

Abstract Interpretation - Part I

Class notes

Or Tamir & Ori Yechieli

Monotone Frameworks with Widening

The Chaotic iteration algorithm:

```
Chaotic( $G(V, E) : \text{graph}, s : \text{Node}, L : \text{lattice}, f : E \rightarrow (L \rightarrow L)$ )
{
  for each  $v \in V$ 
     $v.\text{dfentry} := \perp$ 
     $v.\text{in} := t$ 
   $WL = \{s\}$ 
  while( $WL \neq \emptyset$ )
    pop element from  $WL, u$ 
    for each  $e = (u, v) \in E$ 
       $\text{temp} = f(e)(u.\text{dfentry})$ 
       $\text{new} = v.\text{dfentry} \sqcup \text{temp}$ 
      if ( $\text{new} \neq v.\text{dfentry}$ )
         $v.\text{dfentry} = \text{new}$ 
         $WL.\text{add}(v)$ 
}
```

One problem this algorithm has is being sometimes more acquired than needs that causes a long running time. For example:

Interval Analysis

We would like to determine a range of values for every variable from various reasons (non-negativity in array, store value in a lighter register etc).

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

$$[a, b] \sqsubseteq [c, d] \text{ iff } [a, b] \subseteq [c, d]$$

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

$$[a, b] \sqcap [c, d] = [a, b] \cap [c, d]$$

$$\perp = \phi$$

$$\tau = [-\infty, \infty]$$

In other words joint of ranges is the minimum range that contains the other two ranges.

Here is a sample program:

$$[X = 1]^1$$

$$\text{while } [X \leq 1000]^2$$

$$[X = X + 1]^3$$

$$[\text{skip}]^4$$

If we will run the Chaotic iteration algorithm on that program in any order the number of iterations will be ≈ 1000 , but we will get very accurate result.

Here is another sample program on which the Chaotic iteration algorithm will not terminate (due to the two sets of infinite states):

$$[X = 1]^1$$

$$\text{while } [X > 0]^2$$

$$[X = X + 1]^3$$

$$[\text{skip}]^4$$

We would like now to improve the running time of the chaotic iterations with some loss to the accuracy. We will define a Widening action that will get us fast to least accurate result but very fast.

The widening action will have several properties:

$$\text{Def1: } l_1 \sqcup l_2 \sqsubseteq l_1 \nabla l_2$$

Less conservative than join.

$$\forall \{l_i\} \text{ s.t. } l_i \sqsubseteq l_{i+1}$$

$$\{y_i\} :=$$

$$\text{Def2: } y_0 = l_0$$

$$y_{i+1} = y_i \nabla l_{i+1}$$

$$\exists k \in \mathbb{N} \text{ s.t. } \forall i > k \ y_k = y_i$$

Meaning that for all sequence (even infinite) ∇ will get us in finite number of steps to stop in the chaotic iterations algorithm.

Theorem: $f : L \rightarrow L$ monotonic function, $x_0 = \perp$, $x_{i+1} = x_i \nabla f(x_i)$ therefore

$$\exists k \ x_k = x_{k+1} \wedge x_k \in \text{Red}(f) = \{l \mid l \in L \wedge l \sqsubseteq f(l)\}$$

Proof:

We will take the proposed $\{x_i\}$ and show that they are monotonic:

$$x_i \sqsubseteq x_i \sqcup f(x_i) \sqsubseteq x_i \nabla f(x_i) = x_{i+1}$$

From Def2 we get that there is $k \in N$ s.t. $x_k = x_{k+1}$ now we will see $x_k \in \text{Red}(f)$

$$x_k = x_{k+1} = x_k \nabla f(x_k) \sqsupseteq x_k \sqcup f(x_k) \sqsupseteq f(x_k)$$

Now let's get back to the example:

Widening definition will be:

$$\phi \nabla [a, b] = [a, b]$$

$$[a, b] \nabla [c, d] = [\text{if } c < a - \infty \ a, \text{ if } b < d \ \infty \ b]$$

The definition of widening is not symmetric and there is no problem with that (moreover for good widening it has to be not symmetric). We can notice that if we see any progress in the new range $[c, d]$ (meaning $c < a$ or $b < d$) we strictly go to infinity.

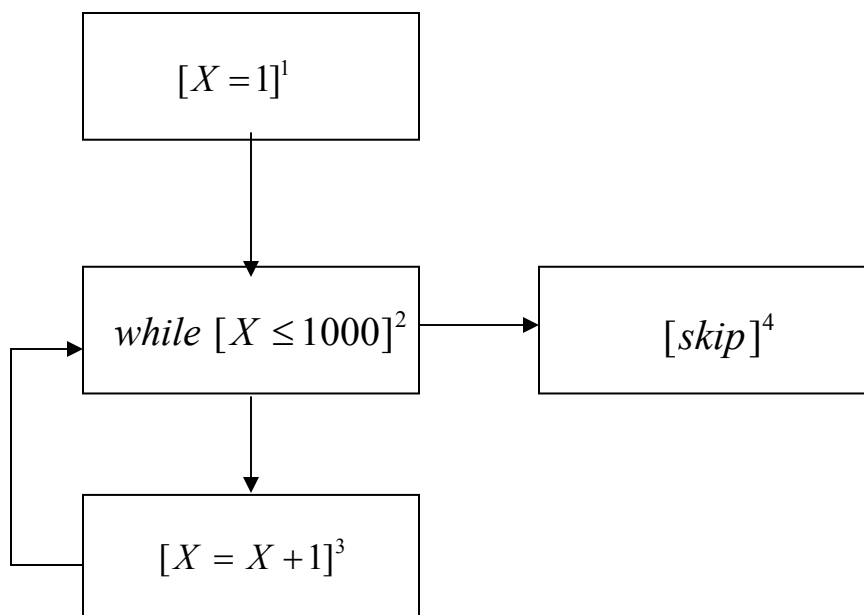
The new chaotic iteration algorithm changes the line:

$$\text{new} = v.d\text{fentry} \sqcup \text{temp}$$

to:

$$\text{new} = v.d\text{fentry} \nabla \text{temp}$$

To be more accurate we use widening once in a loop, from connivance we do it at the condition of the loop.



Now we'll run the improved version of the algorithm on this example.

$$\begin{aligned} v1.defEntry &= [1,1] \\ v2.defEntry &= v1.defEntry \nabla v3.defEntry \\ v3.defEntry &= v2.defEntry \sqcap [-\infty, 1000] + [1,1] \\ v4.defEntry &= v2.defEntry \sqcap [1001, \infty] \end{aligned}$$

After running this equations (1, 2, 3, 2, 3, 4) we will get:

$$\begin{aligned} v1.defEntry &= [1,1] \\ v2.defEntry &= [1, \infty] \\ v3.defEntry &= [2, 1001] \\ v4.defEntry &= [1001, \infty] \end{aligned}$$

Here as you can see $v2.defEntry$, $v4.defEntry$ are inexact. So there is a motivation to define narrowing.

The properties of narrowing:

$$\text{Def1: } y \sqsubseteq x \Rightarrow y \sqsubseteq (x \Delta y) \sqsubseteq x$$

The narrowing gets something between x and y.

$$\begin{aligned} \forall \{x_i\} \quad s.t \quad x_{i+1} \sqsubseteq x_i \\ \{y_i\} := \end{aligned}$$

$$\begin{aligned} \text{Def2:} \quad y_0 &= x_0 \\ y_{i+1} &= y_i \Delta x_{i+1} \\ \exists k \in \mathbb{N} \quad s.t \quad \forall i > k \quad y_k &= y_i \end{aligned}$$

Theorem: $f : L \rightarrow L$ monotonic function, $y_0 = x \in \text{Re } d(f)$, $y_{i+1} = y_i \Delta f(x_i)$ therefore $\exists k \quad y_k = y_{k+1} \wedge y_k \in \text{Re } d(f) = \{l \mid l \in L \wedge l \sqsubseteq f(l)\}$

Proof:

First lets' see that: $y_{i+1} \sqsubseteq y_i$

$y_{i+1} = y_i \Delta f(y_i) \sqsubseteq y_i$ That holds if $f(y_i) \sqsubseteq y_i$ by Def1.

$f(y_0) \sqsubseteq y_0$ Since $x \in \text{Re } d(f)$, now in induction we need to prove that if $f(y_i) \sqsubseteq y_i$ then $f(y_{i+1}) \sqsubseteq y_{i+1}$: $f(y_{i+1}) \sqsubseteq f(y_i) \sqsubseteq y_i \Delta f(y_i) = y_{i+1}$. Now by applying Def2 there is a y_k s.t $y_k = y_{k+1}$ and we proved that for each y_i , $f(y_i) \sqsubseteq y_i$ therefore $f(y_k) \sqsubseteq y_k$ from that by definition of $\text{Re } d(f)$, $y_k \in \text{Re } d(f)$.

If we will get back to Interval analysis example we will define:

$$[a,b]\Delta\phi = [a,b]$$

$$[a,b]\Delta[c,d] = [if\ a = -\infty\ c\ a,\ if\ b = \infty\ d\ b]$$

The definition of narrowing is not symmetric. The definition gets us back from ∞ that widening may have caused. (When narrowing “sees” ∞ it acts like \sqcap)

Now let’s see the desired example again with narrowing. We actually run the widening code but at the end we add iteration with narrowing.

The new chaotic iteration algorithm changes the line:

$$new = v.defEntry \sqcup temp$$

To:

$$new = v.defEntry \nabla temp$$

And then for one more iteration to the line:

$$new = v.defEntry \Delta temp$$

Now the equations are:

$$v1.defEntry = [1,1]$$

$$v2.defEntry = v2.defEntry \Delta (v1.defEntry \sqcup v3.defEntry)$$

$$v3.defEntry = v2.defEntry \sqcap [-\infty, 1000] + [1,1]$$

$$v4.defEntry = v2.defEntry \sqcap [1001, \infty]$$

$$v1.defEntry = [1,1]$$

$$v2.defEntry = [1, \infty]$$

$$v3.defEntry = [2, 1001]$$

$$v4.defEntry = [1001, \infty]$$

After the widening iterations.

After running this once we will get (1, 2, 4):

$$v1.defEntry = [1,1]$$

$$v2.defEntry = [1, \infty] \Delta ([1,1] \sqcup [2,1001]) = [1, \infty] \Delta [1,1001] = [1,1001]$$

$$v3.defEntry = [2,1001]$$

$$v4.defEntry = [1,1001] \sqcap [1001, \infty] = [1001,1001]$$

We got the accurate results of the standard chaotic iterations algorithm but we managed to get them in 9 iterations instead of more than 1000.

Now let’s go back to the second sample program – on which the Chaotic iteration algorithm will not terminate

$$[X = 1]^1$$

$$while [X > 0]^2$$

$$[X = X + 1]^3$$

$$[skip]^4$$

We'll run the improved version of the algorithm on this example

$$\begin{aligned}v1.defEntry &= [1,1] \\v2.defEntry &= v1.defEntry \nabla v3.defEntry \\v3.defEntry &= v2.defEntry \sqcap [1, \infty] + [1,1] \\v4.defEntry &= v2.defEntry \sqcap [-\infty, 1000]\end{aligned}$$

After running this equations (1, 2, 3, 2, 3, 4) we will get:

$$\begin{aligned}v1.defEntry &= [1,1] \\v2.defEntry &= [1, \infty] \\v3.defEntry &= [2, \infty] \\v4.defEntry &= \emptyset\end{aligned}$$

The result here is precise!
Narrowing will not change the result.

Widening and narrowing ensure sound results, therefore, we can conclude that executing the program above will never exit the loop.

One more thing to think about is that results are function of the order of the evaluation (in the standard algorithm only time was function of the order) that's because widening isn't monotonic:

$$\begin{aligned}[1,2] \nabla [0,4] &= [-\infty, \infty] \\[0,4] \nabla [0,4] &= [0,4]\end{aligned}$$

Narrowing can also yield imprecise result for example we will modify the program:

$$\begin{aligned}[X = 1]^1 \\while [X = 4 \vee X \leq 1000]^2 \\ \quad [X = X + 1]^3 \\[skip]^4\end{aligned}$$

In that case we will get the same result as before but now it's imprecise, since the precise result is:

$$\begin{aligned}v1.defEntry &= [1,1] \\v2.defEntry &= [1,4] \\v3.defEntry &= [2,4] \\v4.defEntry &= [4,4]\end{aligned}$$

Galois Connections and Galois Insertions

Static analysis can be viewed as interpreting the program over an “abstract domain”
The program is executed over larger set of execution paths.

Let \mathbf{C} and \mathbf{A} be two domains (or lattices)

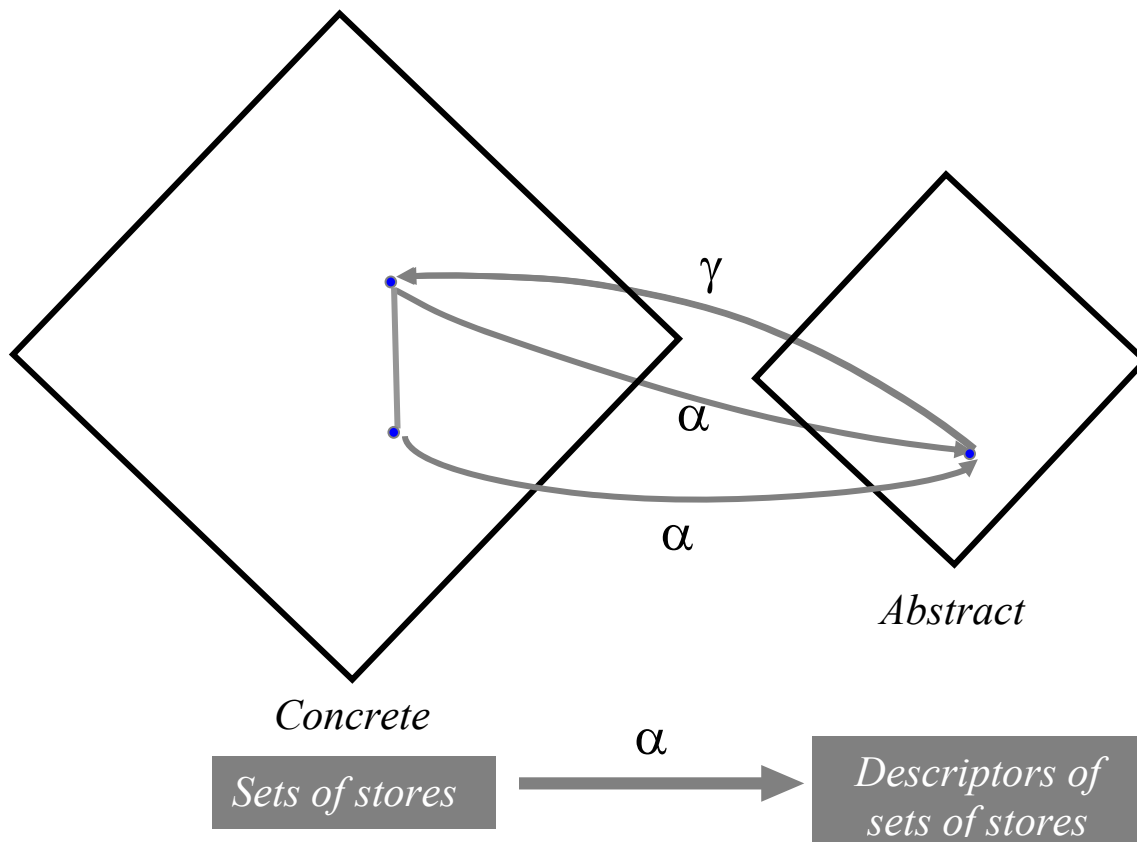
\mathbf{C} – concrete, \mathbf{A} – abstract

Let $\alpha: \mathbf{C} \rightarrow \mathbf{A}$ and $\gamma: \mathbf{A} \rightarrow \mathbf{C}$ be an abstraction function and a concretization function respectively.

The pair of functions (α, γ) form a **Galois Connection** if:

- ❖ both α and γ are monotone (order preserving)
- ❖ $\forall a \in \mathbf{A} \ \alpha(\gamma(a)) \subseteq a$
- ❖ $\forall c \in \mathbf{C} \ c \subseteq \gamma(\alpha(c))$
- ❖ alternatively if
 $\forall a \in \mathbf{A} \ \forall c \in \mathbf{C} \ \alpha(c) \subseteq a \Leftrightarrow c \subseteq \gamma(a)$
(derives both $\forall a \in \mathbf{A} \ \alpha(\gamma(a)) \subseteq a$ and $\forall c \in \mathbf{C} \ c \subseteq \gamma(\alpha(c))$)

α and γ uniquely determine each other



The pair of functions (α, γ) form a **Galois Insertion** if

- ❖ it forms a **Galois Connection**
- ❖ $\forall a \in A \ a \subseteq \alpha(\gamma(a)) \Rightarrow (\forall a \in A \ a = \alpha(\gamma(a)))$

For $C = P(\Sigma)$ let $\beta : \Sigma \rightarrow A$ then the Galois connection is defined by:

$$\alpha(c) = \cup \{ \beta(\sigma) \mid \sigma \in c \}$$

$$\gamma(a) = \{ \sigma \mid \beta(\sigma) \subseteq a \}$$

β works on an individual concrete state, α works on a set of concrete states.

Example: Odd/Even abstract interpretation

The abstract states are $\perp, E, O, ?$

$$\alpha(\emptyset) = \perp, \gamma(\perp) = \emptyset$$

$$\alpha(\{0\}) = \alpha(\{2\}) = \alpha(\{0, 2\}) = \alpha(\{x : x \in \text{Even}\}) = E$$

$$\gamma(E) = \{x : x \in \text{Even}\}$$

$$\alpha(\{-2, 1, 5\}) = ?$$

$$\gamma(?) = \text{All concrete states}$$

Example: Constant Propagation

In order to prove soundness of the constant propagation algorithm we need to:

- ❖ Define an "appropriate" structural operation semantics
- ❖ Define "collecting" structural operation semantics
- ❖ Establish a Galois connection between collecting states and the abstract states
- ❖ (Local correctness) Show that the abstract interpretation of every atomic statement is sound with respect to the collecting semantics
- ❖ (Global correctness) Conclude that the analysis is sound according to [CC76]

The Abstraction Function, maps collecting states into constants

for an individual state $\beta_{CP} : [\text{Var}^* \rightarrow Z] \rightarrow [\text{Var}^* \rightarrow Z \cup \{\perp, T\}]$

$$\beta_{CP}(\sigma) = \sigma$$

for a set of states $\alpha_{CP} : P([\text{Var}^* \rightarrow Z]) \rightarrow [\text{Var}^* \rightarrow Z \cup \{\perp, T\}]$

$$\alpha_{CP}(CS) = \cup \{ \beta_{CP}(\sigma) \mid \sigma \in CS \} = \cup \{ \sigma \mid \sigma \in CS \}$$

soundness: $\alpha_{CP}(\text{Reach}(v)) \subseteq \text{df}(v)$

The Concretization Function, maps constants into collecting states
the formal meaning of constants

the concretization $\gamma_{CP} : [\text{Var}^* \rightarrow Z \cup \{\perp, T\}] \rightarrow P([\text{Var}^* \rightarrow Z])$

$\gamma_{CP}(df) = \{\sigma \mid \beta_{CP}(\sigma) \subseteq df\} = \{\sigma \mid \sigma \subseteq df\}$

soundness: $\text{Reach}(v) \subseteq \gamma_{CP}(df(v))$

α_{CP} is monotone, γ_{CP} is monotone

$\forall df \in [\text{Var}^* \rightarrow Z \cup \{\perp, T\}] \alpha_{CP}(\gamma_{CP}(df)) \subseteq df$

$\forall c \in P([\text{Var}^* \rightarrow Z]) c \subseteq \gamma_{CP}(\alpha_{CP}(c))$

$\gamma(\{[x \rightarrow 5, y \rightarrow \perp]\}) = \emptyset$

$\alpha(\emptyset) = \{[x \rightarrow \perp, y \rightarrow \perp]\}$

$[x \rightarrow 5, y \rightarrow \perp] \subseteq \alpha(\gamma(\{[x \rightarrow 5, y \rightarrow \perp]\})) = \{[x \rightarrow \perp, y \rightarrow \perp]\}$

A state with a constant value of \perp is actually equivalent to the state where all constants have value of \perp

Example: Interval Analysis

$I = [a, b] \quad \Sigma = \mathbf{Z}$

$\beta : \mathbf{Z} \rightarrow I$

$\beta(z) = [z, z]$

$\alpha : P(\mathbf{Z}) \rightarrow I$

$\alpha(X) = \bigcup_{z \in X} [z, z] = [\min(X), \max(X)]$

$\gamma : I \rightarrow P(\mathbf{Z})$

$\gamma([l, u]) = \{z \mid l \leq z \leq u\}$

Example: Points-To Analysis

$PI = P(\text{Var}^* \text{Var}) \quad \Sigma = \text{Var} \rightarrow \text{Var} \cup \mathbf{Z}$

$\beta : \Sigma \rightarrow PI$

$\beta(\sigma) = \{(x, y) \mid \sigma(x) = y\}$

$\alpha : P(\Sigma) \rightarrow PI$

$\alpha(W) = \bigcup_{w \in W} \{\beta(w)\} = \{(x, y) \mid \exists \sigma \in W \sigma(x) = y\}$

$\gamma : PI \rightarrow P(\Sigma)$

$\gamma(Pt) = \{\sigma \mid \beta(\sigma) \subseteq Pt\} = \{\sigma \mid \forall x, y \in \text{Var} \sigma(x) = y \Rightarrow (x, y) \in Pt\}$

Collecting Semantics

The collecting semantics is used to define a collection of all the states, defined by structural operational semantics, of the program in a given point.

The input state is not known at compile-time so all the states for all possible inputs to the program are collected.

It may be incomputable, however it is well defined.

It does not lose information nor precision.

Example 1

```
    {[x → 0]}
while (true) do
    {[x → 0], [x → 1], [x → 2] ...}
    x := x + 1
    {[x → 1], [x → 2], [x → 3] ...}
```

Example 2

```
    {[x → 0, y → 0, z → 0]}
z = 3
    {[x → 0, y → 0, z → 3]}
x = 1
    {[x → 1, y → 0, z → 3]}
while (x > 0) do
    {[x → 1, y → 0, z → 3], [x → 3, y → 0, z → 3]}
    If (x == 1) then
        y = 7
        {[x → 1, y → 7, z → 3], [x → 3, y → 7, z → 3]}
    else
        y = z + 4
        {[x → 1, y → 7, z → 3], [x → 3, y → 7, z → 3]}
    x = 3
    {[x → 3, y → 7, z → 3]}
    print y
    {[x → 3, y → 7, z → 3]}
```

If the meaning of every statement is locally sound then, the solution computed by the iterative algorithm overapproximates the collecting semantics

$$\alpha(CS) \subseteq df$$

$$CS \subseteq \gamma(df)$$

Example (rule of signs)

```
P      {[x → 0]}
while (true) do
P      {[x → 0], [x → 1], [x → 2] ...}
      x := x + 1
P      {[x → 1], [x → 2], [x → 3] ...}
```

Bad Example (rule of signs)

```
P      {[x → 0]}
x := x - 1
?      {[x → -1]}
x := x + 1
?      {[x → 0]}
```

An “Iterative” Definition of Collecting Semantics:

Generate a system of monotone equations, the least solution is well defined.

The least solution is the collecting interpretation.

Again, it may be incomputable.

Equations generated for collecting interpretation:

- **Equations for elementary statements:**

- [skip]

$$CS_{\text{exit}}(l) = CS_{\text{entry}}(l)$$

- [b]

$$CS_{\text{exit}}(l) = \{\sigma : \sigma \in CS_{\text{entry}}(l), \llbracket b \rrbracket \sigma = \text{tt}\}$$

- [x := a]

$$CS_{\text{exit}}(l) = \{(s[x \rightarrow A\llbracket a \rrbracket s]) \mid s \in CS_{\text{entry}}(l)\}$$

- **Equations for control flow constructs:**

$$CS_{\text{entry}}(l) = \cup CS_{\text{exit}}(l') \quad l' \text{ immediately precedes } l \text{ in the CFG.}$$

- **An Equation for the entry**

$$CS_{\text{entry}}(l) = \{\sigma : \sigma \in \text{Var}^* \rightarrow Z\}$$

System of Equations (Collecting Semantics):

$$S = \left\{ \begin{array}{l} \text{CSentry}[s] = \{ \sigma_0 \} \\ \text{CSentry}[v] = \cup \{ f(e)(\text{CSentry}[u]) \mid (u, v) \in E \} \\ \text{where } f(e) = \lambda X. \{ \llbracket \text{st}(e) \rrbracket \sigma \mid \sigma \in X \} \text{ for atomic statements} \\ f(e) = \lambda X. \{ \sigma \mid \llbracket b(e) \rrbracket \sigma = \text{tt} \} \end{array} \right\}$$

S here, is a set of n equations.

The Least Solution

An alternative definition is using both CSentry[l] and CSexit[l] of each node l:

$$S = \left\{ \begin{array}{l} \text{CSentry}[s] = \{ \sigma_0 \} \\ \text{CSentry}[v] = \cup \{ \text{CSexit}[u] \mid (u, v) \in E \} \\ \text{CSexit}[v] = f(v)(\text{CSentry}[v]) \end{array} \right\}$$

CSentry[1], ..., CSentry[n], CSexit[1], ..., CSexit[n]

S here, is a set of 2n equations.

The set of equations can be written in vectorial form:

$$F_S : L^n \rightarrow L^n$$

$$F_S(X)[v] = \cup \{ f(e)[u] \mid (u, v) \in E \}$$

$$\overline{CS} = F_S(\overline{CS})$$

The least fixed point of the function F_S , used to define the collecting semantics, is well-defined, and from the equivalence between the system of equations $\text{lfp}(S) = \text{lfp}(F_S)$.

Each component of F_S is minimal and since F_S is monotone such a solution always exists.

Example

```
[x := 0]0  
while [(true)]1 do  
  [x := x + 1]2
```

$$CS_{\text{entry}}[0] = \{ [x \rightarrow 0] \}$$
$$CS_{\text{exit}}[0] = \{ \llbracket x := 0 \rrbracket \sigma \mid \sigma \in CS_{\text{entry}}[0] \}$$
$$CS_{\text{entry}}[1] = CS_{\text{exit}}[0] \cup CS_{\text{exit}}[2]$$
$$CS_{\text{exit}}[1] = CS_{\text{entry}}[1]$$
$$CS_{\text{entry}}[2] = CS_{\text{exit}}[1]$$
$$CS_{\text{exit}}[2] = \{ \llbracket x := x + 1 \rrbracket \sigma \mid \sigma \in CS_{\text{entry}}[2] \}$$

The set of equations presents exactly all reachable states.

We would like to thank Ori's wife, Liat, aka mami.