relations as formulas

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

- $V$ - finite tuple of program variables
- $pc$ - program counter variable (pc included in $V$)
- $\varphi_{\text{init}}$ - initiation condition given by formula over $V$
- $R$ - a finite set of transition relations
- $\varphi_{\text{err}}$ - an error condition given by a formula over $V$
- transition relation $\rho \in R$ given by
  formula over the variables $V$ and their primed versions $V'$
transition relation $\rho$ expressed by logica formula

\[
\begin{align*}
\rho_1 &\equiv (\text{move}(l_1, l_2) \land y \geq z \land \text{skip}(x, y, z)), \\
\rho_2 &\equiv (\text{move}(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)), \\
\rho_3 &\equiv (\text{move}(l_2, l_3) \land x \geq y \land \text{skip}(x, y, z)), \\
\rho_4 &\equiv (\text{move}(l_3, l_4) \land x \geq z \land \text{skip}(x, y, z)), \\
\rho_5 &\equiv (\text{move}(l_3, l_5) \land x + 1 \leq z \land \text{skip}(x, y, z))
\end{align*}
\]

abbreviations:

\[
\begin{align*}
\text{move}(l, l') &\equiv (pc = l \land pc' = l') \\
\text{skip}(v_1, \ldots, v_n) &\equiv (v'_1 = v_1 \land \ldots \land v'_n = v_n)
\end{align*}
\]
1: assume(y >= z);
2: while (x < y) {
    x++;
}
3: assert(x >= z);
4: exit
5: error

\[ \rho_1 = (\text{move}(\ell_1, \ell_2) \land y \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_2 = (\text{move}(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)) \]
\[ \rho_3 = (\text{move}(\ell_2, \ell_3) \land x \geq y \land \text{skip}(x, y, z)) \]
\[ \rho_4 = (\text{move}(\ell_3, \ell_4) \land x \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_5 = (\text{move}(\ell_3, \ell_5) \land x + 1 \leq z \land \text{skip}(x, y, z)) \]
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 $\varphi_{reach} = \text{set of reachable program states}$
1: assume(y >= z);
2: while (x < y) {
   x++;
}
3: assert(x >= z);
4: exit
5: error

set of reachable states:

\[ \varphi_{reach} = (pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_4 \land y \geq z \land x \geq y) \]
post operator

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

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

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

$$post(\varphi, \rho_1 \lor \rho_2) = (post(\varphi, \rho_1) \lor post(\varphi, \rho_2))$$
$$post(\varphi_1 \lor \varphi_2, \rho) = (post(\varphi_1, \rho) \lor post(\varphi_2, \rho))$$
application of $post(\phi, \rho)$ in examples

- $\rho$ has no primed variables
application of $post(\phi, \rho)$ in examples

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

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

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

  \[
  \rho = x := e(y, z) \\
  post(\phi, \rho) = \exists x \phi \land x = e
  \]
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 \( post \):

\[
\varphi_{\text{reach}} = \varphi_{\text{init}} \lor post(\varphi_{\text{init}}, \rho_\mathcal{R}) \lor post(post(\varphi_{\text{init}}, \rho_\mathcal{R}), \rho_\mathcal{R}) \lor \ldots \\
= \lor_{i \geq 0} post^i(\varphi_{\text{init}}, \rho_\mathcal{R})
\]

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

“fixpoint reached in $n$ steps” if

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

then

$$\forall i = 0 \ post^i(\varphi_{init}, \rho_{\mathcal{R}}) = \forall i \geq 0 \ post^i(\varphi_{init}, \rho_{\mathcal{R}})$$
‘distributed’ iteration of $\text{post}(\cdot, \rho_R)$

- $\rho_R$ is itself a disjunction: $\rho_R = \rho_1 \lor \ldots \lor \rho_m$
- $\text{post}(\phi, \rho)$ 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 $\rho_{j_1}, \ldots, \rho_{j_n}$

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

- $\phi_k \not= \emptyset$ only if sequence of transitions $\rho_{j_1}, \ldots, \rho_{j_n}$ corresponds to path in control flow graph of program since:

$$\text{post}(pc = \ell_i \land \ldots, \text{move}(\ell_j, \ell_{\ldots}) \land \ldots) = \emptyset \text{ if } i \not= j$$

- 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 \(\phi_k\) in \(\Phi\) is contained in one of the disjuncts \(\phi_{k'}\) in \(\Phi\) is

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

apply post on set of initial states:

$$post(pc = l_1, \rho_R) = post(pc = l_1, \rho_1) = pc = l_2 \land y \geq z$$

apply post on successor states:

$$post(pc = l_2 \land y \geq z, \rho_R) = post(pc = l_2 \land y \geq z, \rho_2) \lor post(pc = l_2 \land y \geq z, \rho_3) = pc = l_2 \land y \geq z \land x \leq y \lor pc = l_3 \land y \geq z \land x \geq y$$
compute $\varphi_{reach}$ for example program (2)

repeat the application step once again:

$$post(pc = l_2 \land y \geq z \land x \leq y \lor$$

$$pc = l_3 \land y \geq z \land x \geq y, \rho_R)$$

$$= post(pc = l_2 \land y \geq z \land x \leq y, \rho_R) \lor$$

$$post(pc = l_3 \land y \geq z \land x \geq y, \rho_R)$$

$$= post(pc = l_2 \land y \geq z \land x \leq y, \rho_2) \lor$$

$$post(pc = l_2 \land y \geq z \land x \leq y, \rho_3) \lor$$

$$post(pc = l_3 \land y \geq z \land x \geq y, \rho_4) \lor$$

$$post(pc = l_3 \land y \geq z \land x \geq y, \rho_5)$$

$$= pc = l_2 \land y \geq z \land x \leq y \lor$$

$$pc = l_3 \land y \geq z \land x = y \lor$$

$$pc = l_4 \land y \geq z \land x \geq y$$
compute $\varphi_{reach}$ for example program

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

\[
\begin{align*}
    pc &= \ell_1 \lor \\
    pc &= \ell_2 \land y \geq z \lor \\
    pc &= \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x \geq y \lor \\
    pc &= \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x = y \lor \\
    pc &= \ell_4 \land y \geq z \land x \geq y
\end{align*}
\]

disjunction in a logically equivalent, simplified form:

\[
\begin{align*}
    pc &= \ell_1 \lor \\
    pc &= \ell_2 \land y \geq z \lor \\
    pc &= \ell_3 \land y \geq z \land x \geq y \lor \\
    pc &= \ell_4 \land y \geq z \land x \geq y
\end{align*}
\]

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 $\varphi$
checking safety $\equiv$ finding safe inductive invariant

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

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

- program is safe if there exists a safe inductive invariant $\varphi$
- inductive:
  \[
  \varphi_{\text{init}} \models \varphi \quad \text{and} \quad \text{post}(\varphi, \rho_R) \models \varphi.
  \]
- safe:
  \[
  \varphi \land \varphi_{\text{err}} \models \text{false}
  \]
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:
inductive invariants for example program

- weakest inductive invariant:  *true* (set of all states)
  contains error states
- strongest inductive invariant (does not contain error states)
  \[
  pc = l_1 \lor (pc = l_2 \land y \geq z) \lor (pc = l_3 \land y \geq z \land x \geq y) \lor (pc = l_4 \land y \geq z \land x \geq y)
  \]
inductive invariants for example program

- weakest inductive invariant:  \textit{true} (set of all states) contains error states
- strongest inductive invariant (does not contain error states)
  \[
  pc = l_1 \lor \\
  (pc = l_2 \land y \geq z) \lor \\
  (pc = l_3 \land y \geq z \land x \geq y) \lor \\
  (pc = l_4 \land y \geq z \land x \geq y)
  \]

- a slightly weaker inductive invariant also proves the safety of our examples:
  \[
  pc = l_1 \lor \\
  (pc = l_2 \land y \geq z) \lor \\
  (pc = l_3 \land y \geq z \land x \geq y) \lor \\
  pc = l_4
  \]
inductive invariants for example program

- weakest inductive invariant:  \textit{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):

\[ \varphi_{reach} = (pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_4) \]
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*

\[ \rho_2 \equiv (\text{move}(l_2, l_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)) \]

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

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

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

\[ \ldots \]

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

\[
(at\_l_2 \land x - 2 \leq z \land x \leq y) \not\subseteq
\]
\[
\begin{align*}
at\_l_2 \land x & \leq z \lor \\
& \\
& at\_l_2 \land x - 1 \leq z \land x \leq y
\end{align*}
\]

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

\[
\forall n \geq 1 : (at\_l_2 \land x - n \leq z \land x \leq y) \not\subseteq
\]
\[
\begin{align*}
& at\_l_2 \land x \leq z \lor \\
& \lor_{1 \leq i < n}(at\_l_2 \land x - i \leq z \land x \leq y)
\end{align*}
\]