Iterative Program Analysis
Abstract Interpretation

Mooly Sagiv
http://www.cs.tau.ac.il/~msagiv/courses/pa11.html
Tel Aviv University
640-6706

Textbook: Principles of Program Analysis

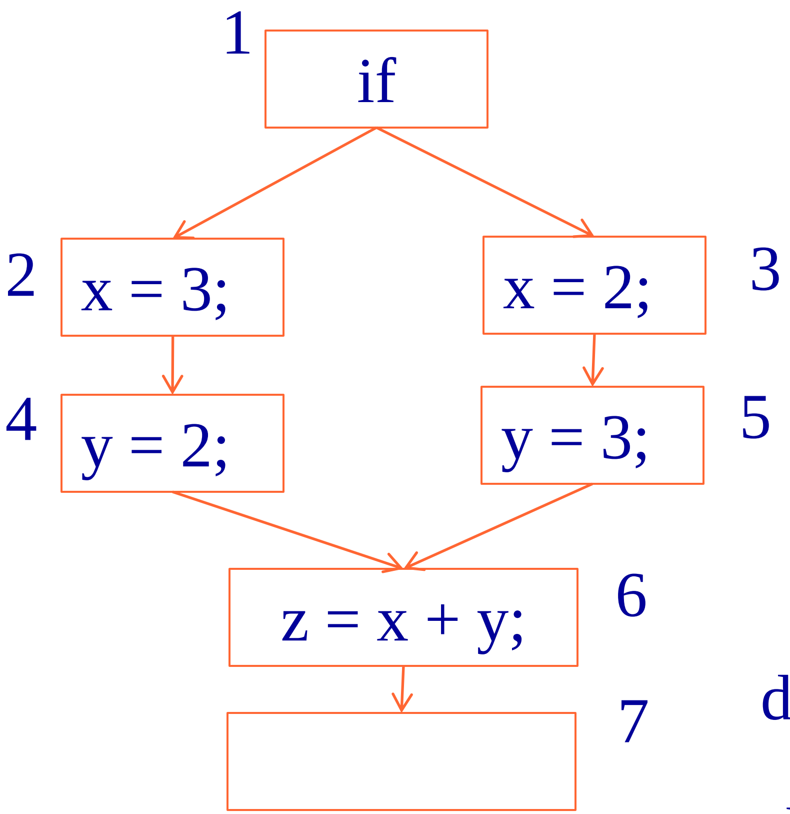
Chapter 4

CC79, CC92

Outline

- The abstract interpretation technique
  - Precision
  - Complexity
  - Widening
  - Combining Analysis
  - Backward Analysis
- Later
  - Interprocedural Analysis
  - Applications
    - String Analysis
    - Shape Analysis
    - Java Safety
    - Device Drivers

Constant Propagation Example



$$df[1] = [x \mapsto \top,\ y \mapsto \top,\ z \mapsto \top]$$

$$df[2] = df[1]$$

$$df[3] = df[1]$$

$$df[4] = df[2]\ [x \mapsto 3]$$

$$df[5] = df[3]\ [x \mapsto 2]$$

$$df[6] = df[4]\ [y \mapsto 2] \sqcup df[5]\ [y \mapsto 3]$$

$$df[7] = df[6]\ [z \mapsto df[6]x +^{\#} df[6]y]$$

Precision

- We cannot usually have
  - $\alpha(CS)$ =df on all programs
- But can we say something about precision in all programs?

The Join-Over-All-Paths (JOP)

- Let paths(v) denote the potentially infinite set paths from start to v (written as sequences of edges)
- For a sequence of edges [e$_1$, e_2, ..., e_n] define $f^\#$[e$_1$, e_2, ..., e_n]: L → L by composing the effects of basic blocks
  $f^\#$[e$_1$, e_2, ..., e_n](l) = $f^\#(e_n)$ (… ($f^\#(e_2)$ ($f^\#(e_1)$ (l)) …)
- JOP[v] = $\sqcup\{f^\#[e_1, e_2, \ldots, e_n](\iota)$
  $[e_1, e_2, \ldots, e_n] \in$ paths(v)$\}$

JOP vs. Least Solution

- The df solution obtained by Chaotic iteration satisfies for every *v*:
  - JOP[v] $\sqsubseteq$ df[*v*]
- A function $f^{\#}$ is additive (distributive) if
  - $f^{\#}(\sqcup\{z \mid z \in X\}) = \sqcup\{f^{\#}(z) \mid z \in X\}$
- If every $f^{\#}_{(u,v)}$ is additive (distributive) for all the edges (u,v)
  - JOP[v] = df[v]
- Examples
  - Maybe garbage
  - Formal Available expressions
  - Constant Propagation
  - Points-to

Notions of precision

- CS = γ (df)
- α(CS) = df
- Meet(Join) over all paths
- Using best(induced) transformers
- Good enough for the client

Complexity of Chaotic Iterations

- Usually depends on the height of the lattice
- In some cases better bound exist
- A function f is fast if $f(f(l)) \sqsubseteq l \sqcup f(l)$
- For fast functions the Chaotic iterations can be implemented in O(nest * |V|) iterations
  - nest is the number of nested loop
  - |V| is the number of control flow nodes
- Examples
  - Maybe garbage
  - Formal Available expressions
  - Constant Propagation
  - Points-to

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 →(L →L)){

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

df[v] = ι

WL = {s}

while (WL ≠ Ø) do

select and remove an element u ∈ WL

for each v, such that. (u, v) ∈E do

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

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

if (new ≠ $df_{entry}[v]$) then

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

WL := WL ∪{v}

Example Interval Analysis

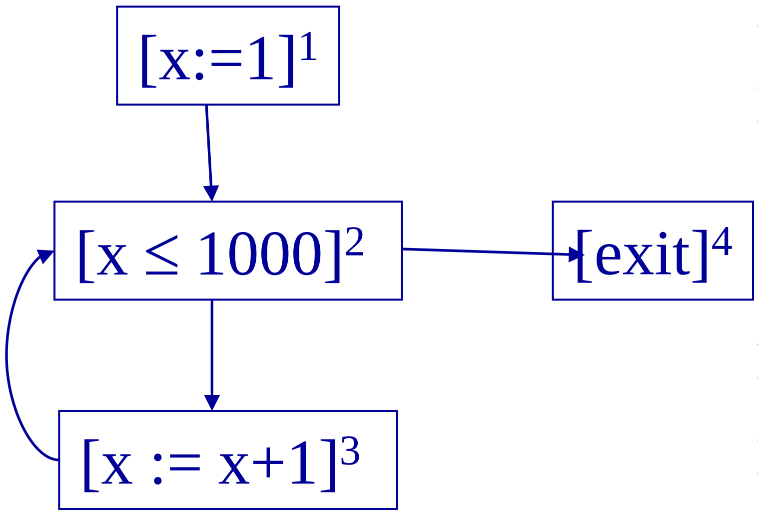
- Find a lower and an upper bound of the value of a variable
- Usages?
- Lattice

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

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

Example Program
Interval Analysis

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



IntEntry(1) = [minint,maxint]

IntExit(1) = [1,1]

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

IntExit(2) = IntEntry(2)

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

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

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

IntExit(4) = 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

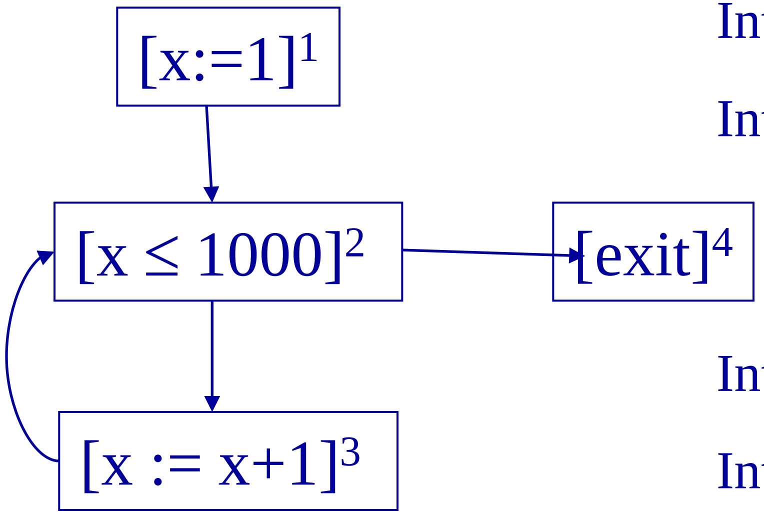
$[x := 1]^1$;
while $[x \leq 1000]^2$ do
    $[x := x + 1;]^3$

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

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

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

$IntExit(2) = IntEntry(2)$



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

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

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

$IntExit(4) = IntEntry(4)$

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

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

  if $a = -\infty$

  then c

  else a,

  if $b = \infty$

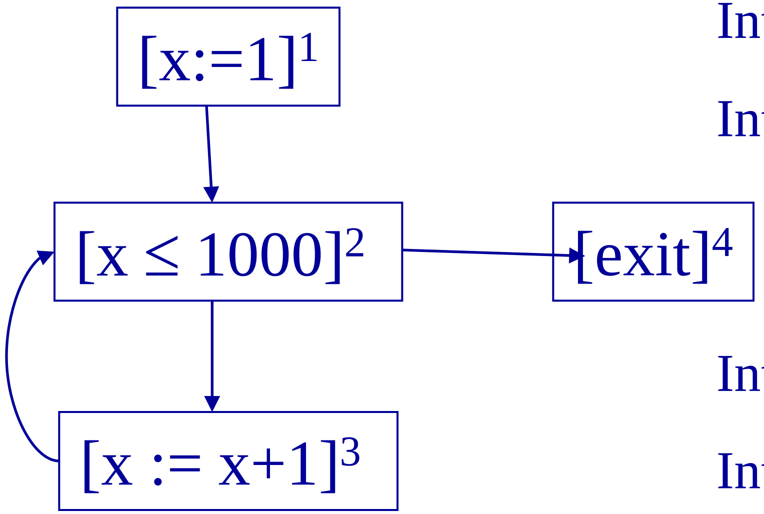
  then d

  else b

  $]$

Example Program
Interval Analysis

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



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]

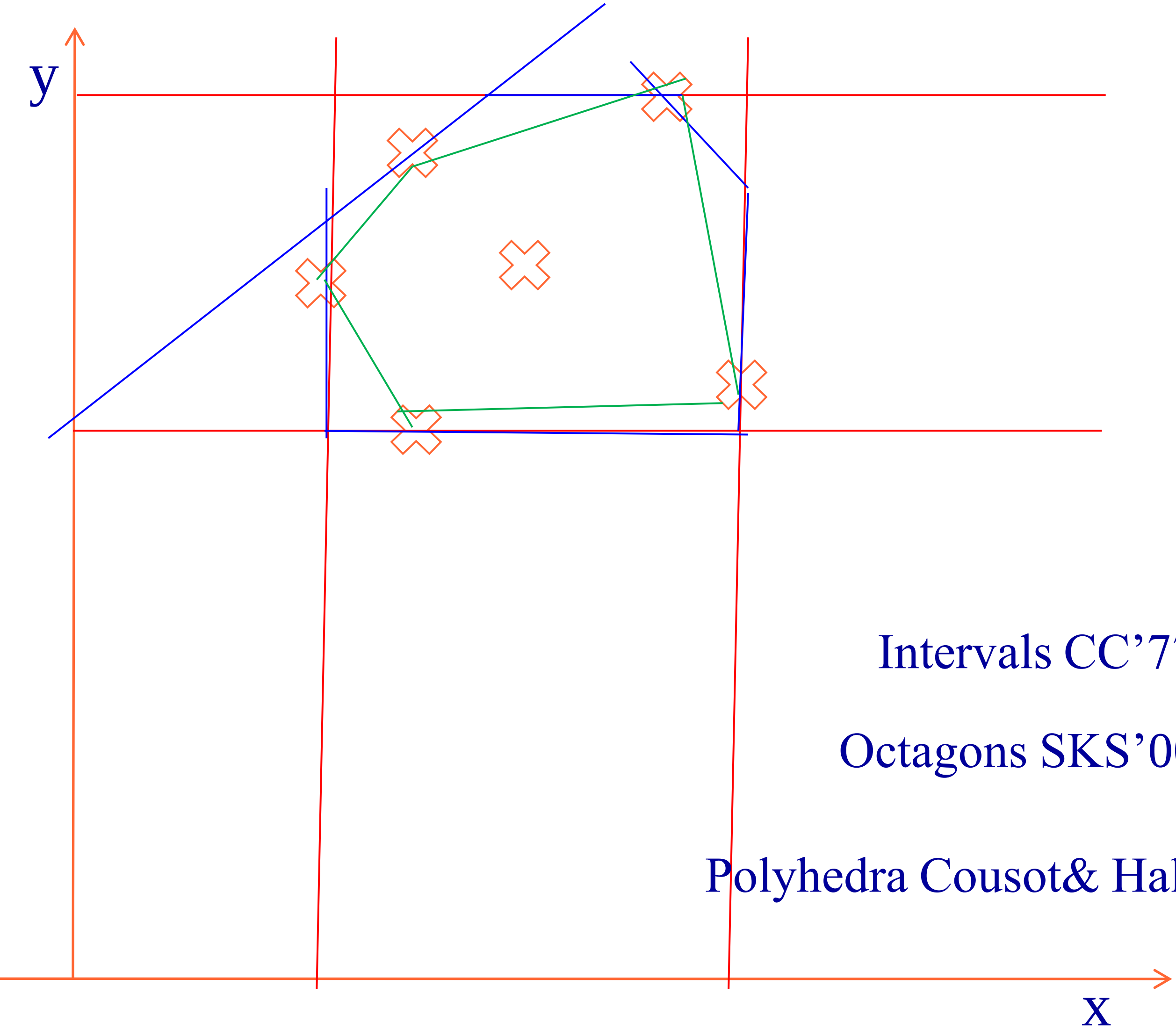
IntEntry(4) = IntExit(2) ⊓ [1001, ∞]

IntExit(4) = IntEntry(4)

Non Montonicity of Widening

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

Domains with Infinite Heights for Integers



Intervals CC'77: $\pm x_i \leq b$

Octagons SKS'00, Mine'01: $\pm x_i \pm y_i \leq b$

Polyhedra Cousot& Halbwachs'78: $\Sigma a_i * x_i \leq b$

Widening and Narrowing Summary

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

```
int f(x) [-∞ , ∞] {
  if x > 100
        [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

Backward Analysis

- Sometimes interesting information involves the future of the computation
- Apply Chaotic Iterations by following edges backward
- Examples
  - Liveness information
    - A variable x is live at a program point if there exists a path from this point to a use of x and that this path does not include an assignment to x
  - Busy expressions
    - A formal expression is busy at the program point if all paths from this point use this expression

Specialized Chaotic Iterations (Backward)

Chaotic(G(V, E): Graph, e: Node, L: Lattice, ι: L, f: E →(L →L)){

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

df[e] = ι

WL = {e}

while (WL ≠ Ø) do

select and remove an element u ∈ WL

for each v, such that. (u, v) ∈E do

temp = $f(e)(df_{exit}[v])$

new := $df_{exit}(u) \sqcup$ temp

if (new ≠ $df_{exit}[u]$) then

$df_{exit}[u]$:= new;

WL := WL ∪{u}

Conclusions(1)

- Good static analysis =
  - Precise enough (for the client)
  - Efficient enough
- Good static analysis
  - Good domain
    - Abstract non-important details
    - Represent relevant concrete information
    - Only maintains important correlations
    - Precise and efficient abstract meaning of abstract interpreters
    - Efficient join implementation
    - Small height or widening

Conclusion

- Chaotic iterations is a powerful technique
- Easy to implement
- Rather precise
- But expensive
  - More efficient methods exist for structured programs
- Abstract interpretation relates runtime semantics and static information
- The concrete semantics serves as a tool in designing abstractions
  - More intuition will be given in the sequel