abstraction of \textit{post} by \textit{post}#

- instead of iteratively applying \textit{post}, use over-approximation \textit{post}# such that always

\[ \text{post}(\varphi, \rho) \models \text{post}#(\varphi, \rho) \]

- decompose computation of \textit{post}# into two steps: first, apply \textit{post} and then, over-approximate result
- define abstraction function \( \alpha \) such that always

\[ \varphi \models \alpha(\varphi) . \]

- for a given abstraction function \( \alpha \), define \textit{post}#:

\[ \text{post}#(\varphi, \rho) = \alpha(\text{post}(\varphi, \rho)) \]
abstraction of $\phi_{\text{reach}}$ by $\phi_{\text{reach}}^\#$

- instead of computing $\phi_{\text{reach}}$, compute over-approximation $\phi_{\text{reach}}^\#$ such that $\phi_{\text{reach}}^\# \supseteq \phi_{\text{reach}}$
- check whether $\phi_{\text{reach}}^\#$ contains any error states
  - if $\phi_{\text{reach}}^\# \land \phi_{\text{err}} \models false$
  - then $\phi_{\text{reach}}^\# \land \phi_{\text{err}} \models false$, i.e., program is safe
- compute $\phi_{\text{reach}}^\#$ by applying iteration

\[
\phi_{\text{reach}}^\# = \alpha(\phi_{\text{init}}) \lor \\
post^\#(\alpha(\phi_{\text{init}}), \rho_R) \lor \\
post^\#(post^\#(\alpha(\phi_{\text{init}}), \rho_R), \rho_R) \lor \ldots \\
= \bigvee_{i \geq 0}(post^\#)^i(\alpha(\phi_{\text{init}}), \rho_R)
\]

- consequence: $\phi_{\text{reach}} \models \phi_{\text{reach}}^\#$
predicate abstraction

- construct abstraction $\alpha(\varphi)$ using a given set of building blocks, so-called predicates
- predicate $= \text{formula over the program variables } V$
- fix finite set of predicates $Preds = \{p_1, \ldots, p_n\}$
- over-approximation of $\varphi$ by conjunction of predicates in $Preds$

$$\alpha(\varphi) = \bigwedge \{p \in Preds \mid \varphi \models p\}$$

- computation of $\alpha(\varphi)$ requires $n$ entailment checks ($n = \text{number of predicates}$)
example: compute $\alpha(at_\ell_2 \land y \geq z \land x + 1 \leq y)$

- $Preds = \{at_\ell_1, \ldots, at_\ell_5, y \geq z, x \geq y\}$

1. to compute $\alpha(\varphi)$, check logical consequence between $\varphi$ and each of the predicates:

<table>
<thead>
<tr>
<th></th>
<th>$y \geq z$</th>
<th>$x \geq y$</th>
<th>$at_\ell_1$</th>
<th>$at_\ell_2$</th>
<th>$at_\ell_3$</th>
<th>$at_\ell_4$</th>
<th>$at_\ell_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$at_\ell_2 \land$ y $\geq z \land$ x + 1 $\leq y$</td>
<td>$</td>
<td>$</td>
<td>$\not</td>
<td>$</td>
<td>$\not</td>
<td>$</td>
<td>$</td>
</tr>
</tbody>
</table>

2. result of abstraction = conjunction over entailed predicates

$$\alpha( at_\ell_2 \land y \geq z \land x + 1 \leq y ) = at_\ell_2 \land y \geq z$$
trivial abstraction $\alpha(\varphi) = true$

- result of applying predicate abstraction is $true$ if none of the predicates is entailed by $\varphi$ ("predicates are too specific")
  
  ... always the case if $Preds = \emptyset$
algorithm AbstReach

begin
\[\alpha := \lambda \varphi . \land \{ p \in \text{Preds} \mid \varphi \models p \}\]
\[\text{post}^\# := \lambda (\varphi, \rho) . \alpha(\text{post}(\varphi, \rho))\]
\[\text{ReachStates}^\# := \{ \alpha(\varphi_{\text{init}}) \}\]
\[\text{Parent} := \emptyset\]
\[\text{Worklist} := \text{ReachStates}^\#\]

while \text{Worklist} \neq \emptyset do
\[\varphi := \text{choose from Worklist}\]
\[\text{Worklist} := \text{Worklist} \setminus \{ \varphi \}\]
for each \(\rho \in \mathcal{R}\) do
\[\varphi' := \text{post}^\#(\varphi, \rho)\]
if \(\varphi' \not\in \text{ReachStates}^\#\) then
\[\text{ReachStates}^\# := \{ \varphi' \} \cup \text{ReachStates}^\#\]
\[\text{Parent} := \{ (\varphi, \rho, \varphi') \} \cup \text{Parent}\]
\[\text{Worklist} := \{ \varphi' \} \cup \text{Worklist}\]

return (\text{ReachStates}^\#, \text{Parent})
end
Abstract Reachability Graph

\[ \varphi_1 : \textit{at}_l \]
\[ \varphi_2 : \textit{at}_l \land y \geq z \]
\[ \varphi_3 : \textit{at}_l \land y \geq z \land x \geq y \]
\[ \varphi_4 : \textit{at}_l \land y \geq z \land x \geq y \]

\[ \rho_1 \]
\[ \rho_2 \]
\[ \rho_3 \]
\[ \rho_4 \]

\[ \varphi_1 = \alpha(\varphi_{\text{init}}) \]
\[ \varphi_2 = \textit{post}\#(\varphi_1, \rho_1) \]
\[ \textit{post}\#(\varphi_2, \rho_2) \models \varphi_2 \]
\[ \varphi_3 = \textit{post}\#(\varphi_2, \rho_3) \]
\[ \varphi_4 = \textit{post}\#(\varphi_3, \rho_4) \]

- \textit{Preds} = \{\textit{false}, \textit{at}_l, \ldots, \textit{at}_l, y \geq z, x \geq y\}
- \textit{nodes} \varphi_1, \ldots, \varphi_4 \in \textit{ReachStates}\#
- \textit{labeled edges} \in \textit{Parent}
- \textit{dotted edge} : entailment relation (here, \textit{post}\#(\varphi_2, \rho_2) \models \varphi_2)
example: predicate abstraction to compute $\varphi_{\text{reach}}$

- $\text{Preds} = \{\text{false}, \text{at}_1, \ldots, \text{at}_5, y \geq z, x \geq y\}$

- over-approximation of the set of initial states $\varphi_{\text{init}}$:
  \[ \varphi_1 = \alpha(\text{at}_1) = \text{at}_1 \]

- apply $\text{post}^\#$ on $\varphi_1$ wrt. each program transition:
  \[ \varphi_2 = \text{post}^\#(\varphi_1, \rho_1) = \alpha(\text{at}_2 \wedge y \geq z) = \text{at}_2 \wedge y \geq z \]

  \[ \text{post}^\#(\varphi_1, \rho_2) = \cdots = \text{post}^\#(\varphi_1, \rho_5) = \wedge\{\text{false}, \ldots\} = \text{false} \]
apply $post^\#$ to $\varphi_2 = (at_\ell_2 \land y \geq z)$

- application of $\rho_1$, $\rho_4$, and $\rho_5$ on $\varphi_2$ results in false (since $\rho_1$, $\rho_4$, and $\rho_5$ are applicable only if either $at_\ell_1$ or $at_\ell_3$ hold)
- for $\rho_2$ we obtain

$$post^\#(\varphi_2, \rho_2) = \alpha(at_\ell_2 \land y \geq z \land x \leq y) = at_\ell_2 \land y \geq z$$

result is $\varphi_2$ which is already in $ReachStates^\#$: nothing to do
- for $\rho_3$ we obtain

$$post^\#(\varphi_2, \rho_3) = \alpha(at_\ell_3 \land y \geq z \land x \geq y)$$

$$= at_\ell_3 \land y \geq z \land x \geq y$$

$$= \varphi_3$$

new node $\varphi_3$ in $ReachStates^\#$, new edge in $Parent$
apply $\text{post}^\#$ to $\varphi_3 = (at_\ell 3 \land y \geq z \land x \geq y)$

- application of $\rho_1$, $\rho_2$, and $\rho_3$ on $\varphi_3$ results in $false$
- for $\rho_4$ we obtain:

$$\text{post}^\#(\varphi_3, \rho_4) = \alpha(at_\ell 4 \land y \geq z \land x \geq y \land x \geq z)$$
$$= at_\ell 4 \land y \geq z \land x \geq y$$
$$= \varphi_4$$

new node $\varphi_4$ in $\text{ReachStates}^\#$, new edge in $\text{Parent}$

- for $\rho_5$ (assertion violation) we obtain:

$$\text{post}^\#(\varphi_3, \rho_5) = \alpha(at_\ell 5 \land y \geq z \land x \geq y \land x + 1 \leq z)$$
$$= false$$

- any further application of program transitions does not compute any additional reachable states
- thus, $\varphi^\#_{\text{reach}} = \varphi_1 \lor \ldots \lor \varphi_4$
- since $\varphi^\#_{\text{reach}} \land at_\ell 5 \models false$, the program is proven safe
abstraction $\alpha(\varphi)$

- monotonicity

\[ \varphi_1 \models \varphi_2 \implies \alpha(\varphi_1) \models \alpha(\varphi_2) \]

- idempotency

\[ \alpha(\alpha(\varphi_1)) = \alpha(\varphi_1) \]

- extensiveness

\[ \varphi_1 \models \alpha(\varphi_1) \]
Abstract reachability computation with $\text{Preds} = \{\text{false}, \text{at}_1, \ldots, \text{at}_5, y \geq z\}$

$\varphi_1 = \alpha(\varphi_{init})$
$\varphi_2 = \text{post}(\varphi_1, \rho_1)$
$\text{post}(\varphi_2, \rho_2) \models \varphi_2$
$\varphi_3 = \text{post}(\varphi_2, \rho_3)$
$\varphi_4 = \text{post}(\varphi_3, \rho_4)$
$\varphi_5 = \text{post}(\varphi_3, \rho_5)$
omitting just one predicate (in the example: \( x \geq y \)) may lead to an over-approximation \( \varphi_{\text{reach}}^{\#} \) such that

\[
\varphi_{\text{reach}}^{\#} \land \varphi_{\text{err}} \not\models false
\]

that is, \texttt{AbstReach} without the predicate \( x \geq y \) fails to prove safety
counterexample path

- *Parent* relation records sequence leading to $\varphi_5$
  - apply $\rho_1$ to $\varphi_1$ and obtain $\varphi_2$
  - apply $\rho_3$ to $\varphi_2$ and obtain $\varphi_3$
  - apply $\rho_5$ to $\varphi_3$ and obtain $\varphi_5$

- counterexample path:
  sequence of program transitions $\rho_1$, $\rho_3$, and $\rho_5$

- Using this path and the functions $\alpha$ and $\text{post}^#$ corresponding to the current set of predicates we obtain

\[ \varphi_5 = \text{post}^#(\text{post}^#(\text{post}^#(\alpha(\varphi_{\text{init}}), \rho_1), \rho_3), \rho_5) \]

that is, $\varphi_5$ is equal to the over-approximation of the post-condition computed along the counterexample path
analysis of counterexample path

- check if the counterexample path also leads to the error states when no over-approximation is applied

- compute

\[
\text{post}(\text{post}(\text{post}(\varphi_{\text{init}}, \rho_1), \rho_3), \rho_5) \\
= \text{post}(\text{post}(\text{at}_\ell 2 \land y \geq z, \rho_3), \rho_5) \\
= \text{post}(\text{at}_\ell 3 \land y \geq z \land x \geq y, \rho_5) \\
= false.
\]

- by executing the program transitions \( \rho_1, \rho_3, \) and \( \rho_5 \) is not possible to reach any error

- conclude that the over-approximation is too coarse when dealing with the above path
need for refinement of abstraction

- need a more precise over-approximation that will prevent \( \varphi_{reach}^\# \) from including error states
need for refinement of abstraction

- need a more precise over-approximation that will prevent $\varphi^{\#}_{reach}$ from including error states
- need a more precise over-approximation that will prevent $\alpha$ from including states that lead to error states along the path $\rho_1$, $\rho_3$, and $\rho_5$
need for refinement of abstraction

- need a more precise over-approximation that will prevent $\varphi^\#_{reach}$ from including error states
- need a more precise over-approximation that will prevent $\alpha$ from including states that lead to error states along the path $\rho_1$, $\rho_3$, and $\rho_5$
- need a refined abstraction function $\alpha$ and a corresponding $post^#$ such that the execution of $\text{ABSTREACH}$ along the counterexample path does not compute a set of states that contains some error states

\[ post^#(post^#(post^#(\alpha(\varphi_{init}), \rho_1), \rho_3), \rho_5) \land \varphi_{err} \models false. \]
over-approximation along counterexample path

- goal:

\[ \text{post}^\#(\text{post}^\#(\text{post}^\#(\alpha(\varphi_{\text{init}}), \rho_1), \rho_3), \rho_5) \land \varphi_{\text{err}} = \text{false} . \]

- define sets of states \( \psi_1, \ldots, \psi_4 \) such that

\[
\begin{align*}
\varphi_{\text{init}} &\models \psi_1 \\
\text{post}(\psi_1, \rho_1) &\models \psi_2 \\
\text{post}(\psi_2, \rho_3) &\models \psi_3 \\
\text{post}(\psi_3, \rho_5) &\models \psi_4 \\
\psi_4 \land \varphi_{\text{err}} &\models \text{false}
\end{align*}
\]

- thus, \( \psi_1, \ldots, \psi_4 \) guarantee that no error state can be reached

may approximate / still allow additional states

- example choice for \( \psi_1, \ldots, \psi_4 \)

<table>
<thead>
<tr>
<th>( \psi_1 )</th>
<th>( \psi_2 )</th>
<th>( \psi_3 )</th>
<th>( \psi_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{at}_-\ell_1 )</td>
<td>( \text{at}_-\ell_2 \land y \geq z )</td>
<td>( \text{at}_-\ell_3 \land x \geq z )</td>
<td>( \text{false} )</td>
</tr>
</tbody>
</table>
refinement of predicate abstraction

- given sets of states $\psi_1, \ldots, \psi_4$ such that
  
  $\varphi_{init} \models \psi_1$
  
  $post(\psi_1, \rho_1) \models \psi_2$
  
  $post(\psi_2, \rho_3) \models \psi_3$
  
  $post(\psi_3, \rho_5) \models \psi_4$
  
  $\psi_4 \land \varphi_{err} \models false$

- add $\psi_1, \ldots, \psi_4$ to the set of predicates $Preds$

- formal property (discussed later) guarantees:

  $\alpha(\varphi_{init}) \models \psi_1$
  
  $post^#(\psi_1, \rho_1) \models \psi_2$
  
  $post^#(\psi_2, \rho_3) \models \psi_3$
  
  $post^#(\psi_3, \rho_5) \models \psi_4$
  
  $\psi_4 \land \varphi_{err} \models false$

  proves: no error state reachable along path $\rho_1, \rho_3,$ and $\rho_5$
next ...

- approach for analysing counterexample computed by AbstReach
- algorithms MakePath, FeasiblePath, and RefinePath
function MakePath
input
\( \psi \) - reachable abstract state
\( Parent \) - predecessor relation
begin
1 \( path := \) empty sequence
2 \( \varphi' := \psi \)
3 while exist \( \varphi \) and \( \rho \) such that \((\varphi, \rho, \varphi') \in Parent\) do
4 \( path := \rho . path \)
5 \( \varphi' := \varphi \)
6 return \( path \)
end
path computation

- input: reachable abstract state $\psi$ + $Parent$ relation
- view $Parent$ as a tree where $\psi$ occurs as a node
- output: sequence of program transitions that labels the tree edges on path from root to $\psi$
- sequence is constructed iteratively by a backward traversal starting from the input node
- variable $path$ keeps track of the construction
- in example, call $\text{MakePath}(\varphi_5, Parent)$
- $path$, initially empty, is extended with transitions $\rho_5$, $\rho_3$, $\rho_1$
- corresponding edges: $(\varphi_3, \rho_5, \varphi_5)$, $(\varphi_2, \rho_3, \varphi_3)$, $(\varphi_1, \rho_1, \varphi_1)$
- output: $path = \rho_1\rho_3\rho_5$
feasibility of a path

function $\text{FeasiblePath}$
input
$\rho_1 \ldots \rho_n$ - path
begin
1 $\varphi ::= \text{post}(\varphi_{\text{init}}, \rho_1 \circ \ldots \circ \rho_n)$
2 if $\varphi \land \varphi_{\text{err}} \not\models false$ then
3 return $true$
4 else
5 return $false$
end
feasibility of a path

- input: sequence of program transitions $\rho_1 \ldots \rho_n$
- checks if there is a computation that produced by this sequence
- check uses the post-condition function and the relational composition of transition
- apply $\text{FeasiblePath}$ on example path $\rho_1 \rho_3 \rho_5$
- relational composition of transitions yields

\[ \rho_1 \circ \rho_3 \circ \rho_5 = false. \]

- $\text{FeasiblePath}$ sets $\varphi$ to $false$ and then returns $false$
counterexample-guided discovery of predicates

function RefinePath
input
\[ \rho_1 \ldots \rho_n - \text{path} \]
begin
1 \[ \varphi_0, \ldots, \varphi_n := \text{compute such that} \]
2 \[ (\varphi_{init} \models \varphi_0) \land \]
3 \[ (post(\varphi_0, \rho_1) \models \varphi_1) \land \ldots \land (post(\varphi_{n-1}, \rho_n) \models \varphi_n) \land \]
4 \[ (\varphi_n \land \varphi_{err} \models false) \]
5 \[ \text{return} \{\varphi_0, \ldots, \varphi_n\} \]
end

\[\triangleright \text{omitted: particular algorithm for finding } \varphi_0, \ldots, \varphi_n\]
counterexample guided discovery of predicates

- input: sequence of program transitions $\rho_1 \ldots \rho_n$
- output: sets of states $\varphi_0, \ldots, \varphi_n$ such that
  - $\varphi_{init} \models \varphi_0$
  - $\text{post}(\varphi_{i-1}, \rho_i) \models \varphi_i$
  - $\varphi_n \land \varphi_{err} \models \text{false}$ for $i \in 1..n$
- if $\varphi_0, \ldots, \varphi_n$ are added to $\text{Preds}$
  then the resulting $\alpha$ and $\text{post}^\#$ guarantee that

$$
\begin{align*}
\alpha(\varphi_{init}) & \models \varphi_0 \\
\text{post}^\#(\varphi_0, \rho_1) & \models \varphi_1 \\
\ldots & \\
\text{post}^\#(\varphi_{n-1}, \rho_n) & \models \varphi_n \\
\varphi_n \land \varphi_{err} & \models \text{false}.
\end{align*}
$$

- in example, application of $\text{RefinePath}$ on $\rho_1 \rho_3 \rho_5$ yields sequence of sets of states $\psi_1, \ldots, \psi_4$
next ...

- algorithm for counterexample-guided abstraction refinement
- put together all building blocks into an algorithm AbstRefineLoop that verifies safety using predicate abstraction and counterexample guided refinement
function AbstRefineLoop
begin
1  $Preds := \emptyset$
2  repeat
3    $(\text{ReachStates}, \text{Parent}) := \text{AbstReach}(Preds)$
4    if exists $\psi \in \text{ReachStates}$ such that $\psi \land \varphi_{\text{err}} \neq \text{false}$
5      then
6        $path := \text{MakePath}(\psi, \text{Parent})$
7        if $\text{FeasiblePath}(path)$ then
8          return “counterexample path: path”
9        else
10       $Preds := \text{RefinePath}(path) \cup Preds$
11      else
12        return “program is correct”
end.
algorithm AbstRefineLoop

- input: program, output: proof or counterexample
- compute $\varphi^\#_{reach}$ using an abstraction defined wrt. set of predicates $Preds$ (initially empty)
- over-approximation $\varphi^\#_{reach}$: set of formulas $ReachStates^#$ where each formula represents a set of states
- if set of error states disjoint from over-approximation: stop
- otherwise, consider a formula $\psi$ in $ReachStates^#$ that witnesses overlap with error states
- refinement is only possible if overlap is caused by imprecision
- construct path, sequence of program transitions leading to $\psi$
- analyze path using FeasiblePath
- if path feasible: stop
- otherwise (path is not feasible), compute a set of predicates that refines the abstraction function
that’s it!