inductive invariants for example program

- weakest inductive invariant: \textit{true} (set of all states) contains error states
- strongest inductive invariant (does not contain error states)
  \[
  pc = l_1 \lor \\
  (pc = l_2 \land y \geq z) \lor \\
  (pc = l_3 \land y \geq z \land x \geq y) \lor \\
  (pc = l_4 \land y \geq z \land x \geq y)
  \]

- a slightly weaker inductive invariant also proves the safety of our examples:
  \[
  pc = l_1 \lor \\
  (pc = l_2 \land y \geq z) \lor \\
  (pc = l_3 \land y \geq z \land x \geq y) \lor \\
  pc = l_4
  \]

- can we drop another conjunct in one of the disjuncts?
1: assume(y >= z);
2: while (x < y) {
    x++;
}
3: assert(x >= z);
4: exit
5: error

inductive invariant (strict superset of reachable states):

\[ \varphi_{reach} = (pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_4) \]
fixpoint iteration

- computation of reachable program states = iterative application of post on initial program states until a fixpoint is reached
  i.e., no new program states are obtained by applying post
- in general, iteration process does not converge
  i.e., does not reach fixpoint in finite number of iterations
example: fixpoint iteration *diverges*

\[ \rho_2 \equiv (move(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land skip(y, z)) \]

\[ post(at \_ l_2 \land x \leq z, \rho_2) = (at \_ l_2 \land x - 1 \leq z \land x \leq y) \]
\[ post^2(at \_ l_2 \land x \leq z, \rho_2) = (at \_ l_2 \land x - 2 \leq z \land x \leq y) \]
\[ post^3(at \_ l_2 \land x \leq z, \rho_2) = (at \_ l_2 \land x - 3 \leq z \land x \leq y) \]
\[ \ldots \]
\[ post^n(at \_ l_2 \land x \leq z, \rho_2) = (at \_ l_2 \land x - n \leq z \land x \leq y) \]
example: fixpoint not reached after $n$ steps, $n \geq 1$

- set of states reachable after applying post twice not included in the union of previous two sets:

$$ \left( at_{-l2} \land x - 2 \leq z \land x \leq y \right) \not\subseteq \left( at_{-l2} \land x \leq z \lor at_{-l2} \land x - 1 \leq z \land x \leq y \right) $$

- set of states reachable after $n$-fold application of post still contains previously unreached states:

$$ \forall n \geq 1 : \left( at_{-l2} \land x - n \leq z \land x \leq y \right) \not\subseteq \left( at_{-l2} \land x \leq z \lor \lor_{1 \leq i < n} (at_{-l2} \land x - i \leq z \land x \leq y) \right) $$
abstraction of $\varphi_{reach}$ by $\varphi_{\text{reach}}^#$

- instead of computing $\varphi_{reach}$, compute over-approximation $\varphi_{\text{reach}}^#$ such that $\varphi_{\text{reach}}^# \supseteq \varphi_{\text{reach}}$
- check whether $\varphi_{\text{reach}}^#$ contains any error states
- if $\varphi_{\text{reach}}^# \land \varphi_{\text{err}} \models false$ holds then $\varphi_{\text{reach}} \land \varphi_{\text{err}} \models false$, and hence the program is safe
- compute $\varphi_{\text{reach}}^#$ by applying iteration
- instead of iteratively applying post, use over-approximation $post^#$ such that always

$$post(\varphi, \rho) \models post^#(\varphi, \rho)$$

- decompose computation of $post^#$ into two steps:
  first, apply post and then, over-approximate result using a function $\alpha$ such that

$$\forall \varphi : \varphi \models \alpha(\varphi) .$$
abstraction of post by post#

- given an abstraction function $\alpha$, define $\text{post}\#$:

$$\text{post}\#(\varphi, \rho) = \alpha(\text{post}(\varphi, \rho))$$

- compute $\varphi_{\text{reach}}$:

$$\varphi_{\text{reach}} = \alpha(\varphi_{\text{init}}) \lor$$

$$\text{post}\#(\alpha(\varphi_{\text{init}}), \rho_R) \lor$$

$$\text{post}\#(\text{post}\#(\alpha(\varphi_{\text{init}}), \rho_R), \rho_R) \lor \ldots$$

$$= \lor_{i \geq 0} (\text{post}\#)^i(\alpha(\varphi_{\text{init}}), \rho_R)$$

- consequence: $\varphi_{\text{reach}} \models \varphi_{\text{reach}}$
predicate abstraction

- construct abstraction 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 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. check logical consequence between argument to the abstraction function 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>$\models$</td>
<td>$\not\models$</td>
<td>$\not\models$</td>
<td>$\models$</td>
<td>$\not\models$</td>
<td>$\not\models$</td>
<td>$\not\models$</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
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")
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 $Preds = \emptyset$
example: predicate abstraction to compute $\varphi_{\text{reach}}$

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

- 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:

  $$\varphi_2 = \text{post}^\#(\varphi_1, \rho_1) = \alpha\left(\underbrace{\text{at}_2 \land y \geq z}_{\text{post}(\varphi_1, \rho_1)}\right) = \text{at}_2 \land y \geq z$$

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

- application of \(\rho_1, \rho_4, \text{and } \rho_5\) on \(\varphi_2\) results in \(false\) (since \(\rho_1, \rho_4, \text{and } \rho_5\) are applicable only if either \(\text{at}_\ell_1\) or \(\text{at}_\ell_3\) hold)

- for \(\rho_2\) we obtain

\[
post^\#(\varphi_2, \rho_2) = \alpha((\text{at}_\ell_2 \land y \geq z \land x \leq y) = \text{at}_\ell_2 \land y \geq z
\]

result is \(\varphi_2\) and, therefore, is discarded

- for \(\rho_3\) we obtain

\[
post^\#(\varphi_2, \rho_3) = \alpha((\text{at}_\ell_3 \land y \geq z \land x \geq y)
= \text{at}_\ell_3 \land y \geq z \land x \geq y
= \varphi_3
\]
apply \( \text{post}^\# \) to \( \varphi_3 = (at_\ell_3 \land y \geq z \land x \geq y) \)

- \( \rho_1, \rho_2, \) and \( \rho_3 \): inconsistency with program counter valuation in \( \varphi_3 \)
- for \( \rho_4 \) we obtain:

\[
\text{post}^\#(\varphi_3, \rho_4) = \alpha(at_\ell_4 \land y \geq z \land x \geq y \land x \geq z) \\
= at_\ell_4 \land y \geq z \land x \geq y \\
= \varphi_4
\]

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

\[
\text{post}^\#(\varphi_3, \rho_5) = \alpha(at_\ell_5 \land y \geq z \land x \geq y \land x + 1 \leq z) \\
= \text{false}
\]

- 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 \( \varphi_{\text{reach}}^\# \land at_\ell_5 \models \text{false} \), the program is proven safe
algorithm AbstReach

begin
\[ \alpha := \lambda \varphi . \bigwedge \{ p \in \text{Preds} \mid \varphi \vdash p \} \]
\[ \text{post}^\# := \lambda (\varphi, \rho) . \alpha(\text{post}(\varphi, \rho)) \]
\[ \text{ReachStates}^\# := \{ \alpha(\varphi_{\text{init}}) \} \]
\[ \text{Parent} := \emptyset \]
\[ \text{Worklist} := \text{ReachStates}^\# \]
while 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 \( \varphi' \not\vdash \bigvee \text{ReachStates}^\# \) then
\[ \text{ReachStates}^\# := \{ \varphi' \} \cup \text{ReachStates}^\# \]
\[ \text{Parent} := \{ (\varphi, \rho, \varphi') \} \cup \text{Parent} \]
\[ \text{Worklist} := \{ \varphi' \} \cup \text{Worklist} \]
return (ReachStates^\#, Parent)
end