Model Checking

Lectures 4, 5, and 6

TUM

Reachability computation

Let φ be a formula over V and let ρ be a formula over V and V'. We define a *post-condition* function *post* as follows.

$$post(\varphi, \rho) \;=\; \exists V'' : \varphi[V''/V] \wedge \rho[V''/V][V/V'] \tag{1}$$

An application $post(\varphi, \rho)$ computes the image of the set φ under the relation ρ. Furthermore, for a natural number n we define $post^n(\varphi, \rho)$ as follows.

$$post^n(\varphi, \rho) \;=\; \begin{cases} \varphi & \text{if } n = 0 \\ post(post^{n-1}(\varphi, \rho), \rho) & \text{otherwise} \end{cases} \tag{2}$$

By $post^n(\varphi, \rho)$ we represent the n-fold application of the *post* function to φ with respect to ρ. We observe the following useful property of the post-condition function.

$$\begin{aligned} &\forall \varphi\ \forall \rho_1\ \forall \rho_2 : post(\varphi, \rho_1 \vee \rho_2) = (post(\varphi, \rho_1) \vee post(\varphi, \rho_2)) \\ &\forall \varphi_1\ \forall \varphi_2\ \forall \rho : post(\varphi_1 \vee \varphi_2, \rho) = (post(\varphi_1, \rho) \vee post(\varphi_2, \rho)) \end{aligned} \tag{3}$$

This property states that the post-condition computation distributes over disjunction wrt. each argument.

Example 1. For example, given the transition relation ρ_2 and the program variables $V = (pc, x, y, z)$ from our example program, we compute the following post condition.

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

We compute the 2-fold application by reusing the above result.

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

□

We characterize φ_{reach} using *post* as follows.

$$\begin{aligned}
\varphi_{reach} &= \varphi_{init} \vee post(\varphi_{init}, \rho_{\mathcal{R}}) \vee post(post(\varphi_{init}, \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \vee \ldots \\
&= \textstyle\bigvee_{i \geq 0} post^i(\varphi_{init}, \rho_{\mathcal{R}})
\end{aligned} \tag{4}$$

The above disjunction (over every number of applications of the post-condition function) ensures that all reachable states are taken into consideration.

Example 2. We compute φ_{reach} for our example program. We first obtain the post-condition after applying the transition relation of the program once.

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

Next, we obtain the post-condition for one more application.

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

We repeat the application step once again.

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

So far, by iteratively applying the post-condition function to φ_{init} we obtained the following disjunction.

$$\begin{aligned}
&at_\ell_1 \vee \\
&at_\ell_2 \wedge y \geq z \vee \\
&at_\ell_2 \wedge y \geq z \wedge x \leq y \vee at_\ell_3 \wedge y \geq z \wedge x \geq y \vee \\
&at_\ell_2 \wedge y \geq z \wedge x \leq y \vee at_\ell_3 \wedge y \geq z \wedge x = y \vee \\
&at_\ell_4 \wedge y \geq z \wedge x \geq y
\end{aligned}$$

We present this disjunction in a logically equivalent, simplified form as follows.

$$\begin{aligned}
&at_\ell_1 \vee \\
&at_\ell_2 \wedge y \geq z \vee \\
&at_\ell_3 \wedge y \geq z \wedge x \geq y \vee \\
&at_\ell_4 \wedge y \geq z \wedge x \geq y
\end{aligned}$$

Any further application of the post-condition function does not produce any additional disjuncts. Hence, φ_{reach} is the above disjunction. □

Inductive Safety Arguments

An *inductive invariant* φ contains the intial states and is closed under successors. Formally, an inductive invariant is a formula over the program variables that represents a superset of the initial program states and is closed under the application of the *post* function wrt. the relation $\rho_{\mathcal{R}}$, i.e.,

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

A program is safe if there exists an inductive invariant φ that does not contain any error states, i.e., $\varphi \wedge \varphi_{err} \models false$.

Example 3. For our example program, the weakest inductive invariant consists of the set of all states and is represented by the formula *true*. The strongest inductive invariant was obtained in Example 2 and is shown below.

$$at\_\ell_1 \vee (at\_\ell_2 \wedge y \geq z) \vee (at\_\ell_3 \wedge y \geq z \wedge x \geq y) \vee (at\_\ell_4 \wedge y \geq z \wedge x \geq y)$$

The strongest inductive invariant does not contain any error states. We observe that a slightly weaker inductive invariant below also proves the safety of our examples.

$$at\_\ell_1 \vee (at\_\ell_2 \wedge y \geq z) \vee (at\_\ell_3 \wedge y \geq z \wedge x \geq y) \vee at\_\ell_4$$

□

Computation of reachable program states requires iterative application of the post-condition function on the initial program states, see Equation (4). The iteration finishes when no new program states are discovered. Unfortunately, such an iteration process does not terminate in finite time.

Example 4. For example, we consider the iterative computation of the set of states that is reachable from $at\_\ell_2 \wedge x \leq z$ by applying the transition ρ_2 of our example program. We obtain the following sequence of post-conditions (where $V = (pc, x, y, z)$).

$$\begin{aligned}
post(at_\ell_2 \wedge x \leq z, \rho_2) &= (\exists V'' : (pc'' = \ell_2 \wedge x'' \leq z'') \wedge \\
&\qquad\qquad (pc'' = \ell_2 \wedge pc = \ell_2 \wedge x'' + 1 \leq y'' \wedge \\
&\qquad\qquad\ x = x'' + 1 \wedge y = y'' \wedge z = z'')) \\
&= (at_\ell_2 \wedge x - 1 \leq z \wedge x \leq y) \\
post^2(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - 2 \leq z \wedge x \leq y) \\
post^3(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - 3 \leq z \wedge x \leq y) \\
&\dots \\
post^n(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - n \leq z \wedge x \leq y)
\end{aligned}$$

In this sequence, we observe that at each iteration yields a set of states that contains states not discovered before. For example, the set of states reachable after applying the post-condition function once is not included in the original set, i.e.,

$$(at\_\ell_2 \wedge x - 1 \leq z \wedge x \leq y) \not\models (at\_\ell_2 \wedge x \leq z) \ .$$

The set of states reachable after applying the post-condition function twice is not included in the union of the above two sets, i.e.,

$$(at\_\ell_2 \wedge x - 2 \leq z \wedge x \leq y) \not\models (at\_\ell_2 \wedge x - 1 \leq z \wedge x \leq y \vee at\_\ell_2 \wedge x \leq z) \ .$$

Furthermore, we observe that the set of states reachable after n-fold application of *post*, where $n \geq 1$, still contains previously unreached states, i.e.,

$$\begin{aligned}
&\forall n \geq 1 : (at_\ell_2 \wedge x - n \leq z \wedge x \leq y) \\
&\qquad\qquad \not\models (at_\ell_2 \wedge x \leq z \vee \textstyle\bigvee_{1 \leq i < n} (at_\ell_2 \wedge x - i \leq z \wedge x \leq y)) \ .
\end{aligned}$$

□

Approximation

Instead of computing φ_{reach} we compute an over-approximation of φ_{reach} by a superset $\varphi^{\#}_{reach}$. Then, we check whether $\varphi^{\#}_{reach}$ contains any error states. If $\varphi^{\#}_{reach} \wedge \varphi_{err} \models false$ holds then $\varphi_{reach} \wedge \varphi_{err} \models false$. Hence the program is safe.

Similarly to the iterative computation of φ_{reach}, we compute $\varphi^{\#}_{reach}$ by applying iteration. However, instead of iteratively applying the post-condition function *post* we use its over-approximation $post^{\#}$ such that

$$\forall \varphi\ \forall \rho : post(\varphi, \rho) \models post^{\#}(\varphi, \rho)\ . \tag{5}$$

We decompose the computation of $post^{\#}$ into two steps. First, we apply *post* and then, we over-approximate the result using a function α such that

$$\forall \varphi : \varphi \models \alpha(\varphi)\ . \tag{6}$$

That is, given an over-approximating function α we define $post^{\#}$ as follows.

$$post^{\#}(\varphi, \rho) \ = \ \alpha(post(\varphi, \rho)) \tag{7}$$

Finally, we obtain $\varphi^{\#}_{reach}$:

$$\begin{aligned} \varphi^{\#}_{reach} &= \alpha(\varphi_{init}) \vee \\ &\quad post^{\#}(\alpha(\varphi_{init}), \rho_{\mathcal{R}}) \vee \\ &\quad post^{\#}(post^{\#}(\alpha(\varphi_{init}), \rho_{\mathcal{R}}), \rho_{\mathcal{R}}) \vee \dots \\ &= \textstyle\bigvee_{i \geq 0} (post^{\#})^{i}(\alpha(\varphi_{init}), \rho_{\mathcal{R}}) \end{aligned} \tag{8}$$

The following lemma formalizes our over-approximation based reachability computation.

Lemma 1. $\varphi_{reach} \models \varphi^{\#}_{reach}$

Predicate abstraction

We construct an over-approximation using a given set of building blocks, so-called predicates. Each predicate is a formula over the program variables V.

We fix a finite set of predicates $Preds = \{p_1, \dots, p_n\}$. Then, we define an over-approximation of φ that is represented using $Preds$ as follows.

$$\alpha(\varphi) = \bigwedge\{p \in Preds \mid \varphi \models p\} \tag{9}$$

Example 5. For example, we consider a set of predicates $Preds = \{at\_\ell_1, \dots, at\_\ell_5, y \geq z, x \geq y\}$. We compute $\alpha(at\_\ell_2 \wedge y \geq z \wedge x + 1 \leq y)$ as follows. First, we check the logical consequence between the argument to the abstraction function and each of the predicates. The results are presented in the following table.

	at_ℓ_1	at_ℓ_2	at_ℓ_3	at_ℓ_4	at_ℓ_5	$y \geq z$	$x \geq y$
$at\_\ell_2 \wedge y \geq z \wedge x + 1 \leq y$	$\not\models$	$\models$	$\not\models$	$\not\models$	$\not\models$	$\models$	$\not\models$

Then, we take the conjunction of the entailed predicates as the result of the abstraction.

$$\alpha(at\_\ell_2 \wedge y \geq z \wedge x + 1 \leq y) = \bigwedge\{at\_\ell_2, y \geq z\} = at\_\ell_2 \wedge y \geq z$$

If the set of predicates is empty then the result of applying predicate abstraction is *true*. For example, for $Preds = \emptyset$ we obtain

$$\alpha(at\_\ell_2 \wedge y \geq z \wedge x + 1 \leq y) = \bigwedge \emptyset = true \ .$$

If no predicates in *Preds* is entailed the resulting abstraction is *true* as well. For example, for $Preds = at\_1, \ldots, at\_3$ we have

$$\alpha(at\_\ell_5) = \bigwedge \emptyset = true \ .$$

□

The predicate abstraction function in Equation (9) approximates φ using a conjunction of predicates, which requires n entailment checks where n is the number of given predicates.

Example 6. We use predicate abstraction to compute $\varphi^{\#}_{reach}$ for our example program following the iterative scheme presented in Equation 8. Let $Preds = \{false, at\_\ell_1, \ldots, at\_\ell_5, y \geq z, x \geq y\}$. First, let φ_1 be the over-approximation of the set of initial states φ_{init}:

$$\varphi_1 = \alpha(at\_\ell_1) = \bigwedge\{at\_\ell_1\} = at\_\ell_1 \ .$$

We apply $post^{\#}$ on φ_1 wrt. each program transition and obtain

$$\varphi_2 = post^{\#}(\varphi_1, \rho_1) = \alpha(\underbrace{at\_\ell_2 \wedge y \geq z}_{post(\varphi_1, \rho_1)}) = \bigwedge\{at\_\ell_2, y \geq z\} = at\_\ell_2 \wedge y \geq z \ ,$$

whereas $post^{\#}(\varphi_1, \rho_2) = \cdots = post^{\#}(\varphi_1, \rho_5) = \bigwedge\{false, \ldots\} = false$.

Now we apply program transitions on φ_2 using $post^{\#}$. The application of ρ_1, ρ_4, and ρ_5 on φ_2 results in *false* for the following reason. φ_2 requires at_ℓ_2, but the transition relations ρ_1, ρ_4, and ρ_5 are applicable if either at_ℓ_1 or at_ℓ_3 holds. For ρ_2 we obtain

$$post^{\#}(\varphi_2, \rho_2) = \alpha(at\_\ell_2 \wedge y \geq z \wedge x \leq y) = \bigwedge\{at\_\ell_2, y \geq z\} = at\_\ell_2 \wedge y \geq z \ .$$

The resulting set above is equal to φ_2 and, therefore, is discarded, since we are already exploring states reachable from φ_2. For ρ_3 we obtain

$$\begin{aligned} post^{\#}(\varphi_2, \rho_3) &= \alpha(at\_\ell_3 \wedge y \geq z \wedge x \geq y) \\ &= \bigwedge\{at\_\ell_3, y \geq z, x \geq y\} = at\_\ell_3 \wedge y \geq z \wedge x \geq y \\ &= \varphi_3 \ . \end{aligned}$$

We compute an over-approximation of the set of states that are reachable from φ_3 by applying $post^{\#}$. The transitions ρ_1, ρ_2, and ρ_3 results in *false* due

function AbstReach
input
  Preds - predicates
begin
  $\alpha \ := \ \lambda\varphi \ . \ \bigwedge\{p \in \mathit{Preds} \mid \varphi \models p\}$
  $\mathit{post}^{\#} \ := \ \lambda(\varphi, \rho) \ . \ \alpha(\mathit{post}(\varphi, \rho))$
  $\mathit{ReachStates}^{\#} \ := \ \{\alpha(\varphi_{init})\}$
  $\mathit{Parent} \ := \ \emptyset$
  $\mathit{Worklist} \ := \ \mathit{ReachStates}^{\#}$
  while $\mathit{Worklist} \neq \emptyset$ **do**
    $\varphi \ :=$ choose from $\mathit{Worklist}$
    $\mathit{Worklist} \ := \ \mathit{Worklist} \setminus \{\varphi\}$
    for each $\rho \in \mathcal{R}$ **do**
      $\varphi' \ := \ \mathit{post}^{\#}(\varphi, \rho)$
      if $\varphi' \not\models \bigvee \mathit{ReachStates}^{\#}$ **then**
        $\mathit{ReachStates}^{\#} \ := \ \{\varphi'\} \cup \mathit{ReachStates}^{\#}$
        $\mathit{Parent} \ := \ \{(\varphi, \rho, \varphi')\} \cup \mathit{Parent}$
        $\mathit{Worklist} \ := \ \{\varphi'\} \cup \mathit{Worklist}$
  return $(\mathit{ReachStates}^{\#}, \mathit{Parent})$
end

Fig. 1. Algorithm AbstReach for abstract reachability computation wrt. a given finite set of predicates.

to an inconsistency caused by the program counter valuations in φ_3 and the respective transition relations. For the transition ρ_4 we obtain

$$\begin{aligned} \mathit{post}^{\#}(\varphi_3, \rho_4) &= \alpha(\mathit{at\_}\ell_4 \wedge y \geq z \wedge x \geq y \wedge x \geq z) \\ &= \bigwedge\{\mathit{at\_}\ell_4, y \geq z, x \geq y\} = \mathit{at\_}\ell_4 \wedge y \geq z \wedge x \geq y \\ &= \varphi_4 \ . \end{aligned}$$

For the transition ρ_5, which corresponds to the assertion violation, we obtain

$$\begin{aligned} \mathit{post}^{\#}(\varphi_3, \rho_5) &= \alpha(\mathit{at\_}\ell_5 \wedge y \geq z \wedge x \geq y \wedge x + 1 \leq z) \\ &= \mathit{false} \ . \end{aligned}$$

Any further application of program transitions does not compute any additional reachable states. We conclude that $\varphi^{\#}_{reach} = \varphi_1 \vee \ldots \vee \varphi_4$. Furthermore, since $\varphi^{\#}_{reach} \wedge \mathit{at\_}\ell_5 \models \mathit{false}$ the program is safe. $\square$

Algorithm AbstReach

We combine the characterization of abstract reachability using Equation (8) with the predicat abstraction function given in Equation (9) and obtain an algorithm AbstReach for computing $ReachStates^{\#}$. The algorithm is shown in Figure 1.

AbstReach takes as input a finite set of predicates *Preds* and computes a set of formulas $ReachStates^{\#}$ that represents an over-approximation $\varphi^{\#}_{reach}$. Furthermore, AbstReach records its intermediate computation steps in a labeled tree *Parent*. (In the next section we will show how this tree can be used to discover new predicates when a refined abstraction is needed.)

The initialization steps of AbstReach are shown in lines 1–5 of Figure 1. First, we construct the abstraction function α according to Equation (9), and then use it to construct an over-approximation $post^{\#}$ of the post-condition function according to Equation (7). We initialize $ReachStates^{\#}$ with an over-approximation of the initial program states, which corresponds to the first disjunct in Equation (8). Since the initial states do not have any predecessors, *Parent* is initially empty. Finally, we create a worklist *Worklist* that contains sets of states on which $post^{\#}$ has not been applied yet.

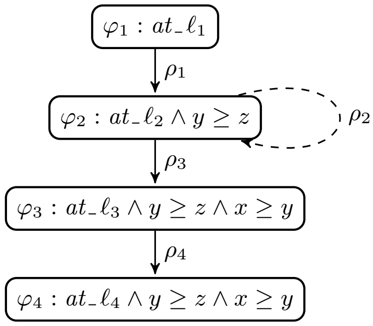
The main part of AbstReach in lines 6–14 implements the iterative application of $post^{\#}$ in Equation (8) using a while loop. The loop termination condition checks if *Worklist* has any items to process. In case the worklist is not empty, we choose such an item, say φ, and remove it from the worklist. For brevity, we leave the selection procedure unspecified, but note that various strategies are possible, e.g., breadth- or depth-first search. Then, we apply $post^{\#}$ wrt. each program transition, say ρ, on φ. Let φ' be the result of such an application. We add φ' to $ReachStates^{\#}$ if φ' contains some program states that are not already contained in one of the formulas in $ReachStates^{\#}$. We formulate the above test as an entailment check between φ' and the disjunction of all formulas in $ReachStates^{\#}$. Often, there is a formula ψ in $ReachStates^{\#}$ such that $\varphi' \models \psi$. In case that φ is added to $ReachStates^{\#}$, we record that φ was computed by applying ρ on φ by adding a tuple $(\varphi, \rho, \varphi')$ to *Parent*. Finally, φ is put on the worklist.

The loop execution terminates after a finite number of steps, since the range of $post^{\#}$ is finite (and is of size 2^n where n is the size of *Preds*). The disjunction of formulas in $ReachStates^{\#}$ is logically equivalent to $\varphi^{\#}_{reach}$.

Example 7. We describe the application of AbstReach on our example program when $Preds = \{false, at\_\ell_1, \ldots, at\_\ell_5, y \geq z, x \geq y\}$. Figure 2 provides a pictorial illustration. Example 6 provides details on computed over-approximations of post-conditions.

After constructing α and $post^{\#}$ for the given predicates, we compute $\varphi_1 = (at\_\ell_1)$ and put it into $ReachStates^{\#}$ and into *Worklist*. See the node φ_1 in Figure 2.

During the first loop iteration, we choose φ_1 to be the element taked from the worklist. Now we compute $post^{\#}$ wrt. each program transition. For ρ_1 we



$$\varphi_1 = \alpha(\varphi_{init})$$
$$\varphi_2 = post^{\#}(\varphi_1, \rho_1)$$
$$post^{\#}(\varphi_2, \rho_2) \models \varphi_2$$
$$\varphi_3 = post^{\#}(\varphi_2, \rho_3)$$
$$\varphi_4 = post^{\#}(\varphi_3, \rho_4)$$

Fig. 2. Applying AbstReach on the program in Figure **??** and the set of predicates $Preds = \{false, at\_\ell_1, \ldots, at\_\ell_5, y \geq z, x \geq y\}$. The nodes $\varphi_1, \ldots, \varphi_4$ represent elements of $ReachStates^{\#}$. Labeled edges connecting the nodes represent *Parent*. The dotted edge denotes the entailment relation between $post^{\#}(\varphi_2, \rho_2)$ and φ_2.

obtain $\varphi_2 = (at\_\ell_2 \wedge y \geq z)$. The entailment check $\varphi_2 \models \bigvee ReachStates^{\#}$ fails, since $\bigvee ReachStates^{\#}$ is equal to φ_1 and $\varphi_2 \not\models \varphi_1$. Hence, φ_2 is added to $ReachStates^{\#}$. As a result, the tuple $(\varphi_1, \rho_1, \varphi_2)$ is added to *Parent* and φ_2 becomes a worklist item. See the node φ_2 as well as the edge between φ_1 and φ_2 in Figure 2. We continue with applying program transitions on φ_1. For ρ_2 we obtain $post^{\#}(\varphi_1, \rho_2) = false$. Since $false \models \bigvee ReachStates^{\#}$ there is no addition to $ReachStates^{\#}$. Similarly, applying $\rho_3, \ldots, \rho_5$ does not modify $ReachStates^{\#}$.

We start the second loop iteration with $ReachStates^{\#} = \{\varphi_1, \varphi_2\}$, $Worklist = \{\varphi_2\}$, and $Parent = \{(\varphi_1, \rho_1, \varphi_2)\}$. We choose φ_2 from the worklist. When applying $post^{\#}$ on φ_2 only ρ_2 and ρ_3 result sets of successor states that are not equal to *false*. We obtain $post^{\#}(\varphi_2, \rho_2) = (at\_\ell_2 \wedge y \geq z)$. Since $(at\_\ell_2 \wedge y \geq z)$ entails φ_2 and hence $\bigvee ReachStates^{\#}$, nothing is added to $ReachStates^{\#}$ and we proceed directly with ρ_3. For $\varphi_3 = post^{\#}(\varphi_2, \rho_3) = (at\_\ell_3 \wedge y \geq z \wedge x \geq y)$ we observe that $\varphi_3 \not\models \bigvee ReachStates^{\#}$. Hence, we add φ_3 to $ReachStates^{\#}$ and *Worklist*, while $(\varphi_2, \rho_3, \varphi_3)$ is recorded in *Parent*. See the node φ_3 as well as the edge between φ_2 and φ_3 in Figure 2.

At the beginning of the third loop iteration we have $ReachStates^{\#} = \{\varphi_1, \varphi_2, \varphi_3\}$, $Worklist = \{\varphi_3\}$, and $Parent = \{(\varphi_1, \rho_1, \varphi_2), (\varphi_2, \rho_3, \varphi_3)\}$. We choose φ_3 from the worklist. After computing φ_4 by applying ρ_4 and discovering that $\varphi_4 \not\models \bigvee ReachStates^{\#}$, we add φ_4 following the algorithm. See the node φ_4 as well as the edge between φ_3 and φ_4 in Figure 2. Since all other program transition yield *false* we proceed with the next iteration.

The fourth loop iteration removes φ_4 from the worklist, but does not add any new elements to it. Hence AbstReach terminates and outputs $\mathit{ReachStates}^{\#} = \{\varphi_1, \ldots, \varphi_4\}$ as well as $\mathit{Parent} = \{(\varphi_1, \rho_1, \varphi_2), (\varphi_2, \rho_3, \varphi_3), (\varphi_3, \rho_4, \varphi_4)\}$. $\square$

Monotonicity:

$$\forall \varphi_1\ \forall \varphi_2 : (\varphi_1 \models \varphi_2) \to (\alpha(\varphi_1) \models \alpha(\varphi_2))$$