Program Analysis: Lecture 2- Chaotic Iterations

Yuval Rochman and Michal Shagam

November 15, 2015

Contents

1 References

The presentation is based on [1, 2].

2 Mathematical Background

In this section we define and use mathematical objects in order to express a result of a program analysis, the exact solution and compare between different solutions.

2.1 Posets

Definition 1. *We define a* ***poset*** *as a pair $(L, \sqsubseteq)$, where $\sqsubseteq$ is a binary relation of defining a* ***partial ordering*** *over L. That means, $\sqsubseteq$ satisfies the following properties:*

1. *Reflexivity: for all $l \in L$, we have $l \sqsubseteq l$.*
2. *Transitivity: Given $l_1, l_2.l_3 \in L$, if $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_3$ then $l_1 \sqsubseteq l_3$.*
3. *Anti-symmetric: Given $l_1, l_2 \in L$, if $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_1$ then $l_1 = l_2$.*

Note that if $\sqsubseteq$ is a partial order, then it **does not** imply that comparing any two elements can be done. That means, there can be two elements $l_1, l_2 \in L$ such that neither $l_1 \sqsubseteq l_2$ nor $l_2 \sqsubseteq l_1$. If a partial order $\sqsubseteq$ can compare every two elements - then $\sqsubseteq$ is called a **total order** relation.

In the context of program analysis, an element $l \in L$ represents a group of available states in the program analysis. If $l_1 \sqsubseteq l_2$ then l_1 represents a subset of l_2 solutions. Thus, l_1 is more precise than l_2. The "smaller" the group of states is, the more precise.

For example, the partial order is not defined for the two sets of states S_1 and S_2 because they each have a state that is not included in the other. $S_1 = \{ a_1, a_2\}$ $S_2 = \{ a_1, a_3\}$

Examples of posets:

1. The total order over the natural numbers ($\mathbb{N}$,$\leq$). This is also a full order, as we can compare between any two elements.

2. The subset order of a powerset of a set S, i.e, $(P(S), \subseteq)$. Note that usually this is **not** a full order. For example, if $S = \{a, b\}$ then $\{a\}, \{b\} \in P(S)$, but neither $\{a\} \sqsubseteq \{b\}$ nor $\{b\} \sqsubseteq \{a\}$.

3. The superset order of a powerset of a set S, i.e, $(P(S), \supseteq)$.

4. The Even \Odd program analysis used in the first Lecture. In the program analysis, we associate an abstraction $L = \{E, O, ?\}$ for an integer variables, where ? represents that the variable is unknown and can be Even or Odd. In this case, we have $Even \subset ?$ and $Odd \subset ?$. We can add a **bottom** $\bot$ to L, so that $\bot \subset Odd, Even$ as we will see later

5. The Constant Propagation program analysis used in the first Lecture. In the program analysis, we use the abstraction $L = Z \cup \{?\}$ for integer variables. Similarly to the the Even \Odd abstraction, ? represents an unknown variable, and $nsqsubset?$ for all $n \in Z$.

6. The Interval abstraction used in the first Lecture, $L =$ ZxZ is the set of all intervals, and $[a, b] \sqsubseteq [c, d]$ iff $a \geq c$ and $b \leq d$.

7. Given a graph $L =< V, E >$, we define $v_1 \sqsubseteq v_2$ iff there is a path between v_1 and v_2. Note that $(L, \sqsubseteq)$ is a poset if and only if L is a acyclic graph (due to the anti-symmetric property).

The pair $(N, <)$ is **not** a poset, as the relation is not reflexive: i.e $3 \not< 3$. Another important example that is not a poset is a graph and the STCONN relation (whether there's a path from s to t). The relation is reflexive and transitive but a path between s and t and a path between t and s does not mean that s and t are the same node.

2.2 Lower and Upper bounds

Definition 2. *Let $(L, \sqsubseteq)$ be a poset and L' a subset of L ($L' \subset L$).*

1. *$l \in L$ is a **lower bound** of L' if l' is smaller than every element $l' \in L$ i.e. $l' \sqsubseteq l$.*

2. $l \in L$ *is a* ***upper bound*** *of* L' *if* l' *is larger than every element* $l' \in L$ *i.e.* $l' \sqsupseteq l$.

3. $l \in L$ *is the* ***greatest lower bound*** *of* L' *if* l *is larger than every lower bound of* L'. *The greatest lower bound is alternativly called* ***meet*** *of* L', *denoted by* $\sqcap L$.

4. $l \in L$ *is the* ***least upper bound*** *of* L' *if* l *is smaller than every upper bound. The lowest upper bound is alternativly called* ***join*** *of* L', *denoted by* $\sqcup L$.

For example, we consider a poset $(P(S), \subseteq)$ where $S = \{a, b, c\}$ and $L' = \{\{a\}, \{a, b\}\} \subseteq P(S)$. Then $\{a, b\}$ and $\{a, b, c\}$ are the upper bounds of L', while $\{a\}$ and $\emptyset$ are lower bounds of L'. The greatest lower bound is $\sqcap L = \{a\}$ and the least upper bound is $\sqcup L = \{a, b\}$.

We show that if the meet (or the join) exists it must be unique.

Claim 2.1. *Let* $l_1, l_2 \in L$ *be greatest lower bound (meet) of a subset* L'. *Then* $l_1 = l_2$.

Proof. Since l_1 is the greatest lower bound of a subset L' and l_2 is a lower bound of L' then $l_1 \sqsubseteq l_2$. In a similiar way we show that $l_1 \sqsupseteq l_2$. By anti symmetric property, we imply that $l_1 = l_2$. □

Not always lower bounds (and upper bounds) exists. For example, the Even \Odd program $L = \{E, O, ?\}$ with the subset $L' = \{E, O\}$ has no lower bound (and no greater lower bound). Even if lower bounds exists, it does not ensure that a greatest lower bound (meet) exists. For example the subset $L' = \{?\}$ has three lower bounds $?, E, O$ but no greatest lower bound.

If we add the bottom $\bot$ is add to the program analysis set L (i.e, $L = \{\bot, E, O, ?\}$), then $\bot$ is the only lower bound of $L' = \{E, O\}$ and thus the meet.

2.3 Complete Lattices

Definition 3. *A poset* $(L, \sqsubseteq)$ *is called a* ***Complete Lattice*** *if every subset* L' *has a meet and a join. We define the* ***bottom*** $\bot$ *as smallest element in* L, *i.e* $\bot \sqsubseteq l$ *for all* $l \in L$. *The* ***top*** $\top$ *is defined as the largest element in* L *i.e,* $l \sqsubseteq \top$ *for every* $l \in L$.

Note that in every complete lattice there is a top and a bottom, as $\bot = \sqcup\emptyset = \sqcap L$ and $\top = \sqcap\emptyset = \sqcup L$.

The following list contains examples of complete lattices:

1. The total order $(N \cup \{\infty\}, \leq)$. The top is $\top = \infty$, the bottom is $\bot = 1$.

2. The powersets $(P(s), \subseteq)$. The top is $\top = P(S)$, the bottom is $\bot = \emptyset$.

3. The powersets $(P(s), \supseteq)$. The top is $\top = \emptyset$, the bottom is $\bot = P(S)$.

4. The Constant Propagation and the Even \Odd analyses. In these analyses, we add an artificial bottom $\bot$ so that these programs will be complete lattices. The top is $\top =?$.

5. The interval abstraction. The bottom is $\bot = \emptyset$ and the top is $\top = [-\infty, \infty]$.

Another example of a complete lattice is shown in 1 for ({a,b,c}, $\subseteq$). For the complete lattice ({a,b,c}, $\supseteq$), the diagram should be flipped.

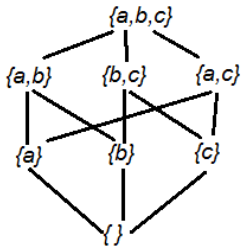

Figure 1: This diagram displays the lattice for ({a,b,c}, $\subseteq$).

Next, we show that every poset $(L, \sqsubseteq)$ such that every subset L' has a meet, must imply that every subset L' also has a join. This is stated in the next claim.

Claim 2.2. *Let $(L, \sqsubseteq)$ be a poset. The following condition are equivalent:*

1. *L is a complete lattice.*
2. *Every subset of L has a lowest upper bound (join operator is well defined).*
3. *Every subset of L has an greatest lower bound (meet operator is well defined).*

Proof. By definition, 1 is equivalent to 2+3. If we show that 2 equivalent to 3- the claim is proven. We will show 2 implies 3 (in a symmetrical way, 3 implies 2).

Let L' be a subset of L. We define $U = \{u \in L | u \text{ is an upper bound of } L'\}$, and $u_0 = \sqcap U$. Then we will show that u_0 is the join of L'.

Let l' be an element in L'. Then for every $u \in U$ we have $l' \sqsubseteq u$. Thus, l' is a lower bound of U. The element u_0 is the meet of U, and is greater than every lower bound, and in particular $l' \sqsubseteq u_0$. Since l' is a random element in L' this proves that u_0 is an upper bound.

Let u be an upper bound of L', i.e, $u \in U$. By the definition of a meet, $u_0 \sqsubseteq u$. Thus u_0 is the least upper bound of L' i.e. the join of L'. As required □

2.4 Other definitions

Definition 4. *Let $(L_1, \sqsubseteq_1)$ and Let $(L_2, \sqsubseteq_2)$ be two complete lattices. We define the **cartesian product** $L = (L_1 \times L_2, \sqsubseteq)$ such that $(x_1, x_2) \sqsubseteq (y_1, y_2)$ if $x_1 \sqsubseteq_1 y_1$ and $x_2 \sqsubseteq_2 y_2$.*

In program analysis cartesian products can be used for a composition of lattices. For example, to define the states of every variable defined in the program.

Claim 2.3. *If L_1 and L_2 are complete lattices then the cartesian product $L = (L_1 \times L_2, \sqsubseteq)$ is also a complete lattice. Where $\sqsubseteq$ is defined as the cartesian of the $\sqsubseteq$ for each of the lattices.*

Proof. For two lattices L_1 and L_2 defined by the tuples (L_1, $\sqsubseteq_1$, $\bigsqcup_1$, $\sqcap_1$, $\bot_1$, $\top_1$) and (L_2, $\sqsubseteq_2$, $\bigsqcup_2$, $\sqcap_2$, $\bot_2$, $\top_2$), we will define a new lattice L_3 that will be the cartesian product of the two lattices.
We will define $L_3 = (L_1 X L_2, \sqsubseteq_1 X \sqsubseteq_2, \bigsqcup_1 X \bigsqcup_2, \sqcap_1 X \sqcap_2, \bot_1 X \bot_2, \top_1 X \top_2)$
$(x_1, x_2) \sqsubseteq_3 (y_1, y_2)$ if $x_1 \sqsubseteq_1 y_1$ and $x_2 \sqsubseteq_2 y_2$
The order defined above is a partial order. It is reflexive because both the posets it uses are reflexive. The same goes for the transitive and anti-symmetric properties. L_3 defines the meet and join between every group of elements in the lattice (unique infimum and supremum) and as demonstrated above, is a poset. Therefore, L_3 is also a lattice. The cartesian product can of course join more than two lattices by using the method recursively. □

Remark 2.4. *The cartesian product can also be defined using the lexicographical order between two complete lattices. This is, however, rarely used in program anaylsis because normally we do not expect different dimensions to have higher priority.*

Definition 5. *Let $(L_1, \sqsubseteq_1)$ be a complete lattice. Let V be a finite set. The set of all fintite sets $V \to L_1$ is the set of all **finite map (functions)**, between function V and L_1. We define the **finite map** $L = (V \to L_1, \sqsubseteq)$ where $f_1 \sqsubseteq f_2$ if for every element $v \in V$ we have $f_1(v) \sqsubseteq f_2(v)$.*

Claim 2.5. *Let $(L_1, \sqsubseteq_1)$ is a complete lattice, then finite map $L = (V \to L_1, \sqsubseteq)$ is a complete lattice.*

Proof. The partial order of the mapped lattice is defined using the partial order of L1.
Let m be the mapping function. $\forall x_1, x_2 \in V$ $x_1 \sqsubseteq x_2$ if $m(x_1) \sqsubseteq_1 m(x_2)$
The order above has the reflexive property since m will map x_1 to itself and $\sqsubseteq_1$ is reflexive. It has the transitive and antisymmetric properties as well from the poset of L1. The meet and join operators are the values that are mapped to the supremum and infimum of the mapped elements in L_1. $x_1 \sqcap x_2 = x_3$ s.t. m(x_3) $= m(x_1) \sqcap m(x_2)$ $x_1 \bigsqcup x_2 = x_3$ s.t. m(x_3) $= m(x_1) \bigsqcup m(x_2)$
□

Definition 6. *Let $(L, \sqsubseteq)$ be a poset. A **ascending chain** is a sequence of elements $l_1, l_2, l_3 \ldots \in L$ such that $l_1 \sqsubseteq l_2 \sqsubseteq l_3, \ldots$. A **strictly ascending chain** is a sequence of elements $l_1, l_2, l_3 \ldots \in L$ such that $l_1 \sqsubset l_2 \sqsubset l_3, \ldots$. A **descending chain** is a sequence of elements $l_1, l_2, l_3 \ldots \in L$ such that $l_1 \sqsupseteq l_2 \sqsupseteq l_3, \ldots$. A **strictly descending chain** is a sequence of elements $l_1, l_2, l_3 \ldots \in L$ such that $l_1 \sqsupset l_2 \sqsupset l_3, \ldots$.*

For example, if $S = \{a, b\}$, then $\emptyset \sqsubset \{a\} \sqsubset \{a, b\}$ is a strictly ascending chain of $(P(S), \subset)$.

Definition 7. *Let (L, $\sqsubseteq$) be a poset. L has a **finite height** if there is no infinite (strictly) descending or ascending chain.*

3 Tarski's fixed point theorem

Definition 8. *Let $(L, \sqsubseteq)$ be a poset. A function f is called **monotone** if for every $l_1 \sqsubset l_2$ we have $f(l_1) \sqsubset f(l_2)$.*

In program analysis, a monotone function can represent an operation done in the program. For example, if a variable x is in an interval $[a, a+1]$, and an operation $x \leftarrow x+1$ is done but at most t loops, the interval of x can be changes $[a, a+t]$.

Remark 3.1. *The monotone function not necessary imply that for every l we have $l \sqsubset f(l)$. For example, over the complete lattice $(Z \cup \{\infty, -\infty\}, \leq)$, the function $f(x) = x - 1$ is monotone function, but of course $x > f(x)$ for $x \in Z$.*

Remark 3.2. *In some places, a monotone function is called also an **order-preserving function**.*

In this section prove one the fundamental theorems used in program analysis to find a fix points in a complete lattice $(L, \sqsubseteq)$ over a monotone function f. To do so, we use the following definitions

Definition 9. *Let f be a monotone function over a complete lattice $(L, \sqsubseteq)$, and let $l \in L$. Then we define the following definitions :*

1. *If $f(l) \sqsubset l$ then l is called **reductive**. We denote $Red(f) = \{l \in L | f(l) \sqsubseteq l\}$ the **Reductive set**.*
2. *If $l \sqsubset f(l)$ then l is called **extensive**. We denote $Ext(f) = \{l \in L | l \sqsubseteq f(l)\}$ the **Extensive set**.*
3. *If $f(l) = l$ then l is called **fixed point**. We denote $Fix(f) = \{l \in L | f(l) = l\}$ the **Fixed point set**.*
4. *The **least fixed point**, denoted by $lfp(f)$, is the meet of the fixed point set ($lfp(f) = \sqcap Fix(f)$).*
5. *The **greatest fixed point**, denoted by $gfp(f)$, is the join of the fixed point set ($gfp(f) = \sqcup Fix(f)$).*

We depict in Fig. 2 the relation between the different sets. For example, the Fixed point set $Fix(f)$ is the intersection between the Reductive and Extensive sets ($Fix(f) = Ext(f) \cap Red(f)$. We have $\bot \sqsubseteq f(\bot)$ as the bottom $\bot$ is smaller than every element in L. Thus, due to the monotonicity of f, we have $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot)$. In a similar way we can prove that $\top \sqsupseteq f(\top) \sqsupseteq f^2(\top)$.

As the figure depicts, $gfs(f)$ and $lfp(f)$ are respectively the meet and join of $Red(f)$ and $Ext(f)$ sets. This is Tarski's theorem stated below.

Theorem 3.3 (Tarski's theorem)**.** *Let $(L, \sqsubseteq)$ be a complete lattice, and f a monotone function. Then the following hold:*

1. *The respectively meet and join of $Red(f)$ and $Ext(f)$ are Fixed points. Moreover, $Fix(f)$ is not empty.*
2. *The greatest and the least fixed points ($gfs(f)$ and $lfp(f)$) are respectively the meet and join of $Red(f)$ and $Ext(f)$ sets.*

Proof. We will prove that $lfp(f) = \sqcap Red(f) \in Fix(f)$. The proof that $gfp(f) = \sqcap Ext(f) \in Fix(f)$ is similar. We first show the first part of the theorem, i.e, $\sqcap Red(f) \in Fix(f)$.

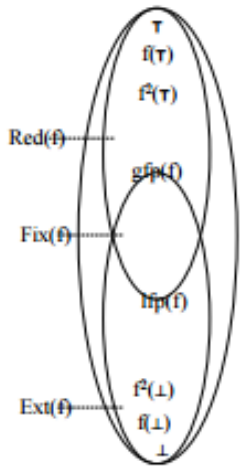

Figure 2: This figure shows the relation between different sets.

1. Denote $\sqcap Red(f) = x_0$. Let z be an element of $Red(f)$. Then according to the meet definition $x_0 \sqsubseteq z$. By monotonicity of f we imply $f(x_0) \sqsubseteq f(z)$. The element z is a reductive point - therefore $f(x_0) \sqsubseteq f(z) \sqsubseteq z$.

We show that for every reductive point z the element $f(x_0)$ is smaller than z. Thus, $f(x_0)$ is a lower bound of $Red(f)$, and by the definition of a meet- $f(x_0) \sqsubseteq x_0$. That means x_0 is a reductive point.

Using monotonicity of f we imply that $f^2(x_0) \sqsubseteq f(x_0)$ and thus $f(x_0)$ is a reductive element. The element x_0 is the meet of Reductive set, and thus $x_0 \sqsubseteq f(x_0)$, i.e., x_0 is an exclusive point. Since x_0 is an exclusive and a reductive element- then it must be a fixed point, i.e, $x_0 \in Fix(f)$.

2. The element $x_0 = \sqcap Red(f)$ is a lower bound of $Red(f)$ and thus a lower bound of $Fix(f)$. The element $x_1 = \sqcap Red(f)$ is larger than any lower bound, and thus $x_0 \sqsubseteq x_1$. The element x_0 is a fixed point, and x_1 is a lower bound of $Fix(f)$- thus $x_1 \sqsubseteq x_0$. By anti-symmetric we imply that $x_0 = x_1$, as required. □

3.1 Application of the theorem

Consider program with a single variable x and suppose our program computes $f(x)$ for an unknown number of iterations. Our goal is to find a group of states l which x can receive, regardless of the number of iterations used actually in the program. That means, our goal is to find a fixed point, $l = f(l)$. As we want the group of states to precise (minimal) as possible, we need to compute the least fixed point $lfp(f)$ (which is a fixed point according to Tarski's theorem).

One algorithm to find the $lfp(f)$ is shown as followed:

Algorithm 1 Compute the least fixed point with single variable

1: Set $l \leftarrow \bot$.
2: **repeat**
3: $\quad l \leftarrow f(l)$
4: **until** l=f(l)
5: **return** the state l.

The bottom is smaller than every element and in particular i.e, $\bot \sqsubseteq f(\bot)$. Using monotonicity of f, we observe that $\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\sqsubseteq) \ldots$. If the lattice $(L, \sqsubseteq)$ has finite height- the algorithm convergence to a fixed point i.e to a point $f^n(l) = z_0$ where $z_0 = f(z_0)$.

If $t \in Fix(f)$, then $\bot \sqsubseteq t$ and therefore $f(\bot) \sqsubseteq f(t) = t$. Repeating the process implies that $f^n(\bot) = z_0 \sqsubseteq t$, i.e., z_0 is a lower bound. The least fix point is larger than every lower bound, i.e. $z_0 \sqsubseteq lfp(f)$. But $lfp(f)$ is smaller then every element in $Fix(f)$, in particular $z_0 \in Fix(f)$. Thus, $lfp(f) = z_0$- as required.

4 Chaotic Iterations

For a lattice, that may or may not be finite, we will assume that there is a limited number of strictly increasing chains. L^n = LxLxL...xL (cartesian product of L for n times) Let f: $L^n \rightarrow L^n$ be a monotone function that handles the dataflow as the program is executed or analyzed.

Remark 4.1. *f is monotone because it is either ι or the join operation. The join operation may only add optional paths. This is why it is inherently a monotone operation.*

We can compute the least fixed point of a function $f(\hat{x}) = (f_1(\hat{x}), f_2(\hat{x}), \ldots f_n(\hat{x})))$, where $\hat{x} = (x_1, x_2, \ldots x_n)$ represents multiple variables of a program. This will imply a precise result as possible for the program (maximum number of $\bot$-s and and maximum number of constants).

Algorithm 2 Compute the least fixed point of function f with multiple variables

1: **for** $i = 1$ to n **do**
2: $\quad x_i \leftarrow \bot$
3: **end for**
4: $WL \leftarrow \{1, 2, \ldots, n\}$
5: **while** $WL \neq \emptyset$ **do**
6: $\quad$ Select i from WL.
7: $\quad$ Remove i from WL.
8: $\quad new \leftarrow f_i(x_1, x_2, \ldots, x_n)$
9: $\quad$ **if** $new \neq x[i]$ **then**
10: $\quad\quad x[i] = new$
11: $\quad\quad$ Add to WL every index j where $f_j(\hat{x})$ was changed (including i)
12: $\quad$ **end if**
13: **end while**
14: **return** the least fixed point $\hat{x} = (x_1, x_2, \ldots, x_n)$.

Note that for this algorithm it is equivalent to use a generalized version of the Algorithm 1 where in every iteration we apply $\hat{x} = f(\hat{x})$.

Complexity:
O(n · h · c · k)

- k, the maximum out degree, is usually 1 or 2.
- h is the height
- n is the number of nodes
- c is the maximum cost of applying the edge operation, comparisons,and the join operation

Definition 10. *An additive function is a function that allows us to extract the join operation. If f is additive, then f(⊔X) = $\bigsqcup_{z \in X} f$ (z)*

Remark 4.2. *If f is additive, then f is monotone but the opposite not always follows. For example, we can define a function over the Even-Odd lattice, such that $f(X) = \{Even\}$ if $|X| = 1, 0$ and otherwise $f(X) = \{Even, Odd\}$. Then $f(\sqcup X) = \{Even, Odd\}$ for $|X| = 2$, but $\bigsqcup_{z \in X} f(z) = \{Even\}$.*

Proof. x ⊑ y ⇒ x ⊔ y = y
f is additive so we get:
f(y) = f(x ⊔ y) = f(x) ⊔ f(y)
f(x) ⊑ f(y)
Therefore, f is also monotone. □

4.1 Dataflow analysis

Programs are translated into graphs. Each node represents an elementary block and the edges connect between elementary blocks based on the possible paths depending on the flow control. The analysis can be approached based on constraints or equations extracted from the program.
The equations can be classified by relating the exit information to the entry information of a node that define sets in terms of each other. These equations can be seen as a function F such that {DF} = F({DF})

In figure 3, we can extract the following flow equations:

1. $df_2 = f_1(df_1) \sqcup f_2(df_1)$
2. $df_1 = \bot$
3. $df_3 = f_3(df_2)$

Or more precisely:

1. $df_{2entry} = f_1(df_{1exit}) \sqcup f_2(df_{1exit})$
2. $df_1 = \bot$
3. $df_{3entry} = f_3(df_{2exit})$

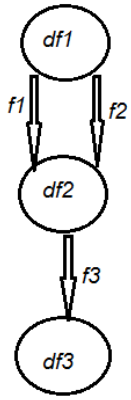

Figure 3: This diagram displays a graph that depicts the flow of a program.

4.2 JOP: Join over all paths

JOP[v] is the result of the join operation on the set of paths from the initial program state to state v. The set of paths is potentially infinite, therefore JOP is undecidable for some domains. This is the reason this method isn't always chosen although it is precise.

Claim 4.3. *JOP[v]* $\sqsubseteq DF_{entry}$ *(v)*
JOP is always more precise than chaotic iteration.

Reminder: DF_{entry}[v] =

Proof. Proof by induction on path length.
Induction basis - zero length path- f[v](ι) = ι
Assume claim holds for path of length n and that the path to node v is of length n+1.
f[$v_1, v_2, ...v_n, v$](ι) = f_{v_n}(f[$v_1, v_2, ...v_n$](ι)) By assumption: f[$v_1, v_2, ...v_n$](ι) $\sqsubseteq$ $DF_{entry}(v_n)$
f_{v_n}(f[$v_1, v_2, ...v_n$](ι)) $\sqsubseteq$ $f_{v_n}(DF_{entry}(v_n))$ From the definition of DF_{entry} and that $DF_{exit}(v_n) \sqsubseteq DF_{entry}(v)$ f_{v_n}(f[$v_1, v_2, ...v_n$](ι)) $\sqsubseteq DF_{entry}(v)$
Since every f_v is additive we get: JOP[v] $\sqsubseteq DF_{entry}$ (v) □

4.3 Widening and Narrowing operators

The widening operator, $\bigtriangledown$, is used to accelerate the termination of chaotic iterations such as in the case of loops. It allows the analysis with lattices of infinite heights (infinite domain).

Definition 11. $l_1 \sqcup l_2 \sqsubseteq l_1 \bigtriangledown l_2$
$y_0 = l_0$ $y_{i+1} = y_i \bigtriangledown l_{i+1}$

Claim 4.4. *We will reach Red(f).*

Proof. $X_n \bigtriangledown f(x_n) \sqsubseteq X_n \sqcup f(x_n) = x_n$
Since X_n is the fixed point:
$x_n \sqsupseteq f(x_n)$ □

The widening operator for intervals increases the boundaries to $\pm\infty$ when they differ.
For example: [0,1] $\bigtriangledown$ [0,2] = [0, ∞] Let us also point out that in this case, the operator is not monotone.
For the same interval [0,2] the following is true: [0,2] $\bigtriangledown$ [0,2] = [0, 2] [0,1] is more precise than [0,2] but the result in the first case is less precise than that of the second one.

The narrowing operator, Δ, fixes the infinite bounds and replaces them with a constant. We will also remain in Red(f).

References

[1] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer, 1999.

[2] C. Urban. *Static Analysis by Abstract Interpretation of Functional Temporal Properties of Programs.* PhD thesis, Department of Computer Science, ETH Zurich, July 2015.