Programm Analysis

Winter semester, 2012

Chaotic Iterations

Lecturer: Mooly Sagiv *Scribe: Anna Itin*

3.1 Posets

Poset is a set with binary partial ordering relation.

$$\sqsubseteq : \mathrm{L} \times \mathrm{L}$$

Also denoted by (L,$\sqsubseteq$).

3.1.1 We demand 3 features:

1. Reflexive: each elemnt is smaller then itself.

$$\forall l \in L : l \sqsubseteq l$$

2. Transitive:

$$\forall l_1, l_2, l_3 \in L : l_1 \sqsubseteq l_2, l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$$

3.Anti-Symmetric:

$$\forall l_1, l_2 \in L : l_1 \sqsubseteq l_2, l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2$$

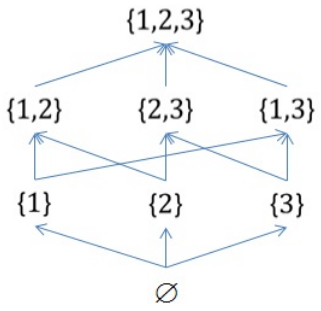
This feature prevents circles.

For many years there was an argument between USA and Europe about the interpretation of the partial ordering relation. Finally USA accepted the Europian version:the expression $l_1 \sqsubseteq l_2$ means that l_1 is smaller then l_2. We say that l_1 is more precise then l_2. Or in other words l_1 represents less concrete states. And yet some people in the USA define it the opposite way.

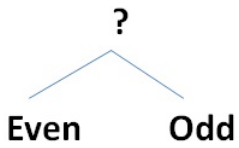
3.1.2 Examples of Posets:

1. Total order. (N,$\leq$)

2. Powersets: sub-groups. For example the powerset of the group S=$\{1,2,3\}$ is P(S)=$\{\emptyset,\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$. The partial ordering relation for this example is the inclusion relation for groups: $\subseteq$.



3. Even/Odd: L = Even,Odd,?. The partial order is: $Even \sqsubseteq ?$, $Odd \sqsubseteq ?$.



3.1.3 More notations

We can use these notations:
(L,$\sqsupseteq$): $l_1, l_2 \in L, l_1 \sqsupseteq l_2 \Leftrightarrow l_2 \sqsubseteq l_1$
(L,$\sqsubset$): $l_1, l_2 \in L, l_1 \sqsubset l_2 \Leftrightarrow l_2 \sqsubseteq l_1 \wedge l_1 \neq l_2$
(L,$\sqsupset$): $l_1, l_2 \in L, l_1 \sqsupset l_2 \Leftrightarrow l_2 \sqsubset l_1$

3.1.4 Upper and Lower Bounds

Lower Bound : A subset L'$\subseteq$L has a lower bound l$\in$L if $\forall l' \in L' : l \sqsubseteq l'$.
Upper Bound : A subset L'$\subseteq$L has a upper bound u$\in$L if $\forall l' \in L' : l' \sqsubseteq u$.
Greatest Lower Bound of a subset $L' \subseteq L$ is a lower bound $l_0 \in$L such that $\forall l \in L$, l is lower bound: $l \sqsubseteq l_0$.

Lowest Upper Bound of a subset $L' \subseteq L$ is an upper bound $u_0 \in$L such that $\forall u \in L'$, u upper bound: $u_0 \sqsubseteq u$.

Example with powersets

In the previous example of the power sets, the lower bound of {{1,2},{2,3}} is {2} and $\oslash$ and the greatest lower bound of is {2}.

Lemma:

Let L be a poset and $L' \subseteq L$.

1. If the greatest lower bound of L' exists, then it is unique. We denote it by $\sqcap L'$, and we call it *meet*.
2. If the lowest upper bound of L' exists, then it is unique. We denote it by $\sqcup L'$, and we call it *join*.

Proof.

1. Lets assume that L is a poset and $L' \subseteq L$. In contradiction lets assume that there exists two greatest lower bounds $l, l' \in L'$. Because they both are greatest lower bounds we get that: $l \subseteq l'$ and $l' \subseteq l$. Because of the Anti-Symmetry we get that $l = l'$.
2. similar to 1.

□

3.2 Complete Lattice

A poset $(L, \sqsubseteq)$ is a complete lattice if every subset of it has least upper bound as well as greatest lower bound. (Including the subset of an empty group.) We denote it with the following tuple : $(L, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$, where:

$$\bot = \sqcup\emptyset = \sqcap L$$

$$\top = \sqcap\emptyset = \sqcup L$$

The least element , $\bot$ is the greatest lower bound of L. Thus, $\forall l \in L, \bot \sqsubseteq l$. The same goes for the $\top$ element.

3.2.1 Examples

- **Powerset:** The previous example is a complete lattice , because : $\top = \{1, 2, 3\}, \bot = \{\emptyset\}$
- **Even-Odd:**This is not a lattice. We don't have the $\bot$ element.
- **Constant propogation:** This is also not a complete lattice because we don't have the $\bot$ element.

3.2.2 Lemma:

For every poset $(L, \sqsubseteq)$ the following conditions are equivalent:

- L is a complete lattice.
- Every subset of L has a least upper bound.
- Every subset of L has a greatest lower bound.

Proof. We will show an equivalence between the second and third condition. By definition of complete lattice we receive the first condition. We will show how to define the greatest lower bound from a least upper bound. The opposite is similar. Let assume that the subset $X \subseteq L$ has a least upper bound, $\sqcup$. We would like to show that it also has a greatest lower bound, $\sqcap$. Lets look at the following subset: $S = \{z \in L | \forall t \in X, z \sqsubseteq t\}$. As we can see S is the set of all lower bounds of X. Because S is also a subset of L , the second condition also hold for it. Thus $s = \sqcup S$ is the least upper bound of S. We will show that s is the greatest lower bound of X. Thus it must hold that:

- s is a lower bound of X: Since S is the set of all the lower bound of X. This means that every element in X is an upper bound of S. But s is the least upper bound of S , thus s is the smallest upper bound of S. Thus s must be smaller then every element in X. Thus s is a lower bound of X.
- s is bigger then every other lower bound of X: As we said before the subset S is a subset of all lower bounds of X. But s is an upper bound of S thus it is bigger then every element in S. Thus it is bigger then every other lower bound of X.

We showed that s is the greatest lower bound of X: $\sqcap X = s = \sqcup S = \sqcup\{z \in L | \forall t \in X, z \sqsubseteq t\}$.

□

3.2.3 Constructs of complete lattices

Cartesian Products

Lets assume we have two complete lattices:

1. $(L_1, \sqsubseteq_1) = (L_1, \sqcap_1, \sqcup_1, \top_1, \bot_1)$
2. $(L_2, \sqsubseteq_2) = (L_2, \sqcap_2, \sqcup_2, \top_2, \bot_2)$

Lets define the set $L = (L_1 \times L_2)$, and the following order $\sqsubseteq$: $x_1, y_1 \in L_1, x_2, y_2 \in L_2, x_1 \sqsubseteq y_1, x_2 \sqsubseteq y_2 \Rightarrow (x_1, x_2) \sqsubseteq (y_1, y_2)$. We can show that $(L, \sqsubseteq)$ is a Poset.

Proof.

- Reflexivity: $(L_1, \sqsubseteq_1), (L_2, \sqsubseteq_2)$ are lattices. Thus both reflexive. Thus $x \sqsubseteq_L x$ *and* $y \sqsubseteq_L y$. Thus $(x, y) \sqsubseteq_L (x, y)$
- Transitivity: Assume that $(x_1, x_2) \sqsubseteq (y_1, y_2)$ *and* $(y_1, y_2) \sqsubseteq (z_1, z_2)$. This means that:

  $x_1 \sqsubseteq y_1$ *and* $y_1 \sqsubseteq z_1$. Because of the transitivity of $L_1 : x_1 \sqsubseteq z_1$.
  $x_2 \sqsubseteq y_2$ *and* $y_2 \sqsubseteq z_2$. Because of the transitivity of $L_2 : x_2 \sqsubseteq z_2$.

  Thus , by definition : $(x_1, z_1) \sqsubseteq (x_2, z_2)$.
- Antisymmetry: Assume that $(x_1, y_1) \sqsubseteq_L (x_2, y_2)$ and $(x_2, y_2) \sqsubseteq_L (x_1, y_1)$. This means that:

  $x_1 \sqsubseteq_1 x_2$ *and* $x_2 \sqsubseteq_1 x_1$. Thus by the antisymmetry of L_1 we get that $x_1 = x_2$. $y_1 \sqsubseteq_2 y_2$ *and* $y_2 \sqsubseteq_2 y_1$. Thus by the antisymmetry of L_2 we get that $y_1 = y_2$.

  Thus $(x_1, y_1) = (x_2, y_2)$.

□

Now we will show that if L_1, L_2 are complete lattices, then $(L, \sqsubseteq)$ is a complete lattice. We will show that every subset of L has a least upper bound. Lets assume that $X \subseteq L$. We will define the following subsets:
$X_1 = \{l_1 \in L_1 | \exists l_2 \in L_2 s.t. (l_1, l_2) \in X\}$
$X_2 = \{l_2 \in L_2 | \exists l_1 \in L_1 s.t. (l_1, l_2) \in X\}$
Both of them are subsets of a complete lattice L_1, L_2 accordingly. Thus both of the have a least upper bound. We will denote it by x_1, x_2 accordingly. We will show that $x = (x_1, x_2)$ is the least upper bound of X.
First of all we will show that it is an upper bound. Lets assume $(x'_1, x'_2) \in X$. Thus $x'_1 \in X_1$. Since X_1 is a subset of a complete lattice L_1 , $x'_1 \sqsubseteq x_1$. In the same way $x'_2 \sqsubseteq x_2$. By definition : $(x'_1, x'_2) \sqsubseteq (x_1, x_2) = x$.
Now we will show that x is the least upper bound. Lets assume that $y = (y_1, y_2) \in X$ is an upper bound. Then $\forall z = (l_1, l_2) \in X : z \sqsubseteq y$. Thus $l_1 \sqsubseteq y_1$, $l_2 \sqsubseteq y_2$. Thus y_1 is a least upper bound of X_1. Thus $x_1 \sqsubseteq y_1$. The same way $x_2 \sqsubseteq y_2$. By definition $x = (x_1, x_2) \sqsubseteq (y_1, y_2) = y$. Thus x is the **least** upper bound of X.

Finite Maps

Lets assume we have a complete lattice $(L_1, \sqsubseteq_1) = (L_1, \sqcap_1, \sqcup_1, \top_1, \bot_1)$ and a finite set V. Lets define the map $L = V \to L_1$, and the following order $\sqsubseteq$: $e_1, e_2 \in L, \forall v \in V, e_1 v \sqsubseteq e_2 v \Rightarrow e_1 \sqsubseteq e_2$. We will show that L is a poset and a complete lattice.

Proof.

- Poset:
  - Reflexivity: Lets assume that $e \in L$. By definition : $\forall v \exists x \in L_1 : ev = x$. Also since L_1 is a lattice it is also reflexive thus $x \sqsubseteq_1 x$. Thus $ev \sqsubseteq_1 ev$. By definition $e \sqsubseteq e$.
  - Transitivity: Lets assume that $e_1, e_2, e_3 \in L$ and $e_1 \sqsubseteq e_2, e_2 \sqsubseteq e_3$. Thus $\forall v \in V \exists x_1, x_2, x_3 \in L_1 : e_1 v = x_1, e_2 v = x_2, e_3 v = x_3$. Thus $x_1 \sqsubseteq_1 x_2, x_2 \sqsubseteq_1 x_3$. Because L_1 is a poset we get that: $x_1 \sqsubseteq_1 x_3$. Thus by definition $e_1 v \sqsubseteq e_3 v$. Thus $e_1 \sqsubseteq e_3$.
  - Antisymmetry: Lets assume that $e_1 \sqsubseteq e_2$ and $e_2 \sqsubseteq e_1$. By definition : $\forall v \in V \exists x_1, x_2 \in L_1 : e_1 v = x_1, e_2 v = x_2$. Also by definition : $e_1 v \sqsubseteq_1 e_2 v, e_2 v \sqsubseteq_1 e_1 v$. Thus $x_1 \sqsubseteq_1 x_2, x_2 \sqsubseteq_1 x_1$. Since L_1 is a poset : $x_1 = x_2$. Thus $e_1 v = e_2 v$, thus $e_1 = e_2$.

- Complete lattice:
  We will prove that every subset of L has a least upper bound. Lets assume the $X \subseteq L$. We will define the following subset: $\forall v \in V, X_v = \{l \in L_1 | \exists e_v \in X : e_v v = l\}$. Since $X_v \subseteq L_1$ and L_1 is a complete lattice , there exist a least upper bound of X_v ; x_v. We will prove that $e := (\forall v \in V : e_v v)$ is the least upper bound of L. First of all we will show that it is an upper bound. Lets assume $e' \in X$. Since $\forall v \in V, x_v$ is an upper bound we get that $e'v \sqsubseteq ev = x_v$. By definition we get that $e' \sqsubseteq e$.
  Now we will show that $e := (\forall v \in V : e_v v)$ is the least upper bound. Lets assume that there exists another upper bound e' of X. Then $\forall v \in V, e'' \in X : e''v \sqsubseteq_1 ev$. Thus $e'v = x'_v$ is an upper bound of X_v. Thus $ev = x_v \sqsubseteq_1 x'_v = e'v$. Thus x_v is the **least** upper bound.

□

The bottom of this lattice is the function that maps every variable to bottom. The same goes for top.

3.2.4 Chains

We say that a subset $Y \subseteq L$ in a poset $(L, \sqsubseteq)$ is a *chain* if every two elements in Y are ordered: $\forall l_1, l_2 \in Y : l_1 \sqsubseteq l_2$ *or* $l_2 \sqsubseteq l_1$. We can have the following types of chains:

- ascending chain : $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq l_4 \sqsubseteq l_5 \sqsubseteq ...$
- strictly ascending chain : $l_1 \sqsubset l_2 \sqsubset l_3 \sqsubset l_4 \sqsubset l_5 \sqsubset ...$
- descending chain : $l_1 \sqsupseteq l_2 \sqsupseteq l_3 \sqsupseteq l_4 \sqsupseteq l_5 \sqsupseteq ...$
- strictly descending chain : $l_1 \sqsupset l_2 \sqsupset l_3 \sqsupset l_4 \sqsupset l_5 \sqsupset ...$

The poset L has a *finite height* if and only if every chain in L is finite. For example in constant propagation the lattice is infinite but the maximal height of the lattice is 3 (for one variable). For n variable it will be $O(n)$.
We say that a chain stabilises if and only if : $\exists n_0 \in N : \forall n \in N : n \geq n_0 \Rightarrow l_n = l_(n_0)$.

Lemma:

If all strictly (descending/ascending) chains of a poset $(L, \sqsubseteq)$ are finite then all the chains of the poset stabilises.

Proof. It is obvious that any finite chain stabilises, thus all the strictly (descending/ascending) chains stabilises. Lets assume by contradiction that a non strictly (ascending/descending) chain Y doesn't stabilise. This means that there is no maximal (or minimal) element in the chain , thus we can extract from it a strictly ascending (or descending) chain which does not stabilise. This is in contradiction to our assumption. □

Lemma:

A poset $(L, \sqsubseteq)$ has a finite height if and only if every strictly decreasing and strictly increasing chains are finite.

Proof.

- If L has a finite height then by definition every chain in L (including the strictly increasing , strictly decreasing) are finite.
- Lets assume that every strictly decreasing and strictly increasing chains are finite. By the previous lemma we can see that every chain of L stabilises. And now by using LemmaA.6 from the book we prove that L has a finite height.

□

3.2.5 Monotone Functions

$(L, \sqsubseteq)$ is a Poset. A function $f : L \to L$ is monotone if for every $l_1, l_2 \in L$: $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$.
Example: $f : x = x + 1$ .This function is monotonic. We have the following options:

$$x = \bot \Rightarrow f(x) = \bot$$
$$x = \top \Rightarrow f(x) = \top$$
$$x = n \Rightarrow f(x) = n + 1$$

3.2.6 Tarski Theorem: Fixed Point

Assume we have a complete lattice : $(L, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$ and a function $f : L \to L$.

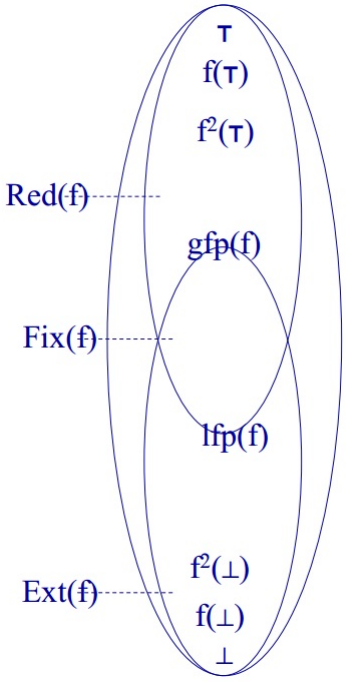
Theorem 3.1 *If f is monotone then:*

$$\begin{array}{c} \textit{least fixed point of } f \textit{ (lfp(f))} = \sqcap Fix(f) = \sqcap Red(f) \in Fix(f) \\ \textit{greatest fixed point of } f \textit{ (gfp(f))} = \sqcup Fix(f) = \sqcup Ext(f) \in Fix(f) \end{array}$$

where :

$$\begin{array}{l} Fix(f) = l \in L, f(l) = l \\ Red(f) = l \in L, f(l) \sqsubseteq l \\ Ext(f) = l \in L, l \sqsubseteq f(l) \end{array}$$

In the figure below we can see a visualization of the theorem. In the figure we can see that $\bot \sqsubseteq f(\bot)$. This is because $\bot$ is less then every value. Also we can see that $f(\bot) \sqsubseteq f(f(\bot))$. This is because f is monotone. The same applies for $\top$.



Proof. We will show the first claim of lfp. The gfp is similar.
Lets define $x_0 = \sqcap Red(f) = \sqcap\{l : f(l) \sqsubseteq l\}$. Since L is a lattice , X_0 is well defined : it exists and is unique. Now we must show that x_0 is the least fixed point. First we will prove that $x_0 \in Fix(f)$. In other words that $x_0 = f(x_0)$. We can show it by proving that $x_0 \sqsubseteq f(x_0)$ and $f(x_0) \sqsubseteq x_0$. Because of the antisymmetry we will get the wanted claim.

- $f(x_0) \sqsubseteq x_0$: By definition $\forall l \in Red(f), x_0 \sqsubseteq l$. Because of the monotony $f(x_0) \sqsubseteq f(l)$. By definition of $Red(f)$ we get that $f(x_0) \sqsubseteq f(l) \sqsubseteq l$. Because of the transitivity we get that $f(x_0) \sqsubseteq l$. This means that $f(x_0)$ is a lower bound of $Red(f)$. Since x_0 is the greatest lower bound of $Red(f)$ we get that $f(x_0) \sqsubseteq x_0$.

- $x_0 \sqsubseteq f(x_0)$: If we apply f on the previous result we will get that $f(f(x_0)) \sqsubseteq f(x_0)$. Thus $f(x_0) \in Red(f)$. And again since x_0 is the greatest lower bound of $Red(f)$ we get that $x_0 \sqsubseteq f(x_0)$.

Thus $x_0 = f(x_0) \Rightarrow x_0 \in Fix(f)$.
Now in order to see that $x_0 = lfp(f)$, notice that because of the reflexivity, $Fix(f) \subseteq Red(f)$. Because of the definition of x_0 we get that $x_0 = \sqcap Fix(f)$ and finally that x_0 is the least fixed point of f. □

3.3 Chaotic Iterations

Now we will show a method for finding the least fixed point of monotone function : $f : L^n \to L^n$, where L is a lattice with finite strictly increasing chains. The lfp(f) is simultaneous: $(x[i] = f_i(x) : 1 \leq i \leq n)$. The idea is applying the function f many times. The pseudo code of the algorithm is shown in the figure below.

$\underline{x} := (\bot, \bot, \ldots, \bot)$

while $(\underline{f}(x) \neq \underline{x})$ do

$\underline{x} := f(\underline{x})$

We start with a vector of bottoms: $x = (\bot, \bot, ..., \bot)$. And We apply the vector function f each time, until we get the fixed point. The disadvantage of this algorithm is that each iteration is very expensive since we must apply

the all vector function.
The following algorithm is normally cheaper:

```
for i :=1  to n do
   x[i] = ⊥
WL = {1, 2, …, n}
while (WL ≠  ∅ )  do
  select and remove an element i ∈ WL
  new := f_i(x)
  if (new ≠ x[i]) then
      x[i] := new;
      Add all the indexes that directly depends on i to WL
```

The initialization is the same as before : vector of $\perp$. In this algorithm each iteration works only on one index from the vector according to the Work List(WL). In the end of the iteration it updates the WL accordingly to the value of f for the chosen index.

3.3.1 Specialized Chaotic Iterations

This is another algorithm for finding a lfp(f)(CFG algorithm). It works for an imperative programs . It uses the control graph of the program and iterates over it.

```
Chaotic(G(V, E): Graph, s: Node, L: Lattice, ι: L, f: E →(L →L) )
  for each v in V to n do df_entry[v] := ⊥
  df[s] = ι
  WL = {s}
  while (WL ≠  ∅ )  do
    select and remove an element u ∈ WL
    for each v, such that. (u, v) ∈E do
          temp = f(e)(df_entry[u])
          new := df_entry(v)⊔ temp
          if (new ≠ df_entry[v]) then
                df_entry[v] := new;
                WL := WL ∪{v}
```

We start in some node s , and we assume that there is no entering edges to s. The initial value is ι. DF is a data flow.

Complexity of Chaotic Iterations

In order to compute the complexity of the algorithm we have the following parameters:

- n - the number of CFG nodes.
- k - the maximum out-degree of edges. Most of the time it is 2 (for example: **if then else**. In programmes with **switch** it is more complex.
- h - the height of the lattice.
- c - the cost for applying f , $\sqcup$ and L comparisons.

The complexity is: $O(n * k * h * c)$.

System of Equations

The algorithm computes a minimal solution to the system of equations. This is for the iterative programmes:

$$\begin{cases} df_{entry}[s] = \iota \\ df_{entry}[v] = \sqcup\{f(u,v)(df_{entry}[u]) | (u,v) \in E\} \end{cases}$$

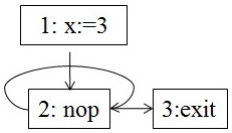
The following one is for the regular chaotic iterations with the vectorial form:

$$F_S : L^n \to L^n = \begin{cases} F_S(X)[s] = \tau \\ F_S(X)[v] = \bigsqcup\{f(u,v)(X[u]) \mid (u,v) \in E\} \end{cases}$$

The least fixed point in both of them is the same: $lfp(S) = lfp(F_S)$.

Example

Assume we have the following graph:



The system of equations is the following:
$DF(1) = [x \to 0]$
$DF(2) = CP(1)[x \to 3] \sqcup DF(2)$
$DF(3) = DF(2)$
Here we can check 4 solutions to the system of equations:

DF[1]	DF[2]	DF[3]	index
[x⟶0]	[x⟶ 3]	[x⟶ 3]	1
[x⟶0]	[x⟶?]	[x⟶?]	2
[x⟶7]	[x⟶ 9]	[x⟶ 7]	3
[x⟶?]	[x⟶ 3]	[x⟶ 3]	4

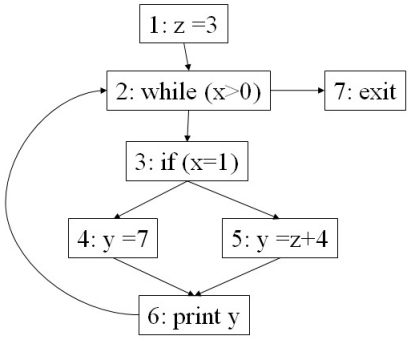
The first solution is a least fixed point. It is easy to see that it is a fixed point, but a bit harder to understand that it is the least one. The second solution is a fixed point but not a minimal. The third is not a solution because doesn't solve the equation. The forth solution is legal but it is not a fixed point.

3.3.2 Another Example

assume we have the following code:

$z \leftarrow 3$
while x > 0 **do**
  if x=1 **then**
    $y \leftarrow 7$
  else
    $y \leftarrow z + 4$
  end if
  print y
end while

This is the control flow graph for our example:



Summary of the states in execution of CFG algorithm on the example is shown in the following figure (We will chose node from WL in DFS order):

node	DF	WL
init	$x \mapsto ?, y \mapsto ?, z \mapsto ?$	{1,2,3,4,5,6,7}
1	$x \mapsto ?, y \mapsto ?, z \mapsto 3$	{2,3,4,5,6,7}
2	$x \mapsto ?, y \mapsto ?, z \mapsto 3$	{3,4,5,6,7}
3	$x \mapsto ?, y \mapsto ?, z \mapsto 3$	{4,5,6,7}
4	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	{5,6,7}
6	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	{2,5,7}
2	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	{3,5,7}
3	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	{5,7}
5	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	{7}
7	$x \mapsto ?, y \mapsto 7, z \mapsto 3$	$\emptyset$

3.3.3 Soundness

The described algorithm is sound. Every computed value is the real value of the algorithm‘s running. All the errors will be detected. The proof will be shown in the next lessons.

3.3.4 Completeness

The algorithm is not complete. Some values may be not detected. It is possible that we will detect errors which wont be found in runtime of the program. The main reason for uncompletness is the fact that we use unknown value ? and we don’t really execute the program , rather use a kind of abstraction.

3.4 Widening

The purpose is to accelerate the termination of the Chaotic iterations by computing a more conservative solution. It can handle latices of infinite heights. The main idea is in stead of using the $join : \sqcup$ operation we will use the widening operation: $\bigtriangledown$. The new CFG algorithm is in the following figure:

```
Chaotic(G(V, E): Graph, s: Node, L: lattice, ι: L, f: E →(L →L) )
  for each v in V to n do df_entry[v] := ⊥
  In[v] = ι
  WL = {s}
  while (WL ≠ ∅ ) do
    select and remove an element u ∈ WL
    for each v, such that. (u, v) ∈E do
          temp = f(e)(df_entry[u])
          new := df_entry(v) ▽ temp
          if (new ≠ df_entry[v]) then
                df_entry[v] := new;
                WL := WL ∪{v}
```

3.4.1 Example of Interval Analysis

We will use this example for better understanding of the concept. Many times we would like to find the lower and upper bound of the value of a variable. It is good for checking array exceptions or for counting number of loop iterations. We will define the lattice in the following way:

$$L = (Z \cup \{-\infty, \infty\} \times Z \cup \{-\infty, \infty\}, \sqsubseteq, \sqcup, \sqcap, \top, \bot)$$

The ordering relation is defined in the following way:

$$[a,b] \sqsubseteq [c,d] \; if \; c \leq a \; and \; d \geq b$$

The join is defined the following way:

$$[a,b] \sqcup [c,d] = [min(a,c), max(d,b)]$$

The join looses information when it includes values in the middle of the ranges. For example $[2,5] \sqcup [7,9] = [2,9]$. The meet acts similarly:

$$[a,b] \sqcap [c,d] = [max(a,c), min(d,b)]$$

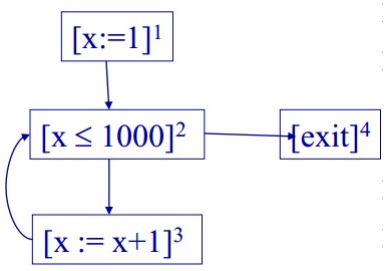
The top is $-\infty, \infty$. The Bottom is every range $[a,b] such that a \geq b$. For example $[7,5]$. But there are many ranges that can represent bottom so this is not lattice. We will unify all these values into one and call it the bottom value.

Example Program

We will use the following code as example:

```
[x := 1]^1 ;
while [x ≤ 1000]^2 do
    [x := x + 1;]^3
```

The graph is the following:



The execution is the following:

1. Entry(1) = $[-\infty, \infty]$.
2. Exit(1) = [1,1]
3. Entry(2) = Exit(1) $\sqcup$ Exit(3) = [1,1] $\sqcup \perp$ = [1,1]

4. Exit(2) = Entry(2)=[1,1]
5. Entry(3) = Exit(2) $\sqcap[-\infty, 1000]$ = [1,1] $\sqcap[-\infty, 1000]$ = [1,1]
6. Exit(3) = Entry(3)+[1,1] = [1,1]+[1,1] = [2,2].
7. Entry(2) = [1,1] $\sqcup$[2,2] = [1,2]
8. Exit(2) = Entry(2)=[1,2]
9. Entry(3) = [1,2] $\sqcap[-\infty, 1000]$ = [1,2]
10. Exit(3) = [1,2]+[1,1] = [2,3]. This will Continue until we get to [2,1001].
11. Exit(2) = Entry(2) = Exit(1) $\sqcup$ Exit(3) = [1,1] $\sqcup$ [2,1001] = [1,1001]
12. Exit(4) = Entry(4) = Exit(2) $\sqcap[1001, \infty]$ = [1,1001] $\sqcap[1001, \infty]$ = [1001,1001]

We saw that our algorithm is dependent on the length of the input. In other examples it is possible to get a non stopping program. We don't want our algorithm to run infinitely. So we use widening. The definition of widening is:

* $\perp \bigtriangledown [c, d] = [c, d]$
* $[a, b] \bigtriangledown [c, d] = [$
  $if\ a \leq c\ then\ a\ else\ -\infty$
  $if\ b \geq d\ then\ b\ else\ \infty]$ n each iteration $if\ a \leq c$ we are getting smaller, but otherwise in order to stop we will give the smallest value that cannot grow. Using this technique we will receive an output which is above the least fixed point.

Now we will execute the algorithm with widening on the previous example. We will traverse the worklist in depth first order. This is the fastest way to converge :

1. WL=$\{1,2,3,4\}$. Entry(1) = $[-\infty, \infty]$.
2. WL=$\{2,3,4\}$. Exit(1) = [1,1]
3. WL=$\{2,3,4\}$. Entry(2) = Exit(2) $\bigtriangledown$ (Exit(1) $\sqcup$ Exit(3)) = $\perp \bigtriangledown ([1, 1] \sqcup \perp) = \perp \bigtriangledown [1, 1] = [1,1]$

4. WL={3,4}. Exit(2) = Entry(2)=[1,1]
5. WL={3,4}. Entry(3) = Exit(2) $\sqcap[-\infty, 1000]$ = [1,1] $\sqcap[-\infty, 1000]$ = [1,1]
6. WL={3,4}. Exit(3) = Entry(3)+[1,1] = [1,1]+[1,1] = [2,2].
7. WL={2,4}. Entry(2) = [1,1] $\bigtriangledown$ ([1,1]$\sqcup$[2,2])=[1,1] $\bigtriangledown$ [1,2] = $[1, \infty]$.
8. WL={3,4}. Exit(2) = $[1, \infty]$.
9. WL={3,4}. Entry(3) = $[1, \infty] \sqcap [-\infty, 1000]$ = [1,1000]
10. WL={2,4}. Exit(3) = [1,1000] + [1,1] = [1,1001]
11. WL={2,4}. Entry(2) = $[1, \infty] \bigtriangledown ([1, 1] \sqcup [2, 1001])$= $[1, \infty] \bigtriangledown [1, 1001]$ = $[1, \infty]$.
12. WL={4}. Exit(2) = $[1, \infty]$.
13. WL={$\emptyset$}. Exit(4) = Entry(4) = Exit(2) $\sqcap[1001, \infty]$ = $[1, \infty]\sqcap[1001, \infty]$ = [1001,∞]

We can see that the solution we got is sound but not precise. Instead of getting [1001,1001] , we received [1001,∞].

3.4.2 Requirements on Widening

We will define the requirements for a widening operator and prove the results of Chaotic Iterations with widening:

1. $\forall l_1, l_2 \in L : l_1 \sqcup l_2 \sqsubseteq l_1 \bigtriangledown l_2$
2. For all ascending chains: $l_1 \sqsubseteq l_2 \sqsubseteq ...$ the following sequence is finite: $y_0 = l_0$, $y_{i+1} = y_i \bigtriangledown l_{i+1}$

For a monotonic function $f : L \to L$ we define the sequence:
$x_0 = \bot$, $x_{i+1} = x_i \bigtriangledown f(x_i)$

Theorem 3.2

- *There exists k such that $x_{k+1} = x_k$. (This means that the sequence is finite.)*

- $x_k \in Red(f)$.

This theorem will prove that chaotic iteration with widening finds a solution in $Red(f)$.

Proof.

- First of all we will prove that the sequence is increasing. $x_0 = \bot \leq x_1$. Also by the definition of join : $x_i \sqsubseteq (x_i \sqcup f(x_i))$. Because of the requirements of widening we get that: $x_i \sqsubseteq (x_i \sqcup f(x_i)) \sqsubseteq x_i \bigtriangledown f(x_i)$. And by definition of the sequence we get that: $x_i \sqsubseteq (x_i \sqcup f(x_i)) \sqsubseteq x_i \bigtriangledown f(x_i) \sqsubseteq x_{i+1}$.

- Because of the requirements the sequence also must be finite. Thus it must converge. This means there exist k such that $x_{k+1} = x_k$.

- Because of the widening requirements : $(x_k \sqcup f(x_k)) \sqsubseteq (x_k \bigtriangledown f(x_k)$. By definition of the sequence : $(x_k \sqcup f(x_k)) \sqsubseteq (x_k \bigtriangledown f(x_k) = x_{k+1}$. Because of the convergence: $(x_k \sqcup f(x_k)) \sqsubseteq (x_k \bigtriangledown f(x_k) = x_{k+1} = x_k$. Thus by definition of join we get that: $f(x_k) \sqsubseteq x_k$. Thus $x_k \in Red(f)$.

□

Non Monotonicity of Widening

From the following example we can see that widening is not monotonic:
$[0, 1] \bigtriangledown [0, 2] = [0, \infty]$
$[0, 2] \bigtriangledown [0, 2] = [0, 2]$
Moreover we can see some non intuitive consequence. In this example when we had a more precise input but we got a less precise output of the widening operation : $[0, \infty]$. And on the other hand when we apply widening operation on a bigger range we receive a more precise result: $[0, 2]$. We also can see this phenomenon in the following example:

```
i ← 0
while true do
    i ← i + 1
    if i=3 then
        i ← 0
    end if
end while
```

The regular run of the widening operation will end up with not so precise result: $[0, \infty]$. But if In the beginning we would start with assumption that i is in the range $[0, 2]$. we would end up with much precise result: $[0, 2]$.

3.5 Narrowing

The purpose is to improve the result of widening. After running widening algorithm we will run the same algorithm but change widening operation with narrowing operation.

3.5.1 Example of Interval Analysis

Back to the previous example we will define the narrowing operation as follows:

* $[a, b] \triangle \perp = [a, b]$
* $[a, b] \triangle [c, d] = [$
  $if\ a = -\infty\ then\ c\ else\ a$
  $if\ b = \infty\ then\ d\ else\ b]$

Now we will execute the algorithm with narrowing on the previous example:

1. WL={1,2,3,4}. Entry(1) = $[-\infty, \infty]$.
2. WL={2,3,4}. Exit(1) = [1,1]
3. WL={2,3,4}. Entry(2) = Exit(2) $\triangle$ (Exit(1) $\sqcup$ Exit(3)) = $[1, \infty] \triangle ([1, 1] \sqcup [1, 1001]) = [1, \infty] \triangle [1, 1001]$ = [1,1001]
4. WL={3,4}. Exit(2) = Entry(2)=[1,1001]
5. WL={3,4}. Entry(3) = Exit(2) $\sqcap[-\infty, 1000]$ = [1,1001] $\sqcap[-\infty, 1000]$ = [1,1000]
6. WL={3,4}. Exit(3) = Entry(3)+[1,1] = [1,1000]+[1,1] = [2,1001].
7. WL={2,4}. Entry(2) = [1,1001] $\triangle$ ([1,1]$\sqcup$[2,1001])=[1,1001] $\triangle$ [1,1001] = [1,1001].
8. WL={4}. Exit(2) = [1,1001].

9. WL=$\{\emptyset\}$. Exit(4) = Entry(4) = Exit(2) $\sqcap[1001, \infty]$ = $[1, 1001] \sqcap [1001, \infty]$ = [1001,1001]

Our result is now precise.

3.5.2 Requirements on Narrowing

We will define the requirements for a narrowing operator and prove the results of Chaotic Iterations with narrowing after widening:

1. $y \sqsubseteq x \Rightarrow y \sqsubseteq (x \triangle y) \sqsubseteq x$
2. For all decreasing chains: $x_0 \sqsupseteq x_1 \sqsupseteq x_2 \sqsupseteq ...$ the following sequence is finite:
   $y_0 = x_0$, $y_{i+1} = y_i \triangle x_{i+1}$

For a monotonic function $f : L \to L$ and $x \in Red(f) = l \in L, f(l) \sqsubseteq l$ we define the sequence:
$y_0 = x$, $y_{i+1} = y_i \triangle f(y_i)$

Theorem 3.3

- *There exists k such that* $y_{k+1} = y_k$.
- $y_k \in Red(f)$.

Proof. We will show using induction that the sequence y_i is decreasing , and that $y_i \in Red(f)$.
Base: $y_0 = x \in Red(f)$.
Induction Step: Lets assume that $y_i \in Red(f)$ this means that $f(y_i) \sqsubseteq y_i$. Using the requirement 1 we get that : $f(y_i) \sqsubseteq f(y_i) \triangle y_i \sqsubseteq y_i$.
Thus be the definition of the given sequence we get that : $f(y_i) \sqsubseteq y_{i+1} \sqsubseteq y_i$. This implies that the sequence is decreasing. Now we apply the function f on the last equation and get the following: $f(yi + 1) \sqsubseteq f(y_i)$. Because of the transitivity we get that $f(y_{i+1}) \sqsubseteq y_{i+1}$. Thus $y_{i+1} \in Red(f)$. Since the sequence is decreasing and using the requirement 2 (the sequence is finite) , the sequence must converge. This means that there exists k such that $y_{k+1} = y_k$. □

3.5.3 Widening and Narrowing Summary

Widening and narrowing are very simple but produces impressive precision. For example the McCarthy 91 function:

```
int f(x) [-inf,inf]
  if x > 100
     then [101,inf] return x -10 [91,inf-10];
     else [-inf,100] return f(f(x+11)) [91,91] ;
```

Executing this function can never stop , but our technique finds that at the end of the function, x indeed has the value of 91. This technique is useful with lattices of finite height to accelerate the Chaotic Iterations process. It also can be used as a methodological tool.

3.6 Summary

Chaotic Iterations is a beautiful technique. It is very easy to implement and rather precise. It is a bit expensive and there are more efficient methods for analysing structured programs.

Bibliography

[1] Flemming Nielson, Hanne R. Nielson, Chris HankinPierce, *Principles of Program Analysis*, Springer, 2010.

[2] Mooly Sagiv, *http://www.cs.tau.ac.il/∼msagiv/courses/pa12-13/chaotic.ppt*, 2012.