

Reachability Analysis

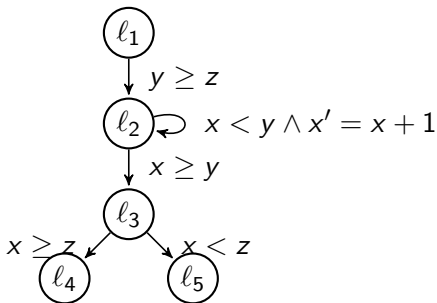
Andreas Podelski

November 28, 2011

```

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

```



$$\rho_1 = (\text{move}(l_1, l_2) \wedge y \geq z \wedge \text{skip}(x, y, z))$$

$$\rho_2 = (\text{move}(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge \text{skip}(y, z))$$

$$\rho_3 = (\text{move}(l_2, l_3) \wedge x \geq y \wedge \text{skip}(x, y, z))$$

$$\rho_4 = (\text{move}(l_3, l_4) \wedge x \geq z \wedge \text{skip}(x, y, z))$$

$$\rho_5 = (\text{move}(l_3, l_5) \wedge x + 1 \leq z \wedge \text{skip}(x, y, z))$$

transition relation ρ expressed by logical formula

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

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

$$\rho_3 \equiv (\text{move}(\ell_2, \ell_3) \wedge x \geq y \wedge \text{skip}(x, y, z))$$

$$\rho_4 \equiv (\text{move}(\ell_3, \ell_4) \wedge x \geq z \wedge \text{skip}(x, y, z))$$

$$\rho_5 \equiv (\text{move}(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge \text{skip}(x, y, z))$$

abbreviations:

$$\text{move}(\ell, \ell') \equiv (pc = \ell \wedge pc' = \ell')$$

$$\text{skip}(v_1, \dots, v_n) \equiv (v'_1 = v_1 \wedge \dots \wedge v'_n = v_n)$$

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'

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'

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} ,

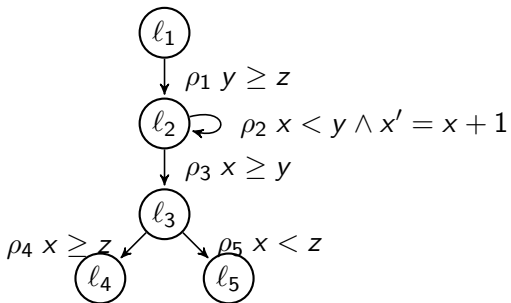
$$\varphi_{err} \wedge \varphi_{reach} \models \text{false} .$$

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:

$$\begin{aligned}
\varphi_{reach} = & (pc = l_1 \vee \\
& pc = l_2 \wedge y \geq z \vee \\
& pc = l_3 \wedge y \geq z \wedge x \geq y \vee \\
& pc = l_4 \wedge y \geq z \wedge x \geq y)
\end{aligned}$$

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 $\phi \equiv pc = l_2 \wedge y \geq z$, transition relation $\rho \equiv \rho_2$,

$$\rho_2 \equiv (move(l_2, l_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$$

$post(\phi, \rho_2)$

$$= (\exists V'' : (pc = l_2 \wedge y \geq z)[V''/V] \wedge \rho_2[V''/V][V/V'])$$

$$= (\exists V'' : (pc'' = l_2 \wedge y'' \geq z'') \wedge$$

$$(pc'' = l_2 \wedge pc' = l_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge y' = y'' \wedge z' = z''))[V/V']$$

$$= (\exists V'' : (pc'' = l_2 \wedge y'' \geq z'') \wedge$$

$$(pc'' = l_2 \wedge pc = l_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge y = y'' \wedge z = z''))$$

$$= (pc = l_2 \wedge y \geq z \wedge x \leq y)$$

[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 $post$:

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

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

finite iteration post may suffice

“fixpoint reached in n steps” if

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

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

'distributed' fixpoint test

- ▶ $\rho_{\mathcal{R}}$ is itself a disjunction: $\rho_{\mathcal{R}} = \bigvee_{\rho \in \mathcal{R}} \rho = \{\rho_1, \dots, \rho_m\}$
- ▶ $post(\phi, \rho)$ 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 $\rho_{j_1}, \dots, \rho_{j_n}$

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

- ▶ "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

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

example iteration

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

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

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

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

- ▶ $post(\varphi_{init}, \rho_1) \equiv (pc = \ell_2 \wedge y \geq z)$
- ▶ $\rho_2 \equiv (move(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$

$post(pc = \ell_2 \wedge y \geq z, \rho_2)$

$$= (\exists V'' : (pc = \ell_2 \wedge y \geq z)[V''/V] \wedge \rho_2[V''/V][V/V'])$$

$$= (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge$$

$$(pc'' = \ell_2 \wedge pc' = \ell_2 \wedge x'' + 1 \leq y'' \wedge x' = x'' + 1 \wedge y' = y'' \wedge z' = z''))[V/V']$$

$$= (\exists V'' : (pc'' = \ell_2 \wedge y'' \geq z'') \wedge$$

$$(pc'' = \ell_2 \wedge pc = \ell_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge y = y'' \wedge z = z''))$$

$$= (pc = \ell_2 \wedge y \geq z \wedge x \leq y)$$

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

$$\begin{aligned} & post^2(pc = l_2 \wedge y \geq z, \rho_2) \\ &= post(post(pc = l_2 \wedge y \geq z, \rho_2), \rho_2) \\ &= post(pc = l_2 \wedge y \geq z \wedge x \leq y, \rho_2) \\ &= (\exists V'' : (pc'' = l_2 \wedge y'' \geq z'' \wedge x'' \leq y'') \wedge \\ &\quad (pc'' = l_2 \wedge pc = l_2 \wedge x'' + 1 \leq y'' \wedge x = x'' + 1 \wedge \\ &\quad y = y'' \wedge z = z'')) \\ &= (pc = l_2 \wedge y \geq z \wedge x - 1 \leq y \wedge x \leq y) \\ &= (pc = l_2 \wedge y \geq z \wedge x \leq y) \end{aligned}$$

compute φ_{reach} for example program (1)

apply transition relation of the program once:

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

obtain the post-condition for one more application:

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

compute φ_{reach} for example program (2)

repeat the application step once again:

$$\begin{aligned} & post(pc = l_2 \wedge y \geq z \wedge x \leq y \vee \\ & \quad pc = l_3 \wedge y \geq z \wedge x \geq y, \rho_{\mathcal{R}}) \\ &= (post(pc = l_2 \wedge y \geq z \wedge x \leq y, \rho_{\mathcal{R}}) \vee \\ & \quad post(pc = l_3 \wedge y \geq z \wedge x \geq y, \rho_{\mathcal{R}})) \\ &= (post(pc = l_2 \wedge y \geq z \wedge x \leq y, \rho_2) \vee \\ & \quad post(pc = l_2 \wedge y \geq z \wedge x \leq y, \rho_3) \vee \\ & \quad post(pc = l_3 \wedge y \geq z \wedge x \geq y, \rho_4) \vee \\ & \quad post(pc = l_3 \wedge y \geq z \wedge x \geq y, \rho_5)) \\ &= (pc = l_2 \wedge y \geq z \wedge x \leq y \vee \\ & \quad pc = l_3 \wedge y \geq z \wedge x = y \vee \\ & \quad pc = l_4 \wedge y \geq z \wedge x \geq y) \end{aligned}$$

compute φ_{reach} for example program

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

$$pc = l_1 \vee$$

$$pc = l_2 \wedge y \geq z \vee$$

$$pc = l_2 \wedge y \geq z \wedge x \leq y \vee pc = l_3 \wedge y \geq z \wedge x \geq y \vee$$

$$pc = l_2 \wedge y \geq z \wedge x \leq y \vee pc = l_3 \wedge y \geq z \wedge x = y \vee$$

$$pc = l_4 \wedge y \geq z \wedge x \geq y$$

disjunction in a logically equivalent, simplified form:

$$pc = l_1 \vee$$

$$pc = l_2 \wedge y \geq z \vee$$

$$pc = l_3 \wedge y \geq z \wedge x \geq y \vee$$

$$pc = l_4 \wedge y \geq z \wedge x \geq y$$

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

Checking Safety

- ▶ an *inductive invariant* φ contains the initial states and is closed under successors

Checking Safety

- ▶ an *inductive invariant* φ contains the initial states and is closed under successors
- ▶ formally:

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

Checking Safety

- ▶ an *inductive invariant* φ contains the initial states and is closed under successors
- ▶ formally:

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

- ▶ program is safe if there exists an inductive invariant φ that does not contain an error state (why?)

$$\varphi \wedge \varphi_{err} \models \text{false}$$

Checking Safety

- ▶ an *inductive invariant* φ contains the initial states and is closed under successors
- ▶ formally:

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

- ▶ program is safe if there exists an inductive invariant φ that does not contain an error state (why?)

$$\varphi \wedge \varphi_{err} \models \text{false}$$

- ▶ 1. “ φ_{reach} is the strongest inductive invariant” (why?)

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

- ▶ 2. program safe if φ_{reach} does not contain an error state:

$$\varphi_{reach} \wedge \varphi_{err} \models \text{false}$$

inductive invariants for example program

- ▶ weakest inductive invariant:

inductive invariants for example program

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

$$pc = \ell_1 \vee$$

$$(pc = \ell_2 \wedge y \geq z) \vee$$

$$(pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee$$

$$(pc = \ell_4 \wedge y \geq z \wedge x \geq y)$$

inductive invariants for example program

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

$$pc = \ell_1 \vee$$

$$(pc = \ell_2 \wedge y \geq z) \vee$$

$$(pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee$$

$$(pc = \ell_4 \wedge y \geq z \wedge x \geq y)$$

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

$$pc = \ell_1 \vee$$

$$(pc = \ell_2 \wedge y \geq z) \vee$$

$$(pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee$$

$$pc = \ell_4$$

inductive invariants for example program

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

$$pc = \ell_1 \vee$$

$$(pc = \ell_2 \wedge y \geq z) \vee$$

$$(pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee$$

$$(pc = \ell_4 \wedge y \geq z \wedge x \geq y)$$

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

$$pc = \ell_1 \vee$$

$$(pc = \ell_2 \wedge y \geq z) \vee$$

$$(pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee$$

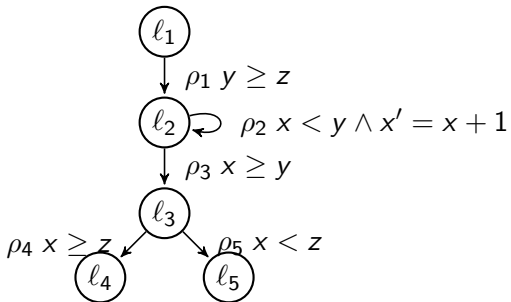
$$pc = \ell_4$$

- ▶ 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):

$$\begin{aligned}
\varphi_{reach} = & (pc = l_1 \vee \\
& pc = l_2 \wedge y \geq z \vee \\
& pc = l_3 \wedge y \geq z \wedge x \geq y \vee \\
& pc = l_4)
\end{aligned}$$