Lecture 18: Abstract Interpretation

Viktor Kunčak

Abstract Interpretation

A Method for Constructing Inductive Invariants

Basic idea of abstract interpretation

Abstract interpretation is a way to infer properties of program computations.
Consider the assignment: $z = x + y$.

Interpreter:

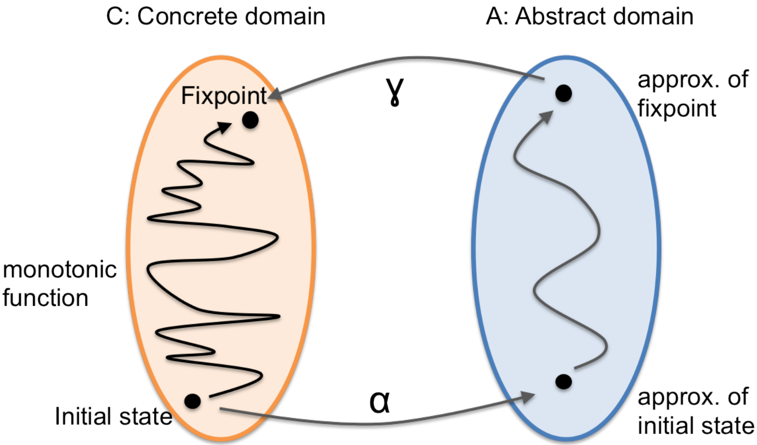
$$\left(\begin{array}{c} x : 10 \\ y : -2 \\ z : 3 \end{array}\right) \xrightarrow{z=x+y} \left(\begin{array}{c} x : 10 \\ y : -2 \\ z : 8 \end{array}\right)$$

Abstract interpreter:

$$\left(\begin{array}{cc} x \in & [0, 10] \\ y \in & [-5, 5] \\ z \in & [0, 10] \end{array}\right) \xrightarrow{z=x+y} \left(\begin{array}{cc} x \in & [0, 10] \\ y \in & [-5, 5] \\ z \in & [-5, 15] \end{array}\right)$$

Each abstract state represents a set of concrete states

Program Meaning is a Fixpoint. We Approximate It.



γ maps abstract states to concrete states

Proving through Fixpoints of Approximate Functions

Meaning of a program (e.g. a relation) is a least fixpoint of F.
Given specification s, the goal is to prove $\mathbf{lfp(F)} \subseteq \mathbf{s}$

- if $F(s) \subseteq s$ then $lfp(F) \subseteq s$ and we are done
- $lfp(F) = \bigcup_{k \geq 0} F^k(\emptyset)$, but that is too hard to compute because it is infinite union unless, by some luck, $F^{n+1}(\emptyset) = F^n$ for some n

Instead, we search for an inductive strengthening of s: find s' such that:

- $F(s') \subseteq s'$ (s' is inductive). If so, theorem says $lfp(F) \subseteq s'$
- $s' \subseteq s$ (s' implies the desired specification). Then $lfp(F) \subseteq s' \subseteq s$

How to find s'? Iterating F is hard, so we try some simpler function $F_\#$:

- suppose $F_\#$ is *approximation*: $F(r) \subseteq F_\#(r)$ for all r
- we can find s' such that: $F_\#(s') \subseteq s'$ (e.g. $s' = F_\#^{n+1}(\emptyset) = F_\#^n(\emptyset)$)

Then: $F(s') \subseteq F_\#(s') \subseteq s'$. So, if $s' \subseteq s$, we have know $lfp(F) \subseteq s'$.
Abstract interpretation: automatically construct $F_\#$ using F and s

Programs as control-flow graphs

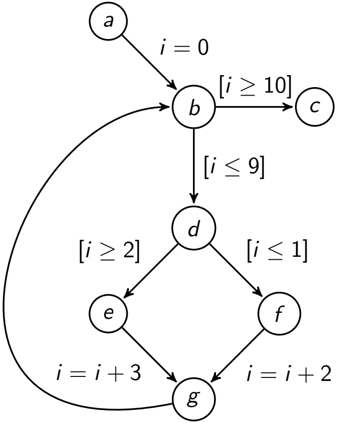
One possible corresponding control-flow graph is:

```
//a
i = 0;
     //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

Programs as control-flow graphs

```
//a
i = 0;
     //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```

One possible corresponding control-flow graph is:



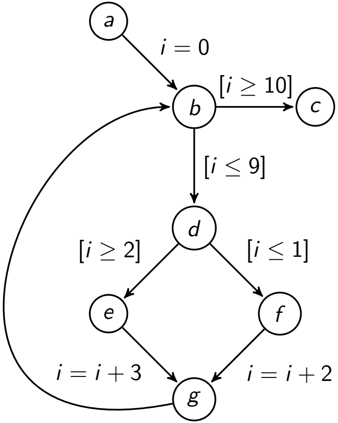
Sets of states at each program point

Suppose that

- program state is given by the value of the integer variable *i*
- initially, it is possible that i has any value

Compute the set of states at each vertex in the CFG.

```
//a
i = 0;
     //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```



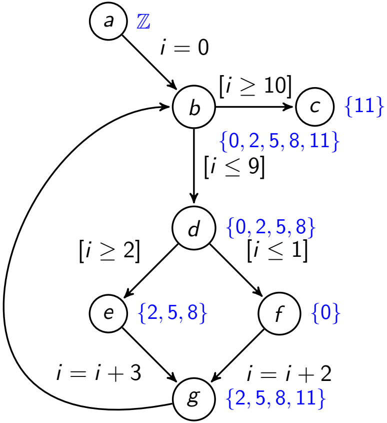
Sets of states at each program point

Suppose that

- program state is given by the value of the integer variable i
- initially, it is possible that i has any value

Compute the set of states at each vertex in the CFG.

```
//a
i = 0;
     //b
while (i < 10) {
  //d
  if (i > 1)
    //e
    i = i + 3;
  else
    //f
    i = i + 2;
  //g
}
//c
```



Sets of states at each program point

Running the Program

One way to describe the set of states for each program point: for each initial state, run the CFG with this state and insert the modified states at appropriate points.

Reachable States as A Set of Recursive Equations

If c is the label on the edge of the graph, let $\rho(c)$ denotes the relation between initial and final state that describes the meaning of statement. For example,

$$\begin{aligned}
&\rho(i = 0) = \{(i, i') \mid i' = 0\}\\
&\rho(i = i + 2) = \{(i, i') \mid i' = i + 2\}\\
&\rho(i = i + 3) = \{(i, i') \mid i' = i + 3\}\\
&\rho([i < 10]) = \{(i, i') \mid i' = i \wedge i < 10\}
\end{aligned}$$

Sets of states at each program point

We will write $T(S, c)$ (transfer function) for the image of set S under relation $\rho(c)$. For example,

$$T(\{10, 15, 20\}, i = i + 2) = \{12, 17, 22\}$$

General definition can be given using the notion of strongest postcondition

$$T(S, c) = sp(S, \rho(c))$$

If [p] is a condition (assume(p), coming from 'if' or 'while') then

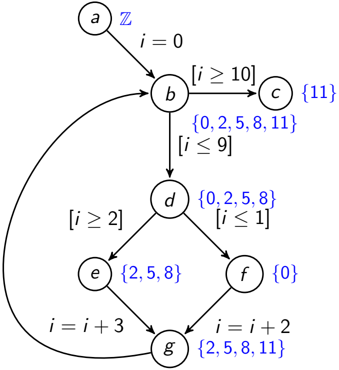
$$T(S, [p]) = \{x \in S \mid p\}$$

If an edge has no label, we denote it skip. So, $T(S, skip) = S$.

Reachable States as A Set of Recursive Equations

Now we can describe the meaning of our program using recursive equations:

$$
\begin{aligned}
S(a) &= \{\ldots, -2, -1, 0, 1, 2, \ldots\}\\
S(b) &= T(S(a), i = 0) \cup T(S(g), \mathit{skip})\\
S(c) &= T(S(b), [\neg(i < 10)])\\
S(d) &= T(S(b), [i < 10])\\
S(e) &= T(S(d), [i > 1])\\
S(f) &= T(S(d), [\neg(i > 1)])\\
S(g) &= T(S(e), i = i + 3)\\
&\quad \cup T(S(f), i = i + 2)
\end{aligned}
$$



Our solution is the unique **least** solution of these equations. Can be computed by iterating starting from empty sets as initial solution.

The problem: These exact equations are as difficult to compute as running the program on all possible input states. Instead, we consider **approximate** descriptions of these sets of states.

A Large Analysis Domain: All Intervals of Integers

For every $L, U \in \mathbb{Z}$ interval:

$$\{x \mid L \leq x \wedge x \leq U\}$$

This domain has infinitely many elements, but is already an approximation of all possible sets of integers.

Smaller Domain: Finitely Many Intervals

We continue with the same example but instead of allowing to denote all possible sets, we will allow sets represented by expressions

$$[L, U]$$

which denote the set $\{x \mid L \leq x \land x \leq U\}$.
Example: $[0, 127]$ denotes integers between 0 and 127.

- L is the lower bound and U is the upper bound, with $L \leq U$.
- to ensure that we have only a few elements, we let

$$L, U \in \{\text{MININT}, -128, 1, 0, 1, 127, \text{MAXINT}\}$$

- $[\text{MININT}, \text{MAXINT}]$ denotes all possible integers, denote it $\top$
- instead of writing $[1, 0]$ and other empty sets, we will always write $\bot$

So, we only work with a finite number of sets $1 + \binom{7}{2} = 22$.
Denote the family of these sets by D (domain).

New Set of Recursive Equations

We want to write the same set of equations as before, but because we have only a finite number of sets, we must approximate. We approximate sets with possibly larger sets.

$$
\begin{aligned}
S^{\#}(a) &= \top \\
S^{\#}(b) &= T^{\#}(S^{\#}(a), i = 0) \\
&\quad \sqcup T^{\#}(S^{\#}(g), skip) \\
S^{\#}(c) &= T^{\#}(S^{\#}(b), [\neg(i < 10)]) \\
S^{\#}(d) &= T^{\#}(S^{\#}(b), [i < 10]) \\
S^{\#}(e) &= T^{\#}(S^{\#}(d), [i > 1]) \\
S^{\#}(f) &= T^{\#}(S^{\#}(d), [\neg(i > 1)]) \\
S^{\#}(g) &= T^{\#}(S^{\#}(e), i = i + 3) \\
&\quad \sqcup T^{\#}(S^{\#}(f), i = i + 2)
\end{aligned}
$$

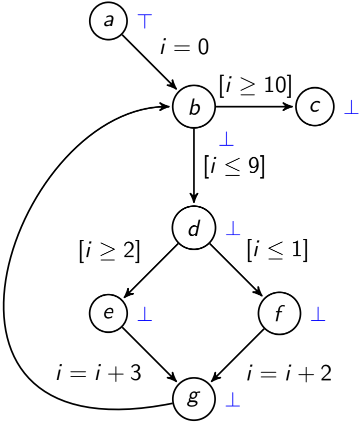
- $S_1 \sqcup S_2$ denotes the approximation of $S_1 \cup S_2$: it is the set that contains both S_1 and S_2, that belongs to D, and is otherwise as small as possible. Here $[a, b] \sqcup [c, d] = [min(a, c), max(b, d)]$
- We use approximate functions $T^{\#}(S, c)$ that give a result in D.

Updating Sets

We solve the equations by starting in the initial state and repeatedly applying them.

- in the 'entry' point, we put $\top$, in all others we put $\bot$.

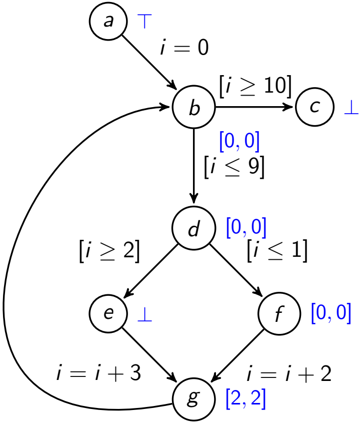
$$
\begin{aligned}
S^{\#}(a) &= \top \\
S^{\#}(b) &= T^{\#}(S^{\#}(a), i = 0) \\
&\sqcup\ T^{\#}(S^{\#}(g), skip) \\
S^{\#}(c) &= T^{\#}(S^{\#}(b), [\neg(i < 10)]) \\
S^{\#}(d) &= T^{\#}(S^{\#}(b), [i < 10]) \\
S^{\#}(e) &= T^{\#}(S^{\#}(d), [i > 1]) \\
S^{\#}(f) &= T^{\#}(S^{\#}(d), [\neg(i > 1)]) \\
S^{\#}(g) &= T^{\#}(S^{\#}(e), i = i + 3) \\
&\sqcup\ T^{\#}(S^{\#}(f), i = i + 2)
\end{aligned}
$$



Updating Sets

Sets after a few iterations:

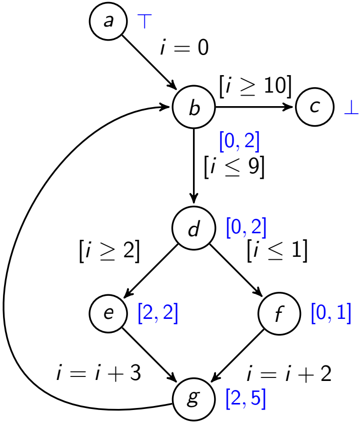
$$
\begin{aligned}
S^{\#}(a) &= \top \\
S^{\#}(b) &= T^{\#}(S^{\#}(a), i = 0) \\
&\sqcup\ T^{\#}(S^{\#}(g), \mathit{skip}) \\
S^{\#}(c) &= T^{\#}(S^{\#}(b), [\neg(i < 10)]) \\
S^{\#}(d) &= T^{\#}(S^{\#}(b), [i < 10]) \\
S^{\#}(e) &= T^{\#}(S^{\#}(d), [i > 1]) \\
S^{\#}(f) &= T^{\#}(S^{\#}(d), [\neg(i > 1)]) \\
S^{\#}(g) &= T^{\#}(S^{\#}(e), i = i + 3) \\
&\sqcup\ T^{\#}(S^{\#}(f), i = i + 2)
\end{aligned}
$$



Updating Sets

Sets after a few more iterations:

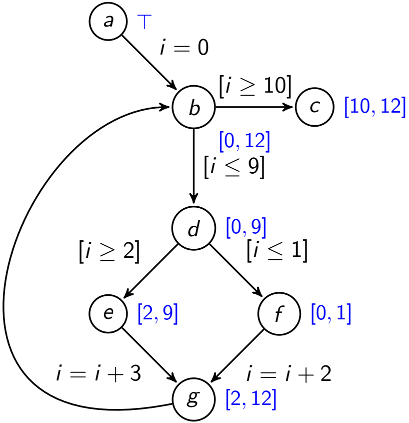
$$
\begin{aligned}
S^{\#}(a) &= \top \\
S^{\#}(b) &= T^{\#}(S^{\#}(a), i = 0) \\
&\sqcup\ T^{\#}(S^{\#}(g), \mathit{skip}) \\
S^{\#}(c) &= T^{\#}(S^{\#}(b), [\neg(i < 10)]) \\
S^{\#}(d) &= T^{\#}(S^{\#}(b), [i < 10]) \\
S^{\#}(e) &= T^{\#}(S^{\#}(d), [i > 1]) \\
S^{\#}(f) &= T^{\#}(S^{\#}(d), [\neg(i > 1)]) \\
S^{\#}(g) &= T^{\#}(S^{\#}(e), i = i + 3) \\
&\sqcup\ T^{\#}(S^{\#}(f), i = i + 2)
\end{aligned}
$$



Fixpoint Found

Final values of sets:

$$
\begin{aligned}
S^{\#}(a) &= \top \\
S^{\#}(b) &= T^{\#}(S^{\#}(a), i = 0) \\
&\sqcup\ T^{\#}(S^{\#}(g), \mathit{skip}) \\
S^{\#}(c) &= T^{\#}(S^{\#}(b), [\neg(i < 10)]) \\
S^{\#}(d) &= T^{\#}(S^{\#}(b), [i < 10]) \\
S^{\#}(e) &= T^{\#}(S^{\#}(d), [i > 1]) \\
S^{\#}(f) &= T^{\#}(S^{\#}(d), [\neg(i > 1)]) \\
S^{\#}(g) &= T^{\#}(S^{\#}(e), i = i + 3) \\
&\sqcup\ T^{\#}(S^{\#}(f), i = i + 2)
\end{aligned}
$$

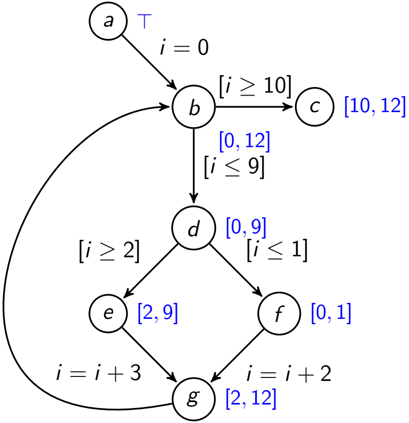


If we map intervals to sets, this is also solution of the original constraints.

Automatically Constructed Hoare Logic Proof

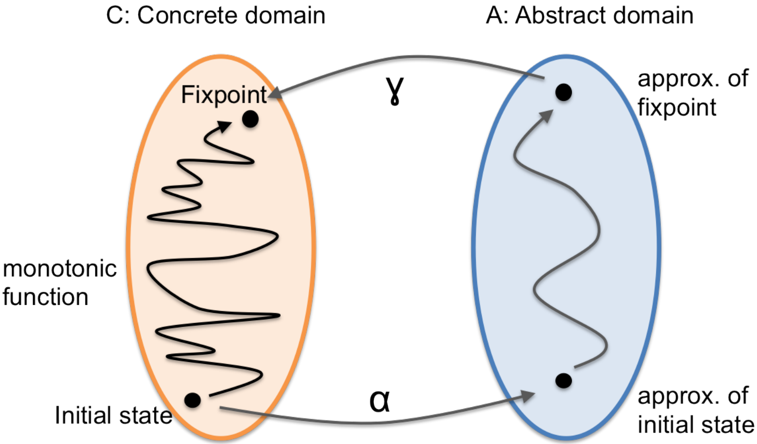
Final values of sets:

```
//a: true
i = 0;
        //b: $0 \le i \le 12$
while (i < 10) {
   //d: $0 \le i \le 9$
   if (i > 1)
      //e: $2 \le i \le 9$
      i = i + 3;
   else
      //f: $0 \le i \le 1$
      i = i + 2;
   //g: $2 \le i \le 12$
}
//c: $10 \le i \le 12$
```



This method constructed a sufficiently annotated program and ensured that all Hoare triples that were constructed hold

Abstract Interpretation Big Picture



Abstract Domains are Partial Orders

Program semantics is given by certain sets (e.g. sets of reachable states).

- subset relation $\subseteq$: used to compare sets
- union of states: used to combine sets coming from different executions (e.g. if statement)

Our goal is to approximate such sets. We introduce a domain of elements $d \in D$ where each d represents a set.

- $\gamma(d)$ is a set of states. γ is called **concretization function**
- given d_1 and d_2, it could happen that there is **no element** d representing union

$$\gamma(d_1) \cup \gamma(d_2) = \gamma(d)$$

Instead, we use a set d that approximates union, and denote it $d_1 \sqcup d_2$

This leads us to review the theory of **partial orders** and **(semi)lattices**.

Partial Orders

Partial ordering relation is a binary relation $\leq$ that is reflexive, antisymmetric, and transitive, that is, the following properties hold for all x, y, z:

- $x \leq x$ (reflexivity)
- $x \leq y \land y \leq x \to x = y$ (antisymmetry)
- $x \leq y \land y \leq z \to x \leq z$ (transitivity)

If A is a set and $\leq$ a binary relation on A, we call the pair $(A, \leq)$ a **partial order**.

Given a partial ordering relation $\leq$, the corresponding **strict ordering relation** $x < y$ is defined by $x \leq y \land x \neq y$ and can be viewed as a shorthand for this conjunction.

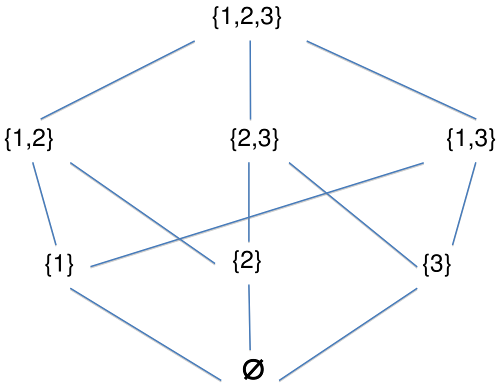
- Orders on integers, rationals, reals are all special cases of partial orders called *linear orders*.
- Given a set U, let A be any set of subsets of U, that is $A \subseteq 2^U$. Then $(A, \subseteq)$ is a partial order.

Example: Let $U = \{1, 2, 3\}$ and let $A = \{\emptyset, \{1\}, \{2\}, \{3\}, \{2, 3\}, \{1, 2, 3\}\}$. Then $(A, \subseteq)$ is a partial order. We can draw it as a *Hasse diagram*.

Hasse diagram

presents the relation as a directed graph in a plane, such that

- the direction of edge is given by which nodes is drawn above
- transitive and reflexive edges are not represented (they can be derived)



Extreme Elements in Partial Orders

Given a partial order $(A, \leq)$ and a set $S \subseteq A$, we call an element $a \in A$

- **upper bound** of S if for all $a' \in S$ we have $a' \leq a$
- **lower bound** of S if for all $a' \in S$ we have $a \leq a'$
- **minimal element** of S if $a \in S$ and there is no element $a' \in S$ such that $a' < a$
- **maximal element** of S if $a \in S$ and there is no element $a' \in S$ such that $a < a'$
- **greatest element** of S if $a \in S$ and for all $a' \in S$ we have $a' \leq a$
- **least element** of S if $a \in S$ and for all $a' \in S$ we have $a \leq a'$
- **least upper bound** (lub, supremum, join, $\sqcup$) of S if a is the least element in the set of all upper bounds of S
- **greatest lower bound** (glb, infimum, meet, $\sqcap$) of S if a is the greatest element in the set of all lower bounds of S

Taking $S = A$ we obtain minimal, maximal, greatest, least elements for the entire partial order.

Extreme Elements in Partial Orders

Notes

- minimal element need not exist: $(0,1)$ interval of rationals
- there may be multiple minimal elements: $\{\{a\},\{b\},\{a,b\}\}$
- if minimal element exists, it need not be least: above example
- there are no two distinct least elements for the same set
- least element is always glb and minimal
- if glb belongs to the set, then it is always least and minimal
- on a family of relations closed under $\cap$ and $\cup$, *glb* is $\cap$ and *lub* is $\cup$ for the partial order $\subseteq$; not all families of sets are closed; these are:
  - the set of all subsets
  - the family of open sets from topology

Least upper bound (lub, supremum, join, $\sqcup$)

Denoted $lub(S)$, least upper bound of S is an element M, if it exists, such that M is the least element of the set

$$U = \{x \mid x \text{ is upper bound on } S\}$$

In other words:

- M is an upper bound on S
- for every other upper bound M' on S, we have that $M \leq M'$

Note: this is the same definition as supremum in real analysis.

Least upper bound (glb, infimum, meet, $\sqcap$)

$a_1 \sqcup a_2$ denotes $lub(\{a_1, a_2\})$

$$(\ldots(a_1 \sqcup a_2)\ldots) \sqcup a_n \quad \text{is in fact } lub(\{a_1, \ldots, a_n\})$$

So the operation is

- associative
- commutative
- idempotent

Real Analysis

Take as S the open interval of reals $(0,1) = \{x \mid 0 < x < 1\}$
Then

- S has no maximal element
- S thus has no greatest element
- 2, 2.5, 3,... are all upper bounds on S
- $lub(S) = 1$

Execise: subsets of U

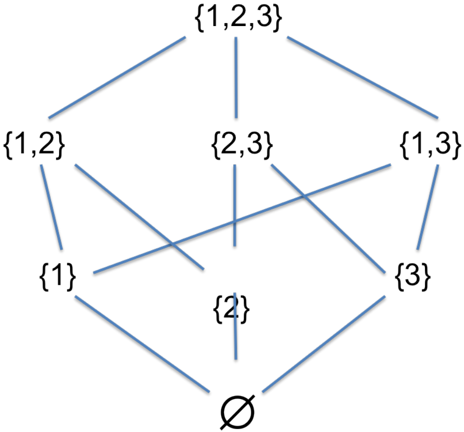
Consider

$$A = 2^U = \{S \mid S \subseteq U\} \qquad \text{and} \qquad (A, \subseteq)$$

Do these exist, and if so, what are they?

- $s_1 \subseteq S, s_2 \subseteq S, lub(\{s_1, s_2\}) =?$
- $lub(S) =?$

Exercise: find the lub



{1,2,3}

{1,2}

{2,3}

{1,3}

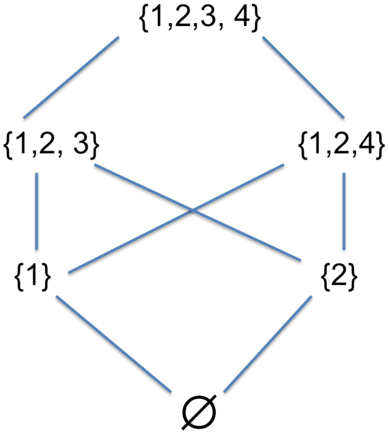
{1}

{2}

{3}

∅

$\{1\} \sqcup \{2\} =$



{1,2,3, 4}

{1,2, 3}

{1,2,4}

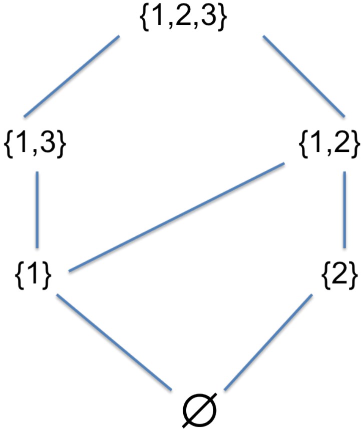
{1}

{2}

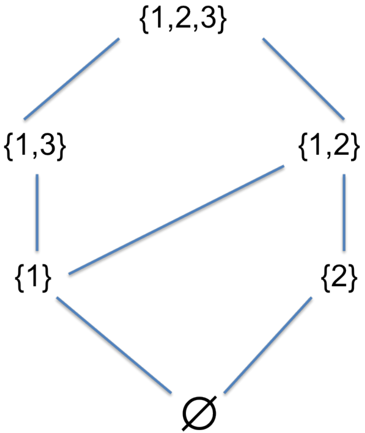
∅

$\{1\} \sqcup \{2\} =$

Does every pair of elements in this order have a least upper bound?



Does every pair of elements in this order have a least upper bound?



Dually, does it have a **greatest lower bound**?

Partial order for the domain of intervals

Domain: $D = \{\bot\} \cup \{(L, U) \mid L \in \{-\infty\} \cup \mathbb{Z}, U \in \{+\infty\} \cup \mathbb{Z}$
such that $L \leq U$.

The associated set of elements is given by the function γ:

$$\gamma : D \to 2^{\mathbb{Z}}, \qquad \gamma((L, U)) = \{x \mid L \leq x \wedge x \leq U\}$$

Lub: for $d_1, d_2 \in D$, $d_1 \sqsubseteq d_2 \quad \leftrightarrow \quad \gamma(d_1) \subseteq \gamma(d_2)$
hence

$$(L_1, U_1) \sqsubseteq (L_2, U_2) \quad \leftrightarrow \quad L_2 \leq L_1 \wedge U_1 \leq U_2$$

$$\bot \sqsubseteq d \quad \forall d \in D$$

$$(L_1, U_1) \sqcup (L_2, U_2) = (min(L_1, L_2), max(U_1, U_2))$$

Remark on constructing orders using inverse images

Suppose $\gamma : D \to C$ where C is some collection of sets.
If we define relation $\sqsubseteq$ by:

$$d_1 \sqsubseteq d_2 \iff \gamma(d_1) \subseteq \gamma(d_2)$$

then

1. $\sqsubseteq$ is reflexive
2. $\sqsubseteq$ is transitive
3. $\sqsubseteq$ is antisymmetric if and only iff γ is injective

If $\sqsubseteq$ is not antisymmetric then we can define equivalence relation

$$d_1 \sim d_2 \iff \gamma(d_1) = \gamma(d_2)$$

and then take D' to be equivalence classes of such new set.
Example: suppose we defined intervals as all possible pairs of integers (L, U). Then there would be many representations of the empty set, all those intervals where $L > U$.

Lattices

Definition: A lattice is a partial order in which every two-element set has a least upper bound and a greatest lower bound.

Lemma: In a lattice every non-empty finite set has a lub ($\sqcup$) and glb ($\sqcap$).

Lattices

Definition: A lattice is a partial order in which every two-element set has a least upper bound and a greatest lower bound.

Lemma: In a lattice every non-empty finite set has a lub ($\sqcup$) and glb ($\sqcap$).

Proof: is by induction!
Case where the set S has three elements x,y and z:
Let $a = (x \sqcup y) \sqcup z$.
By definition of $\sqcup$ we have $z \sqsubseteq a$ and $x \sqcup y \sqsubseteq a$.
Then we have again by definition of $\sqcup$, $x \sqsubseteq x \sqcup y$ and $y \sqsubseteq x \sqcup y$. Thus by transitivity we have $x \sqsubseteq a$ and $y \sqsubseteq a$.
Thus we have $S \sqsubseteq a$ and a is an upper bound.
Now suppose that there exists a' such that $S \sqsubseteq a'$. We want $a \sqsubseteq a'$ (a least upper bound):
We have $x \sqsubseteq a'$ and $y \sqsubseteq a'$, thus $x \sqcup y \sqsubseteq a'$. But $z \sqsubseteq a'$, thus $((x \sqcup y) \sqcup z) \sqsubseteq a'$.
Thus a is the lub of our 3 elements set.

Examples of Lattices

Lemma: Every linear order is a lattice.

Example: Every bounded subset of the set of real numbers has a lub. This is an axiom of real numbers, the way they are defined (or constructed from rationals).

- If a lattice has least and greatest element, then every finite set (including empty set) has a lub and glb.
- This does not imply there are lub and glb for infinite sets.
  Example: In the oder $([0,1), \leq)$ with standard ordering on reals is a lattice, the entire set has no lub. The set of all rationals of interval $[0, 10]$ is a lattice, but the set $\{x \mid 0 \leq x \wedge x^2 < 2\}$ has no lub.

Exercises

Prove the following:

1. $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
2. $\sqcup A \sqsubseteq \sqcap B \iff \forall x \in A. \forall y \in B. x \sqsubseteq y$
3. Let $(A, \sqsubseteq)$ be a partial order such that every set $S \subseteq A$ has the greatest lower bound. Prove that then every set $S \subseteq A$ has the least upper bound.

Solution

1. Let $(x \sqcup y) \sqcup z = a$, then $z \sqsubseteq a$ and $x \sqcup y \sqsubseteq a$ and hence $x \sqsubseteq a$ and $y \sqsubseteq a$. From $z \sqsubseteq a$ and $y \sqsubseteq a$ it follows $z \sqcup y \sqsubseteq a$ and finally $x \sqcup (y \sqcup z) \sqsubseteq a = (x \sqcup y) \sqcup z$.
   Symmetrically, we also get that $(x \sqcup y) \sqcup z \sqsubseteq x \sqcup (y \sqcup z)$ and hence $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$.

2. Let $a = \sqcup A$ and $b = \sqcap B$. Then

   $$a = \sqcup A \Leftrightarrow \forall x \in A.x \sqsubseteq a \wedge \forall a'.\forall x \in A.x \sqsubseteq a' \rightarrow a \sqsubseteq a'$$

   and

   $$b = \sqcap B \Leftrightarrow \forall y \in B.b \sqsubseteq y \wedge \forall b'.\forall y \in B.b' \sqsubseteq y \rightarrow b' \sqsubseteq b$$

   $\Rightarrow$
   Suppose $a \sqsubseteq b$, then by transitivity it follows that $\forall x \in A.\forall y \in B.x \sqsubseteq y$.

   $\Leftarrow$
   Now suppose $\forall x \in A.\forall y \in B.x \sqsubseteq y$. This implies that the set A is a set of lower bounds to B, and B is a set of upper bounds on A. Hence $a \sqsubseteq b$ (intuitively).

Solution

3. Suppose we have some arbitrary set $S \subseteq A$, then we know that $\sqcap S$ exists.
   Let U be the set of all its upper bounds, i.e. $U = \{x | \forall y \in S . y \sqsubseteq x\}$. Since every subset of A has a greatest lower bound, we know that $a = \sqcap U$ exists. Now we want to show that a is an upper bound on S and that it is the least one.
   We want to show that $\forall y \in S . y \sqsubseteq a$.
   Let L be the set of lower bounds on U, i.e. $L = \{z | \forall x \in U . z \sqsubseteq x\}$.
   Clearly, $S \subseteq L$, since U are the upper bounds on S.
   Then a is the greatest element in L, and thus $\forall y \in S . y \sqsubseteq a$ and so a is an upper bound on S.
   Now take any upper bound $x \in U$. Since $a = \sqcap U$, we have $a \sqsubseteq x$ and so a is the least upper bound.

Constructing Partial Orders using Maps

Example: Let A be the set of all propositional formulas containing only variables p, q. For a formula $F \in A$ define

$$[F] = \{(u, v).\ u, v \in \{0, 1\} \wedge F \text{ is true for } p \mapsto u, q \mapsto v\}$$

i.e. $[F]$ denotes the set of assignments for which F is true. Note that $F \implies G$ is a tautology iff $[F] \subseteq [G]$. Define ordering on formulas A by

$$F \leq G \iff [F] \subseteq [G]$$

Is $\leq$ a partial order? Which laws does $\leq$ satisfy?

Constructing Partial Orders using Maps

Lemma: Let $(C, \leq)$ be an lattice and A a set. Let $\gamma : A \to C$ be an injective function. Define oder $x \sqsubseteq y$ on A by $\gamma(x) \leq \gamma(y)$. Then $(A, \sqsubseteq)$ is a partial order.

Note: even if $(C, \leq)$ had top and bottom element and was a lattice, the constructed order need not have top and bottom or be a lattice. For example, we take A to be a subset of A and define γ to be identity.

Lattices

Definition: A lattice is a partial order in which every two-element set has a least upper bound and a greatest lower bound (so, we have $\sqcap$ and $\sqcup$ as well-defined binary operations).
Lemma: In every lattice, $x \sqcup (x \sqcap y) = x$.

Lattices

Definition: A lattice is a partial order in which every two-element set has a least upper bound and a greatest lower bound (so, we have $\sqcap$ and $\sqcup$ as well-defined binary operations).
Lemma: In every lattice, $x \sqcup (x \sqcap y) = x$.

Proof:
We trivially have $x \sqsubseteq x \sqcup (x \sqcap y)$.
Let's prove that $x \sqcup (x \sqcap y) \sqsubseteq x$:
x is an upper bound of x and $x \sqcap y$, $x \sqcup (x \sqcap y)$ is the least upper bound of x and $x \sqcap y$, thus $x \sqcup (x \sqcap y) \sqsubseteq x$.
Definition: A lattice is *distributive* iff

$$\begin{aligned} x \sqcap (y \sqcup z) &= (x \sqcap y) \sqcup (x \sqcap z) \\ x \sqcup (y \sqcap z) &= (x \sqcup y) \sqcap (x \sqcup z) \end{aligned}$$

Lattice of all subsets of a set is distributive. Linear order is a distributive lattice.

Products of Lattices

Note: for $n = 2$ a function $f : \{1, 2\} \to (L_1 \cup L_2)$ with $f(1) \in L_1$, $f(2) \in L_2$ is isomorphic to an ordered pair $(f(1), f(2))$. We denote the product by $(L_1, \leq_1) \times (L_2, \leq_2)$.
Example: Let $R = \{a, b, c, d\}$ denote set of values. Let $A_1 = A_2 = 2^R$. Let

$$s_1 \leq_1 s_2 \iff s_1 \subseteq s_2$$

and let

$$t_1 \leq_2 t_2 \iff t_1 \supseteq t_2$$

Then we can define the product $(A_1, \leq_1) \times (A_2, \leq_2)$. In this product, $(s_1, t_1) \leq (s_2, t_2)$ iff: $s_1 \subseteq s_2$ and $t_1 \supseteq t_2$. The original partial orders were lattices, so the product is also a lattice. For example, we have

$$(\{a, b, c\}, \{a, b, d\}) \sqcap (\{b, c, d\}, \{c, d\}) = (\{b, c\}, \{a, b, c, d\})$$

Products of Lattices

Lattice elements can be combined into finite or infinite-dimensional vectors, and the result is again a lattice.

Lemma: Let $(A_1, \leq_1), \ldots, (A_n, \leq_n)$ be partial orders. Define $(L, \leq)$ by

$$A = \{f \mid f : \{1, \ldots, n\} \to (A_1 \cup \ldots \cup A_n) \text{ where } \forall i. f(i) \in A_i\}$$

For $f, g \in A$ define

$$f \leq g \iff \forall i. f(i) \leq_i g(i)$$

Then $(A, \leq)$ is a partial order. We denote $(A, \leq)$ by

$$\prod_{i=1}^{n} (L_i, \leq_i)$$

Moreover, if for each i, $(A_i, \leq_i)$ is a lattice, then $(A, \leq)$ is also a lattice.

Properties of $\sqcap S$ and $\sqcup S$

Consider a partial order $(A, \sqsubseteq)$.

- Suppose $S_1 \subseteq S_2 \subseteq A$ and $\sqcup S_1$ and $\sqcup S_2$ exist. In what relationship are these two elements?
- Suppose $S_1 \subseteq S_2 \subseteq A$ and $\sqcap S_1$ and $\sqcap S_2$ exist. In what relationship are these two elements?
- Suppose $\sqcup\emptyset$ exists. Describe this element.
- Suppose $\sqcap\emptyset$ exists. Describe this element.

Properties of $\sqcap S$ and $\sqcup S$

Consider a partial order $(A, \sqsubseteq)$.

- Suppose $S_1 \subseteq S_2 \subseteq A$ and $\sqcup S_1$ and $\sqcup S_2$ exist. In what relationship are these two elements?
- Suppose $S_1 \subseteq S_2 \subseteq A$ and $\sqcap S_1$ and $\sqcap S_2$ exist. In what relationship are these two elements?
- Suppose $\sqcup\emptyset$ exists. Describe this element.
- Suppose $\sqcap\emptyset$ exists. Describe this element.

$\sqcup\emptyset = \bot$ and $\sqcap\emptyset = \top$. This is because every element is an upper bound and a lower bound of $\emptyset$: $\forall x. \forall y \in \emptyset. y \sqsubseteq x$ is valid, as well as $\forall x. \forall y \in \emptyset. y \sqsupseteq x$.

Complete Semilattice is a Complete Lattice

If we have all $\sqcap$-s we then also have all $\sqcup$-s:
Theorem: Let $(A, \sqsubseteq)$ be a partial order such that every set $S \subseteq A$ has the greatest lower bound ($\sqcap$). Prove that then every set $S \subseteq A$ has the least upper bound ($\sqcup$).

Example: Application of the Previous Theorem

Let U be a set and $A \subseteq U \times U$ the set of all **equivalence relations** on this set. Consider the partial order $(A, \subseteq)$.

Lemma

If $I \subseteq A$ is a set of equivalence relations, then $\cap I$ is also an equivalence relation.

Consequence: Given $I \subseteq A$ there exists the least equivalence relation containing every relation from I (equivalence closure of relations in I).

Note: **congruence** is equivalence relation that agrees with some operations. For example, $x \sim x'$ and $y \sim y'$ implies $(x + y) \sim (x' + y')$. The analogous properties hold for congruence relations.

Complete Lattices

Definition: A **complete** lattice is a lattice where for every set S (including empty set and infinite sets) there exist $\sqcup S$ and $\sqcap S$.

Monotonic functions

Given two partial orders $(C, \leq)$ and $(A, \sqsubseteq)$, we call a function $\alpha : C \to A$ *monotonic* iff for all $x, y \in C$,

$$x \leq y \;\to\; \alpha(x) \sqsubseteq \alpha(y)$$

Reminder: Fixpoints

Definition: Given a set A and a function $f : A \to A$ we say that $x \in A$ is a fixed point (fixpoint) of f if $f(x) = x$.

Definition: Let $(A, \leq)$ be a partial order, let $f : A \to A$ be a monotonic function on $(A, \leq)$, and let the set of its fixpoints be $S = \{x \mid f(x) = x\}$. If the least element of S exists, it is called the **least fixpoint**, if the greatest element of S exists, it is called the **greatest fixpoint**.

Fixpoints

Let $(A, \sqsubseteq)$ be a complete lattice and $G : A \rightarrow A$ a monotonic function.

Definition:
Post $= \{x \mid G(x) \sqsubseteq x\}$ - the set of *postfix points* of G
(e.g. $\top$ is a postfix point)
Pre $= \{x \mid x \sqsubseteq G(x)\}$ - the set of *prefix points* of G
Fix $= \{x \mid G(x) = x\}$ - the set of *fixed points* of G.

Note that Fix $\subseteq$ Post.

Tarski's fixed point theorem

Theorem: Let $a = \sqcap\mathrm{Post}$. Then a is the least element of Fix (dually, $\sqcup\mathrm{Pre}$ is the largest element of Fix).

Proof:

Let x range over elements of Post.

- applying monotonic G from $a \sqsubseteq x$ we get $G(a) \sqsubseteq G(x) \sqsubseteq x$
- so $G(a)$ is a lower bound on Post, but a is the greatest lower bound, so $G(a) \sqsubseteq a$
- therefore $a \in \mathrm{Post}$
- Post is closed under G, by monotonicity, so $G(a) \in \mathrm{Post}$
- a is a lower bound on Post, so $a \sqsubseteq G(a)$
- from $a \sqsubseteq G(a)$ and $G(a) \sqsubseteq a$ we have $a = G(a)$, so $a \in \mathrm{Fix}$
- a is a lower bound on Post so it is also a lower bound on a smaller set Fix

In fact, the set of all fixpoints Fix is a lattice itself.

Tarski's fixed point theorem

Tarski's Fixed Point theorem shows that in a complete lattice with a monotonic function G on this lattice, there is at least one fixed point of G, namely the least fixed point $\sqcap$Post.

- Tarski's theorem guarantees fixpoints in complete lattices, but the above proof does not say how to find them.
- How difficult it is to find fixpoints depends on the structure of the lattice.

Let G be a monotonic function on a lattice. Let $a_0 = \bot$ and $a_{n+1} = G(a_n)$. We obtain a sequence $\bot \sqsubseteq G(\bot) \sqsubseteq G^2(\bot) \sqsubseteq \cdots$. Let $a_* = \bigsqcup_{n \geq 0} a_n$.

Lemma: The value a_* is a prefix point.
Observation: a_* need not be a fixpoint (e.g. on lattice [0,1] of real numbers).

Omega continuity

Definition: A function G is ω-continuous if for every chain $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \sqsubseteq x_n \sqsubseteq \ldots$ we have

$$G(\bigsqcup_{i \geq 0} x_i) = \bigsqcup_{i \geq 0} G(x_i)$$

Lemma: For an ω-continuous function G, the value $a_* = \bigsqcup_{n \geq 0} G^n(\bot)$ is the least fixpoint of G.

Iterating sequences and omega continuity

Lemma: For an ω-continuous function G, the value $a_* = \bigsqcup_{n\geq 0} G^n(\bot)$ is the least fixpoint of G.

Proof:

- By definition of ω-continuous we have $G(\bigsqcup_{n\geq 0} G^n(\bot)) = \bigsqcup_{n\geq 0} G^{n+1}(\bot) = \bigsqcup_{n\geq 1} G^n(\bot)$.
- But $\bigsqcup_{n\geq 0} G^n(\bot) = \bigsqcup_{n\geq 1} G^n(\bot) \sqcup \bot = \bigsqcup_{n\geq 1} G^n(\bot)$ because $\bot$ is the least element of the lattice.
- Thus $G(\bigsqcup_{n\geq 0} G^n(\bot)) = \bigsqcup_{n\geq 0} G^n(\bot)$ and a_* is a fixpoint.

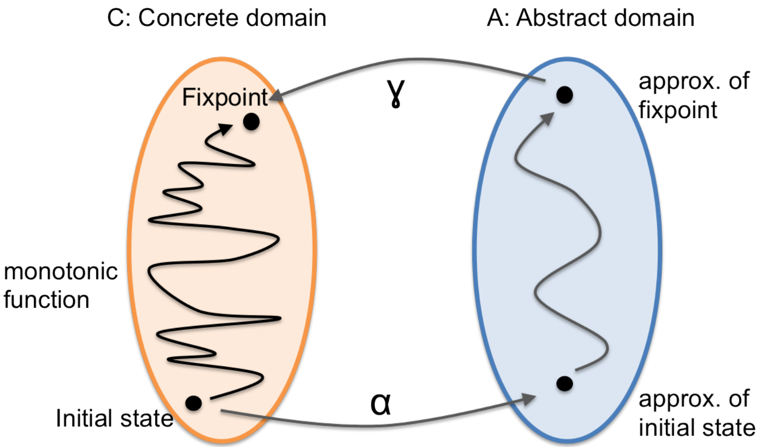
Now let's prove it is the least. Let c be such that $G(c) = c$. We want $\bigsqcup_{n\geq 0} G^n(\bot) \sqsubseteq c$. This is equivalent to $\forall n \in \mathbb{N}. G^n(\bot) \sqsubseteq c$.
We can prove this by induction : $\bot \sqsubseteq c$ and if $G^n(\bot) \sqsubseteq c$, then by monotonicity of G and by definition of c we have $G^{n+1}(\bot) \sqsubseteq G(c) \sqsubseteq c$.

Iterating sequences and omega continuity

Lemma: For an ω-continuous function G, the value $a_* = \bigsqcup_{n \geq 0} G^n(\perp)$ is the least fixpoint of G.

When the function is not ω-continuous, then we obtain a_* as above (we jump over a discontinuity) and then continue iterating. We then take the limit of such sequence, and the limit of limits etc., ultimately we obtain the fixpoint.

Abstract Interpretation Big Picture



Galois Connection

Galois connection (named after Évariste Galois) is defined by two monotonic functions $\alpha : C \to A$ and $\gamma : A \to C$ between partial orders $\leq$ on C and $\sqsubseteq$ on A, such that

$$\forall c, a. \quad \alpha(c) \sqsubseteq a \iff c \leq \gamma(a) \qquad (*)$$

(intuitively the condition means that c is approximated by a).

Lemma: The condition $(*)$ holds iff the conjunction of these two conditions:

$$\begin{array}{rcl} c & \leq & \gamma(\alpha(c)) \\ \alpha(\gamma(a)) & \sqsubseteq & a \end{array}$$

holds for all c and a.

Exercise

A Galois connection is defined by two monotonic functions $\alpha : C \to A$ and $\gamma : A \to C$ between partial orders $\leq$ on C and $\sqsubseteq$ on A, such that

$$\forall a, c. \quad \alpha(c) \sqsubseteq a \iff c \leq \gamma(a) \qquad (*)$$

(intuitively, the condition means that c is approximated by a).

a) Show that the condition $(*)$ is equivalent to the conjunction of these two conditions:

$$\begin{array}{rrcl} \forall c. & c & \leq & \gamma(\alpha(c)) \\ \forall a. & \alpha(\gamma(a)) & \sqsubseteq & a \end{array}$$

b) Let α and γ satisfy the condition of a Galois connection. Show that the following three conditions are equivalent:

1. $\alpha(\gamma(a)) = a$ for all a
2. α is a surjective function
3. γ is an injective function

c) State the condition for $c = \gamma(\alpha(c))$ to hold for all c. When C is the set of sets of concrete states and A is a domain of static analysis, is it more reasonable to expect that $c = \gamma(\alpha(c))$ or $\alpha(\gamma(a)) = a$ to be satisfied, and why?

Abstract Interpretation Recipe: Setup

Given control-flow graph: (V, E, r) where

- $V = \{v_1, \ldots, v_n\}$ is set of program points
- $E \subseteq V \times V$ are control-flow graph edges
- $r : E \to 2^{S \times S}$, so each $r(v, v') \subseteq S \times S$ is relation describing the meaning of command between v and v'

Key steps:

- design abstract domain A that represents sets of program states
- define $\gamma : A \to C$ giving meaning to elements of A
- define lattice ordering $\sqsubseteq$ on A such that $a_1 \sqsubseteq a_2 \to \gamma(a_1) \subseteq \gamma(a_2)$
- define $sp^\# : A \times 2^{S \times S} \to A$ that maps an abstract element and a CFG statement to new abstract element, such that $sp(\gamma(a), r) \subseteq \gamma(sp^\#(a, r))$
  For example, by defining function α so that (α, γ) becomes a *Galois Connection* and defining $sp^\#(a) = \alpha(sp(\gamma(a), r))$.

Running Abstract Interpretation

- Extend $sp^{\#}$ to work on control-flow graphs, by defining $F^{\#} : (V \to A) \to (V \to A)$ as follows (below, $g^{\#} : V \to A$)

$$F^{\#}(g^{\#})(v') = \mathit{Init}(v') \sqcup \bigsqcup_{(v,v') \in E} sp^{\#}(g^{\#}(v), r(v, v'))$$

- Compute $g^{\#}_{*} = \mathit{lfp}(F^{\#})$ (this is easier than computing semantics because lattice A^n is simpler than C^n):

$$g^{\#}_{*} = \bigsqcup_{n \geq 0} (F^{\#})^n(\bot^{\#})$$

  where $\bot^{\#}(v) = \bot_A$ for all $v \in V$.

The resulting fixpoint describes an inductive program invariant.

Concrete Domain: Sets of States

Because there is only one variable:

- state is an element of $\mathbb{Z}$ (value of x)
- sets of states are sets of integers, $C = 2^{\mathbb{Z}}$ (concrete domain)
- for each command K, strongest postcondition function $sp(\cdot, K) : C \to C$

Strongest Postondition

Compute *sp* on example statements:

$$sp(P, x := 0) = \{0\}$$

$$sp(P, x := x + 3) = \{x + 3 \mid x \in P\}$$

$$sp(P, assume(x < 10)) = \{x \mid x \in P \land x < 10\}$$

$$sp(P, assume(\neg(x < 10)) = \{x \mid x \in P \land x \geq 10\}$$

Sets of States at Each Program Point

Collecting semantics computes with sets of states at each program point

$$g : \{v_0, v_1, v_2, v_3\} \to C$$

We sometimes write g_i as a shorthand for $g(v_i)$, for $i \in \{0, 1, 2, 3\}$.
In the initial state the value of variable is arbitrary: $I = \mathbb{Z}$

post Function for the Collecting Semantics
From here we can derive F that maps g to new value of g:

$$\begin{array}{l} F(g_0, g_1, g_2, g_3) = \\ \;(\mathbb{Z}, \\ \;sp(g_0, x := 0) \cup sp(g_2, x := x + 3), \\ \;sp(g_1, assume(x < 10)), \\ \;sp(g_1, assume(\neg(x < 10)))) \end{array}$$

Sets of States at Each Program Point

The fixpoint condition $F(g) = g$ becomes a system of equations

$$\begin{aligned}
&g_0 = \mathbb{Z} \\
&g_1 = sp(g_0, x := 0) \cup sp(g_2, x := x + 3) \\
&g_2 = sp(g_1, assume(x < 10)) \\
&g_3 = sp(g_1, assume(\neg(x < 10))))
\end{aligned}$$

whereas the postfix point (see Tarski's fixpoint theorem) becomes

$$\begin{aligned}
&\mathbb{Z} \subseteq g_0 \\
&sp(g_0, x := 0) \cup sp(g_2, x := x + 3) \subseteq g_1 \\
&sp(g_1, assume(x < 10)) \subseteq g_2 \\
&sp(g_1, assume(\neg(x < 10))) \subseteq g_3
\end{aligned}$$

Computing Fixpoint

To find the fixpoint, we compute the sequence $F^n(\emptyset, \emptyset, \emptyset, \emptyset)$ for $n \geq 0$:

$$\begin{array}{l}
(\emptyset, \emptyset, \emptyset, \emptyset) \\
(\mathbb{Z}, \emptyset, \emptyset, \emptyset) \\
(\mathbb{Z}, \{0\}, \emptyset, \emptyset) \\
(\mathbb{Z}, \{0\}, \{0\}, \emptyset) \\
(\mathbb{Z}, \{0, 3\}, \{0\}, \emptyset) \\
(\mathbb{Z}, \{0, 3\}, \{0, 3\}, \emptyset) \\
(\mathbb{Z}, \{0, 3, 6\}, \{0, 3\}, \emptyset) \\
(\mathbb{Z}, \{0, 3, 6\}, \{0, 3, 6\}, \emptyset) \\
(\mathbb{Z}, \{0, 3, 6, 9\}, \{0, 3, 6, 9\}, \emptyset) \\
(\mathbb{Z}, \{0, 3, 6, 9, 12\}, \{0, 3, 6, 9\}, \emptyset) \\
(\mathbb{Z}, \{0, 3, 6, 9, 12\}, \{0, 3, 6, 9\}, \{12\}) \\
(\mathbb{Z}, \{0, 3, 6, 9, 12\}, \{0, 3, 6, 9\}, \{12\})
\end{array}$$

Thus, all subsequent values remain the same and $(\mathbb{Z}, \{0, 3, 6, 9, 12\}, \{0, 3, 6, 9\}, \{12\})$ is the fixpoint of collecting semantics equations. In general we may need infinitely many iterations to converge.

Question

Suppose that we have a program that terminates for every possible initial state. Can we always find a finite constant n such that

$$F^n(\emptyset, \ldots, \emptyset) = F^{n+1}(\emptyset, \ldots, \emptyset)$$

i.e. the sequence such as the one above is guaranteed to stabilize?

Example: Assume an arbitrary initial value and consider the loop. Compute a sequence of sets of states at the point after the increment statement in the loop, following the equations for collecting semantics.

```
if (y > 0) {
  x = 0
  while (x < y) {
    x = x + 1
  }
}
```

What always works from omega continuity:

$$lfp(F) = \bigcup_{n \geq 0} F^n(\emptyset, \ldots, \emptyset)$$

where $\bigcup$ on a tuple above means taking union of each component separately, so $(A, B) \cup (A', B') = (A \cup A', B \cup B')$.

Variable Range Analysis for Example Program

The general form of abstract interpretation of the collecting semantics is analogous to collecting semantics, but replaces operations on sets with operations on the lattice:

$$F^{\#} : (V \to A) \to (V \to A)$$

$$F(g^{\#})(v') = g^{\#}_{init}(v') \sqcup \bigsqcup_{(v,v') \in E} sp^{\#}(g^{\#}(v), r(v, v'))$$

Here $g^{\#}_{init}(v')$ will be $\bot$ except at the entry into our control-flow graph, where it approximates the set of initial states at the entry point.

Abstract Analysis Domain

Before we had representation for all possible sets of states:

$$C = 2^{\mathbb{Z}}$$

Here we have representation of only certain states, namely intervals:

$$\begin{aligned} A = \ & \{\bot\} \cup \\ & \{(-\infty, q] \mid q \in \mathbb{Z}\} \cup \\ & \{[p, +\infty) \mid p \in \mathbb{Z}\} \cup \\ & \{[p, q] \mid p \leq q\} \cup \\ & \{\top\} \end{aligned}$$

Abstract Analysis Domain

The meaning of domain elements is given by a monotonic *concretization function* $\gamma : A \to C$:

$$\begin{array}{rcl} \gamma(\bot) & = & \emptyset \\ \gamma(\{(-\infty, q]) & = & \{x \mid x \leq q\} \\ \gamma(\{[p, +\infty)) & = & \{x \mid p \leq x\} \\ \gamma(\{[p, q]) & = & \{x \mid p \leq x \wedge x \leq q\} \\ \gamma(\top) & = & \mathbb{Z} \end{array}$$

From monotonicity and $a_1 \sqsubseteq a_1 \sqcup a_2$ it follows

$$\gamma(a_1) \;\subseteq\; \gamma(a_1 \sqcup a_2)$$

and thus

$$\gamma(a_1) \cup \gamma(a_2) \;\subseteq\; \gamma(a_1 \sqcup a_2)$$

We try to define γ to be as small as possible while satisfying this condition.

Abstract Analysis Domain

Define *abstraction function* $\alpha : C \rightarrow A$ such that

- $\alpha(s) = [\min s, \max s]$ if those values exist (set is bounded from below and above)
- $\alpha(s) = [\min s, +\infty)$ if there is lower but no upper bound
- $\alpha(s) = (-\infty, \max s]$ if there is upper but no lower bound
- $\alpha(s) = \top$ if there is no upper and no lower bound
- $\alpha(\emptyset) = \bot$

Lemma: The pair (α, γ) form a Galois Connection.

Abstract Analysis Domain

By property of Galois Connection, the condition $\gamma(a_1) \cup \gamma(a_2) \ \subseteq \ \gamma(a_1 \sqcup a_2)$ is equivalent to

$$\alpha(\gamma(a_1) \cup \gamma(a_2)) \ \sqsubseteq \ a_1 \sqcup a_2$$

To make $a_1 \sqcup a_2$ as small as possible, we let the equality hold, defining

$$a_1 \sqcup a_2 \ = \ \alpha(\gamma(a_1) \cup \gamma(a_2))$$

For example,

$$\begin{array}{rcl} [0,2] \sqcup [5,8] & = & \alpha(\gamma([0,2]) \cup \gamma([5,8])) \\ & = & \alpha(\{0,1,2,5,6,7,8\}) \\ & = & [0,8] \end{array}$$

Abstract Postcondition

We had: $sp(\cdot, c) : C \to C$
Now we have: $sp^{\#}(\cdot, c) : A \to A$
For correctness, we need that for each $a \in A$ and each command r:

$$sp(\gamma(a), r) \subseteq \gamma(sp^{\#}(a, r))$$

We would like $sp^{\#}$ to be *as small as possible so that this condition holds.*
By property of Galois Connection, the condition $sp(\gamma(a), r) \subseteq \gamma(sp^{\#}(a, r))$ is equivalent to

$$\alpha(sp(\gamma(a), r)) \sqsubseteq sp^{\#}(a, r)$$

Because we want $sp^{\#}$ to be as small as possible (to obtain correct result), we let equality hold:

$$sp^{\#}(a, r) = \alpha(sp(\gamma(a), r))$$

Because we know α, γ, sp, we can compute the value of $sp^{\#}(a, r)$ by simplifying certain expressions involving sets of states.

Example

For $p \leq q$ we have:

$$\begin{array}{rcl} sp^{\#}([p,q], x := x+3) & = & \alpha(sp(\gamma([p,q]), x := x+3)) \\ & = & \alpha(sp(\{x \mid \ p \leq x \wedge x \leq q\}, x := x+3)) \\ & = & \alpha(\{x+3 \mid p \leq x \wedge x \leq q\}) \\ & = & \alpha(\{y \mid p+3 \leq y \wedge y \leq q+3\}) \\ & = & [p+3, q+3] \end{array}$$

For K an integer constant and $a \neq \perp$, we have

$$sp^{\#}(a, x := K) = [K, K]$$

Note that for every command given by relation r, we have

$$\begin{array}{rcl} sp^{\#}(\perp, r) & = & \alpha(sp(\gamma(\perp), r)) \\ & = & \alpha(sp(\emptyset, r)) \\ & = & \alpha(\emptyset) \\ & = & \perp \end{array}$$

Abstract Semantic Function for the Program

In Collecting Semantics for Example Program we had

$$\begin{aligned}&F(g_0, g_1, g_2, g_3) = \\ &\ (\mathbb{Z}, \\ &\ sp(g_0, x := 0) \cup sp(g_2, x := x + 3), \\ &\ sp(g_1, assume(x < 10)), \\ &\ sp(g_1, assume(\neg(x < 10))))\end{aligned}$$

Here we have:

$$\begin{aligned}&F^{\#}(g_0^{\#}, g_1^{\#}, g_2^{\#}, g_3^{\#}) = \\ &\ (\top, \\ &\ sp^{\#}(g_0^{\#}, x := 0) \sqcup sp^{\#}(g_2^{\#}, x := x + 3), \\ &\ sp^{\#}(g_1^{\#}, assume(x < 10)), \\ &\ sp^{\#}(g_1^{\#}, assume(\neg(x < 10))))\end{aligned}$$

Solving Abstract Function

Doing the analysis means computing $(F^{\#})^n(\bot, \bot, \bot, \bot)$ for $n \geq 0$:

$$
\begin{array}{l}
(\bot, \bot, \bot, \bot) \\
(\top, \bot, \bot, \bot) \\
(\top, [0,0], \bot, \bot) \\
(\top, [0,0], [0,0], \bot) \\
(\top, [0,3], [0,3], \bot) \\
(\top, [0,3], [0,3], \bot) \\
(\top, [0,6], [0,3], \bot) \\
(\top, [0,6], [0,6], \bot) \\
(\top, [0,9], [0,9], \bot) \\
(\top, [0,12], [0,9], \bot) \\
(\top, [0,12], [0,9], [10,12]) \\
(\top, [0,12], [0,9], [10,12]) \\
\ldots
\end{array}
$$

Note the approximation (especially in the last step) compared to the collecting semantics we have computed before for our example program.

Exercises

Exercise 1:
Consider an analysis that has two integer variables, for which we track intervals, and one boolean variable, whose value we track exactly.
Give the type of $F^{\#}$ for such program.

Exercise 2:
Consider the program that manipulates two integer variables x, y.
Consider any assignment $x = e$, where e is a linear combination of integer variables, for example,

$$x = 2 * x - 5 * y$$

Consider an interval analysis that maps each variable to its value.
Describe an algorithm that will, given a syntax tree of $x = e$ and intervals for x (denoted $[a_x, b_x]$) and y (denoted $[a_y, b_y]$) find the new interval $[a, b]$ for x after the assignment statement.

Exercise 3

a)

For a program whose state is one integer variable and whose abstraction is an interval, derive general transfer functions $sp^{\#}(a, c)$ for the following statements c, where K is an arbitrary compile-time constant known in the program:

- x= K;
- x= x + K;
- assume(x $\leq$ K)
- assume(x $\geq$ K)

b)

Consider a program with two integer variables, x,y. Consider analysis that stores one interval for each variable.

- Define the domain of lattice elements a that are computed for each program point.
- Give the definition for statement $sp^{\#}(a, y = x + y + K)$

Exercise 3 c)

Draw the control-flow graph for the following program.

Run abstract interpretation that maintains an interval for *x* at each program point, until you reach a fixpoint.

What are the fixpoint values at program points v_4 and v_5?

```
// v0
x := 0;
// v1
while (x < 10) {
  // v2
  x := x + 3;
}
// v3
if (x >= 0) {
  if (x <= 15) {
    a[x]=7; // made sure index is within range
  } else {
    // v4
    error;
  }
} else {
  // v5
  error;
}
```

Termination and Efficiency of Abstract Interpretation Analysis

Definition: A **chain** of length n is a sequence $s_0, s_1, \ldots, s_n$ such that

$$s_0 \sqsubset s_1 \sqsubset s_2 \sqsubset \ldots \sqsubset s_n$$

where $x \sqsubset y$ means, as usual, $x \sqsubseteq y \wedge x \neq y$

Definition: A partial order has a **finite height** n if it has a chain of length n and every chain is of length at most n.

A finite lattice is of finite height.

Example

The constant propagation lattice $\mathbb{Z} \cup \{\bot, \top\}$ is an infinite lattice of height 2. One example chain of length 2 is

$$\bot \sqsubset 42 \sqsubset \top$$

Here the γ function is given by

- $\gamma(k) = \ldots$ when $k \in \mathbb{Z}$
- $\gamma(\top) = \ldots$
- $\gamma(\bot) = \ldots$

The ordering is given by $a_1 \subseteq a_2$ iff $\gamma(a_1) \subseteq \gamma(a_2)$

Example

If a state of a (one-variable) program is given by an integer, then a concrete lattice element is a set of integers. This lattice has infinite height. There is a chain

$$\{0\} \subset \{0,1\} \subset \{0,1,2\} \subset \ldots \subset \{0,1,2,\ldots,n\}$$

for every n.

Convergence in Lattices of Finite Heignth

Consider a finite-height lattice $(L, \sqsubseteq)$ of height n and function

$$F : L \rightarrow L$$

What is the maximum length of sequence $\bot, F(\bot), F^2(\bot), \ldots$?
Give an effectively computable expression for $lfp(F)$.

Computing the Height when Combining Lattices

Let $H(L, \leq)$ denote the height of the lattice $(L, \leq)$.
Product
Given lattices $(L_1, \sqsubseteq_1)$ and $(L_2, \sqsubseteq_2)$, consider product lattice with set $L_1 \times L_2$ and potwise order

$$(x_1, x_2) \sqsubseteq (x_1', x_2')$$

iff ...
What is the height of the product lattice?
Exponent
Given lattice $(L, \sqsubseteq)$ and set V, consider the lattice $(L^V, \sqsubseteq')$ defined by

$$g \sqsubset' h$$

iff $\forall v \in V . g(v) \sqsubseteq h(v)$.
What is the height of the exponent lattice?

Computing the Height when Combining Lattices

Let $H(L, \leq)$ denote the height of the lattice $(L, \leq)$.

Product

Given lattices $(L_1, \sqsubseteq_1)$ and $(L_2, \sqsubseteq_2)$, consider product lattice with set $L_1 \times L_2$ and potwise order

$$(x_1, x_2) \sqsubseteq (x_1', x_2')$$

iff ...

What is the height of the product lattice?

Exponent

Given lattice $(L, \sqsubseteq)$ and set V, consider the lattice $(L^V, \sqsubseteq')$ defined by

$$g \sqsubset' h$$

iff $\forall v \in V.g(v) \sqsubseteq h(v)$.

What is the height of the exponent lattice?

Answer: height of L times the cardinality of V.

Widening and Narrowing in Variable Range Analysis

Interval analysis domain, for each program point, maps each program variable to an interval. Analysis domain has elements $g^{\#} : V \to I$ where I denotes the set of such intervals.

Height of lattice for unbounded integers: infinite.

Height of lattice of one interval for 64-bit integers: around 2^{64}

Moreover, if we have q variables in program and p program points, height of lattice for the analysis domain is pq times larger.

How to guarantee (reasonably fast) termination?

Widening technique

If the iteration does not seem to be converging, take a "jump" and make the interval much **wider** (larger).
Finite set of *jump points* J (e.g. set of all integer constants in the program)
In fixpoint computation, compose H_i with function

$$w([a, b]) = [\max\{x \in J \mid x \leq a\}, \min\{x \in J \mid b \leq x\}]$$

We require the condition:

$$x \sqsubseteq W(x)$$

for all x.
The condition holds for the example above.

Approaches

- always apply widening (we will assume this)
- iterate a few times with H_i only (without using w), if we are not at a fixpoint at this program point, then widen.
- this is not monotonic: if you start at fixpoint, it converges, if start below, can jump over fixpoint!

Standard iteration: $\bot, \ldots, (F^{\#})^n(\bot), \ldots$
Widening: $\bot, \ldots, ((W \circ F^{\#})^n(\bot), \ldots$

Example where widening works nicely

Consider program:

```
x = 0;
while (x < 1000) {
  x = x + 1;
}
```

Interval analysis without widening will need around 1000 iterations to converge to interval $[1000, 1000]$ for x at the end of the program.
This may be too slow.

Let us derive the set J by taking all constants that appear in the program, as well as $-\infty$ and $+\infty$:

$$J = \{-\infty, 0, 1, 1000, +\infty\}$$

After a few iterations, widening maps interval $[0, 2]$ into $[0, 1000]$. This gives $[0, 999]$ for x at loop entry and again $[1000, 1000]$ for x at the end of the program, but in many fewer iterations.

Example showing problems with widening

Consider program:

```
x = 0;
y = 1;
while (x < 1000) {
  x = x + 1;
  y = 2*x;
  y = y + 1;
  print(y);
}
```

Interval analysis without widening will need around 1000 iterations to converge to

$$x \mapsto [1000, 1000]; \quad y \mapsto [1, 2001]$$

This may be too slow.
Now apply widening with the same J as before. When within loop we obtain $x \mapsto [0, 1000]$, applying widening function to the interval $[0, 2000]$ for y results in $[0, +\infty)$. We obtain $y \mapsto [1, +\infty)$ at the end of the program:

$$x \mapsto [1000, 1000]; \quad y \mapsto [1, +\infty)$$

Narrowing

Observation

Consider a monotonic function, such as $f(x) = 0.5x + 1$ on the set of real numbers.
If we consider a sequence $x_0, f(x_0), \ldots,$ this sequence is

- monotonically increasing iff $x_0 < x_1$ (e.g. for $x_0 = 0$)
- monotonically decreasing iff $x_1 < x_0$ (e.g. for $x_0 = 3$)

Informally, the sequence continues of the direction in which it starts in the first step.

This is because $x_0 < x_1$ implies by monotonicity of f that $x_1 < x_2$ etc., whereas $x_1 < x_0$ implies $x_2 < x_1$.

The Idea

Let $W : A \to A$ such that $x \sqsubseteq W(x)$.
After finding fixpoint of $(W \circ F)^{\#}$, apply $F^{\#}$ to improve precision.

Widen and Narrow

Lemma: Let $F^{\#}$ and W be monotonic functions on a partial order $\sqsubseteq$ such that $x \sqsubseteq W(x)$ for all x. Define the following:

- $x_* = \sqcup_{n \geq 0} (F^{\#})^n(\bot)$
- $y_* = \sqcup_{n \geq 0} (W \circ F^{\#})^n(\bot)$
- $z_* = \sqcap_{n \geq 0} (F^{\#})^n(y_*)$

where we also assume that the two $\sqcup$ and one $\sqcap$ exist. Then

- x_* is the least fixpoint of $F^{\#}$ and z_*, is the least fixpoint of $W \circ F^{\#}$ (by Tarski's Fixpoint Theorem), and
- $x_* \sqsubseteq z_* \sqsubseteq y_*$.

Proof

By induction, for each n we have

$$(F^{\#})^n(\bot) \sqsubseteq (W \circ F^{\#})^n(\bot)$$

Thus by Comparing Fixpoints of Sequences, we have $x_* \sqsubseteq y_*$.
Next, we have that

$$x_* = F^{\#}(x_*) \sqsubseteq F^{\#}(y_*) \sqsubseteq (W \circ F^{\#})(y_*) \sqsubseteq y_*$$

Thus, $F^{\#}(y_*) \sqsubseteq y_*$. From there by induction and monotonicity of $F^{\#}$ we obtain

$$(F^{\#})^{n+1}(y_*) \sqsubseteq (F^{\#})^n(y_*)$$

i.e. the sequence $(F^{\#})^n(y_*)$ is **decreasing**. Therefore, y_* is its upper bound and therefore $z_* \sqsubseteq y_*$.
On the other hand, we have by monotonicity of $F^{\#}$, the fact that x_* is fixpoint, and $x_* \sqsubseteq y_*$ that:

$$x_* = (F^{\#})^n(x_*) \sqsubseteq (F^{\#})^n(y_*)$$

Thus, x_* is the lower bound on $(F^{\#})^n(y_*)$, so $x_* \sqsubseteq z_*$.

Note

Even if z_* does not exist, we can simply compute $(F^{\#})^n(y_*)$ for any chosen value of n, it is still a sound over-approximation, because it approximates x_*, which approximates the concrete value:

$$x_* \sqsubseteq z_n$$

so

$$s_* \subseteq \gamma(x_*) \subseteq \gamma(z_n)$$

Being able to stop at any point gives us an **anytime algorithm**.

Example showing how narrowing may improve result after widening

In the above example for the program, the results obtained using widening are:

```
x = 0;
y = 1;
// x -> [0,0], y -> [1,1]
// (merge point)
// x -> [0,1000], y -> [1,+infty)
while (x < 1000) {
  // x -> [0,999], y -> [1,+infty)
  x = x + 1;
  // x -> [0,1000], y -> [1,+infty)
  y = 2*x;
  // x -> [0,1000], y -> [0,+infty)
  y = y + 1;
  // x -> [0,1000], y -> [1,+infty)
  print(y);
}
// x -> [1000,1000], y -> [1,+infty)
```

Example cont.

Let us now apply one ordinary iteration, without widening. We obtain:

```
x = 0;
y = 1;
// x -> [0,0], y -> [1,1]
// (merge point)
// x -> [0,1000], y -> [1,2001]
while (x < 1000) {
   // x -> [0,999], y -> [1,+infty)
   x = x + 1;
   // x -> [0,1000], y -> [1,+infty)
   y = 2*x;
   // x -> [0,1000], y -> [0,2000]
   y = y + 1;
   // x -> [0,1000], y -> [1,2001]
   print(y);
}
// x -> [1000,1000], y -> [1,2001]
```

Thus, we obtained a good first approximation by a few iterations with widening and then improved it with a single iteration without widening.