Reachability Analysis

Andreas Podelski

December 6, 2011
relations as formulas

- formula with free variables in $V$ and $V' =$ binary relation over program states
  - first component of each pair assigns values to $V$
  - second component of the pair assigns values to $V'$
program $\mathbf{P} = (V, pc, \varphi_{init}, \mathcal{R}, \varphi_{err})$

- $V$ - finite tuple of *program variables*
- $pc$ - *program counter variable* (pc included in $V$)
- $\varphi_{init}$ - *initiation condition* given by formula over $V$
- $\mathcal{R}$ - a finite set of *transition relations*
- $\varphi_{err}$ - an *error condition* given by a formula over $V$

- transition relation $\rho \in \mathcal{R}$ given by formula over the variables $V$ and their primed versions $V'$
transition relation $\rho$ expressed by logica formula

$\rho_1 \equiv (move(\ell_1, \ell_2) \land y \geq z \land skip(x, y, z))$

$\rho_2 \equiv (move(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land skip(y, z))$

$\rho_3 \equiv (move(\ell_2, \ell_3) \land x \geq y \land skip(x, y, z))$

$\rho_4 \equiv (move(\ell_3, \ell_4) \land x \geq z \land skip(x, y, z))$

$\rho_5 \equiv (move(\ell_3, \ell_5) \land x + 1 \leq z \land skip(x, y, z))$

abbreviations:

$move(\ell, \ell') \equiv (pc = \ell \land pc' = \ell')$

$skip(v_1, \ldots, v_n) \equiv (v'_1 = v_1 \land \ldots \land v'_n = v_n)$
1: assume(y >= z);
2: while (x < y) {
    x++;
}
3: assert(x >= z);
4: exit
5: error

\[ \rho_1 = (move(\ell_1, \ell_2) \land y \geq z \land skip(x, y, z)) \]
\[ \rho_2 = (move(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land skip(y, z)) \]
\[ \rho_3 = (move(\ell_2, \ell_3) \land x \geq y \land skip(x, y, z)) \]
\[ \rho_4 = (move(\ell_3, \ell_4) \land x \geq z \land skip(x, y, z)) \]
\[ \rho_5 = (move(\ell_3, \ell_5) \land x + 1 \leq z \land skip(x, y, z)) \]
correctness: safety

- a state is *reachable* if it occurs in some program computation
- a program is *safe* if no error state is reachable
- ... if and only if no error state lies in $\varphi_{reach}$,

\[
\varphi_{err} \land \varphi_{reach} \models false.
\]

where $\varphi_{reach} =$ set of reachable program states
1: assume(y >= z);
2: while (x < y) {
    x++;
}
3: assert(x >= z);
4: exit
5: error

set of reachable states:

\[
\varphi_{reach} = (pc = \ell_1 \lor \\
               pc = \ell_2 \land y \geq z \lor \\
               pc = \ell_3 \land y \geq z \land x \geq y \lor \\
               pc = \ell_4 \land y \geq z \land x \geq y)
\]
post operator

- let $\varphi$ be a formula over $V$ and $\rho$ a formula over $V$ and $V'$
- define a post-condition function $\text{post}$ by:

$$\text{post}(\varphi, \rho) = (\exists V : \varphi \land \rho)[V/V']$$

an application $\text{post}(\varphi, \rho)$ computes the image of the set $\varphi$ under the relation $\rho$

- post distributes over disjunction wrt. each argument:

$$\text{post}(\varphi, \rho_1 \lor \rho_2) = (\text{post}(\varphi, \rho_1) \lor \text{post}(\varphi, \rho_2))$$

$$\text{post}(\varphi_1 \lor \varphi_2, \rho) = (\text{post}(\varphi_1, \rho) \lor \text{post}(\varphi_2, \rho))$$
application of $post(\phi, \rho)$ in examples

- $\rho$ has no primed variables

$\rho$ is an update of $x$ by an expression $e$ without $x$, say $\rho = x := e(y, z)$
application of $\text{post}(\phi, \rho)$ in examples

- $\rho$ has no primed variables
  \[ \text{post}(\phi, \rho) = \phi \land \rho \]
application of $post(\phi, \rho)$ in examples

- $\rho$ has no primed variables
  
  $post(\phi, \rho) = \phi \land \rho$

- $\rho$ has only primed variables
application of $\text{post}(\phi, \rho)$ in examples

- $\rho$ has no primed variables
  $\text{post}(\phi, \rho) = \phi \land \rho$

- $\rho$ has only primed variables
  $\text{post}(\phi, \rho) = \rho[V/V']$
application of $post(\phi, \rho)$ in examples

- $\rho$ has no primed variables
  
  $post(\phi, \rho) = \phi \land \rho$

- $\rho$ has only primed variables
  
  $post(\phi, \rho) = \rho[V/V']$

- $\rho$ is an update of $x$ by an expression $e$ without $x$, say
  
  $\rho = x := e(y, z)$
application of $\text{post}(\phi, \rho)$ in examples

- $\rho$ has no primed variables
  \[ \text{post}(\phi, \rho) = \phi \land \rho \]

- $\rho$ has only primed variables
  \[ \text{post}(\phi, \rho) = \rho[V/V'] \]

- $\rho$ is an update of $x$ by an expression $e$ without $x$, say
  \[ \rho = x := e(y, z) \]
  \[ \text{post}(\phi, \rho) = \exists x \phi \land x = e \]
iteration of post

$$post^n(\varphi, \rho) = n\text{-fold application of } post \text{ to } \varphi \text{ under } \rho$$

$$post^n(\varphi, \rho) = \begin{cases} 
\varphi & \text{if } n = 0 \\
post(post^{n-1}(\varphi, \rho), \rho) & \text{otherwise}
\end{cases}$$

characterize $$\varphi_{\text{reach}}$$ using iterates of $$post$$:

$$\varphi_{\text{reach}} = \varphi_{\text{init}} \lor post(\varphi_{\text{init}}, \rho_R) \lor post(post(\varphi_{\text{init}}, \rho_R), \rho_R) \lor \ldots$$

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

$$n\text{-th disjunct} = \text{iterate for natural number } n \text{ (disjunction } = \text{ “}\omega\text{ iteration”}$$
finite iteration post may suffice

“fixpoint reached in $n$ steps” if

\[
\bigvee_{i=0}^{n} \text{post}^i(\varphi_{\text{init}}, \rho_R) = \bigvee_{i=0}^{n+1} \text{post}^i(\varphi_{\text{init}}, \rho_R)
\]

then

\[
\bigvee_{i=0}^{n} \text{post}^i(\varphi_{\text{init}}, \rho_R) = \bigvee_{i\geq 0} \text{post}^i(\varphi_{\text{init}}, \rho_R)
\]
‘distributed’ iteration of $post(\cdot, \rho_R)$

- $\rho_R$ is itself a disjunction: $\rho_R = \rho_1 \lor \ldots \lor \rho_m$
- $post(\phi, \rho)$ distributes over disjunction in both arguments
- In ‘distributed’ disjunction $\Phi = \{\phi_k \mid k \in M\}$, every disjunct $\phi_k$ corresponds to a sequence of transitions $\rho_{j_1}, \ldots, \rho_{j_n}$

$$\phi_k = post(post(\ldots post(\varphi_{init}, \rho_{j_1}), \ldots), \rho_{j_n})$$

- $\phi_k \neq \emptyset$ only if sequence of transitions $\rho_{j_1}, \ldots, \rho_{j_n}$ corresponds to path in control flow graph of program since:

$$post(pc = \ell_i \land \ldots, move(\ell_j, \ell\ldots) \land \ldots) = \emptyset \text{ if } i \neq j$$

- Chaotic fixpoint iteration follows paths in control flow graph
‘distributed’ fixpoint test: ‘local’ entailment

- “fixpoint reached in \( n \) steps” if (but not only if):
  every application of \( \text{post}(\cdot, \cdot) \) to any disjunct \( \phi_k \) in \( \Phi \) is contained in one of the disjuncts \( \phi_{k'} \) in \( \Phi \) is

\[
\forall k \in M \ \forall j = 1, \ldots, m \ \exists k' \in M : \ \text{post}(\phi_k, \rho_j) \subseteq \phi_{k'}
\]
compute $\varphi_{reach}$ for example program (1)

apply post on set of initial states:

$$post(pc = \ell_1, \rho_R)$$
$$= post(pc = \ell_1, \rho_1)$$
$$= pc = \ell_2 \land y \geq z$$

apply post on successor states:

$$post(pc = \ell_2 \land y \geq z, \rho_R)$$
$$= post(pc = \ell_2 \land y \geq z, \rho_2) \lor post(pc = \ell_2 \land y \geq z, \rho_3)$$
$$= pc = \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x \geq y$$
compute $\varphi_{\text{reach}}$ for example program (2)

repeat the application step once again:

\[
p\text{ost}(pc = \ell_2 \land y \geq z \land x \leq y \lor \\
      pc = \ell_3 \land y \geq z \land x \geq y, \rho_R) \\
= \text{post}(pc = \ell_2 \land y \geq z \land x \leq y, \rho_R) \lor \\
\text{post}(pc = \ell_3 \land y \geq z \land x \geq y, \rho_R) \\
= \text{post}(pc = \ell_2 \land y \geq z \land x \leq y, \rho_2) \lor \\
\text{post}(pc = \ell_2 \land y \geq z \land x \leq y, \rho_3) \lor \\
\text{post}(pc = \ell_3 \land y \geq z \land x \geq y, \rho_4) \lor \\
\text{post}(pc = \ell_3 \land y \geq z \land x \geq y, \rho_5) \\
= pc = \ell_2 \land y \geq z \land x \leq y \lor \\
      pc = \ell_3 \land y \geq z \land x = y \lor \\
      pc = \ell_4 \land y \geq z \land x \geq y
\]
compute $\varphi_{\text{reach}}$ for example program

disjunction obtained by iteratively applying post to $\varphi_{\text{init}}$:

\[ pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_2 \land y \geq z \land x \leq y \lor pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_2 \land y \geq z \land x \leq y \lor pc = l_3 \land y \geq z \land x = y \lor \]
\[ pc = l_4 \land y \geq z \land x \geq y \]

disjunction in a logically equivalent, simplified form:

\[ pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_4 \land y \geq z \land x \geq y \]

above disjunction = $\varphi_{\text{reach}}$ since any further application of post
does not produce any additional disjuncts
checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant $\varphi$
checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant $\varphi$
- inductive:

\[ \varphi_{init} \models \varphi \quad \text{and} \quad post(\varphi, \rho_R) \models \varphi . \]
checking safety $\Rightarrow$ finding safe inductive invariant

- program is safe if there exists a safe inductive invariant $\varphi$
- inductive:

\[
\varphi_{init} \models \varphi \quad \text{and} \quad \text{post}(\varphi, \rho_R) \models \varphi .
\]

- safe:

\[
\varphi \land \varphi_{err} \models false
\]
checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant $\varphi$

- inductive:

$$\varphi_{init} \models \varphi \quad \text{and} \quad post(\varphi, \rho_R) \models \varphi.$$ 

- safe:

$$\varphi \land \varphi_{err} \models false$$

- justification:

1. "$\varphi_{reach}$ is the strongest inductive invariant"

$$\varphi_{reach} \models \varphi$$

2. program safe if $\varphi_{reach}$ does not contain an error state:

$$\varphi_{reach} \land \varphi_{err} \models false$$
inductive invariants for example program

- weakest inductive invariant:

  \[
  \text{true} \quad (\text{set of all states})
  \]

  \[
  \text{contains error states}
  \]

  strongest inductive invariant (does not contain error states)

  \[
  \text{pc} = \ell_1 \lor \left( \text{pc} = \ell_2 \land y \geq z \right) \lor \left( \text{pc} = \ell_3 \land y \geq z \land x \geq y \right) \lor \left( \text{pc} = \ell_4 \land y \geq z \land x \geq y \right)
  \]

- a slightly weaker inductive invariant also proves the safety of our examples:

  \[
  \text{pc} = \ell_1 \lor \left( \text{pc} = \ell_2 \land y \geq z \right) \lor \left( \text{pc} = \ell_3 \land y \geq z \land x \geq y \right) \lor \text{pc} = \ell_4
  \]

- can we drop another conjunct in one of the disjuncts?
inductive invariants for example program

- weakest inductive invariant: true (set of all states) contains error states
- strongest inductive invariant (does not contain error states)

\[ pc = \ell_1 \lor (pc = \ell_2 \land y \geq z) \lor (pc = \ell_3 \land y \geq z \land x \geq y) \lor (pc = \ell_4 \land y \geq z \land x \geq y) \]

- can we drop another conjunct in one of the disjuncts?
inductive invariants for example program

- weakest inductive invariant: true (set of all states) contains error states
- strongest inductive invariant (does not contain error states)

\[ pc = \ell_1 \lor \]
\[ (pc = \ell_2 \land y \geq z) \lor \]
\[ (pc = \ell_3 \land y \geq z \land x \geq y) \lor \]
\[ (pc = \ell_4 \land y \geq z \land x \geq y) \]

- a slightly weaker inductive invariant also proves the safety of our examples:

\[ pc = \ell_1 \lor \]
\[ (pc = \ell_2 \land y \geq z) \lor \]
\[ (pc = \ell_3 \land y \geq z \land x \geq y) \lor \]
\[ pc = \ell_4 \]
inductive invariants for example program

- weakest inductive invariant:  *true* (set of all states) contains error states
- strongest inductive invariant (does not contain error states)

\[ pc = \ell_1 \lor (pc = \ell_2 \land y \geq z) \lor (pc = \ell_3 \land y \geq z \land x \geq y) \lor (pc = \ell_4 \land y \geq z \land x \geq y) \]

- a slightly weaker inductive invariant also proves the safety of our examples:

\[ pc = \ell_1 \lor (pc = \ell_2 \land y \geq z) \lor (pc = \ell_3 \land y \geq z \land x \geq y) \lor pc = \ell_4 \]

- 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):

\[
\varphi_{reach} = (pc = \ell_1 \lor \\
   pc = \ell_2 \land y \geq z \lor \\
   pc = \ell_3 \land y \geq z \land x \geq y \lor \\
   pc = \ell_4)
\]
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*

\[ \rho_2 \equiv (\text{move}(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)) \]

\[
\text{post}(at_{-} \ell_2 \land x \leq z, \rho_2) = (at_{-} \ell_2 \land x - 1 \leq z \land x \leq y)
\]

\[
\text{post}^2(at_{-} \ell_2 \land x \leq z, \rho_2) = (at_{-} \ell_2 \land x - 2 \leq z \land x \leq y)
\]

\[
\text{post}^3(at_{-} \ell_2 \land x \leq z, \rho_2) = (at_{-} \ell_2 \land x - 3 \leq z \land x \leq y)
\]

\[\ldots\]

\[
\text{post}^n(at_{-} \ell_2 \land x \leq z, \rho_2) = (at_{-} \ell_2 \land x - n \leq z \land x \leq y)
\]
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:

$$(\text{at}_{-} \ell_2 \land x - 2 \leq z \land x \leq y) \neq \text{at}_{-} \ell_2 \land x \leq z \lor \text{at}_{-} \ell_2 \land x - 1 \leq z \land x \leq y$$

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

$$\forall n \geq 1 : (\text{at}_{-} \ell_2 \land x - n \leq z \land x \leq y) \neq \text{at}_{-} \ell_2 \land x \leq z \lor \mathop{\lor}_{1 \leq i < n}(\text{at}_{-} \ell_2 \land x - i \leq z \land x \leq y)$$
abstraction of $\varphi_{reach}$ by $\varphi_{reach}^#$

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

\begin{itemize}
  \item given an abstraction function $\alpha$, define \textit{post}^#:
    \[
    \textit{post}^#(\varphi, \rho) = \alpha(\textit{post}(\varphi, \rho))
    \]
  \item compute $\varphi^\#_{\text{reach}}$:
    \[
    \varphi^\#_{\text{reach}} = \alpha(\varphi_{\text{init}}) \vee
    \]
    \[
    \textit{post}^#(\alpha(\varphi_{\text{init}}), \rho_{\mathcal{R}}) \vee
    \]
    \[
    \textit{post}^#(\textit{post}^#(\alpha(\varphi_{\text{init}}), \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \vee \ldots
    \]
    \[
    = \bigvee_{i \geq 0} (\textit{post}^#)^i(\alpha(\varphi_{\text{init}}), \rho_{\mathcal{R}})
    \]
  \item consequence: $\varphi_{\text{reach}} \models \varphi^\#_{\text{reach}}$
\end{itemize}
predicate abstraction

- construct abstraction using a given set of building blocks, so-called predicates
- predicate = 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 $Preds$

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

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

$\triangleright$ $\text{Preds} = \{\text{at}_-\ell_1, \ldots, \text{at}_-\ell_5, y \geq z, x \geq y\}$

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>$\text{at}_-\ell_1$</th>
<th>$\text{at}_-\ell_2$</th>
<th>$\text{at}_-\ell_3$</th>
<th>$\text{at}_-\ell_4$</th>
<th>$\text{at}_-\ell_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{at}_-\ell_2 \land$ $y \geq z \land x + 1 \leq y$</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

$$\alpha(\begin{array}{l}
\text{at}_-\ell_2 \land \\
y \geq z \land x + 1 \leq y
\end{array}) = \text{at}_-\ell_2 \land y \geq z$$
trivial abstraction $\alpha(\varphi) = true$

- result of applying predicate abstraction is *true* if
trivial abstraction $\alpha(\varphi) = \text{true}$

- result of applying predicate abstraction is \text{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 $\varphi_{\text{reach}}$

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

over-approximation of the set of initial states $\varphi_{\text{init}}$:

$\varphi_1 = \alpha(at_{-\ell_1}) = at_{-\ell_1}$

apply $\text{post}^\#$ on $\varphi_1$ wrt. each program transition:

$\varphi_2 = \text{post}^\#(\varphi_1, \rho_1) = \alpha(at_{-\ell_2} \land y \geq z) = at_{-\ell_2} \land y \geq z$

$\text{post}^\#(\varphi_1, \rho_2) = \cdots = \text{post}^\#(\varphi_1, \rho_5) = \bigwedge\{\text{false}, \ldots\} = \text{false}$
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 $\varphi_2$ and, therefore, is discarded
- for $\rho_3$ we obtain

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

  $$= at\_\ell_3 \land y \geq z \land x \geq y$$

  $$= \varphi_3$$
apply \textit{post}^\# \text{ to } \varphi_3 = (\textit{at}_{\odot 3} \land y \geq z \land x \geq y)

\begin{itemize}
\item \(\rho_1, \rho_2, \text{ and } \rho_3\): inconsistency with program counter valuation in \(\varphi_3\)
\item for \(\rho_4\) we obtain:
\begin{align*}
\text{post}^\#(\varphi_3, \rho_4) &= \alpha(\textit{at}_{\odot 4} \land y \geq z \land x \geq y \land x \geq z) \\
&= \textit{at}_{\odot 4} \land y \geq z \land x \geq y \\
&= \varphi_4
\end{align*}
\item for \(\rho_5\) (assertion violation) we obtain:
\begin{align*}
\text{post}^\#(\varphi_3, \rho_5) &= \alpha(\textit{at}_{\odot 5} \land y \geq z \land x \geq y \land x + 1 \leq z) \\
&= false
\end{align*}
\item any further application of program transitions does not compute any additional reachable states
\item thus, \(\varphi^\#_{\text{reach}} = \varphi_1 \lor \ldots \lor \varphi_4\)
\item since \(\varphi^\#_{\text{reach}} \land \textit{at}_{\odot 5} \models false\), the program is proven safe
algorithm \texttt{AbstReach}

\begin{algorithmic}
\begin{itemize}
  \item \(\alpha := \lambda \varphi \cdot \bigwedge \{ p \in \text{_preds} \mid \varphi \models p \} \)
  \item \(\text{post}^\# := \lambda (\varphi, \rho) \cdot \alpha(\text{post}(\varphi, \rho))\)
  \item \(\text{ReachStates}^\# := \{ \alpha(\varphi_{\text{init}}) \}\)
  \item \(\text{Parent} := \emptyset\)
  \item \(\text{Worklist} := \text{ReachStates}^\#\)
  \item \textbf{while} \(\text{Worklist} \neq \emptyset\) \textbf{do}
    \begin{itemize}
      \item \(\varphi := \text{choose from Worklist}\)
      \item \(\text{Worklist} := \text{Worklist} \setminus \{ \varphi \}\)
      \item \textbf{for each} \(\rho \in \mathcal{R}\) \textbf{do}
        \begin{itemize}
          \item \(\varphi' := \text{post}^\#(\varphi, \rho)\)
          \item \textbf{if} \(\varphi' \not\models \bigvee \text{ReachStates}^\#\) \textbf{then}
            \begin{itemize}
              \item \(\text{ReachStates}^\# := \{ \varphi' \} \cup \text{ReachStates}^\#\)
              \item \(\text{Parent} := \{(\varphi, \rho, \varphi')\} \cup \text{Parent}\)
              \item \(\text{Worklist} := \{ \varphi' \} \cup \text{Worklist}\)
            \end{itemize}
        \end{itemize}
    \end{itemize}
  \item \textbf{return} \((\text{ReachStates}^\#, \text{Parent})\)
\end{itemize}
\end{algorithmic}