Recap: While programs with assertions

**arithmetic expressions**, which denote integer values

\[
a ::= k | x | a_1 + a_2 | a_1 - a_2 | a_1 * a_2 | a_1 / a_2
\]

\(k\) is an integer constant, \(x\) is a variable

**Boolean expressions**

\[
b ::= a_1 = a_2 | a_1 < a_2 | \neg b | b_1 \land b_2 | b_1 \lor b_2 | \text{true}
\]

**programs**

\[
c ::= \text{[skip]} | [x := a] | [\text{read}(x)] | [\text{print}(a)] | c_1 ; c_2
\]

\[
| \text{if } [b] \text{ then } \{c_1\} \text{ else } \{c_2\}
\]

\[
| \text{while } [b] \text{ do } \{c\} | [\text{assert}(b)]
\]

\(x\) is a variable, \(a\) is arithmetic expression, \(b\) is a Boolean expression
Recap: Assert statements

\texttt{assert}(b) aborts the execution if \( b \) is false, and has no effect otherwise. We will only insert it at where it is guaranteed to hold.

The definition of transfer functions for \texttt{assert} blocks depends on the analysis problem. Often, insight into the analysis problem is required to define non-trivial and sound constraints for \texttt{assert}.

Idea: use assertions as filters that only let valid information pass
Recap: Constant propagation analysis with assertions
extend the evaluation function $eval$ to boolean expressions: maps a boolean expression $b$ and $d \in D \setminus \{\bot\}$ to an element of $\mathbb{B}$

- if $b$ is true, then $eval(b, d) := true$,

- if $b$ is $a_1 = a_2$, then $eval(b, d) := true$, if $eval(a_1, d) = eval(a_2, d)$, and $eval(b, d) := false$, otherwise

- if $b$ is $\neg b'$, then $eval(b, d) := true$, if $eval(b', d) = false$, and $eval(b, d) := false$, otherwise

- if $b$ is $b_1 \land b_2$, then $eval(b, d) := true$, if $eval(b_1, d) = eval(b_2, d) = true$, and $eval(b, d) := false$, otherwise

Remark For simplicity, the definition of $eval(a_1 = a_2)$ assumes that the value of both expressions $a_1$ and $a_2$ is defined (this holds for expressions that do not contain division). To handle operations with undefined value (e.g., division by zero) the range of $eval$ should be extended with the value $\top$, and the value $\bot$ should be taken into consideration in the transfer function for $assert$. 
Recap: Constant propagation analysis with assertions

for \([\text{assert}(b)]^l\) we define the transfer function \(f_l : D \rightarrow D\)

\[
f_l(d) = \bot \quad \text{if } d = \bot
\]

\[
f_l(d) = \bot \quad \text{if } \neg(\exists d' \in \alpha(d) : \text{eval}(b, d') = \text{true})
\]

\[
f_l(d)(v) = n \quad \text{if } \forall d' \in \alpha(d) : \text{eval}(b, d') = \text{true} \Rightarrow d'(v) = n
\]

\[
f_l(d)(v) = ? \quad \text{otherwise}
\]

where \(\alpha(d)\) is the set of functions from \(Vars\) to \(\mathbb{Z}\) such that \(d' \in \alpha(d)\) if and only if for all \(x \in Vars\) it holds that

\[
d'(x) \in \begin{cases} 
\{n\} & \text{if } d(x) = n \in \mathbb{Z} \\
\mathbb{Z} & \text{if } d(x) = ?
\end{cases}
\]
Example: Constant propagation analysis with assertions

\[ x := 0 \]
\[ x \geq 0 \]
\[ \text{assert}(x \geq 0) \]
\[ \text{assert}(x < 0) \]
\[ y := 0 \]
\[ y := 1 \]
\[ \text{skip} \]

\((d_x, d_y)\): represents function \(d\) with \(d(x) = d_x\) and \(d(y) = d_y\)

Consider \(d = (0, ?)\)

\(\alpha(d) = \{(0, m) \mid m \in \mathbb{Z}\}\)

d' \in \alpha(d) if and only if

for all \(v \in \{x, y\}\) it holds that

\(d'(v) \in \begin{cases} \{n\} & \text{if } d(v) = n \in \mathbb{Z} \\ \mathbb{Z} & \text{if } d(v) = ? \end{cases}\)
Example: Constant propagation analysis with assertions

\[(d_x, d_y): \text{ represents function } d \text{ with } d(x) = d_x \text{ and } d(y) = d_y\]

consider \(d = (0, ?)\)

\(\alpha(d) = \{(0, m) \mid m \in \mathbb{Z}\}\)

\(\forall d' \in \alpha(d) : eval(x \geq 0, d') = \text{true}\)

\(\forall d' \in \alpha(d) : eval(x < 0, d') = \text{false}\)

\(f_3(d)(x) = 0, f_3(d)(y) = ?\)

\(f_4(d) = \bot\)

\(f_5(f_3(d)) = f_5((0, ?)) = (0, 0)\)

\(f_6(f_4(d)) = f_6(\bot) = \bot\)
Correlations between branches

\[
\text{if } [x \geq 0]^{1}\{
\]
\[
\begin{align*}
&[\text{assert}(x \geq 0)]^{2}; \\
&[\text{open()}]^{3}; \\
&[\text{flag} := 1]^{4}; \\
\}
\]

\[
\text{else } \{
\]
\[
\begin{align*}
&[\text{assert}(x < 0)]^{5}; \\
&[\text{flag} := 0]^{6}; \\
\}
\]

\[
\text{if } [\neg(\text{flag} = 0)]^{7}\{
\]
\[
\begin{align*}
&[\text{assert}(\text{flag} \neq 0)]^{8}; \\
&[\text{close()}]^{9}; \\
\}
\]

\[
\text{else } \{
\]
\[
\begin{align*}
&[\text{assert}(\text{flag} = 0)]^{10}; \\
&[\text{skip}]^{11}; \\
\}
\]

\[
[\text{skip}]^{12}; \\
\]

- open() and close() are functions for opening and closing a specific file
- the file is initially closed
- Goal: develop an analysis that allows us to check whether close() is called only if open() was called before
- Forward may analysis to determine the possible status of the file (open or closed)
Example: first attempt

\[ x \geq 0 \]

\[ \text{assert}(x \geq 0) \]

\[ \text{assert}(x < 0) \]

\[ \text{open}() \]

\[ \text{flag} := 0 \]

\[ \text{flag} := 1 \]

\[ \neg(\text{flag} = 0) \]

\[ \text{assert}((\text{flag} \neq 0)) \]

\[ \text{assert}((\text{flag} = 0)) \]

\[ \text{close}() \]

\[ \text{skip} \]

\[ \text{skip} \]

\[ \neg(\text{flag} = 0) \]

\[ \text{assert}(\text{flag} = 0) \]

\[ \text{open}() \]

\[ \text{closed} \]

\[ \text{assert}((\text{flag} \neq 0)) \]

\[ \text{assert}(\text{flag} = 0) \]

\[ \text{close}() \]

\[ \text{skip} \]

\[ \text{skip} \]

\[ \text{assert}(\text{flag} = 0) \]

\[ \text{assert}(\text{flag} = 0) \]

\[ \text{close}() \]

\[ \text{skip} \]

\[ \text{skip} \]

\[ \text{lattice: } (2\{\text{open, closed}\}, \subseteq) \]

\[ \text{initial value: } i = \{\text{closed}\} \]

\[ \text{transfer functions} \]

\[ f_b(d) = d \text{ for } b \neq 3, 9 \]

\[ f_3(d) = \{\text{open}\} \]

\[ f_9(d) = \{\text{closed}\} \]

\[ X_7^{LFP} = \{\text{open}\} \cup \{\text{closed}\} \]

\[ X_9^{LFP} = \{\text{open, closed}\} \]

Program does not pass the check.
Example: second attempt

lattice:
\((2\{\text{open}, \text{close}\}, \subseteq) \times (2\{\text{flag}=0, \text{flag} \neq 0\}, \subseteq)\)

initial value:
i = (\{\text{closed}\}, \{\text{flag} = 0, \text{flag} \neq 0\})

transfer functions
\(f_b(d) = d\) for \(b \neq 3, 4, 6, 8, 9, 10\)
\(f_3((d_1, d_2)) = (\{\text{open}\}, d_2)\)
\(f_9((d_1, d_2)) = (\{\text{closed}\}, d_2)\)
\(f_4((d_1, d_2)) = (d_1, \{\text{flag} \neq 0\})\)
\(f_6((d_1, d_2)) = (d_1, \{\text{flag} = 0\})\)

least fixpoint solution
\(X_7^{LFP} = (\{\text{open, closed}\}, \{\text{flag} = 0, \text{flag} \neq 0\})\)
\(X_9^{LFP} = (\{\text{open, closed}\}, \{\text{flag} \neq 0\})\)
Relational analysis: lattice

lattice: \( (2^{\{\text{flag}=0, \text{flag} \neq 0\} \times \{\text{open}, \text{close}\}), \subseteq}) \)

\( \{\text{flag} = 0, \text{flag} \neq 0\} \) is set of path contexts. A path context carries information about the path leading to the program point. The resulting lattice captures relations between variables.

initial value:
\( i = \{(\text{flag} = 0, \text{closed}), (\text{flag} \neq 0, \text{closed})\} \)
Relational analysis: transfer functions

- for $\text{open()}^l$
  
  \[ f_b(d) = \{(c, \text{open}) \mid \exists t : (c, t) \in d\} \]

- for $\text{close()}^l$
  
  \[ f_b(d) = \{(c, \text{closed}) \mid \exists t : (c, t) \in d\} \]

- for $\text{flag} := 0^l$
  
  \[ f_b(d) = \{(\text{flag} = 0, t) \mid \exists c : (c, t) \in d\} \]

- for $\text{flag} := k^l$ for a constant $k \in \mathbb{Z} \setminus \{0\}$
  
  \[ f_b(d) = \{(\text{flag} \neq 0, t) \mid \exists c : (c, t) \in d\} \]

- for $\text{flag} := a^l$ for a non-constant expression $a$ or $\text{read(flag)}^l$
  
  \[ f_b(d) = \{(\text{flag} = 0, t), (\text{flag} \neq 0, t) \mid \exists c : (c, t) \in d\} \]
Relational analysis: transfer functions

- for \([\text{assert}(\text{flag} = 0)]^l\)
  
  \[ f_b(d) = \{(\text{flag} = 0, t) \mid (\text{flag} = 0, t) \in d\} \]

- for \([\text{assert}(\text{flag} \neq 0)]^l\)
  
  \[ f_b(d) = \{(\text{flag} \neq 0, t) \mid (\text{flag} \neq 0, t) \in d\} \]

- for all remaining cases
  
  \[ f_b(d) = d \]

Remark: The transfer functions could be made more precise.
Example: relational analysis

Example: relational analysis

least fixpoint solution

$X_7^{LFP} = \{(\text{flag} \neq 0, \text{open}), (\text{flag} = 0, \text{closed})\}$

$X_9^{LFP} = \{(\text{flag} \neq 0, \text{open})\}$

With this information we can establish that in this program close() is called only when the file was opened.
Interval analysis

**Goal:** Compute for each program point a lower and an upper bound for the possible values of each variable.

**Application:** checking array bounds and numerical overflow

**Classification:** forward may analysis
Interval analysis: lattice

Let $I = \{[l, h] \mid l \leq h, l \in \mathbb{Z} \cup \{-\infty\}, h \in \mathbb{Z} \cup \{\infty\}\}$, where $\leq$ on $\mathbb{Z}$ is extended to an ordering on $\mathbb{Z} \cup \{-\infty, \infty\}$ by setting $-\infty \leq z, z \leq \infty$ and $-\infty \leq \infty$ for all $z \in \mathbb{Z}$.

Let $\inf([l, h]) = l$ and $\sup([l, h]) = h$. Let $[l_1, h_1] \sqsubseteq [l_2, h_2]$ iff

$$\inf(l_2, h_2) \leq \inf(l_1, h_1) \text{ and } \sup(l_1, h_1) \leq \sup(l_2, h_2).$$

We consider $(D, \leq)$, where $((\text{Vars} \to I) \cup \{\bot\}, \leq)$, and the partial ordering $\leq$ on $D$ is defined by

- $\bot \leq d$ for all $d \in D$,
- $d_1 \leq d_2$ iff for every $v \in \text{Vars}$ we have $d_1(v) \sqsubseteq d_2(v)$. 
(\(D, \leq\)) defined on the previous slide is a complete lattice:

\[
\bigcup Y = \begin{cases} 
\bot & \text{if } Y \subseteq \{\bot\}, \\
\bigwedge \{d' \mid d' \in Y \setminus \{\bot\}\} & \text{otherwise},
\end{cases}
\]

where for each \(v \in \text{Vars}\) is defined by

\[
d(v) = \left[\inf' \{\inf(d'(v)) \mid d' \in Y \setminus \{\bot\}\}, \sup' \{\sup(d'(v)) \mid d' \in Y \setminus \{\bot\}\}\right],
\]

where \(\inf'\) and \(\sup'\) are infimum and supremum operators on \(\mathbb{Z} \cup \{-\infty, \infty\}\) corresponding to the ordering \(\leq\) on \(\mathbb{Z} \cup \{-\infty, \infty\}\):

- \(\inf'(\emptyset) = \infty\), \(\inf'(Z)\) is the least element of \(Z\) or \(-\infty\)
- \(\sup'(\emptyset) = -\infty\), \(\sup'(Z)\) is the greatest element of \(Z\) or \(+\infty\)
Interval analysis: lattice (for a single variable)

\[ [-\infty, \infty] \]

\[ \begin{array}{ccc}
[-\infty, 1] & \supset & [-1, \infty] \\
[-\infty, 0] & \supset & [-2, 2] \\
[-\infty, -1] & \supset & [0, \infty] \\
[-2, 0] & \supset & [1, \infty] \\
[-2, -1] & \supset & [1, 2] \\
[-2, -2] & \supset & [2, 2] \\
\end{array} \]
Interval analysis: lattice

\[ [0, 0] \subseteq [0, 1] \subseteq [0, 2] \subseteq [0, 3] \subseteq [0, 4] \subseteq [0, 5] \ldots \]

The interval lattice \((\mathcal{D}, \leq)\) does not satisfy (ACC).
Interval analysis: transfer functions

We first define the function $eval$ that maps each arithmetic expression $a$ and $d \in D \setminus \{\bot\}$ to an element of $I$.

- if $a$ is a variable $v$, then $eval(a, d) := d(v)$
- if $a$ is a constant $k$, then $eval(a, d) := [k, k]$
- if $a = a_1 \otimes a_2$, then
  - if $eval(a_1, d) = [l_1, h_1]$, and $eval(a_2, d) = [l_2, h_2]$, and for some $z_1 \in [l_1, h_1]$ and $z_2 \in [l_2, h_2]$ the value of $z_1 \otimes z_2$ is undefined, then $eval(a, d) := [-\infty, \infty]$;
  - otherwise $eval(a, d) := eval(a_1, d) \hat{\otimes} eval(a_2, d)$:

\[ [l_1, h_1] \hat{\otimes} [l_2, h_2] := [l, h], \text{ where} \]

\[ l = \inf' \{z_1 \otimes z_2 \mid z_1 \in [l_1, h_1], z_2 \in [l_2, h_2]\} \]
\[ h = \sup' \{z_1 \otimes z_2 \mid z_1 \in [l_1, h_1], z_2 \in [l_2, h_2]\} \]
For $d : Vars \rightarrow I$, $x, v \in Vars$, and $[l, h] \in I$, define the function $d[x \mapsto [l, h]] : Vars \rightarrow I$, such that

$$d[x \mapsto [l, h]](v) = \begin{cases} [l, h] & \text{if } v = x, \\ d(v) & \text{otherwise.} \end{cases}$$
Interval analysis: transfer functions

- for $[x := a]^b$ let

  $$f_b(d) := \begin{cases} \perp & \text{if } d = \perp, \\ d[x \mapsto \text{eval}(a, d)] & \text{otherwise} \end{cases}$$

- for $[\text{read}(x)]^b$ let

  $$f_b(d) := \begin{cases} \perp & \text{if } d = \perp, \\ d[x \mapsto [-\infty, \infty]] & \text{otherwise} \end{cases}$$

- for $[\text{print}(a)]^b$, for $[\text{skip}]^b$, and for condition $[e]^b$ let

  $$f_b(d) = d \text{ for all } d \in D$$
Interval analysis: example

The iterative algorithm for least fixpoint computation may not terminate.

For the program point after the loop (entry of block 7) the iteration starting from \((\bot, \ldots, \bot)\) gives

\[
\begin{align*}
\bot \\
\ldots \\
(x \mapsto [7, 7], y \mapsto [0, 0]) \\
\ldots \\
(x \mapsto [7, 8], y \mapsto [0, 1]) \\
\ldots \\
(x \mapsto [7, 8], y \mapsto [0, 2]) \\
\ldots \\
(x \mapsto [7, 8], y \mapsto [0, 3]) \\
\ldots
\end{align*}
\]
Non-ACC domains

Let $f$ be a monotonic function.

When the lattice does not satisfy (ACC), the sequence

$$f^0(\bot), f^1(\bot), f^2(\bot), \ldots$$

need not stabilize and its least upper bound does may not be $\text{lfp}(f)$

**Solution:** (soundly) approximate the least fixpoint by using **widening operators** to enforce termination of the iteration.
Widening operators

Let \((D, \leq)\) be a complete lattice. A function \(\nabla : D \times D \rightarrow D\) is a **widening operator** if it satisfies the following conditions:

1. for every \(d_1, d_2 \in D\),

\[
d_1 \sqcup d_2 \leq d_1 \nabla d_2
\]

2. for every ascending chain \(d_0 \leq d_1 \leq \ldots\) the ascending chain \(d_0 \nabla \leq d_1 \nabla \leq \ldots\), where \(d_0 \nabla = d_0, d_{i+1} \nabla = d_i \nabla \nabla d_{i+1}\), eventually stabilizes: there exists \(n \in \mathbb{N}\) such that \(d_i \nabla = d_n \nabla\) for all \(i \geq n\).

**Remark:** \((d_i \nabla)_{i \in \mathbb{N}}\) is clearly an ascending chain because

\[
d_{i+1} \nabla = d_i \nabla \nabla d_{i+1} \geq d_i \nabla \sqcup d_{i+1} \geq d_i \nabla
\]
Widening operators

Let \((D, \leq)\) be a complete lattice. A function \(\nabla : D \times D \rightarrow D\) is a **widening operator** if it satisfies the following conditions:

1. for every \(d_1, d_2 \in D\),
   \[
   d_1 \sqcup d_2 \leq d_1 \nabla d_2
   \]

2. for every ascending chain \(d_0 \leq d_1 \leq \ldots\) the ascending chain \(d_0^{\nabla} \leq d_1^{\nabla} \leq \ldots\), where \(d_0^{\nabla} = d_0\), \(d_{i+1}^{\nabla} = d_i^{\nabla} \nabla d_{i+1}\), eventually stabilizes: there exists \(n \in \mathbb{N}\) such that \(d_i^{\nabla} = d_n^{\nabla}\) for all \(i \geq n\).

**Soundness:** The requirement \(d_1 \sqcup d_2 \leq d_1 \nabla d_2\) guarantees soundness.
Interval analysis with widening

Let $K \subseteq \mathbb{Z} \cup \{-\infty, \infty\}$ be a finite set of consisting of constants appearing in the program and $-\infty$ and $\infty$. We define $\nabla$ as follows:

- $\bot \nabla d := d$, $\nabla \bot := d$,
- otherwise, for all $v \in \text{Vars}$, if $d_1(v) = [l_1, h_1]$ and $d_2(v) = [l_2, h_2]$, then $d_1 \nabla d_2(v) := [l, h]$, where
  
  $$
  l := \begin{cases} 
  l_1 & \text{if } l_1 \leq l_2, \\
  k & \text{if } l_2 < l_1, k = \max\{k \in K \mid k \leq l_2\} \\
  -\infty & \text{if } l_2 < l_1, \forall k \in K : l_2 < k
  \end{cases}
  $$

  $$
  h := \begin{cases} 
  h_1 & \text{if } h_2 \leq h_1, \\
  k & \text{if } h_1 < h_2, k = \min\{k \in K \mid h_2 \leq k\} \\
  \infty & \text{if } h_1 < h_2, \forall k \in K : k < h_2
  \end{cases}
  $$

$\nabla$ is a widening operator.
Interval analysis with widening: example

\[
\begin{align*}
[x & := 7]^1; \\
[y & := 0]^2; \\
\textbf{while } [\text{true}]^3 \textbf{ do } \{ \\
[x & := 7]^4; \\
[x & := x + 1]^5; \\
[y & := y + 1]^6 \\
\} \\
\textbf{print}(x)^7
\end{align*}
\]

For the program point after the loop (entry of block 7) the iteration starting from \((\bot, \ldots, \bot)\) using widening based on the set \(K = \{-\infty, 0, 1, 7, \infty\}\) gives

\[
\begin{align*}
\bot \\
& \ldots \\
(x & \mapsto [7, 7], y \mapsto [0, 0]) \\
& \ldots \\
(x & \mapsto [7, \infty], y \mapsto [0, 1]) \\
& \ldots \\
(x & \mapsto [7, \infty], y \mapsto [0, 7]) \\
& \ldots \\
(x & \mapsto [7, \infty], y \mapsto [0, \infty]) \\
& \ldots
\end{align*}
\]
Properties of widening operators

Let $f$ be a monotonic function on a complete lattice $(D, \leq)$ and $
abla : D \times D \to D$ be a widening operator. We define $f^0_\nabla(\perp) = \perp$ and for $i \in \mathbb{N}$,

$$f^{i+1}_\nabla(\perp) := \begin{cases} f^i_\nabla(\perp) & \text{if } f(f^i_\nabla(\perp)) \leq f^i_\nabla(\perp), \\ f^i_\nabla(\perp) \nabla f(f^i_\nabla(\perp)) & \text{otherwise} \end{cases}$$

- $(f^i_\nabla(\perp))_{i \in \mathbb{N}}$ is an ascending chain.
- $(f^i_\nabla(\perp))_{i \in \mathbb{N}}$ eventually stabilizes.
- $\bigcup\{f^i_\nabla(\perp) \mid i \in \mathbb{N}\} \geq \text{lfp}(f)$. 

24/27
Narrowing

Problem: Results obtained using widening are often imprecise.

Idea: Use narrowing to improve the precision of the result.

Recall that \( \bigcup \{ f_i^\nabla(\bot) \mid i \in N \} \geq \text{lfp}(f) \).

Let \( l^\nabla = \bigcup \{ f_i^\nabla(\bot) \mid i \in N \} \). Then, for all \( i \in \mathbb{N} \)

\[
\text{lfp}(f) \leq f^{i+1}(l^\nabla) \leq f^i(l^\nabla) \leq l^\nabla
\]

Iterate \( f \) starting from \( l \) to improve the precision of the result.

Remarks:
- Narrowing may not terminate (if (DCC) does not hold).
- It is possible to stop after any step with sound result.
Interval analysis with widening and narrowing: example

\[\begin{align*}
[x := 7] &; \\
[y := 0] &;
\end{align*}\]

\textbf{while} [true] \textbf{do} \{ \\
\begin{align*}
[x := 7] &; \\
[x := x + 1] &; \\
[y := y + 1] &;
\end{align*}\}

\textbf{print}(x)

For the program point after the loop (entry of block 7) the iteration starting from \((\bot, \ldots, \bot)\) using widening based on the set \(K = \{-\infty, 0, 1, 7, \infty\}\) gives

\((x \mapsto [7, \infty], y \mapsto [0, \infty])\)

Applying narrowing we get

\((x \mapsto [7, \infty], y \mapsto [0, \infty])\)

\(\ldots\)

\((x \mapsto [7, 8], y \mapsto [0, \infty])\)

\(\ldots\)

\((x \mapsto [7, 8], y \mapsto [0, \infty])\)
Example: the role of assert statements

[read\(x\)]\(^1\);
[y \(:=\) 0]\(^2\);
[z \(:=\) 0]\(^3\);

while \([x > 0]\)\(^4\) do 
{
[assert\((x > 0)\)]\(^5\);
[z \(:=\) z + x]\(^6\);
if \([y < 17]\)\(^7\) 
{
[assert\((y < 17)\)]\(^8\);
[y \(:=\) y + 1]\(^9\);
}
else 
{
[assert\((y \geq 17)\)]\(^10\);
[skip\)]\(^11\);
}
[x \(:=\) x – 1]\(^12\);
}

[assert\((x \leq 0)\)]\(^13\);
[skip\)]\(^14\);

Without the **assert** statements the approximate fixpoint iteration with widening based on the set \(K = \{ -\infty, 0, 1, 17, \infty \}\) yields

\([x \mapsto [-\infty, \infty], y \mapsto [0, \infty], z \mapsto [-\infty, \infty]]\)

for the program point right before block 14.

With the **assert** statements, the result of the analysis for this program point is

\([x \mapsto [-\infty, 0], y \mapsto [0, 17], z \mapsto [0, \infty]]\)