Reachability Analysis

Andreas Podelski

November 28, 2011

program with **assume** () and **assert** ()

- **assume** $(e) \equiv$ **if** e **then skip else halt**

program with **assume** () and **assert** ()

- **assume** $(e) \equiv$ **if** e **then skip else halt**
- **assert** $(e) \equiv$ **if** e **then skip else error**

program with **assume** () and **assert** ()

- **assume** $(e) \equiv$ **if** e **then skip else halt**
- **assert** $(e) \equiv$ **if** e **then skip else error**
- generalize *partial correctness*:

program with **assume** () and **assert** ()

- **assume** $(e) \;\equiv\;$ **if** e **then skip else halt**
- **assert** $(e) \;\equiv\;$ **if** e **then skip else error**
- generalize *partial correctness*:
  correctness of program wrt. Hoare triple:

$$\{\phi\}\ C\ \{\psi\}$$

  $\equiv$

program with **assume** () and **assert** ()

- **assume** (e) $\equiv$ **if** e **then skip else halt**
- **assert** (e) $\equiv$ **if** e **then skip else error**
- generalize *partial correctness*:
  correctness of program wrt. Hoare triple:

  $$\{\phi\}\ C\ \{\psi\}$$

  $\equiv$ *safety* of program: **assume** (ϕ) ; C ; **assert** (ψ)
- safety = non-reachability of **error**
  (no execution of **error** branch)

validity of Hoare triple:

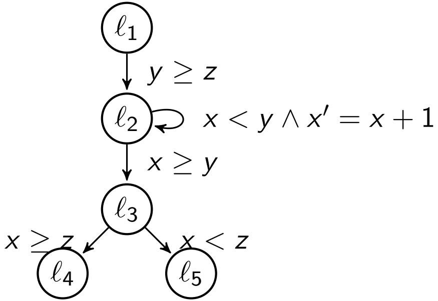
```
{y >= z}
while (x < y) {
  x++;
}
{x >= z}
```

$\equiv$ safety of program:

```
assume(y >= z);
while (x < y) {
  x++;
}
assert(x >= z);
```

.

```
1:  assume(y >= z);
2:  while (x < y) {
      x++;
    }
3:  assert(x >= z);
4: exit
5: error
```



.

$$\rho_1 = (move(\ell_1, \ell_2) \wedge y \geq z \wedge skip(x, y, z))$$
$$\rho_2 = (move(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$$
$$\rho_3 = (move(\ell_2, \ell_3) \wedge x \geq y \wedge skip(x, y, z))$$
$$\rho_4 = (move(\ell_3, \ell_4) \wedge x \geq z \wedge skip(x, y, z))$$
$$\rho_5 = (move(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge skip(x, y, z))$$

transition relation ρ expressed by logica formula

$$
\begin{aligned}
\rho_1 &\equiv \; (move(\ell_1, \ell_2) \wedge y \geq z \wedge skip(x, y, z)) \\
\rho_2 &\equiv \; (move(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z)) \\
\rho_3 &\equiv \; (move(\ell_2, \ell_3) \wedge x \geq y \wedge skip(x, y, z)) \\
\rho_4 &\equiv \; (move(\ell_3, \ell_4) \wedge x \geq z \wedge skip(x, y, z)) \\
\rho_5 &\equiv \; (move(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge skip(x, y, z))
\end{aligned}
$$

abbreviations:

$$
\begin{aligned}
move(\ell, \ell') &\equiv \; (pc = \ell \wedge pc' = \ell') \\
skip(v_1, \ldots, v_n) &\equiv \; (v_1' = v_1 \wedge \ldots \wedge v_n' = v_n)
\end{aligned}
$$

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)
- φ_{init} - *initiation condition* given by formula over V
- $\mathcal{R}$ - a finite set of *transition relations*
- φ_{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'

states, sets, and relations

- each program variable is assigned a *domain* of values

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* $=$ function that assigns each program variable a value from its respective domain

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* = function that assigns each program variable a value from its respective domain
- Σ = set of program states

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* = function that assigns each program variable a value from its respective domain
- Σ = set of program states
- formula with free variables in V = set of program states

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* $=$ function that assigns each program variable a value from its respective domain
- Σ $=$ set of program states
- formula with free variables in V $=$ set of program states
- 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'

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* $=$ function that assigns each program variable a value from its respective domain
- Σ $=$ set of program states
- formula with free variables in V $=$ set of program states
- 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'
- identify formulas with sets and relations that they represent

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* $=$ function that assigns each program variable a value from its respective domain
- Σ $=$ set of program states
- formula with free variables in V $=$ set of program states
- 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'
- identify formulas with sets and relations that they represent
- identify the logical consequence relation between formulas $\models$ with set inclusion $\subseteq$

states, sets, and relations

- each program variable is assigned a *domain* of values
- *program state* $=$ function that assigns each program variable a value from its respective domain
- Σ $=$ set of program states
- formula with free variables in V $=$ set of program states
- 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'
- identify formulas with sets and relations that they represent
- identify the logical consequence relation between formulas $\models$ with set inclusion $\subseteq$
- identify the satisfaction relation $\models$ between valuations and formulas, with the membership relation $\in$

example: states, sets, and relations

- formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z

example: states, sets, and relations

- formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z

- formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1

example: states, sets, and relations

- formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z

- formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1

- if program state s assigns 1, 3, 2, and ℓ_1
to program variables x, y, z, and pc, respectively,
then $s \models y \geq z$

example: states, sets, and relations

- formula $y \geq z$ = set of program states in which the value of the variable y is greater than the value of z

- formula $y' \geq z$ = binary relation over program states, = set of pairs of program states (s_1, s_2) in which the value of the variable y in the second state s_2 is greater than the value of z in the first state s_1

- if program state s assigns 1, 3, 2, and ℓ_1 to program variables x, y, z, and pc, respectively, then $s \models y \geq z$

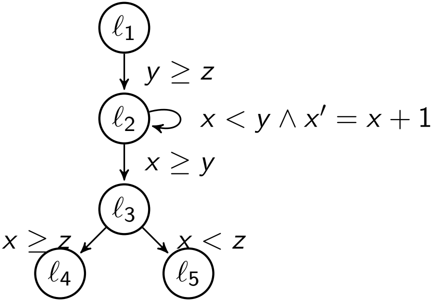
- logical consequence: $y \geq z \models y + 1 \geq z$

example program $\mathbf{P} = (V, pc, \varphi_{init}, \mathcal{R}, \varphi_{err})$

- program variables $V = (pc, x, y, z)$
- program counter pc
- program variables x, y, and z range over integers
- set of control locations $\mathcal{L} = \{\ell_1, \ldots \ell_5\}$
- initiation condition $\varphi_{init} = (pc = pc = \ell_1)$
- error condition $\varphi_{err} = (pc = pc = \ell_5)$
- program transitions $\mathcal{R} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$

$$
\begin{aligned}
\rho_1 &= (move(\ell_1, \ell_2) \wedge y \geq z \wedge skip(x, y, z)) \\
\rho_2 &= (move(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z)) \\
\rho_3 &= (move(\ell_2, \ell_3) \wedge x \geq y \wedge skip(x, y, z)) \\
\rho_4 &= (move(\ell_3, \ell_4) \wedge x \geq z \wedge skip(x, y, z)) \\
\rho_5 &= (move(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge skip(x, y, z))
\end{aligned}
$$

```
1:  assume(y >= z);
2:  while (x < y) {
      x++;
    }
3:  assert(x >= z);
4: exit
5: error
```



.

$$
\begin{aligned}
\rho_1 &= (\mathit{move}(\ell_1, \ell_2) \wedge y \geq z \wedge \mathit{skip}(x, y, z)) \\
\rho_2 &= (\mathit{move}(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge \mathit{skip}(y, z)) \\
\rho_3 &= (\mathit{move}(\ell_2, \ell_3) \wedge x \geq y \wedge \mathit{skip}(x, y, z)) \\
\rho_4 &= (\mathit{move}(\ell_3, \ell_4) \wedge x \geq z \wedge \mathit{skip}(x, y, z)) \\
\rho_5 &= (\mathit{move}(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge \mathit{skip}(x, y, z))
\end{aligned}
$$

initial state, error state, transition relation $\mathcal{R}$

- each state that satisfies the initiation condition φ_{init} is called an *initial* state
- each state that satisfies the error condition φ_{err} is called an *error* state
- program transition relation $\rho_\mathcal{R}$ is the union of the "single-statement" transition relations, i.e.,

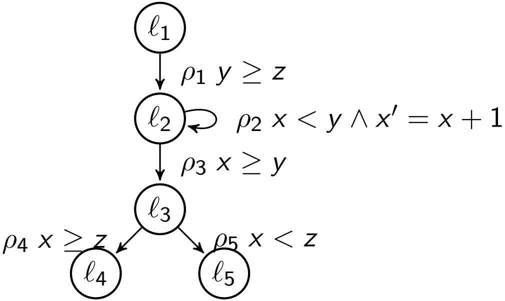
$$\rho_\mathcal{R} = \bigvee_{\rho \in \mathcal{R}} \rho \ .$$

- the state s has a transition to the state s' if the pair of states (s, s') lies in the program transition relation $\rho_\mathcal{R}$, i.e., if $(s, s') \models \rho_\mathcal{R}$

program computation $s_1, s_2, \ldots$

- the first element is an initial state, i.e., $s_1 \models \varphi_{init}$
- each pair of consecutive states (s_i, s_{i+1}) is connected by a program transition, i.e., $(s_i, s_{i+1}) \models \rho_{\mathcal{R}}$
- if the sequence is finite
  then the last element does not have any successors
  i.e., if the last element is s_n,
  then there is no state s such that $(s_n, s) \models \rho_{\mathcal{R}}$

```
1:   assume(y >= z);
2:   while (x < y) {
       x++;
     }
3:   assert(x >= z);
4:  exit
5:  error
```



.

example of a computation:

$$(\ell_1, 1, 3, 2),\ (\ell_2, 1, 3, 2),\ (\ell_2, 2, 3, 2),\ (\ell_2, 3, 3, 2),\ (\ell_3, 3, 3, 2),\ (\ell_4, 3, 3, 2)$$

- sequence of transitions $\rho_1, \rho_2, \rho_2, \rho_3, \rho_4$
- state $=$ tuple of values of program variables pc, x, y, and z
- last program state does not any successors

.

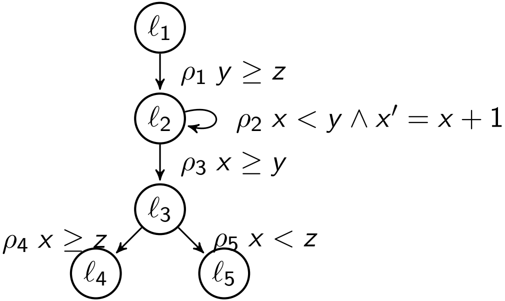
Correctness: Safety

- 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 φ_{reach},

$$\varphi_{err} \wedge \varphi_{reach} \models \mathit{false} \ .$$

where φ_{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:

$$
\begin{aligned}
\varphi_{reach} = (&pc = \ell_1 \vee \\
&pc = \ell_2 \wedge y \geq z \vee \\
&pc = \ell_3 \wedge y \geq z \wedge x \geq y \vee \\
&pc = \ell_4 \wedge y \geq z \wedge x \geq y)
\end{aligned}
$$

post operator

- let φ be a formula over V
- let ρ be a formula over V and V'
- define a *post-condition* function *post* by:

$$post(\varphi, \rho) \;=\; \exists V'' : \varphi[V''/V] \wedge \rho[V''/V][V/V']$$

  an application $post(\varphi, \rho)$ computes the image of the set φ under the relation ρ
- post distributes over disjunction wrt. each argument:

$$post(\varphi, \rho_1 \vee \rho_2) = (post(\varphi, \rho_1) \vee post(\varphi, \rho_2))$$
$$post(\varphi_1 \vee \varphi_2, \rho) = (post(\varphi_1, \rho) \vee post(\varphi_2, \rho))$$

application of $post(\phi, \rho)$ in example program

set of states $\phi \;\equiv\; pc = \ell_2 \wedge y \geq z,\quad$ transition relation $\rho \;\equiv\; \rho_2,$

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

$$\begin{aligned}
&post(\phi, \rho_2)\\
&\quad = (\exists V'' : (pc = \ell_2 \wedge y \geq z)[V''/V] \wedge \rho_2[V''/V][V/V'])\\
&\quad = (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge\\
&\qquad\qquad (pc'' = \ell_2 \wedge pc' = \ell_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge\\
&\qquad\qquad\; y' = y'' \wedge z' = z'')[V/V'])\\
&\quad = (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge\\
&\qquad\qquad (pc'' = \ell_2 \wedge pc = \ell_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge\\
&\qquad\qquad\; y = y'' \wedge z = z''))\\
&\quad = (pc = \ell_2 \wedge y \geq z \wedge x \leq y)
\end{aligned}$$

[renamed] program variables:
$V = (pc, x, y, z),\quad V' = (pc', x', y', z'),\quad V'' = (pc'', x'', y'', z'')$

iteration of post

$post^n(\varphi, \rho)$ = n-fold application of *post* to φ under ρ

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

characterize φ_{reach} using iterates of *post*:

$$\begin{aligned} \varphi_{reach} &= \varphi_{init} \lor post(\varphi_{init}, \rho_{\mathcal{R}}) \lor post(post(\varphi_{init}, \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \lor \ldots \\ &= \bigvee_{i \geq 0} post^i(\varphi_{init}, \rho_{\mathcal{R}}) \end{aligned}$$

disjuncts = iterates for every natural number n ("ω iteration")

finite iteration post may suffice

"fixpoint reached in n steps" if

if $\quad \bigvee_{i=1}^{n} post^i(\varphi_{init}, \rho_{\mathcal{R}}) = \bigvee_{i=1}^{n+1} post^i(\varphi_{init}, \rho_{\mathcal{R}})$

then $\quad \bigvee_{i=1}^{n} post^i(\varphi_{init}, \rho_{\mathcal{R}}) = \bigvee_{i\geq 0} post^i(\varphi_{init}, \rho_{\mathcal{R}})$

'distributed' fixpoint test

- $\rho_{\mathcal{R}}$ is itself a disjunction: $\rho_{\mathcal{R}} = \bigvee_{\rho \in \mathcal{R}} \rho \; = \{\rho_1, \ldots, \rho_m\}$
- $post(\phi, \rho)$ distributes over disjunction in both arguments
- in 'distributed' disjunction $\Phi = \{\phi_k \mid k \in M\}$, every disjunct ϕ_k corresponds to a sequence of transitions $\rho_{j_1}, \ldots, \rho_{j_n}$

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

- "fixpoint reached in n steps" if (but not only if):
  every application of $post(\cdot, \cdot)$ to any disjunct ϕ_k is contained in one of the disjuncts $\phi_{k'}$ in "big" disjunction

$$\forall k \in M \; \forall j = 1, \ldots, m \; \exists k' \in M : \; post(\phi_k, \rho_j) \subseteq \phi_{k'}$$

example iteration

- $$\begin{aligned} post(\varphi_{init}, \rho_1) &\equiv post(pc = \ell_1, \rho_1) \\ &\equiv pc = \ell_2 \wedge y \geq z \end{aligned}$$

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

- $post((pc = \ell_i), \rho_j) \equiv \emptyset$ if $\rho_j \wedge pc = \ell_i \equiv \emptyset$

loop applied to $post(\varphi_{init}, \rho_1)$

- $post(\varphi_{init}, \rho_1) \ \equiv \ (pc = \ell_2 \wedge y \geq z)$

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

$$
\begin{aligned}
& post(pc = \ell_2 \wedge y \geq z, \rho_2) \\
& \quad = (\exists V'' : (pc = \ell_2 \wedge y \geq z)[V''/V] \wedge \rho_2[V''/V][V/V']) \\
& \quad = (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge \\
& \qquad\qquad (pc'' = \ell_2 \wedge pc' = \ell_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge \\
& \qquad\qquad\ y' = y'' \wedge z' = z'')[V/V']) \\
& \quad = (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge \\
& \qquad\qquad (pc'' = \ell_2 \wedge pc = \ell_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge \\
& \qquad\qquad\ y = y'' \wedge z = z'')) \\
& \quad = (pc = \ell_2 \wedge y \geq z \wedge x \leq y)
\end{aligned}
$$

loop applied twice to $post(\varphi_{init}, \rho_1)$

$$
\begin{aligned}
& post^2(pc = \ell_2 \wedge y \geq z, \rho_2) \\
& \quad = post(post(pc = \ell_2 \wedge y \geq z, \rho_2), \rho_2) \\
& \quad = post(pc = \ell_2 \wedge y \geq z \wedge x \leq y, \rho_2) \\
& \quad = (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'' \wedge x'' \leq y'') \wedge \\
& \qquad\qquad\qquad (pc'' = \ell_2 \wedge pc = \ell_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge \\
& \qquad\qquad\qquad\; y = y'' \wedge z = z'')) \\
& \quad = (pc = \ell_2 \wedge y \geq z \wedge x - 1 \leq y \wedge x \leq y) \\
& \quad = (pc = \ell_2 \wedge y \geq z \wedge x \leq y)
\end{aligned}
$$

compute φ_{reach} for example program (1)

apply transition relation of the program once:

$$
\begin{aligned}
& post(pc = \ell_1, \rho_{\mathcal{R}}) \\
& \quad = (post(pc = \ell_1, \rho_1) \lor post(pc = \ell_1, \rho_2) \lor post(pc = \ell_1, \rho_3) \lor \\
& \qquad\quad post(pc = \ell_1, \rho_4) \lor post(pc = \ell_1, \rho_5)) \\
& \quad = post(pc = \ell_1, \rho_1) \\
& \quad = (pc = \ell_2 \land y \geq z)
\end{aligned}
$$

obtain the post-condition for one more application:

$$
\begin{aligned}
& post(pc = \ell_2 \land y \geq z, \rho_{\mathcal{R}}) \\
& \quad = (post(pc = \ell_2 \land y \geq z, \rho_2) \lor post(pc = \ell_2 \land y \geq z, \rho_3)) \\
& \quad = (pc = \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x \geq y)
\end{aligned}
$$

compute φ_{reach} for example program (2)

repeat the application step once again:

$$
\begin{aligned}
& post(pc = \ell_2 \wedge y \geq z \wedge x \leq y \vee \\
& \qquad pc = \ell_3 \wedge y \geq z \wedge x \geq y, \rho_{\mathcal{R}}) \\
& = (post(pc = \ell_2 \wedge y \geq z \wedge x \leq y, \rho_{\mathcal{R}}) \vee \\
& \quad post(pc = \ell_3 \wedge y \geq z \wedge x \geq y, \rho_{\mathcal{R}})) \\
& = (post(pc = \ell_2 \wedge y \geq z \wedge x \leq y, \rho_2) \vee \\
& \quad post(pc = \ell_2 \wedge y \geq z \wedge x \leq y, \rho_3) \vee \\
& \quad post(pc = \ell_3 \wedge y \geq z \wedge x \geq y, \rho_4) \vee \\
& \quad post(pc = \ell_3 \wedge y \geq z \wedge x \geq y, \rho_5)) \\
& = (pc = \ell_2 \wedge y \geq z \wedge x \leq y \vee \\
& \quad pc = \ell_3 \wedge y \geq z \wedge x = y \vee \\
& \quad pc = \ell_4 \wedge y \geq z \wedge x \geq y)
\end{aligned}
$$

compute φ_{reach} for example program

disjunction obtained by iteratively applying post to φ_{init}:

$$
\begin{aligned}
& pc = \ell_1 \lor \\
& pc = \ell_2 \land y \geq z \lor \\
& pc = \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x \geq y \lor \\
& pc = \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x = y \lor \\
& pc = \ell_4 \land y \geq z \land x \geq y
\end{aligned}
$$

disjunction in a logically equivalent, simplified form:

$$
\begin{aligned}
& pc = \ell_1 \lor \\
& pc = \ell_2 \land y \geq z \lor \\
& pc = \ell_3 \land y \geq z \land x \geq y \lor \\
& pc = \ell_4 \land y \geq z \land x \geq y
\end{aligned}
$$

above disjunction $= \varphi_{reach}$ since any further application of post does not produce any additional disjuncts