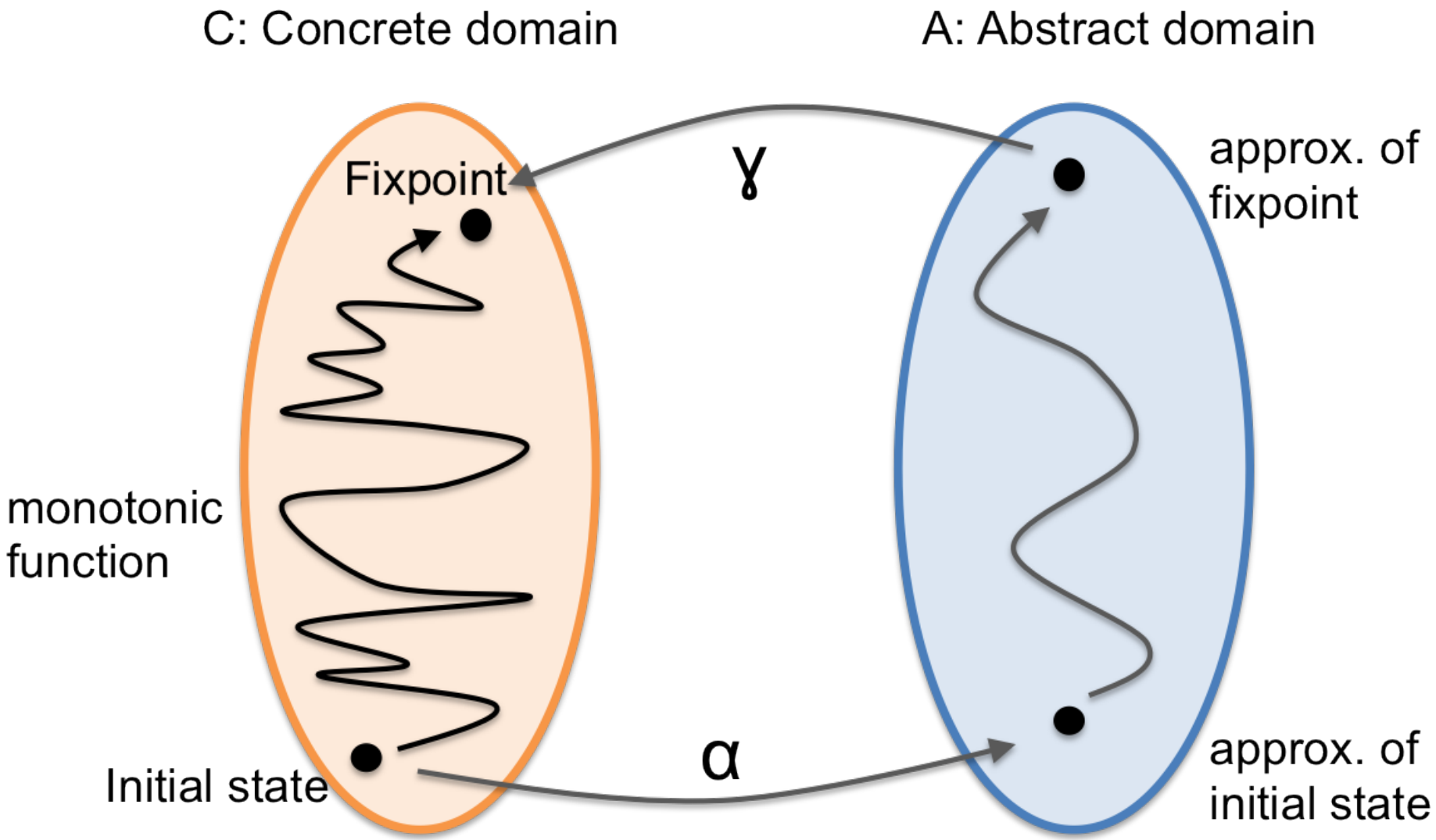
Lecturecise 10
Galois Connection and Abstract Interpretation

Viktor Kuncak

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:

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

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:

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

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, $g^{\#} : V \to A$)

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

- Compute $g_*^{\#} = lfp(F^{\#})$ (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:

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

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

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

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

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:

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

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 $n \geq 0$:

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

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:

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

Abstract Analysis Domain

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

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

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* $\alpha : C \to A$ 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,

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

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:

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

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

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

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

$$\begin{aligned} sp^{\#}(\bot, r) &= \alpha(sp(\gamma(\bot), r)) \\ &= \alpha(sp(\emptyset, r)) \\ &= \alpha(\emptyset) \\ &= \bot \end{aligned}$$

Abstract Semantic Function for the Program

In Collecting Semantics for Example Program we had

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

Here we have:

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

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.

```
// 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

$$F : L \to L$$

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?

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

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

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:

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

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.