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)
- 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'

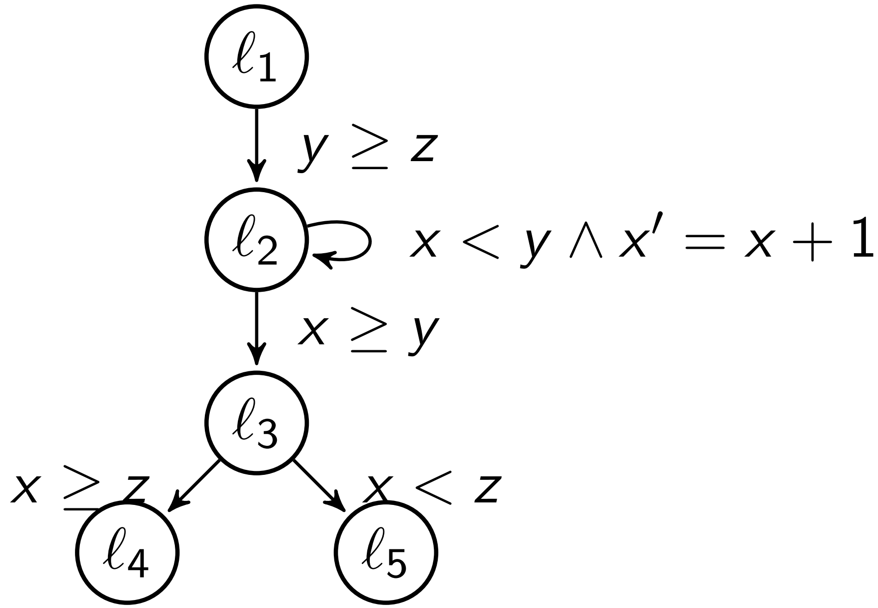
transition relation ρ expressed by logica formula

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

abbreviations:

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

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



.

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

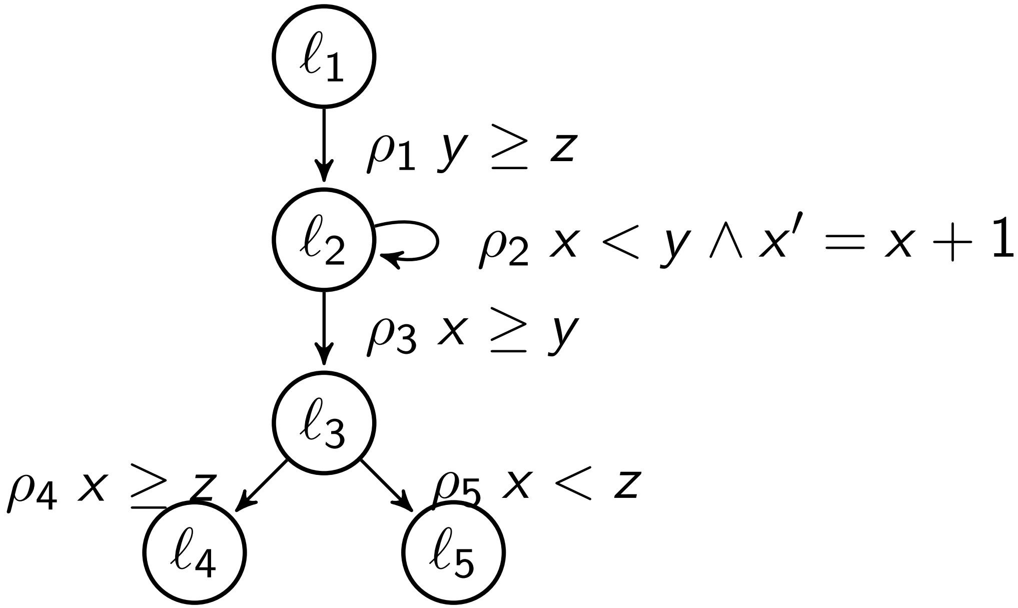
correctness: safety

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

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

  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*(ϕ, ρ) in examples

- ρ has no primed variables

application of $\mathit{post}(\phi, \rho)$ in examples

- ρ has no primed variables
  $\mathit{post}(\phi, \rho) = \phi \wedge \rho$

application of *post*(ϕ, ρ) in examples

- ρ has no primed variables
  post$(\phi, \rho) = \phi \wedge \rho$
- ρ has only primed variables

application of $\mathit{post}(\phi, \rho)$ in examples

- ρ has no primed variables
  $\mathit{post}(\phi, \rho) = \phi \wedge \rho$
- ρ has only primed variables
  $\mathit{post}(\phi, \rho) = \rho[V/V']$

application of $post(\phi, \rho)$ in examples

- ρ has no primed variables
  $post(\phi, \rho) = \phi \land \rho$
- ρ has only primed variables
  $post(\phi, \rho) = \rho[V/V']$
- ρ is an update of x by an expression e without x, say
  $\rho \; = \;\; x := e(y, z)$

application of $post(\phi, \rho)$ in examples

- ρ has no primed variables
  $post(\phi, \rho) = \phi \wedge \rho$
- ρ has only primed variables
  $post(\phi, \rho) = \rho[V/V']$
- ρ is an update of x by an expression e without x, say
  $\rho \ = \ \ x := e(y, z)$
  $post(\phi, \rho) = \exists x \phi \wedge x = e$

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*:

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

n-th disjunct = iterate for natural number n (disjunction = "ω iteration")

finite iteration post may suffice

"fixpoint reached in n steps" if

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

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

‘distributed’ iteration of $post(\cdot, \rho_{\mathcal{R}})$

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

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

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

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

- *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 $post(\cdot,\cdot)$ to any disjunct ϕ_k in Φ is contained in one of the disjuncts $\phi_{k'}$ in Φ is

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

compute φ_{reach} for example program (1)

apply post on set of initial states:

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

apply post on successor states:

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

compute φ_{reach} for example program (2)

repeat the application step once again:

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

compute φ_{reach} for example program

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

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

disjunction in a logically equivalent, simplified form:

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

above disjunction $= \varphi_{reach}$ 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 φ

checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant φ
- inductive:

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

checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant φ
- inductive:

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

- safe:

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

checking safety = finding safe inductive invariant

- program is safe if there exists a safe inductive invariant φ
- inductive:

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

- safe:

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

- justification:
  1. "φ_{reach} is the strongest inductive invariant"

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

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

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

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)

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

inductive invariants for example program

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

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

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

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

inductive invariants for example program

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

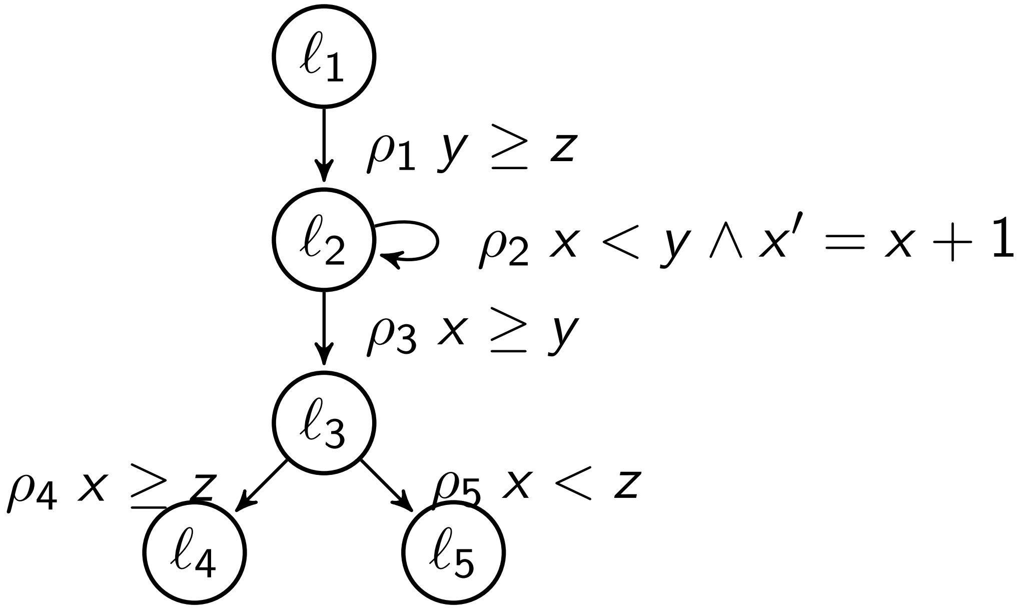
$$
\begin{aligned}
& pc = \ell_1 \vee \\
& (pc = \ell_2 \wedge y \geq z) \vee \\
& (pc = \ell_3 \wedge y \geq z \wedge x \geq y) \vee \\
& (pc = \ell_4 \wedge y \geq z \wedge x \geq y)
\end{aligned}
$$

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

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

- 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):

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

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*

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

$$\begin{aligned}
post(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - 1 \leq z \wedge x \leq y) \\
post^2(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - 2 \leq z \wedge x \leq y) \\
post^3(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - 3 \leq z \wedge x \leq y) \\
&\cdots \\
post^n(at_\ell_2 \wedge x \leq z, \rho_2) &= (at_\ell_2 \wedge x - n \leq z \wedge x \leq y)
\end{aligned}$$

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:

$$\begin{aligned}(at\_\ell_2 \wedge x - 2 \leq z \wedge x \leq y) \quad &\not\models \\ at\_\ell_2 \wedge x \leq z \vee \\ at\_\ell_2 \wedge x - 1 \leq z \wedge x \leq y\end{aligned}$$

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

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