IA159 Formal Verification Methods

Static Analysis and Abstract Interpretation

Jan Strejček

Faculty of Informatics
Masaryk University

Focus and sources

Focus

- lattices and fixpoints
- static analysis
- abstract interpretation

Source

- P. Cousot and R. Cousot: *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, POPL 1977.

Special thanks to Marek Trtík for providing me his slides.

Motivation for static analysis

Floyd's conjecture

To prove static properties of program it is often sufficient to consider sets of states associated with each program point.

Examples

- to check safety properties (reachability of an error state), one only needs to know reachable states
- for many optimizations during compilation, static information is sufficient (e.g. detection of live variables, available expressions, etc.)

Motivation for static analysis

Operational semantics

- defines how a state changes along program execution
- it is concerned about computational sequences
- computes a function relating input and output states

Motivation for static analysis

Operational semantics

- defines how a state changes along program execution
- it is concerned about computational sequences
- computes a function relating input and output states

Static semantic

- observes which states pass which program location
- it is concerned about observed sets of states at locations
- computes a function assigning set of states to each program location

Motivation for abstract interpretation

- It is usually impossible to compute the sets of reachable states precisely
- we can compute them on some level of abstraction
- for example, instead with precise numbers we work only with abstract values $\{+, 0, -\}$
- abstraction brings some level of imprecission, for example, $15 - 17$ is seen as $(+) - (+)$, which can be $+, 0, -$

Lattices and fixpoints

Introduction to lattices

Let $(L, \leq)$ be a partially ordered set and $M \subseteq L$.

- $x \in L$ is an upper bound of M iff $y \leq x$ holds for all $y \in M$
- $x \in L$ is a lower bound of M iff $x \leq y$ holds for all $y \in M$
- supremum of M is the least upper bound of M
- infimum of M is the greatest lower bound of M
- $sup(M)$ and $inf(M)$ denote supremum and infimum of M, respectively

Introduction to lattices

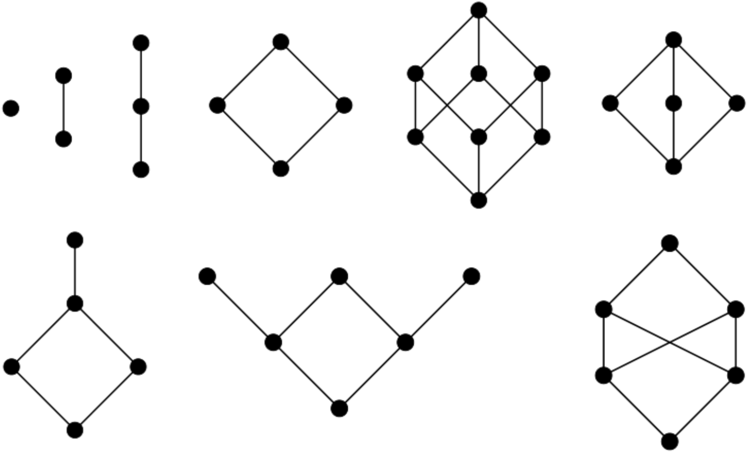
Let $(L, \leq)$ be a partially ordered set and $M \subseteq L$.

- $x \in L$ is an upper bound of M iff $y \leq x$ holds for all $y \in M$
- $x \in L$ is a lower bound of M iff $x \leq y$ holds for all $y \in M$
- supremum of M is the least upper bound of M
- infimum of M is the greatest lower bound of M
- $sup(M)$ and $inf(M)$ denote supremum and infimum of M, respectively

Definition (Complete lattice)

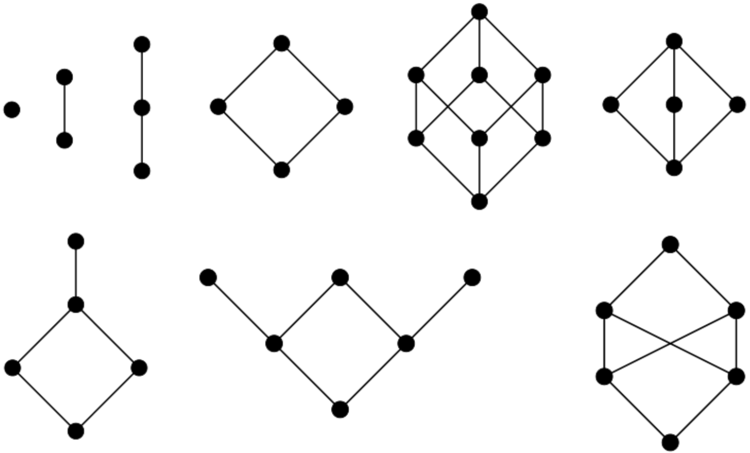
An ordered set $(L, \leq)$ is called complete lattice, if for each $M \subseteq L$ there exist both $sup(M)$ and $inf(M)$.

Introduction to lattices



Which of the partially ordered sets are complete lattices?

Introduction to lattices

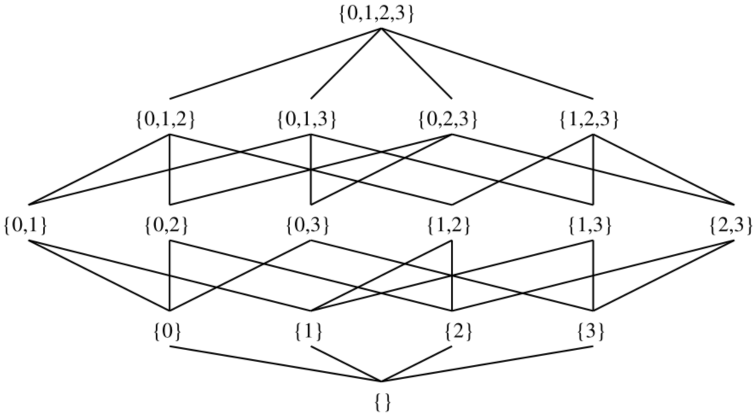


Which of the partially ordered sets are complete lattices?
(All of the top row and the left of the bottom row.)

Introduction to lattices

For every set S, the powerset $\mathcal{P}(S)$ with the partial order $\subseteq$ is a complete lattice.

For example, $(\mathcal{P}(\{0, 1, 2, 3\}), \subseteq)$ looks like:



Introduction to lattices

Let $(L, \leq)$ be a complete lattice.

- the greatest element $\top = sup(L)$ is called one of L
- the least element $\bot = inf(L)$ of L is called zero of L
- the lattice is of finite height if there exists $h \in \mathbb{N}$ such that the length of each strictly increasing chain of elements of L is less than or equal to h
- minimal such h is called lattice height

Fixpoint and Knaster-Tarski fixpoint theorem

Let $(L, \leq)$ be a complete lattice.

- a function $f : L \to L$ is monotone if for all $x, y \in L$ it holds

$$x \leq y \implies f(x) \leq f(y)$$

- $x \in L$ is called a fixpoint of f if $f(x) = x$

Fixpoint and Knaster-Tarski fixpoint theorem

Let $(L, \leq)$ be a complete lattice.

- a function $f : L \to L$ is monotone if for all $x, y \in L$ it holds

$$x \leq y \implies f(x) \leq f(y)$$

- $x \in L$ is called a fixpoint of f if $f(x) = x$

Theorem (Knaster-Tarski)

Let $(L, \leq)$ be a complete lattice and $f : L \to L$ be a monotone function. Then the set of fixpoints of f with partial order $\leq$ is also a complete lattice.

Kleene fixpoint theorem

Theorem (Kleene)

Let $(L, \leq)$ *be a complete lattice of finite height and* $f : L \to L$ *a monotone function. Then there exists* $n \in \mathbb{N}$ *such that for all* $k \in \mathbb{N}$ *it is* $f^n(\bot) = f^{n+k}(\bot)$ *and* $f^n(\bot)$ *is the least fixpoint of* f.

Kleene fixpoint theorem

Theorem (Kleene)

Let $(L, \leq)$ be a complete lattice of finite height and $f : L \to L$ a monotone function. Then there exists $n \in \mathbb{N}$ such that for all $k \in \mathbb{N}$ it is $f^n(\bot) = f^{n+k}(\bot)$ and $f^n(\bot)$ is the least fixpoint of f.

Proof: Since $\bot$ is the least element of L, we have $\bot \leq f(\bot)$. Since f is monotone, them $f(\bot) \leq f(f(\bot))$ and by induction $f^i(\bot) \leq f^{i+1}(\bot)$. Thus, we have a nondecreasing chain $\bot \leq f(\bot) \leq f^2(\bot) \leq \ldots$. Since L is assumed to be of a finite height, there must exist $n \in \mathbb{N}$ such that $f^n(\bot) = f^{n+1}(\bot)$. To show that $f^n(\bot)$ is a least fixpoint of f, let us assume x is another fixpoint of f. Since $\bot \leq x$ and $f(\bot) \leq f(x) = x$ from monotonicity of f, we get by induction $f^n(\bot) \leq x$. $\square$

Fixpoint computation

Algorithm for the least fixpoint computation

```
x := ⊥;
do { t := x; x := f(x); } while (x ≠ t);
```

If we start with `x := ⊤;`, we get the greatest fixpoint.

Product lattice

Lemma (Product lattice)

Let $(L_1, \leq_1), \ldots, (L_n, \leq_n)$ *be complete lattices and order* $\leq$ *on* $L_1 \times \ldots \times L_n$ *is defined as* $(x_1, \ldots, x_n) \leq (y_1, \ldots, y_n)$ *iff*

$$x_1 \leq_1 y_1 \ \wedge \ \ldots \ \wedge \ x_n \leq_n y_n.$$

Then $(L_1 \times \ldots \times L_n, \leq)$ *is a complete lattice.*

Fixpoints on product lattices

Let $(L, \leq)$ be a complete lattice and $(L^n, \sqsubseteq)$ be the corresponding product lattice. Further, let $F_1, \ldots, F_n : L^n \to L$ be monotone functions, i.e. $(x_1, \ldots, x_n) \sqsubseteq (y_1, \ldots, y_n)$ implies $F_i(x_1, \ldots, x_n) \leq F_i(y_1, \ldots, y_n)$ for each $1 \leq i \leq n$. Then the function $F : L^n \to L^n$ defined as

$$F(x_1, \ldots, x_n) = (F_1(x_1, \ldots, x_n), \ldots, F_n(x_1, \ldots, x_n))$$

is a monotone function in $(L^n, \sqsubseteq)$. Further, the least fixpoint of F is the least solution of the system

$$\begin{array}{rcl} x_1 & = & F_1(x_1, \ldots, x_n) \\ & \vdots & \\ x_n & = & F_n(x_1, \ldots, x_n) \end{array}$$

Fixpoint comutation of product lattices

Naive algorithm for fixpoint computation

$\vec{x}$:= $\vec{\bot}$;

do { $\vec{t}$:= $\vec{x}$; $\vec{x}$:= F($\vec{x}$); } while ($\vec{x} \neq \vec{t}$);

Fixpoint comutation of product lattices

Naive algorithm for fixpoint computation

$\vec{x}$ `:=` $\vec{\bot}$`;`
`do {` $\vec{t}$ `:=` $\vec{x}$`;` $\vec{x}$ `:= F(`$\vec{x}$`); } while (`$\vec{x} \neq \vec{t}$`);`

Better algorithm for fixpoint computation (faster convergence)

x_1 `:=` $\bot$`; ...` x_n `:=` $\bot$`;`
`do {`
  t_1 `:=` x_1`; ...` t_n `:=` x_n`;`
  x_1 `:=` $F_1(x_1, \ldots, x_n)$`;`
  $\vdots$
  x_n `:=` $F_n(x_1, \ldots, x_n)$`;`
`} while (`$x_1 \neq t_1 \vee \ldots \vee x_n \neq t_n$`);`

Moving to abstraction

Abstract interpretation

Abstract interpretation

- an abstract interpretation of a program is kind of a static semantic, where original data domains are replaced with abstract ones
- abstract data domain must constitute a complete lattice
- semantic of program instructions have to be changed as well: we define unique monotone function for each program instruction

Abstract interpretation: Definition

Definition (Abstract interpretation)

An abstract interpretation I of a program P with n program locations is a tuple

$$I = \langle L, \circ, \leq, \top, \bot, F \rangle$$

where $(L, \leq)$ is complete lattice, $\top$ and $\bot$ are one and zero of $(L, \leq)$, $\circ$ is equal either to join or meet operation, and F is a monotone function on product lattice $(L^n, \leq)$ defining the interpretation of basic instructions.

The meet operator is defined as $a \circ b = inf(\{a, b\})$, while the join operator is defined as $a \circ b = sup(\{a, b\})$.

Abstract interpretation: Definition

Definition (Abstract interpretation)

An abstract interpretation I of a program P with n program locations is a tuple

$$I = \langle L, \circ, \leq, \top, \bot, F \rangle$$

where $(L, \leq)$ is complete lattice, $\top$ and $\bot$ are one and zero of $(L, \leq)$, $\circ$ is equal either to join or meet operation, and F is a monotone function on product lattice $(L^n, \leq)$ defining the interpretation of basic instructions.

The meet operator is defined as $a \circ b = inf(\{a, b\})$, while the join operator is defined as $a \circ b = sup(\{a, b\})$.

Typically, $F(\vec{x}) = (F_1(\vec{x}), \ldots, F_n(\vec{x}))$, where each $F_i : L^n \to L$ defines effect of i-th program instruction.

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
  a := a+1;
  x := a+b;
}
```

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\texttt{a+b}, \texttt{a}\star\texttt{b}, \texttt{y>a+b}, \texttt{a+1}\}$

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
  a := a+1;
  x := a+b;
}
```

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\texttt{a+b}, \texttt{a}\star\texttt{b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(\mathit{AExprs}), \cap, \subseteq, \mathit{AExprs}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\texttt{a+b}, \texttt{a}\star\texttt{b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(\mathit{AExprs}), \cap, \subseteq, \mathit{AExprs}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(\mathit{AExprs}), \leq)$.

```
var x,y,z,a,b;        x1
z := a+b;             x2
y := a*b;             x3
while (y > a+b) {     x4
   a := a+1;          x5
   x := a+b;          x6
}
```

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\mathtt{a+b}, \mathtt{a*b}, \mathtt{y>a+b}, \mathtt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(\mathit{AExprs}), \cap, \subseteq, \mathit{AExprs}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(\mathit{AExprs}), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

$$x_1 = F_1(\vec{x}) = \emptyset$$
$$x_2 = F_2(\vec{x}) = (x_1 \cup \{\mathtt{a+b}\}) \setminus \emptyset$$
$$x_3 = F_3(\vec{x}) = (x_2 \cup \{\mathtt{a*b}\}) \setminus \{\mathtt{y>a+b}\}$$
$$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\mathtt{a+b}, \mathtt{y>a+b}\}$$
$$x_5 = F_5(\vec{x}) = (x_4 \cup \{\mathtt{a+1}\}) \setminus \mathit{AExprs}$$
$$x_6 = F_6(\vec{x}) = (x_5 \cup \{\mathtt{a+b}\}) \setminus \emptyset$$

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $AExprs = \{\texttt{a+b}, \texttt{a}\star\texttt{b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(AExprs), \cap, \subseteq, AExprs, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \dots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(AExprs), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

$x_1 = F_1(\vec{x}) = \emptyset$
$x_2 = F_2(\vec{x}) = (x_1 \cup \{\texttt{a+b}\}) \setminus \emptyset$
$x_3 = F_3(\vec{x}) = (x_2 \cup \{\texttt{a}\star\texttt{b}\}) \setminus \{\texttt{y>a+b}\}$
$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\texttt{a+b}, \texttt{y>a+b}\}$
$x_5 = F_5(\vec{x}) = (x_4 \cup \{\texttt{a+1}\}) \setminus AExprs$
$x_6 = F_6(\vec{x}) = (x_5 \cup \{\texttt{a+b}\}) \setminus \emptyset$

Direction: Forward

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $AExprs = \{\texttt{a+b}, \texttt{a*b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(AExprs), \cap, \subseteq, AExprs, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(AExprs), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
  a := a+1;
  x := a+b;
}
```

$$x_1 = F_1(\vec{x}) = \emptyset$$
$$x_2 = F_2(\vec{x}) = (x_1 \cup \{\texttt{a+b}\}) \setminus \emptyset$$
$$x_3 = F_3(\vec{x}) = (x_2 \cup \{\texttt{a*b}\}) \setminus \{\texttt{y>a+b}\}$$
$$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\texttt{a+b}, \texttt{y>a+b}\}$$
$$x_5 = F_5(\vec{x}) = (x_4 \cup \{\texttt{a+1}\}) \setminus AExprs$$
$$x_6 = F_6(\vec{x}) = (x_5 \cup \{\texttt{a+b}\}) \setminus \emptyset$$

Analysis: Must

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\texttt{a+b}, \texttt{a}\star\texttt{b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(\mathit{AExprs}), \cap, \subseteq, \mathit{AExprs}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(\mathit{AExprs}), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

$$x_1 = F_1(\vec{x}) = \emptyset$$
$$x_2 = F_2(\vec{x}) = (x_1 \cup \{\texttt{a+b}\}) \setminus \emptyset$$
$$x_3 = F_3(\vec{x}) = (x_2 \cup \{\texttt{a}\star\texttt{b}\}) \setminus \{\texttt{y>a+b}\}$$
$$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\texttt{a+b}, \texttt{y>a+b}\}$$
$$x_5 = F_5(\vec{x}) = (x_4 \cup \{\texttt{a+1}\}) \setminus \mathit{AExprs}$$
$$x_6 = F_6(\vec{x}) = (x_5 \cup \{\texttt{a+b}\}) \setminus \emptyset$$

Are all functions F_i monotone?

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $\mathit{AExprs} = \{\texttt{a+b}, \texttt{a*b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(\mathit{AExprs}), \cap, \subseteq, \mathit{AExprs}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(\mathit{AExprs}), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
  a := a+1;
  x := a+b;
}
```

$$x_1 = F_1(\vec{x}) = \emptyset$$
$$x_2 = F_2(\vec{x}) = (x_1 \cup \{\texttt{a+b}\}) \setminus \emptyset$$
$$x_3 = F_3(\vec{x}) = (x_2 \cup \{\texttt{a*b}\}) \setminus \{\texttt{y>a+b}\}$$
$$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\texttt{a+b}, \texttt{y>a+b}\}$$
$$x_5 = F_5(\vec{x}) = (x_4 \cup \{\texttt{a+1}\}) \setminus \mathit{AExprs}$$
$$x_6 = F_6(\vec{x}) = (x_5 \cup \{\texttt{a+b}\}) \setminus \emptyset$$

Proof F_4: Let $\vec{x}, \vec{y} \in \mathcal{P}^6(\mathit{AExprs})$ such that $\vec{x} \leq \vec{y}$. ...

Example: Available expressions

A nontrivial expression in a program is available at a program location if its current value has already been computed earlier in the execution.

Available expressions: $AExprs = \{\texttt{a+b}, \texttt{a*b}, \texttt{y>a+b}, \texttt{a+1}\}$
A.I.: $I = \langle \mathcal{P}(AExprs), \cap, \subseteq, AExprs, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_6(\vec{x}))\rangle$
Product lattice: $(\mathcal{P}^6(AExprs), \leq)$.

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
  a := a+1;
  x := a+b;
}
```

$$x_1 = F_1(\vec{x}) = \emptyset$$
$$x_2 = F_2(\vec{x}) = (x_1 \cup \{\texttt{a+b}\}) \setminus \emptyset$$
$$x_3 = F_3(\vec{x}) = (x_2 \cup \{\texttt{a*b}\}) \setminus \{\texttt{y>a+b}\}$$
$$x_4 = F_4(\vec{x}) = (x_3 \cap x_6) \cup \{\texttt{a+b}, \texttt{y>a+b}\}$$
$$x_5 = F_5(\vec{x}) = (x_4 \cup \{\texttt{a+1}\}) \setminus AExprs$$
$$x_6 = F_6(\vec{x}) = (x_5 \cup \{\texttt{a+b}\}) \setminus \emptyset$$

Then $x_3 \subseteq y_3$ and $x_6 \subseteq y_6$, which implies $(x_3 \cap x_6) \subseteq (y_3 \cap y_6)\ldots$

Example: Available expressions

After fixpoint computation ...

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

$x_1 = \emptyset$

$x_2 = \{\mathtt{a+b}\}$

$x_3 = \{\mathtt{a+b}, \mathtt{a*b}\}$

$x_4 = \{\mathtt{a+b}, \mathtt{y>a+b}\}$

$x_5 = \emptyset$

$x_6 = \{\mathtt{a+b}\}$

Solution: Minimal

Example: Available expressions

After fixpoint computation ...

```
var x,y,z,a,b;
z := a+b;
y := a*b;
while (y > a+b) {
   a := a+1;
   x := a+b;
}
```

$$x_1 = \emptyset$$
$$x_2 = \{\texttt{a+b}\}$$
$$x_3 = \{\texttt{a+b}, \texttt{a*b}\}$$
$$x_4 = \{\texttt{a+b}, \texttt{y>a+b}\}$$
$$x_5 = \emptyset$$
$$x_6 = \{\texttt{a+b}\}$$

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

$$\textit{Vars} = \{\texttt{x}, \texttt{y}, \texttt{z}\} \text{ and}$$
$$I = \langle \mathcal{P}(\textit{Vars}), \cup, \subseteq, \textit{Vars}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_{11}(\vec{x}))\rangle$$

```
var x,y,z;
x := input;
while (x>1) {
   y := x/2;
   if (y>3)
      x := x-y;
   z := x-4;
   if (z>0)
      x := x/2;
   z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

Product lattice is $(\mathcal{P}^{11}(\mathit{Vars}), \leq)$.

$$
\begin{aligned}
x_1 &= x_2 \setminus \{\mathtt{x},\mathtt{y},\mathtt{z}\} \\
x_2 &= x_3 \setminus \{\mathtt{x}\} \\
x_3 &= (x_4 \cup x_{11}) \cup \{\mathtt{x}\} \\
x_4 &= (x_5 \setminus \{\mathtt{y}\}) \cup \{\mathtt{x}\} \\
x_5 &= (x_6 \cup x_7) \cup \{\mathtt{y}\} \\
x_6 &= (x_7 \setminus \{\mathtt{x}\}) \cup \{\mathtt{x},\mathtt{y}\} \\
x_7 &= (x_8 \setminus \{\mathtt{z}\}) \cup \{\mathtt{x}\} \\
x_8 &= (x_9 \cup x_{10}) \cup \{\mathtt{z}\} \\
x_9 &= (x_{10} \setminus \{\mathtt{x}\}) \cup \{\mathtt{x}\} \\
x_{10} &= (x_3 \setminus \{\mathtt{z}\}) \cup \{\mathtt{z}\} \\
x_{11} &= \{\mathtt{x}\}
\end{aligned}
$$

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

Direction: Backward

$$
\begin{aligned}
x_1 &= x_2 \setminus \{\mathtt{x},\mathtt{y},\mathtt{z}\} \\
x_2 &= x_3 \setminus \{\mathtt{x}\} \\
x_3 &= (x_4 \cup x_{11}) \cup \{\mathtt{x}\} \\
x_4 &= (x_5 \setminus \{\mathtt{y}\}) \cup \{\mathtt{x}\} \\
x_5 &= (x_6 \cup x_7) \cup \{\mathtt{y}\} \\
x_6 &= (x_7 \setminus \{\mathtt{x}\}) \cup \{\mathtt{x},\mathtt{y}\} \\
x_7 &= (x_8 \setminus \{\mathtt{z}\}) \cup \{\mathtt{x}\} \\
x_8 &= (x_9 \cup x_{10}) \cup \{\mathtt{z}\} \\
x_9 &= (x_{10} \setminus \{\mathtt{x}\}) \cup \{\mathtt{x}\} \\
x_{10} &= (x_3 \setminus \{\mathtt{z}\}) \cup \{\mathtt{z}\} \\
x_{11} &= \{\mathtt{x}\}
\end{aligned}
$$

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

Analysis: May

$$
\begin{aligned}
x_1 &= x_2 \setminus \{\mathtt{x},\mathtt{y},\mathtt{z}\}\\
x_2 &= x_3 \setminus \{\mathtt{x}\}\\
x_3 &= (x_4 \cup x_{11}) \cup \{\mathtt{x}\}\\
x_4 &= (x_5 \setminus \{\mathtt{y}\}) \cup \{\mathtt{x}\}\\
x_5 &= (x_6 \cup x_7) \cup \{\mathtt{y}\}\\
x_6 &= (x_7 \setminus \{\mathtt{x}\}) \cup \{\mathtt{x},\mathtt{y}\}\\
x_7 &= (x_8 \setminus \{\mathtt{z}\}) \cup \{\mathtt{x}\}\\
x_8 &= (x_9 \cup x_{10}) \cup \{\mathtt{z}\}\\
x_9 &= (x_{10} \setminus \{\mathtt{x}\}) \cup \{\mathtt{x}\}\\
x_{10} &= (x_3 \setminus \{\mathtt{z}\}) \cup \{\mathtt{z}\}\\
x_{11} &= \{\mathtt{x}\}
\end{aligned}
$$

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

Solution: Minimal

$$
\begin{aligned}
X_1 &= \emptyset \\
X_2 &= \emptyset \\
X_3 &= \{\mathtt{x}\} \\
X_4 &= \{\mathtt{x}\} \\
X_5 &= \{\mathtt{x},\mathtt{y}\} \\
X_6 &= \{\mathtt{x},\mathtt{y}\} \\
X_7 &= \{\mathtt{x}\} \\
X_8 &= \{\mathtt{x},\mathtt{z}\} \\
X_9 &= \{\mathtt{x},\mathtt{z}\} \\
X_{10} &= \{\mathtt{x},\mathtt{z}\} \\
X_{11} &= \{\mathtt{x}\}
\end{aligned}
\qquad
\begin{aligned}
X_1 &= X_2 \smallsetminus \{\mathtt{x},\mathtt{y},\mathtt{z}\} \\
X_2 &= X_3 \smallsetminus \{\mathtt{x}\} \\
X_3 &= (X_4 \cup X_{11}) \cup \{\mathtt{x}\} \\
X_4 &= (X_5 \smallsetminus \{\mathtt{y}\}) \cup \{\mathtt{x}\} \\
X_5 &= (X_6 \cup X_7) \cup \{\mathtt{y}\} \\
X_6 &= (X_7 \smallsetminus \{\mathtt{x}\}) \cup \{\mathtt{x},\mathtt{y}\} \\
X_7 &= (X_8 \smallsetminus \{\mathtt{z}\}) \cup \{\mathtt{x}\} \\
X_8 &= (X_9 \cup X_{10}) \cup \{\mathtt{z}\} \\
X_9 &= (X_{10} \smallsetminus \{\mathtt{x}\}) \cup \{\mathtt{x}\} \\
X_{10} &= (X_3 \smallsetminus \{\mathtt{z}\}) \cup \{\mathtt{z}\} \\
X_{11} &= \{\mathtt{x}\}
\end{aligned}
$$

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Live variables

A variable is live at a program point if its current value may be read during the remaining execution of the program.

Variables y, z are never live together.

$$
\begin{array}{lll}
x_1 = \emptyset & x_1 = x_2 \smallsetminus \{\mathtt{x},\mathtt{y},\mathtt{z}\} \\
x_2 = \emptyset & x_2 = x_3 \smallsetminus \{\mathtt{x}\} \\
x_3 = \{\mathtt{x}\} & x_3 = (x_4 \cup x_{11}) \cup \{\mathtt{x}\} \\
x_4 = \{\mathtt{x}\} & x_4 = (x_5 \smallsetminus \{\mathtt{y}\}) \cup \{\mathtt{x}\} \\
x_5 = \{\mathtt{x},\mathtt{y}\} & x_5 = (x_6 \cup x_7) \cup \{\mathtt{y}\} \\
x_6 = \{\mathtt{x},\mathtt{y}\} & x_6 = (x_7 \smallsetminus \{\mathtt{x}\}) \cup \{\mathtt{x},\mathtt{y}\} \\
x_7 = \{\mathtt{x}\} & x_7 = (x_8 \smallsetminus \{\mathtt{z}\}) \cup \{\mathtt{x}\} \\
x_8 = \{\mathtt{x},\mathtt{z}\} & x_8 = (x_9 \cup x_{10}) \cup \{\mathtt{z}\} \\
x_9 = \{\mathtt{x},\mathtt{z}\} & x_9 = (x_{10} \smallsetminus \{\mathtt{x}\}) \cup \{\mathtt{x}\} \\
x_{10} = \{\mathtt{x},\mathtt{z}\} & x_{10} = (x_3 \smallsetminus \{\mathtt{z}\}) \cup \{\mathtt{z}\} \\
x_{11} = \{\mathtt{x}\} & x_{11} = \{\mathtt{x}\}
\end{array}
$$

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Reaching definitions

The reaching definitions for a given program point are those assignments that may have defined the current values of variables.

```
var x,y,z;
x := input;
while (x>1) {
  y := x/2;
  if (y>3)
    x := x-y;
  z := x-4;
  if (z>0)
    x := x/2;
  z := z-1; }
output x;
```

Example: Reaching definitions

The reaching definitions for a given program point are those assignments that may have defined the current values of variables.

```
var x,y,z;
x := input;
while (x>1) {
   y := x/2;
   if (y>3)
      x := x-y;
   z := x-4;
   if (z>0)
      x := x/2;
   z := z-1; }
output x;
```

Assignments:

$$\mathit{Asgns} = \{\texttt{x=input}, \texttt{y=x/2}, \texttt{x=x-y}, \texttt{z=x-4}, \texttt{x=x/2}, \texttt{z=z-1}\}$$

Example: Reaching definitions

The reaching definitions for a given program point are those assignments that may have defined the current values of variables.

```
var x,y,z;
x := input;
while (x>1) {
   y := x/2;
   if (y>3)
      x := x-y;
   z := x-4;
   if (z>0)
      x := x/2;
   z := z-1; }
output x;
```

Assignments:

$$\mathit{Asgns} = \{\texttt{x=input}, \texttt{y=x/2}, \texttt{x=x-y}, \texttt{z=x-4}, \texttt{x=x/2}, \texttt{z=z-1}\}$$

$$I = \langle \mathcal{P}(\mathit{Asgns}), \cup, \subseteq, \mathit{Asgns}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_{11}(\vec{x}))\rangle$$

Example: Reaching definitions

The reaching definitions for a given program point are those assignments that may have defined the current values of variables.

```
var x,y,z;
x := input;
while (x>1) {
   y := x/2;
   if (y>3)
      x := x-y;
   z := x-4;
   if (z>0)
      x := x/2;
   z := z-1; }
output x;
```

Assignments:

$$\mathit{Asgns} = \{\texttt{x=input}, \texttt{y=x/2}, \texttt{x=x-y}, \texttt{z=x-4}, \texttt{x=x/2}, \texttt{z=z-1}\}$$

$$I = \langle \mathcal{P}(\mathit{Asgns}), \cup, \subseteq, \mathit{Asgns}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_{11}(\vec{x}))\rangle$$

Product lattice: $(\mathcal{P}^{11}(\mathit{Asgns}), \subseteq)$

Example: Reaching definitions

The reaching definitions for a given program point are those assignments that may have defined the current values of variables.

```
var x,y,z;
x := input;
while (x>1) {
   y := x/2;
   if (y>3)
      x := x-y;
   z := x-4;
   if (z>0)
      x := x/2;
   z := z-1; }
output x;
```

Assignments:

$$\mathit{Asgns} = \{\texttt{x=input}, \texttt{y=x/2}, \texttt{x=x-y}, \texttt{z=x-4}, \texttt{x=x/2}, \texttt{z=z-1}\}$$

$$I = \langle \mathcal{P}(\mathit{Asgns}), \cup, \subseteq, \mathit{Asgns}, \emptyset, \lambda\vec{x}.(F_1(\vec{x}), \ldots, F_{11}(\vec{x}))\rangle$$

Product lattice: $(\mathcal{P}^{11}(\mathit{Asgns}), \subseteq)$

Direction: Forward
Analysis: May
Solution: Minimal

Example: Busy expressions

An expression is busy if it will definitely be evaluated again before its value changes.

Example: Busy expressions

An expression is busy if it will definitely be evaluated again before its value changes.

Direction: Backward
Analysis: Must
Solution: Minimal

Computing variable values: different abstraction levels

We may consider different abstraction levels of variable values:

- sets of integer values: $\mathcal{P}(\mathbb{Z})$
- intervals: $\{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, \infty\}, l \leq u\} \cup \{\bot\}$
- only signs with zero: $\mathcal{P}(\{-, 0, +\})$
- initialized or not: $\{\bot, \top\}$

Computing variable values: different abstraction levels

We may consider different abstraction levels of variable values:

- sets of integer values: $\mathcal{P}(\mathbb{Z})$
- intervals: $\{[l, u] \mid l, u \in \mathbb{Z} \cup \{-\infty, \infty\}, l \leq u\} \cup \{\perp\}$
- only signs with zero: $\mathcal{P}(\{-, 0, +\})$
- initialized or not: $\{\perp, \top\}$

Which abstraction is more precise than other?

Widening and narrowing

Fixpoint approximation techniques

When the extreme fixpoints of the system of equations cannot be computed in finitely many steps, they can be approximated.

Generally, we have these two approaches:

1. we can find more abstract interpretation
2. we can make approximations in current interpretation to accelerate convergence of Kleene's sequence

Here we are concerned about second approach – the technique called widening.

Fixpoint approximation techniques

Widening makes Kleene's sequence to converge

- to a fixpoint possibly greater than the least one or
- to an element s, such that $s > F(s)$.

In the second case, since s is greater then the least fixpoint, we can use narrowing to make the solution more precise – i.e. to find some fixpoint smaller than s but possibly greater than the least fixpoint.

Widening

- If the Kleene's sequence does not converge, then there exists a location x_i on a program loop where the sequence does not converge.
- We need a widening function $\triangledown : L \times L \to L$, which is applied every time the location x_i is updated: $x_i = x_i \triangledown F_i(\vec{x})$.
- We must define $\triangledown$ such that
  - for each $x, y \in L$, $x \circ y \leq x \triangledown y$, i.e. $\triangledown$ overapproximates operation $\circ$,
  - it ensures, that every infinite sequence of elements occurring in x_i is not strictly increasing.

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3       x := x + 1;
4   }
```

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3       x := x + 1;
4   }
```

{functions}

$$x_1 = [1, 1]$$
$$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$$
$$x_3 = x_2 + [1, 1]$$
$$x_4 = (x_1 \cup x_3) \cap [101, \infty]$$

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3       x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$

$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$

$x_3 = x_2 + [1, 1]$

$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Widening operator $\triangledown$:

$[i, j] \triangledown [k, l] = [ite(k < i, -\infty, i), ite(l > j, \infty, j)]$

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3      x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Widening operator ∇:
$[i, j] \nabla [k, l] = [ite(k < i, -\infty, i), ite(l > j, \infty, j)]$

{no widening}
$x_1 = [1, 1]$
$x_2 = [1, 100]$
$x_3 = [2, 101]$
$x_4 = [101, 101]$
100 iterations

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3       x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Widening operator $\triangledown$:
$[i, j] \triangledown [k, l] = [ite(k < i, -\infty, i), ite(l > j, \infty, j)]$

{no widening}	$\{x_3 = x_3 \triangledown (x_2 + [1, 1])\}$
$x_1 = [1, 1]$	$x_1 = [1, 1]$
$x_2 = [1, 100]$	$x_2 = [1, 100]$
$x_3 = [2, 101]$	$x_3 = [2, \infty]$
$x_4 = [101, 101]$	$x_4 = [101, \infty]$
100 iterations	2 iterations

Widening

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3      x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$

$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$

$x_3 = x_2 + [1, 1]$

$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Widening operator ∇:

$[i, j] \nabla [k, l] = [ite(k < i, -\infty, i), ite(l > j, \infty, j)]$

{no widening}	$\{x_3 = x_3 \nabla (x_2 + [1, 1])\}$
$x_1 = [1, 1]$	$x_1 = [1, 1]$
$x_2 = [1, 100]$	$x_2 = [1, 100]$
$x_3 = [2, 101]$	$x_3 = [2, \infty]$
$x_4 = [101, 101]$	$x_4 = [101, \infty]$
100 iterations	2 iterations

Narrowing

- When widening ends with $s > F(s)$, we improve solution s as follows: $s \geq F(s) \geq \ldots \geq F^n(s) \geq \ldots \geq s_0$, where s_0 is the least fixpoint.
- When the sequence is finite, its limit is better approximation of s_0.
- If the sequence is infinite, we apply narrowing function $\triangle \colon L \times L \to L$ at not stabilizing location x_i such that $x_i = x_i \triangle F_i(\vec{x})$.
- Operator $\triangle$ must satisfy:
  - for each $x, y \in L$, $x > y \to (x \geq x \triangle y \geq y)$, i.e. $\triangle$ tries to slow down the decreasing of the sequence,
  - it ensures, that every infinite sequence of elements starting from any s is not strictly decreasing.

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
x := 1;
while (x <= 100) {
    x := x + 1;
}
```

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3       x := x + 1;
4   }
```

{functions}

$$x_1 = [1, 1]$$
$$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$$
$$x_3 = x_2 + [1, 1]$$
$$x_4 = (x_1 \cup x_3) \cap [101, \infty]$$

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3      x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Narrowing operator $\triangle$:
$[i, j] \triangle [k, l] = [ite(i = -\infty, k, min(i, k)), ite(j = \infty, l, max(j, l))]$

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3      x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Narrowing operator $\triangle$:
$[i, j] \triangle [k, l] = [ite(i = -\infty, k, min(i, k)), ite(j = \infty, l, max(j, l))]$

{no widening}	{widening}
$x_1 = [1, 1]$	$x_1 = [1, 1]$
$x_2 = [1, 100]$	$x_2 = [1, 100]$
$x_3 = [2, 101]$	$x_3 = [2, \infty]$
$x_4 = [101, 101]$	$x_4 = [101, \infty]$
100 iterations	2 iteration

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
1  x := 1;
2  while (x <= 100) {
3     x := x + 1;
4  }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Narrowing operator $\triangle$:
$[i, j] \triangle [k, l] = [ite(i = -\infty, k, min(i, k)), ite(j = \infty, l, max(j, l))]$

{no widening}	{widening}	$\{x_3 = x_3 \triangle (x_2 + [1, 1])\}$
$x_1 = [1, 1]$	$x_1 = [1, 1]$	$x_1 = [1, 1]$
$x_2 = [1, 100]$	$x_2 = [1, 100]$	$x_2 = [1, 100]$
$x_3 = [2, 101]$	$x_3 = [2, \infty]$	$x_3 = [2, 101]$
$x_4 = [101, 101]$	$x_4 = [101, \infty]$	$x_4 = [101, 101]$
100 iterations	2 iteration	+1 iteration

Narrowing

Example: Interval bounds of integer variable `x`

```
{locations are after}
1   x := 1;
2   while (x <= 100) {
3      x := x + 1;
4   }
```

{functions}

$x_1 = [1, 1]$
$x_2 = (x_1 \cup x_3) \cap [-\infty, 100]$
$x_3 = x_2 + [1, 1]$
$x_4 = (x_1 \cup x_3) \cap [101, \infty]$

Narrowing operator $\triangle$:

$[i, j] \triangle [k, l] = [ite(i = -\infty, k, min(i, k)), ite(j = \infty, l, max(j, l))]$

{no widening}	{widening}	$\{x_3 = x_3 \triangle (x_2 + [1, 1])\}$
$x_1 = [1, 1]$	$x_1 = [1, 1]$	$x_1 = [1, 1]$
$x_2 = [1, 100]$	$x_2 = [1, 100]$	$x_2 = [1, 100]$
$x_3 = [2, 101]$	$x_3 = [2, \infty]$	$x_3 = [2, 101]$
$x_4 = [101, 101]$	$x_4 = [101, \infty]$	$x_4 = [101, 101]$
100 iterations	2 iteration	+1 iteration

Coming next week

Shape Analysis via 3-Valued Logic

- Static analysis of dynamic memory.
- It can detect NULL dereferences, memory leaks, etc.
- Applicable to real code.