Verification of Real-Time Systems

Foundations of Abstract Interpretation

Jan Reineke

Advanced Lecture, Summer 2015

Applications of Abstract Interpretation

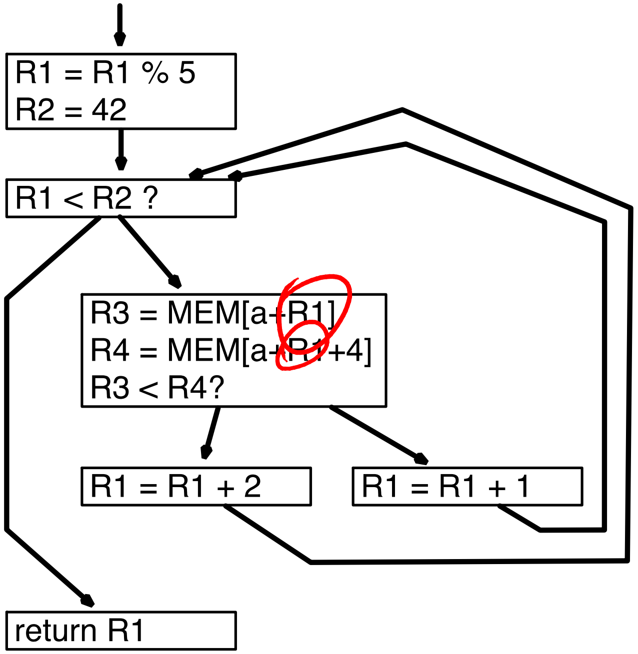
- Compilers:
  - Is it safe to perform a certain program transformation?
- Verification:
  - Can there be runtime errors? E.g. array out-of-bounds access, division-by-zero, dereferencing of null pointers
  - Security analysis: does information about secrets leak?
  - Timing analysis: loop bounds, cache analysis,…

Recap: Value Analysis

Determines invariants on values of registers at different program points. Invariants are often in the form of enclosing intervals of all possible values.

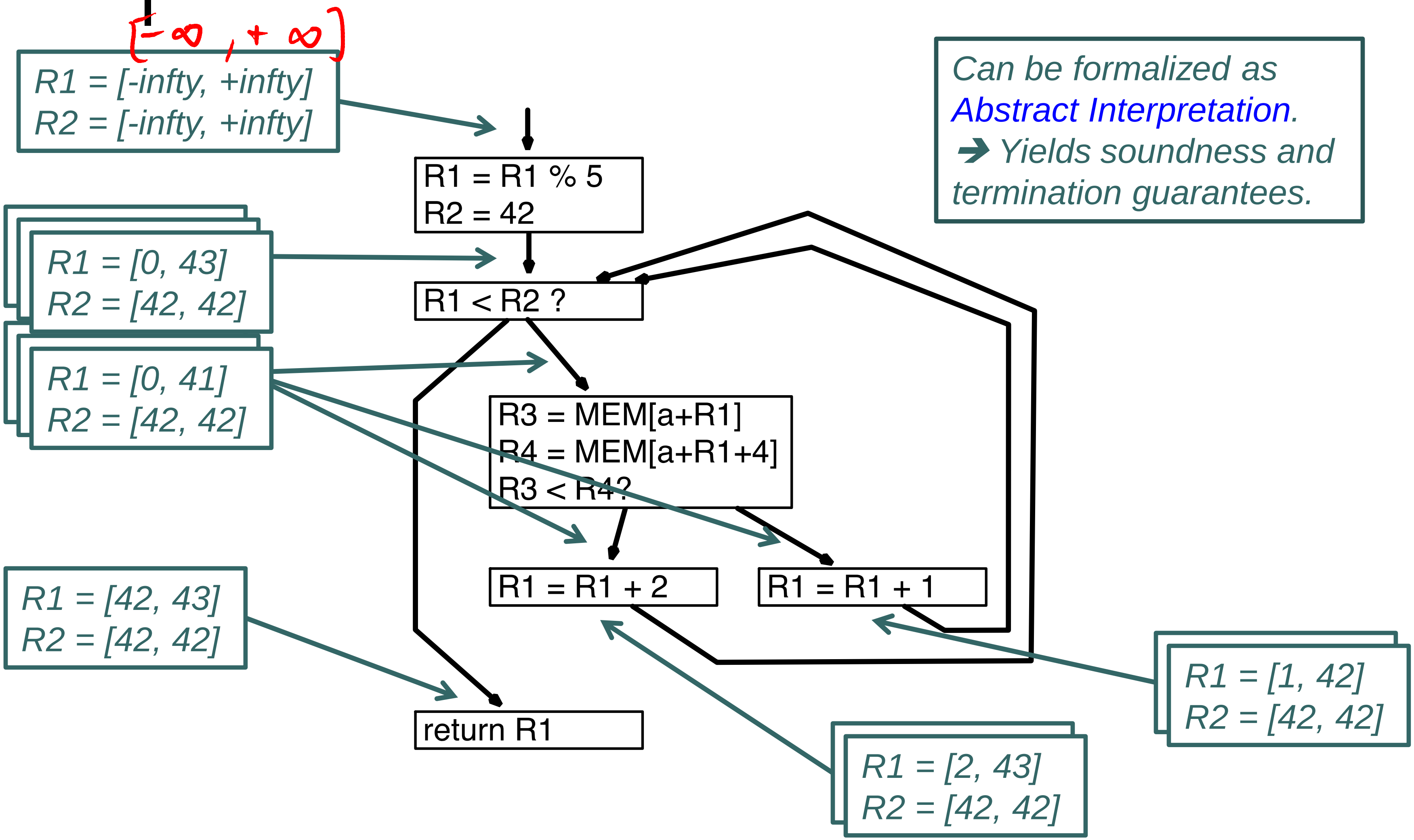
Where is this information used?

- Microarchitectural Analysis
  - Pipeline Analysis
  - Cache Analysis
- Control-Flow Analysis
  - Detect infeasible paths
  - Derive loop bounds



Value Analysis
Intuition of Interval Analysis

Can be formalized as ***Abstract Interpretation****.*
➔ Yields soundness and termination guarantees.



Abstract Interpretation

- Semantics-based approach to program analysis
- Framework to develop provably correct and terminating analyses

Ingredients:

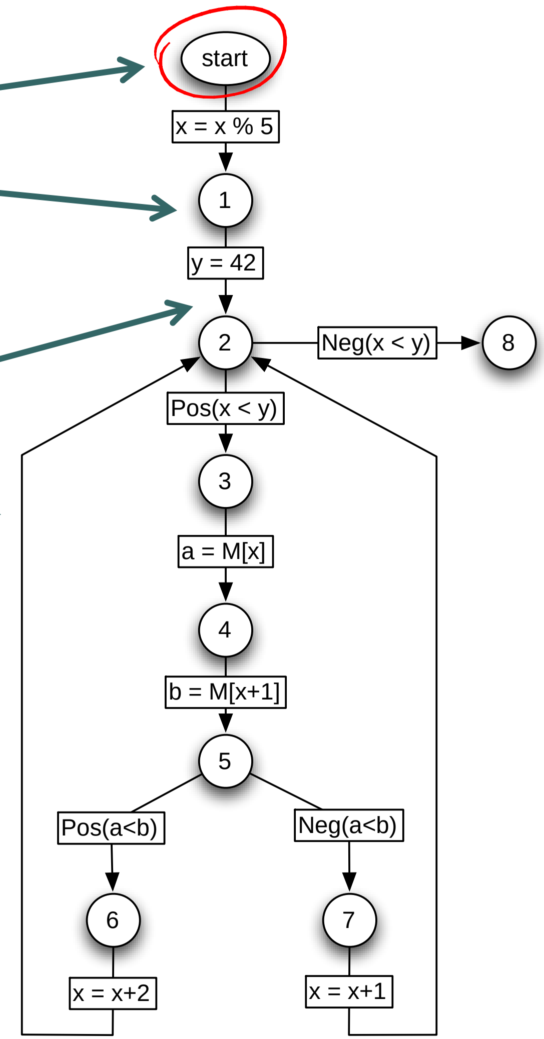
- Concrete semantics: Formalizes meaning of a program
- Abstract semantics
- Both semantics defined as fixpoints of monotone functions over some domain
- Relation between the two semantics establishing correctness

Representing Programs by Control-Flow Graphs

Vertices

$$CFG\ G = (V, E, start, labeling)$$

Edges



$$start \in V$$

$$E \subseteq V \times V$$

$$\underline{labeling} : E \to Statement$$

$labeling(3, 4) = a = M[x]$

Four Kinds of Statements

1. Assignment: $R = e$
2. Load: $R = M[e]$
3. Store: $M[e_1] = e_2$ R
4. Test: Pos(e) or Neg(e)

$e = x + y + 7z$

Meaning of Statements

$\rho' = \rho[R \mapsto x]$

$\rho'(R) = x \mid \forall R' \neq R :$

$\rho'(R') = \rho(R$

States consists of variables and memory:

$$s = (\rho, \mu) \in \mathit{States}$$

$$\rho : \mathit{Vars} \to \mathit{int}$$ *Values of Variables*

$$\mu : \mathbb{N} \to \mathit{int}$$ *Contents of Memory*

Execution of a statement transforms states:

$$[\![\mathit{statement}]\!] \subseteq \mathit{States} \times \mathit{States}$$

$$[\![R = e]\!] := \{((\rho, \mu), (\rho[R \mapsto [\![e]\!]\rho], \mu)) \mid (\rho, \mu) \in \mathit{States}\}$$

$$[\![R = M[e]]\!] := \{((\rho, \mu), (\rho[R \mapsto \mu([\![e]\!]\rho)], \mu)) \mid (\rho, \mu) \in \mathit{States}\}$$

$$[\![M[e_1] = e_2]\!] := \{((\rho, \mu), (\rho, \mu[\underline{[\![e_1]\!]\rho} \mapsto \underline{[\![e_2]\!]\rho}])) \mid (\rho, \mu) \in \mathit{States}\}$$

$$[\![\mathit{Pos}(e)]\!] := \{((\rho, \mu), (\rho, \mu)) \mid (\rho, \mu) \in \mathit{States} \wedge [\![e]\!]\rho \neq 0\}$$

$$[\![\mathit{Neg}(e)]\!] := \{((\rho, \mu), (\rho, \mu)) \mid (\rho, \mu) \in \mathit{States} \wedge [\![e]\!]\rho = 0\}$$

Meaning of Expressions

Evaluation of expressions is as expected:

$$\llbracket a \rrbracket \rho := \rho(a) \qquad\qquad if\ a \in\ Vars$$

$$\llbracket e_1 \otimes e_2 \rrbracket \rho := \llbracket e_1 \rrbracket \rho \otimes \llbracket e_2 \rrbracket \rho$$

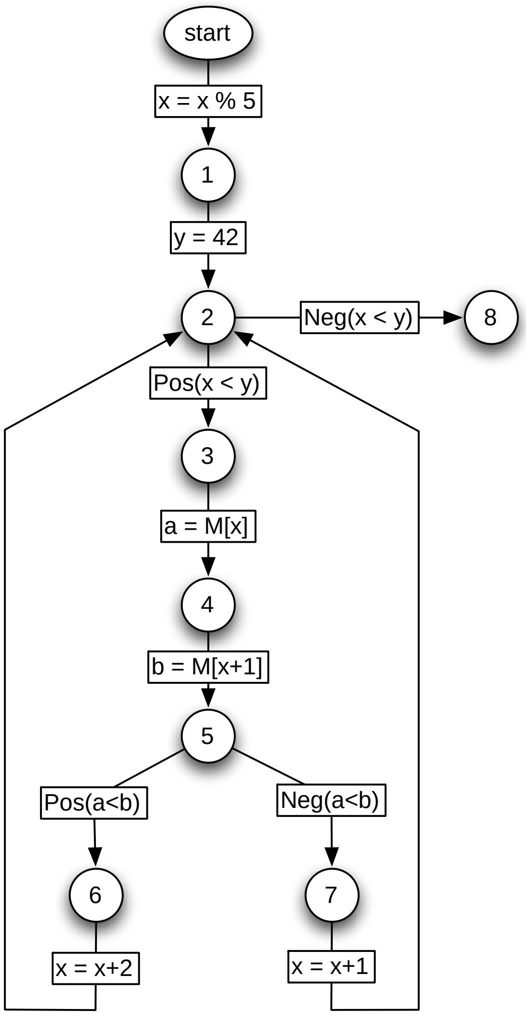
$$\llbracket a < b \rrbracket \rho := \begin{cases} 1 & : \llbracket a \rrbracket \rho < \llbracket b \rrbracket \rho \\ 0 & : otherwise \end{cases}$$

$$\ldots$$

Concrete Semantics

Different semantics are required for different properties:

- “Is there an execution in which the value of x alternates between 3 and 5?” ➔ Trace Semantics
- “Is the final value of x always the same as the initial value of x?” ➔ “Input/Output” Semantics
- “May x ever assume the value 45 at program point 7?” ➔ Reachability Semantics



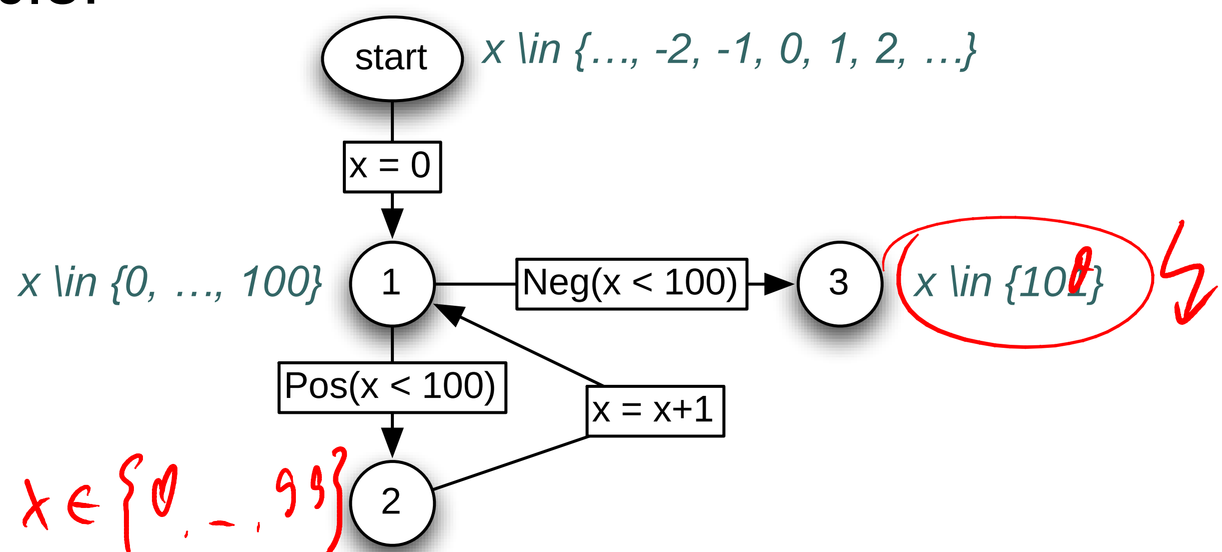
Concrete Semantics

- Trace Semantics: Captures set of traces of states that the program may execute.
- Input/Output Semantics: Captures the pairs of initial and final states of execution traces.
  - Abstraction of Trace Semantics
- Reachability Semantics: Captures the set of reachable states at each program point
  - Abstraction of Trace Semantics

Reachability Semantics

Captures the set of reachable states at each program point. Formally: $Reach : V \to \mathcal{P}(States)$

Example:

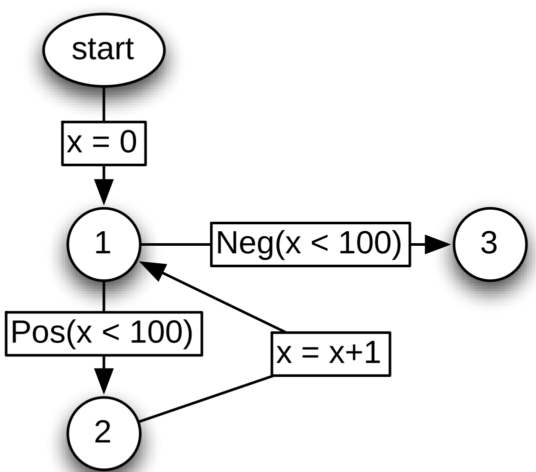


Reachability Semantics

Can be captured as the least solution of:

$$Reach(start) = \underline{States}$$

$$\forall v' \in V \setminus \{start\} : Reach(v') = \bigcup_{v \in V, (v,v') \in E} [\![labeling(v, v')]\!](Reach(v))$$



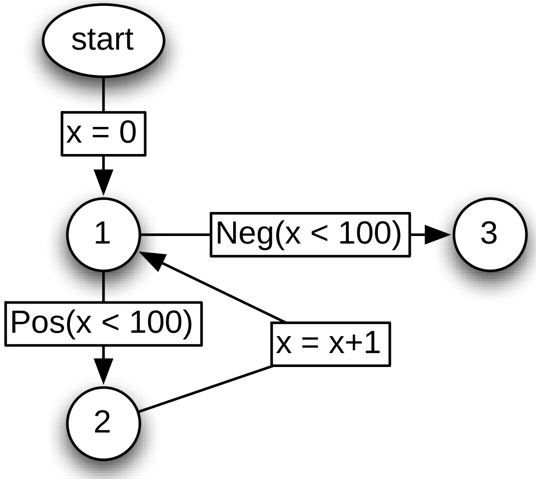
$$Reach(1) = [\![labeling(start, 1)]\!](Reach(start)) \cup [\![labeling(2, 1)]\!](Reach(2))$$
$$Reach(2) = [\![labeling(1, 2)]\!](Reach(1))$$
$$Reach(3) = [\![labeling(1, 3)]\!](Reach(1))$$

$$Reach(1) = [\![x = 0]\!](Reach(start)) \cup [\![x = x + 1]\!](Reach(2))$$
$$Reach(2) = [\![Pos(x < 100)]\!](Reach(1))$$
$$Reach(3) = [\![Neg(x < 100)]\!](Reach(1))$$

$$Reach(1) = \{0\} \cup \{v + 1 \mid v \in Reach(2)\}$$
$$Reach(2) = Reach(1) \cap \{\dots, 98, 99\}$$
$$Reach(3) = Reach(1) \cap \{100, 101, \dots\}$$

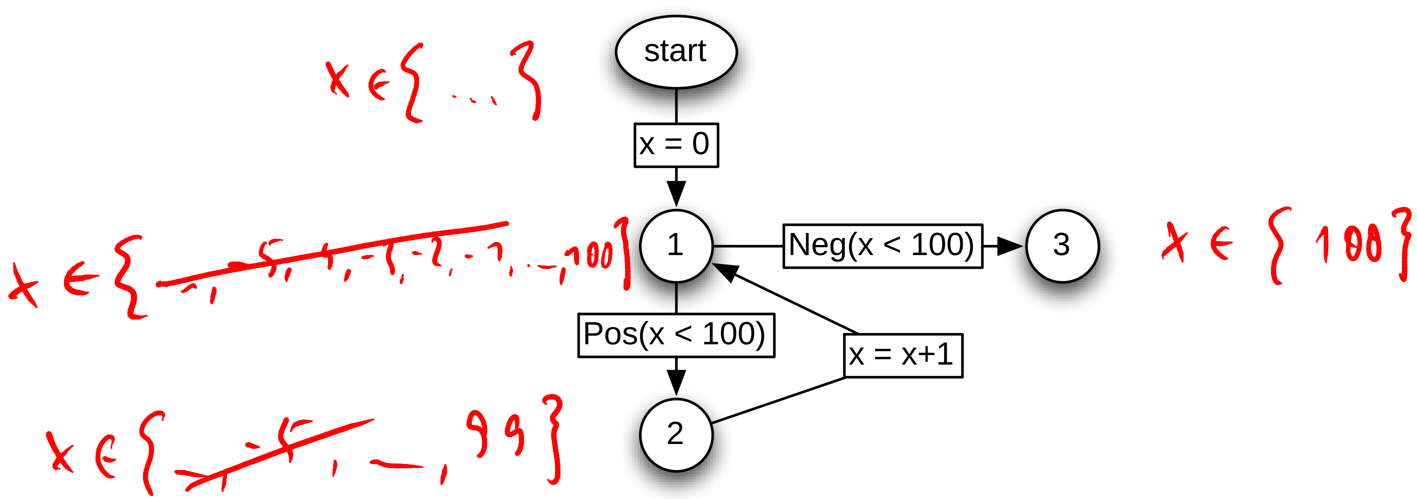
Questions

- Why the least solution?
- Is there more than one solution?
- Is there a unique least solution?
- Can we systematically compute it?



Answers

- Is there more than one solution? Yes!
- Is there a unique least solution? Yes!
- Can we systematically compute it? Yes and No.



Why? Knaster-Tarski Fixpoint Theorem!

THEOREM 1 (KNASTER-TARSKI, 1955).
Assume $(D, \leq)$ *is a* complete lattice. *Then every* monotonic *function* $f : D \to D$ *has a* least fixed point $d_0 \in D$.

Raises more questions:

- What is a complete lattice?
- What is a monotonic function?
- What is a fixed point?

Monotone Functions

$f : A \to B$

Let $(D, \leq)$ be *partially-ordered set.*
For example: $D = \mathbb{N}$ and $\leq$ the order on natural numbers.

$$\boxed{\begin{array}{l}\text{Function } f : D \to D \text{ is } \textit{monotone} \text{ (order-preserving) iff} \\ \qquad\qquad\text{for all } d_1, d_2 \in D : d_1 \leq d_2 \Rightarrow f(d_1) \leq f(d_2).\end{array}}$$

Examples:

$$f(x) = x \quad \checkmark \quad d_1 \leq d_2 \Rightarrow d_1 \leq d_2$$

$$g(x) = -x \quad \times$$

$$h(x) = x - 1 \quad \checkmark$$

Which of these are monotone?

$(\mathcal{P}(\mathbb{N}), \subseteq)$

$$F(X) = \{f(x) \mid x \in X\}$$

$$G(X) = \{y \mid x \in X \wedge (x, y) \in R\}$$

Need to know what the order is.

Partial Orders

A binary relation $\leq: D \times D$ is a *partial order*, iff for all $a, b, c \in D$, we have that:

- $a \leq a$ (reflexivity),
- if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry),
- if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity).

A set with a partial order is called a *partially-ordered set*.

Partial Orders: Examples I

The natural numbers ordered by the standard less-than-or-equal relation: $(\mathbb{N}, \leq)$. $(\mathbb{N}, \geq)$

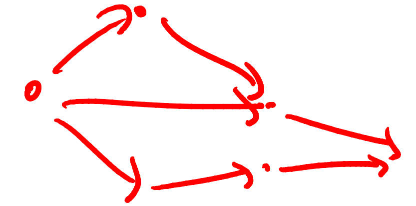
The set of subsets of a given set (its powerset) ordered by the subset relation: $(\mathcal{P}(A), \subseteq)$.

The set of subsets of a given set (its powerset) ordered by the subset relation: $(\mathcal{P}(A), \supseteq)$.

The natural numbers ordered by *divisibility*: $(\mathbb{N}, |)$.

$3 | 9 \quad \neg(9 | 3)$

Partial Orders: Examples II



The vertex set V of a directed acyclic graph $G = (V, E)$ ordered by reachability (reflexive, transitive closure of edge relation).

~~The vertex set V of an arbitrary graph $G = (V, E)$ ordered by reachability.~~ X

$a \rightleftarrows v$ $\quad a \leq v \wedge v \leq a \quad a \neq v$

For a set X and a partially-ordered set P, the function space $F : X \to P$, where $\underline{f \leq g}$ if and only if $f(x) \leq g(x)$ for all x in X.

What about $Reach : \underline{V} \to \underline{\mathcal{P}(States)}$?

$f \leq g \;:\Leftrightarrow\; \forall v \in V.\ f(v) \leq g(v)$

Complete Lattices

A partially-ordered set $(L, \leq)$ is a *complete lattice* if every subset A of L has both a *least upper bound* (denoted $\bigsqcup A$) and a *greatest lower bound* (denoted $\bigsqcap A$).

What is an upper bound of a set A?

An element x is an upper bound of a set A if x if for every element a of A, we have $a \leq x$.

What is the least upper bound (also: join, supremum) of a set A?

x is the *least upper bound* of A, denoted $\bigsqcup A$, if

1. x is an upper bound of A,
2. for every upper bound y of A, we have $x \leq y$.