Recap: 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 $r_{Abs} \not\models \varphi$
Recap: CEGAR

1. Program $c$, property $\varphi$
2. Compute $c_{Abs}$
   - 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 $r_{Abs} \not\models \varphi$
     - No: Refine the abstraction constructing $c_{Abs}$ where $r_{Abs}$ is not an execution
3. $c_{Abs} \models \varphi$?
   - Yes: \( \checkmark \)
   - No: \( \times \)
Recap: Abstract transition relation

We defined the abstract transition relation

\[ \Rightarrow \subseteq (Progs \times Abs(P)) \times ((Progs \times Abs(P)) \cup Abs(P)) \]

by the following rules

**SKIP**

\[ ([\text{skip}]^l, q) \Rightarrow q \]

**ASSIGN**

\[ ([x := a]^l, q) \Rightarrow sp(q, x := a) \]

**READ**

\[ ([\text{read}(x)]^l, q) \Rightarrow sp(q, \text{read}(x)) \]

**PRINT**

\[ ([\text{print}(x)]^l, q) \Rightarrow q \]
Recap: Abstract transition relation

\[
\begin{align*}
\text{SEQ}_1 & \quad (c_1, q) \Rightarrow q' \\
& \quad (c_1; c_2, q) \Rightarrow (c_2, q') \\

\text{SEQ}_2 & \quad (c_1, q) \Rightarrow (c_1', q') \\
& \quad (c_1; c_2, q) \Rightarrow (c_1'; c_2, q') \\

\text{IF}_{\text{true}} & \quad (\text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_1, q \land b) \\

\text{IF}_{\text{false}} & \quad (\text{if } [b]^l \text{ then } \{c_1\} \text{ else } \{c_2\}, q) \Rightarrow (c_2, q \land \neg b) \\

\text{WHILE}_{\text{true}} & \quad (\text{while } [b]^l \text{ do } \{c\}, q) \Rightarrow (c; \text{while } [b]^l \text{ do } \{c\}, q \land b) \\

\text{WHILE}_{\text{false}} & \quad (\text{while } [b]^l \text{ do } \{c\}, q) \Rightarrow q \land \neg b
\end{align*}
\]
Example

```
if \[x > y\]^1 \text{ then } \{
  \text{while } [\neg(y = 0)]^2 \text{ do } \{
    \[x := x - 1\]^3;
    \[y := y - 1\]^4
  \}
  \text{if } [x > y]^5 \text{ then } \{
    \text{[skip]}^6
  \}
  \text{else } \{
    \text{[skip]}^7
  \}
\}
\text{else } \{
  \text{[skip]}^8
\}
```

Prove that location 7 in the program is not reachable

\(x > y\) is a loop invariant

In order to establish the invariant \(x > y\), we need to also add the predicate \(x \geq y\)

Take \(P = \{p_1, p_2\}\), where

\[
p_1 = x > y \\
p_2 = x \geq y
\]
Example: abstract transition relation

\[(\text{if } [x > y]^1 \text{ then } \ldots, true) \implies ([\text{skip}]^8, \neg p_1) \implies \neg p_1\]

\[(\text{while } [\neg(y = 0)]^2 \text{ do } \ldots, p_1 \land p_2) \implies (\text{if } [x > y]^5 \text{ then } \ldots, p_1 \land p_2) \implies ([\text{skip}]^7, p_1 \land p_2 \land \neg p_1)\]

\[(\text{[x := x} - 1]^3; \ldots, p_1 \land p_2)\]

\[(\text{[y := y} - 1]^4; \ldots, p_2)\]

\[(\text{[skip]}^6, p_1 \land p_2)\]

\[(\text{[skip]}^7, p_1 \land p_2)\]

\[p_1 \land p_2\]

\[([\text{skip}]^7, q)\]

The only abstract configuration of the form \([\text{skip}]^7, q\) reachable from the configuration \((c, true)\) has \(q \models false\).
Implementing post

Computing post$_{x:=a}(q)$ requires computing $sp(q, x := a)$.

By definition

$$sp(q, x := a) \models \bigwedge \{ l \in (P \cup \neg P) \mid sp(q, x := a) \models l \}$$

Recall that $sp(q, x := a)$ contains an existential quantifier.

Solution: Reduce to checking the validity of an implication containing $wp$ by using the relationship between $sp$ and $wp$.

Theorem (Dijkstra’76) For every pair of predicates $b$ and $p$

$$sp(b, x := a) \models p \quad \text{iff} \quad b \models wp(x := a, p)$$

Theorem (Graf and Saidi’97)

$$sp(q, x := a) \models \bigwedge \{ l \in (P \cup \neg P) \mid q \rightarrow wp(x := a, l) \}$$
Implementing post

\[ \overline{sp(q, x := a)} \models \bigwedge \{ l \in (P \cup \neg P) \mid \models q \rightarrow wp(x := a, l) \} \]

Proof

\[ \overline{sp(q, x := a)} \models \bigwedge \{ l \in (P \cup \neg P) \mid sp(q, x := a) \models l \} \]
\[ \models \bigwedge \{ l \in (P \cup \neg P) \mid q \models wp(x := a, l) \} \] (Dijkstra)
\[ \models \bigwedge \{ l \in (P \cup \neg P) \mid \models q \rightarrow wp(x := a, l) \} \]
Implementing post

Checking if an implication \( q \rightarrow wp(x := a, l) \) is valid is usually done using an SMT (Satisfiability Modulo Theory) solver (Z3, CVC4, ...).

Checking validity of first-order logical formulas is in general undecidable. Termination of the solver is guaranteed for formulas in decidable fragments of first order logic.

<table>
<thead>
<tr>
<th>theory</th>
<th>description</th>
<th>full</th>
<th>quantifier-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_E )</td>
<td>theory of equality</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td>( T_{PA} )</td>
<td>Peano arithmetic (( \mathbb{N} ) with + and ( \cdot ))</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>( T_N )</td>
<td>Presburger arithmetic (( \mathbb{N} ) with +)</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>( T_Z )</td>
<td>( \mathbb{Z} ) with +</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>( T_Q )</td>
<td>( \mathbb{Q} ) with + and ( \cdot )</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>( T_R )</td>
<td>( \mathbb{R} ) with + and ( \cdot )</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>recursive data structures</td>
<td>☒</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>acyclic recursive data structures</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td></td>
<td>arrays</td>
<td>☒</td>
<td>☑</td>
</tr>
</tbody>
</table>
CEGAR

program $c$, property $\varphi$

compute $c_{Abs}$

$c_{Abs} \models \varphi$?

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 $r_{Abs} \not\models \varphi$
Verification of safety properties

**Safety property:** "Something bad never happens"

Examples:

- no division by 0
- variable $x$ never becomes negative
- a given program location is not reachable

Formally: given a program $c$ and a program $c_{bad}$ that corresponds to a location in the program $c$, we want to prove that for any states $\sigma$ and $\sigma'$, the configuration $(c_{bad}, \sigma')$ is not reachable from the configuration $(c, \sigma)$ in the concrete semantics $\rightarrow$ of $c$

Notation: the relation $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$, that is, $(c', \sigma') \rightarrow^* (c'', \sigma'')$ if and only if there exist $(c_0, \sigma_0), \ldots, (c_k, \sigma_k)$ where $(c_0, \sigma_0) = (c', \sigma')$, $(c_k, \sigma_k) = (c'', \sigma'')$ and $(c_i, \sigma_i) \rightarrow (c_{i+1}, c_{i+1})$ for all $0 \leq i < k$. 
Example: safety property

\[ \text{if } [x > y] \text{ then } \{ \]
\[ \text{while } [\neg(y = 0)] \text{ do } \{ \]
\[ [x := x - 1]; \]
\[ [y := y - 1] \]
\[ \} \]
\[ \text{if } [x > y] \text{ then } \{ \]
\[ [\text{skip}] \]
\[ \} \]
\[ \text{else } \{ \]
\[ [\text{skip}] \]
\[ \} \]
\[ \} \]
\[ \text{else } \{ \]
\[ [\text{skip}] \]
\[ \} \]

Safety property: location 7 is never reached.
Soundness of predicate abstraction

The family of functions \( \text{post}_c^\# \) and \( \text{post}_{c,c'}^\# \) is the most precise abstract semantics, that is, for all \( c, c' \in \text{Progs} \) and \( q \in \text{Cubes}(P) \)

\[
\begin{align*}
\alpha(\text{post}_c(\gamma(q))) & \models \text{post}_c^\#(q) \\
\alpha(\text{post}_{c,c'}(\gamma(q))) & \models \text{post}_{c,c'}^\#(q)
\end{align*}
\]

Proof We give the proof only for the case when \( c \) is \( x := a \). The other cases are similar/easy.

We show \( \alpha(\text{post}_{x:=a}(\gamma(q))) \models \text{post}_{x:=a}^\#(q) \).

\[
\begin{align*}
\alpha(\text{post}_{x:=a}(\gamma(q))) \\
= \alpha(\{\sigma \in \text{States} \mid \sigma \models \text{sp}(q, x := a)\}) \quad \text{( to show)} \\
= \bigsqcup\{q_{\sigma} \mid \sigma \in \text{States}, \sigma \models \text{sp}(q, x := a)\} \quad \text{( definition of } \alpha) \\
\models \text{sp}(q, x := a) \quad \text{( to show)} \\
\models \text{post}_{x:=a}^\#(q) \quad \text{( definition of } \Rightarrow) 
\end{align*}
\]
Lemma

$$\text{post}_{x:=a}(\gamma(q)) = \{ \sigma \in \text{States} \mid \sigma \models sp(q, x := a) \}$$

Intuition: The formula $sp(q, x := a)$ characterizes the successors of the states in $\gamma(q)$ with respect to the program $x := a$.

Proof

$$\sigma \in \text{post}_{x:=a}(\gamma(q)) \iff \exists \sigma' \in \text{States} : \sigma' \in \gamma(q) \text{ and } (x := a, \sigma') \rightarrow \sigma \iff \exists \sigma' \in \text{States} : \sigma' \models q \text{ and } (x := a, \sigma') \rightarrow \sigma \iff \sigma \models sp(q, x := a)$$

What remains to prove is the last equivalence.
1. Suppose that for some $\sigma' \in States$, $\sigma' \models q$ and $(x := a, \sigma') \rightarrow \sigma$.

Since we have that

- $\{q\}x := a\{sp(q, x := a)\}$ is a valid Hoare triple,
- $\sigma' \models q$,
- $(x := a, \sigma') \rightarrow \sigma$,

then by the definition of valid Hoare triple, it holds that $\sigma \models sp(q, x := a)$. 
Proof of Lemma continued

2. Suppose that $\sigma \models sp(q, x := a)$. This means that

$$\sigma \models \exists x'(q[x'/x] \land x = (a[x'/x])).$$

We will now define a state $\sigma'$ and show that it satisfies:

a) $\sigma' \models q$,

b) $(x := a, \sigma') \rightarrow \sigma$.

By the choice of $\sigma$, there exists $d \in \mathbb{Z}$ such that

$$\sigma[x' := d] \models q[x'/x] \quad (\ast)$$

$$\sigma[x' := d] \models x = (a[x'/x]) \quad (\ast\ast)$$

where $\sigma[x' := d] : (\text{Vars} \cup \{x'\}) \rightarrow \mathbb{Z}$ extends $\sigma$ by mapping $x'$ to $d$.

Let us define $\sigma' = \sigma[x \mapsto d]$. We have to show a) and b).
Proof of Lemma continued

a) We have

\[ \sigma[x' := d] \models q[x'/x] \quad (\ast) \]

This implies \( \sigma[x \mapsto d] \models q \), which is \( \sigma' \models q \).

b) By the definition of the concrete transition relation \( \rightarrow \) we have

\[(x := a, \sigma') \rightarrow \sigma'[x \mapsto A[a](\sigma')].\]

From

\[ \sigma[x' := d] \models x = (a[x'/x]) \quad (\ast\ast) \]

we get that

\[
\begin{align*}
\sigma(x) & = (\sigma[x' := d])(x) \\
& = A[a[x'/x]](\sigma[x' := d]) \quad \text{by (\ast\ast)} \\
& = A[a](\sigma[x \mapsto d]) \quad \text{since } x' \not\in \text{Vars}(a) \\
& = A[a](\sigma')
\end{align*}
\]

Thus, \((x := a, \sigma') \rightarrow \sigma'[x \mapsto \sigma(x)], which is (x := a, \sigma') \rightarrow \sigma.\)
Theorem

\[ \operatorname{sp}(q, x := a) \models \bigsqcup \{ q_\sigma \mid \sigma \in \text{States} \land \sigma \models \operatorname{sp}(q, x := a) \} \]

We will show that for any formula \( b \) over \( \text{Vars} \)

\[ \overline{b} \models \bigsqcup \{ q_\sigma \mid \sigma \in \text{States} \land \sigma \models b \} \]

To this end, we show that for every formula \( b \) over \( \text{Vars} \) we have:

1) \( b \models \bigvee \{ q_\sigma \mid \sigma \models b \} \)
2) \( \bigvee \{ q_\sigma \mid \sigma \models b \} \models \overline{b} \)

Recall that: \( q_\sigma = \bigwedge \{ l \in (P \cup \neg P) \mid \sigma \models l \} \)
Proof of Theorem continued

1) Let $\sigma' \in \text{States}$ be such that $\sigma' \models b$. Thus, $q_{\sigma'} \subseteq \{q_{\sigma} \mid \sigma \models b\}$. Since $\sigma' \models q_{\sigma'}$, we can conclude that $\sigma' \models \bigvee \{q_{\sigma} \mid \sigma \models b\}$.

2) Let $\sigma' \in \text{States}$ be such that $\sigma' \models \bigvee \{q_{\sigma} \mid \sigma \models b\}$. We have to show that $\sigma' \models \overline{b}$, which is equivalent to showing that

$$\sigma' \models \bigwedge \{l \in (P \cup \neg P) \mid b \models l\}.$$ 

Take an arbitrary $l' \in (P \cup \neg P)$ with $b \models l'$. We will prove $\sigma' \models l'$.

Since $\sigma' \models \bigvee \{q_{\sigma} \mid \sigma \models b\}$, then there exists $\sigma \in \text{States}$ such that $\sigma' \models q_{\sigma}$ and $\sigma \models b$. From $\sigma \models b$ and $b \models l'$ we get $\sigma \models l'$. Thus,

$$l' \in \{l \in (P \cup \neg P) \mid \sigma \models l\}.$$ 

Since by definition we have

$$q_{\sigma} = \bigwedge \{l \in (P \cup \neg P) \mid \sigma \models l\}$$

and $\sigma' \models q_{\sigma}$, we have that in particular $\sigma' \models l'$. 
Proof of Theorem continued

Now we will use 1) and 2) to show that for any formula $b$ over $\text{Vars}$,

\[ \overline{b} \models \bigcup \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} . \]

- We will show that $\overline{b} \models \bigcup \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \}$.

\[
\begin{align*}
b & \models \bigvee \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} \quad \text{(by 1))} \\
\overline{b} & \models \bigvee \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} \quad \text{(homework)} \\
\overline{b} & \models \bigcup \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} \quad \text{(def. of \bigcup)}
\end{align*}
\]

- We will show that $\bigcup \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} \models \overline{b}$.

\[
\begin{align*}
\bigvee \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} & \models \overline{b} \quad \text{(by 2))} \\
\bigvee \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} & \models \overline{b} \quad \text{(homework)} \\
\bigvee \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} & \models \overline{b} \quad \text{(homework)} \\
\bigcup \{ q_\sigma \mid \sigma \in \text{States}, \sigma \models b \} & \models \overline{b} \quad \text{(def. of \bigcup)}
\end{align*}
\]
By establishing the soundness of predicate abstraction we get that: For any program $c$ and program $c_{bad}$ corresponding to a location in $c$, if $(c, true) \Rightarrow^* (c_{bad}, q)$ implies $q \models false$, then there do not exist $\sigma, \sigma' \in States$ such that $(c, \sigma) \rightarrow^* (c_{bad}, \sigma')$.

That is, if $c$ satisfies a safety property with respect to $\Rightarrow$, then $c$ satisfies this property in the concrete semantics $\rightarrow$. 