = b \rightarrow x = y\]

since \(x = y\) is not generated
Example: non-terminating refinement

\[
\begin{align*}
[x := a]^1; \\
[y := b]^2
\end{align*}
\]

\textbf{while} \ [\neg(x = 0)]^3 \ \textbf{do} \ \{ \\
\begin{align*}
[x := x - 1]^4; \\
[y := y - 1]^5
\end{align*}
\}

\textbf{if} \ [a = b \land \neg(y = 0)]^6 \ \textbf{then} \ \{ \\
[skip]^7
\}

\textbf{else} \ \{ \\
[skip]^8
\}

initial set of predicates \( P = \emptyset \)

abstract counterexample

\((1, true) \Rightarrow (2, true) \Rightarrow (3, true) \Rightarrow (6, true) \Rightarrow (7, true) \)

abstract counterexample analysis

\begin{align*}
p_0 & := true \\
p_1 & := x = a \\
p_2 & := x = a \land y = b \\
p_3 & := x = a \land y = b \land x = 0 \\
p_4 & := x = a \land y = b \land x = 0 \land \neg(y = 0) \\
\implies & \quad false
\end{align*}
Example: non-terminating refinement

\[
\begin{align*}
x &:= a^1; \\
y &:= b^2 \\
\textbf{while } &\neg(x = 0) \textbf{ do } \\
&\{ \\
&x := x - 1^4; \\
y := y - 1^5 \\
&\} \\
\textbf{if } &a = b \land \neg(y = 0) \textbf{ then } \{}\\
&\text{skip}^7 \\
\textbf{else } \{}\\
&\text{skip}^8 \\
&\}
\end{align*}
\]

refined set of predicates
\[
P' = \{ a = b, x = 0, y = 0, \\
x = a, y = b \}
\]

abstract counterexample
\[
(1, true) \Rightarrow (2, true) \Rightarrow (3, true) \\
\Rightarrow (4, true) \Rightarrow (5, true) \Rightarrow (3, true) \\
\Rightarrow (6, true) \Rightarrow (7, true)
\]

abstract counterexample analysis
\[
\ldots
\]
\[
p_3 := x = a \land y = b \land \neg(x = 0) \\
p_4 := x = a - 1 \land y = b \land \neg(a = 0) \\
p_5 := x = a - 1 \land y = b - 1 \land \neg(a = 0) \\
\ldots
\]

generate predicates \( x = a - 1, y = b - 1 \), and go on \ldots
Optimizations for CEGAR

In practice: numerous optimizations of this scheme. Two examples:

- Generation of simpler predicates involving only relevant variables. The most common technique is Craig interpolation.

  A Craig interpolant for formulas $b$ and $p$ with $b \models p$ is a formula $r$ that satisfies the following conditions:
  
  - $b \models r$ and $r \models p$
  - $\text{Vars}(r) \subseteq \text{Vars}(b) \cap \text{Vars}(p)$.

  Craig interpolants are guaranteed to exist for formulas in First-Order logic and can be computed from resolution proofs.

- Local refinement: associate sets of predicates with blocks of the program to reduce the size of the abstraction.

Refinement via Craig interpolation

Let \((c, true) \Rightarrow (c_1, q_1) \Rightarrow \ldots \Rightarrow (c_k, q_k)\) be a spurious abstract counterexample and define the strongest postconditions \(b_0, b_1, \ldots, b_k\) where

\[
\begin{align*}
  b_0 &:= true, \\
  b_i &:= sp(b_{i-1}, r_i) \text{ for } 1 \leq i \leq k.
\end{align*}
\]

Similarly we can define the weakest preconditions \(p_0, p_1, \ldots, p_k\) by

\[
\begin{align*}
  p_k &:= false, \\
  p_i &:= wp(r_{i+1}, p_{i+1}) \text{ for } 0 \leq i < k.
\end{align*}
\]

Intuition:

- \(b_i\): the set of concrete states reachable by executing \(r_1; \ldots; r_i\)
- \(p_i\): the set of concrete states from which the program location \(c_{bad}\) cannot be reached by executing \(r_{i+1}; \ldots; r_k\)

Property: \(b_i \models p_i\) for each \(i = 0, \ldots, k\)

Abstraction refinement: use Craig interpolants \(r_i\) for \(b_i\) and \(p_i\)
Weakest preconditions

\[
\begin{align*}
wp(\text{skip}, p) & \models p \\
wp(\text{print}(a), p) & \models p \\
wp(x := a, p) & \models p[a/x] \\
wp(\text{read}(x), p) & \models \forall x'(p[x'/x]) \\
wp(\text{assert}(b), p) & \models p \lor \lnot b
\end{align*}
\]
Example: refinement via Craig interpolation

\[
\begin{align*}
[x := z] &; \\
[z := z + 1] &; \\
y := z &; \\
\textbf{if} \ [y = x] &; \textbf{then} \{ \\
\quad \text{[skip]} &; \\
\} \\
\textbf{else} \{ \\
\quad \text{[skip]} &; \\
\}
\end{align*}
\]

spurious counterexample

\[(1, true) \Rightarrow (2, true) \Rightarrow (3, true) \Rightarrow (4, true) \Rightarrow (5, true)\]

strongest postconditions:

\begin{align*}
\textcolor{blue}{{b_0}} &:= \textcolor{blue}{true} \\
\textcolor{blue}{{b_1}} &:= (x = z) \\
\textcolor{blue}{{b_2}} &:= (z = x + 1) \\
\textcolor{blue}{{b_3}} &:= (z = x + 1 \land y = z) \\
\textcolor{blue}{{b_4}} &:= \textcolor{blue}{false}
\end{align*}

weakest preconditions:

\begin{align*}
p_4 &:= false \\
p_3 &:= \neg(y = x) \\
p_2 &:= \neg(z = x) \\
p_1 &:= \neg(z + 1 = x) \\
p_0 &:= \neg(z + 1 = z)
\end{align*}

an interpolant for \(b_1\) and \(p_1\): \(x = z\)

an interpolant for \(b_3\) and \(p_3\): \(\neg(y = x)\)
Some verification tools based on CEGAR

**SLAM** (Software, Languages, Analysis and Modeling)
- first implementation of CEGAR for C programs
- supports pointers, memory allocation
- developed into **SDV** (Static Driver Verifier) by Microsoft

**CPA checker** (Configurable Program Analysis)
- reimplements **BLAST** (Berkeley Lazy Abstraction Software verification Tool) [http://mtc.epfl.ch/software-tools/blast/](http://mtc.epfl.ch/software-tools/blast/)
- software model checker for C programs
- uses Craig interpolation and lazy abstraction
- [https://cpachecker.sosy-lab.org/](https://cpachecker.sosy-lab.org/)