Reachability Analysis

Andreas Podelski

November 28, 2011
program with **assume ()** and **assert ()**

- **assume (e) ≡ if e then skip else halt**
program with **assume ()** and **assert ()**

- **assume (e) ≡ if e then skip else halt**
- **assert (e) ≡ 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*:

$$
\text{safety} = \text{non-reachability of error (no execution of error branch)}
$$
program with \texttt{assume} () and \texttt{assert} ()

\begin{itemize}
\item \texttt{assume} (e) \equiv \text{if } e \text{ then skip else halt}
\item \texttt{assert} (e) \equiv \text{if } e \text{ then skip else error}
\item generalize partial correctness:
  correctness of program wrt. Hoare triple:
  \[
  \{\phi\} \ C \ {\psi}\n  \]
\end{itemize}
program with \texttt{assume} () and \texttt{assert} ()

\begin{itemize}
\item \texttt{assume} \( (e) \equiv \text{if } e \text{ then skip else halt} \)
\item \texttt{assert} \( (e) \equiv \text{if } e \text{ then skip else error} \)
\item generalize \textit{partial correctness}:
    correctness of program wrt. Hoare triple:
    \[
    \{ \phi \} \ C \ {\psi} \equiv \text{safety of program: } \texttt{assume} \ (\phi) \ ; \ C \ ; \texttt{assert} \ (\psi) \]
    \text{safety} = \text{non-reachability of error}
    (no execution of error branch)
\end{itemize}
validity of Hoare triple:

\[
\{ y \geq z \}
\]
while (x < y) {
\[
    x++;
\]
}
\[
\{ x \geq z \}
\]
≡ 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

ρ₁ = (move(ℓ₁, ℓ₂) ∧ y ≥ z ∧ skip(x, y, z))
ρ₂ = (move(ℓ₂, ℓ₂) ∧ x + 1 ≤ y ∧ x' = x + 1 ∧ skip(y, z))
ρ₃ = (move(ℓ₂, ℓ₃) ∧ x ≥ y ∧ skip(x, y, z))
ρ₄ = (move(ℓ₃, ℓ₄) ∧ x ≥ z ∧ skip(x, y, z))
ρ₅ = (move(ℓ₃, ℓ₅) ∧ x + 1 ≤ z ∧ skip(x, y, z))
transition relation $\rho$ expressed by logica formula

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

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

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

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

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

abbreviations:

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

$$
\text{skip}(v_1, \ldots, v_n) \equiv (v'_1 = v_1 \land \ldots \land v'_n = v_n)
$$
program $P = (V, pc, \varphi_{init}, R, \varphi_{err})$

- $V$ - finite tuple of program variables
- $pc$ - program counter variable (pc included in $V$)
- $\varphi_{init}$ - initiation condition given by formula over $V$
- $R$ - a finite set of transition relations
- $\varphi_{err}$ - 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 $V'$
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
- \( \Sigma \) = 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
- $\Sigma = \text{set of program states}$
- formula with free variables in $V = \text{set of program states}$
- 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 $V'$
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
- $\Sigma = \text{set of program states}$
- formula with free variables in $\mathcal{V} = \text{set of program states}$
- formula with free variables in $\mathcal{V}$ and $\mathcal{V} = \text{binary relation over program states}$
  - first component of each pair assigns values to $\mathcal{V}$
  - second component of the pair assigns values to $\mathcal{V}$
- 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
- $\Sigma$ = set of program states
- formula with free variables in $V = \text{set of program states}$
- 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 $V'$
- 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
- $\Sigma = \text{set of program states}$
- formula with free variables in $V = \text{set of program states}$
- 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 $V'$
- 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 $y \geq z = \text{set of program states in which the value of the variable } y \text{ is greater than the value of } z$
example: states, sets, and relations

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

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

- if program state $s$ assigns 1, 3, 2, and $\ell_1$ to program variables $x, y, z, \text{ and } pc$, respectively, then $s \models y \geq z$
example: states, sets, and relations

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

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

- if program state $s$ assigns 1, 3, 2, and $\ell_1$ 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 $\mathcal{R} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$

\[
\begin{align*}
\rho_1 &= (move(\ell_1, \ell_2) \land y \geq z \land skip(x, y, z)) \\
\rho_2 &= (move(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land skip(y, z)) \\
\rho_3 &= (move(\ell_2, \ell_3) \land x \geq y \land skip(x, y, z)) \\
\rho_4 &= (move(\ell_3, \ell_4) \land x \geq z \land skip(x, y, z)) \\
\rho_5 &= (move(\ell_3, \ell_5) \land x + 1 \leq z \land skip(x, y, z))
\end{align*}
\]
1: assume(y >= z);
2: while (x < y) {
   x++; 
}
3: assert(x >= z);
4: exit
5: error

\[ \rho_1 = (move(\ell_1, \ell_2) \land y \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_2 = (move(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)) \]
\[ \rho_3 = (move(\ell_2, \ell_3) \land x \geq y \land \text{skip}(x, y, z)) \]
\[ \rho_4 = (move(\ell_3, \ell_4) \land x \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_5 = (move(\ell_3, \ell_5) \land x + 1 \leq z \land \text{skip}(x, y, z)) \]
initial state, error state, transition relation $\mathcal{R}$

- each state that satisfies the initiation condition $\varphi_{init}$ is called an \textit{initial} state
- each state that satisfies the error condition $\varphi_{err}$ is called an \textit{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., $(s_i, s_{i+1}) \models \rho_R$
- 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 $(s_n, s) \models \rho_R$
1: assume(y >= z);
2: while (x < y) {
    x++;
}
3: assert(x >= z);
4: exit
5: error

example of a computation:

(ℓ₁, 1, 3, 2), (ℓ₂, 1, 3, 2), (ℓ₂, 2, 3, 2), (ℓ₂, 3, 3, 2), (ℓ₃, 3, 3, 2), (ℓ₄, 3, 3, 2)

- sequence of transitions ρ₁, ρ₂, ρ₂, ρ₃, ρ₄
- 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 $\varphi_{reach}$,

  $\varphi_{err} \land \varphi_{reach} \models false$.

Where $\varphi_{reach} =$ 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:

\[ \varphi_{\text{reach}} = (pc = \ell_1 \lor
\begin{align*}
pc &= \ell_2 \land y \geq z \lor \\
pc &= \ell_3 \land y \geq z \land x \geq y \lor \\
pc &= \ell_4 \land y \geq z \land x \geq y
\end{align*} \]
post operator

- let \( \varphi \) be a formula over \( V \)
- let \( \rho \) be a formula over \( V \) and \( V' \)
- define a *post-condition* function \( \text{post} \) by:

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

an application \( \text{post}(\varphi, \rho) \) computes the image of the set \( \varphi \) under the relation \( \rho \)

- post distributes over disjunction wrt. each argument:

\[
\text{post}(\varphi, \rho_1 \lor \rho_2) = (\text{post}(\varphi, \rho_1) \lor \text{post}(\varphi, \rho_2))
\]

\[
\text{post}(\varphi_1 \lor \varphi_2, \rho) = (\text{post}(\varphi_1, \rho) \lor \text{post}(\varphi_2, \rho))
\]
application of $\text{post}(\phi, \rho)$ in example program

set of states $\phi \equiv pc = l_2 \land y \geq z$, transition relation $\rho \equiv \rho_2$,

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

\[
\text{post}(\phi, \rho_2) = (\exists V'' : (pc = l_2 \land y \geq z)[V''/V] \land \rho_2[V''/V][V/V'])
\]

$$= (\exists V'' : (pc'' = l_2 \land y'' \geq z'') \land (pc'' = l_2 \land pc' = l_2 \land x'' + 1 \leq y'' \land x' = x'' + 1 \land y' = y'' \land z' = z'')[V/V'])$$

$$= (\exists V'' : (pc'' = l_2 \land y'' \geq z'') \land (pc'' = l_2 \land pc = l_2 \land x'' + 1 \leq y'' \land x = x'' + 1 \land y = y'' \land z = z''))$$

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

[renamed] program variables:

$V = (pc, x, y, z), \ V' = (pc', x', y', z'), \ V'' = (pc'', x'', y'', z'')$
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 \( \varphi_{reach} \) using iterates of \( post \):

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

disjuncts = iterates for every natural number \( n \) ("\( \omega \) iteration")
finite iteration post may suffice

“fixpoint reached in $n$ steps” if

if $\bigvee_{i=1}^{n} \text{post}^i(\varphi_{\text{init}}, \rho_R) = \bigvee_{i=1}^{n+1} \text{post}^i(\varphi_{\text{init}}, \rho_R)$

then $\bigvee_{i=1}^{n} \text{post}^i(\varphi_{\text{init}}, \rho_R) = \bigvee_{i \geq 0} \text{post}^i(\varphi_{\text{init}}, \rho_R)$
‘distributed’ fixpoint test

- $\rho_R$ is itself a disjunction: $\rho_R = \bigvee_{\rho \in R} \rho = \{\rho_1, \ldots, \rho_m\}$
- $post(\phi, \rho)$ distributes over disjunction in both arguments
- in ‘distributed’ disjunction $\Phi = \{\phi_k \mid k \in M\}$, every disjunct $\phi_k$ 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 $\phi_k$ is contained in one of the disjuncts $\phi_k'$ in “big” disjunction

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

$\triangleright \text{post}(\varphi_{\text{init}}, \rho_1) \equiv \text{post}(pc = \ell_1, \rho_1) \\
\equiv pc = \ell_2 \land y \geq z$

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

$\triangleright \text{post}((pc = \ell_i), \rho_j) \equiv \emptyset$ if $\rho_j \land pc = \ell_i \equiv \emptyset$
loop applied to \( post(\varphi_{init}, \rho_1) \)

1. \( post(\varphi_{init}, \rho_1) \equiv (pc = l_2 \land y \geq z) \)

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

\( post(pc = l_2 \land y \geq z, \rho_2) \)

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

\[= (\exists V'' : (pc'' = l_2 \land y'' \geq z'')) \land \]

\[\quad (pc'' = l_2 \land pc' = l_2 \land x'' + 1 \leq y'' \land x' = x'' + 1 \land \]

\[\quad \quad y' = y'' \land z' = z'')[V/V']) \]

\[= (\exists V'' : (pc'' = l_2 \land y'' \geq z'')) \land \]

\[\quad (pc'' = l_2 \land pc = l_2 \land x'' + 1 \leq y'' \land x = x'' + 1 \land \]

\[\quad \quad y = y'' \land z = z'') \]

\[= (pc = l_2 \land y \geq z \land x \leq y) \]
loop applied twice to $\text{post}(\varphi_{init}, \rho_1)$

\[
\begin{align*}
\text{post}^2(pc = l_2 \land y \geq z, \rho_2) \\
&= \text{post}(\text{post}(pc = l_2 \land y \geq z, \rho_2), \rho_2) \\
&= \text{post}(pc = l_2 \land y \geq z \land x \leq y, \rho_2) \\
&= (\exists V'' : (pc'' = l_2 \land y'' \geq z'' \land x'' \leq y'') \land \\
& \quad (pc'' = l_2 \land pc = l_2 \land x'' + 1 \leq y'' \land x = x'' + 1 \land \\
& \quad \quad y = y'' \land z = z'')) \\
&= (pc = l_2 \land y \geq z \land x - 1 \leq y \land x \leq y) \\
&= (pc = l_2 \land y \geq z \land x \leq y)
\end{align*}
\]
compute $\varphi_{reach}$ for example program (1)

apply transition relation of the program once:

$$post(pc = \ell_1, \rho_R)$$

$$= (post(pc = \ell_1, \rho_1) \lor post(pc = \ell_1, \rho_2) \lor post(pc = \ell_1, \rho_3) \lor$$

$$post(pc = \ell_1, \rho_4) \lor post(pc = \ell_1, \rho_5))$$

$$= post(pc = \ell_1, \rho_1)$$

$$= (pc = \ell_2 \land y \geq z)$$

obtain the post-condition for one more application:

$$post(pc = \ell_2 \land y \geq z, \rho_R)$$

$$= (post(pc = \ell_2 \land y \geq z, \rho_2) \lor post(pc = \ell_2 \land y \geq z, \rho_3))$$

$$= (pc = \ell_2 \land y \geq z \land x \leq y \lor pc = \ell_3 \land y \geq z \land x \geq y)$$
compute $\varphi_{reach}$ for example program (2)

repeat the application step once again:

\[
\begin{align*}
post(pc &= \ell_2 \land y \geq z \land x \leq y \lor \\
& \quad pc = \ell_3 \land y \geq z \land x \geq y, \rho_R)
= (post(pc = \ell_2 \land y \geq z \land x \leq y, \rho_R) \lor \\
& \quad post(pc = \ell_3 \land y \geq z \land x \geq y, \rho_R))
= (post(pc = \ell_2 \land y \geq z \land x \leq y, \rho_2) \lor \\
& \quad post(pc = \ell_2 \land y \geq z \land x \leq y, \rho_3) \lor \\
& \quad post(pc = \ell_3 \land y \geq z \land x \geq y, \rho_4) \lor \\
& \quad post(pc = \ell_3 \land y \geq z \land x \geq y, \rho_5))
= (pc = \ell_2 \land y \geq z \land x \leq y \lor \\
& \quad pc = \ell_3 \land y \geq z \land x = y \lor \\
& \quad pc = \ell_4 \land y \geq z \land x \geq y)
\end{align*}
\]
compute $\varphi_{reach}$ for example program

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

\[ pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_2 \land y \geq z \land x \leq y \lor pc = l_3 \land y \geq z \land x \geq y \lor \]
\[ pc = l_2 \land y \geq z \land x \leq y \lor pc = l_3 \land y \geq z \land x = y \lor \]
\[ pc = l_4 \land y \geq z \land x \geq y \]

disjunction in a logically equivalent, simplified form:

\[ pc = l_1 \lor \]
\[ pc = l_2 \land y \geq z \lor \]
\[ pc = l_3 \land y \geq z \land x \geq y \land \]
\[ pc = l_4 \land y \geq z \land x \geq y \]

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