CounterExample-Guided Abstraction Refinement (CEGAR)

Martin Fränzle\textsuperscript{a}

(with many slides © S. Ratschan\textsuperscript{b})

\textsuperscript{a} Carl von Ossietzky Universität, Oldenburg, Germany

\textsuperscript{b} Czech Academy of Sciences, Prague, Czech Rep.
The problem

• Abstraction is a powerful method for verifying systems
  • maps complex system (e.g., infinite state) to simpler system (e.g., finite Kripke structure)
  • simpler model may be amenable to automatic state-exploratory verification
• but finding the right abstraction is hard
  • may be too coarse \(\leadsto\) verification fails
  • may be too fine \(\leadsto\) state-space exploration impossible
  • may even be too fine in some places and too coarse in others
The idea

In manual verification, we often add information on demand:

- Upon a failing proof, we analyze the reasons and
- add preconditions as necessary.

Can we do the same within abstraction-based model checking?

- Upon a failing proof, let the model-checker analyze the reasons and
- refine the abstraction as necessary.
Abstraction Refinement

Idea:

• conservatively approximate the hybrid system by a finite Kripke structure (the abstraction)

• if abstraction safe, done

• while abstraction not safe, refine it

• counter-example based: refine to remove a given spurious counter-example (Clarke et al. 03, Alur et al. 03)
Basic CEGAR
Spurious counterexample

**Def:** Let $A \succ C$ be an homomorphic abstraction wrt. abstraction function $h$. Let $\phi$ be an $\forall$CTL formula and $\pi = (c_1, c_2 \ldots)$ be an anchored path of $C$ witnessing violation of $\phi$ on $C$. Then $\pi$ is called a counterexample for $\phi$ on $C$. Furthermore, $h(\pi) = (h(c_1), h(c_2), \ldots)$ then is an anchored path of $A$ which violates $\phi$, i.e. a counterexample on $A$. We do then call $h(\pi)$ the abstract counterexample corresponding to $\pi$ and we call $\pi$ the concrete counterexample corresponding to $h(\pi)$.

**Def:** If $\pi_A$ is a counterexample on the abstraction $A \succ C$ which has no corresponding concrete counterexample on $C$ then we call $\pi_A$ a spurious counterexample.
Abstraction Refinement

**Def:** If $C \prec A' \prec A$ then $A$ and $A'$ are called abstraction of $C$ and $A'$ is called an abstraction refinement of $A$.

**Idea:** Whenever there is a spurious counterexample in $A$, identify an abstraction refinement $A'$ that lacks that particular spurious counterexample.
CEGAR algorithm (simple version: invariants)

To verify $C \models AGp$ do

1. build finite Kripke structure $A \succ C$,
2. model-check $A \models AGp$,
3. if this holds then report $C \models AGp$ and stop,
4. otherwise validate the counterexample on $C$, i.e., find a corresponding concrete counterexample,
5. if a corresponding concrete counterexample exists then report $C \not\models AGp$ and stop,
6. otherwise use the spurious counterexample to refine $A$ and restart from 2.
The crucial ingredients of CEGAR

- Model checking,
- validation/concretization of counterexample,
- guided refinement of abstraction.
Validation of counterexample

Given: $A \succ C$ and an abstract counterexample $\phi = (a_1, a_2, \ldots, a_n)$ on $A$.

Alg: Provided we can effectively manipulate pre-images of the abstraction morphism $h$, proceed as follows:

1. Compute $S_1 := h^{-1}(a_1) \cap I_C$, where $I_C$ is the set of initial states of $C$,
2. For $i = 2$ to $n$, compute $S_i := h^{-1}(a_i) \cap \text{Post}(S_{i-1})$.
   Abort as soon as some $S_i$ becomes $\emptyset$.
   In this case, the counterexample has been shown to be spurious.
3. In case of proper termination of the loop, the counterexample is real.

N.B. Assumes that $h^{-1}(a_i)$, $\text{Post}(S_i)$, and their intersections are computable (in the sense of an effective emptiness test)!
State splitting

Idea: For a set $C_i = h^{-1}(a_i)$ of concrete states represented by an abstract state $a_i$ occurring in the spurious counterexample, split it into $C_i \cap \text{Post}(h^{-1}(a_{i-1}))$ and $C_i \setminus \text{Post}(h^{-1}(a_{i-1}))$, provided both non-empty (or into $C_1 \cap I_C$ and $C_1 \setminus I_C$ in case $i = 1$).

Approach: Replace $a_i$ by two states $a_i^+$ and $a_i^-$ representing $C_i \cap \text{Post}(h^{-1}(a_{i-1}))$ and $C_i \setminus \text{Post}(h^{-1}(a_{i-1}))$, resp.

Technique: Replace the Kripke structure $A = (V, E, L, I)$ by $A' = (V', E', L', I')$ with

\begin{itemize}
  \item $V' = V \setminus \{a_i\} \cup \{a_i^+, a_i^-\}$, where the latter are $\notin V$,
  \item $E' = E \cap (V' \times V') \cup \{(a_i^+, a_i^-), (a_i^-, a_i^+)\} \cup \{(a, a_i^+) | (a, a_i) \in E\} \cup $
  $\{(a, a_i^-) | (a, a_i) \in E, a \neq a_{i-1}\} \cup \{(a_i^+, a), (a_i^-, a) | (a_i, a) \in E\}$
  \item $L'(v) = \begin{cases} L(v) & \text{if } v \in V, \\ L(a_i) & \text{if } v \in \{a_i^+, a_i^-\}, \end{cases}$
  \item $I' = \begin{cases} I & \text{if } C_i \cap I_C = \emptyset, \\ I \setminus \{a_i\} \cup \{a_i^+\} & \text{otherwise}. \end{cases}$
\end{itemize}
Resulting morphism

\[ h'(c) = \begin{cases} 
  a_i^+ & \text{if } c \in C_i \cap \text{Post}(h^{-1}(a_{i-1})), \\
  a_i^- & \text{if } c \in C_i \setminus \text{Post}(h^{-1}(a_{i-1})), \\
  h(c) & \text{otherwise.}
\end{cases} \]
Refining $E'$: transition pruning

Observation: Pre- and post-images of $h'^{-1}(a_i^+)$ or $h'^{-1}(a_i^-)$ may well have empty intersections with sets that the pre- or post-set of $h'^{-1}(a_i)$ did intersect with. In such cases, $E'$ contains spurious edges.

Solution: Remove such edges by pruning $E'$ to

$$E'' = \{(s, t) \in E' | \text{Post}(h'^{-1}(s)) \cap h'^{-1}(t) \neq \emptyset\}$$
CEGAR algorithm (simple version: invariants)

To verify $C \models AGp$ do

1. build finite Kripke structure $A \triangleright C$,
2. model-check $A \models AGp$,
3. if this holds then report $C \models AGp$ and stop,
4. otherwise validate the counterexample on $C$, i.e., find a corresponding concrete counterexample,
5. if a corresponding concrete counterexample exists then report $C \not\models AGp$ and stop,
6. otherwise use the spurious counterexample to split states in $A$,
7. perform transition pruning on the resulting refinement $A'$,
8. goto 2.

Concrete version is just an example, variants of split/prune rules abound.
Application to hybrid systems

- Above procedure is effective if $h^{-1}(a_i)$, $\text{Post}(S_i)$, and their intersections are computable (in the sense of an effective emptiness test).
- This is in general not true for hybrid systems.
⇒ Need to embed an appropriate form of approximation of the above sets into CEGAR.
CEGAR on hybrid states

Conservative approximation of state sets
Application to hybrid systems

• “Naive” CEGAR procedure is effective if $h^{-1}(a_i)$, $\text{Post}(S_i)$, and their intersections are computable (in the sense of an effective emptiness test).

• In general not true for hybrid systems, thus embed an appropriate form of approximation of the above sets into CEGAR.

• Main difficulty is computation of successor states: explicit (jumps) and implicit transitions (flows, defined by ODE)
  
  • Multiple shapes of overapproximation can be used
    • various effective representations of subsets of $\mathbb{R}^n$: rectangular boxes, zonotopes, polyhedra, ellipsoids, . . .,
    • multiple techniques for conservatively approximating hybrid transitions (jumps & flows)
  
  • can be combined to obtain an adaptive CEGAR algorithm
    • e.g., proceeds from coarse to fine, investing computational effort to increase precision when necessary.
Computing successors

- CEGAR algorithm applies different approximations of successor computation in sequence,
- proceeds from coarse to fine, investing more computational effort to increase precision only when necessary,
- hope is that crucial deductions (absence of counterexamples, non-concretizability of a certain counter-example) can often be obtained on coarse abstractions,
- CEGAR needs to compute different relative successors $\text{Succ}(X, Y) = \text{Post}(X) \cap Y$, where $X, Y \in \mathcal{P}(\mathbb{R}^n)$.
- Can approximate these by any operation $\text{SUCC} : \mathcal{P}(\mathbb{R}^n) \times \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$ with
  1. Overapproximation: $\text{SUCC}(X, Y) \supseteq \text{Post}(X) \cap Y$,
  2. Reasonability: $\text{SUCC}(X, Y) \subseteq Y$. 
Validation of counterexample

Given: $A \succ C$ and an abstract counterexample $\phi = (a_1, a_2, \ldots, a_n)$ on $A$.

Alg: For a sequence of successively tighter overapproximations $(SUCC_i)_{i=1,\ldots,k}$, proceed as follows:

1. Start with $i = 1$, i.e., the coarsest approximation.
2. Compute $S^i_1 := overapprox_i(h^{-1}(a_1) \cap I_C)$, where $I_C$ is the set of initial states of $C$.
3. For $j = 2$ to $n$, compute $S^i_j := SUCC_i(S^i_{j-1}, h^{-1}(a_i))$.
   Abort as soon as some $S^i_j$ becomes $\emptyset$.
   In this case, the counterexample is spurious.
4. In case of proper termination of the inner loop, restart at 1. with $i := i + 1$, i.e., the next finer approximation, if $i < k$.
5. If the inner loop terminates regularly for $i = k$, then the abstract counterexample can’t be refuted by any of the overapproximations. (Probably is real.)
HSolver

Overapproximation via Constraint-based Reasoning

Stefan Ratschan, Czech Academy of Sciences
Shikun She, MPII, Saarbrücken
Starting Point: Interval Grid Method

Stursberg/Kowalewski et al., one-mode case:

\[ [-5, -1] \leq x \leq [1, 1] \]

- put transitions between all neighboring hyperrectangles (boxes), mark all as initial/unsafe
- remove impossible transitions/marks (interval arithmetic check on boundaries/boxes)

Result: finite abstraction
Interval arithmetic

Is a method for calculating an interval *covering* the possible values of a real operator if its arguments range over intervals:

\[
[a, A] \circ [b, B] = [a + b, A + B]
\]

\[
[a, A] \cdot [b, B] = [\min\{ab, aB, Ab, AB\}, \max\{ab, aB, Ab, AB\}]
\]

\[
\min ([a, A], [b, B]) = [\min\{a, b\}, \min\{A, B\}]
\]

\[
\sin ([a, A]) = \left[ \min\{\sin x \mid x \in [a, A]\}, \max\{\sin x \mid x \in [a, A]\} \right]
\]

\[
f ([a, A], [b, B], \ldots) = \left[ \min\{f(\vec{x}) \mid \vec{x} \in [a, A] \times [b, B] \times \ldots\}, \max\{f(\vec{x}) \mid \vec{x} \in [a, A] \times [b, B] \times \ldots\} \right]
\]

**Theorem:** For each term \( t \) with free variables \( \vec{v} \):

\[
\{ t(\vec{v} \mapsto \vec{x}) \mid \vec{x} \in [a, A] \times [b, B] \times \ldots \} \subseteq \circ t (v_1 \mapsto [a, A], v_2 \mapsto [b, B], \ldots)
\]
Is the approximation tight?

1. In the limit: yes!

\[ t(\vec{v} \mapsto \vec{x}) = \circ t (v_1 \mapsto [x_1, x_1], v_2 \mapsto [x_2, x_2], \ldots) \]

\[ t(\vec{v} \mapsto \vec{x}) = \lim_{\varepsilon \to 0} \circ t (v_1 \mapsto [x_1 - \varepsilon, x_1 + \varepsilon], v_2 \mapsto [x_2 - \varepsilon, x_2 + \varepsilon], \ldots) \]

provided \( t \) is uniformly continuous.

2. In general: No! If \( a < A \) then

\[ x - x(x \mapsto [a, A]) = [a, A] - [a, A] = [a - A, A - a] \neq [0, 0] \]

Dependency problem of interval arithmetic:

😊 Tight bounds only if each variable occurs at most once!
Check safety on resulting finite abstraction
if safe: finished, otherwise: refine grid;
continue until success
More modes: separate grid for each mode
Jumps: also check using interval arithmetic
Discussion

Advantages:
- can deal with constants that are only known up to intervals
- interval tests cheap (e.g., compare to explicit computation of continuous reach sets, or full decision procedures)

Disadvantages:
- may require a very fine grid to provide an affirmative answer (curse of dimensionality)
- ignores the continuous behavior within the grid elements

Let’s remove them!
Removing Disadvantages

reflect more information in abstraction without creating more boxes by splitting

Observation: we do not need to include information on unreachable state space, remove such parts from boxes
Reach Set Pruning

A point in a box $B$ can be reachable

- from the initial set via a flow in $B$
- from a jump via a flow in $B$
- from a neighboring box via a flow in $B$

formulate corresponding constraints, remove all points from box that do not fulfill one of these constraints
Constraints in Specification

We specify system using constraints:

- **Flow** \((s, \vec{x}, \dot{\vec{x}})\) (e.g., \(s = \text{off} \rightarrow \dot{x} = x \sin(x) + 1 \ldots\))
  - purely syntactic!
  - even implicit and algebraic!

- **Jump** \((s, \vec{x}, s', \vec{x}')\) (e.g.,
  \((s = \text{off} \land x \geq 10) \rightarrow (s' = \text{on} \land x' = 0))\)

- **Init** \((s, \vec{x})\)
Reachability Constraints

Lemma (*n*-dimensional mean value theorem): For a box $B$, mode $s$, if a point $(y_1, \ldots, y_n) \in B$ is reachable from a point $(x_1, \ldots, x_n) \in B$ via a flow in $B$ then

$$\exists t \in \mathbb{R}_{\geq 0} \land \exists a_1, \ldots, a_k, \dot{a}_1, \ldots, \dot{a}_k[(a_1, \ldots, a_k) \in B \land 1 \leq i \leq n \land \text{Flow}(s, (a_1, \ldots, a_k), (\dot{a}_1, \ldots, \dot{a}_k)) \land y_i = x_i + \dot{a}_i \cdot t]$$

Denote this constraint by $\text{flow}_B(s, \vec{x}, \vec{y})$. 
Reachability Constraints

Lemma: For a box $B \subseteq \mathbb{R}^k$, mode $s$, if $\vec{y} \in B$ is reachable from the initial set via a flow in $B$ then

$$\exists \vec{x} \in B \ [\text{Init}(s, \vec{x}) \land \text{flow}_B(s, \vec{x}, \vec{y})]$$

Lemma: For a box $B \subseteq \mathbb{R}^k$, mode $s$, $\vec{y} \in B$, $(s, \vec{y})$ is reachable from a jump from a box $B^*$ and mode $s^*$ via a flow in $B$ then

$$\exists \vec{x}^* \in B^* \exists \vec{x} \in B \ [\text{Jump}(s^*, \vec{x}^*, s, \vec{x}) \land \text{flow}_B(s, \vec{x}, \vec{y})]$$
**Lemma:** For a box $B \subseteq \mathbb{R}^k$, mode $s$, if $\vec{y} \in B$ is reachable from a neighboring box over a face $F$ of $B$ and a flow in $B$ then

$$\exists \vec{x} \in F \left[ \text{incoming}_F(s, \vec{x}) \land \text{flow}_B(s, \vec{x}, \vec{y}) \right],$$

where $\text{incoming}(s, \vec{x})$ is of the form

$$\exists \dot{x}_1, \ldots, \dot{x}_k \left[ \text{Flow}(s, \vec{x}, (\dot{x}_1, \ldots, \dot{x}_k)) \land \dot{x}_j \mathbin{r} 0 \right]$$

where $r \in \{\leq, \geq\}$, $j \in \{1, \ldots, k\}$ depends on the face $F$

for corners etc. a little bit more involved
Using Constraints

After substituting definitions, getting rid of quantifiers, interval constraint propagation algorithms can remove parts from boxes not fulfilling such constraints.

• correct handling of rounding errors
• almost negligible time
• result not necessarily tight (but tight for \( \text{flow}_B(s, \vec{x}, \vec{y}) \) in linear case)

http://rsolver.sourceforge.net