Program Analysis and Verification

0368-4479

Noam Rinetzky

Lecture 6: Abstract Interpretation

Slides credit: Roman Manevich, Mooly Sagiv, Eran Yahav

Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis

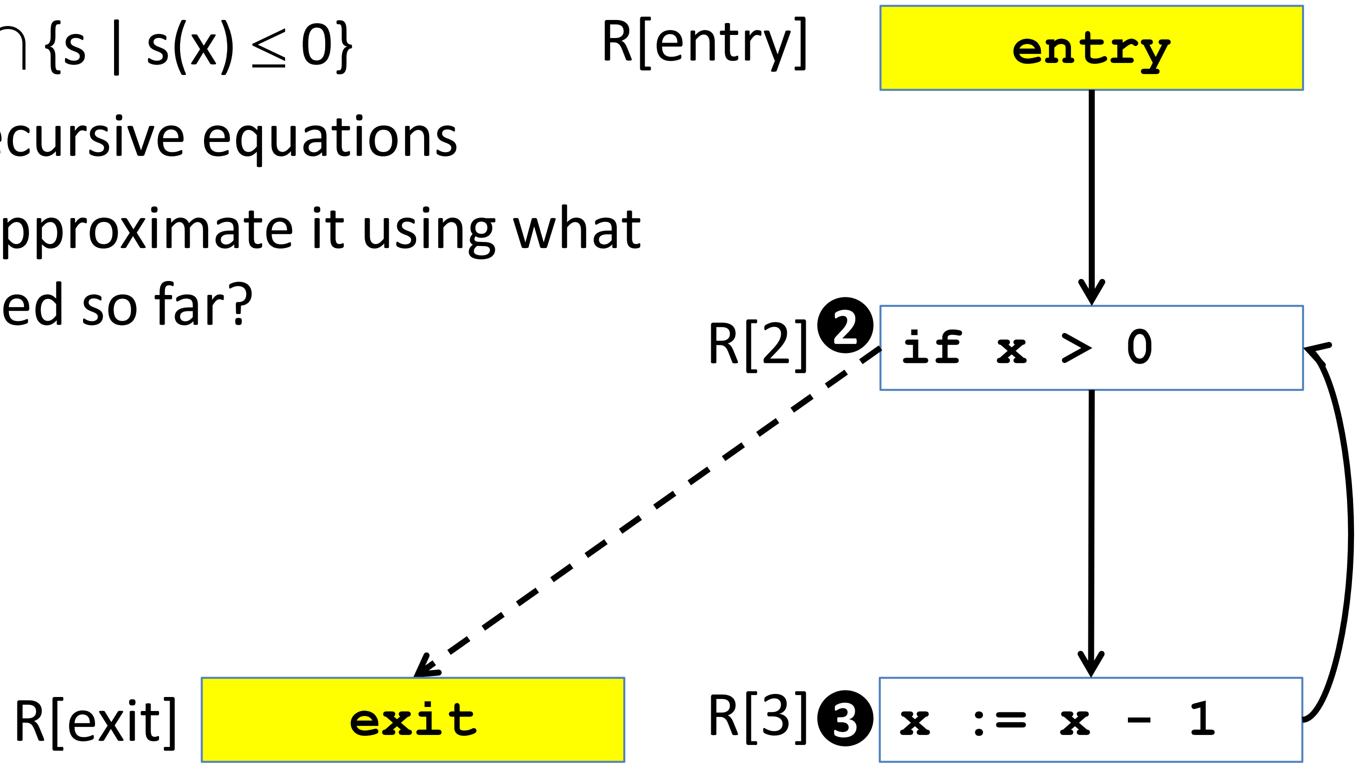
RECAP

The collecting lattice

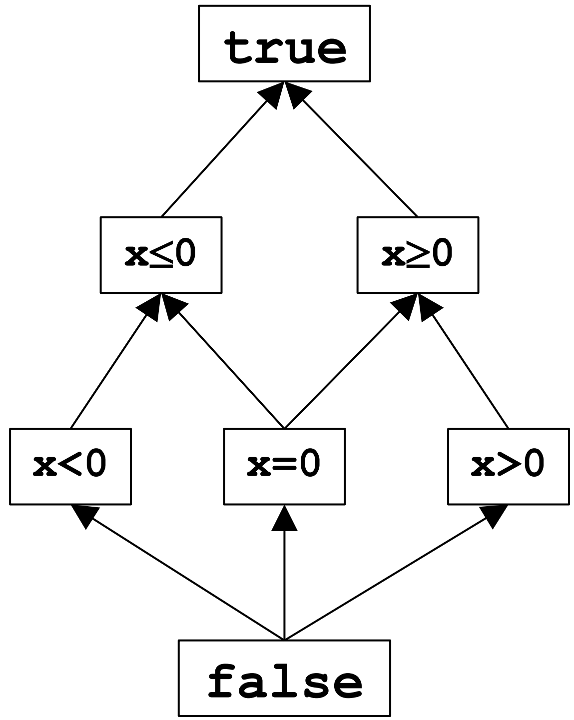
- Lattice for a given control-flow node *v*:
  L_v=($2^{\mathbf{State}}$, $\subseteq$, $\cup$, $\cap$, $\emptyset$, **State**)
- Lattice for entire control-flow graph with nodes *V*:
  $$L_{CFG} = \mathrm{Map}(V, L_v)$$
- We will use this lattice as a baseline for static analysis and define abstractions of its elements

Equational definition of the collecting semantics

- R[2] = R[entry] $\cup$ $[\![$**x:=x-1**$]\!]$ R[3]
- R[3] = R[2] $\cap$ {s | s(x) > 0}
- R[exit] = R[2] $\cap$ {s | s(x) $\leq$ 0}
- A system of recursive equations
- How can we approximate it using what we have learned so far?



Abstract Domain



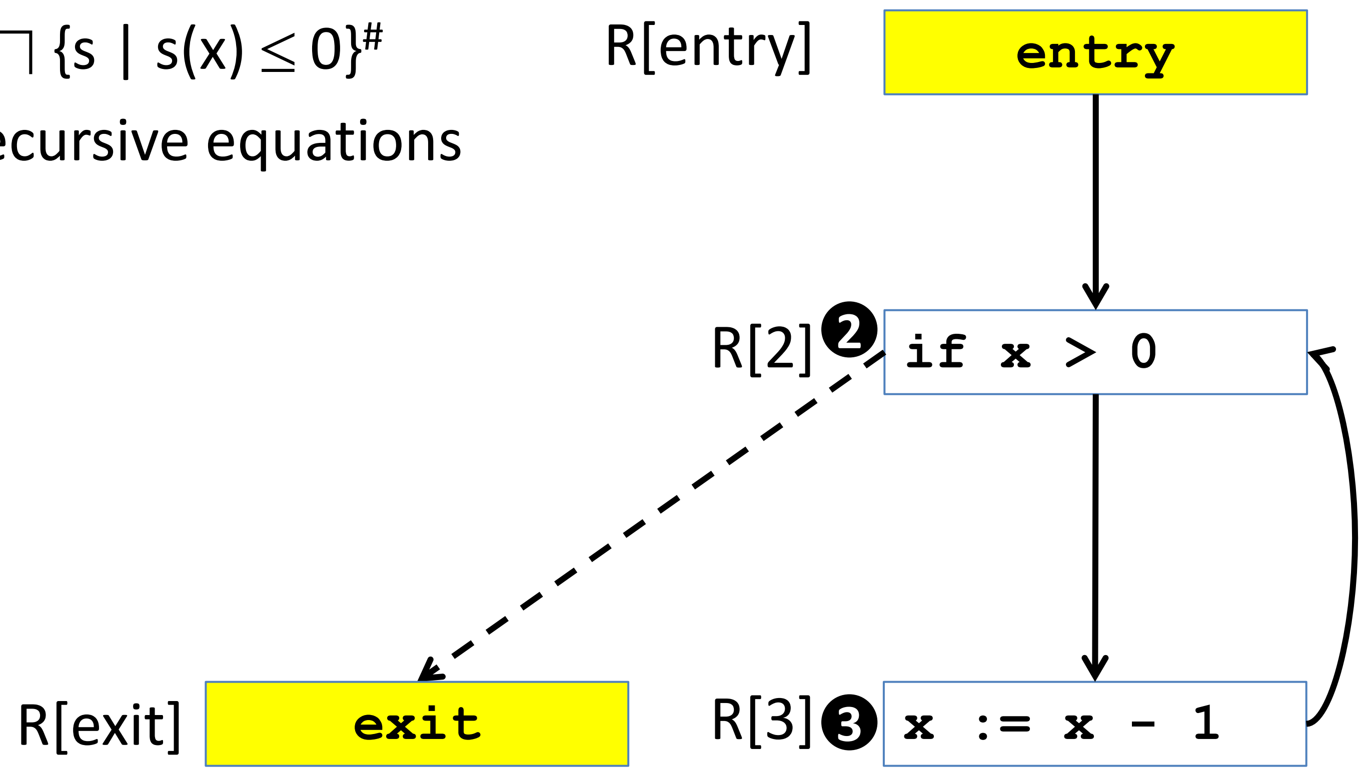
X

An abstract semantics

Abstract transformer for $\mathtt{x:=x-1}$

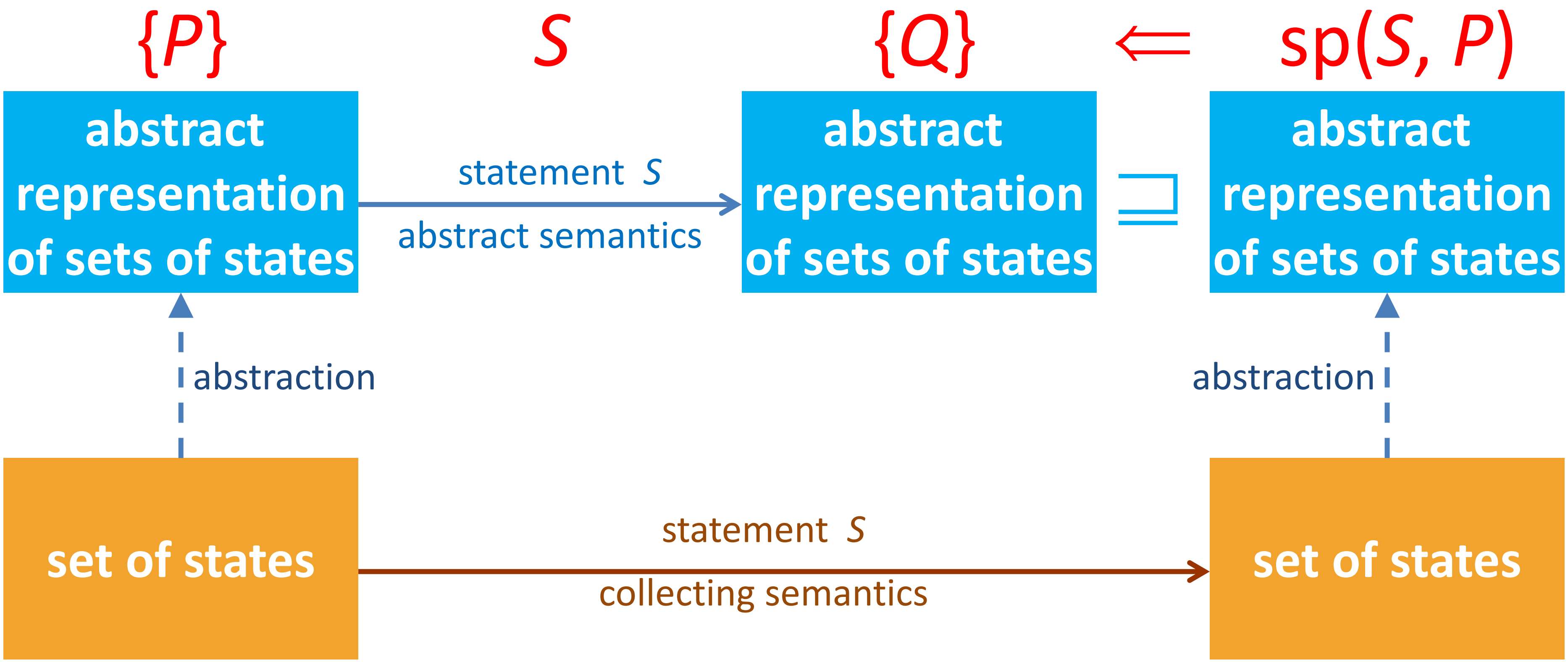
- $R[2] = R[entry] \sqcup [\![\mathtt{x:=x-1}]\!]^{\#}\ R[3]$
- $R[3] = R[2] \sqcap \{s \mid s(x) > 0\}^{\#}$
- $R[exit] = R[2] \sqcap \{s \mid s(x) \leq 0\}^{\#}$
- A system of recursive equations

Abstract representation of $\{s \mid s(x) < 0\}$

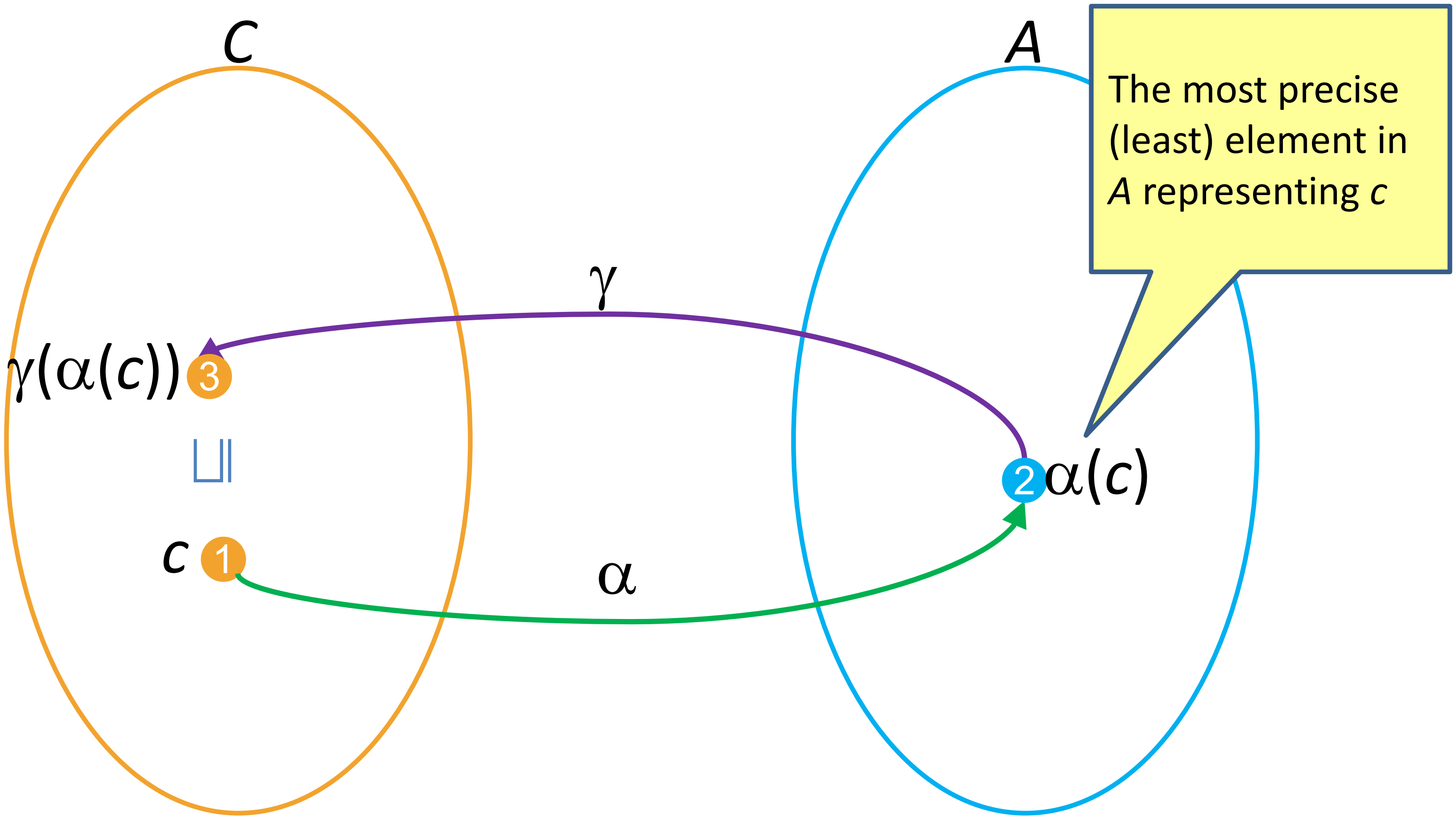


Abstract interpretation via abstraction

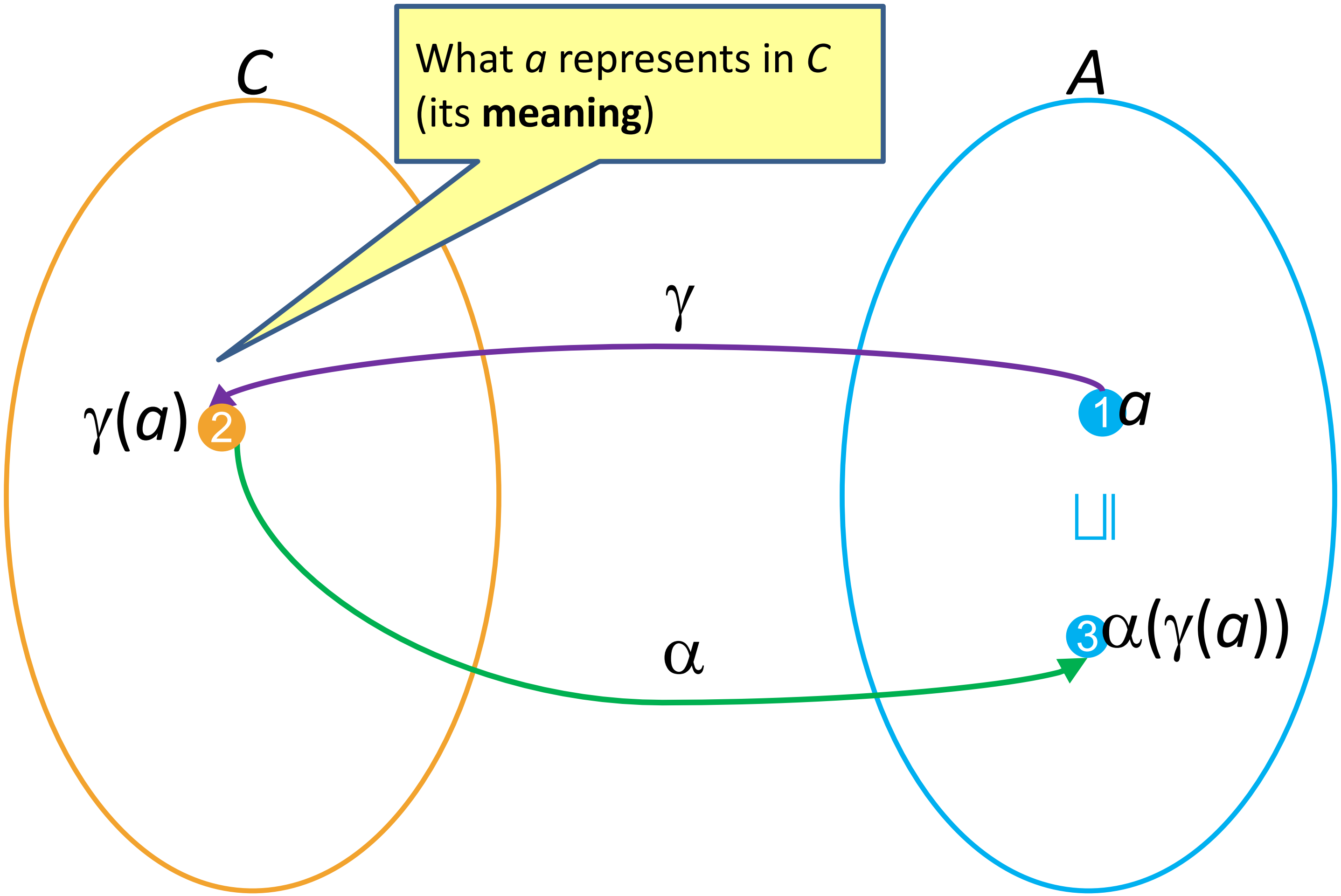
generalizes axiomatic verification



Galois Connection: $c \sqsubseteq \gamma(\alpha(c))$

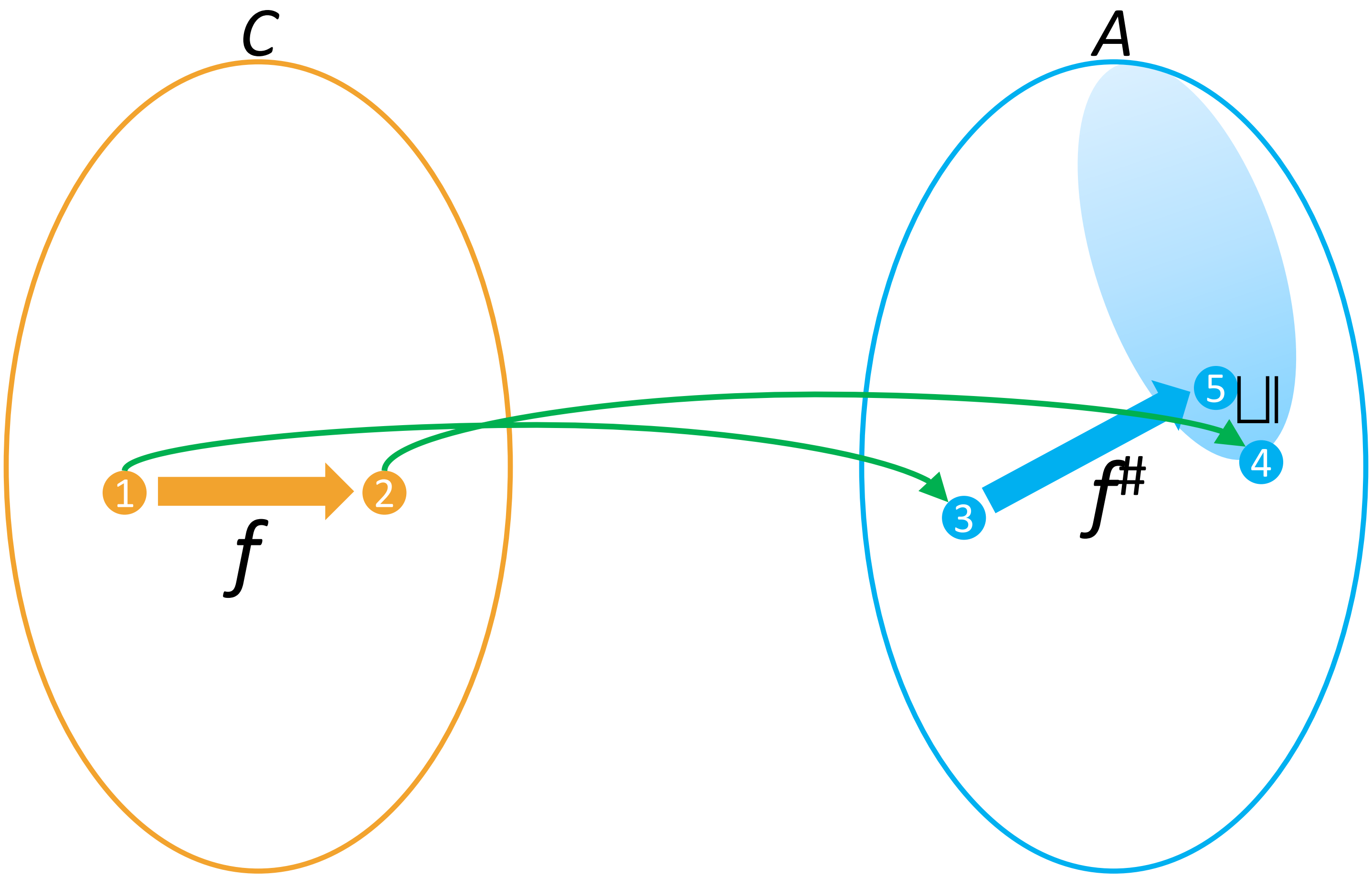


Galois Connection: $\alpha(\gamma(a)) \sqsubseteq a$



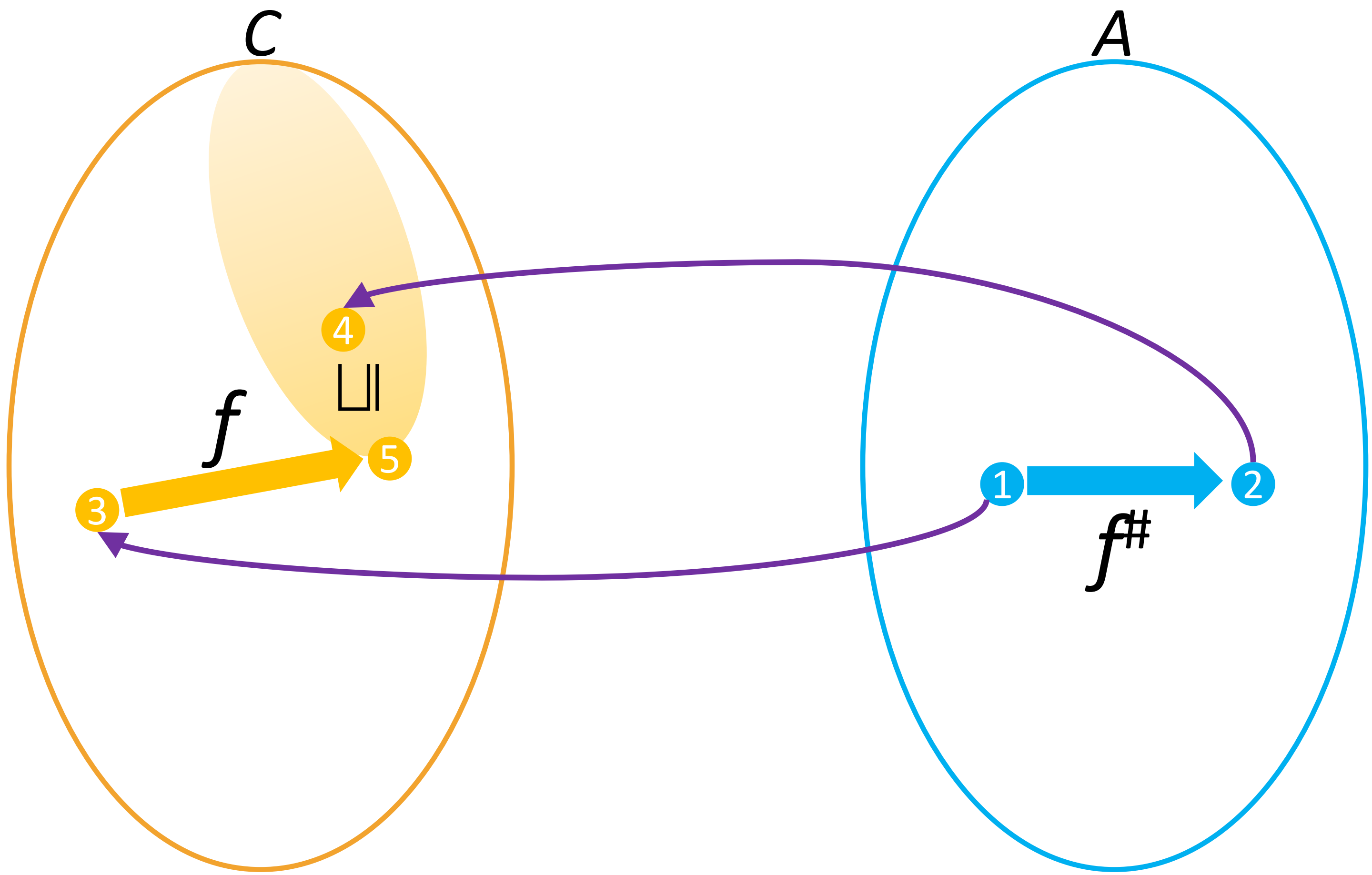
Transformer soundness condition 1

$$\forall c\colon f(c)=c' \Rightarrow \alpha(f^{\#}(c)) \sqsupseteq \alpha(c')$$



Transformer soundness condition 2

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



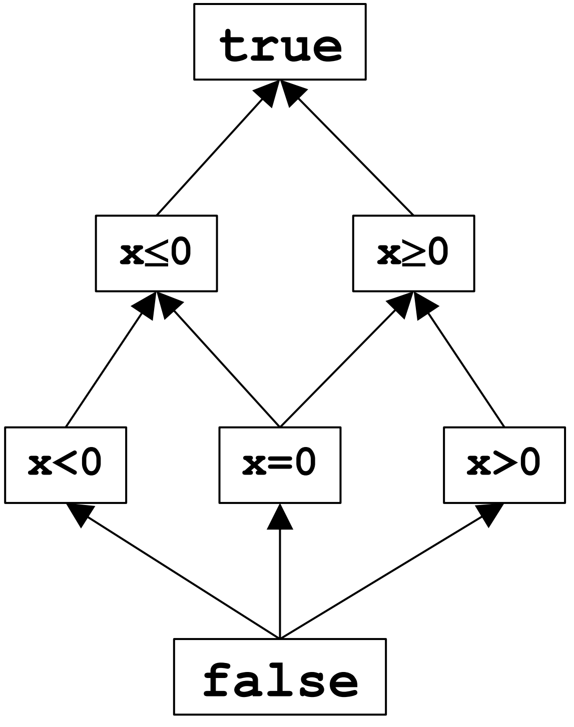
Soundness theorem 1

1. Given two complete lattices
   $C = (D^C, \sqsubseteq^C, \sqcup^C, \sqcap^C, \perp^C, \top^C)$
   $A = (D^A, \sqsubseteq^A, \sqcup^A, \sqcap^A, \perp^A, \top^A)$
   and $GC^{C,A}=(C, \alpha, \gamma, A)$ with
2. Monotone concrete transformer $f : D^C \rightarrow D^C$
3. Monotone abstract transformer $f^\# : D^A \rightarrow D^A$
4. $\forall a \in D^A : f(\gamma(a)) \sqsubseteq \gamma(f^\#(a))$

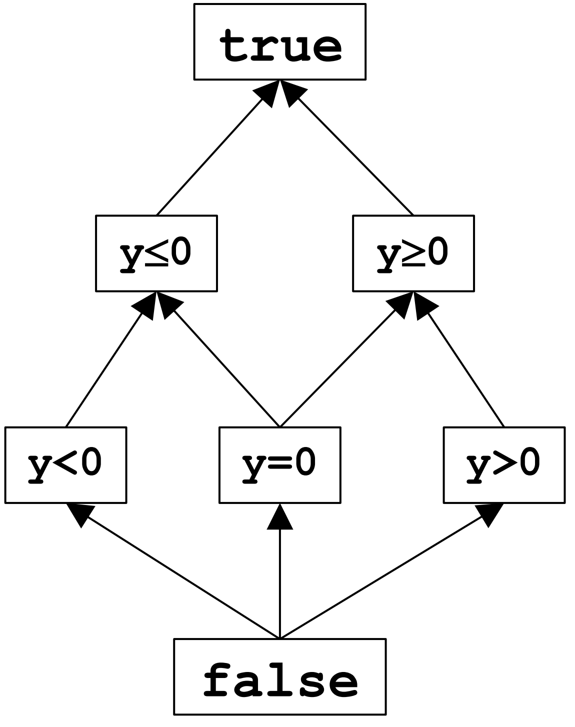
Then

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

Abstract Domain



x



y

How can we compose them?

Composing Analyses

Three example analyses

- Abstract states are conjunctions of constraints
- **V**ariable **E**qualities
  - $\mathit{VE\text{-}factoids} = \{\, x{=}y \mid x, y \in \mathrm{Var}\} \cup \mathrm{false}$
    $\mathit{VE} = (2^{\mathit{VE\text{-}factoids}}, \supseteq, \cap, \cup, \mathit{false}, \emptyset)$
- **C**onstant **P**ropagation
  - $\mathit{CP\text{-}factoids} = \{\, x{=}c \mid x \in \mathrm{Var}, c \in \mathbf{Z}\} \cup \mathrm{false}$
    $\mathit{CP} = (2^{\mathit{CP\text{-}factoids}}, \supseteq, \cap, \cup, \mathit{false}, \emptyset)$
- **A**vailable **E**xpressions
  - $\mathit{AE\text{-}factoids} = \{\, x{=}y{+}z \mid x \in \mathrm{Var}, y{,}z \in \mathrm{Var} \cup \mathbf{Z}\} \cup \mathrm{false}$
    $A = (2^{\mathit{AE\text{-}factoids}}, \supseteq, \cap, \cup, \mathit{false}, \emptyset)$

Lattice combinators reminder

- Cartesian Product
  - $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
    $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
  - $\text{Cart}(L_1, L_2) = (D_1 \times D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$
- Disjunctive completion
  - $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
  - $\text{Disj}(L) = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
- Relational Product
  - $\text{Rel}(L_1, L_2) = \text{Disj}(\text{Cart}(L_1, L_2))$

Cartesian product of complete lattices

- For two complete lattices
  $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- Define the poset
  $L_{cart} = (D_1 \times D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \bot_{cart}, \top_{cart})$
  as follows:
  - $(x_1, x_2) \sqsubseteq_{cart} (y_1, y_2)$ iff
    $x_1 \sqsubseteq_1 y_1$ and
    $x_2 \sqsubseteq_2 y_2$
  - $\sqcup_{cart} = ?$ $\sqcap_{cart} = ?$ $\bot_{cart} = ?$ $\top_{cart} = ?$
- **Lemma:** L is a complete lattice
- Define the Cartesian constructor $L_{cart} = \text{Cart}(L_1, L_2)$

Cartesian product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Cartesian Product

  $$GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$$

  – $\alpha^{C,A\times B}(X)=$?

  – $\gamma^{A\times B,C}(Y) =$?

Cartesian product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Cartesian Product

  $$GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$$

  - $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$
  - $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- What about transformers?

Cartesian product transformers

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$ $F^{A}[st] : A \to A$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$ $F^{B}[st] : B \to B$
- Cartesian Product
  $$GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$$
  - $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$
  - $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- How should we define $F^{A\times B}[st] : A\times B \to A\times B$

Cartesian product transformers

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$ $\quad F^A[\text{st}] : A \to A$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$ $\quad F^B[\text{st}] : B \to B$
- Cartesian Product
  $$GC^{C,A\times B} = (C, \alpha^{C,A\times B}, \gamma^{A\times B,C}, A\times B)$$
  - $\alpha^{C,A\times B}(X) = (\alpha^{C,A}(X), \alpha^{C,B}(X))$
  - $\gamma^{A\times B,C}(Y) = \gamma^{A,C}(X) \cap \gamma^{B,C}(X)$
- How should we define $F^{A\times B}[\text{st}] : A\times B \to A\times B$
- Idea: $F^{A\times B}[\text{st}](a, b) = (F^A[\text{st}]\ a, F^B[\text{st}]\ b)$
- Are component-wise transformers precise?

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP analysis

```
a := 9;      {a=9}
b := 9;      {a=9, b=9}
c := a;      {a=9, b=9, c=9}
```

VE analysis

```
a := 9;      {}
b := 9;      {}
c := a;      {c=a}
```

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP analysis + VE analysis

```
a := 9;       {a=9}
b := 9;       {a=9, b=9}
c := a;       {a=9, b=9, c=9, c=a}
```

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP×VE analysis

```
a := 9;      {a=9}
b := 9;      {a=9, b=9}
c := a;      {a=9, b=9, c=9} {c=a}
```

{a=b, b=c}

Missing

Cartesian product analysis example

- Abstract interpreter 1: **C**onstant **P**ropagation
- Abstract interpreter 2: **V**ariable **E**qualities
- Let's compare
  - Running them separately and combining results
  - Running the analysis with their Cartesian product

CP×VE analysis

```
assume(a=b);{}, {a=b}
assume(a=0);{a=0} , {a=b}
```

{b=0} Missing

Transformers for Cartesian product

- Naïve (component-wise) transformers do not utilize information from both components
  - Same as running analyses separately and then combining results
- Can we treat transformers from each analysis as black box and obtain best transformer for their combination?

Can we combine transformer modularly?

- No generic method for any abstract interpretations

Reducing values for CP$\times$VE

- X = set of CP constraints of the form x=c (e.g., **`a=9`**)
- Y = set of VE constraints of the form x=y
- Reduce$^{CP\times VE}(X, Y) = (X', Y')$ such that $(X', Y') \sqsubseteq (X', Y')$
- Ideas?

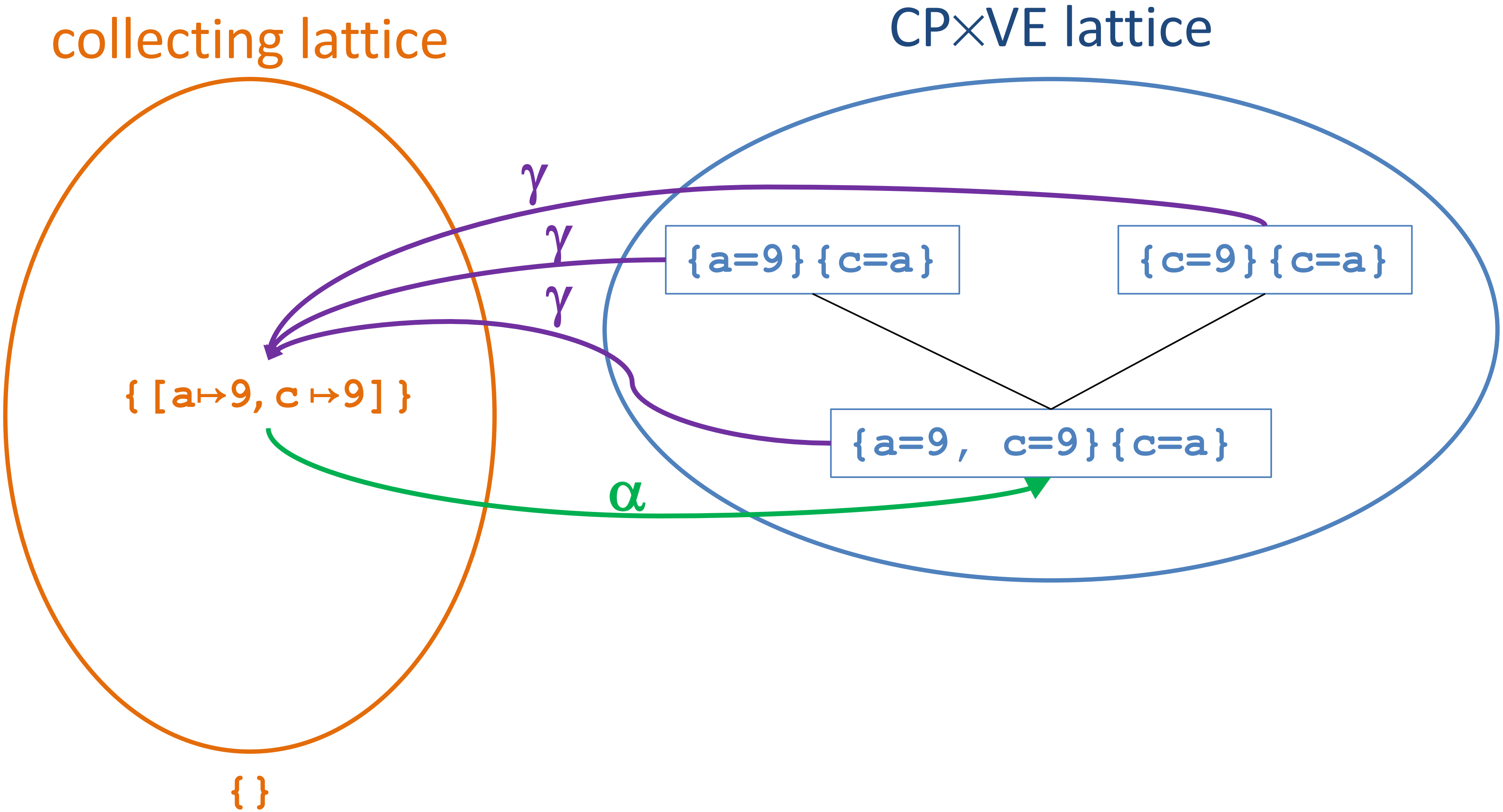
Reducing values for CP$\times$VE

- X = set of CP constraints of the form x=c (e.g., **`a=9`**)
- Y = set of VE constraints of the form x=y
- Reduce$^{CP\times VE}(X, Y) = (X', Y')$ such that $(X', Y') \sqsubseteq (X', Y')$
- ReduceRight:
  - if a=b $\in X$ and a=c $\in Y$ then add b=c to Y
- ReduceLeft:
  - If a=c and b=c $\in Y$ then add a=b to X
- Keep applying ReduceLeft and ReduceRight and reductions on each domain separately until reaching a fixed-point

Transformers for Cartesian product

- Do we get the best transformer by applying component-wise transformer followed by reduction?
  - Unfortunately, no (what’s the intuition?)
  - Can we do better?
  - Logical Product [Gulwani and Tiwari, PLDI 2006]

Product vs. reduced product



Reduced product

- For two complete lattices
  $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \perp_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \perp_2, \top_2)$
- Define the reduced poset
  $D_1 \sqcap D_2 = \{(d_1,d_2) \in D_1 \times D_2 \mid (d_1,d_2) = \alpha \circ \gamma\ (d_1,d_2)\ \}$
  $L_1 \sqcap L_2 = (D_1 \sqcap D_2, \sqsubseteq_{cart}, \sqcup_{cart}, \sqcap_{cart}, \perp_{cart}, \top_{cart})$

Transformers for Cartesian product

- Do we get the best transformer by applying component-wise transformer followed by reduction?
  - Unfortunately, no (what's the intuition?)
  - Can we do better?
  - Logical Product [Gulwani and Tiwari, PLDI 2006]

Combining Abstract Interpreters

Sumit Gulwani
Microsoft Research
sumitg@microsoft.com

Ashish Tiwari
SRI International
tiwari@csl.sri.com

Abstract

We present a methodology for automatically combining abstract interpreters over given lattices to construct an abstract interpreter for the combination of those lattices. This lends modularity to the process of design and implementation of abstract interpreters.

We define the notion of logical product of lattices. This kind of combination is more precise than the reduced product combination. We give algorithms to obtain the join operator and the existential quantification operator for the combined lattice from the corresponding operators of the individual lattices. We also give a bound on the number of steps required to reach a fixed point across loops during analysis over the combined lattice in terms of the corresponding bounds for the individual lattices. We prove that our combination methodology yields the most precise abstract interpretation operators over the logical product of lattices when the individual lattices are over theories that are convex, stably infinite, and disjoint.

We also present an interesting application of logical product wherein some lattices can be reduced to combination of other (unrelated) lattices with known abstract interpreters.

Categories and Subject Descriptors D.2.4 [*Software Engineering*]: Software/Program Verification; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Program analysis

General Terms Algorithms, Theory, Verification

Keywords Abstract Interpreter, Logical Product, Reduced Product, Nelson-Oppen Combination

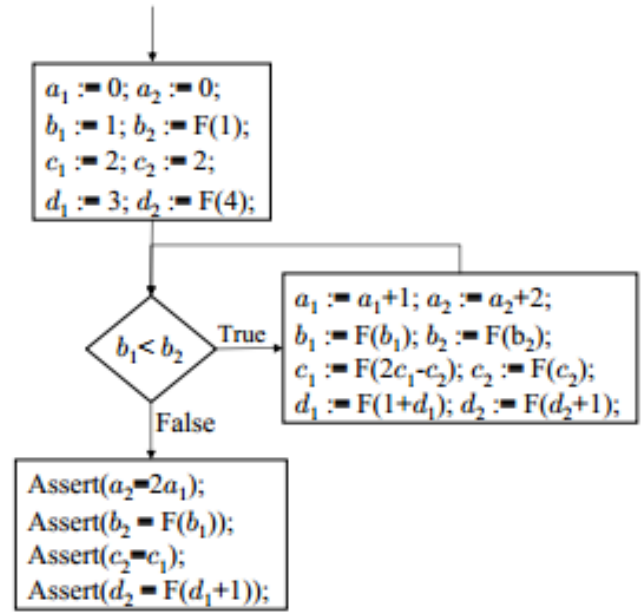


Figure 1. This program illustrates the difference between precision of performing analysis over *direct product*, *reduced product*, and *logical product* of the linear arithmetic lattice and uninterpreted functions lattice. Analysis over direct product can verify the first two assertions, while analysis over reduced product can verify the first three assertions. The analysis over logical product can verify all assertions. F denotes some function without any side-effects and can be modeled as an uninterpreted function for purpose of proving the assertions.

Logical product--

- Assume $A=(D,\ldots)$ is an abstract domain that supports two operations: for $x \in D$
  - inferEqualities(x) = $\{ a=b \mid \gamma(x) \models a=b \}$
    returns a set of equalities between variables that are satisfied in all states given by x
  - refineFromEqualities(x, $\{a=b\}$) = y
    such that
    - $\gamma(x)=\gamma(y)$
    - $y \sqsubseteq x$

Developing a transformer for *EQ* - 1

- Input has the form $X = \bigwedge\{a=b\}$
- $\mathrm{sp}(x{:=}expr,\ \varphi) = \exists v.\ x{=}\mathrm{expr}[v/\mathrm{x}] \wedge \varphi[v/\mathrm{x}]$
- $\mathrm{sp}(x{:=}y,\ X) = \exists v.\ x{=}y[v/\mathrm{x}] \wedge \bigwedge\{a{=}b\}[v/\mathrm{x}] = \ldots$
- Let's define helper notations:
  - $\mathrm{EQ}(X, y) = \{y{=}a, b{=}y \in X\}$
    - Subset of equalities containing *y*
  - $\mathrm{EQc}(X, y) = X \setminus \mathrm{EQ}(X, y)$
    - Subset of equalities **not** containing *y*

Developing a transformer for *EQ* - 2

- $\mathrm{sp}(x{:=}y,\ X) = \exists v.\ x{=}y[v/\mathrm{x}] \wedge \bigwedge\{a{=}b\}[v/\mathrm{x}] = \ldots$
- Two cases
  - x is y: $\mathrm{sp}(x{:=}y,\ X) = X$
  - x is different from y:
    $\mathrm{sp}(x{:=}y,\ X) = \exists v.\ x{=}y \wedge \mathrm{EQ}(\mathrm{X}, \mathrm{x})[v/\mathrm{x}] \wedge \mathrm{EQc}(X, x)[v/\mathrm{x}]$
    $= x{=}y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}(\mathrm{X}, \mathrm{x})[v/\mathrm{x}]$
    $\Rightarrow x{=}y \wedge \mathrm{EQc}(X, x)$
- Vanilla transformer: $[\![x{:=}y]\!]^{\#1}\ X = x{=}y \wedge \mathrm{EQc}(X, x)$
- Example: $[\![x{:=}y]\!]^{\#1} \bigwedge\{\mathrm{x{=}p,\ q{=}x,\ m{=}n}\} = \bigwedge\{\mathrm{x{=}y,\ m{=}n}\}$
  Is this the most precise result?

Developing a transformer for *EQ* - 3

- $[\![x:=y]\!]^{\#1} \bigwedge\{x{=}p,\ x{=}q,\ m{=}n\} = \bigwedge\{x{=}y,\ m{=}n\} \sqsupseteq \bigwedge\{x{=}y,\ m{=}n,\ p{=}q\}$
  - Where does the information p=q come from?
- $\mathrm{sp}(x:=y,\ X) =$

  $$x=y \wedge \mathrm{EQc}(X, x) \wedge \exists v.\ \mathrm{EQ}(\mathrm{X}, \mathrm{x})[v/\mathrm{x}]$$
- $\exists v.\ \mathrm{EQ}(\mathrm{X}, \mathrm{x})[v/\mathrm{x}]$ holds possible equalities between different *a*'s and *b*'s – how can we account for that?

Developing a transformer for *EQ* - 4

- Define a reduction operator:
  Explicate(X) = if exist {a=b, b=c}$\subseteq$X
  but not {a=c} $\subseteq$X then
  Explicate(X$\cup${a=c})
  else
  X
- Define $[\![x{:=}y]\!]^{\#2} = [\![x{:=}y]\!]^{\#1} \circ$ Explicate
- $[\![x{:=}y]\!]^{\#2}(\wedge\{x=p, x=q, m=n\}) = \bigwedge\{x=y, m=n, p=q\}$
  is this the best transformer?

Developing a transformer for *EQ* - 5

- $[\![x{:=}y]\!]^{\#2}$ $(\bigwedge\{y{=}z\})$ = {x=y, y=z} $\sqsupseteq$ {x=y, y=z, x=z}
- Idea: apply reduction operator again after the vanilla transformer
- $[\![x{:=}y]\!]^{\#3}$ = Explicate $\circ$ $[\![x{:=}y]\!]^{\#1}$ $\circ$ Explicate

Logical Product-

The element E after an assignment node $x := e$ is the strongest postcondition of the element E' before the assignment node. It is computed by using an existential quantification operator $Q_{L_1 \bowtie L_2}$ as described below.

$$
\begin{array}{rcl}
E & = & Q_{L_1 \bowtie L_2}(E_1, \{x'\}) \\
\text{where } E_1 & = & E'[x'/x] \wedge E'_1 \\
\text{and } E'_1 & = & \begin{cases} x = e[x'/x] & \text{if } \mathit{Symbols}(e) \subseteq \Sigma_{\mathbb{T}_1 \cup \mathbb{T}_2} \\ \mathit{true} & \text{otherwise} \end{cases}
\end{array}
$$

safely abstracting the existential quantifier

basically the strongest postcondition

Example

Information loss example

```
if (…)           {}
  b := 5         {b=5}
else
  b := -5        {b=-5}
                 {b=⊤}
if (b>0)
  b := b-5       {b=⊤}
else
  b := b+5       {b=⊤}
assert b==0      can't prove
```

Disjunctive completion of a lattice

- For a complete lattice
  $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Define the powerset lattice
  $L_\vee = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
  $\sqsubseteq_\vee = ?$ $\sqcup_\vee = ?$ $\sqcap_\vee = ?$ $\bot_\vee = ?$ $\top_\vee = ?$
- **Lemma:** $L_\vee$ is a complete lattice
- $L_\vee$ contains all subsets of *D*, which can be thought of as disjunctions of the corresponding predicates
- Define the disjunctive completion constructor
  $L_\vee = \text{Disj}(L)$

Disjunctive completion for GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Disjunctive completion
  $$GC^{C,P(A)} = (C, \alpha^{P(A)}, \gamma^{P(A)}, P(A))$$
  - $\alpha^{C,P(A)}(X) = ?$
  - $\gamma^{P(A),C}(Y) = ?$

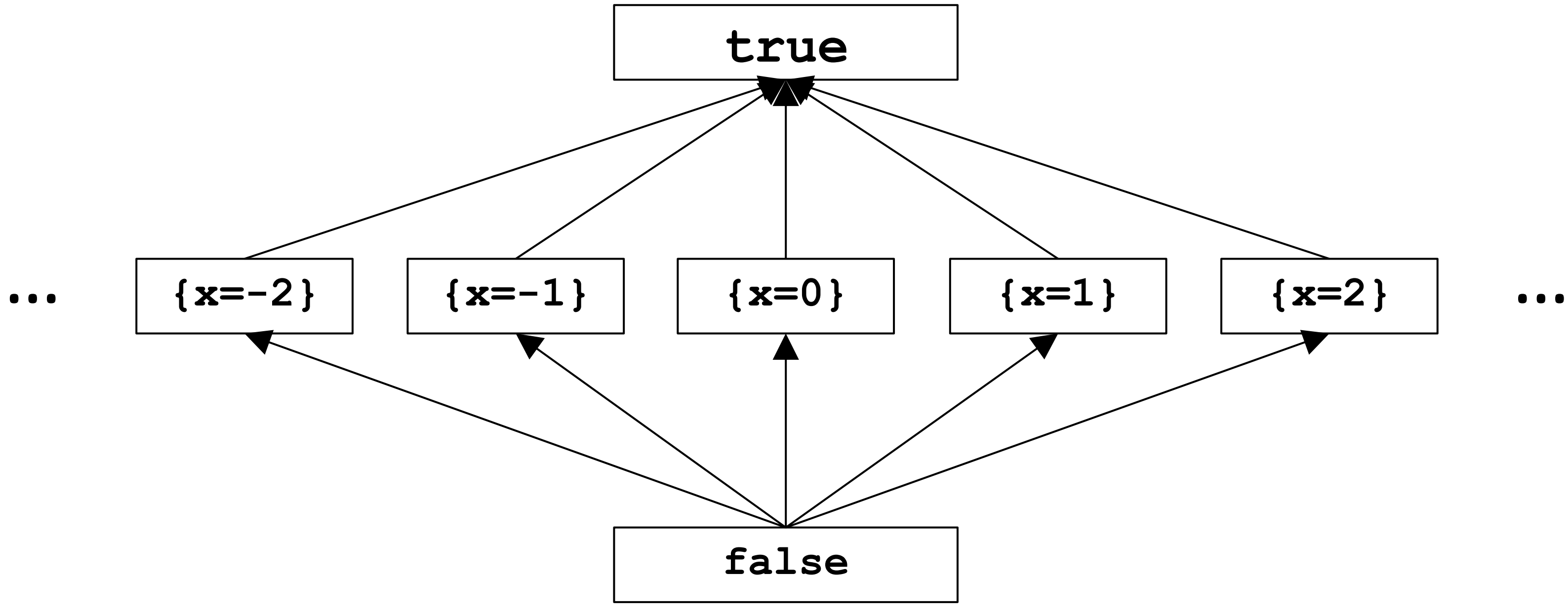
Disjunctive completion for GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Disjunctive completion
  $$GC^{C,P(A)} = (C, \alpha^{P(A)}, \gamma^{P(A)}, P(A))$$
  - $\alpha^{C,P(A)}(X) = \{\alpha^{C,A}(\{x\}) \mid x \in X\}$
  - $\gamma^{P(A),C}(Y) = \cup\{\gamma^{P(A)}(y) \mid y \in Y\}$
- What about transformers?

Information loss example

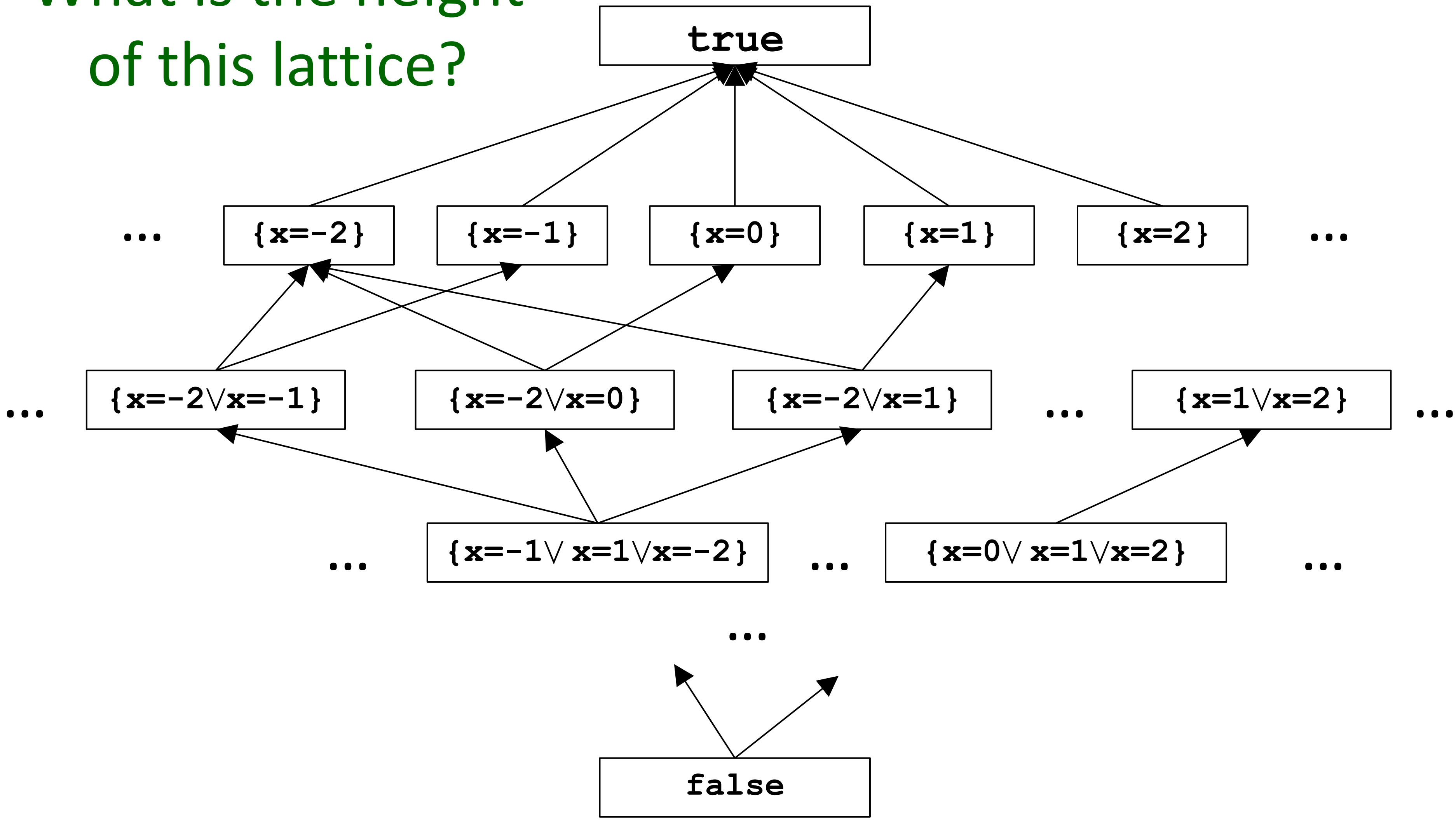
`if (…)`	{}
`  b := 5`	{b=5}
`else`	
`  b := -5`	{b=-5}
	{b=5 ∨ b=-5}
`if (b>0)`	
`  b := b-5`	{b=0}
`else`	
`  b := b+5`	{b=0}
`assert b==0`	proved

The base lattice CP



The disjunctive completion of CP

What is the height of this lattice?



Taming disjunctive completion

- Disjunctive completion is very precise
  - Maintains correlations between states of different analyses
  - Helps handle conditions precisely
  - But very expensive – number of abstract states grows exponentially
  - May lead to non-termination
- Base analysis (usually product) is less precise
  - Analysis terminates if the analyses of each component terminates
- How can we combine them to get more precision yet ensure termination and state explosion?

Taming disjunctive completion

- Use different abstractions for different program locations
  - At loop heads use coarse abstraction (base)
  - At other points use disjunctive completion
- Termination is guaranteed (by base domain)
- Precision increased **inside** loop body

With Disj(CP)

```
while (…) {
  if (…)
    b := 5
  else
    b := -5

  if (b>0)
    b := b-5
  else
    b := b+5
  assert b==0
}
```

Doesn’t terminate

With tamed Disj(CP)

CP

Disj(CP)

```
while (…) {
   if (…)
     b := 5
   else
     b := -5

   if (b>0)
     b := b-5
   else
     b := b+5
   assert b==0
}
```

terminates

What **`MultiCartDomain`** implements

Reducing disjunctive elements

- A disjunctive set X may contain within it an ascending chain $Y=a \sqsubseteq b \sqsubseteq c\ldots$
- We only need max(Y) – remove all elements below

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = ?

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \bigsqcup_1, \bigsqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \bigsqcup_2, \bigsqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \bigsqcup_{rel}, \bigsqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  – L_{rel} = Disj(Cart(L_1, L_2))
- **Lemma:** L is a complete lattice
- What does it buy us?
  – How is it relative to Cart(Disj(L_1), Disj(L_2))?
- What about transformers?

Relational product of GCs

- $GC^{C,A} = (C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B} = (C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Relational Product
  $GC^{C,P(A \times B)} = (C, \alpha^{C,P(A \times B)}, \gamma^{P(A \times B),C}, P(A \times B))$
  - $\alpha^{C,P(A \times B)}(X) = ?$
  - $\gamma^{P(A \times B),C}(Y) = ?$

Relational product of GCs

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Relational Product
  $GC^{C,P(A\times B)} = (C, \alpha^{C,P(A\times B)}, \gamma^{P(A\times B),C}, P(A\times B))$
  - $\alpha^{C,P(A\times B)}(X) = \{(\alpha^{C,A}(\{x\}), \alpha^{C,B}(\{x\})) \mid x\in X\}$
  - $\gamma^{P(A\times B),C}(Y) = \cup\{\gamma^{A,C}(y_A) \cap \gamma^{B,C}(y_B) \mid (y_A,y_B)\in Y\}$

Cartesian product example

```
V[10] = V[9] // goto [?= (branch)]
V[11] = P(Reduce_([AssignConstantToVarTransformer, Id]))(V[6]) // b = 9
V[12] = P(Reduce_([AssignVarToVarTransformer, Reduce_VEDomain(AssignVarToVarTransformer)]))(V[11]) // a = d
V[15] = Join_DisjunctiveDomain(V[10], V[12]) // if b != 8 goto (branch)
```

```
public void relationalProductExample(int a, int b, int c, int d) {
    if (a > 5) {
        b = 8;
        a = c;
    } else {
        b = 9;
        a = d;
    }

    if (b == 8) {
        if (a != c)
            error("Unable to prove a==c!");
    }
    else if (b == 9) {
        if (a != d)
            error("Unable to prove a==d!");
    }
    else {
        error("Can't get here");
    }
```

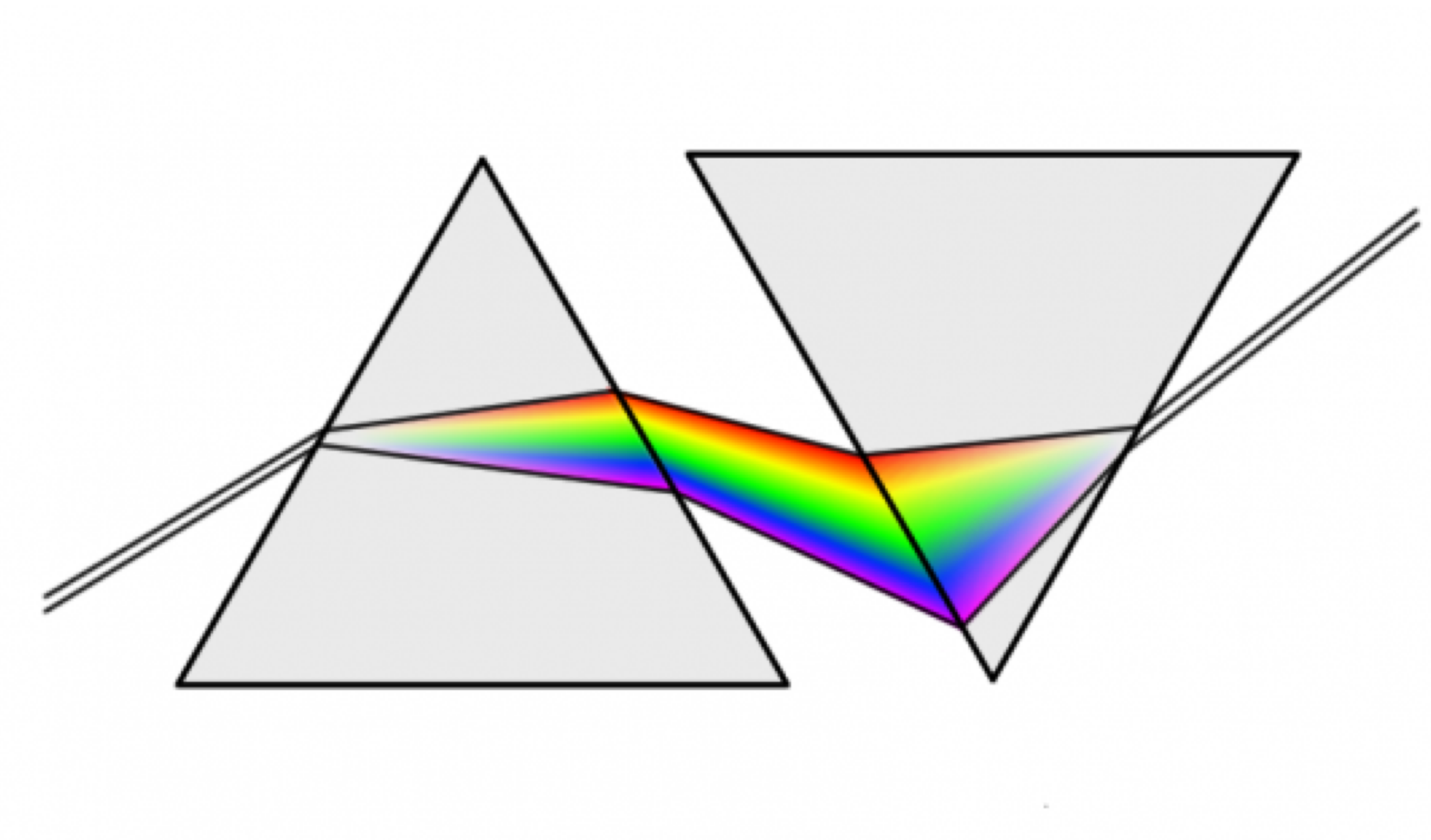
```
Reached fixed-point after 28 iterations.
Solution = {
  V[0] : (true, true)
  V[1] : (true, true)
  V[2] : (true, true)
  V[3] : (true, true)
  V[4] : (true, true)
  V[5] : (true, true)
  V[6] : (true, true)
  V[7] : (true, true)
  V[8] : (b=8, true)
  V[9] : (b=8, a=c)
  V[10] : (b=8, a=c)
  V[11] : (b=9, true)
  V[12] : (b=9, a=d)
  V[15] : or((b=9, a=d), (b=8, a=c))
  V[13] : (b=9, a=d)
  V[14] : (b=8, a=c)
  V[16] : (b=8, a=c)
  V[17] : false
  V[18] : false
  V[19] : false
  V[20] : false
  V[21] : (b=9, a=d)
  V[22] : (b=9, a=d)
  V[23] : false
  V[24] : false
  V[25] : false
  V[26] : false
  V[28] : or((b=9, a=d), (b=8, a=c))
  V[27] : or((b=9, a=d), (b=8, a=c))
}
0 possible errors found.
```

Correlations preserved

Function space

- $GC^{C,A}=(C, \alpha^{C,A}, \gamma^{A,C}, A)$
  $GC^{C,B}=(C, \alpha^{C,B}, \gamma^{B,C}, B)$
- Denote the set of monotone functions from A to B by $A\rightarrow B$
- Define $\sqcup$ for elements of $A\rightarrow B$ as follows
  $(a_1, b_1) \sqcup (a_2, b_2)$ = if $a_1=a_2$ then $\{(a_1, b_1 \sqcup_B b_1)\}$
  else $\{(a_1, b_1), (a_2, b_2)\}$
- **Reduced cardinal power**
  $GC^{C,A\rightarrow B} = (C, \alpha^{C,A\rightarrow B}, \gamma^{A\rightarrow B,C}, A\rightarrow B)$
  - $\alpha^{C,A\rightarrow B}(X) = \sqcup\{(\alpha^{C,A}(\{x\}), \alpha^{C,B}(\{x\})) \mid x\in X\}$
  - $\gamma^{A\rightarrow B,C}(Y) = \cup\{\gamma^{A,C}(y_A) \cap \gamma^{B,C}(y_B) \mid (y_A,y_B)\in Y\}$
- Useful when A is small and B is much larger
  - E.g., typestate verification

Widening/Narrowing



How can we prove this automatically?

```java
public void loopExample() {
    int x = 7;
    while (x < 1000) {
        ++x;
    }
    if (!(x == 1000))
        error("Unable to prove x == 1000!");
}
```

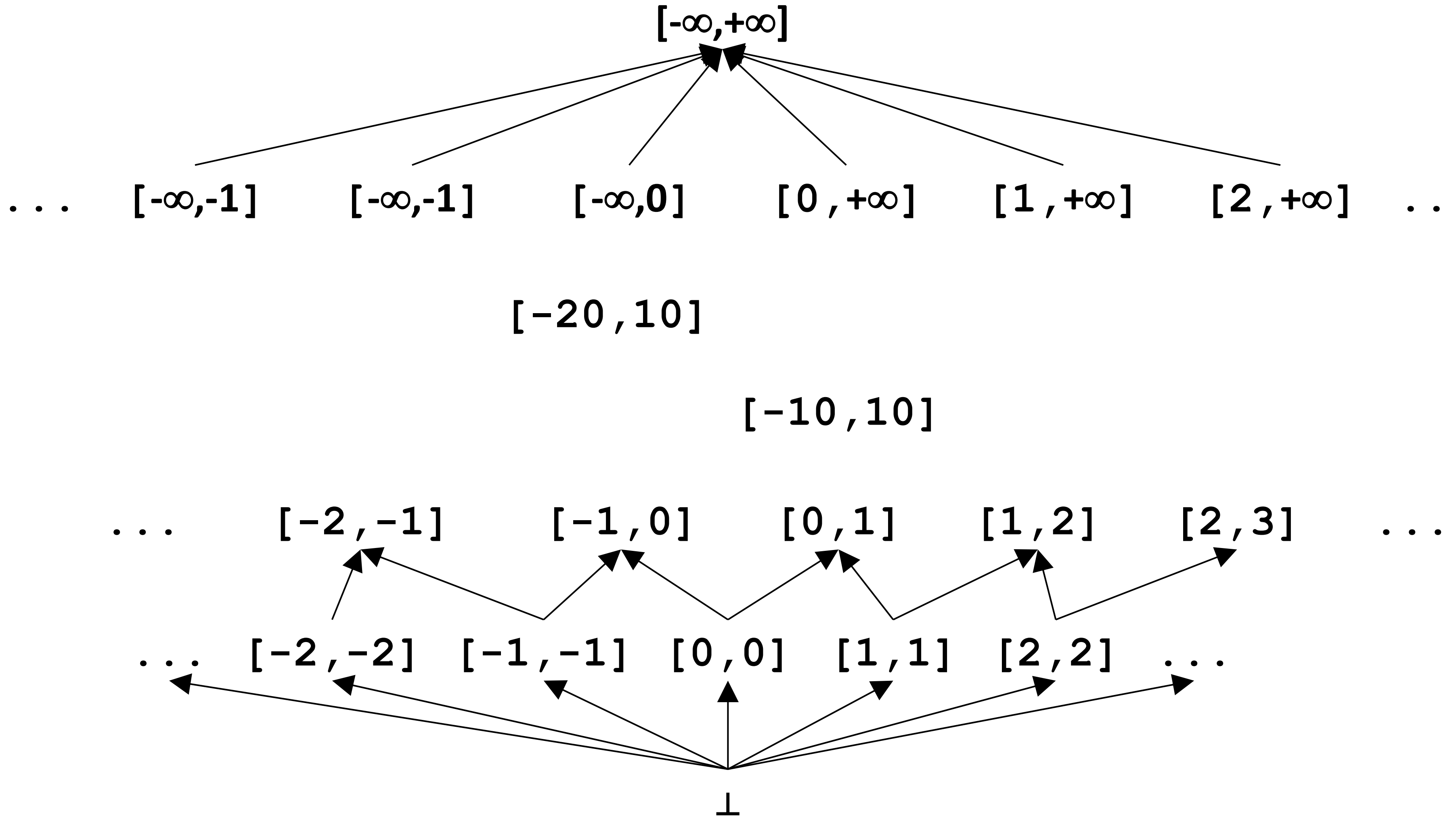
RelProd(CP, VE)

```
Reached fixed-point after 19 iterations.
Solution = {
  V[0] : (true, true)
  V[1] : (true, true)
  V[2] : (x=7, true)
  V[3] : (x=7, true)
  V[4] : (true, true)
  V[7] : (true, true)
  V[5] : (true, true)
  V[6] : (true, true)
  V[8] : (true, true)
  V[9] : (true, true)
  V[10] : (true, true)
  V[12] : (true, true)
  V[11] : (true, true)
}
1 possible errors found.
```

Intervals domain

- One of the simplest numerical domains
- Maintain for each variable x an interval $[L,H]$
  - L is either an integer of $-\infty$
  - H is either an integer of $+\infty$
- A (non-relational) numeric domain

Intervals lattice for variable *x*



Intervals lattice for variable *x*

- $D^{int}[x]$ = { (L,H) | L$\in$-∞,**Z** and H$\in$**Z**,+∞ and $L \leq H$}
- $\bot$
- $\top$=[-∞,+∞]
- $\sqsubseteq$ = ?
  - [1,2] $\sqsubseteq$ [3,4] ?
  - [1,4] $\sqsubseteq$ [1,3] ?
  - [1,3] $\sqsubseteq$ [1,4] ?
  - [1,3] $\sqsubseteq$ [-∞,+∞] ?
- What is the lattice height?

Intervals lattice for variable *x*

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty, \mathbf{Z}$ and $H \in \mathbf{Z}, +\infty$ and $L \leq H \}$
- $\bot$
- $\top = [-\infty, +\infty]$
- $\sqsubseteq$ = ?
  - $[1,2] \sqsubseteq [3,4]$ no
  - $[1,4] \sqsubseteq [1,3]$ no
  - $[1,3] \sqsubseteq [1,4]$ yes
  - $[1,3] \sqsubseteq [-\infty, +\infty]$ yes
- What is the lattice height? Infinite

Joining/meeting intervals

- $[a,b] \sqcup [c,d] = ?$
  - $[1,1] \sqcup [2,2] = ?$
  - $[1,1] \sqcup [2, +\infty] = ?$
- $[a,b] \sqcap [c,d] = ?$
  - $[1,2] \sqcap [3,4] = ?$
  - $[1,4] \sqcap [3,4] = ?$
  - $[1,1] \sqcap [1,+\infty] = ?$
- Check that indeed $x \sqsubseteq y$ if and only if $x \sqcup y = y$

Joining/meeting intervals

- $[a,b] \sqcup [c,d] = [\min(a,c), \max(b,d)]$
  - $[1,1] \sqcup [2,2] = [1,2]$
  - $[1,1] \sqcup [2,+\infty] = [1,+\infty]$
- $[a,b] \sqcap [c,d] = [\max(a,c), \min(b,d)]$ if a proper interval and otherwise $\perp$
  - $[1,2] \sqcap [3,4] = \perp$
  - $[1,4] \sqcap [3,4] = [3,4]$
  - $[1,1] \sqcap [1,+\infty] = [1,1]$
- Check that indeed $x \sqsubseteq y$ if and only if $x \sqcup y = y$

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z}$ and $H \in \mathbf{Z},+\infty$ and $L \leq H\}$
- For a program with variables $Var=\{x_1,\ldots,x_k\}$
- $D^{int}[Var] = ?$

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty,\mathbf{Z}$ and $H \in \mathbf{Z},+\infty$ and $L \leq H\}$
- For a program with variables $Var=\{x_1,\ldots,x_k\}$
- $D^{int}[Var] = D^{int}[x_1] \times \ldots \times D^{int}[x_k]$
- How can we represent it in terms of formulas?

Interval domain for programs

- $D^{int}[x] = \{ (L,H) \mid L \in -\infty, \mathbf{Z}$ and $H \in \mathbf{Z}, +\infty$ and $L \leq H \}$
- For a program with variables $Var = \{x_1, \ldots, x_k\}$
- $D^{int}[Var] = D^{int}[x_1] \times \ldots \times D^{int}[x_k]$
- How can we represent it in terms of formulas?
  - Two types of factoids $x \geq c$ and $x \leq c$
  - Example: $S = \wedge\{x \geq 9, y \geq 5, y \leq 10\}$
  - Helper operations
    - $c + +\infty = +\infty$
    - remove(S, x) = S without any x-constraints
    - lb(S, x) =
    - up(S,x) =

Assignment transformers

- $[\![x := c]\!]^\# S = ?$
- $[\![x := y]\!]^\# S = ?$
- $[\![x := y+c]\!]^\# S = ?$
- $[\![x := y+z]\!]^\# S = ?$
- $[\![x := y*c]\!]^\# S = ?$
- $[\![x := y*z]\!]^\# S = ?$

Assignment transformers

- $[\![x := c]\!]^{\#}\, S$ = remove(S,x) $\cup$ $\{x \geq c,\ x \leq c\}$
- $[\![x := y]\!]^{\#}\, S$ = remove(S,x) $\cup$ $\{x \geq \mathrm{lb}(S,y),\ x \leq \mathrm{ub}(S,y)\}$
- $[\![x := y+c]\!]^{\#}\, S$ = remove(S,x) $\cup$ $\{x \geq \mathrm{lb}(S,y)+\mathrm{c},\ x \leq \mathrm{ub}(S,y)+\mathrm{c}\}$
- $[\![x := y+z]\!]^{\#}\, S$ = remove(S,x) $\cup$ $\{x \geq \mathrm{lb}(S,y)+\mathrm{lb}(S,z),$
  $x \leq \mathrm{ub}(S,y)+\mathrm{ub}(S,z)\}$
- $[\![x := y*c]\!]^{\#}\, S$ = remove(S,x) $\cup$ if c>0 $\{x \geq \mathrm{lb}(S,y)*\mathrm{c},\ x \leq \mathrm{ub}(S,y)*\mathrm{c}\}$
  else $\{x \geq \mathrm{ub}(S,y)*\text{-}\mathrm{c},\ x \leq \mathrm{lb}(S,y)*\text{-}\mathrm{c}\}$
- $[\![x := y*z]\!]^{\#}\, S$ = remove(S,x) $\cup$?

`assume` transformers

- $[\![$**`assume`** $x{=}c]\!]^\#\ S$ = ?
- $[\![$**`assume`** $x{<}c]\!]^\#\ S$ = ?
- $[\![$**`assume`** $x{=}y]\!]^\#\ S$ = ?
- $[\![$**`assume`** $x{\neq}c]\!]^\#\ S$ = ?

assume transformers

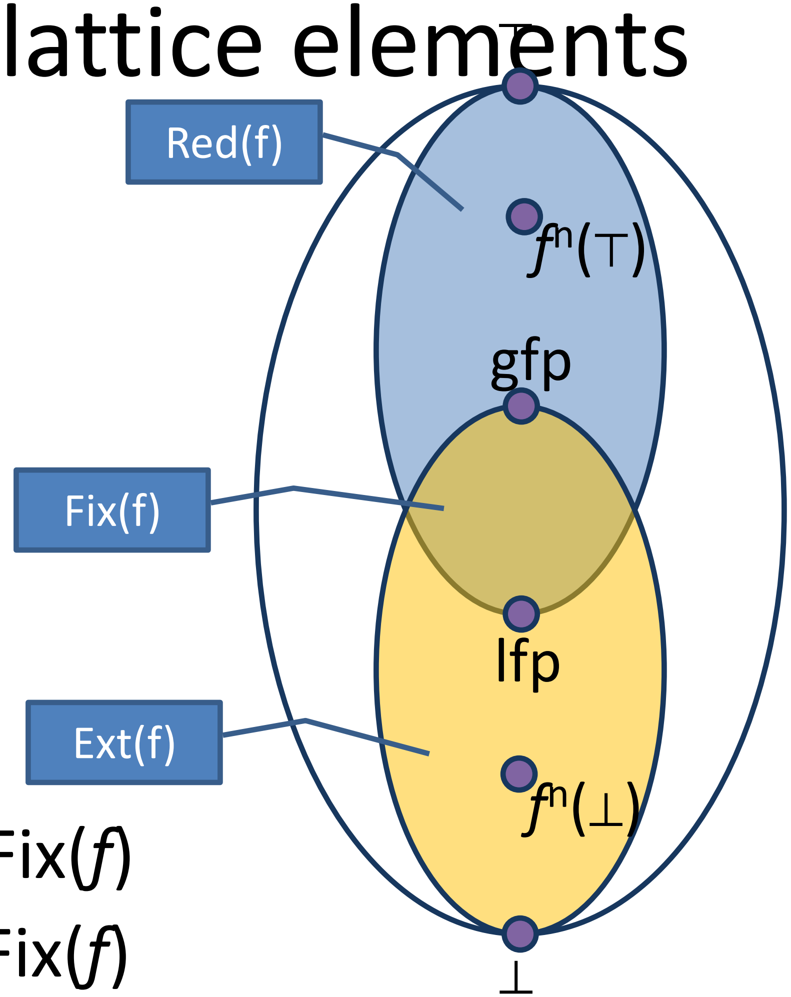
- $[\![\texttt{assume}\ x{=}c]\!]^{\#}\ S = S \sqcap \{x \geq c,\ x \leq c\}$
- $[\![\texttt{assume}\ x{<}c]\!]^{\#}\ S = S \sqcap \{x \leq c\text{-}1\}$
- $[\![\texttt{assume}\ x{=}y]\!]^{\#}\ S = S \sqcap \{x \geq \mathrm{lb}(S,y),\ x \leq \mathrm{ub}(S,y)\}$
- $[\![\texttt{assume}\ x{\neq}c]\!]^{\#}\ S = ?$

`assume` transformers

- $[\![$**`assume`** $x{=}c]\!]^{\#}\ S = S \sqcap \{x{\geq}c,\ x{\leq}c\}$
- $[\![$**`assume`** $x{<}c]\!]^{\#}\ S = S \sqcap \{x{\leq}c\text{-}1\}$
- $[\![$**`assume`** $x{=}y]\!]^{\#}\ S = S \sqcap \{x{\geq}\mathrm{lb}(S,y),\ x{\leq}\mathrm{ub}(S,y)\}$
- $[\![$**`assume`** $x{\neq}c]\!]^{\#}\ S = (S \sqcap \{x{\leq}c\text{-}1\}) \sqcup (S \sqcap \{x{\geq}c{+}1\})$

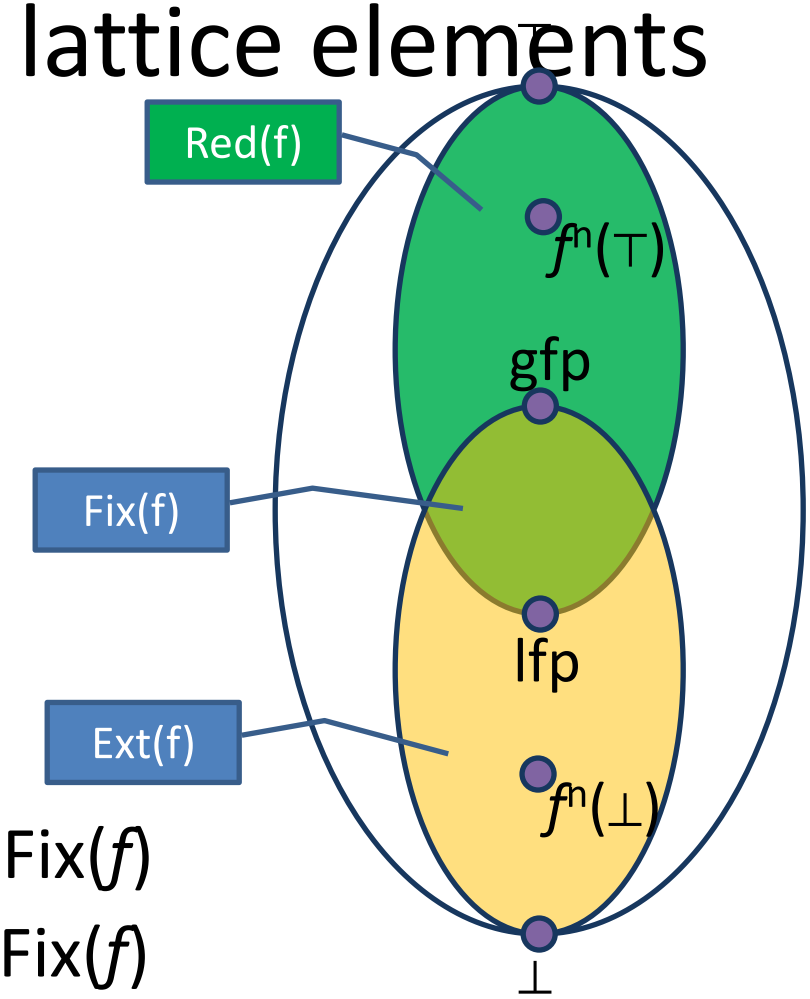
Effect of function f on lattice elements

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$
- $f : D \to D$ **monotone**
- $\mathrm{Fix}(f) = \{ d \mid f(d) = d \}$
- $\mathrm{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\mathrm{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\mathrm{lfp}(f) = \sqcap\mathrm{Fix}(f) = \sqcap\mathrm{Red}(f) \in \mathrm{Fix}(f)$
  - $\mathrm{gfp}(f) = \sqcup\mathrm{Fix}(f) = \sqcup\mathrm{Ext}(f) \in \mathrm{Fix}(f)$



Effect of function f on lattice elements

- $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- $f : D \to D$ **monotone**
- $\text{Fix}(f) = \{ d \mid f(d) = d \}$
- $\text{Red}(f) = \{ d \mid f(d) \sqsubseteq d \}$
- $\text{Ext}(f) = \{ d \mid d \sqsubseteq f(d) \}$
- **Theorem** [Tarski 1955]
  - $\text{lfp}(f) = \sqcap\text{Fix}(f) = \sqcap\text{Red}(f) \in \text{Fix}(f)$
  - $\text{gfp}(f) = \sqcup\text{Fix}(f) = \sqcup\text{Ext}(f) \in \text{Fix}(f)$



Continuity and ACC condition

- Let $L = (D, \sqsubseteq, \sqcup, \bot)$ be a complete partial order
  - Every ascending chain has an upper bound
- A function f is continuous if for every increasing chain $Y \subseteq D^*$,

$$f(\sqcup Y) = \sqcup\{ f(y) \mid y \in Y \}$$

- L satisfies the ascending chain condition (ACC) if every ascending chain eventually stabilizes:

$$d_0 \sqsubseteq d_1 \sqsubseteq \ldots \sqsubseteq d_n = d_{n+1} = \ldots$$

Fixed-point theorem [Kleene]

- Let $L = (D, \sqsubseteq, \sqcup, \perp)$ be a complete partial order and a **continuous** function f: $D \to D$ then

$$\mathrm{lfp}(f) = \bigsqcup_{n \in N} f^n(\perp)$$

Resulting algorithm

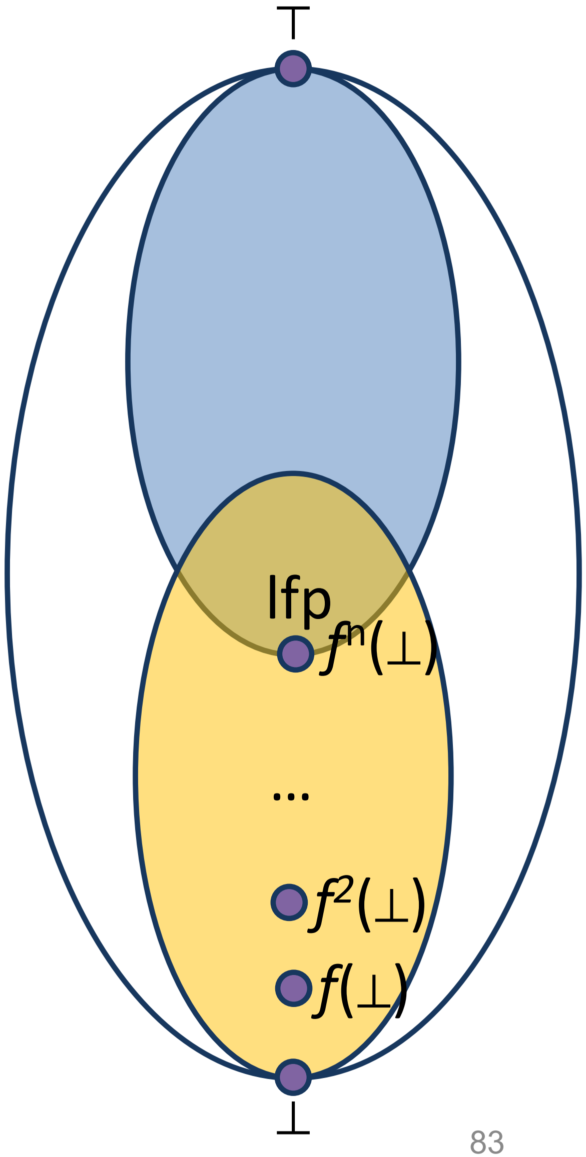
- Kleene's fixed point theorem gives a constructive method for computing the lfp

Mathematical definition

$$\mathrm{lfp}(f) = \bigsqcup_{n \in N} f^n(\bot)$$

Algorithm

$d := \bot$
while $f(d) \neq d$ **do**
  $d := d \sqcup f(d)$
return d



Chaotic iteration

- Input:
  - A cpo $L = (D, \sqsubseteq, \sqcup, \perp)$ satisfying ACC
  - $L^n = L \times L \times \ldots \times L$
  - A monotone function $f : D^n \rightarrow D^n$
  - A system of equations $\{ X[\mathrm{i}] \mid f(X) \mid 1 \leq i \leq n \}$
- Output: lfp(f)
- A worklist-based algorithm

```
for i:=1 to n do
   X[i] := ⊥
WL = {1,…,n}
while WL ≠ ∅ do
   j := pop WL // choose index non-deterministically
   N := F[i](X)
   if N ≠ X[i] then
      X[i] := N
      add all the indexes that directly depend on i to WL
      (X[j] depends on X[i] if F[j] contains X[i])
return X
```

Concrete semantics equations

```
public void loopExample() {
R[0]  int x = 7;  R[1]
R[2]  while (x < 1000) {
R[3]       ++x;  R[4]
      }
R[5]  if (!(x == 1000))
R[6]       error("Unable to prove x == 1000!");
}
```

- $R[0] = \{\mathbf{x} \in \mathbf{Z}\}$
  $R[1] = [\![\mathbf{x:=7}]\!]$
  $R[2] = R[1] \cup R[4]$
  $R[3] = R[2] \cap \{s \mid s(x) < 1000\}$
  $R[4] = [\![\mathbf{x:=x+1}]\!]\ R[3]$
  $R[5] = R[2] \cap \{s \mid s(x) \geq 1000\}$
  $R[6] = R[5] \cap \{s \mid s(x) \neq 1001\}$

Abstract semantics equations

```
public void loopExample() {
R[0]  int x = 7;  R[1]
R[2]  while (x < 1000) {
R[3]        ++x;  R[4]
      }
R[5]  if (!(x == 1000))
R[6]        error("Unable to prove x == 1000!");
}
```

- $R[0] = \alpha(\{\mathbf{x} \in \mathbf{Z}\})$
  $R[1] = [\![\mathbf{x:=7}]\!]^{\#}$
  $R[2] = R[1] \sqcup R[4]$
  $R[3] = R[2] \sqcap \alpha(\{s \mid s(x) < 1000\})$
  $R[4] = [\![\mathbf{x:=x+1}]\!]^{\#}\ R[3]$
  $R[5] = R[2] \sqcap \alpha(\{s \mid s(x) \geq 1000\})$
  $R[6] = R[5] \sqcap \alpha(\{s \mid s(x) \geq 1001\}) \sqcup R[5] \sqcap \alpha(\{s \mid s(x) \leq 999\})$

Abstract semantics equations

```
public void loopExample() {
R[0]  int x = 7; R[1]
R[2]  while (x < 1000) {
R[3]      ++x; R[4]
      }
R[5]  if (!(x == 1000))
R[6]      error("Unable to prove x == 1000!");
}
```

- $R[0] = \top$
  $R[1] = [7,7]$
  $R[2] = R[1] \sqcup R[4]$
  $R[3] = R[2] \sqcap [-\infty,999]$
  $R[4] = R[3] + [1,1]$
  $R[5] = R[2] \sqcap [1000,+\infty]$
  $R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Too many iterations to converge

```
Iteration 3981: processing V[8] = Interval[x==1000](V[6]) // if x == 1000 goto return
              V[8] : false
              V[6] : and(x=1000)
              V[8]' : and(x=1000)
              Adding [V[12] = Join_IntervalDomain(V[8], V[10]) // return]
              workSet = {V[12]}
Iteration 3982: processing V[12] = Join_IntervalDomain(V[8], V[10]) // return
              V[12] : false
              V[8] : and(x=1000)
              V[10] : false
              V[12]' : and(x=1000)
              Adding [V[11] = V[12] // return]
              workSet = {V[11]}
Iteration 3983: processing V[11] = V[12] // return
              V[11] : false
              V[12] : and(x=1000)
              V[11]' : and(x=1000)
              Adding []
Reached fixed-point after 3983 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[5] : and(7<=x<=999)
  V[6] : and(x=1000)
  V[8] : and(x=1000)
  V[9] : false
  V[10] : false
  V[12] : and(x=1000)
  V[11] : and(x=1000)
}
0 possible errors found.
Writing to sootOutput\IntervalExample.jimple
Soot finished on Wed Jun 12 06:24:14 IDT 2013
Soot has run for 0 min. 1 sec.
```

How many iterations for this one?

```
public void loopExample2(int y) {
    int x = 7;
    if (x < y) {
        while (x < y) {
            ++x;
        }

        if (x != y)
            error("Unable to prove x = y!");
    }
}
```

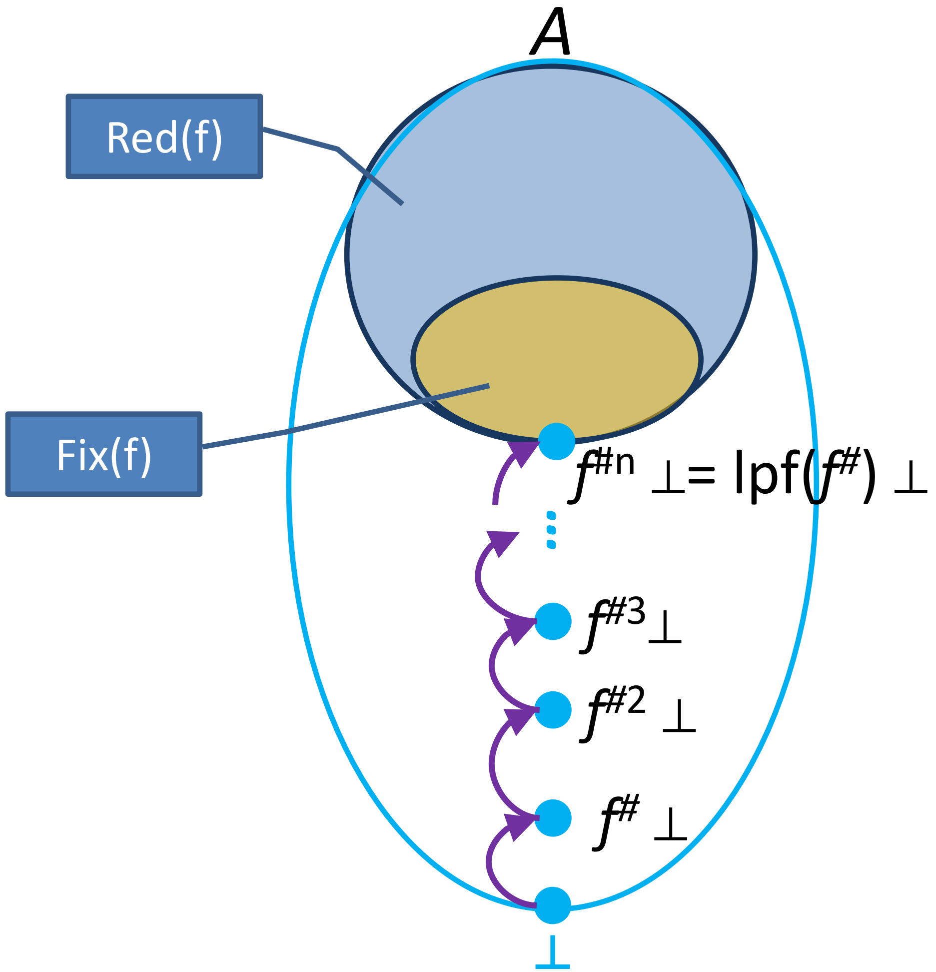
Widening

- Introduce a new binary operator to ensure termination
  – A kind of extrapolation
- Enables static analysis to use infinite height lattices
  – Dynamically adapts to given program
- Tricky to design
- Precision less predictable then with finite-height domains (widening non-monotone)

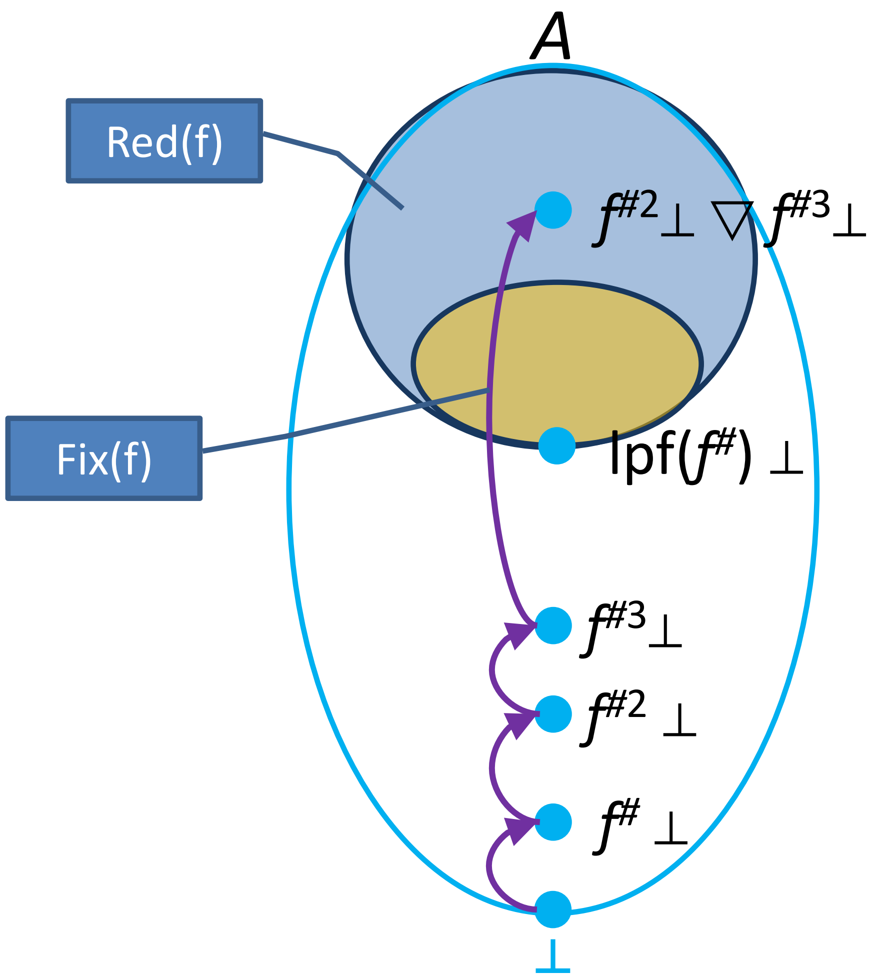
Formal definition

- For all elements $d_1 \sqcup d_2 \sqsubseteq d_1 \triangledown d_2$
- For all ascending chains $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \ldots$ the following sequence is finite
  - $y_0 = d_0$
  - $y_{i+1} = y_i \triangledown d_{i+1}$
- For a monotone function $f : D \rightarrow D$ define
  - $x_0 = \bot$
  - $x_{i+1} = x_i \triangledown f(x_i)$
- Theorem:
  - There exits k such that $x_{k+1} = x_k$
  - $x_k \in \mathrm{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$

Analysis with finite-height lattice



Analysis with widening



Widening for Intervals Analysis

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

  if $a \leq c$

  then a

  else $-\infty$,

  if $b \geq d$

  then b

  else ∞

Semantic equations with widening

```
public void loopExample() {
R[0]  int x = 7;  R[1]
R[2]  while (x < 1000) {
R[3]       ++x;  R[4]
      }
R[5]  if (!(x == 1000))
R[6]       error("Unable to prove x == 1000!");
}
```

- $R[0] = \top$
  $R[1] = [7,7]$
  $R[2] = R[1] \sqcup R[4]$
  $R[2.1] = R[2.1] \triangledown R[2]$
  $R[3] = R[2.1] \sqcap [-\infty,999]$
  $R[4] = R[3] + [1,1]$
  $R[5] = R[2] \sqcap [1001,+\infty]$
  $R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Choosing analysis with widening

```java
/**
 * Adds the Interval analysis transform to Soot.
 *
 * @author romanm
 */
public class IntervalMain {
    public static void main(String[] args) {
        PackManager
                .v()
                .getPack("jtp")
                .add(new Transform("jtp.IntervalAnalysis",
                        new IntervalAnalysis()));
        soot.Main.main(args);
    }

    public static class IntervalAnalysis extends BaseAnalysis<IntervalState> {
        public IntervalAnalysis() {
            super(new IntervalDomain());
            useWidening(true);
        }
    }
}
```

Enable widening

Non monotonicity of widening

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

Non monotonicity of widening

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

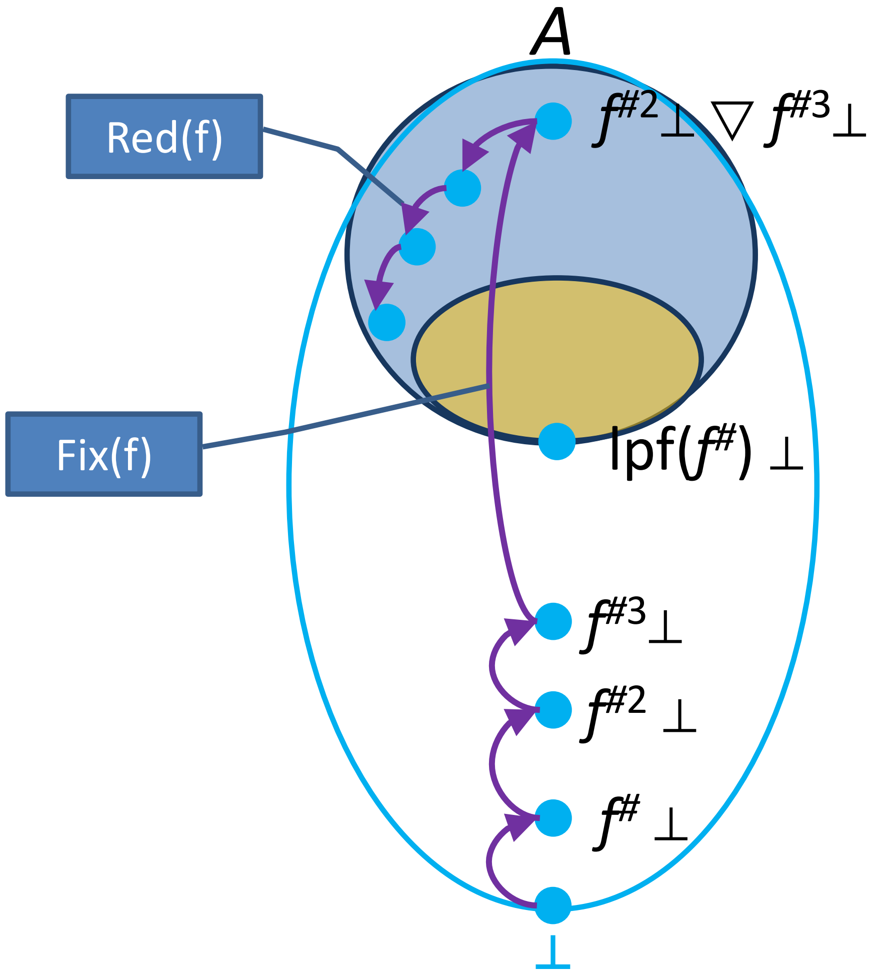
Analysis results with widening

```
Analyzing method loopExample
------------------------------------------
Solving the following equation system =
V[0] = true // this := @this: IntervalExample
V[1] = AssignTopTransformer(V[0]) // this := @this: IntervalExample
V[2] = AssignConstantToVarTransformer(V[1]) // x = 7
V[3] = V[2] // goto [?= (branch)]
V[4] = AssignAddExprToVarTransformer(V[5]) // x = x + 1
V[7] = JoinLoop_IntervalDomain(V[3], V[4]) // if x < 1000 goto x = x + 1
V[8] = IntervalDomain[Widening|Narrowing](V[8], V[7]) // if x < 1000 goto x = x + 1
V[5] = Interval[x<1000](V[8]) // if x < 1000 goto x = x + 1
V[6] = Interval[x>=1000](V[8]) // if x < 1000 goto x = x + 1
V[9] = Interval[x==1000](V[6]) // if x == 1000 goto return
V[10] = Interval[x!=1000](V[6]) // if x == 1000 goto return
V[11] = V[10] // specialinvoke this.<IntervalExample: void error(java.lang.String)>("Unable to prove x == 1000!")
V[13] = Join_IntervalDomain(V[9], V[11]) // return
V[12] = V[13] // return
```

```
Reached fixed-point after 23 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(x>=7)
  V[5] : and(7<=x<=999)
  V[6] : and(x>=1000)
  V[9] : and(x=1000)
  V[10] : and(x>=1001)
  V[11] : and(x>=1001)
  V[13] : and(x>=1000)
  V[12] : and(x>=1000)
}
```

Did we prove it?

Analysis with narrowing



Formal definition of narrowing

- Improves 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 monotone function f: $D \rightarrow D$ and $x_k \in \mathrm{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$ define
  - $y_0 = x$
  - $y_{i+1} = y_i \triangle f(y_i)$
- Theorem:
  - There exits k such that $y_{k+1} = y_k$
  - $y_k \in \mathrm{Red}(f) = \{ d \mid d \in D \text{ and } f(d) \sqsubseteq d \}$

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

  $]$

Semantic equations with narrowing

```
public void loopExample() {
R[0]  int x = 7;  R[1]
R[2]  while (x < 1000) {
R[3]        ++x;  R[4]
      }
R[5]  if (!(x == 1000))
R[6]        error("Unable to prove x == 1000!");
}
```

- $R[0] = \top$
  $R[1] = [7,7]$
  $R[2] = R[1] \sqcup R[4]$
  $R[2.1] = R[2.1] \triangle R[2]$
  $R[3] = R[2.1] \sqcap [-\infty,999]$
  $R[4] = R[3]+[1,1]$
  $R[5] = R[2]^{\#} \sqcap [1000,+\infty]$
  $R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Analysis with widening/narrowing

- Two phases
  - Phase 1: analyze with widening until converging
  - Phase 2: use values to analyze with narrowing

```
public void loopExample() {
    int x = 7;
    while (x < 1000) {
        ++x;
    }
    if (!(x == 1000))
        error("Unable to prove x == 1000!");
}
```

Phase 1:

$R[0] = \top$

$R[1] = [7,7]$

$R[2] = R[1] \sqcup R[4]$

$R[2.1] = R[2.1] \triangledown R[2]$

$R[3] = R[2.1] \sqcap [-\infty,999]$

$R[4] = R[3] + [1,1]$

$R[5] = R[2] \sqcap [1001,+\infty]$

$R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Phase 2:

$R[0] = \top$

$R[1] = [7,7]$

$R[2] = R[1] \sqcup R[4]$

$R[2.1] = R[2.1] \vartriangle R[2]$

$R[3] = R[2.1] \sqcap [-\infty,999]$

$R[4] = R[3]+[1,1]$

$R[5] = R[2]^{\#} \sqcap [1000,+\infty]$

$R[6] = R[5] \sqcap [999,+\infty] \sqcup R[5] \sqcap [1001,+\infty]$

Analysis with widening/narrowing

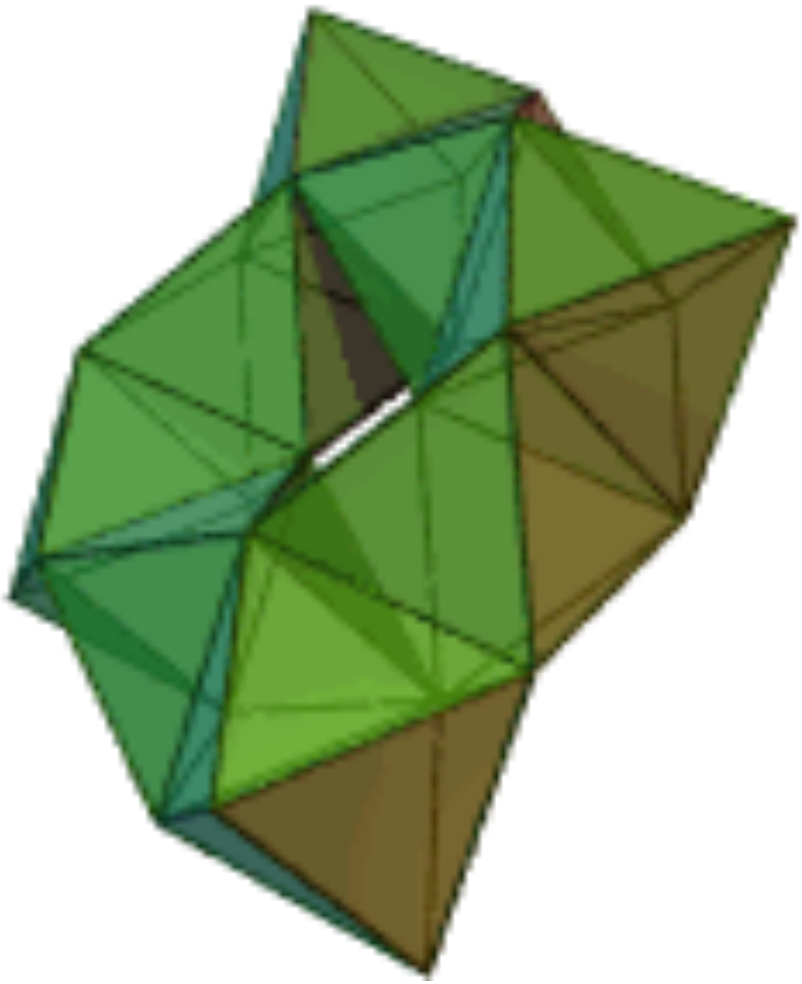
```
Reached fixed-point after 23 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(x>=7)
  V[5] : and(7<=x<=999)
  V[6] : and(x>=1000)
  V[9] : and(x=1000)
  V[10] : and(x>=1001)
  V[11] : and(x>=1001)
  V[13] : and(x>=1000)
  V[12] : and(x>=1000)
}

Starting chaotic iteration: narrowing phase...
              workSet = {V[0], V[1], V[2], V[3], V[4], V[7], V[8], V[5], V[6], V[9], V[10], V[11], V[13], V[12]}
Iteration 24: processing V[0] = true // this := @this: IntervalExample
              V[0] : true
              V[0]' : true
              workSet = {V[12], V[1], V[2], V[3], V[4], V[7], V[8], V[5], V[6], V[9], V[10], V[11], V[13]}
```

Analysis results widening/narrowing

```
Iteration 44: processing V[1] = AssignTopTransformer(V[0]) // this := @this: IntervalExample
              V[1] : true
              V[0] : true
              V[1]' : true
Reached fixed-point after 44 iterations.
Solution = {
  V[0] : true
  V[1] : true
  V[2] : and(x=7)
  V[3] : and(x=7)
  V[4] : and(8<=x<=1000)
  V[7] : and(7<=x<=1000)
  V[8] : and(7<=x<=1000)
  V[5] : and(7<=x<=999)
  V[6] : and(x=1000)
  V[9] : and(x=1000)
  V[10] : false
  V[11] : false
  V[13] : and(x=1000)
  V[12] : and(x=1000)
}
0 possible errors found.
Writing to sootOutput\IntervalExample.jimple
Soot finished on Wed Jun 12 06:47:24 IDT 2013
Soot has run for 0 min. 0 sec.
```

Precise invariant



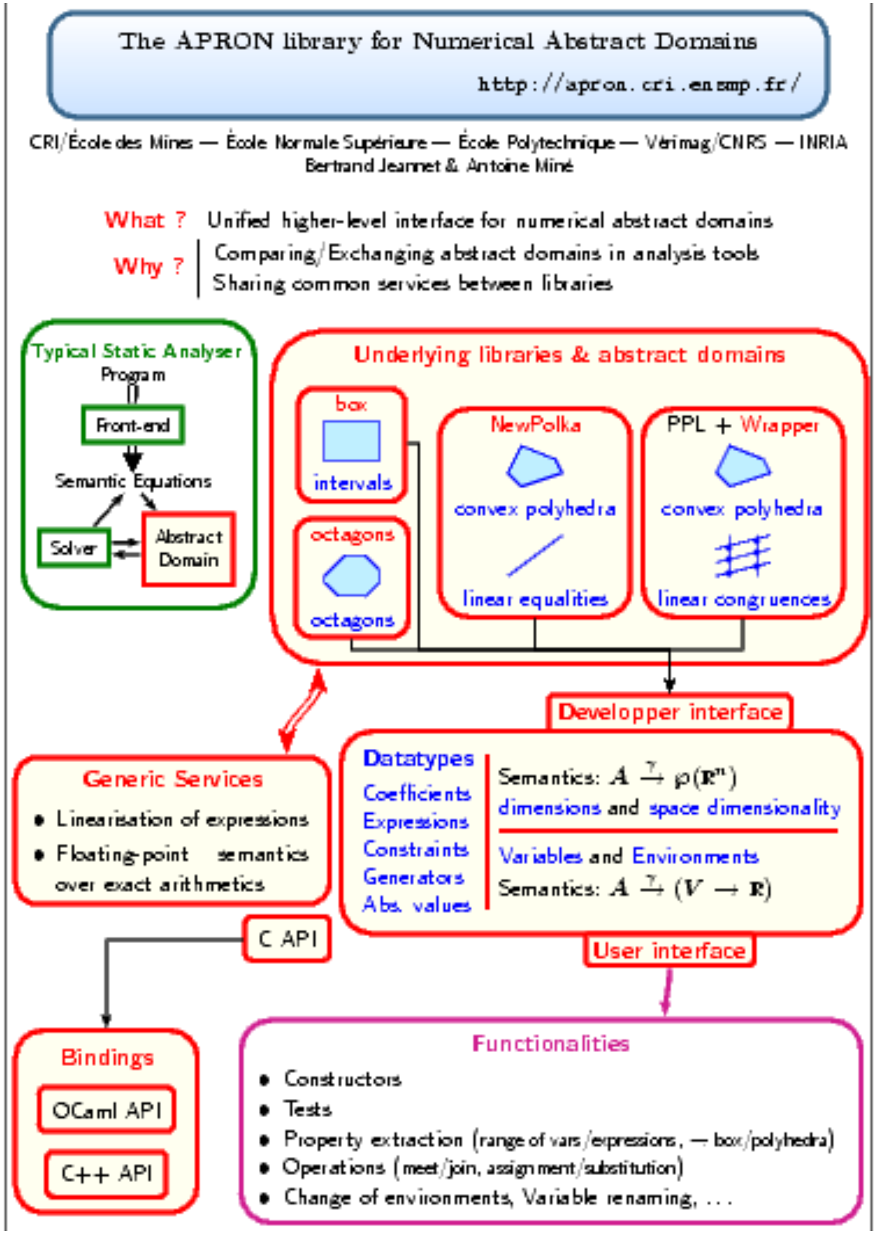
Numerical Abstractions

By Quilbert (own work, partially derived from en:Image:Poly.pov) [GPL (http://www.gnu.org/licenses/gpl.html)], via Wikimedia Commons

Overview

- Goal: infer numeric properties of program variables (integers, floating point)
- Applications
  - Detect division by zero, overflow, out-of-bound array access
  - Help non-numerical domains
- Classification
  - Non-relational
  - (Weakly-)relational
  - Equalities / Inequalities
  - Linear / non-linear
  - Exotic

Implementation

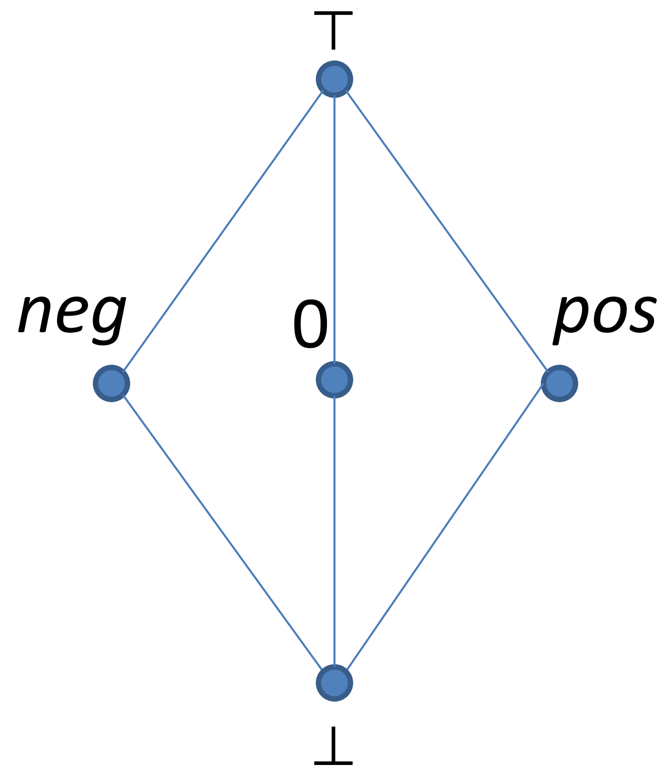


Non-relational abstractions

- Abstract each variable individually
  - Constant propagation [Kildall'73]
  - Sign
  - Parity (congruences)
  - Intervals (Box)

Sign abstraction for variable *x*

- Concrete lattice: $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- $Sign = \{\bot, neg, 0, pos, \top\}$
- $GC^{C,Sign}=(C, \alpha, \gamma, Sign)$
- $\gamma(\bot) = ?$
- $\gamma(neg) = ?$
- $\gamma(0) = ?$
- $\gamma(pos) = ?$
- $\gamma(\top) = ?$
- How can we represent ≥ 0?

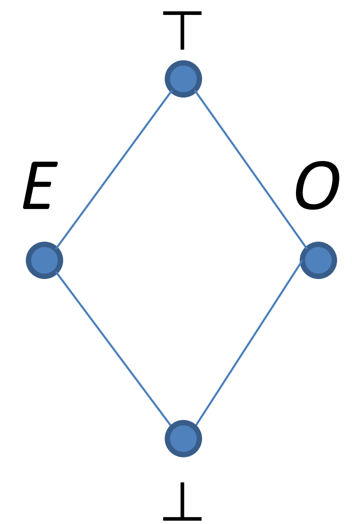


Transformer x:=y+z

	$\perp$	neg	0	pos	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
neg	$\perp$	neg	neg	$\top$	$\top$
0	$\perp$	neg	0	pos	$\top$
pos	$\perp$	$\top$	pos	pos	$\top$
$\top$	$\perp$	$\top$	$\top$	$\top$	$\top$

Parity abstraction for variable x

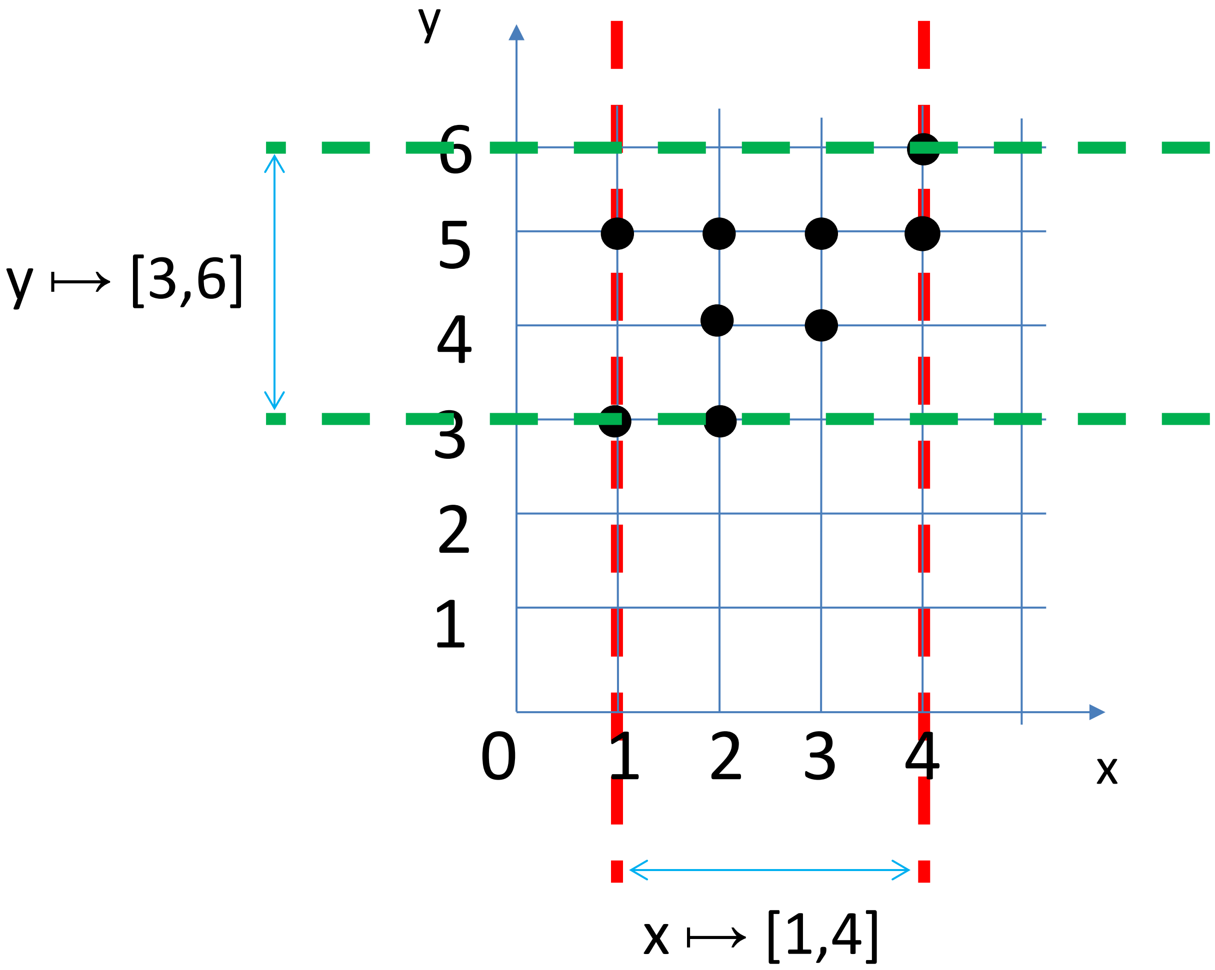
- Concrete lattice: $C = (2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$
- $Parity = \{\bot, E, O, \top\}$
- $GC^{C,Parity} = (C, \alpha, \gamma, Parity)$
- $\gamma(\bot) = ?$
- $\gamma(E) = ?$
- $\gamma(O) = ?$
- $\gamma(\top) = ?$



Transformer x:=y+z

	$\perp$	E	O	$\top$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
E	$\perp$	E	O	$\top$
O	$\perp$	O	E	$\top$
$\top$	$\perp$	$\top$	$\top$	$\top$

Boxes (intervals)



Non-relational abstractions

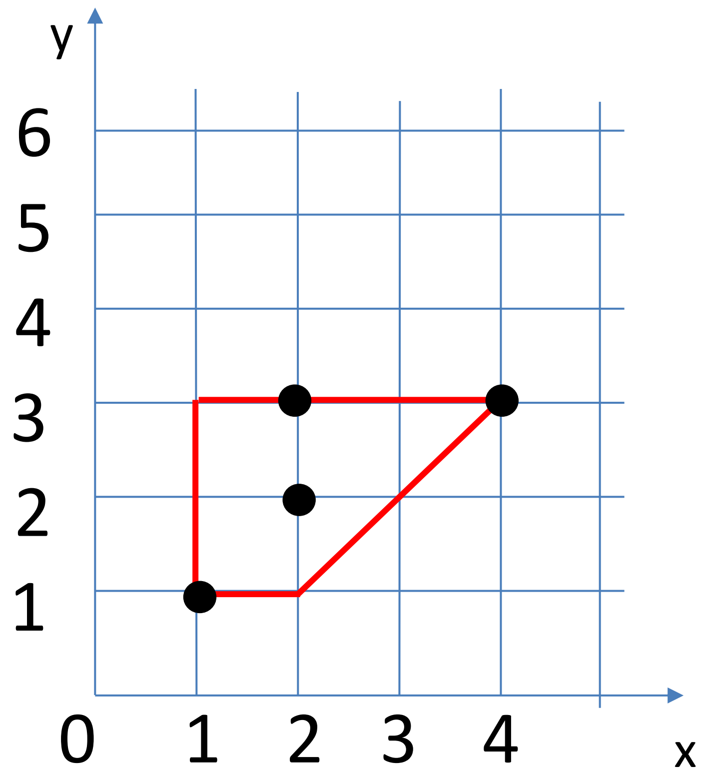
- Cannot prove properties that hold simultaneous for several variables
  - $x = 2*y$
  - $x \leq y$

```
public void loopExample2() {
    int x = 7;
    int y = x;
    while (x < 1000) {
        ++x;
        ++y;
    }
    if (!(y == 1000))
        error("Unable to prove y == 1000!");
}
```

Zone abstraction [Mine]

- Maintain bounded differences between a pair of program variables (useful for tracking array accesses)
- Abstract state is a conjunction of linear inequalities of the form x-y≤c

$$\begin{cases} x \le 4 \\ -x \le -1 \\ y \le 3 \\ -y \le -1 \\ x - y \le 1 \end{cases}$$



Difference bound matrices

- Add a special V0 variable for the number 0
- Represent non-existent relations between variables by $+\infty$ entries
- Convenient for defining the partial order between two abstract elements… $\sqsubseteq$=?

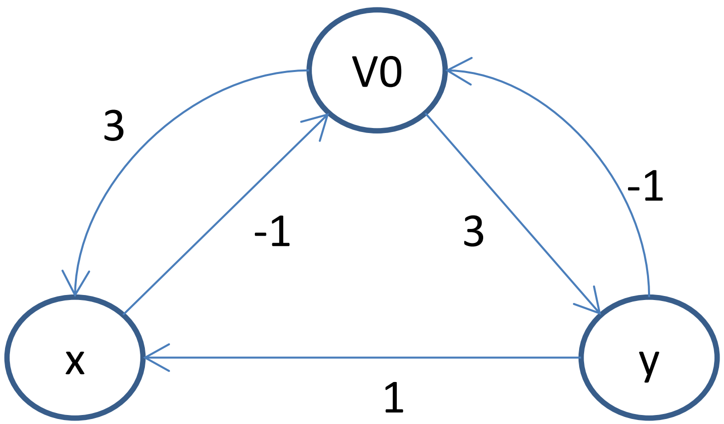
$$\begin{cases} x \le 4 \\ -x \le -1 \\ y \le 3 \\ -y \le -1 \\ x - y \le 1 \end{cases}$$

	V0	x	y
V0	$+\infty$	4	3
x	-1	$+\infty$	$+\infty$
y	-1	1	$+\infty$

Potential graph

- A vertex per variable
- A directed edge with the weight of the inequality
- Enables computing semantic reduction by shortest-path algorithms

$$
\begin{cases}
x \le 4 \\
-x \le -1 \\
y \le 3 \\
-y \le -1 \\
x - y \le 1
\end{cases}
$$



Can we tell whether a system of constraints is satisfiable?

Semantic reduction for zones

- Apply the following rule repeatedly

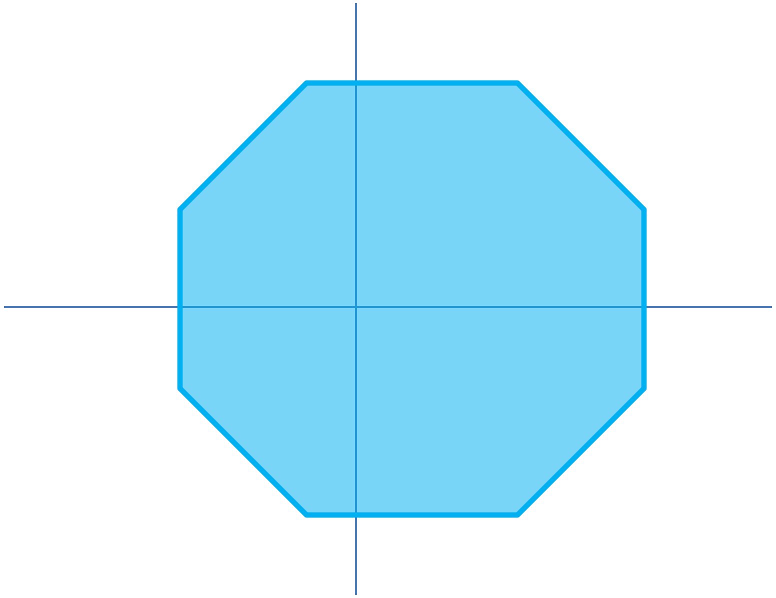
  $$\frac{x - y \leq c \qquad y - z \leq d}{x - z \leq c+d}$$

- When should we stop?
- Theorem 3.3.4. Best abstraction of potential sets and zones

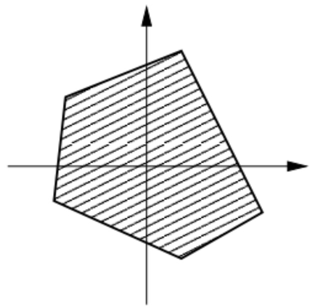
  $m* = (\alpha^{\text{Pot}} \circ \gamma^{\text{Pot}})(m)$

Octagon abstraction [Mine-01]

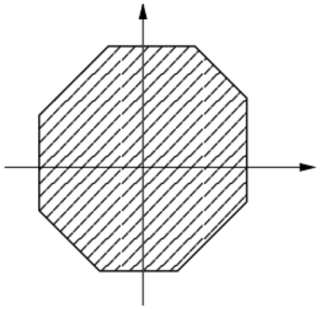
- Abstract state is an intersection of linear inequalities of the form $\pm x \pm y \leq c$



Some inequality-based relational domains



policy iteration



Polyhedra

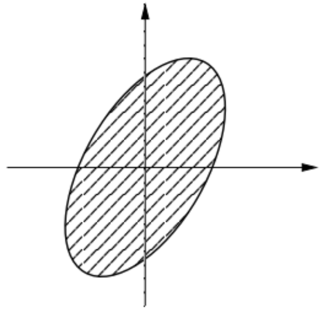
$\sum_i \alpha_i X_i \geq \beta$

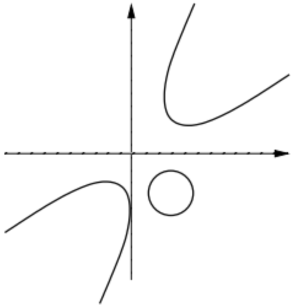
[Cousot-Halbwachs-78]

Octagons

$\pm X_i \pm X_j \leq \beta$

[Miné-01]





Ellipsoids

$X^2 + \beta Y^2 + \gamma XY \leq \delta$

[Feret-04]

Varieties

$P(\vec{X}) = 0,\ P \in \mathbb{R}[\mathrm{Var}]$

[Sankaranarayanan-Sipma-Man

Equality-based domains

- Simple congruences [Granger'89]: y=a mod k
- **Linear relations:** y=a*x+b
  - Join operator a little tricky
- Linear equalities [Karr'76]: $a_1*x_1+\ldots+a_k*x_k = c$
- Polynomial equalities:
  $a_1*x_1^{d1}*\ldots*x_k^{dk} + b_1*y_1^{z1}*\ldots*y_k^{zk} + \ldots = c$
  - Some good results are obtainable when $d_1+\ldots+d_k < n$ for some small n