correctness proof via forward derivation

- given a Hoare triple \( \{ \phi \} C \{ \psi \} \),
correctness proof via forward derivation

- given a Hoare triple $\{\phi\} C \{\psi\}$,
- construct a forwards derivation
correctness proof via forward derivation

- given a Hoare triple \( \{ \phi \} C \{ \psi \} \),
- construct a *forwards derivation*
- derivation = sequence of Hoare triples,
  each Hoare triple is an axiom (skip, update)
  or it is inferred by one of the inference rules (seq, cond, while)
correctness proof via forward derivation

- given a Hoare triple \( \{ \phi \} C \{ \psi \} \),
- construct a forwards derivation
- derivation = sequence of Hoare triples,
  each Hoare triple is an axiom (skip, update)
  or it is inferred by one of the inference rules (seq, cond, while)
- Hoare triples with \( \psi \) and strongest postcondition
  for larger and larger program fragments
correctness proof via forward derivation

- given a Hoare triple \( \{ \phi \} \ C \{ \psi \} \),
- construct a *forwards derivation*
- derivation = sequence of Hoare triples,
  each Hoare triple is an axiom (skip, update)
  or it is inferred by one of the inference rules (seq, cond, while)
- Hoare triples with \( \psi \) and *strongest postcondition*
  for larger and larger program fragments
- verification condition:
  strongest postcondition of \( \phi \) under \( C \) entails \( \psi \)
  (+ special treatment of while)
strongest postcondition post(\( C, \psi \))

- post(skip, \( \psi \)) \equiv
strongest postcondition $\text{post}(C, \psi)$

- $\text{post}($skip$, \phi) \equiv \phi$
- $\text{post}(x := e, \phi) \equiv$
Strongest postcondition \( post(C, \psi) \)

- \( post(skip, \phi) \equiv \phi \)
- \( post(x := e, \phi) \equiv \phi[x_{old}/x] \land x = e[x_{old}/x] \)
- \( post(C_1 ; C_2, \phi) \)
strongest postcondition $\text{post}(C, \psi)$

- $\text{post}($skip, $\phi)$ $\equiv$ $\phi$
- $\text{post}(x := e, \phi)$ $\equiv$ $\phi[x_{\text{old}}/x] \land x = e[x_{\text{old}}/x]$
- $\text{post}(C_1 ; C_2, \phi)$ $\equiv$ $\text{post}(C_2, \text{post}(C_1, \phi))$
- $\text{post}(\text{if } b \text{ then } C_1 \text{ else } C_2, \phi)$ $\equiv$
strongest postcondition $\text{post}(C, \psi)$

- $\text{post}(\text{skip}, \phi) \equiv \phi$
- $\text{post}(x := e, \phi) \equiv \phi[x_{\text{old}}/x] \land x = e[x_{\text{old}}/x]$
- $\text{post}(C_1 ; C_2, \phi) \equiv \text{post}(C_2, \text{post}(C_1, \phi))$
- $\text{post}(\text{if } b \text{ then } C_1 \text{ else } C_2, \phi) \equiv$
  
  
  
  
  $\text{post}(C_1, b \land \phi) \lor \text{post}(C_2, \neg b \land \phi)$
- $\text{post}(\text{while } b \text{ do } \{\theta\} C_0, \phi) \equiv$
strongest postcondition \( \text{post}(C, \psi) \)

- \( \text{post}(\text{skip}, \phi) \equiv \phi \)
- \( \text{post}(x := e, \phi) \equiv \phi_{x := e}[x_{old}/x] \land x = e_{x := e}[x_{old}/x] \)
- \( \text{post}(C_1 ; C_2, \phi) \equiv \text{post}(C_2, \text{post}(C_1, \phi)) \)
- \( \text{post}(\text{if } b \text{ then } C_1 \text{ else } C_2, \phi) \equiv \text{post}(C_1, b \land \phi) \lor \text{post}(C_2, \neg b \land \phi) \)
- \( \text{post}(\text{while } b \text{ do } \{ \theta \} C_0, \phi) \equiv \theta \land \neg b \)

next:
static analysis constructs candidate for \( \theta \) via forward analysis
“reachability analysis”
program code for specifications

validity of Hoare triple:

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

≡ safety of program:

assume(y \geq z);
while (x < y) {
    x++;
}
assert(x \geq z);
program with \texttt{assume()} and \texttt{assert()}

\begin{itemize}
\item \texttt{assume(e) \equiv if e then skip else halt}
\end{itemize}
program with assume () and assert ()

- assume \((e)\) \(\equiv\) if \(e\) then skip else halt
- assert \((e)\) \(\equiv\) if \(e\) then skip else error
program with assume () and assert ()

- assume (e) ≡ if e then skip else halt
- assert (e) ≡ if e then skip else error
- generalize partial correctness:
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*: correctness of program wrt. Hoare triple:

\[ \{\phi\} C \{\psi\} \]

\(\equiv\)
program with \textbf{assume} () and \textbf{assert} ()

- \textbf{assume} \((e)\) \(\equiv\) if \(e\) then skip else halt
- \textbf{assert} \((e)\) \(\equiv\) if \(e\) then skip else \textbf{error}
- generalize \textit{partial correctness}:
  correctness of program wrt. Hoare triple:

\[
\{\phi\} C \{\psi\}
\]

\(\equiv\) \textit{safety of program}: \textbf{assume} \((\phi)\) ; \(C\) ; \textbf{assert} \((\psi)\)

\textbf{safety} = non-reachability of \textbf{error}
(no execution of \textbf{error} branch)
control flow graph

source code

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

control flow graph

$\ell_1$ \rightarrow \ell_2$

$\ell_2$ \rightarrow $\ell_3$

$\ell_3$ \rightarrow $\ell_4, \ell_5$

$\ell_4$ \rightarrow assume(x >= y)

$\ell_5$ \rightarrow assume(x < z)
control flow graph

source code

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

control flow graph

encode transition as logical formula
assume( y >= z)  ~→

source code

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

encode transition as logical formula

assume( y >= z)  $\iff$  $y \geq z$

x++  $x' = x + 1$
source code

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

control flow graph

encode transition as logical formula

assume( y >= z)  y >= z
x++  x' = x + 1
control flow graph

source code

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

encode transition as logical formula

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

\[ \rho_1 = (\text{move}(\ell_1, \ell_2) \land y \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_2 = (\text{move}(\ell_2, \ell_2) \land x + 1 \leq y \land x' = x + 1 \land \text{skip}(y, z)) \]
\[ \rho_3 = (\text{move}(\ell_2, \ell_3) \land x \geq y \land \text{skip}(x, y, z)) \]
\[ \rho_4 = (\text{move}(\ell_3, \ell_4) \land x \geq z \land \text{skip}(x, y, z)) \]
\[ \rho_5 = (\text{move}(\ell_3, \ell_5) \land x + 1 \leq z \land \text{skip}(x, y, z)) \]
transition relation $\rho$ expressed by logica formula

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

abbreviations:

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

- $V$ - finite tuple of program variables
- $pc$ - program counter variable (pc included in $V$)
- $\varphi_{init}$ - initiation condition given by formula over $V$
- $\mathcal{R}$ - a finite set of transition relations
- $\varphi_{err}$ - 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 $V'$
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
- $\Sigma$ = 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 $V = \text{set of program states}$
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
- 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 \( V' \)
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
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 $\subseteq$
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 $y \geq z =$ set of program states in which the value of
  the variable $y$ is greater than the value of $z$

- logical consequence: $y \geq z \models y + 1 \geq z$
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,
  = set of pairs of program states $(s_1, s_2)$ in which the value of
  the variable $y$ in the second state $s_2$ is greater than the value
  of $z$ 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, set of pairs of program states \((s_1, s_2)\) in which the value of the variable \( y \) in the second state \( s_2 \) is greater than the value of \( z \) 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: 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, set of pairs of program states $(s_1, s_2)$ in which the value of the variable $y$ in the second state $s_2$ is greater than the value of $z$ 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 $P = (V, pc, \varphi_{init}, R, \varphi_{err})$

- program variables $V = (pc, x, y, z)$
- program counter $pc$
- program variables $x$, $y$, and $z$ range over integers
- set of control locations $L = \{\ell_1, \ldots, \ell_5\}$
- initiation condition $\varphi_{init} = (pc = pc = \ell_1)$
- error condition $\varphi_{err} = (pc = pc = \ell_5)$
- program transitions $R = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$

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

\[\begin{align*}
\ell_1 & \xrightarrow{y \geq z} \ell_2 \\
\ell_2 & \xrightarrow{x < y \land x' = x + 1} \ell_3 \\
\ell_3 & \xrightarrow{x \geq y} \ell_4 \\
\ell_4 & \xrightarrow{x < z} \ell_5 \\
\ell_5 & \xrightarrow{x \geq z} \ell_4
\end{align*}\]

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

- each state that satisfies the initiation condition $\varphi_{\text{init}}$ is called an initial state
- each state that satisfies the error condition $\varphi_{\text{err}}$ is called an 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 $\mathcal{V}_{reach}$,

$$\mathcal{V}_{err} \wedge \mathcal{V}_{reach} \models false.$$  

where $\mathcal{V}_{reach} =$ 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:

\[ \varphi_{reach} = (pc = \ell_1 \lor \]
\[ pc = \ell_2 \land y \geq z \lor \]
\[ pc = \ell_3 \land y \geq z \land x \geq y \lor \]
\[ pc = \ell_4 \land y \geq z \land x \geq y) \]
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:

$$\text{post}(\varphi, \rho_1 \lor \rho_2) = (\text{post}(\varphi, \rho_1) \lor \text{post}(\varphi, \rho_2))$$
$$\text{post}(\varphi_1 \lor \varphi_2, \rho) = (\text{post}(\varphi_1, \rho) \lor \text{post}(\varphi_2, \rho))$$