Program Analysis **November 1, 2015**

Lecture 3: Abstract Interpretation

Lecturer: Mooly Sagiv *Scribe: Dor Wucher, Yael Tiomkin*

3.1 Computing Constants

The steps to computing constants are:

1. Construct a control flow graph (CFG).
2. Associate transfer functions with control flow graph edges.
3. Define a system of equations.
4. Compute the simultaneous least fixed point via Chaotic Iterations.

The solution is unique, but the order of evaluation may affect the number of iterations.

3.1.1 Example

We will use the following example and go over all the steps:

```
[z := 3]1
[x := 1]2
while ([x > 0]3) (
      if [x = 1]4 then [y := 7]5
                  else [y := z + 4]6
       [x := 3]7
                              x :=3
)
```

Step 1 and 2 - Construct a control flow graph and associate transfer functions with edges:

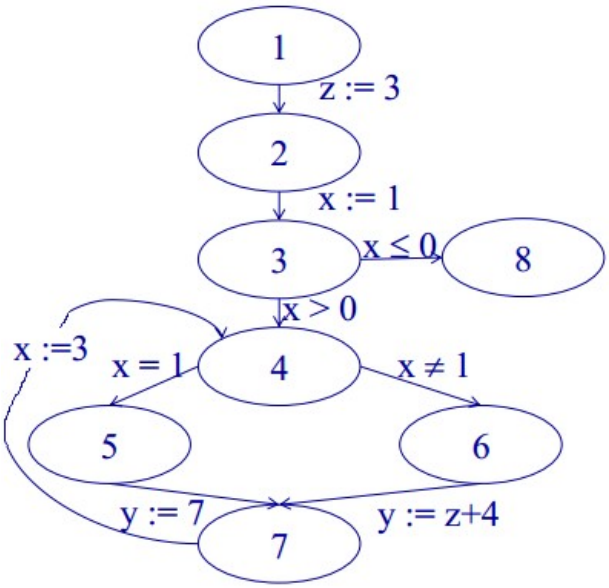


Figure 3.1: Every node represenates a line and between the nodes we have the matching transfer functions

Step 3 - Define a system of equations:

$$DF[1] = [x \mapsto 0, z \mapsto 0]$$
$$DF[2] = DF[1][\![z \mapsto 3]\!]^{\#}$$
$$DF[3] = DF[2][\![x \mapsto 1]\!]^{\#}$$
$$DF[4] = DF[3][\![x > 0]\!]^{\#} \sqcup DF[7][\![x \mapsto 3]\!]^{\#}$$
$$DF[5] = DF[4][\![x = 1]\!]^{\#}$$
$$DF[6] = DF[4][\![x \neq 1]\!]^{\#}$$
$$DF[7] = DF[5][\![y := 7]\!]^{\#} \sqcup DF[6][\![y := z+4]\!]^{\#}$$
$$DF[8] = DF[3][\![x \leq 1]\!]^{\#}$$

5

Figure 3.2: First, we initilizing all constants to 0. we proceed to each state by the following: State 2: from state 1 by applying the transfer function z→3. State 3: from state 2 by applying the transfer function x→1. State 4: from state 3 when x¿0 or from state 7 by applying the transfer function x→3. State 5: from state 4 when x→1. State 6: from state 4 when x1. State 7: from state 5 by applying the transfer function y→7 or from state 6 by applying the transfer function y→z+4. State 8: from state 3 when $x \leq 1$.

Step 4 - Compute the simultaneous least fixed point via Chaotic Iterations:

N	DF[N]	WL
		{1}
1	[x↦0, y↦0, z ↦0]	{2}
2	[x↦0, y↦0, z ↦3]	{3}
3	[x↦1, y↦0, z ↦3]	{4, 8}
4	[x↦1, y↦0, z ↦3]	{5, 6, 8}
5	[x↦1, y↦0, z ↦3]	{6, 7, 8}
6		{7, 8}
7	[x↦1, y↦7, z ↦3]	{3, 8}
3	[x↦⊤, y ↦⊤, z ↦3]	{4, 8}
4	[x↦⊤, y↦⊤, z ↦3]	{5, 6, 8}
5	[x↦1, y↦⊤, z ↦3]	{6, 7, 8}
6	[x↦⊤, y↦⊤, z ↦3]	{7, 8}
7	[x↦⊤, y↦7, z ↦3]	{3, 8}
3		{8}
8	[x↦⊤, y ↦⊤, z ↦3]	{}

Figure 3.3: We compute the simultaneous least fixed point via Chaotic Iterations as follows: The initial WL is 1. We update DF[1] to [x→0, y→0, z→0] by initializing and add 2 to the WL. We update DF[2] to [x→0, y→0, z→3] by applying the transfer function [z→3] and add 3 to the WL. We update DF[3] to [x→1, y→0, z→3] by applying the transfer function [x→1] and add 4,8 to the WL. We apply the transfer function [x¿0] and add 5,6 to the WL. We apply the transfer function [x→1] and add 7 to the WL. We apply the transfer function [x1] and add nothing to the WL since x→1. We update DF[7] to [x→1, y→7, z→3] by applying the transfer function [y→7] and add 3 to the WL. We update DF[3] to [x→3, y→7, z→3] by applying the transfer function [x→3] and then join with [x→1, y→7, z→3]. We update DF[3] to [x→ ⊤, y→ ⊤, z→3] and add 4 to the WL. We apply the transfer function [x¿0] and add 5,6 to the WL. We apply the transfer function [x→1] and add 7 to the WL. We apply the transfer function [x1] and add nothing to the WL since x→1.

3.1.2 Losing precision

- Correlated branches - In case we have some correlated branches, we lose precision by choosing the same value in all branches.
- "Join-over-all-path - In a given node, we'll take all the possible (infinite number of) paths to it and join them (union). This equals to using the Least Fixed Point. It gives completeness, because it can discover paths with problems (because it contains all the paths), even though they are actually not
- Initial value - the initial value of the variables can be set to 0, $\top$, $\bot$, etc. and may influence the precision of the result.
- Dynamic vs. Static values.

3.1.3 Conclusions

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- Expensive

  More efficient methods exist for structured programs

3.2 Abstract Interpretation

3.2.1 Specialized Chaotic Iterations

This is another algorithm for finding a lfp(f) (CFG algorithm). It works for 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}
```

Figure 3.4: We start in some node s, and we assume that there are no entering edges to s. The initial value is T. DF is a data flow.

3.2.2 The Abstract Interpretation Technique

3.2.2.1 Motivation

- The foundation of program analysis
- Defines the meaning of the information computed by static tools
- A mathematical framework
- Allows proving that an analysis is sound in a local way
- Identify design bugs
- Understand where precision is lost
- New analysis from old
- Not limited to certain programming style

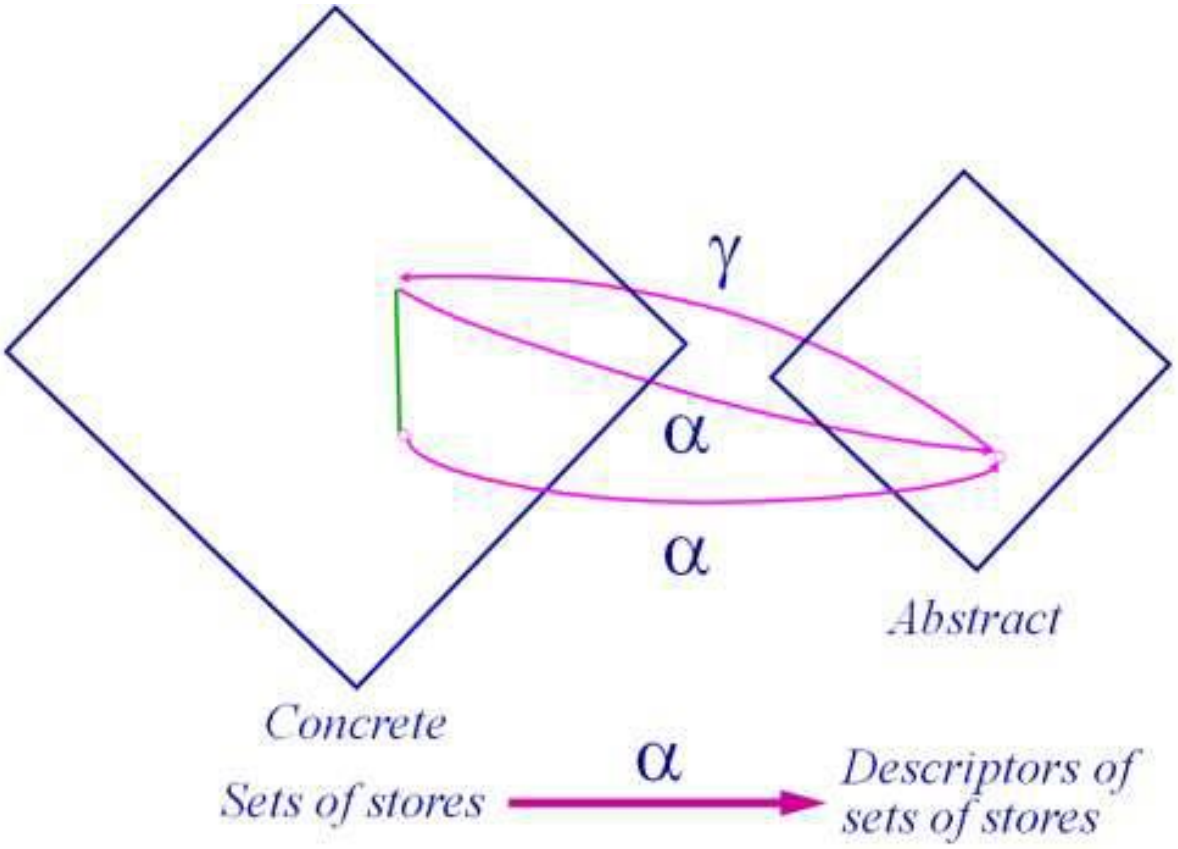


Figure 3.5: The illustration shows that concretization of an abstraction of every element from the concrete domain should be equal or larger than the original element. We can see in the illustration two objects from the concrete domain that are mapped to a single object in the abstract domain. When the matching object in the abstract domain is mapped back to the concrete domain it is mapped to a group in the lattice that is larger (or at least not smaller) than the group containing each of the original concrete values

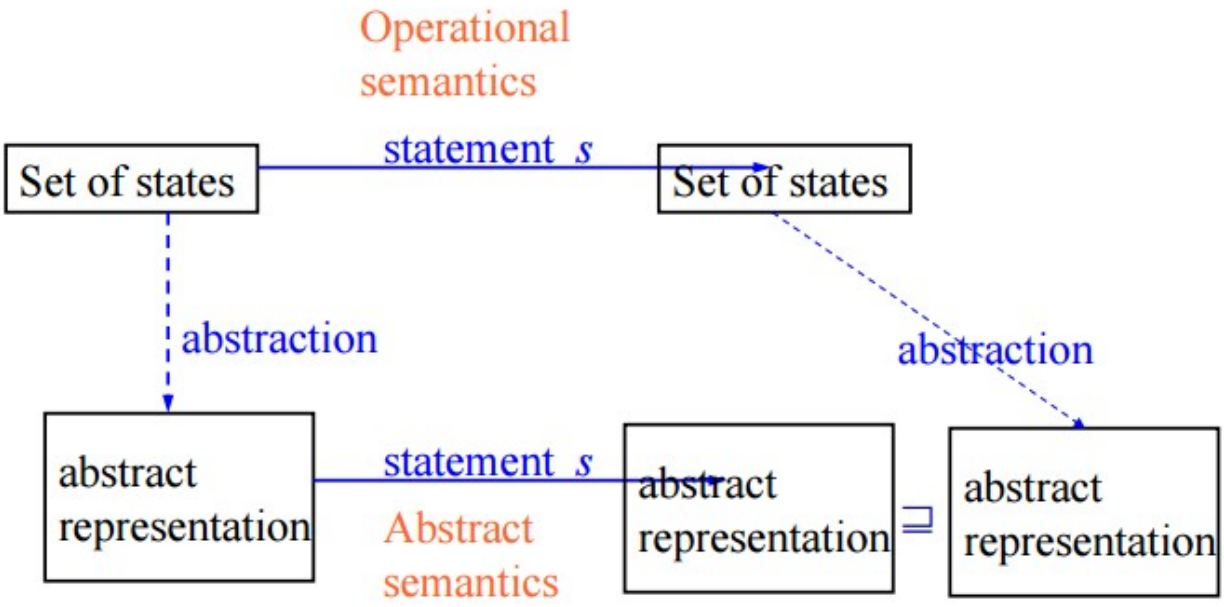


Figure 3.6: When we do an abstraction on a set of states and then apply a statement s on the abstract representation, the abstract representation we get will be greater than the abstract representation we get by first applying statement s and then doing the abstraction.

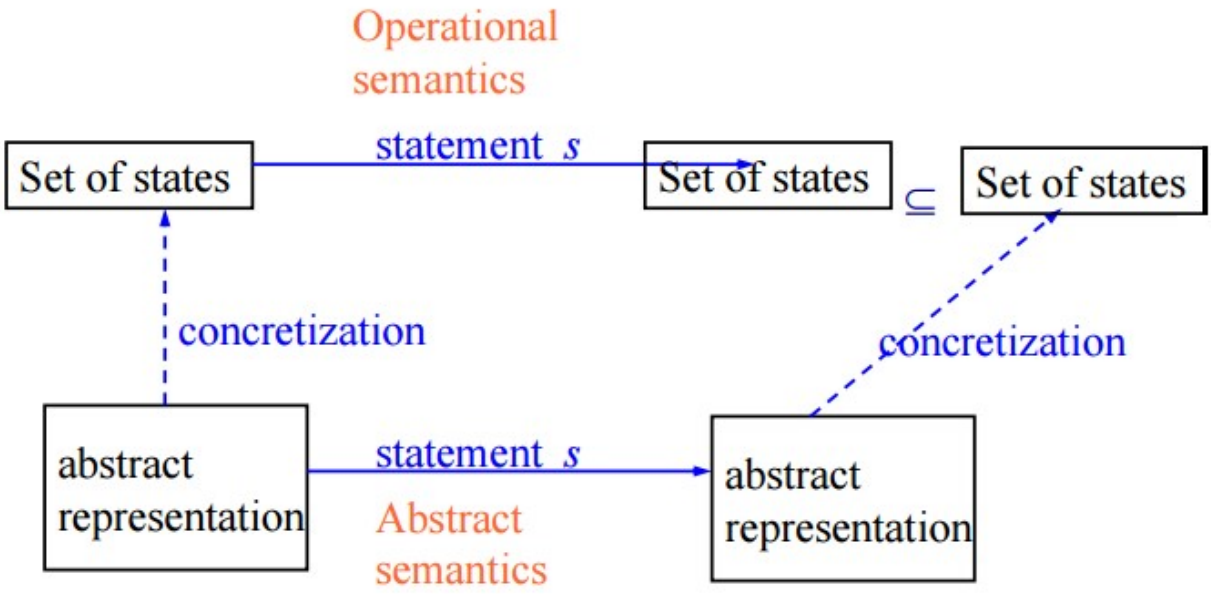


Figure 3.7: When we do a concretization on an abstract representation and then apply a statement s on the set of states, the set of states we get will be contained in the set of states we get by first applying statement s and then doing the concretization on the abstract representation.

3.2.2.2 Abstraction function (Constant Propagation)

The abstraction of an individual state:

$$\beta_{CP}:[Var_* \to Z] \to [Var_* \to Z \cup \{\bot, \top\}]$$
$$\beta_{CP}(\sigma) = \sigma$$

As we can see, the abstraction of individual state is the same state, without any changes.

The abstraction of set of states:

$$\alpha_{CP}:P([Var_* \to Z]) \to [Var_* \to Z \cup \{\bot, \top\}]$$
$$\alpha_{CP}(CS) = \sqcup \{ \beta_{CP}(\sigma) \mid \sigma \in CS\} = \sqcup\{\sigma \mid \sigma \in CS\}$$

The abstraction of set of states is the join of abstractions of each state in the set. In other words, abstraction of set of states gives us the lowest constant that contains all the states.

Monotonic: It's immediate from the definition that the function is monotonic (for a bigger set of states, the constant that includes all the states doesnt become smaller).

Soundness: The abstraction function result is sound:

$$\alpha_{CP}(\text{Reach}(v)) \sqsubseteq df(v)$$

Figure 3.8: Reach(v) = all the possible variables' values that are accessible from the state v. df(v) = the result we get from the chaotic iterations.

For example:

$\alpha_{CP}([X \to 1, Y \to 2], [X \to 2, Y \to 2]) = [X \to ?, Y \to 2]$

We can see in that example that we lose information when we use abstraction, because the result includes $[X \to 3, Y \to 2]$, which is not in the original set of states.

Completeness: always yields a result.

3.2.2.3 Concretization function (Constant Propagation)

The concretization of a data flow:

$$\gamma_{CP}: [Var_* \to Z \cup \{\bot, \top\}] \to P([Var_* \to Z])$$
$$\gamma_{CP}(df) = \{\sigma \mid \beta_{CP}(\sigma) \sqsubseteq df\} = \{\sigma \mid \sigma \sqsubseteq df\}$$

The concretization function on a state s gives us all the concrete states that s can represent.

Monotonic: same as the abstraction function, it's immediate from the definition that the concretization function is also monotonic.

Soundness: The concretization function result is sound:

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

For example:

$\gamma_{CP}([X \to 1, Y \to ?]) = [X \to 1, Y \to 1], [X \to 1, Y \to 2], [X \to 1, Y \to 3],$

In this example we can see the concretization function gives us more information, meaning that the result states are substantial.

Completeness: always yields a result.

3.2.2.4 Galois connections

Definition:
Given lattices C and A, and functions $\alpha : C \to A$ and $\gamma : A \to C$, the pair of functions (α, γ) form Galois connection if:

- α and γ are monotone.
- $\forall a \in A : \alpha(\gamma(a)) \subseteq a$
- $\forall c \in C : c \subseteq \gamma(\alpha(a))$

Alternatively, if:

- $\forall a \in A, \forall c \in C : \alpha(c) \subseteq a \iff c \subseteq \gamma(a)$

Usage:
By using Galois connection we can prove that α and γ uniquely determine each other.
γ determined uniquely by α proof (the "α determined uniquely by γ" proof is similar):

- Suppose that α and γ_1 forms a Galois connection and α and γ_2 forms a Galois connection.
- From the second definition we get $\alpha(c) \subseteq a$ iff $c \subseteq \gamma_1(a) \iff c \subseteq \gamma_2(a)$.
- It holds that $\forall a \in A : \gamma_1(a) \subseteq \gamma_1(a)$ and from the previous sentence we derive that $\forall a \in A : \gamma_1(a) \subseteq \gamma_2(a)$.
- It holds that $\forall a \in A : \gamma_2(a) \subseteq \gamma_2(a)$ and from the previous sentence we derive that $\forall a \in A : \gamma_2(a) \subseteq \gamma_1(a)$
- Finally we get from the previous 2 sentences that $\gamma_1 = \gamma_2 \to \gamma$ determined uniquely by α.

The abstraction and concretization functions form a Galois connection by fulfilling the following conditions:

- Both functions are monotonic.
- $\forall df \in [Var_* \to Z \cup [\top, \bot]] : \alpha_{CP}(\gamma_{CP}(df)) \subseteq df$
- $\forall c \in P([Var_* \to Z]) : c_{CP} \subseteq \gamma_{CP}(\alpha_{CP}(c))$

Example:
Let df = $[X \to 1, Y \to ?]$
$\alpha_{CP}(\gamma_{CP}(df)) = \alpha_{CP}(\{..., [X \to 1, Y \to 1], [X \to 1, Y \to 2], ...\}) = [X \to 1, Y \to ?]$
we get: $[X \to 1, Y \to ?] \subseteq [X \to 1, Y \to ?]$ and the second condition is fulfilled.
Let S = $\{[X \to 1, Y \to 1], [X \to 1, Y \to 2]\}$
$\gamma_{CP}(\alpha_{CP}(S)) = \gamma_{CP}([X \to 1, Y \to \bot]) = \{[X \to 1, Y \to 1], [X \to 1, Y \to 2], ...\}$
we get: $\{[X \to 1, Y \to 1], [X \to 1, Y \to 2]\} \subseteq \{[X \to 1, Y \to 1], [X \to 1, Y \to 2], ...\}$ and the third condition is fulfilled.

3.2.2.5 Upper closures

Upper closure is a function $\uparrow: P(\Sigma) \to P(\Sigma)$ such that:

- $\uparrow$ is monotonic ($X \subseteq Y \to \uparrow X \subseteq \uparrow Y$)
- $\uparrow$ is extensive ($X \subseteq \uparrow X$)
- $\uparrow$ is closure ($\uparrow (\uparrow X) = \uparrow X$)

Every Galois connection defines an upper closure by composing γ and α:
This function is monotonic, because γ and α are both monotonic, and we can see that the second and third conditions are fulfilled immediately from Galois Connection definitions.

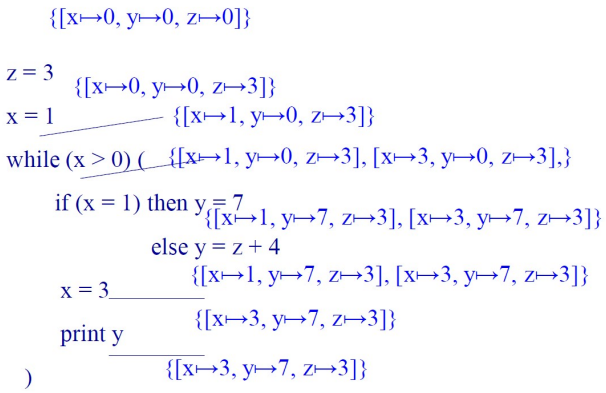
3.2.2.6 Proof of soundness

- Define an appropriate operational semantics.
- Define collecting operational semantics by pointwise extension.
- Establish a Galois connection between collecting states and abstract states.
- (Local correctness) Show that the abstract interpretation of every atomic statement is sound. w.r.t. the collecting semantics.
- (Global correct ness) Conclude that the analysis is sound.

3.2.2.7 Collecting semantics

Definition:
Collect all the states for all possible inputs (the input state is not known at the compile-time) to the program. In other words, execute the program on each possible input, and add the states we get in this execution to the group of states. In this approach, there is no reachable state, which is not in this "collection" and we have no loss of precision.

Example:



Another definition (Iterative one):

- Generate a system of monotone equations (the equations are monotonic, because if we generate the collecting method on a bigger set of inputs, we will get a bigger group of states).
- The least solution is well-defined
- The least solution is the collecting interpretation
- But may not be computable

Equations Generated for Collecting Interpretation:

- Equations for elementary statements:

[skip]

$$\text{CSexit(l)} = \text{CSentry(l)}$$

[b]

$$\text{CSexit(l)} = \{\sigma: \sigma \in \text{CSentry(l)}, [\![b]\!]\sigma = tt\}$$

[x := a]

$$\text{CSexit(l)} = \{(s[x \rightarrow [\![\mathbf{A}]\!]s]) \mid s \in \text{CSentry(l)}\}$$

- Equations for control flow constructs:
  $CS_{entry}(l) = \cup CS_{exit}(l')$ when l' immediately precedes l in the control flow graph. Meaning, the collecting semantic in before executing the line l, is the union of all the possible collecting semantics we get at the end of the previous line (regarding to the execution).
- An equation for the entry (the first line in the execution):
  $CS_{entry}(l) = \{\sigma | \sigma \in Var_* \rightarrow Z\}$

Specialized Chaotic Iterations System of Equations (Collecting Semantics):

$$S = \begin{cases} CS_{entry}[s] = \{\sigma_0\} \\ CS_{entry}[v] = \cup\{f(e)(CS_{entry}[u]) \mid (u, v) \in E\} \\ \text{where } f(e) = \lambda X.\ \{[\![st(e)]\!]\ \sigma \mid \sigma \in X\} \text{ for atomic statements} \\ \qquad f(e) = \lambda X.\{\sigma \mid [\![b(e)]\!]\ \sigma = tt\} \end{cases}$$

Figure 3.9: The solution of this system of equations is the reachable states in the program.

3.2.2.8 The least solution

Defenition:

- 2n sets of equations: $CS_{entry}(1), ..., CS_{entry}(n), CS_{exit}(1), ..., CS_{exit}(n)$.
- Can be written in vectorial form: $\overrightarrow{CS} = F_{CS}(\overrightarrow{CS})$.
- The least solution lfp(F_{CS}) is well-defined.
- Every component is minimal.
- Since F_{CS} is monotone such a solution always exists.
- $CS_{entry}(v) = \{s | \exists s_0 | < P, s_0 > \rightarrow^* (S', s)\}, init(S') = v$
- Simplify the soundness criteria.

Now we will see use of the concretization function, in finding a fix point of a function:
Let $f^{\#}$ be a proximity function of f, which holds:
We compute the least fix point of the function $f^{\#}$. if we gives the concretization function this least fix point as input, we get a point that is bigger than the least fix point of f. We will show now that this is true: let b be the lfp of $f^{\#}$, so it's also holds $f^{\#}(b) = b$ and we get $f(\gamma(b)) \subseteq \gamma(f^{\#}(b)) \subseteq \gamma(b) \rightarrow \gamma(b) \in Red(f) \rightarrow lfp(f) \subseteq \gamma(b)$. Using this fact, we can get a sound approximation to the lfp of f.

3.2.2.9 Soundness Theorem

There are 3 versions of the Soundness Theorem that are different only in their 4^{th} condition:

Version 1:

1. Let (α, γ) form Galois connection from C to A.
2. $f : C \to C$ be a monotone function.
3. $f^{\#} : A \to A$ be a monotone function.
4. $\forall a \in A : f(\gamma(a)) \subseteq \gamma(f^{\#}(a))$

$lfp(f) \subseteq \gamma(lfp(f^{\#}))$
$\alpha(lfp(f)) \subseteq lfp(f^{\#})$

Version 2:

1. Let (α, γ) form Galois connection from C to A.
2. $f : C \to C$ be a monotone function.
3. $f^{\#} : A \to A$ be a monotone function.
4. $\forall c \in C : \aleph(f(c)) \subseteq f^{\#}(a(c))$

$\alpha(lfp(f)) \subseteq lfp(f^{\#})$
$lfp(f) \subseteq \gamma(lfp(f^{\#}))$

Version 3:

1. Let (α, γ) form Galois connection from C to A.
2. $f : C \to C$ be a monotone function.
3. $f^{\#} : A \to A$ be a monotone function.
4. $\forall a \in A : \alpha(f(\gamma(a))) \subseteq f^{\#}(a)$

$\alpha(lfp(f)) \subseteq lfp(f^{\#})$
$lfp(f) \subseteq \gamma(lfp(f^{\#}))$

We mentioned in class that the 3^{rd} version provides a way to define the best transformer.

3.2.2.10 Completeness

$\alpha(lfp(f)) \subseteq lfp(f^{\#})$

$lfp(f) \subseteq \gamma(lfp(f^{\#}))$

Completeness is harder to achieve than soundness, because here we require equality compared to the inclusion we had before. This means that completeness requires the accurate values (and not a group that includes them).