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 $(L, \sqsubseteq, \sqcup, \forall, \bot, \top)$ is a complete lattice
- $\text{Fix}(f) = \{ l: l \in L, f(l) = l \}$
- $\text{Red}(f) = \{ l: l \in L, f(l) \sqsubseteq l \}$
- $\text{Ext}(f) = \{ l: l \in L, l \sqsubseteq f(l) \}$
  - $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$
- Tarski’s Theorem 1955: if $f$ is monotone then:
  - $\text{lfp}(f) = \sqcap \text{Fix}(f) = \sqcap \text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \sqcup \text{Fix}(f) = \sqcup \text{Ext}(f) \in \text{Fix}(f)$
Chaotic Iterations

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

```plaintext
for i := 1 to n do
    x[i] = \bot

WL = \{ 1, 2, \ldots, n \}

x := (\bot, \bot, \ldots, \bot) while (WL \neq \emptyset) do
    select and remove an element $i \in WL$
    new := $f_i(x)$
    if (new \neq x[i]) then
        x[i] := new;
    Add all the indexes that directly depends on $i$ to WL
```

while ($f(x) \neq x$) do
Specialized Chaotic Iterations
System of Equations

\[ S = \]
\[ \begin{cases} 
\text{df}_{\text{entry}}[s] = \tau \\
\text{df}_{\text{entry}}[v] = \bigsqcup \{ f(u, v) \left( \text{df}_{\text{entry}}[u] \right) \mid (u, v) \in E \} 
\end{cases} \]

\[ F_S : L^n \rightarrow L^n \]

\[ F_S (X)[s] = \tau \]

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

\[ \text{lfp}(S) = \text{lfp}(F_S) \]
Specialized Chaotic Iterations

Chaotic(G(V, E): Graph, s: Node, L: Lattice, \( \tau: L \rightarrow (L \rightarrow L) \)){
    for each \( v \) in V to n do \( df_{entry}[v] := \perp \)
    \( df[s] = \tau \)
    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 := \( df_{entry}(v) \parallel temp \)
            if (new \neq df_{entry}[v]) then
                \( df_{entry}[v] := new; \)
            WL := WL \cup \{v\}
Specialized Chaotic Iterations
System of Equations

\[ S = \begin{cases} 
\text{df}_{\text{entry}}[s] = \tau \\
\text{df}_{\text{entry}}[v] = \bigsqcup \{ f(u, v) (\text{df}_{\text{entry}}[u]) \mid (u, v) \in E \} 
\end{cases} \]

\[ F_S : \mathbb{L}^n \rightarrow \mathbb{L}^n \]

\[ F_S(X)[s] = \tau \]

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

\[ \text{lfp}(S) = \text{lfp}(F_S) \]
Specialized Chaotic Iterations

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

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

\[
df[s] = i
\]

\[
WL = \{s\}
\]

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

select and remove an element \(u \in WL\)

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

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

\[
\text{new} := df_{\text{entry}}(v) \sqcup temp
\]

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

\[
df_{\text{entry}}[v] := \text{new};
\]

\[
WL := WL \cup \{v\}
\]
\[z = 3\]
\[x = 1\]
\[\text{while } (x > 0)\]
\[\text{if } (x = 1)\]
\[y = 7\]
\[y = z + 4\]
\[\text{print } y\]
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

Operational semantics

Set of states

abstraction

abstract representation

statement \( s \)

Set of states

abstraction

abstract representation

Abstract semantics

statement \( s \)

abstract representation
Abstract (Conservative) interpretation

Set of states \( \subseteq \) Set of states

abstract representation \( \supseteq \) abstract representation

Operational semantics

concretization

Abstract semantics

statement \( s \)
Abstract Interpretation

Concrete
Sets of stores

Abstract
Descriptors of sets of stores
Galois Connections

- Lattices C and A and functions \( \alpha : C \to A \) and \( \gamma : A \to C \)

- The pair of functions \( (\alpha, \gamma) \) form a Galois connection if
  - \( \alpha \) and \( \gamma \) are monotone
  - \( \forall a \in A \) \( \implies \alpha(\gamma(a)) \subseteq a \)
  - \( \forall c \in C \) \( \implies c \subseteq \gamma(\alpha(C)) \)

- Alternatively if:
  \( \forall c \in C \)
  \( \forall a \in A \)
  \( \alpha(c) \subseteq a \) iff \( c \subseteq \gamma(a) \)

- \( \alpha \) and \( \gamma \) uniquely determine each other
The Abstraction Function (CP)

- Map collecting states into constants
- The abstraction of an individual state
  \[ \beta_{CP} : [\text{Var}_* \rightarrow Z] \rightarrow [\text{Var}_* \rightarrow Z \cup \{\bot, \top\}] \]
  \[ \beta_{CP}(\sigma) = \sigma \]
- The abstraction of set of states
  \[ \alpha_{CP} : \text{P}([\text{Var}_* \rightarrow Z]) \rightarrow [\text{Var}_* \rightarrow Z \cup \{\bot, \top\}] \]
  \[ \alpha_{CP}(CS) = \bigcup \{ \beta_{CP}(\sigma) \mid \sigma \in CS \} = \bigcup \{\sigma \mid \sigma \in CS\} \]
- Soundness
  \[ \alpha_{CP}(\text{Reach}(v)) \subseteq \text{df}(v) \]
- Completeness
The Concretization Function

- Map constants into collecting states
- The formal meaning of constants
- The concretization
  \[ \gamma_{CP} : [\text{Var}_* \rightarrow \mathbb{Z} \cup \{\bot, \top\}] \rightarrow \mathcal{P}([\text{Var}_* \rightarrow \mathbb{Z}]) \]
  \[ \gamma_{CP}(df) = \{ \sigma | \beta_{CP}(\sigma) \subseteq df \} = \{ \sigma | \sigma \subseteq df \} \]
- Soundness
  Reach (v) \subseteq \gamma_{CP}(df(v))
- Completeness
Galois Connection Constant Propagation

- $\alpha_{CP}$ is monotone
- $\gamma_{CP}$ is monotone
- $\forall \ df \in [Var_\ast \rightarrow Z \cup \{\bot, \top\}]$
  - $\alpha_{CP}(\gamma_{CP}(df)) \subseteq df$
- $\forall \ c \in P([Var_\ast \rightarrow Z])$
  - $c_{CP} \subseteq \gamma_{CP}(\alpha_{CP}(C))$
Upper Closures

- Define abstractions on sets of concrete states
- \[ \uparrow: \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma) \] such that
  - \[ \uparrow \text{ is monotone, i.e., } X \subseteq Y \to \uparrow X \subseteq \uparrow Y \]
  - \[ \uparrow \text{ is extensive, i.e., } \uparrow X \supseteq X \]
  - \[ \uparrow \text{ is closure, i.e., } \uparrow(\uparrow X) = \uparrow X \]
- 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]\}\n
\begin{align*}
z &= 3 \\
\{[x \mapsto 0, y \mapsto 0, z \mapsto 3]\} \\
x &= 1 \\
\{[x \mapsto 1, y \mapsto 0, z \mapsto 3]\} \quad \text{while (}x > 0\text{)} ( \\
\quad \{[x \mapsto 1, y \mapsto 0, z \mapsto 3], [x \mapsto 3, y \mapsto 0, z \mapsto 3]\}, \\
\quad \text{if (}x = 1\text{) then } y = 7 \\
\quad \{[x \mapsto 1, y \mapsto 7, z \mapsto 3], [x \mapsto 3, y \mapsto 7, z \mapsto 3]\} \\
\quad \text{else } y = z + 4 \\
\quad \{[x \mapsto 1, y \mapsto 7, z \mapsto 3], [x \mapsto 3, y \mapsto 7, z \mapsto 3]\} \\
x &= 3 \\
\{[x \mapsto 1, y \mapsto 7, z \mapsto 3], [x \mapsto 3, y \mapsto 7, z \mapsto 3]\} \\
\text{print } y \\
\{[x \mapsto 3, y \mapsto 7, z \mapsto 3]\} \quad )
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: \sigma \in CS_{entry}(l), \[b]\sigma=tt \} \]
  - [x := a]
    \[ CS_{exit}(1) = \{ (s[x \mapsto A[a][s]]) | s \in CS_{entry}(l) \} \]

◆ Equations for control flow constructs
  \[ CS_{entry}(l) = \bigcup CS_{exit}(l') \]
  where \( l' \) immediately precedes \( l \) in the control flow graph

◆ An equation for the entry
  \[ CS_{entry}(1) = \{ \sigma | \sigma \in Var \ast \rightarrow Z \} \]
Specialized Chaotic Iterations System of Equations (Collecting Semantics)

\[ S = \]

\[
\begin{align*}
\text{CS}_{\text{entry}}[s] &= \{\sigma_0\} \\
\text{CS}_{\text{entry}}[v] &= \bigcup \{ f(e)(\text{CS}_{\text{entry}}[u]) \mid (u, v) \in E \}
\end{align*}
\]

where \( f(e) = \lambda X. \{ \sem{\text{st}(e)} \sigma \mid \sigma \in X \} \) for atomic statements

\[
f(e) = \lambda X. \{ \sigma \mid \sem{\text{b}(e)} \sigma = \text{tt} \}
\]

\[ F_S : L^n \rightarrow L^n \]

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

\[ \text{lfp}(S) = \text{lfp}(F_S) \]
The Least Solution

- 2n sets of equations
  \[ \text{CS}_{\text{entry}}(1), \ldots, \text{CS}_{\text{entry}}(n), \text{CS}_{\text{exit}}(1), \ldots, \text{CS}_{\text{exit}}(n) \]

- Can be written in vectorial form

- The least solution \( \text{lfp}(F_{cs}) \) is well-defined

- Every component is minimal

- Since \( F_{cs} \) is monotone such a solution always exists

\[ \text{CS}_{\text{entry}}(v) = \{ s | \exists s_0 \ such \ that \ <P, s_0> \Rightarrow^* (S', s), \ init(S') = v \} \]

- Simplify the soundness criteria
∀a: f(γ(a)) ⊆ γ(f#(a))
Finite Height Case
**Soundness Theorem (1)**

1. Let \((\alpha, \gamma)\) 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. \(\forall a \in A: f(\gamma(a)) \sqsubseteq \gamma(f^\#(a))\)

\[
\text{lfp}(f) \sqsubseteq \gamma(\text{lfp}(f^\#))
\]

\[
\alpha(\text{lfp}(f)) \sqsubseteq \text{lfp}(f^\#)
\]
Soundness Theorem (2)

1. Let \((\alpha, \gamma)\) 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. \(\forall c \in C: \alpha(f(c)) \subseteq f^\#(\alpha(c))\)

\[\alpha(lfp(f)) \subseteq lfp(f^\#)\]

\[lfp(f) \subseteq \gamma(lfp(f^\#))\]
Soundness Theorem (3)

1. Let \((\alpha, \gamma)\) 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. \(\forall a \in A: \alpha(f(\gamma(a))) \subseteq f^#(a)\)

\[\alpha(\text{lfp}(f)) \subseteq \text{lfp}(f^#)\]

\[\text{lfp}(f) \subseteq \gamma(\text{lfp}(f^#))\]
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

\[ \alpha(\text{lfp}(f)) = \text{lfp}(f^\#) \]

\[ \text{lfp}(f) = \gamma(\text{lfp}(f^\#)) \]
Constant Propagation

- $\beta: [\text{Var} \rightarrow Z] \rightarrow [\text{Var} \rightarrow Z \cup \{\top, \bot\}]$
  - $\beta(\sigma) = (\sigma)$
- $\alpha: \mathcal{P}(\text{[Var} \rightarrow Z]) \rightarrow [\text{Var} \rightarrow Z \cup \{\top, \bot\}]$
  - $\alpha(X) = \bigcup \{ \beta(\sigma) \mid \sigma \in X \} = \bigcup \{ \sigma \mid \sigma \in X \}$
- $\gamma: [\text{Var} \rightarrow Z \cup \{\top, \bot\}] \rightarrow \mathcal{P}(\text{[Var} \rightarrow Z])$
  - $\gamma(\sigma^\#) = \{ \sigma \mid \beta(\sigma) \subseteq \sigma^\# \} = \{ \sigma \mid \sigma \subseteq \sigma^\# \}$

- **Local Soundness**
  - $\llbracket \text{st} \rrbracket^\#(\sigma^\#) \supseteq \alpha(\{ \llbracket \text{st} \rrbracket \sigma \mid \sigma \in \gamma(\sigma^\#) \}) = \bigcup \{ \llbracket \text{st} \rrbracket \sigma \mid \sigma \subseteq \sigma^\# \}$

- **Optimality (Induced)**
  - $\llbracket \text{st} \rrbracket^\#(\sigma^\#) = \alpha(\{ \llbracket \text{st} \rrbracket \sigma \mid \sigma \in \gamma(\sigma^\#) \}) = \bigcup \{ \llbracket \text{st} \rrbracket \sigma \mid \sigma \subseteq \sigma^\# \}$

- **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

Set of states

Abstract representation

Operational semantics

Statement $s$

Concretization

Abstract semantics

Set of states

Set of states

Abstraction

Concretization
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**
  - $\sum a_i x_i \leq c$
Example Dataflow Problem

- Formal available expression analysis
- Find out which expressions are available at a given program point
- Example program
  
  \[
  \begin{align*}
  x &= y + t \\
  z &= y + r \\
  \text{while (…)} \{ \\
  & \quad t = t + (y + r) \\
  \} 
  \end{align*}
  \]

- 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

\[ a := x \mid \ast x \mid \& x \mid n \mid a_1 \text{ op}_a a_2 \]

\[ b := \text{true} \mid \text{false} \mid \text{not} \; b \mid b_1 \text{ op}_b b_2 \mid a_1 \text{ op}_r a_2 \]

\[ S := x := a \mid \ast x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if} \; b \; \text{then} \; S_1 \; \text{else} \; S_2 \mid \text{while} \; b \; \text{do} \; S \]
Concrete Semantics 1 for PWhile

State1 = [Loc → Loc ∪ Z]

For every atomic statement S

\[[S]\] : States1 → States1

\[[x := a]\](\sigma) = \sigma[\text{loc}(x) → 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 $S$

$\left[ S \right] \# : P(\text{Var}* \times \text{Var}*) \rightarrow P(\text{Var}* \times \text{Var}*)$

$\left[ x := \& y \right] \#$

$\left[ x := \ast y \right] \#$

$\left[ x := y \right] \#$

$\left[ \ast x := y \right] \#$
t := &a;
y := &b;
z := &c;

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

*p := t;
```c
/* ∅ */ 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, \( \iota: L \), f: E \( \rightarrow \) (L \( \rightarrow \) L )){
  for each v in V to n do df\_entry[v] := \bot
  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) \( \triangledown \) temp
      if (new \neq df\_entry[v]) then
        df\_entry[v] := new;
        WL := WL \cup \{v\}
  WL := WL \cup \{v\}
Example Interval Analysis

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

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

- \([a, b] \sqsubseteq [c, d]\) if \(c \leq a\) and \(d \geq b\)
- \([a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]\)
- \([a, b] \sqcap [c, d] = [\max(a, c), \min(b, d)]\)
- \(\top = \)
- \(\bot = \)
Example Program

Interval Analysis

\[ x := 1 \]

while \[ x \leq 1000 \] do

\[ x := x + 1; \]

\begin{align*}
\text{IntEntry}(1) &= [\text{minint}, \text{maxint}] \\
\text{IntExit}(1) &= [1,1] \\
\text{IntEntry}(2) &= \text{IntExit}(1) \sqcup \text{IntExit}(3) \\
\text{IntExit}(2) &= \text{IntEntry}(2) \\
\text{IntEntry}(3) &= \text{IntExit}(2) \sqcap [\text{minint},1000] \\
\text{IntExit}(3) &= \text{IntEntry}(3) + [1,1] \\
\text{IntEntry}(4) &= \text{IntExit}(2) \sqcap [1001, \text{maxint}] \\
\text{IntExit}(4) &= \text{IntEntry}(4)
\end{align*}
Widening for Interval Analysis

- $\bot \nabla [c, d] = [c, d]$
- $[a, b] \nabla [c, d] = [\begin{array}{l}
\text{if } a \leq c \\
\hspace{1em} \text{then } a \\
\hspace{1em} \text{else } -\infty,
\end{array} \begin{array}{l}
\text{if } b \geq d \\
\hspace{1em} \text{then } b \\
\hspace{1em} \text{else } \infty
\end{array}]$
Example Program
Interval Analysis

[x := 1]

while [x ≤ 1000]
do
[x := x + 1;]

IntEntry(1) = [-∞, ∞]
IntExit(1) = [1,1]
IntEntry(2) = IntExit(2) ∨ (IntExit(1) ∪ IntExit(3))
IntExit(2) = IntEntry(2)
IntEntry(3) = IntExit(2) ∩ [-∞,1000]
IntExit(3) = IntEntry(3)+[1,1]
IntEntry(4) = IntExit(2) ∩ [1001, ∞]
IntExit(4) = IntEntry(4)
Requirements on Widening

- For all elements \( l_1 \sqcup l_2 \sqsubseteq l_1 \sqcap l_2 \)
- For all ascending chains \( l_0 \sqsubseteq l_1 \sqsubseteq l_2 \sqsubseteq \ldots \)
  the following sequence is finite
  - \( y_0 = l_0 \)
  - \( y_{i+1} = y_i \sqcap l_{i+1} \)
- For a monotonic function \( f: L \rightarrow L \)
  define
  - \( x_0 = \bot \)
  - \( x_{i+1} = x_i \sqcap f(x_i) \)

- Theorem:
  - There exits \( k \) such that \( x_{k+1} = x_k \)
  - \( x_k \in \text{Red}(f) = \{ l : l \in L, f(l) \sqsubseteq l \} \)
Narrowing

◆ Improve the result of widening
◆ $y \subseteq x \Rightarrow y \subseteq (x \triangle y) \subseteq x$
◆ 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 $x \in \text{Red}(f) = \{l: l \in L, f(l) \sqsubseteq l\}$
  define
  - $y_0 = x$
  - $y_{i+1} = y_i \triangle f(y_i)$
◆ Theorem:
  - There exists $k$ such that $y_{k+1} = y_k$
  - $y_k \in \text{Red}(f) = \{l: l \in L, f(l) \sqsubseteq l\}$
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] ;
while [x ≤ 1000] do
  [x := x + 1;]

IntEntry(1) = [-∞, ∞]
IntExit(1) = [1,1]
IntEntry(2) = InExit(2) △ (IntExit(1) ∪ IntExit(3))
IntExit(2) = IntEntry(2)
IntEntry(3) = IntExit(2) ∩ [-∞, 1000]
IntExit(3) = IntEntry(3) + [1,1]
IntEntry(4) = IntExit(2) ∩ [1001, ∞]
IntExit(4) = IntEntry(4)
Non Montonicity of Widening

- $[0,1] \sqcup [0,2] = [0, \infty]$
- $[0,2] \sqcup [0,2] = [0,2]$
Widening and Narrowing

Summary

- Very simple but produces impressive precision
- Sometimes non-monotonic
- The McCarthy 91 function
  \[\text{int } f(x) = \begin{cases} \infty, & \text{if } x > 100 \\ 101, & \text{then } 101, \infty \end{cases} \]
  \[\text{return } x - 10 [91, \infty-10]; \]
  \[\text{else } [-\infty, 100] \text{ return } f(f(x+11)) [91, 91]; \]

- 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