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

Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis

  - Abstract (semantic) domains ("abstract states")
  - Transformer functions ("abstract steps")
  - Chaotic iteration ("abstract computation")

Abstract Interpretation [CC77]

- A very general mathematical framework for approximating semantics
  - Generalizes Hoare Logic
  - Generalizes weakest precondition calculus
- Allows designing sound static analysis algorithms
  - Usually compute by iterating to a fixed-point
  - *Not specific to any programming language style*
- Results of an abstract interpretation are (loop) invariants
  - Can be interpreted as axiomatic verification assertions and used for verification

Abstract Interpretation in 5 Slides

- Disclaimer
  - Do not worry if you feel that you do not understand the next 5 slides
    - You are not expected to …
  - This is just to give you a view of the land …

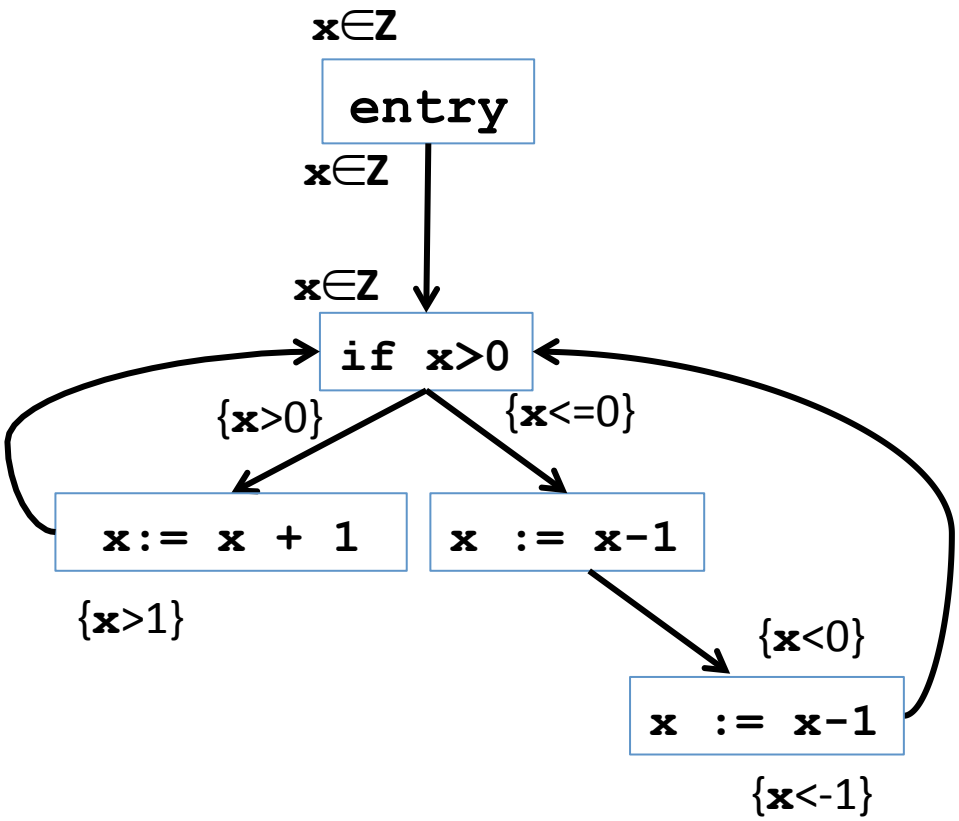
Collecting semantics

- For a set of program states **State**, we define the collecting lattice

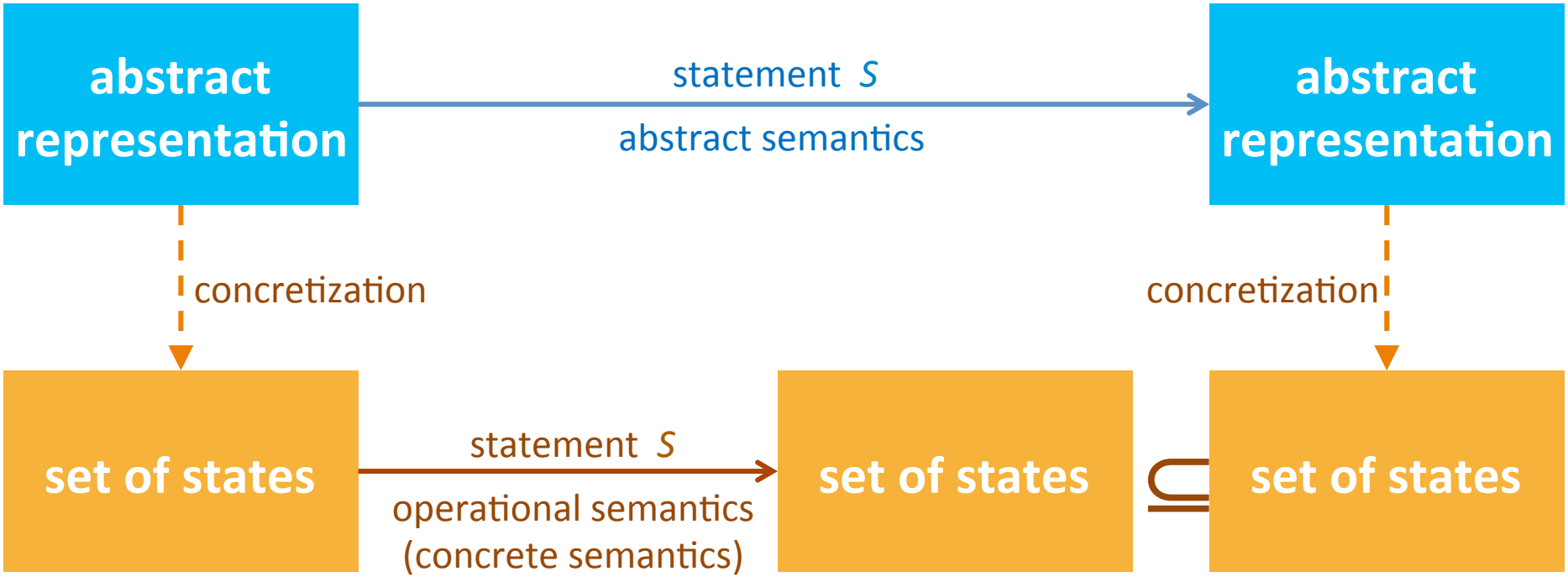
  $(2^{\mathbf{State}}, \subseteq, \cup, \cap, \varnothing, \mathbf{State})$

- The collecting semantics accumulates the (possibly infinite) sets of states generated during the execution
  - Not computable in general

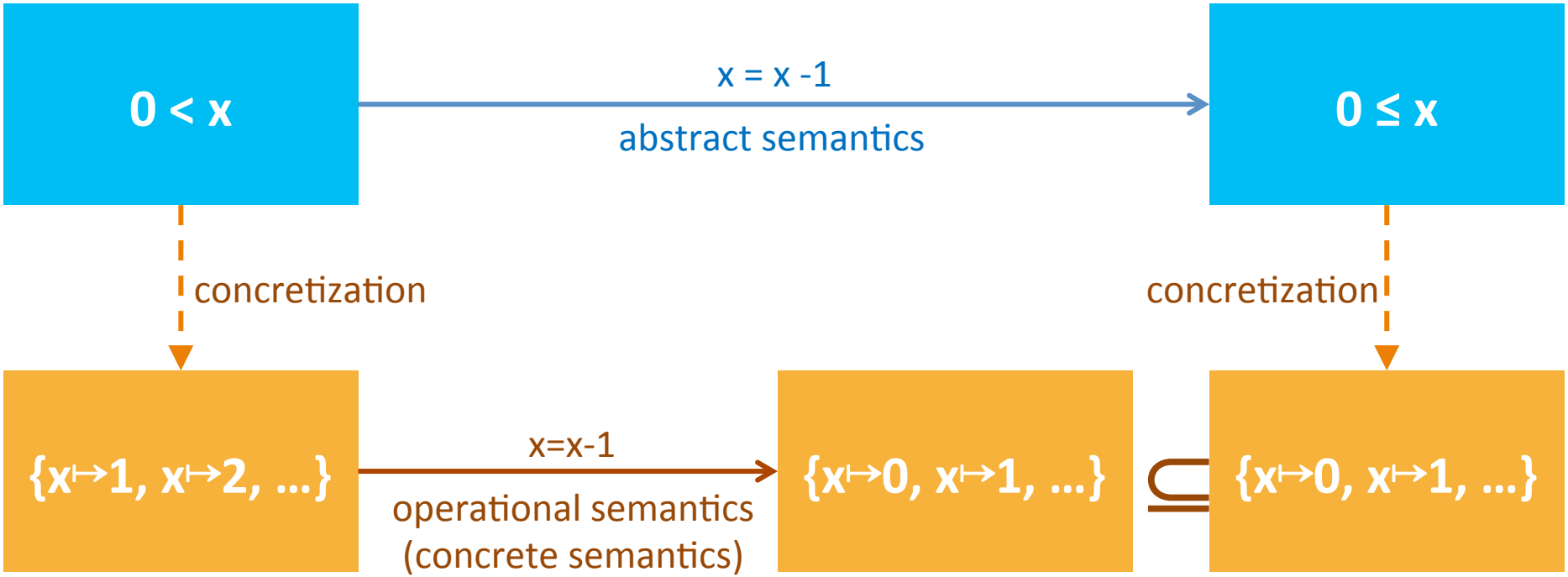
“Proof”



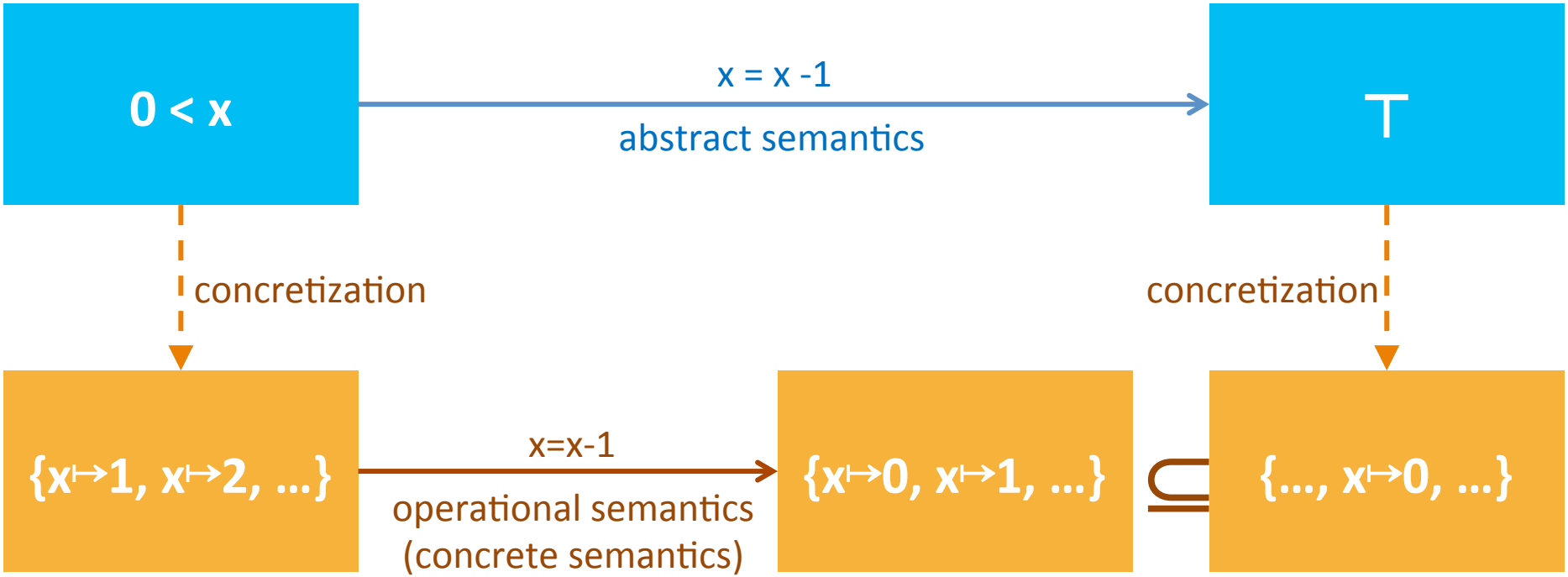
Abstract (conservative) interpretation



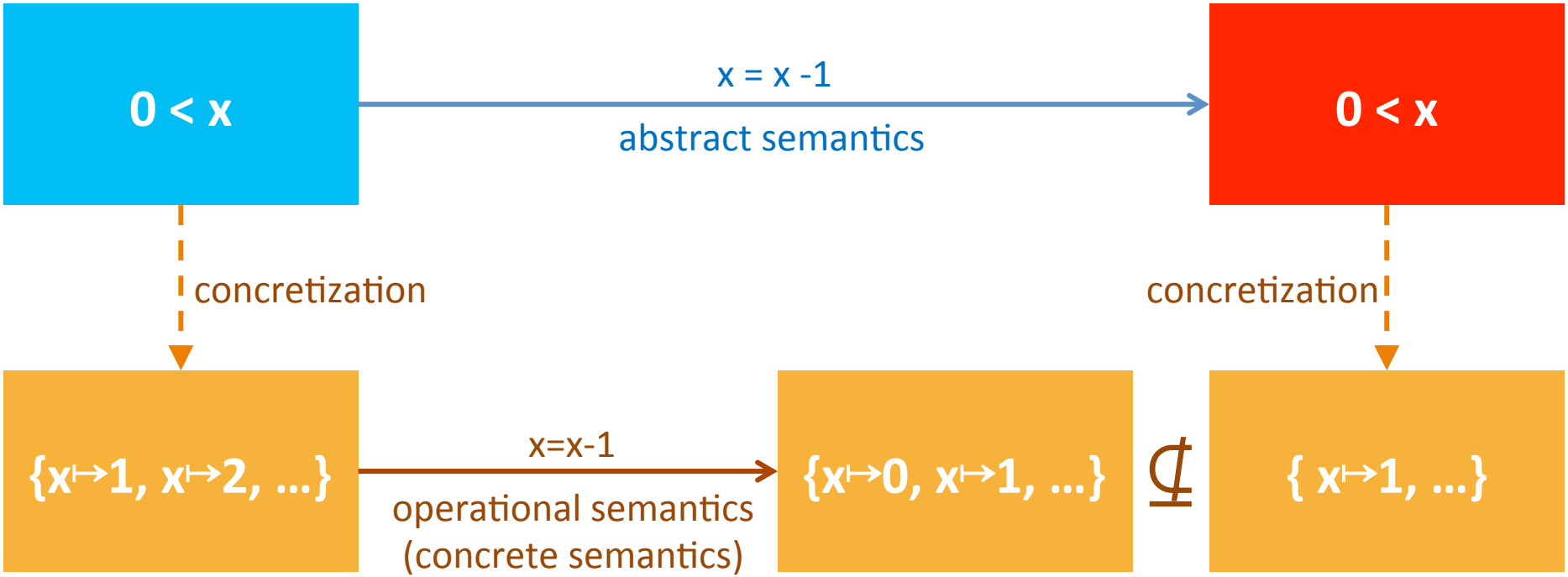
Abstract (conservative) interpretation



Abstract (conservative) interpretation



Abstract (non-conservative) interpretation



Abstract Interpretation by Example

Available Expressions Analysis

- A static analysis that infers for every program point a set of facts of the form
  $AV = \{ x = y \mid x, y \in \text{Var} \} \cup$
  $\{ x = -y \mid x, y \in \text{Var} \} \cup$
  $\{ x = y \; op \; z \mid y, z \in \text{Var}, op \in \{+, -, *, <=\} \}$

- For every program with n=|Var| variables number of possible facts is finite: $|AV| = O(n^3)$
  - Yields a trivial algorithm … but, is it efficient?

What do we need to prove?

```
{ true }
C1
x := a op b
C2
{ x = a op b }
y := a op b
C3
```


```
{ true }
C1
x := a op b
C2
{ x = a op b }
y := x
C3
```

Developing a theory of approximation

- Formulae are suitable for many analysis-based proofs but we may want to represent predicates in other ways:
  - Sets of “facts”
  - Automata
  - Linear (in)equalities
  - … ad-hoc representation

- Wanted: a uniform theory to represent semantic values and approximations

Preorder

- We say that a binary order relation $\sqsubseteq$ over a set D is a preorder if the following conditions hold for every d, d', d'' $\in$ D
  - Reflexive: $d \sqsubseteq d$
  - Transitive: $d \sqsubseteq d'$ and $d' \sqsubseteq d''$ implies $d \sqsubseteq d''$

- There may exist d, d' such that
  $d \sqsubseteq d'$ and $d' \sqsubseteq d$ yet $d \neq d'$

Partial order

- A binary order relation $\sqsubseteq$ over a set D is a partial order if
  - $\sqsubseteq$ is a preorder
  - $\sqsubseteq$ is anti-symmetric

- For any d, d' if
  $d \sqsubseteq d'$ and $d' \sqsubseteq d$ then $d = d'$
- Notation: if $d \sqsubseteq d'$ and $d \neq d'$ we write $d \sqsubset d'$

Some posets-related terminology

- If x $\sqsubseteq$ y we can say
  - x is *lower* than y
  - x is *more precise* than y
  - x is *more concrete* than y
  - x *under-approximates* y

  - y is *greater* than x
  - y is *less precise* than x
  - y is *more abstract* than x
  - y *over-approximates* x

Intuition

- $\{x > 0\} \sqsubseteq x \in \mathbb{Z}$

Pointed poset

- A poset $(D, \sqsubseteq)$ with a least element $\bot$ is called a pointed poset, and denoted by $(D , \sqsubseteq, \bot)$

- For all $d \in D$ we have that $\bot \sqsubseteq d$

- We can always transform a poset $(D, \sqsubseteq)$ into a pointed poset by adding a special bottom element

$$(D \cup \{\bot\}, \sqsubseteq \cup \{\bot \sqsubseteq d \mid d \in D\}, \bot)$$

Join operator

- Assume a poset $(D, \sqsubseteq)$
- Let $X \subseteq D$ be a subset of D (finite/infinite)
- The join of X is defined as
  - $\bigsqcup X$ = the least upper bound (LUB) of all elements in X *if it exists*
  - $\bigsqcup X = \min_{\sqsubseteq}\{ b \mid \text{forall } x \in X \text{ we have that } x \sqsubseteq b\}$
  - The supremum of the elements in X
  - A kind of abstract union (disjunction) operator
- Properties of a join operator
  - Commutative: $x \sqcup y = y \sqcup x$
  - Associative: $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$
  - Idempotent: $x \sqcup x = x$

Join operator

- Assume a poset $(D, \sqsubseteq)$
- Let $X \subseteq D$ be a subset of D (finite/infinite)
- The join of X is defined as
  - $\bigsqcup X$ = the least upper bound (LUB) of all elements in X *if it exists*
  - $\bigsqcup X = \min_{\sqsubseteq}\{ b \mid \text{forall } x \in X \text{ we have that } x \sqsubseteq b\}$
  - The supremum of the elements in X
  - A kind of abstract union (disjunction) operator

Meet operator

- Assume a poset $(D, \sqsubseteq)$
- Let $X \subseteq D$ be a subset of D (finite/infinite)
- The meet of X is defined as
  - $\sqcap X$ = the greatest lower bound (GLB) of all elements in X *if it exists*
  - $\sqcap X = \max_{\sqsubseteq}\{ b \mid \text{forall } x \in X \text{ we have that } b \sqsubseteq x\}$
  - The infimum of the elements in X
  - A kind of abstract intersection (conjunction) operator
- Properties of a join operator
  - Commutative: $x \sqcap y = y \sqcap x$
  - Associative: $(x \sqcap y) \sqcap z = x \sqcap (y \sqcap z)$
  - Idempotent: $x \sqcap x = x$

Meet operator

- Assume a poset $(D, \sqsubseteq)$
- Let $X \subseteq D$ be a subset of D (finite/infinite)
- The meet of X is defined as
  - $\sqcap X$ = the greatest lower bound (GLB) of all elements in X *if it exists*
  - $\sqcap X = \max_{\sqsubseteq}\{ b \mid \text{forall } x \in X \text{ we have that } b \sqsubseteq x\}$
  - The infimum of the elements in X
  - A kind of abstract intersection (conjunction) operator

Complete lattices

- A complete lattice $(D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$ is
- A set of elements D
- A partial order $x \sqsubseteq y$
- A join operator $\sqcup$
- A meet operator $\sqcap$
- A bottom element
  $\bot = \sqcup \varnothing = \sqcap D$
- A top element
  $\top = \sqcup D = \sqcap \varnothing$

Transfer Functions

- Mathematical foundations

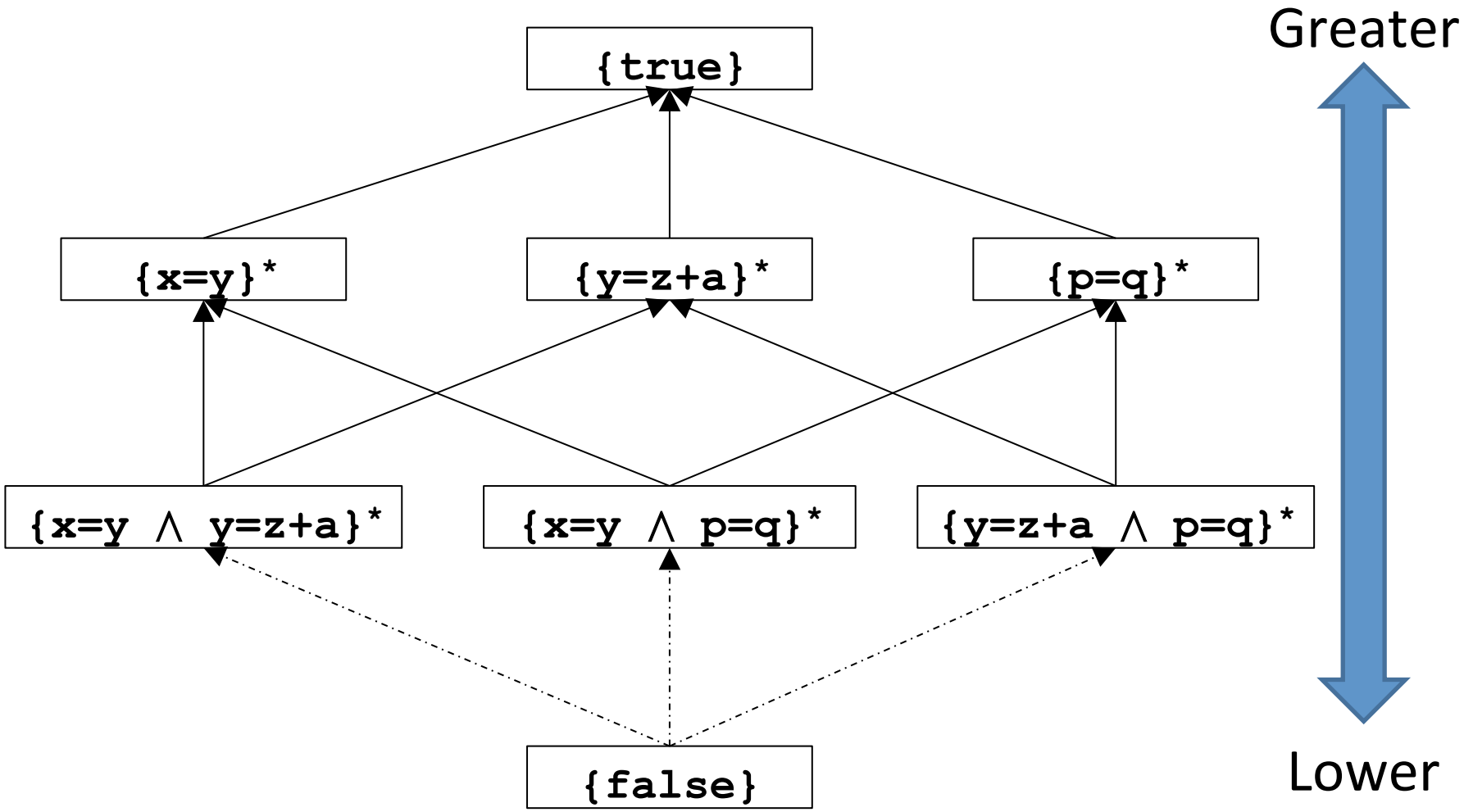
Towards an automatic proof

- **Goal:** automatically compute an annotated program proving as many facts of the form $x = y + z$ as possible
- **Decision 1:** develop a forward-going proof
- **Decision 2:** draw predicates from a finite set D
  - "looking under the light of the lamp"
  - A compromise that simplifies problem by focusing attention – possibly miss some facts that hold
- **Challenge 1:** handle straight-line code
- **Challenge 2:** handle conditions
- **Challenge 3:** handle loops

Domain for SAV

- Define *atomic facts* (for SAV) as
  $\theta = \{ x = y \mid x, y \in \text{Var} \} \cup \{ x = y + z \mid x, y, z \in \text{Var} \}$
  - For n=|Var| number of atomic facts is $O(n^3)$
- Define *sav-predicates* as $\Pi = 2^{\theta}$
- For $D \subseteq \theta$, Conj(D) = $\bigwedge D$
  - Conj(*{a=b, c=b+d, b=c}*) = (a=b) $\wedge$ (c=b+d) $\wedge$ (b=c)
- Note:
  - $\text{Conj}(D_1 \cup D_2) = \text{Conj}(D_1) \wedge \text{Conj}(D_1)$
  - Conj({}) $\Leftrightarrow$ true

Visualizing ordering for SAV



D={x=y, y=x, p=q, q=p, y=z+a, y=a+z, z=y+z, x=z+a}

An algorithm for annotating SLP

- Annotate(P, x:=a) = $\{P\}\ x{:=}a\ \mathsf{F}^*[x{:=}a](P)$

- Annotate(P, S_1; S_2) = $\{P\}\ \mathsf{S}_1; \{Q_1\}\ \mathsf{S}_2\ \{Q_2$
  - Annotate(P, S_1) = $\{P\}\ \mathsf{S}_1\ \{Q_1\}$
  - Annotate(Q_1, S_2) = $\{Q_1\}\ \mathsf{S}_2\ \{Q_2\}$

handling conditions: Goal

$$[\text{if}_p]\quad \frac{\{\, b \wedge P \,\}\, S_1\, \{\, Q \,\},\quad \{\, \neg b \wedge P \,\}\, S_2\, \{\, Q \,\}}{\{\, P \,\}\ \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\, Q \,\}}$$

- Annotate a program
  `if` b `then` S_1 `else` S_2
  with predicates from Π

$\{\, P \,\}$
`if` b `then`
  $\{\, b \wedge P \,\}$
  S_1
  $\{\, Q_1 \,\}$
`else`
  $\{\, \neg b \wedge P \,\}$
  S_2
  $\{\, Q_2 \,\}$
$\{\, Q \,\}$

handling conditions: Goal

$$[\text{if}_p]\quad \frac{\{\, b \wedge P \,\}\, S_1\, \{\, Q \,\},\quad \{\, \neg b \wedge P \,\}\, S_2\, \{\, Q \,\}}{\{\, P \,\}\ \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\, Q \,\}}$$

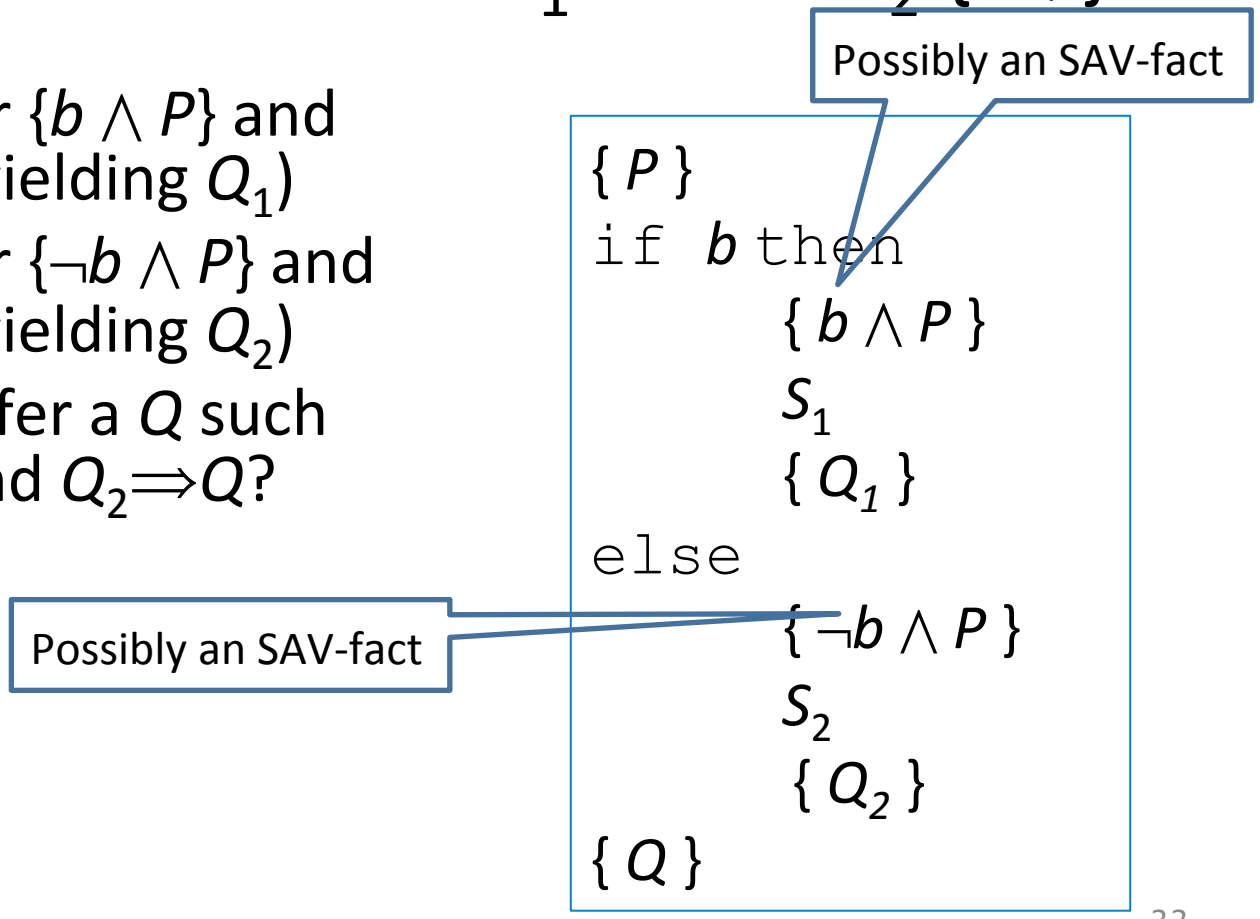
- Annotate a program
  `if` b `then` S_1 `else` S_2
  with predicates from Π
- **Assumption 1:** P is given
  (otherwise use true)
- **Assumption 2:** b is a simple binary expression
  e.g., x=y, x≠y, x<y (why?)

$\{\, P \,\}$
`if` b `then`
  $\{\, b \wedge P \,\}$
  S_1
  $\{\, Q_1 \,\}$
`else`
  $\{\, \neg b \wedge P \,\}$
  S_2
  $\{\, Q_2 \,\}$
$\{\, Q \,\}$

Annotating conditions

$$[\text{if}_p]\quad \frac{\{\, b \wedge P \,\}\, S_1\, \{\, Q \,\},\quad \{\, \neg b \wedge P \,\}\, S_2\, \{\, Q \,\}}{\{\, P \,\}\ \texttt{if}\ \ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\, Q \,\}}$$

1. Start with P or $\{b \wedge P\}$ and annotate S_1 (yielding Q_1)
2. Start with P or $\{\neg b \wedge P\}$ and annotate S_2 (yielding Q_2)
3. How do we infer a Q such that $Q_1 \Rightarrow Q$ and $Q_2 \Rightarrow Q$?



Joining predicates

$$[\text{if}_p]\quad \frac{\{\, b \wedge P \,\}\, S_1\, \{\, Q \,\},\quad \{\, \neg b \wedge P \,\}\, S_2\, \{\, Q \,\}}{\{\, P \,\}\ \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\, Q \,\}}$$

1. Start with P or $\{b \wedge P\}$ and annotate S_1 (yielding Q_1)
2. Start with P or $\{\neg b \wedge P\}$ and annotate S_2 (yielding Q_2)
3. How do we infer a Q such that $Q_1 \Rightarrow Q$ and $Q_2 \Rightarrow Q$?

$Q_1 = \text{Conj}(D_1)$, $Q_2 = \text{Conj}(D_2)$
Define: $Q = Q_1 \sqcup Q_2$
$= \text{Conj}(D_1 \cap D_2)$

The join operator for SAV

$\{\, P \,\}$
`if` b `then`
$\{\, b \wedge P \,\}$
S_1
$\{\, Q_1 \,\}$
`else`
$\{\, \neg b \wedge P \,\}$
S_2
$\{\, Q_2 \,\}$
$\{\, Q \,\}$

Joining predicates

- Q_1=Conj(D_1), Q_2=Conj(D_2)
- We want to soundly approximate $Q_1 \vee Q_2$ in Π
- Define: $Q = Q_1 \sqcup Q_2$
  $= \text{Conj}(D_1 \cap D_2)$
- Notice that $Q_1 \Rightarrow Q$ and $Q_2 \Rightarrow Q$
  meaning $Q_1 \vee Q_2 \Rightarrow Q$

Handling conditional expressions

- Let *D* be a set of facts and *b* be an expression
- Goal: Elements in Π that soundly approximate
  - $D \wedge bexpr$
  - $D \wedge \neg bexpr$
- Technique: Add statement `assume` *bexpr*

  $\langle \texttt{assume}\ bexpr, s \rangle \Rightarrow^{SOS} s$ if $\mathcal{B}[\![bexpr]\!]\, s = \mathbf{tt}$

- Find a function F[`assume` *bexpr*] : $\Pi \rightarrow \Pi$

  Conj(*D*) $\wedge$ *bexpr* $\Rightarrow$ Conj(F[`assume` *bexpr*])

Handling conditional expressions

- F[`assume` *bexpr*] : $\Pi \rightarrow \Pi$ such that
  Conj(*D*) $\wedge$ *bexpr* $\Rightarrow$ Conj(F[`assume` *bexpr*])

- β(*bexpr*) = if *bexpr* is an SAV-fact then {*bexpr*} else {}
  - Notice *bexpr* $\Rightarrow$ β(*bexpr*)
  - Examples
    - β(y=z) = {y=z}
    - β(y<z) = {}

- F[`assume` *bexpr*](*D*) = *D* $\cup$ β(*bexpr*)

Example

```
{                }
if (x = y)
    {            }
    a := b + c
    {                        }
    d := b – c
    {                        }
else
    {                }
    a := b + c
    {                }
    d := b + c
    {                                    }
{                }
```

Example

```
{                  }
if (x = y)
     { x=y, y=x }
     a := b + c
     { x=y, y=x, a=b+c, a=c+b }
     d := b - c
     { x=y, y=x, a=b+c, a=c+b }
else
     {                  }
     a := b + c
     { a=b+c, a=c+b }
     d := b + c
     { a=b+c, a=c+b, d=b+c, d=c+b, a=d, d=a }
{ a=b+c, a=c+b }
```

Handling assumes

- Meet or join?

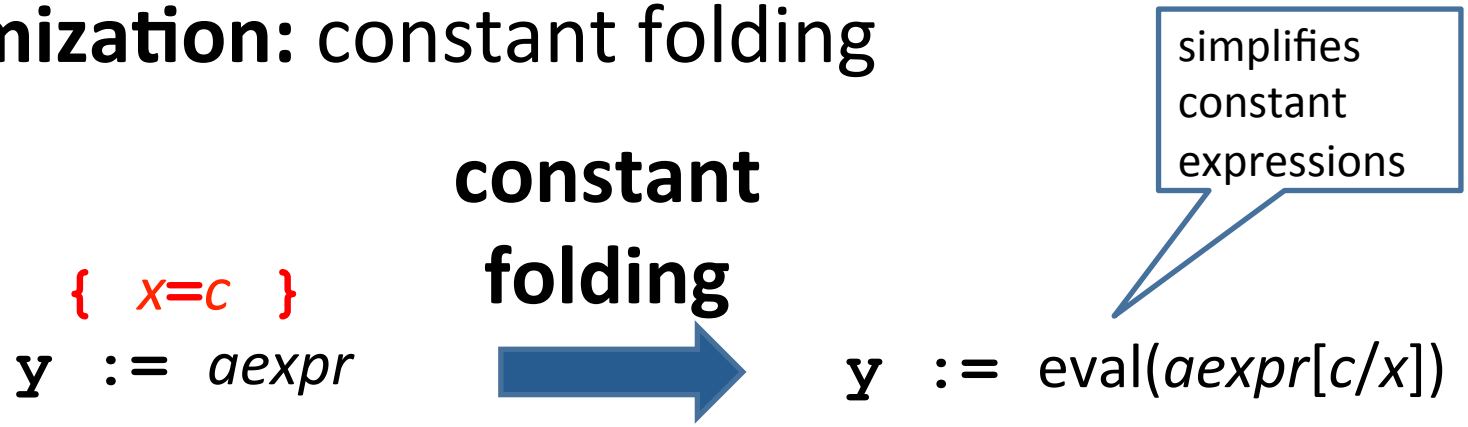
Another Example

Constant Propagation (CP)

- Goal: infers facts of the form $x=c$

Motivation: Constant folding

- **Optimization:** constant folding



  - Example: `x:=7; y:=x*9`
    transformed to: `x:=7; y:=7*9`
    and then to: `x:=7; y:=63`

CP semantic domain

?

CP semantic domain

- Define CP-*factoids*:
  $\theta = \{ x = c \mid x \in \text{Var}, c \in \text{Z} \}$
  - How many factoids are there?
- Define *predicates* as $\Pi = 2^{\theta}$
  - How many predicates are there?
  - Do all predicates make sense? $(x=5) \wedge (x=7)$
- Treat conjunctive formulas as sets of factoids

  $\{x=5, y=7\} \sim (x=5) \wedge (y=7)$

CP abstract transformer

- **Goal:** define a function
  $F^{CP}[x{:=}aexpr] : \Pi \rightarrow \Pi$ such that
  if $F^{CP}[x{:=}aexpr]\ P = P'$
  then **sp**$(x{:=}aexpr, P) \Longrightarrow P'$

?

CP abstract transformer

- **Goal:** define a function
  $F^{CP}[x:=aexpr] : \Pi \rightarrow \Pi$ such that
  if $F^{CP}[x:=aexpr]\ P = P'$
  then **sp**$(x:=aexpr, P) \Longrightarrow P'$

[kill] $\{ x=c \}\ x:=aexpr\ \{ \}$

[gen-1] $\{ \}\ x:=c\ \{ x=c \}$

[gen-2] $\{ y=c, z=c' \}\ x:=y \text{ op } z\ \{ x=c \text{ op } c' \}$

[preserve] $\{ y=c \}\ x:=aexpr\ \{ y=c \}$

Gen-kill formulation of transformers

- Suited for analysis propagating sets of factoids
  - Available expressions,
  - Constant propagation, etc.
- For each statement, define a set of killed factoids and a set of generated factoids

  $F[S]\ P = (P \setminus kill(S)) \cup gen(S)$
  - $F^{CP}[x{:=}aexpr]\ P = (P \setminus \{x{=}c\})$ *aexpr* is not a constant
  - $F^{CP}[x{:=}k]\ P = (P \setminus \{x{=}c\}) \cup \{x{=}k\}$
- Used in dataflow analysis – a special case of abstract interpretation

Does this still work?

Annotate(P, S_1; S_2) =

let Annotate(P, S_1) be $\{P\}\ A_1\ \{Q_1\}$

let Annotate(Q_1, S_2) be $\{Q_1\}\ A_2\ \{Q_2\}$

return $\{P\}\ A_1$; $\{Q_1\}\ A_2\ \{Q_2\}$

Handling conditional expressions

- We want to soundly approximate $D \wedge$ *bexpr* and $D \wedge \neg$*bexpr* in Π
- Define an artificial statement `assume` *bexpr*

  $\langle$`assume` *bexpr*, *s*$\rangle \Longrightarrow^{\text{SOS}} s$ if $\mathcal{B}[\![\textit{bexpr}]\!]\, s = \textbf{tt}$
- Define β(*bexpr*) = if *bexpr* is CP-factoid

  {*bexpr*} else {}
- Define F[`assume` *bexpr*](*D*) = $D \cup \beta$(*bexpr*)

Does this still work?

let P_t = F[`assume` *bexpr*] P

let P_f = F[`assume` ¬*bexpr*] P

let Annotate(P_t, S_1) be $\{P_t\}\ S_1\ \{Q_1\}$

let Annotate(P_f, S_2) be $\{P_f\}\ S_2\ \{Q_2\}$

return $\{P\}$

```
        if bexpr then
            {P_t} S_1 {Q_1}
        else
            {P_f} S_2 {Q_2}
      {Q_1 ⊔ Q_2}
```

How do we define join for CP?

Join example

- {x=5, y=7} ⊔ {x=3, y=7, z=9} =

Does this still work?

Annotate(P, `while` *bexpr* `do` S) =
  N' := N_c := P // Initialize
  repeat
    let P_t = F[`assume` *bexpr*] N_c
    let Annotate(P_t, S) be {N_c} A_{body} {N'}
    N_c := $N_c \sqcup N'$
  until N' = Nc
  return {P} INV= {N'} `while` *bexpr* `do` {P_t} A_{body} {F[`assume` $\neg$*bexpr*](N)}

- What about correctness?
- What about termination?

Does this still work?

Annotate(P, `while` *bexpr* `do` S) =
  N' := N_c := P // Initialize
  repeat
    let P_t = F[`assume` *bexpr*] N_c
    let Annotate(P_t, S) be {N_c} A_{body} {N'}
    N_c := $N_c \sqcup N'$
  until N' = Nc
  return {P} INV= {N'} `while` *bexpr* `do` {P_t} A_{body} {F[`assume` $\neg$*bexpr*](N)}

- What about correctness?
  - If loop terminates then is N' a loop invariant?
- What about termination?

What were the common elements?

- Two static analyses
  - Available Expressions (extended with equalities)
  - Constant Propagation
- Semantic domain
  - An approximation relation $\Rightarrow$
    - A weaker one given by set inclusion
  - Join operator
- Abstract transformers for basic statements
  - Assignments
  - **`assume`** statements
- Initial precondition

Abstract Interpretation [Cousot'77] More Formally …

- Mathematical foundation of static analysis

Abstract Interpretation [Cousot'77]

- Mathematical framework for approximating semantics (aka abstraction)
  - Allows designing sound static analysis algorithms
    - Usually compute by iterating to a fixed-point
  - Computes (loop) invariants
    - Can be interpreted as axiomatic verification assertions
    - Generalizes Hoare Logic & WP / SP calculus

Abstract Interpretation [Cousot'77]

- Mathematical foundation of static analysis
  - Abstract domains
    - Abstract states ~ Assertions
    - Join (⊔) ~ Weakening
  - Transformer functions
    - Abstract steps ~ Axioms
  - Chaotic iteration
    - Structured Programs ~ Control-flow graphs
    - Abstract computation ~ Loop invariants

Introduction to Domain Theory

Motivation

- Let “isone” be a function that must return “1$” when the input string has at least a 1 and “0$” otherwise
  - isone(00…0$) = 0$
  - isone(xx…1…$) =1$
  - isone(0…0) =?
- **Monotonicity**: in terms of information
  - Output is never retracted
    - More information about the input is reflected in more information about the output
  - How do we express monotonicity precisely?

Montonicity

- Define a partial order
  $x \sqsubseteq y$
  - A partial order is reflexive, transitive, and anti-symmetric
  - y is a refinement of x
    - "more precise"
- For streams of bits $x \sqsubseteq y$ when x is a prefix of y
- For programs, a typical order is:
  - No output (yet) $\sqsubseteq$ some output

Montonicity

- A set equipped with a partial order is a poset
- Definition:
  - D and E are postes
  - A function f: D $\rightarrow$E is monotonic if
    $\forall x, y \in D: x \sqsubseteq_D y \Rightarrow f(x) \sqsubseteq_E f(y)$
  - The semantics of the program ought to be a monotonic function
    - More information about the input leads to more information about the output

Montonicity Example

- Consider our "isone" function with the prefix ordering
- Notation:
  - 0^k is the stream with k consecutive 0's
  - 0^∞ is the infinite stream with only 0's
- Question (revisited): what is isone(0^k)?
  - By definition, isone(0^k\$) = 0\$ and isone(0^k1\$) = 1\$
  - But $0^k \sqsubseteq 0^k$\$ and $0^k \sqsubseteq 0^k$1\$
  - "isone" must be monotone, so:
    - isone(0^k) $\sqsubseteq$ isone(0^k\$) = 0\$
    - isone(0^k) $\sqsubseteq$ isone(0^k1\$) = 1\$
  - Therefore, monotonicity requires that isone(0^k) is a common prefix of 0\$ and 1\$, namely ε

Motivation

- Are there other constraints on "isone"?
- Define "isone" to satisfy the equations
  - isone(ε)=ε
  - isone(1s)=1$
  - isone(0s)=isone(s)
  - isone($)=0$
- What about 0^{∞}?

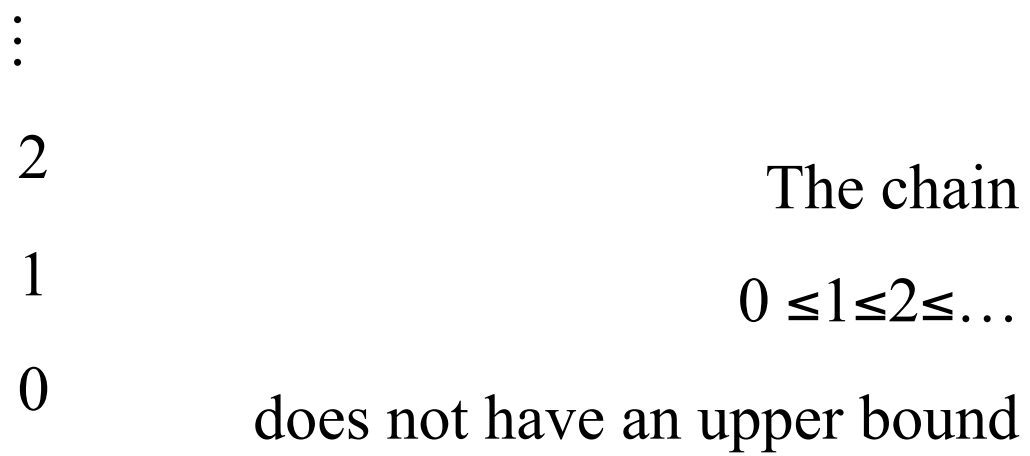
- **Continuity**: finite output depends only on finite input (no infinite lookahead)
  - Intuition: A program that can produce observable results can do it in a finite time

Chains

- A chain is a countable increasing sequence $\langle x_i \rangle = \{x_i \in X \mid x_0 \sqsubseteq x_1 \sqsubseteq \dots \}$
- An upper bound of a set if an element "bigger" than all elements in the set
- The least upper bound is the "smallest" among upper bounds:
  - $x_i \sqsubseteq \bigsqcup \langle x_i \rangle$ for all $i \in N$
  - $\bigsqcup \langle x_i \rangle \sqsubseteq y$ for all upper bounds y of $\langle x_i \rangle$ and it is unique if it exists

Complete Partial Orders

- Not every poset has an upper bound
  - with $\bot \sqsubseteq n$ and $n \sqsubseteq n$ for all $n \in \mathbb{N}$
  - {1, 2} does not have an upper bound
- Sometimes chains have no upper bound



Complete Partial Orders

- It is convenient to work with posets where every chain (not necessarily every set) has a least upper bound
- A partial order P is complete if every chain in P has a least upper bound also in P
- We say that P is a complete partial order (cpo)
- A cpo with a least ("bottom") element $\perp$ is a pointed cpo (pcpo)

Examples of cpo's

- Any set P with the order
  $x \sqsubseteq y$ if and only if $x = y$ is a cpo
  It is discrete or flat
- If we add $\perp$ so that $\perp \sqsubseteq x$ for all $x \in P$, we get a flat pointed cpo
- The set N with $\leq$ is a poset with a bottom, but not a complete one
- The set $N \cup \{ \infty \}$ with $n \leq \infty$ is a pointed cpo
- The set N with $\geq$ is a cpo without bottom
- Let S be a set and P(S) denotes the set of all subsets of S ordered by set inclusion
  - P(S) is a pointed cpo

Constructing cpos

- If D and E are pointed cpos, then so is
  $D \times E$
  $(x, y) \sqsubseteq_{D \times E} (x', y')$ iff $x \sqsubseteq_D x'$ and $y \sqsubseteq_E y'$
  $\bot_{D \times E} = (\bot_D, \bot_E)$
  $\bigsqcup (x_i, y_i) = (\bigsqcup_D x_i, \bigsqcup_E y_i)$

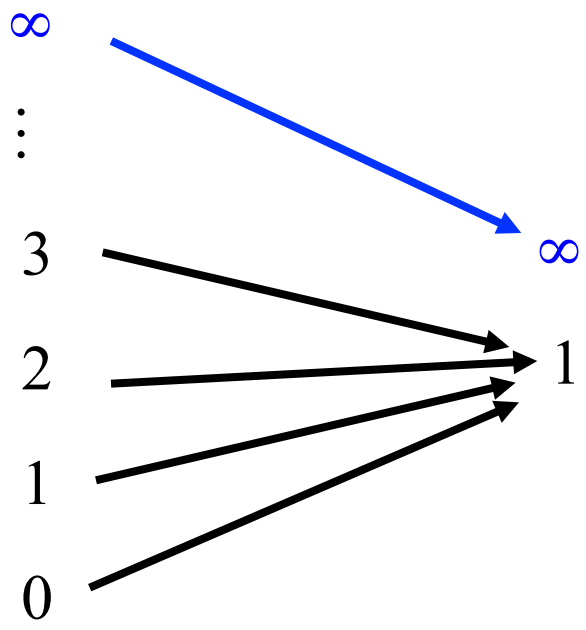
Constructing cpos (2)

- If S is a set of E is a pcpos, then so is
  $S \rightarrow E$
  $m \sqsubseteq m' \text{ iff } \forall s \in S: m(s) \sqsubseteq_E m'(s)$
  $\bot_{S \rightarrow E} = \lambda s.\ \bot_E$
  $\sqcup (m , m') = \lambda s.m(s) \sqcup_E m'(s)$

Continuity

- A monotonic function maps a chain of inputs into a chain of outputs:
  $x_0 \sqsubseteq x_1 \sqsubseteq \ldots \Rightarrow f(x_0) \sqsubseteq f(x_1) \sqsubseteq \ldots$
- It is always true that:
  $\bigsqcup_i \langle f(x_i) \rangle \sqsubseteq f(\bigsqcup_i \langle x_i \rangle)$
- But
  $f(\bigsqcup_i \langle x_i \rangle) \sqsubseteq \bigsqcup_i \langle f(x_i) \rangle$
  is not always true

A Discontinuity Example



$$f(\sqcup_i \langle x_i \rangle) \neq \sqcup_i \langle f(x_i) \rangle$$

Continuity

- Each $f(x_i)$ uses a "finite" view of the input
- $f(\bigsqcup\langle x_i \rangle)$ uses an "infinite" view of the input
- A function is **continuous** when
  $f(\bigsqcup\langle xi \rangle) = \bigsqcup_i \langle f(x_i) \rangle$
- The output generated using an infinite view of the input does not contain more information than all of the outputs based on finite inputs

Continuity

- Each $f(x_i)$ uses a "finite" view of the input
- $f(\sqcup\langle x_i\rangle)$ uses an "infinite" view of the input
- A function is **continuous** when
  $f(\sqcup\langle xi\rangle) = \sqcup_i \langle f(x_i)\rangle$
- The output generated using an infinite view of the input does not contain more information than all of the outputs based on finite inputs

- **Scott's thesis**: The semantics of programs can be described by a continuous functions

Examples of Continuous Functions

- For the partial order ($N \cup \{\infty\}, \leq$)
  - The identity function is continuous
    $id(\bigsqcup n_i) = \bigsqcup id(n_i)$
  - The constant function "five(n)=5" is continuous
    $five(\bigsqcup n_i) = \bigsqcup five(n_i)$
  - If isone(0^∞) =ε then isone is continuos

- For a flat cpo A, any monotonic function f: $A_\perp \rightarrow A_\perp$ such that f is strict is continuous

- Chapter 8 of the Wynskel textbook includes many more continuous functions

Fixed Points

- Solve equation:

$$W(\sigma) = \begin{cases} W(S[\![s]\!]\ \sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)= \perp \end{cases}$$

where $W: \sum_{\perp} \rightarrow \sum_{\perp}$; W= S[[while b do S]]

Fixed Points

- Solve equation:

$$W(\sigma) = \begin{cases} W(S[\![s]\!]\,\sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

where $W:\sum_{\perp} \rightarrow \sum_{\perp}$; $W = S[\![\text{while be do s}]\!]$

- Alternatively, W = F(W) where:

$$F(W) = \lambda\sigma. \begin{cases} W(S[\![s]\!]\,\sigma) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

Fixed Point (cont)

- Thus we are looking for a solution for W = F(W)
  - a fixed point of F
- Typically there are many fixed points
- We may argue that W ought to be continuous
  $W \in [\sum_{\perp} \rightarrow \sum_{\perp}]$
- Cut the number of solutions
- We will see how to find the least fixed point for such an equation provided that F itself is continuous

Fixed Point Theorem

- Define $F^k = \lambda x.\ F(\ F(\ldots\ F(\ x)\ldots))$ (F composed k times)
- If D is a pointed cpo and $F : D \to D$ is continuous, then
  - for any fixed-point x of F and $k \in N$
    $F^k(\bot) \sqsubseteq x$
  - The least of all fixed points is
    $\bigsqcup_k F^k(\bot)$
- Proof:
  i. By induction on k.
     - Base: $F^0(\bot) = \bot \sqsubseteq x$
     - Induction step: $F^{k+1}(\bot) = F(\ F^k(\bot)) \sqsubseteq F(\ x) = x$
  ii. It suffices to show that $\bigsqcup_k F^k(\bot)$ is a fixed-point
     - $F(\bigsqcup_k F^k(\bot)) = \bigsqcup_k F^{k+1}(\ \bot\ ) = \bigsqcup_k F^k(\bot)$

Fixed-Points (notes)

- If F is continuous on a pointed cpo, we know how to find the least fixed point
- All other fixed points can be regarded as refinements of the least one
  - They contain more information, they are more precise
  - In general, they are also more arbitrary

Fixed-Points (notes)

- If F is continuous on a pointed cpo, we know how to find the least fixed point
- All other fixed points can be regarded as refinements of the least one
  - They contain more information, they are more precise
  - In general, they are also more arbitrary
  - They also make less sense for our purposes

Denotational Semantics

- Meaning of programs

Denotational Semantics of While

- $\sum_{\perp}$ is a flat pointed cpo
  - A state has more information on non-termination
  - Otherwise, the states must be equal to be comparable (information-wise)

- We want strict functions $\sum_{\perp} \rightarrow \sum_{\perp}$
  - therefore, continuous functions

- The partial order on $\sum_{\perp} \rightarrow \sum_{\perp}$
  $f \sqsubseteq g$ iff $f(x) = \perp$ or $f(x) = g(x)$ for all $x \in \sum_{\perp}$
  - g terminates with the same state whenever f terminates
  - g might terminate for more inputs

Denotational Semantics of While

- Recall that W is a fixed point of $F:[[\sum_{\perp} \to \sum_{\perp}] \to [\sum_{\perp} \to \sum_{\perp}]]$
- F is continuous

$$F(w) = \lambda\sigma. \begin{cases} w(S[\![s]\!](\sigma)) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)= \perp \end{cases}$$

- Thus, we set $S[\![\text{while b do c}]\!] = \sqcup F^k(\perp)$
  - Least fixed point
    - Terminates least often of all fixed points
- Agrees on terminating states with all fixed point

Denotational Semantics of While

- $S[\![\text{skip}]\!] = \lambda\sigma.\sigma$
- $S[\![X := exp]\!] = \lambda\sigma.\sigma[X \mapsto A[\![exp]\!]\sigma]$
- $S[\![s_0 ; s_1]\!] = \lambda\sigma.\ S[\![s_1]\!](S[\![s_0]\!]\sigma)$
- $S[\![\text{if } b \text{ then } s_0 \text{ else } s_1]\!] =$

  $\lambda\sigma.$ if $B[\![b]\!]\sigma$ then $S[\![s_0]\!]\sigma$ else $S[\![s_1]\!]\sigma$
- $S[\![\text{ while } b \text{ do } s]\!] = \bigsqcup F^k(\bot)$
  - k=0, 1, …
  - $F = \lambda w.\ \lambda\sigma.$ if $B[\![b]\!](\sigma)$=true $w(S[\![s]\!](\sigma))$ else σ

Example(1)

- while true do skip
- $F:[[\sum_{\perp} \rightarrow \sum_{\perp}] \rightarrow [\sum_{\perp} \rightarrow \sum_{\perp}]]$

$$F = \lambda w.\lambda\sigma.\begin{cases} w(S[\![s]\!](\sigma)) & \text{if } B[\![b]\!](\sigma)=\text{true} \\ \sigma & \text{if } B[\![b]\!](\sigma)=\text{false} \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

$B[\![\text{true}]\!]=\lambda\sigma.\text{true}$

$S[\![\text{skip}]\!]=\lambda\sigma.\sigma$

$F = \lambda\, w.\lambda\sigma.w(\sigma)$

$F^0(\perp)= \perp \sqcup F^1(\perp) = \perp \sqcup F^2(\perp) = \perp = \perp$

Example(2)

- while false do s
- $F:[[\sum_{\perp} \rightarrow \sum_{\perp}] \rightarrow [\sum_{\perp} \rightarrow \sum_{\perp}]]$

$$F = \lambda w.\lambda\sigma.\begin{cases} w(S[\![s]\!](\sigma)) & \text{if } B[\![b]\!](\sigma)=true \\ \sigma & \text{if } B[\![b]\!](\sigma)=false \\ \perp & \text{if } B[\![b]\!](\sigma)=\perp \end{cases}$$

$B[\![false]\!]=\lambda\sigma.false$

$F = \lambda w.\lambda\sigma.\sigma$

$F^0(\perp)= \perp \quad \sqcup \quad F^1(\perp) = \lambda\sigma.\sigma \quad \sqcup F^2(\perp) = \lambda\sigma.\sigma \quad = \lambda\sigma.\sigma$

Example(3)

$[\![$ while $x \neq 3$ do $x = x - 1$ $]\!] = \bigsqcup F^k(\bot)$ $k=0, 1, \ldots$

where

$F = \lambda w.\ \lambda\sigma.$ if $\sigma(x) \neq 3$ $w(\sigma[x \mapsto \sigma(x) - 1])$ else σ

$F^0(\bot)$	$\bot$
$F^1(\bot)$	if $\sigma(x) \neq 3$ $\bot(\sigma[x \mapsto \sigma(x) - 1])$ else σ if $\sigma(x) \neq 3$ then $\bot$ else σ
$F^2(\bot)$	if $\sigma(x) \neq 3$ then $F^1(\sigma[x \mapsto \sigma(x) - 1])$ else σ if $\sigma(x) \neq 3$ then (if $\sigma[x \mapsto \sigma(x) - 1]\ x \neq 3$ then $\bot$ else $\sigma[x \mapsto \sigma(x) - 1]$) else σ if $\sigma(x) \neq 3$ (if $\sigma(x) \neq 4$ then $\bot$ else $\sigma[x \mapsto \sigma(x) - 1]$) else σ if $\sigma(x) \in \{3, 4\}$ then $\sigma[x \mapsto 3]$ else $\bot$
$F^k(\bot)$ lfp(F)	if $\sigma(x) \in \{3, 4, \ldots k\}$ then $\sigma[x \mapsto 3]$ else $\bot$ if $\sigma(x) \geq 3$ then $\sigma[x \mapsto 3]$ else $\bot$

Complete Lattice

- Let $(D, \sqsubseteq)$ be a partial order

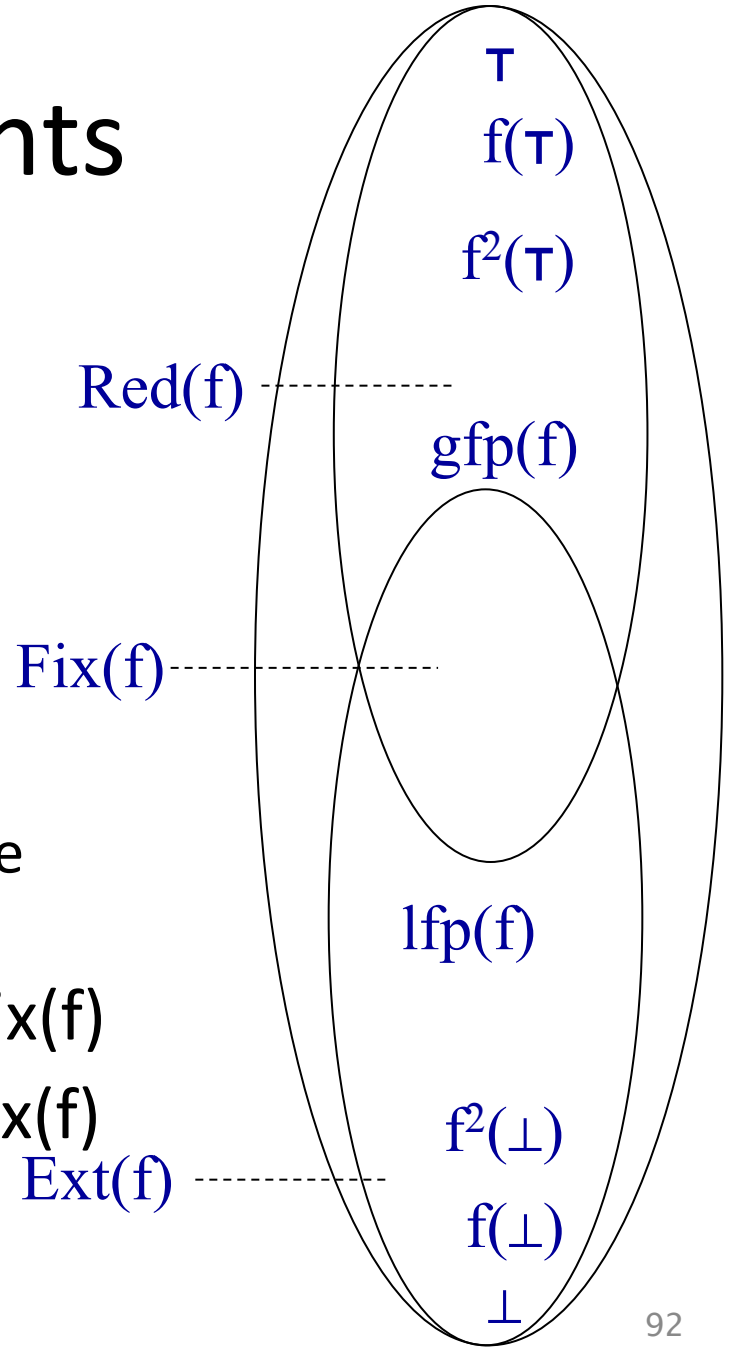
- D is a complete lattice if every subset has both greatest lower bounds and least upper bounds

Knaster-Tarski Theorem

- Let f: L $\rightarrow$L be a monotonic function on a complete lattice L
- The least fixed point lfp(f) exists
  - lfp(f) = $\sqcap\{x \in L: f(x) \sqsubseteq x\}$

Fixed Points

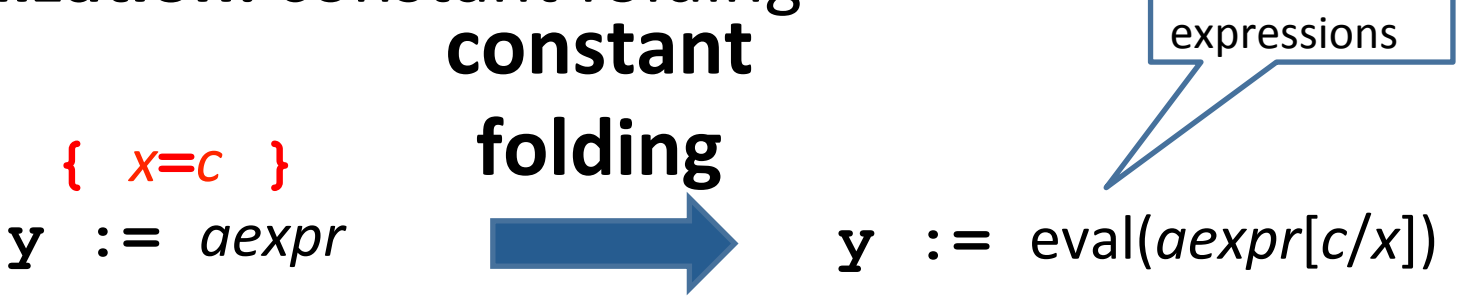
- A monotone function f: L → L where (L, ⊑, ⊔, ⊓, ⊥, ⊤) is a complete lattice
- Fix(f) = { l: l ∈ L, f(l) = l}
- Red(f) = {l: l ∈ L, f(l) ⊑ l}
- Ext(f) = {l: l ∈ L, l ⊑ f(l)}
  - $l_1 \sqsubseteq l_2 \Rightarrow f(l_1) \sqsubseteq f(l_2)$
- Tarski's Theorem 1955: if f is monotone then:
  - lfp(f) = ⊓ Fix(f) = ⊓ Red(f) ∈ Fix(f)
  - gfp(f) = ⊔ Fix(f) = ⊔ Ext(f) ∈ Fix(f)



Constant Propagation - Again

Constant Propagation

- **Optimization:** constant folding



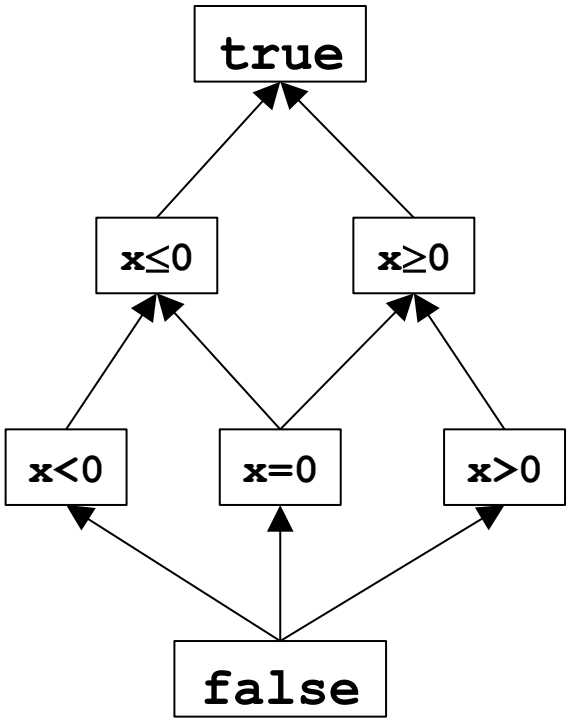
  - Example: `x:=7; y:=x*9`
    transformed to: `x:=7; y:=7*9`
    and then to: `x:=7; y:=63`
- **Analysis:** constant propagation (CP)
  - Infers facts of the form *x=c*

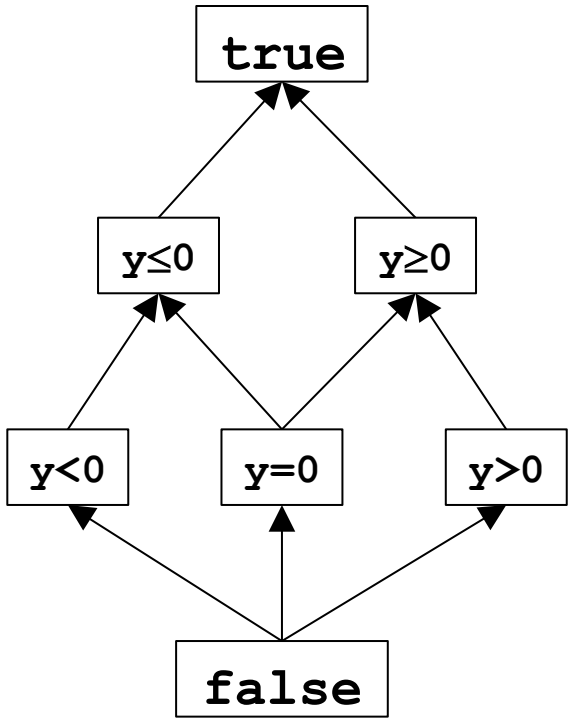
CP semantic domain

- Define CP-*factoids*:
  $\theta = \{ x = c \mid x \in \text{Var}, c \in \text{Z} \}$
  - How many factoids are there?
- Define *predicates* as $\Pi = 2^{\theta}$
  - How many predicates are there?
  - Do all predicates make sense? $(x=5) \wedge (x=7)$
- Treat conjunctive formulas as sets of factoids

  $\{x=5, y=7\} \sim (x=5) \wedge (y=7)$

One lattice per variable



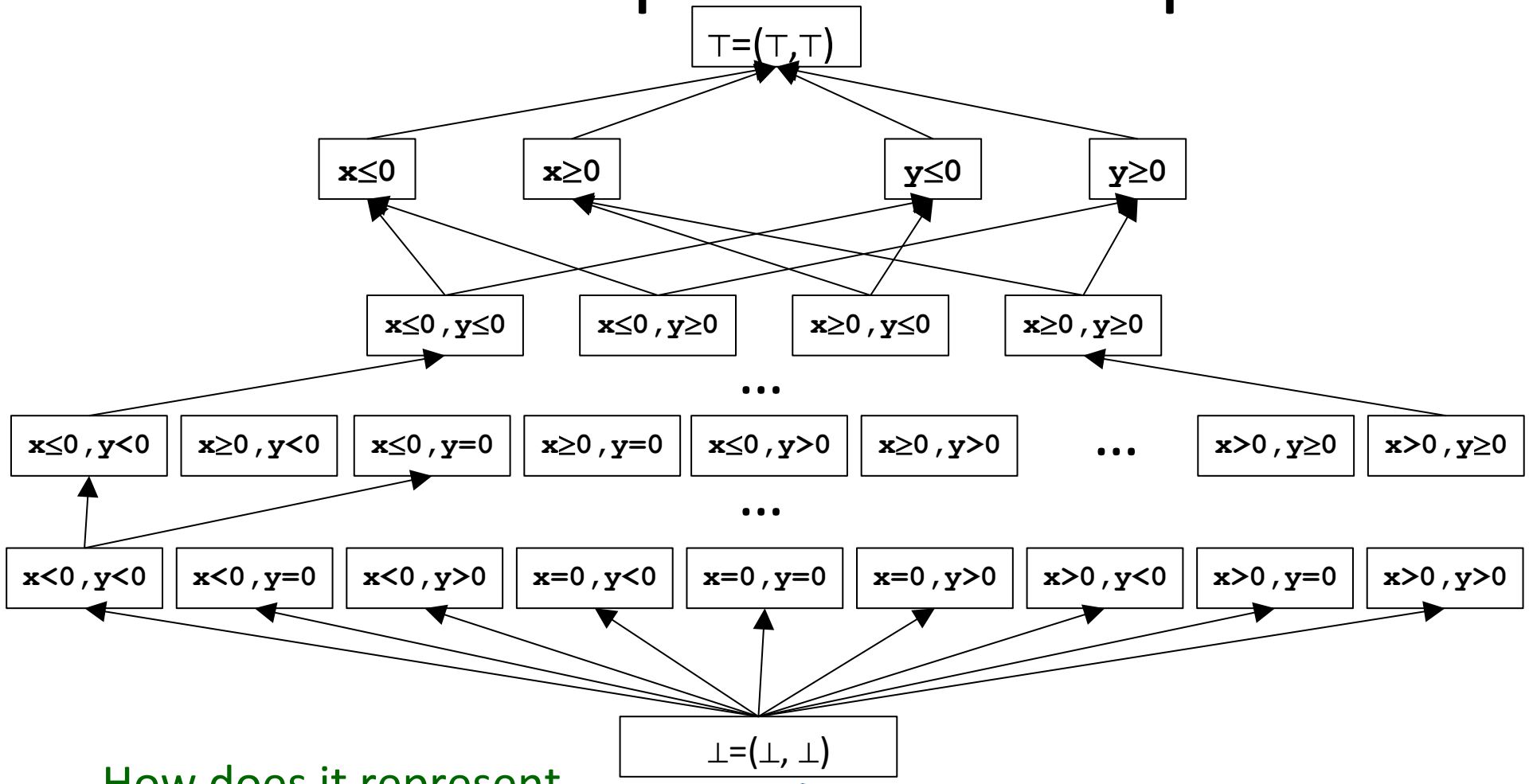


How can we compose them?

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* $x_1 \sqsubseteq_1 y_1 \; x_2 \sqsubseteq_2 y_2$
  - $\sqcup_{cart} = ?$ $\sqcap_{cart} = ?$ $\bot_{cart} = ?$ $\top_{cart} = ?$
- **Lemma:** L is a complete lattice
- Define the Cartesian constructor $L_{cart} = \text{Cart}(L_1, L_2)$

Cartesian product example



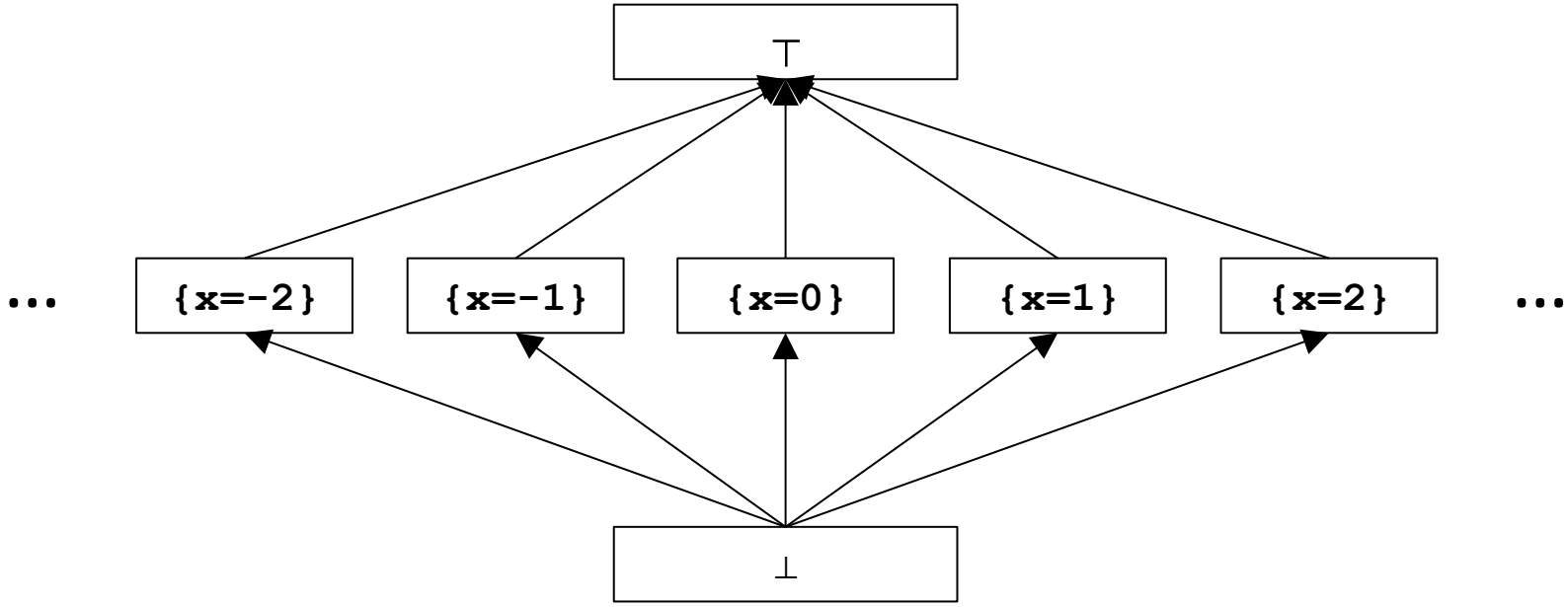
How does it represent
$(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

(**false**, **false**)

Disjunctive completion

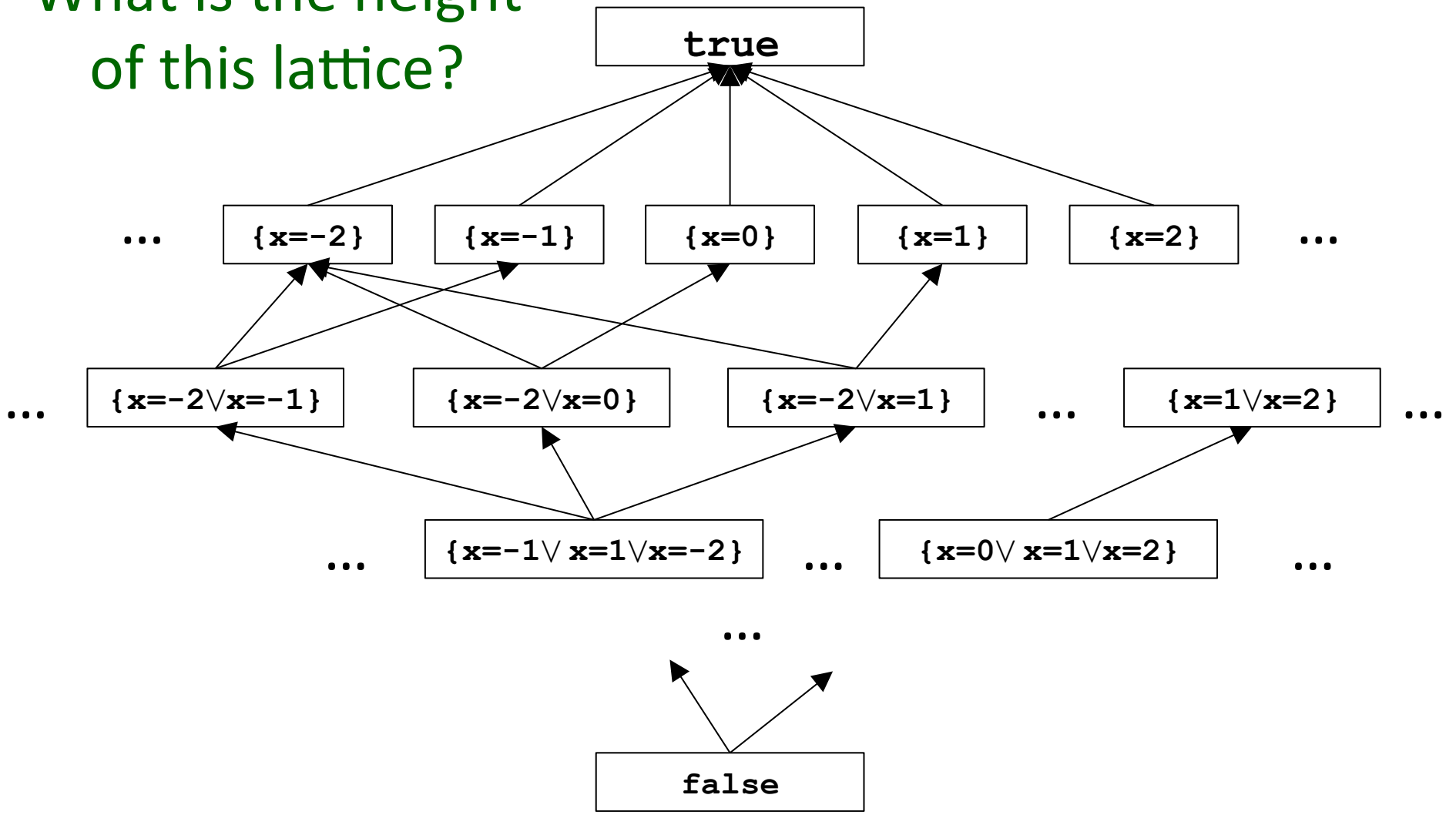
- For a complete lattice
  $L = (D, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$
- Define the powerset lattice
  $L_\vee = (2^D, \sqsubseteq_\vee, \sqcup_\vee, \sqcap_\vee, \bot_\vee, \top_\vee)$
  $\sqsubseteq_\vee$ = ? $\sqcup_\vee$ = ? $\sqcap_\vee$ = ? $\bot_\vee$ = ? $\top_\vee$ = ?
- **Lemma:** $L_\vee$ 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
  $L_\vee$ = Disj(L)

The base lattice CP



The disjunctive completion of CP

What is the height of this lattice?



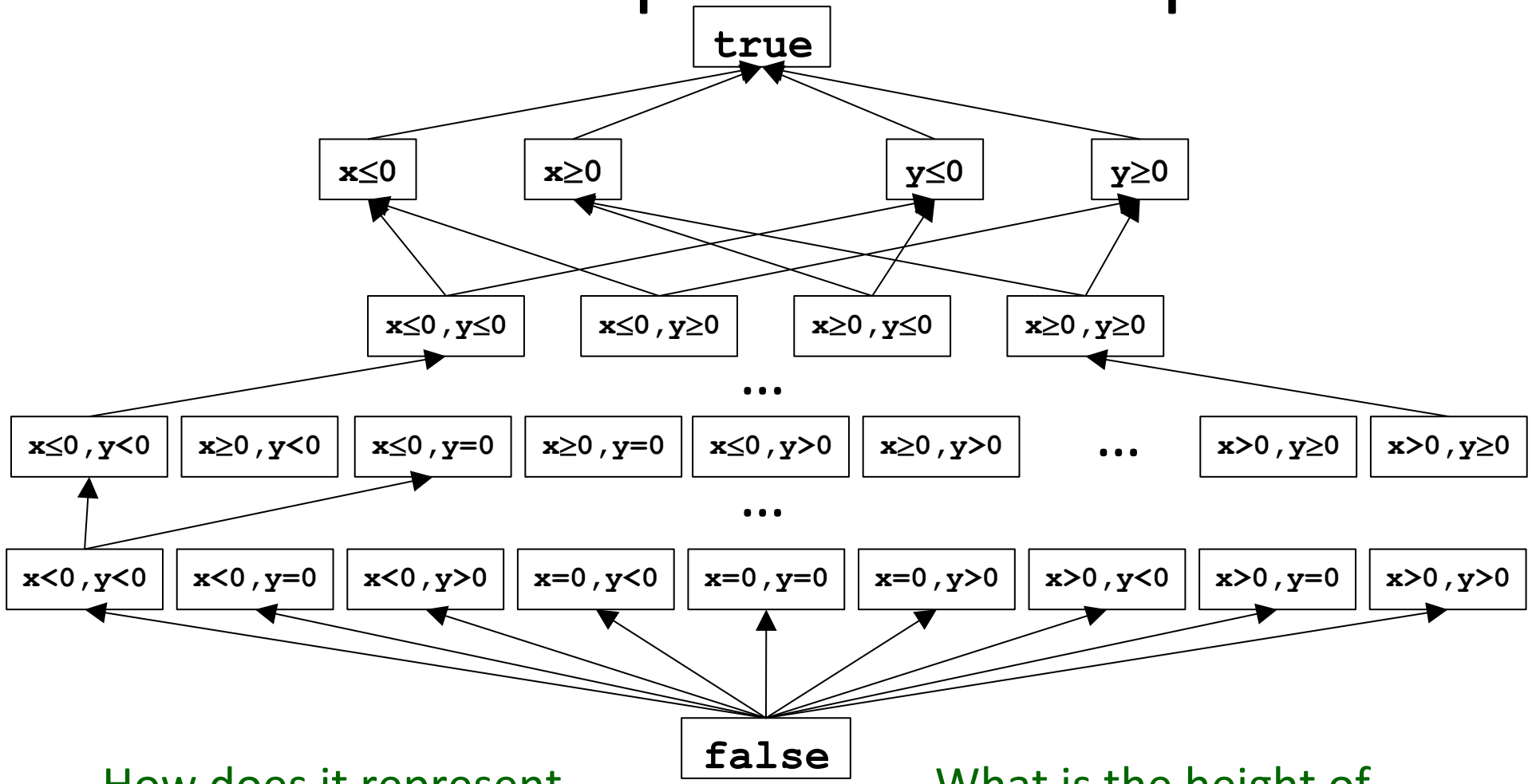
Relational product of lattices

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

Relational product of lattices

- $L_1 = (D_1, \sqsubseteq_1, \sqcup_1, \sqcap_1, \bot_1, \top_1)$
  $L_2 = (D_2, \sqsubseteq_2, \sqcup_2, \sqcap_2, \bot_2, \top_2)$
- $L_{rel} = (2^{D_1 \times D_2}, \sqsubseteq_{rel}, \sqcup_{rel}, \sqcap_{rel}, \bot_{rel}, \top_{rel})$
  as follows:
  - L_{rel} = Disj(Cart(L_1, L_2))
- **Lemma:** *L* is a complete lattice
- What does it buy us?

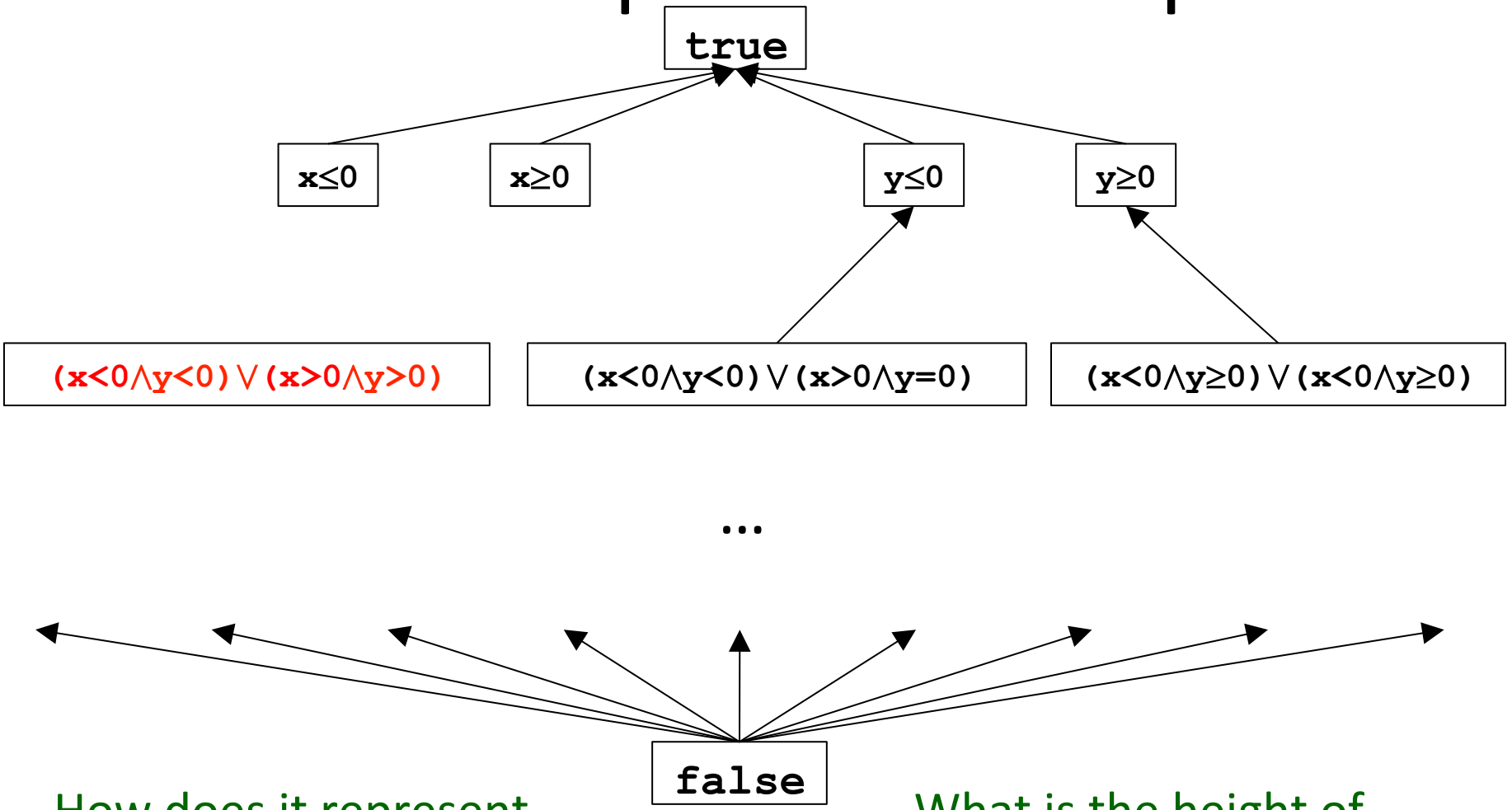
Cartesian product example



How does it represent $(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

What is the height of this lattice?

Relational product example



How does it represent $(x<0 \wedge y<0) \vee (x>0 \wedge y>0)$?

What is the height of this lattice?