Iterative Program Analysis
Abstract Interpretation

Mooly Sagiv

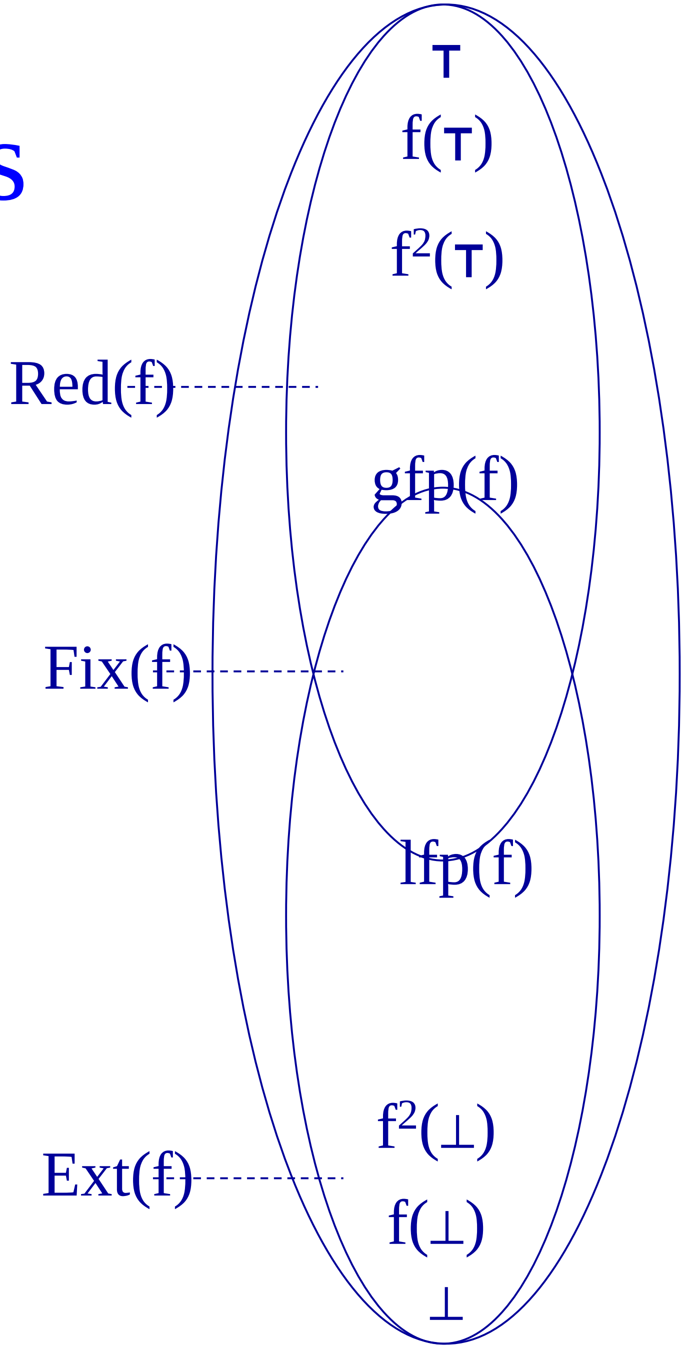
Textbook: Principles of Program Analysis

Chapter 4

CC79, CC92

Fixed Points

- A monotone function f: L $\rightarrow$ L where $(L, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ is a complete lattice
- $Fix(f) = \{ l: l \in L, f(l) = l\}$
- $Red(f) = \{l: l \in L, f(l) \sqsubseteq l\}$
- $Ext(f) = \{l: l \in L, l \sqsubseteq f(l)\}$
  - $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$
- Tarski's Theorem 1955: if f is monotone then:
  - $lfp(f) = \sqcap Fix(f) = \sqcap Red(f) \in Fix(f)$
  - $gfp(f) = \sqcup Fix(f) = \sqcup Ext(f) \in Fix(f)$



Chaotic Iterations

- A lattice L = $(L, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ with finite strictly increasing chains
- $L^n = L \times L \times \ldots \times L$
- A monotone function $\underline{f}: L^n \rightarrow L^n$
- Compute lfp($\underline{f}$)
- The simultaneous least fixed of the system $\{x[i] = \underline{f}_i(x) : 1 \leq i \leq n\}$

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

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

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

for i :=1 to n do

x[i] = $\perp$

WL = {1, 2, …, n}

while (WL $\neq \emptyset$) do

select and remove an element i $\in$ WL

new := $f_i(\underline{x})$

if (new $\neq$ x[i]) then

x[i] := new;

Add all the indexes that directly depends on i to WL

Specialized Chaotic Iterations System of Equations

$S =$

$$\left\{ \begin{array}{l} df_{entry}[s] = \iota \\ df_{entry}[v] = \sqcup \{ f(u, v)\ (df_{entry}[u]) \mid (u, v) \in E \} \end{array} \right\}$$

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

$$F_S\ (X)[s] = \iota$$

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

$$lfp(S) = lfp(F_S)$$

Specialized Chaotic Iterations

Chaotic(G(V, E): Graph, s: Node, L: Lattice, ι: L, f: E $\rightarrow$(L $\rightarrow$L)){

for each v in V to n do $df_{entry}[v] := \perp$

df[s] = ι

WL = {s}

while (WL $\neq$ Ø) do

select and remove an element u $\in$ WL

for each v, such that. (u, v) $\in$E do

temp = $f(e)(df_{entry}[u])$

new := $df_{entry}(v) \sqcup$ temp

if (new $\neq df_{entry}[v]$) then

$df_{entry}[v]$:= new;

WL := WL $\cup${v}

Specialized Chaotic Iterations System of Equations

$S =$

$$\left\{ \begin{array}{l} df_{entry}[s] = \iota \\ df_{entry}[v] = \sqcup\{f(u, v)\ (df_{entry}[u]) \mid (u, v) \in E\ \} \\ \\ F_S : L^n \rightarrow L^n \end{array} \right\}$$

$$F_S\ (X)[s] = \iota$$

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

$$lfp(S) = lfp(F_S)$$

Specialized Chaotic Iterations

Chaotic(G(V, E): Graph, s: Node, L: Lattice, ι: L, f: E $\rightarrow$(L $\rightarrow$L)){

for each v in V to n do $df_{entry}[v] := \perp$

$df[s] = \iota$

WL = {s}

while (WL $\neq$ $\emptyset$) do

select and remove an element u $\in$ WL

for each v, such that. (u, v) $\in$E do

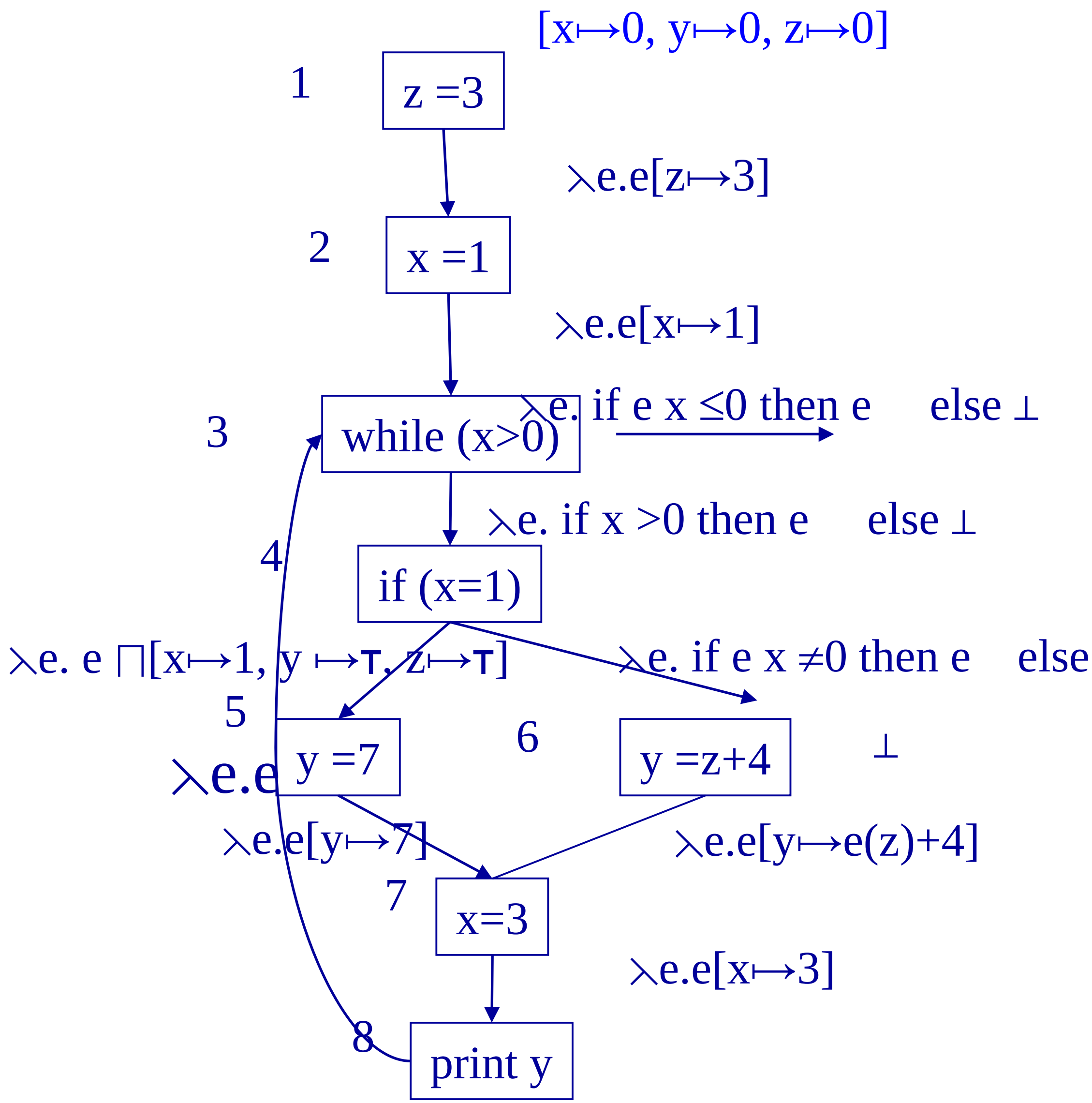
temp = $f(e)(df_{entry}[u])$

new := $df_{entry}(v) \sqcup$ temp

if (new $\neq df_{entry}[v]$) then

$df_{entry}[v]$:= new;

WL := WL $\cup$ {v}

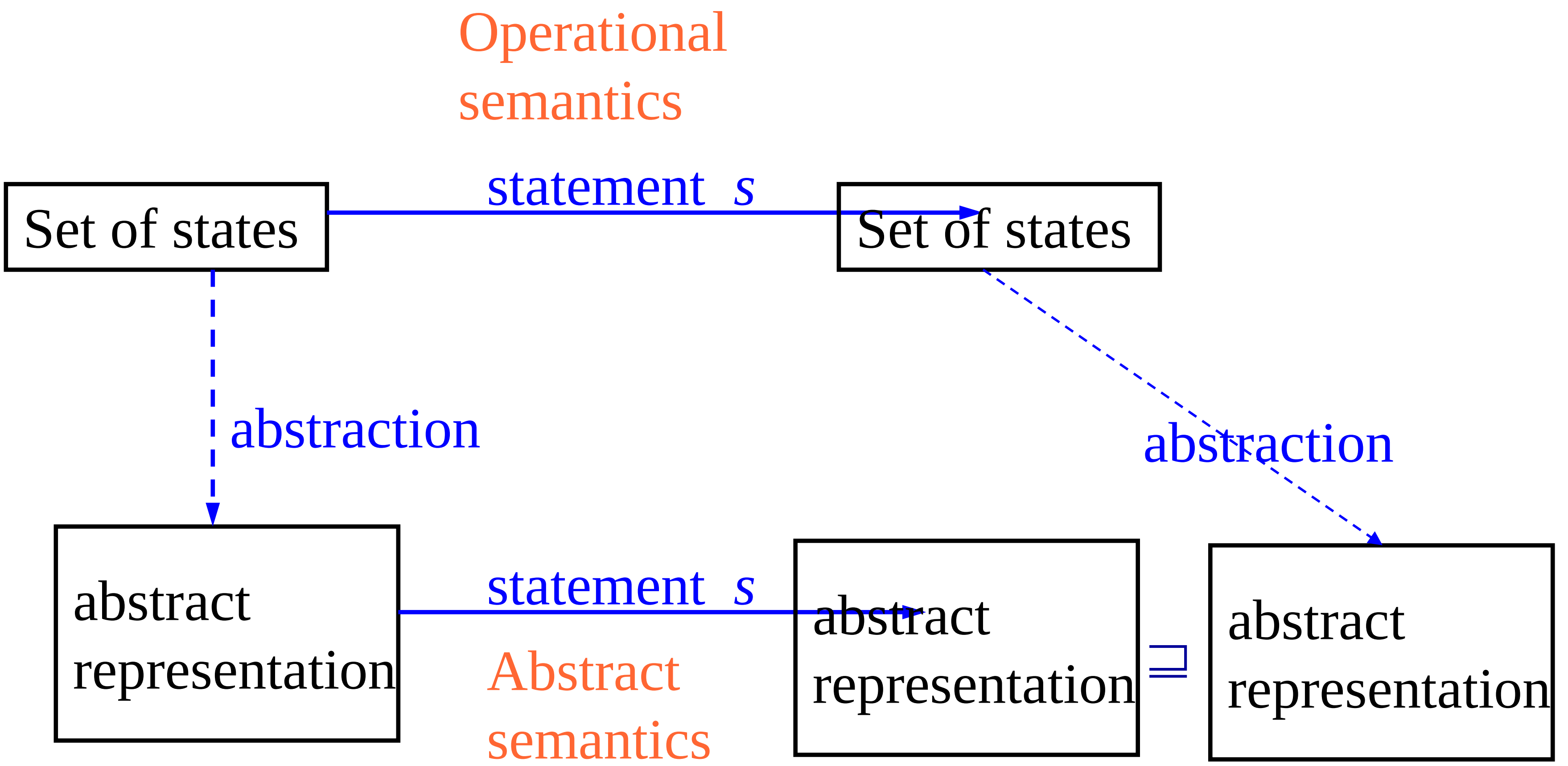


WL	$df_{entry}[v]$
{1}	
{2}	df[2]:=[x⟼0, y⟼0, z⟼3]
{3}	df[3]:=[x⟼1, y⟼0, z⟼3]
{4}	df[4]:=[x⟼1, y⟼0, z⟼3]
{5}	df[5]:=[x⟼1, y⟼0, z⟼3]
{7}	df[7]:=[x⟼1, y⟼7, z⟼3]
{8}	df[8]:=[x⟼3, y⟼7, z⟼3]
{3}	df[3]:=[x⟼T, y⟼T, z⟼3]
{4}	df[4]:=[x⟼T, y⟼T, z⟼3]
{5,6}	df[5]:=[x⟼1, y⟼T, z⟼3]
{6,7}	df[6]:=[x⟼T, y⟼T, z⟼3]
{7}	df[7]:=[x⟼T, y⟼7, z⟼3]

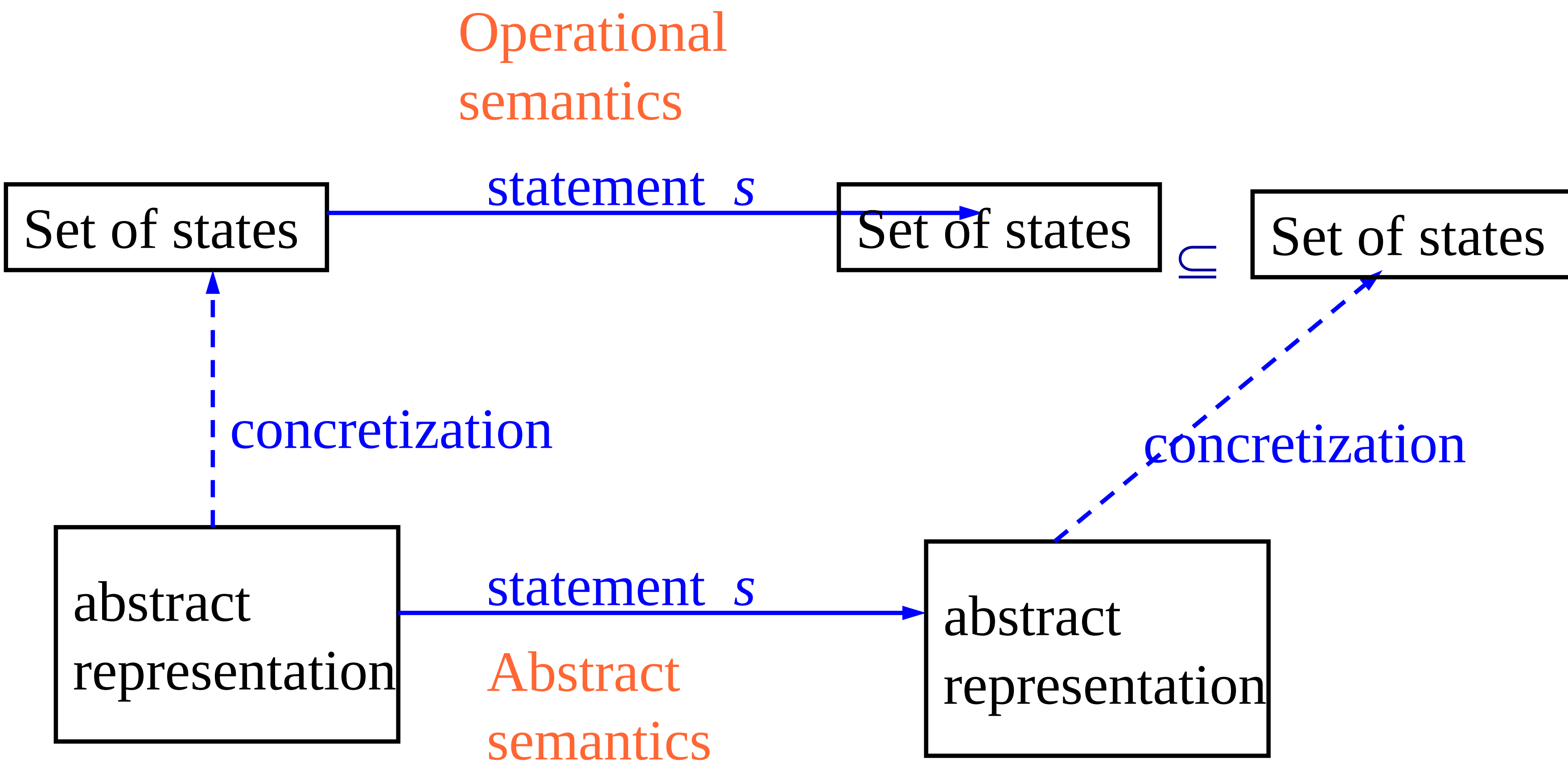
The Abstract Interpretation Technique (Cousot & Cousot)

- 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

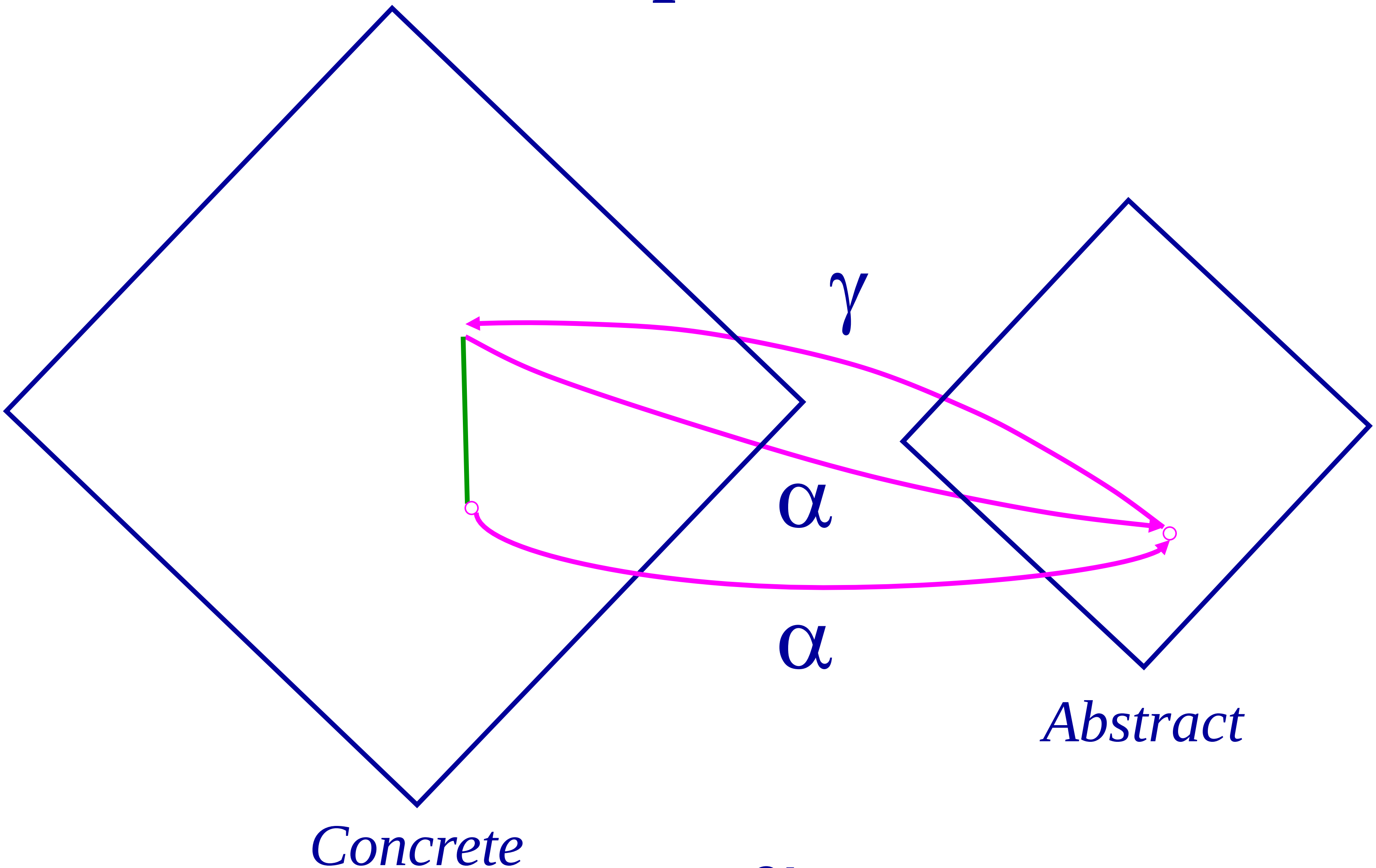
Abstract (Conservative) interpretation



Abstract (Conservative) interpretation



Abstract Interpretation



Sets of stores $\xrightarrow{\alpha}$ *Descriptors of sets of stores*

Galois Connections

- Lattices C and A and functions α: C $\rightarrow$A and γ: A $\rightarrow$C
- The pair of functions (α, γ) form
  Galois connection if
  - α and γ are monotone
  - $\forall$ a $\in$ A
    - » $\alpha(\gamma(a)) \sqsubseteq a$
  - $\forall$ c $\in$ C
    - » $c \sqsubseteq \gamma(\alpha(C))$
- Alternatively if:
  $\forall$ c $\in$ C
  $\forall$ a $\in$ A
  $$\alpha(c) \sqsubseteq a \text{ iff } c \sqsubseteq \gamma(a)$$
- α and γ uniquely determine each other

The Abstraction Function (CP)

- Map collecting states into constants
- The abstraction of an individual state

  $\beta_{CP}:[Var_* \rightarrow Z] \rightarrow [Var_* \rightarrow Z \cup \{\bot, \top\}]$

  $\beta_{CP}(\sigma) = \sigma$
- The abstraction of set of states

  $\alpha_{CP}:P([Var_* \rightarrow Z]) \rightarrow [Var_* \rightarrow Z \cup \{\bot, \top\}]$

  $\alpha_{CP}(CS) = \sqcup \{ \beta_{CP}(\sigma) \mid \sigma \in CS\} = \sqcup\{\sigma \mid \sigma \in CS\}$
- Soundness

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

The Concretization Function

- Map constants into collecting states
- The formal meaning of constants
- The concretization

  $\gamma_{CP}: [Var_* \to Z \cup \{\perp, \top\}] \to P([Var_* \to Z])$

  $\gamma_{CP}(df) = \{\sigma | \beta_{CP}(\sigma) \sqsubseteq df\} = \{\sigma \mid \sigma \sqsubseteq df\}$
- Soundness

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

Galois Connection Constant Propagation

- α_{CP} is monotone
- γ_{CP} is monotone
- $\forall$ df $\in$ [$Var_* \rightarrow Z \cup \{\perp, \top\}$]
  - $\alpha_{CP}(\gamma_{CP}(df)) \sqsubseteq df$
- $\forall$ c $\in$ P([$Var_* \rightarrow Z$])
  - $c_{CP} \sqsubseteq \gamma_{CP}(\alpha_{CP}(C))$

Upper Closures

- Define abstractions on sets of concrete states
- $\uparrow: P(\Sigma) \rightarrow P(\Sigma)$ such that
  - $\uparrow$ is monotone, i.e., $X \subseteq Y \rightarrow \uparrow X \subseteq \uparrow Y$
  - $\uparrow$ is extensive, i.e., $\uparrow X \supseteq X$
  - $\uparrow$ is closure, i.e., $\uparrow(\uparrow X) = \uparrow X$
- Every Galois connection defines an upper closure

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 correctness) Conclude that the analysis is sound

Collecting Semantics

- The input state is not known at compile-time
- "Collect" all the states for all possible inputs to the program
- No lost of precision

A Simple Example Program

$\{[x\mapsto 0, y\mapsto 0, z\mapsto 0]\}$

z = 3 $\{[x\mapsto 0, y\mapsto 0, z\mapsto 3]\}$

x = 1 $\{[x\mapsto 1, y\mapsto 0, z\mapsto 3]\}$

while (x > 0) ($\{[x\mapsto 1, y\mapsto 0, z\mapsto 3], [x\mapsto 3, y\mapsto 0, z\mapsto 3],\}$

if (x = 1) then y = 7 $\{[x\mapsto 1, y\mapsto 7, z\mapsto 3], [x\mapsto 3, y\mapsto 7, z\mapsto 3]\}$

else y = z + 4

$\{[x\mapsto 1, y\mapsto 7, z\mapsto 3], [x\mapsto 3, y\mapsto 7, z\mapsto 3]\}$

x = 3

$\{[x\mapsto 3, y\mapsto 7, z\mapsto 3]\}$

print y

$\{[x\mapsto 3, y\mapsto 7, z\mapsto 3]\}$

)

Another Example

```
x= 0
while (true) do
   x = x +1
```

An “Iterative” Definition

- Generate a system of monotone equations
- 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]

    $CS_{exit}(1) = CS_{entry}(l)$
  - [b]

    $CS_{exit}(1) = \{\sigma \colon \sigma \in CS_{entry}(l), [\![b]\!]\sigma = tt\}$
  - [x := a]

    $CS_{exit}(1) = \{(s[x \mapsto \mathbf{A}[\![a]\!]s]) \mid s \in CS_{entry}(l)\}$
- Equations for control flow constructs

  $CS_{entry}(l) = \ \cup CS_{exit}(l')$ l' immediately precedes l in the control flow graph
- An equation for the entry

  $CS_{entry}(1) = \{\sigma \mid \sigma \in Var_{*} \rightarrow Z\}$

Specialized Chaotic Iterations System of Equations (Collecting Semantics)

S =

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

$$f(e) = \lambda X. \{\sigma \mid [\![b(e)]\!]\,\sigma = tt\}$$

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

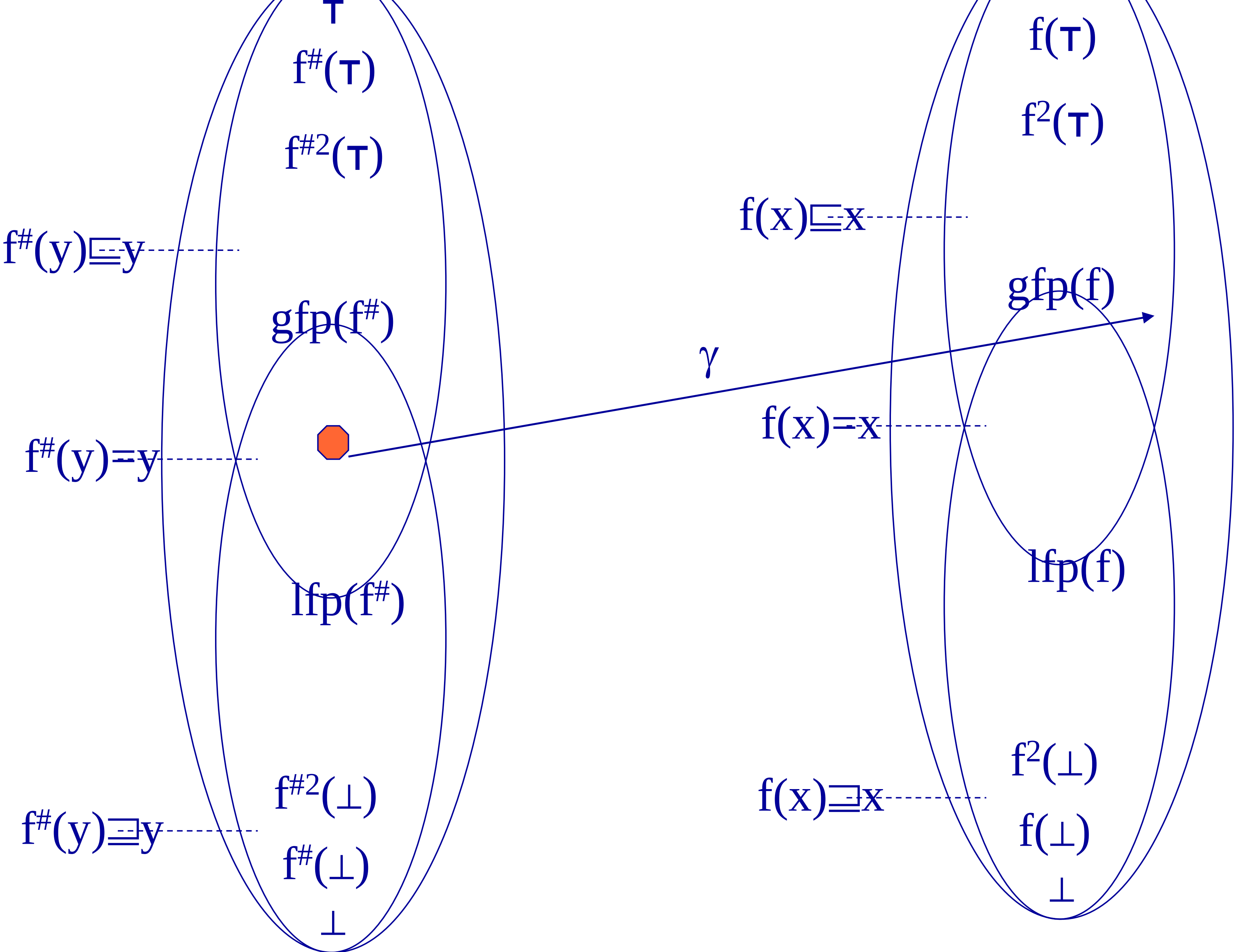
$$F_s(X)[v] = \cup\{f(e)[u] \mid (u, v) \in E\}$$

$$lfp(S) = lfp(F_S)$$

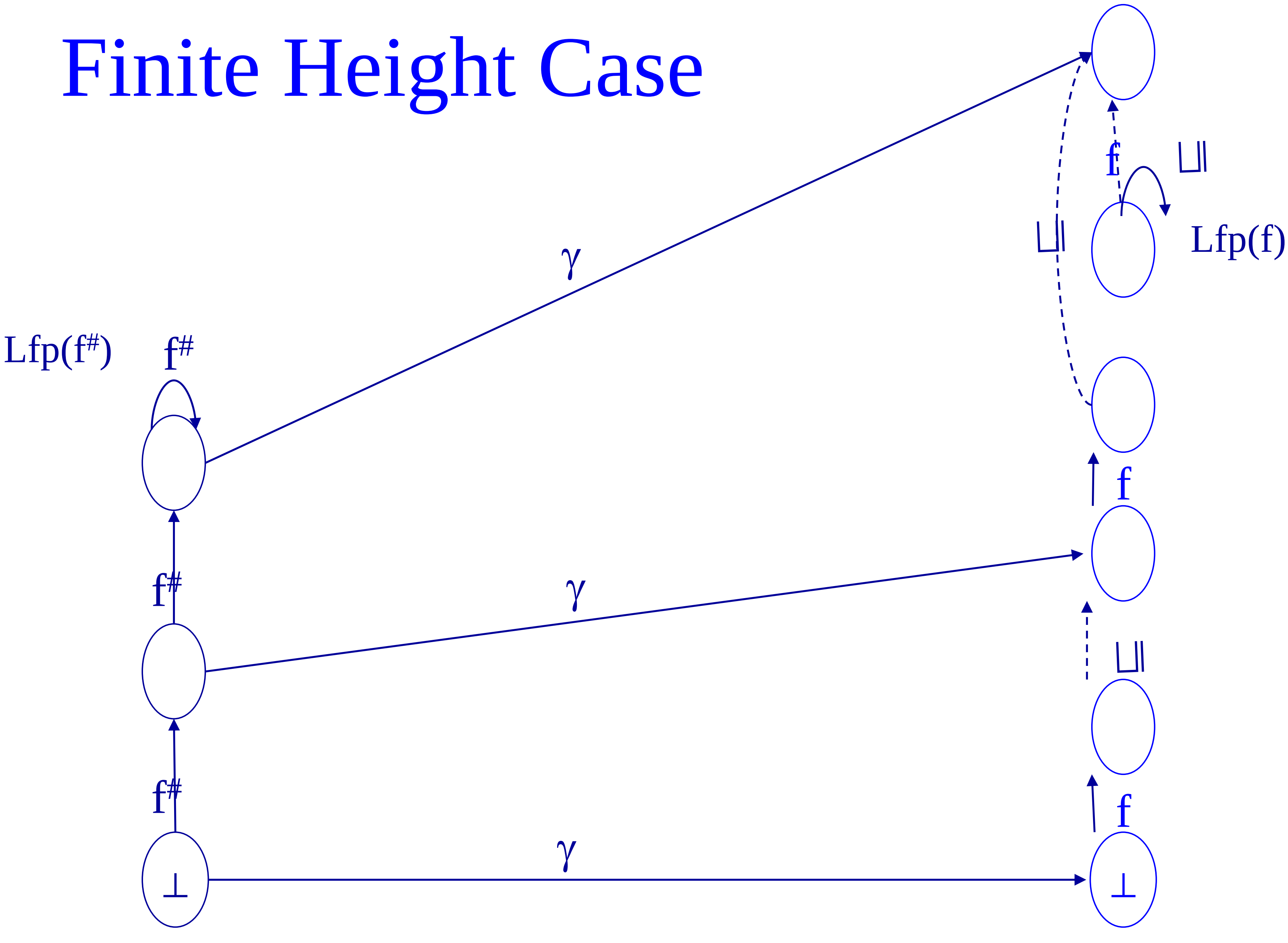
The Least Solution

- 2n sets of equations
  $CS_{entry}(1), \ldots, CS_{entry}(n), CS_{exit}(1), \ldots, CS_{exit}(n)$
- Can be written in vectorial form
- The least solution $lfp(F_{cs})$ is well-defined
- Every component is minimal
- Since F_{cs} is monotone such a solution always exists
- $CS_{entry}(v) = \{s \mid \exists s_0 \mid \langle P, s_0 \rangle \Rightarrow^{*} (S', s)),$
  $\mathit{init}(S')=v\}$
- Simplify the soundness criteria

$$\forall a: f(\gamma(a)) \sqsubseteq \gamma(f^{\#}(a))$$



Finite Height Case



Soundness Theorem(1)

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

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

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

Soundness Theorem(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. $\forall c \in C: \alpha(f(c)) \sqsubseteq f^{\#}(\alpha(c))$

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^{\#})$$

$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^{\#}))$$

Soundness Theorem(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. $\forall a \in A: \alpha(f(\gamma(a))) \sqsubseteq f^{\#}(a)$

$$\alpha(\mathrm{lfp}(f)) \sqsubseteq \mathrm{lfp}(f^{\#})$$

$$\mathrm{lfp}(f) \sqsubseteq \gamma(\mathrm{lfp}(f^{\#}))$$

Proof of Soundness (Summary)

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

Completeness

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

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

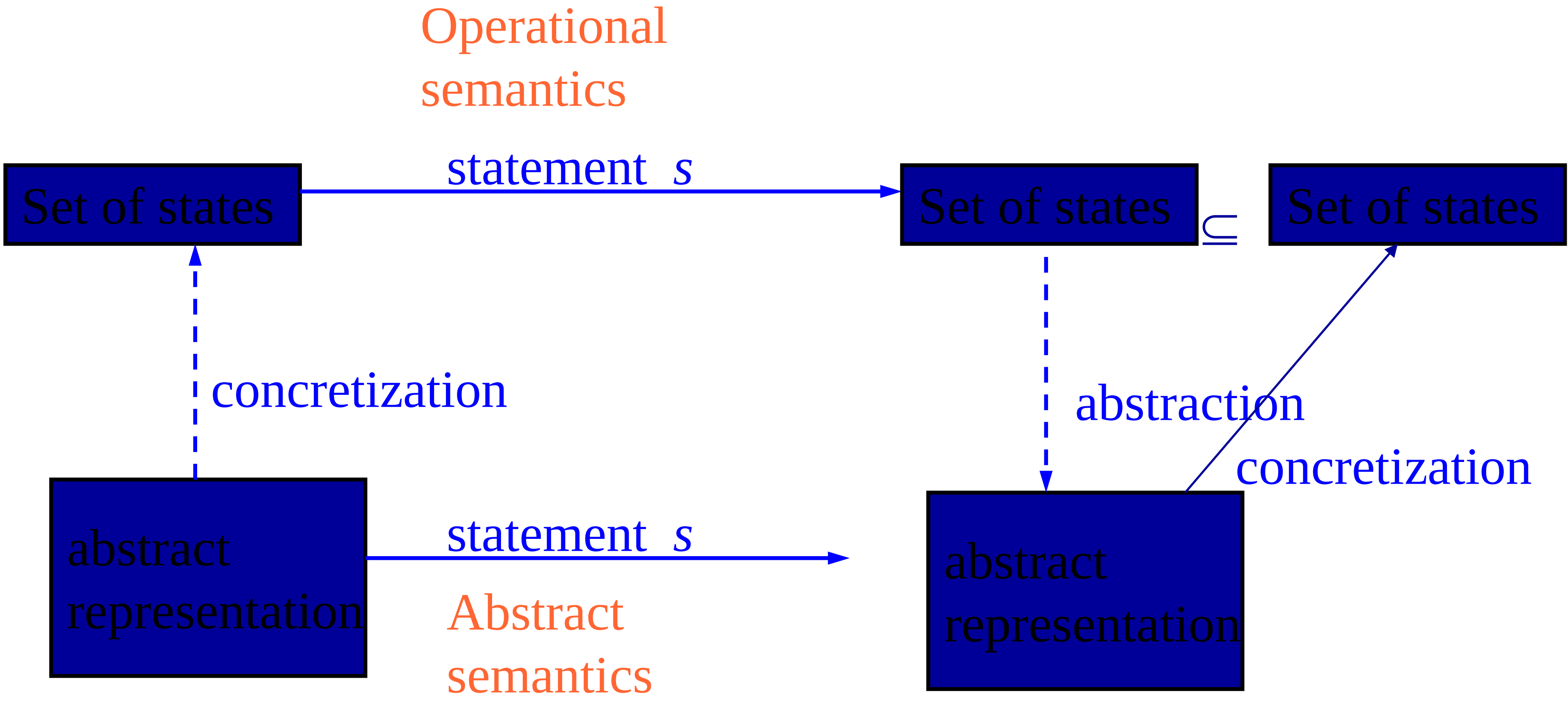
Constant Propagation

- β: [Var $\rightarrow$ Z] $\rightarrow$ [Var $\rightarrow$ Z$\cup${$\top$, $\bot$}]
  - $\beta(\sigma) = (\sigma)$
- α: P([Var $\rightarrow$ Z]) $\rightarrow$ [Var $\rightarrow$ Z$\cup${$\top$, $\bot$}]
  - $\alpha(X) = \sqcup \{\beta(\sigma) \mid \sigma \in X\} = \sqcup \{\sigma \mid \sigma \in X\}$
- γ:[Var $\rightarrow$ Z $\cup${$\top$, $\bot$}] $\rightarrow$ P([Var $\rightarrow$ Z])
  - $\gamma(\sigma^\#) = \{\sigma \mid \beta(\sigma) \sqsubseteq \sigma^\# \} = \{\sigma \mid \sigma \sqsubseteq \sigma^\# \}$
- Local Soundness
  - $[\![st]\!]^\#(\sigma^\#) \sqsupseteq \alpha(\{ [\![st]\!]\, \sigma \ \mid \sigma \in \gamma(\sigma^\#) = \sqcup \{ [\![st]\!]\, \sigma \mid \sigma \sqsubseteq \sigma^\# \}$
- Optimality (Induced)
  - $[\![st]\!]^\#(\sigma^\#) = \alpha(\{ [\![st]\!]\, \sigma \mid \sigma \in\gamma (\sigma^\#)\} = \sqcup \{ [\![st]\!]\, \sigma \mid \sigma \sqsubseteq \sigma^\# \}$
- Soundness
- Completeness

Proof of Soundness (Summary)

- Define an “appropriate” structural operational semantics
- Define “collecting” structural operational semantics
- Establish a Galois connection between collecting states and reaching definitions
- (Local correctness) Show that the abstract interpretation of every atomic statement is sound w.r.t. the collecting semantics
- (Global correctness) Conclude that the analysis is sound

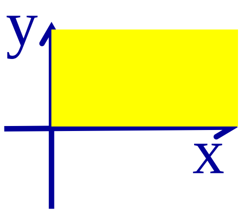
Best (Conservative) interpretation



Induced Analysis (Relatively Optimal)

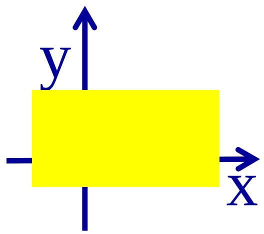
- It is sometimes possible to show that a given analysis is not only sound but optimal w.r.t. the chosen abstraction
  - but not necessarily optimal!
- Define
  $[\![S]\!]^{\#}(df) = \alpha(\{[\![S]\!]\sigma \mid \sigma \in \gamma(df)\})$
- But this $[\![S]\!]^{\#}$ may not be computable
- Derive (at compiler-generation time) an alternative form for $[\![S]\!]^{\#}$
- A useful measure to decide if the abstraction must lead to overly imprecise results

Numeric Abstract Domain Examples



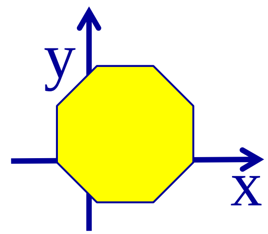
signs

$x \geq 0$



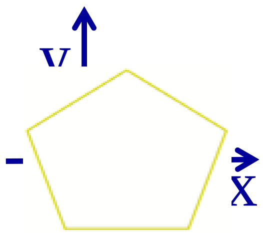
intervals

$x \in [a, b]$



octagons

$\pm x \pm y \leq c$



polyhedra

$\Sigma a_i x_i \leq c$

Example Dataflow Problem

- Formal available expression analysis
- Find out which expressions are available at a given program point
- Example program

```
x = y + t
z = y + r
while (...) {
        t = t + (y + r)
                }
```

- Lattice
- Galois connection
- Basic statements
- Soundness

Example: May-Be-Garbage

- A variable x may-be-garbage at a program point v if there exists a execution path leading to v in which x's value is unpredictable:
  - Was not assigned
  - Was assigned using an unpredictable expression
- Lattice
- Galois connection
- Basic statements
- Soundness

Points-To Analysis

- Determine if a pointer variable p may point to q
  on some path leading to a program point
- "Adapt" other optimizations
  - Constant propagation
    x = 5;
    *p = 7 ;
    … x …
- Pointer aliases
  - Variables p and q are may-aliases at v if the points-to set at v contains entries (p, x) and (q, x)
- Side-effect analysis
  *p = *q + * * t

The **PWhile** Programming Language
Abstract Syntax

$a := x \mid *x \mid \&x \mid n \mid a_1\ op_a\ a_2$

$b :=$ true | false | not b | $b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2$

$S := x := a \mid *x := a \mid$ skip | S_1 ; S_2 |
if b **then** S_1 **else** S_2 | **while** b do S

Concrete Semantics 1 for PWhile

State1= [Loc→Loc∪Z]

For every atomic statement S

$$[\![S]\!] : \text{States1} \rightarrow \text{States1}$$

$$[\![x := a]\!](\sigma)=\sigma[loc(x) \mapsto \mathbf{A}[\![a]\!]\ \sigma]$$

$$[\![x := \&y]\!](\sigma)$$

$$[\![x := *y]\!](\sigma)$$

$$[\![x := y]\!](\sigma)$$

$$[\![*x := y]\!](\sigma)$$

Points-To Analysis

- Lattice L_{pt} =
- Galois connection

Abstract Semantics for PWhile

- For every atomic statement S

$[\![S]\!]^{\#}: P(Var^* \times Var^*) \rightarrow P(Var^* \times Var^*)$

$[\![x := \&y]\!]^{\#}$

$[\![x := *y]\!]^{\#}$

$[\![x := y]\!]^{\#}$

$[\![*x := y]\!]^{\#}$

```
t := &a;
y := &b;

z := &c;

 if x> 0;
         then p:= &y;
         else p:= &z;

 *p := t;
```

```
/*  ∅ */ t := &a; /* {(t, a)}*/
 /* {(t, a)}*/  y := &b; /* {(t, a), (y, b) }*/

/* {(t, a), (y, b)}*/ z := &c; /* {(t, a), (y, b), (z, c) }*/

 if x> 0;
         then p:= &y; /* {(t, a),  (y, b), (z, c), (p, y)}*/


         else p:= &z; /* {(t, a),  (y, b), (z, c), (p, z)}*/
/* {(t, a),  (y, b), (z, c), (p, y), (p, z)}*/

 *p := t;

/* {(t, a), (y, b), (y, c), (p, y), (p, z), (y, a), (z, a)}*/
```

Flow insensitive points-to-analysis Steengard 1996

- Ignore control flow
- One set of points-to per program
- Can be represented as a directed graph
- Conservative approximation
  - Accumulate pointers
- Can be computed in almost linear time

```
t := &a;
y := &b;

z := &c;

 if x> 0;
         then p:= &y;
         else p:= &z;

 *p := t;
```

Precision

- We cannot usually have
  - $\alpha(CS) = DF$
    on all programs
- But can we say something about precision in all programs?
- Precision criteria
  - Join over all paths
  - Induced analysis

Summary

- Abstract interpretation Connects Abstract and Concrete Semantics
- Galois Connection
- Local Correctness
- Global Correctness

Widening

- Accelerate the termination of Chaotic iterations by computing a more conservative solution
- Can handle lattices of infinite heights

Specialized Chaotic Iterations+ ∇

Chaotic(G(V, E): Graph, s: Node, L: lattice, ι: L, f: E $\rightarrow$(L $\rightarrow$L)){

for each v in V to n do $df_{entry}[v] := \perp$

$In[v] = \iota$

WL = {s}

while (WL $\neq$ $\emptyset$) do

select and remove an element u $\in$ WL

for each v, such that. (u, v) $\in$E do

temp = $f(e)(df_{entry}[u])$

new := $df_{entry}(v) \nabla$ temp

if (new $\neq$ $df_{entry}[v]$) then

$df_{entry}[v]$:= new;

WL := WL $\cup$\{v\}

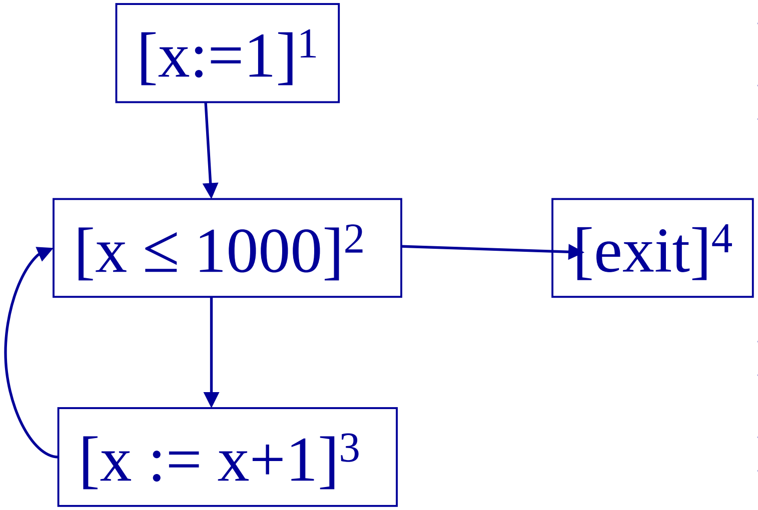
Example Interval Analysis

- Find a lower and an upper bound of the value of a variable
- Usages?
- Lattice

  $L = (Z \cup \{-\infty, \infty\} \times Z \cup \{-\infty, \infty\}, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$
  - $[a, b] \sqsubseteq [c, d]$ if $c \leq a$ and $d \geq b$
  - $[a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]$
  - $[a, b] \sqcap [c, d] = [\max(a, c), \min(b, d)]$
  - $\top =$
  - $\perp =$

Example Program
Interval Analysis

$[x := 1]^1$;
while $[x \leq 1000]^2$ do
  $[x := x + 1;]^3$



$\mathrm{IntEntry}(1) = [\mathrm{minint},\mathrm{maxint}]$

$\mathrm{IntExit}(1) = [1,1]$

$\mathrm{IntEntry}(2) = \mathrm{IntExit}(1) \sqcup \mathrm{IntExit}(3)$

$\mathrm{IntExit}(2) = \mathrm{IntEntry}(2)$

$\mathrm{IntEntry}(3) = \mathrm{IntExit}(2) \sqcap [\mathrm{minint},1000]$

$\mathrm{IntExit}(3) = \mathrm{IntEntry}(3)+[1,1]$

$\mathrm{IntEntry}(4) = \mathrm{IntExit}(2) \sqcap [1001,\mathrm{maxint}]$

$\mathrm{IntExit}(4) = \mathrm{IntEntry}(4)$

Widening for Interval Analysis

- $\perp \nabla$ [c, d] = [c, d]
- [a, b] ∇ [c, d] = [

    if $a \leq c$

        then a

        else $-\infty$,

    if $b \geq d$

        then b

        else ∞

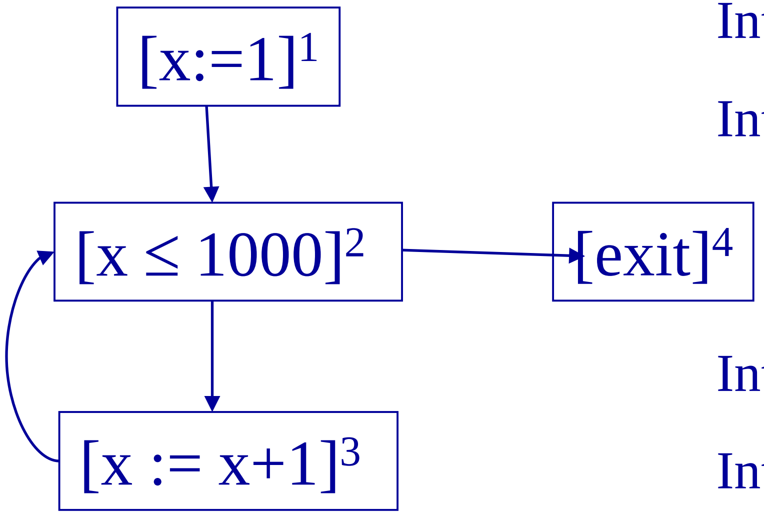
    ]

Example Program
Interval Analysis

$[x := 1]^1$;

while $[x \leq 1000]^2$ do

$\quad [x := x + 1;]^3$



$\text{IntEntry}(1) = [-\infty, \infty]$

$\text{IntExit}(1) = [1,1]$

$\text{IntEntry}(2) = \text{InExit}(2) \bigtriangledown (\text{IntExit}(1) \sqcup \text{IntExit}(3))$

$\text{IntExit}(2) = \text{IntEntry}(2)$

$\text{IntEntry}(3) = \text{IntExit}(2) \sqcap [-\infty,1000]$

$\text{IntExit}(3) = \text{IntEntry}(3)+[1,1]$

$\text{IntEntry}(4) = \text{IntExit}(2) \sqcap [1001, \infty]$

$\text{IntExit}(4) = \text{IntEntry}(4)$

Requirements on Widening

- For all elements $l_1 \sqcup l_2 \sqsubseteq l_1 \nabla l_2$
- For all ascending chains
  $l_0 \sqsubseteq l_1 \sqsubseteq l_2 \sqsubseteq \ldots$
  the following sequence is finite
  - $y_0 = l_0$
  - $y_{i+1} = y_i \nabla l_{i+1}$
- For a monotonic function
  $f: L \rightarrow L$
  define
  - $x_0 = \bot$
  - $x_{i+1} = x_i \nabla f(x_i)$
- Theorem:
  - There exits k such that $x_{k+1} = x_k$
  - $x_k \in Red(f) = \{l: l \in L, f(l) \sqsubseteq l\}$

Narrowing

- Improve the result of widening
- $y \sqsubseteq x \Rightarrow y \sqsubseteq (x \triangle y) \sqsubseteq x$
- For all decreasing chains $x_0 \sqsupseteq x_1 \sqsupseteq \ldots$ the following sequence is finite
  - $y_0 = x_0$
  - $y_{i+1} = y_i \triangle x_{i+1}$
- For a monotonic function $f: L \rightarrow L$ and $x \in Red(f) = \{l: l \in L, f(l) \sqsubseteq l\}$ define
  - $y_0 = x$
  - $y_{i+1} = y_i \triangle f(y_i)$
- Theorem:
  - There exits k such that $y_{k+1} = y_k$
  - $y_k \in Red(f) = \{l: l \in L, f(l) \sqsubseteq l\}$

Narrowing for Interval Analysis

- $[a, b] \triangle \perp = [a, b]$
- $[a, b] \triangle [c, d] = [$

  if $a = -\infty$

  then c

  else a,

  if $b = \infty$

  then d

  else b

  $]$

Example Program
Interval Analysis

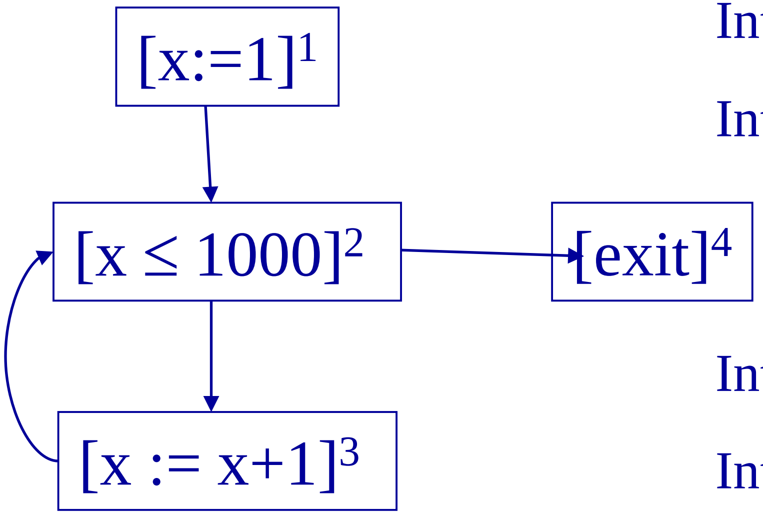
$[x := 1]^1$;
while $[x \leq 1000]^2$ do
    $[x := x + 1;]^3$

IntEntry(1) = $[-\infty , \infty]$

IntExit(1) = $[1,1]$

IntEntry(2) = InExit(2) $\triangle$(IntExit(1) $\sqcup$ IntExit(3))

IntExit(2) = IntEntry(2)



IntEntry(3) = IntExit(2) $\sqcap$ $[-\infty,1000]$

IntExit(3) = IntEntry(3)+$[1,1]$

IntEntry(4) = IntExit(2) $\sqcap$ $[1001, \infty]$

IntExit(4) = IntEntry(4)

Non Montonicity of Widening

- $[0,1] \triangledown [0,2] = [0, \infty]$
- $[0,2] \triangledown [0,2] = [0,2]$

Widening and Narrowing Summary

- Very simple but produces impressive precision
- Sometimes non-monotonic
- The McCarthy 91 function

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

- Also useful in the finite case
- Can be used as a methodological tool

Conclusions

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- But expensive
  - More efficient methods exist for structured programs