Reinhard Wilhelm

Universität des Saarlandes

Development of Safety-Critical Embedded Systems

Static Program Analysis

Winter Semester 2012/2013

Slides based on:

- H. Seidl, R. Wilhelm, S. Hack: Compiler Design, Volume 3, Analysis and Transformation, Springer Verlag, 2012
- F. Nielson, H. Riis Nielson, C. Hankin: Principles of Program Analysis, Springer Verlag, 1999
- R. Wilhelm, B. Wachter: Abstract Interpretation with Applications to Timing Validation. CAV 2008: 22-36
- Helmut Seidl's slides

A Short History of Static Program Analysis

- Early high-level programming languages were implemented on very small and very slow machines.
- Compilers needed to generate executables that were extremely efficient in space and time.
- Compiler writers invented efficiency-increasing program transformations, wrongly called optimizing transformations.
- Transformations must not change the semantics of programs.
- Enabling conditions guaranteed semantics preservation.
- Enabling conditions were checked by static analysis of programs.

Theoretical Foundations of Static Program Analysis

- Theoretical foundations for the solution of recursive equations: Kleene (30s), Tarski (1955)
- Gary Kildall (1972) clarified the lattice-theoretic foundation of data-flow analysis.
- Patrick Cousot (1974) established the relation to the programming-language semantics.

Static Program Analysis as a Verification Method

- Automatic method to derive invariants about program behavior, answers questions about program behavior:
  - will index always be within bounds at program point p?
  - will memory access at p always hit the cache?
- answers of sound static analysis are correct, but approximate: don't know is a valid answer!
- analyses proved correct wrt. language semantics,

Proposed Lectures Content:

1. Introductory example: rules-of-sign analysis
2. theoretical foundations: lattices
3. an operational semantics of the language
4. another example: constant propagation
5. relating the semantics to the analysis—correctness proofs
6. Further static analyses in compilers: Elimination of superfluous computations
   - → available expressions
   - → live variables
   - → array-bounds checks
7. timing (WCET) analysis
8. analysis for runtime errors

1 Introduction

... in this course and in the Seidl/Wilhelm/Hack book:

a simple imperative programming language with:

• variables	//	registers
• $R = e;$	//	assignments
• $R = M[e];$	//	loads
• $M[e_1] = e_2;$	//	stores
• if (e) s_1 else s_2	//	conditional branching
• goto $L;$	//	no loops

An intermediate language into which (almost) everything can be translated.

In particular, no procedures. So, only intra-procedural analyses!

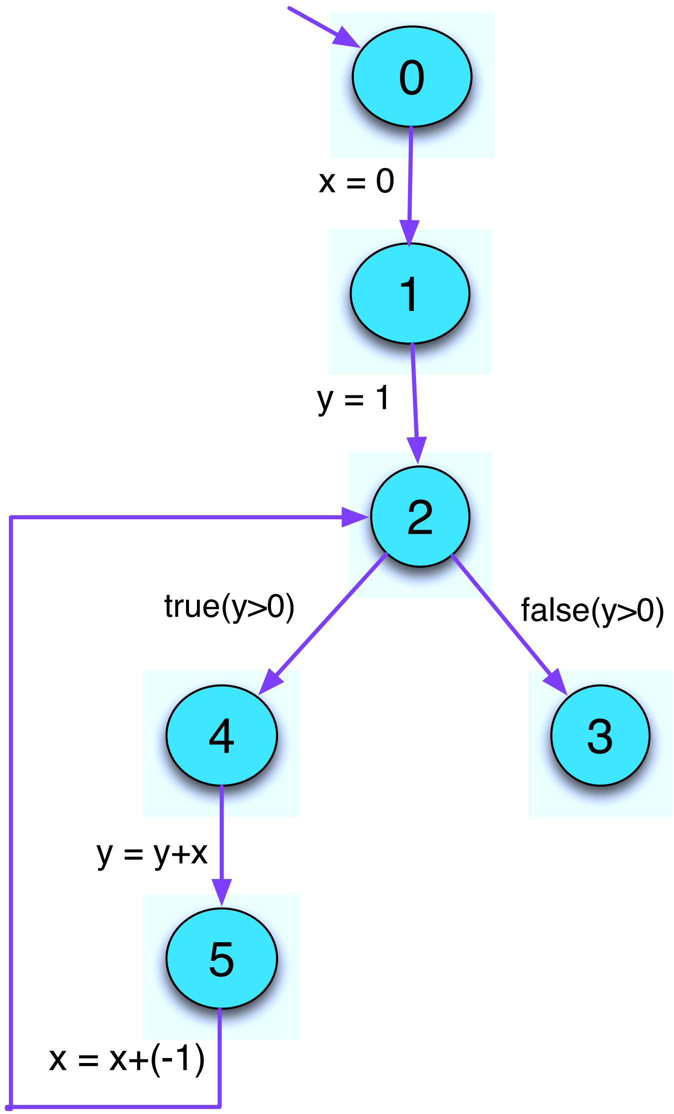
2 Example — Rules-of-Sign Analysis

Problem: Determine at each program point the sign of the values of all variables of numeric type.

Example program:

```
x = 0;
y = 1;
while (y > 0) do
    y = y + x;
    x = x + (-1);
```

Program representation as *control-flow graphs*



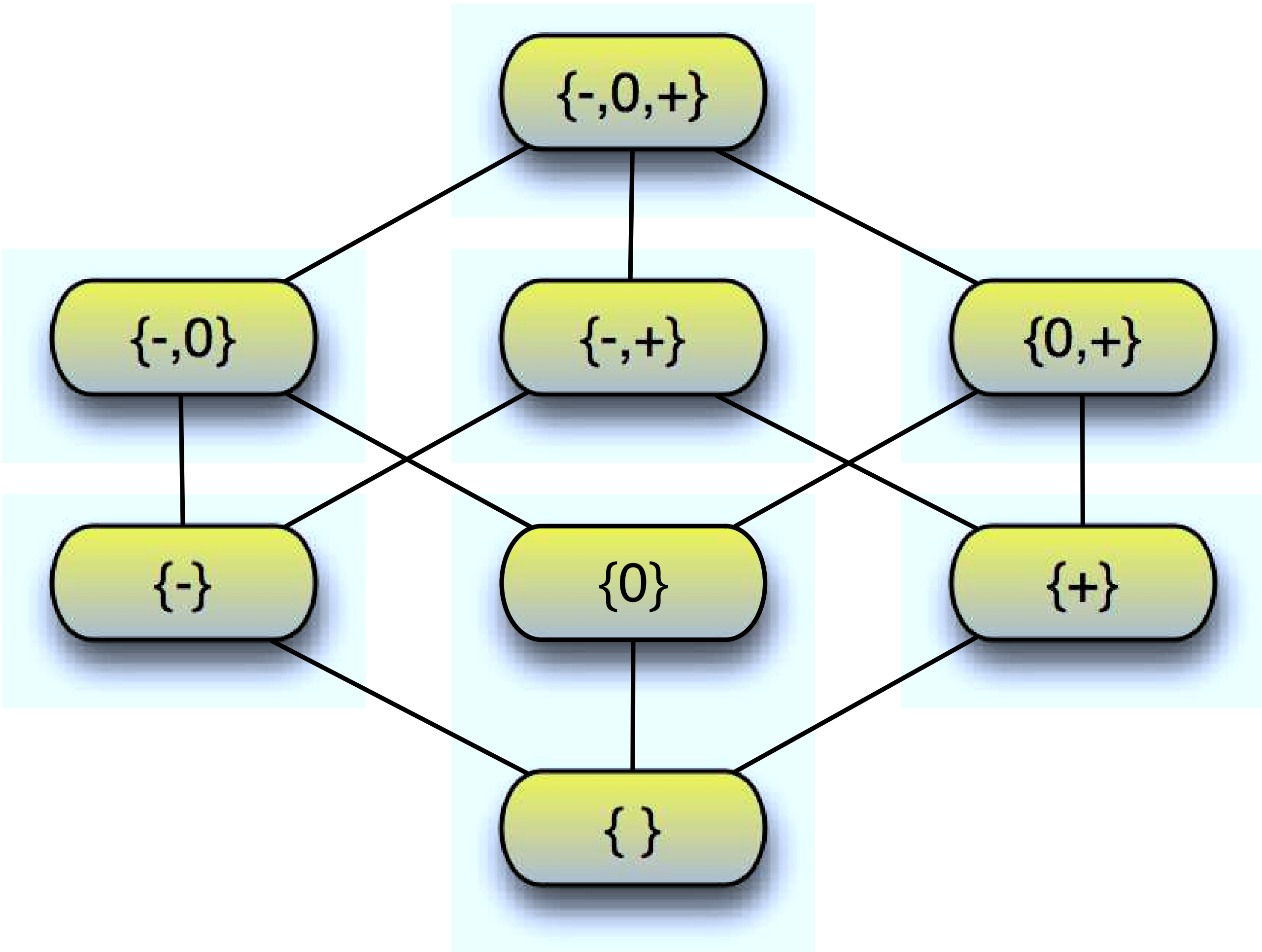
What are the ingredients that we need?

More ingredients?

All the ingredients:

- a set of information elements, each a set of possible signs,
- a partial order, "$\sqsubseteq$", on these elements, specifying the "relative strength" of two information elements,
- these together form the abstract domain, a lattice,
- functions describing how signs of variables change by the execution of a statement, abstract edge effects,
- these need an abstract arithmetic, an arithmetic on signs.

We construct the abstract domain for single variables starting with the lattice $\quad Signs = 2^{\{-,0,+\}}$ with the relation "$\sqsubseteq$" ="$\supseteq$".



The analysis should "bind" program variables to elements in Signs.

So, the abstract domain is $\mathbb{D} = (\mathit{Vars} \rightarrow \mathit{Signs})_\perp$, a Sign-environment.

$\perp \in \mathbb{D}$ is the function mapping all arguments to $\{\}$.

The partial order on $\mathbb{D}$ is $\quad D_1 \sqsubseteq D_2 \quad$ iff

$$D_1 = \perp \qquad \text{or}$$

$$D_1\, x \supseteq D_2\, x \qquad (x \in \mathit{Vars})$$

Intuition?

The analysis should ”bind” program variables to elements in Signs.

So, the abstract domain is $\mathbb{D} = (\mathit{Vars} \to \mathit{Signs})_\perp$. a Sign-environment.

$\perp \in \mathbb{D}$ is the function mapping all arguments to $\{\}$.

The partial order on $\mathbb{D}$ is $\quad D_1 \sqsubseteq D_2 \quad$ iff

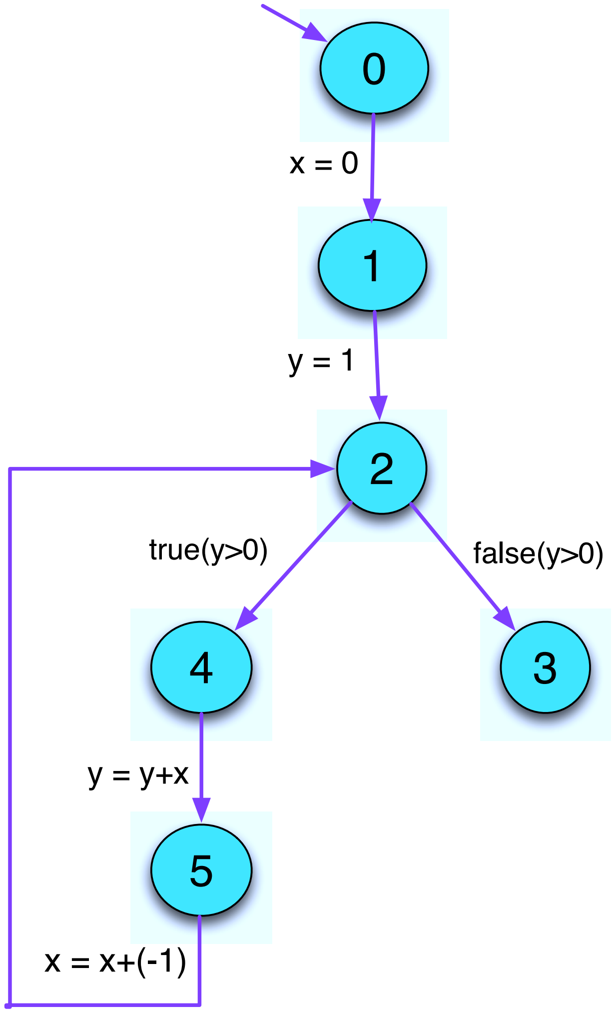
$$D_1 = \perp \qquad \text{or}$$

$$D_1\,x \supseteq D_2\,x \qquad (x \in \mathit{Vars})$$

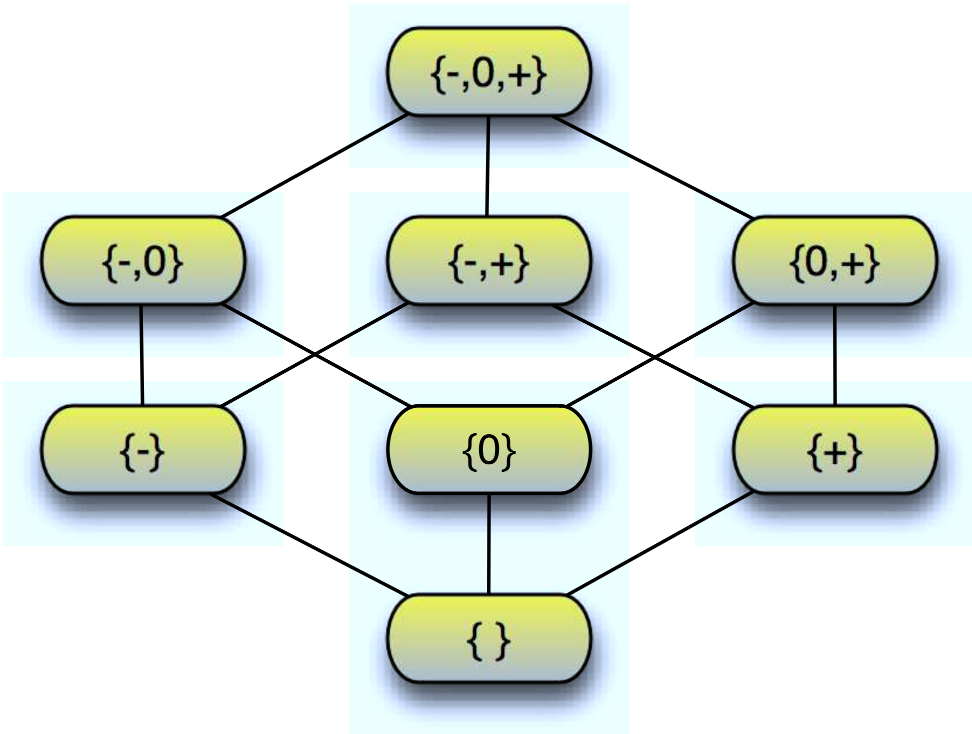
Intuition?

D_1 is at least as precise as D_2 since D_2 admits at least as many signs as D_1

How did we analyze the program?

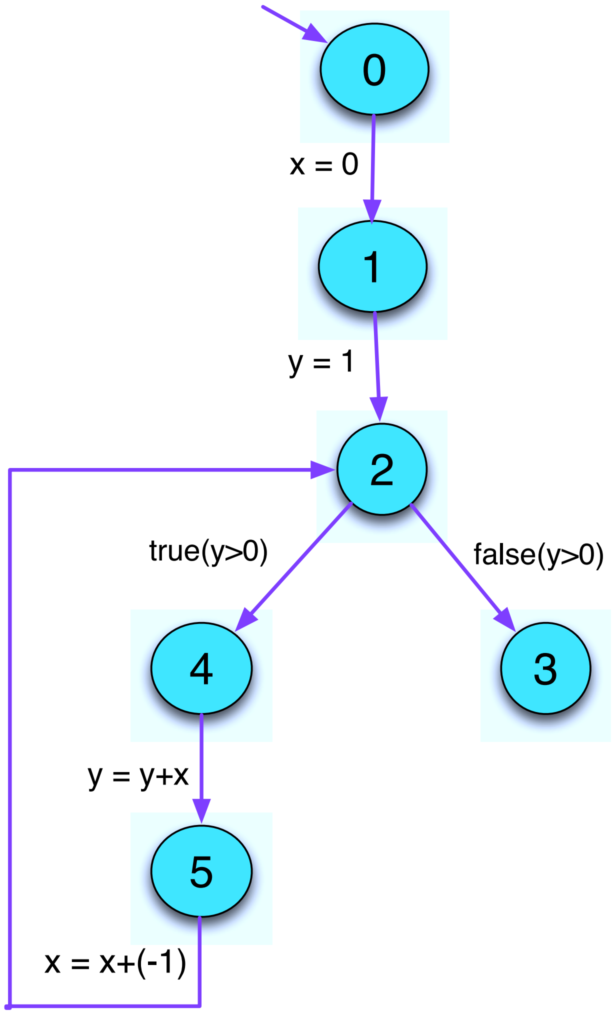


In particular, how did we walk the lattice for y at program point 5?



How is a solution found?

Iterating until a fixed-point is reached



0		1		2		3		4		5	
x	y	x	y	x	y	x	y	x	y	x	y

Idea:

- We want to determine the sign of the values of expressions.

Idea:

- We want to determine the sign of the values of expressions.
- For some sub-expressions, the analysis may yield $\{+,-,0\}$, which means, it couldn't find out.

Idea:

- We want to determine the signs of the values of expressions.
- For some sub-expressions, the analysis may yield $\{+, -, 0\}$, which means, it couldn't find out.
- We replace the concrete operators $\Box$ working on values by abstract operators $\Box^\sharp$ working on signs:

Idea:

- We want to determine the signs of the values of expressions.
- For some sub-expressions, the analysis may yield $\{+, -, 0\}$, which means, it couldn't find out.
- We replace the concrete operators $\Box$ working on values by abstract operators $\Box^\sharp$ working on signs:
- The abstract operators allow to define an abstract evaluation of expressions:

$$[\![e]\!]^\sharp \;:\; (\mathit{Vars} \to \mathit{Signs}) \to \mathit{Signs}$$

Determining the sign of expressions in a Sign-environment works as follows:

$$
\begin{array}{lcl}
[\![c]\!]^\sharp\, D & = & \begin{cases} \{+\} & \text{if c} > 0 \\ \{-\} & \text{if c} < 0 \\ \{0\} & \text{if c} = 0 \end{cases} \\
[\![v]\!]^\sharp & = & D(v) \\
[\![e_1 \,\Box\, e_2]\!]^\sharp\, D & = & [\![e_1]\!]^\sharp\, D \,\Box^\sharp\, [\![e_2]\!]^\sharp\, D \\
[\![\Box e]\!]^\sharp\, D & = & \Box^\sharp [\![e]\!]^\sharp\, D
\end{array}
$$

Abstract operators working on signs (Addition)

$+^{\#}$	{0}	{+}	{-}	{-, 0}	{-, +}	{0, +}	{-, 0, +}
{0}	{0}	{+}					
{+}							
{-}							
{-, 0}							
{-, +}							
{0, +}							
{-, 0, +}	{-, 0, +}						

Abstract operators working on signs (Multiplication)

$\times^{\#}$	{0}	{+}	{-}	{-, 0}	{-, +}	{0, +}	{-, 0, +}
{0}	{0}	{0}					
{+}							
{-}							
{-, 0}							
{-, +}							
{0, +}							
{-, 0, +}	{0}						

Abstract operators working on signs (unary minus)

$-^{\#}$	{0}	{+}	{-}	{-, 0}	{-, +}	{0, +}	{-, 0, +}
	{0}	{-}	{+}	{+, 0}	{-, +}	{0, -}	{-, 0, +}

Working an example: $D = \{x \mapsto \{+\}, y \mapsto \{+\}\}$

$$
\begin{aligned}
[\![x + 7]\!]^\sharp\, D &= [\![x]\!]^\sharp\, D \;+^\sharp\; [\![7]\!]^\sharp\, D \\
&= \{+\} \;+^\sharp\; \{+\} \\
&= \{+\}
\end{aligned}
$$

$$
\begin{aligned}
[\![x + (-y)]\!]^\sharp\, D &= \{+\} \;+^\sharp\; (-^\sharp [\![y]\!]^\sharp\, D\,) \\
&= \{+\} \;+^\sharp\; (-^\sharp \{+\}) \\
&= \{+\} \;+^\sharp\; \{-\} \\
&= \{+, -, 0\}
\end{aligned}
$$

$[\![lab]\!]^\sharp$ is the abstract edge effects associated with edge k.

It depends only on the label *lab*:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, D & = & D \\
[\![\text{true}\,(e)]\!]^\sharp\, D & = & D \\
[\![\text{false}\,(e)]\!]^\sharp\, D & = & D \\
[\![x = e;]\!]^\sharp\, D & = & D \oplus \{x \mapsto [\![e]\!]^\sharp\, D\} \\
[\![x = M[e];]\!]^\sharp\, D & = & D \oplus \{x \mapsto \{+, -, 0\}\} \\
[\![M[e_1] = e_2;]\!]^\sharp\, D & = & D
\end{array}
$$

... whenever $\qquad D \neq \bot$

These edge effects can be composed to the effect of a path $\pi = k_1 \dots k_r$:

$$[\![\pi]\!]^\sharp = [\![k_r]\!]^\sharp \circ \dots \circ [\![k_1]\!]^\sharp$$

Consider a program node v:

- $\rightarrow$ For every path π from program entry *start* to v the analysis should determine for each program variable x the set of all signs that the values of x may have at v as a result of executing π.
- $\rightarrow$ Initially at program start, no information about signs is available.
- $\rightarrow$ The analysis computes a superset of the set of signs as safe information.

$\Longrightarrow$ For each node v, we need the set:

$$\mathcal{S}[v] \quad = \quad \bigsqcup\{[\![\pi]\!]^\sharp \top \mid \pi : start \rightarrow^* v\}$$

Question:

How do we compute $\mathcal{S}[u]$ for every program point u?

Question:

How can we compute $\mathcal{S}[u]$ for every program point? u

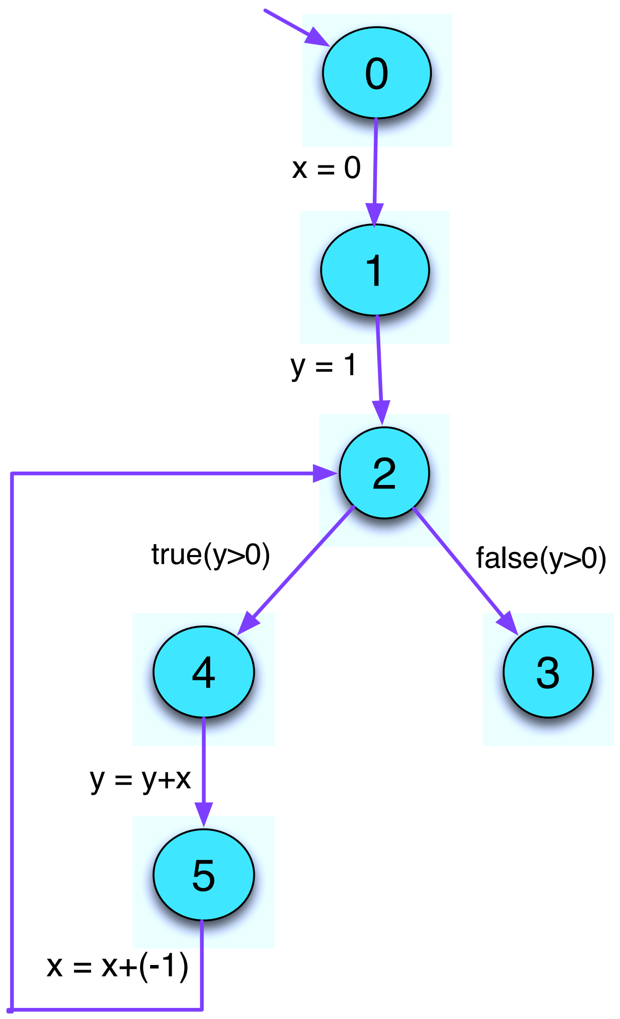
Collect all constraints on the values of $\mathcal{S}[u]$ into a system of constraints:

$$
\begin{array}{lll}
\mathcal{S}[start] & \sqsupseteq & \top \\
\mathcal{S}[v] & \sqsupseteq & [\![k]\!]^\sharp\,(\mathcal{S}[u]) \qquad\qquad k = (u, \_, v) \quad \text{edge}
\end{array}
$$

Wanted:

- a least solution (why least?)
- an algorithm that computes this solution

Example:



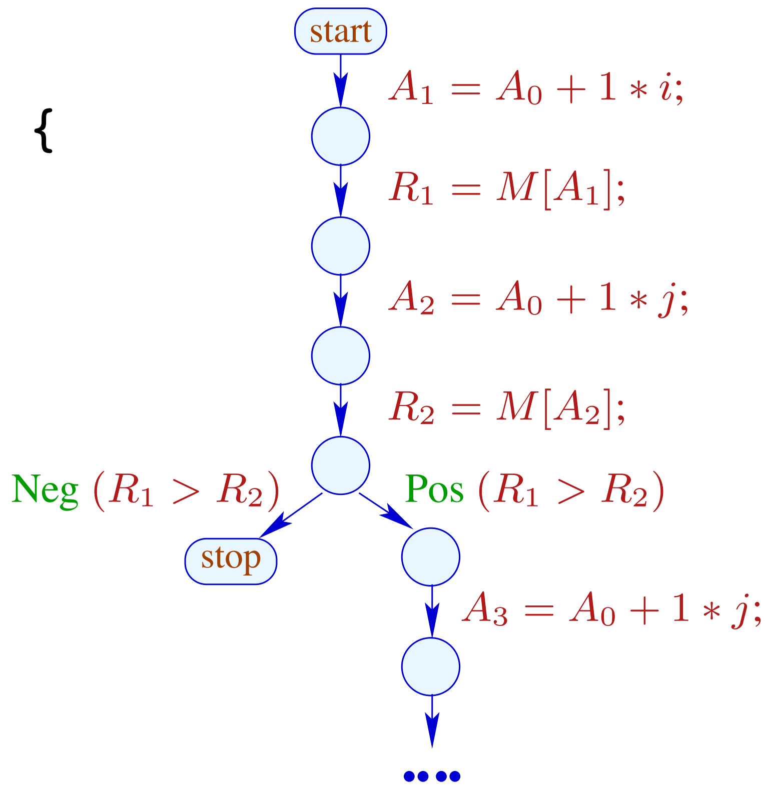
$$
\begin{array}{lcl}
\mathcal{S}[0] & \supseteq & \top \\
\mathcal{S}[1] & \supseteq & \mathcal{S}[0] \oplus \{x \mapsto \{0\}\} \\
\mathcal{S}[2] & \supseteq & \mathcal{S}[1] \oplus \{y \mapsto \{+\}\} \\
\mathcal{S}[2] & \supseteq & \mathcal{S}[5] \oplus \{x \mapsto [\![x + (-1)]\!]^\sharp \, \mathcal{S}[5]\} \\
\mathcal{S}[3] & \supseteq & \mathcal{S}[2] \\
\mathcal{S}[4] & \supseteq & \mathcal{S}[2] \\
\mathcal{S}[5] & \supseteq & \mathcal{S}[4] \oplus \{y \mapsto [\![y + x]\!]^\sharp \, \mathcal{S}[4]\}
\end{array}
$$

3 An Operational Semantics

Programs are represented as control-flow graphs.

Example:

```
void swap (int i, int j) {
    int t;
    if (a[i] > a[j]) {
        t = a[j];
        a[j] = a[i];
        a[i] = t;
      }
    }
```



Thereby, represent:

vertex	program point
start	program start
stop	program exit
edge	labeled with a statement or a condition

Thereby, represent:

vertex	program point
start	program start
stop	program exit
edge	step of computation

Edge Labelings:

Test : Pos (e) or Neg (e)

Assignment : $R = e;$

Load : $R = M[e];$

Store : $M[e_1] = e_2;$

Nop : ;

Execution of a path is a computation.

A computation transforms a state $s = (\rho, \mu)$ where:

$\rho : \mathit{Vars} \to \mathbf{int}$	values of variables (contents of symbolic registers)
$\mu : \mathbb{N} \to \mathbf{int}$	contents of memory

Every edge $k = (u, \mathit{lab}, v)$ defines a partial transformation

$$[\![k]\!] = [\![\mathit{lab}]\!]$$

of the state:

$$
\begin{aligned}
\llbracket ;\, \rrbracket\,(\rho,\mu) &= (\rho,\mu) \\[1em]
\llbracket \mathrm{Pos}\,(e) \rrbracket\,(\rho,\mu) &= (\rho,\mu) && \text{if } \llbracket e \rrbracket\,\rho \neq 0 \\
\llbracket \mathrm{Neg}\,(e) \rrbracket\,(\rho,\mu) &= (\rho,\mu) && \text{if } \llbracket e \rrbracket\,\rho = 0
\end{aligned}
$$

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\
 & & & \\
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x+y]\!]\,\{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!]\,\{x \mapsto 5\} = 1$

$$
\begin{array}{lcll}
[\![;]\!] \, (\rho, \mu) & = & (\rho, \mu) & \\
\\
[\![\mathrm{Pos}\,(e)]\!] \, (\rho, \mu) & = & (\rho, \mu) & \text{if } [\![e]\!] \, \rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!] \, (\rho, \mu) & = & (\rho, \mu) & \text{if } [\![e]\!] \, \rho = 0
\end{array}
$$

// $[\![e]\!]$: evaluation of the expression e, e.g.

// $[\![x + y]\!] \, \{x \mapsto 7, y \mapsto -1\} = 6$

// $[\![!(x == 4)]\!] \, \{x \mapsto 5\} = 1$

$$
[\![R = e;]\!] \, (\rho, \mu) \quad = \quad (\boxed{\rho \oplus \{R \mapsto [\![e]\!] \, \rho\}}, \mu)
$$

// where "$\oplus$" modifies a mapping at a given argument

$$\begin{array}{lcl} [\![R = M[e];]\!]\,(\rho,\mu) & = & (\boxed{\rho \oplus \{R \mapsto \mu([\![e]\!]\,\rho))\}}, \mu) \\ [\![M[e_1] = e_2;]\!]\,(\rho,\mu) & = & (\rho, \boxed{\mu \oplus \{[\![e_1]\!]\,\rho \mapsto [\![e_2]\!]\,\rho\}}) \end{array}$$

Example:

$[\![x = x + 1;]\!]\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where

$$\begin{array}{lcl} \rho & = & \{x \mapsto 5\} \oplus \{x \mapsto [\![x+1]\!]\,\{x \mapsto 5\}\} \\ & = & \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\ & = & \{x \mapsto 6\} \end{array}$$

A path $\pi = k_1 k_2 \ldots k_m$ defines a computation in the state s if

$$s \in \mathit{def}\,([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])$$

The result of the computation is $[\![\pi]\!]\,s = ([\![k_m]\!] \circ \ldots \circ [\![k_1]\!])\,s$

The approach:

A static analysis needs to collect correct and hopefully precise information about a program in a terminating computation.

Concepts:

- partial orders relate information for their contents/quality/precision,
- least upper bounds combine information in the best possible way,
- monotonic functions preserve the order, prevent loss of collected information, prevent oscillation.

4 Complete Lattices

A set $\mathbb{D}$ together with a relation $\sqsubseteq \; \subseteq \; \mathbb{D} \times \mathbb{D}$ is a partial order if for all $a, b, c \in \mathbb{D}$,

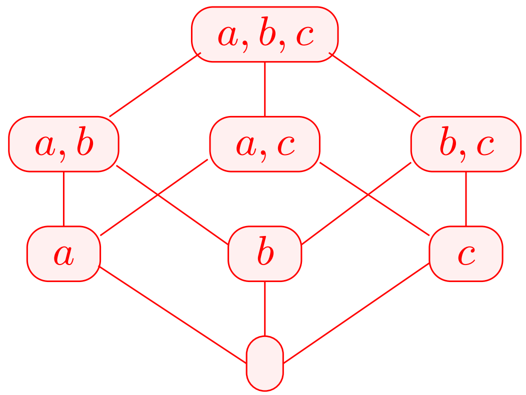
$$
\begin{array}{lll}
a \sqsubseteq a & & \mathit{reflexivity} \\
a \sqsubseteq b \wedge b \sqsubseteq a & \Longrightarrow \quad a = b & \mathit{anti-symmetry} \\
a \sqsubseteq b \wedge b \sqsubseteq c & \Longrightarrow \quad a \sqsubseteq c & \mathit{transitivity}
\end{array}
$$

Intuition: $\sqsubseteq$ represents precision.

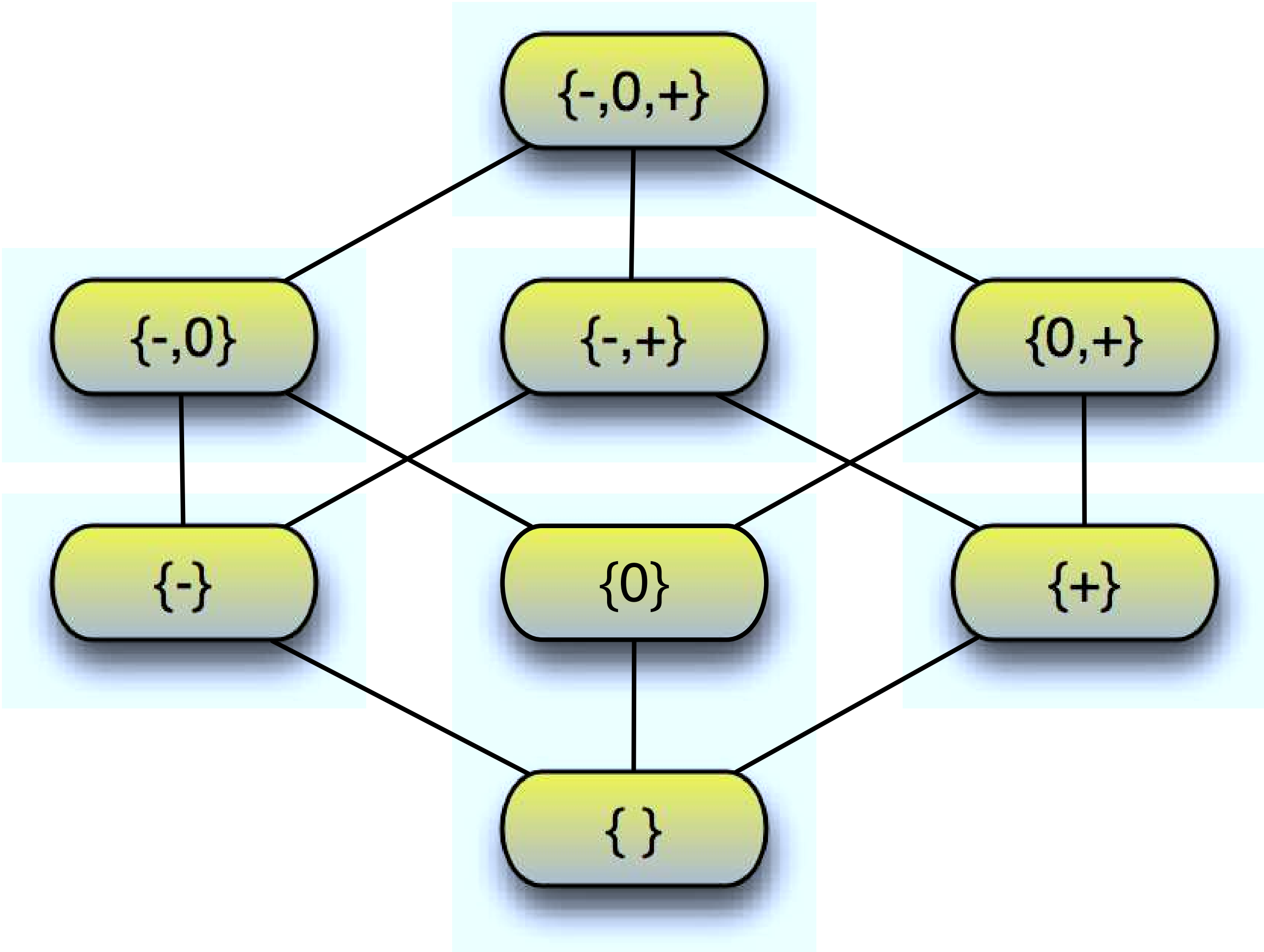
By convention: $a \sqsubseteq b$ means a is at least as precise as b.

Examples:

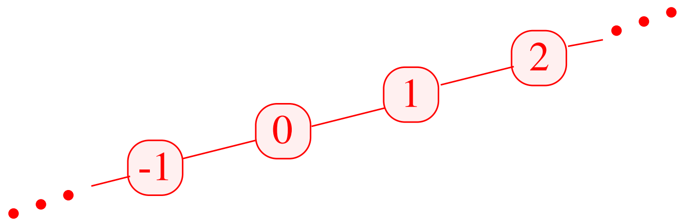
1. $\mathbb{D} = 2^{\{a,b,c\}}$ with the relation "$\subseteq$" :



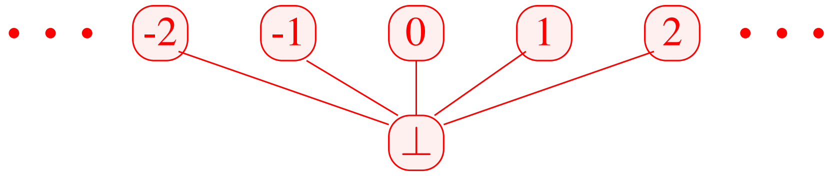
2. The rules-of-sign analysis uses the following lattice $\mathbb{D} = 2^{\{-,0,+\}}$ with the relation "$\subseteq$" :



3. $\mathbb{Z}$ with the relation "$\leq$" :



4. $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$ with the ordering:



$d \in \mathbb{D}$ is called upper bound for $X \subseteq \mathbb{D}$ if

$$x \sqsubseteq d \qquad \text{for all } x \in X$$

$d \in \mathbb{D}$ is called upper bound for $X \subseteq \mathbb{D}$ if

$$x \sqsubseteq d \qquad \text{for all } x \in X$$

d is called least upper bound (lub) if

1. d is an upper bound and
2. $d \sqsubseteq y$ for every upper bound y of X.

$d \in \mathbb{D}$ is called upper bound for $X \subseteq \mathbb{D}$ if

$$x \sqsubseteq d \qquad \text{for all } x \in X$$

d is called least upper bound (lub) if

1. d is an upper bound and
2. $d \sqsubseteq y$ for every upper bound y of X.

The least upper bound is the youngest common ancestor in the pictorial representation of lattices.

Intuition: It is the best combined information for X.

Caveat:

- $\{0, 2, 4, \ldots\} \subseteq \mathbb{Z}$ has no upper bound!
- $\{0, 2, 4\} \subseteq \mathbb{Z}$ has the upper bounds $4, 5, 6, \ldots$

A partially ordered set $\mathbb{D}$ is a complete lattice (cl) if every subset $X \subseteq \mathbb{D}$ has a least upper bound $\bigsqcup X \in \mathbb{D}$.

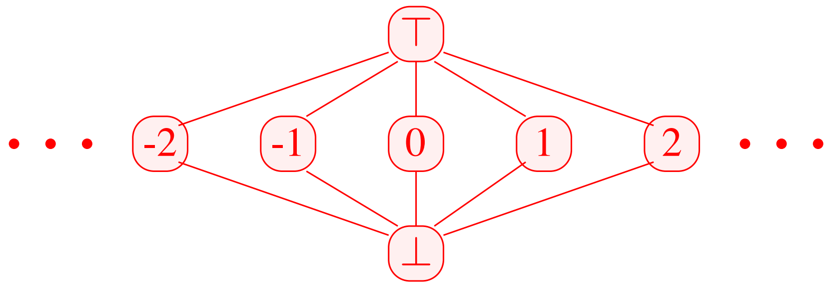
Note:

Every complete lattice has

- $\rightarrow$ a least element $\bot = \bigsqcup \emptyset \in \mathbb{D}$;
- $\rightarrow$ a greatest element $\top = \bigsqcup \mathbb{D} \in \mathbb{D}$.

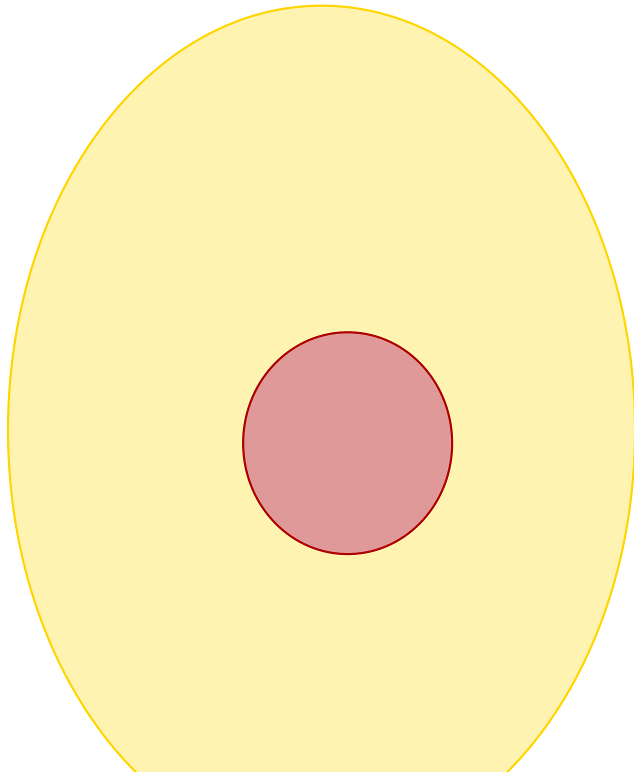
Examples:

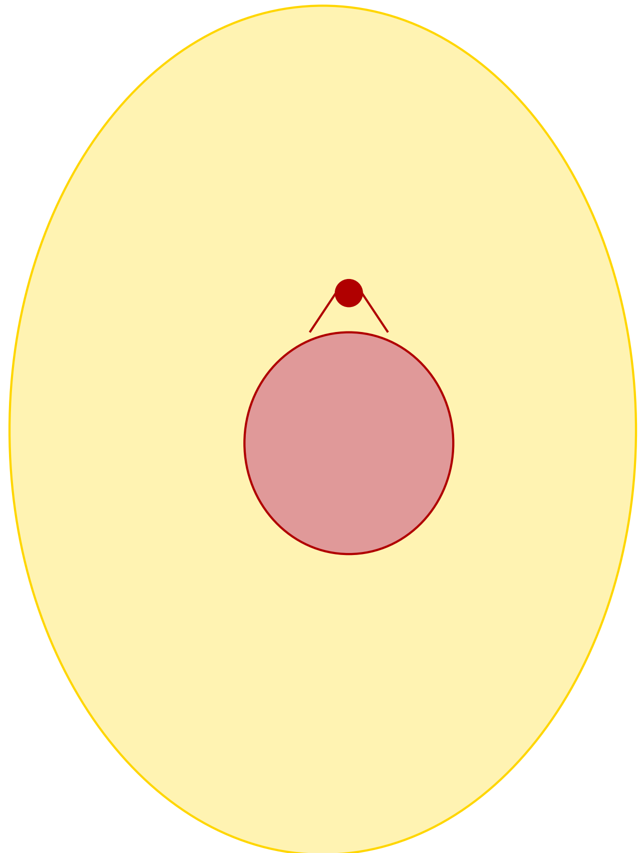
1. $\mathbb{D} = 2^{\{a,b,c\}}$ is a complete lattice
2. $\mathbb{D} = \mathbb{Z}$ with "$\leq$" is not a complete lattice.
3. $\mathbb{D} = \mathbb{Z}_\perp$ is also not a complete lattice
4. With an extra element $\top$, we obtain the flat lattice
   $\mathbb{Z}_\perp^\top = \mathbb{Z} \cup \{\perp, \top\}$:

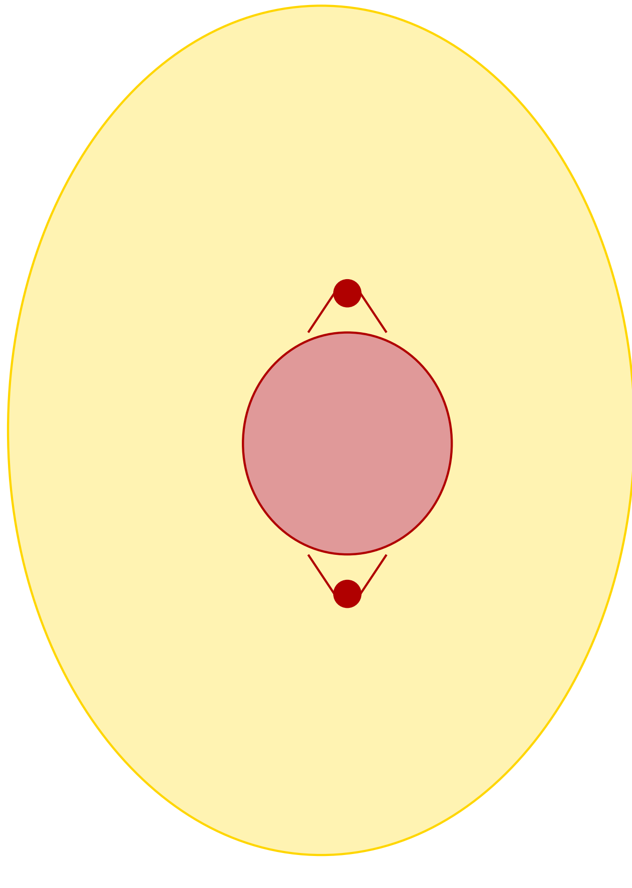


Theorem:

If $\mathbb{D}$ is a complete lattice, then every subset $X \subseteq \mathbb{D}$ has a greatest lower bound $\bigsqcap X$.







Back to the system of constraints for Rules-of-Signs Analysis!

$$\begin{array}{lll} \mathcal{S}[start] & \sqsupseteq & \top \\ \mathcal{S}[v] & \sqsupseteq & [\![k]\!]^\sharp\,(\mathcal{S}[u]) \qquad\qquad k = (u, \_, v) \quad \text{edge} \end{array}$$

Combine all constraints for each variable v by applying the least-upper-bound operator $\bigsqcup$:

$$\mathcal{S}[v] \quad \sqsupseteq \quad \bigsqcup\{[\![k]\!]^\sharp\,(\mathcal{S}[u]) \mid k = (u, \_, v) \;\; \text{edge}\}$$

Correct because:

$$x \sqsupseteq d_1 \wedge \ldots \wedge x \sqsupseteq d_k \quad \text{iff} \quad x \sqsupseteq \bigsqcup\{d_1, \ldots, d_k\}$$

Our generic form of the systems of constraints:

$$x_i \quad \sqsupseteq \quad f_i(x_1, \ldots, x_n) \qquad\qquad (*)$$

Relation to the running example:

x_i	unknown	here:	$\mathcal{S}[u]$
$\mathbb{D}$	values	here:	$Signs$
$\sqsubseteq \quad \subseteq \quad \mathbb{D} \times \mathbb{D}$	ordering relation	here:	$\subseteq$
f_i: $\mathbb{D}^n \to \mathbb{D}$	constraint	here:	...

A mapping $f : \mathbb{D}_1 \to \mathbb{D}_2$ is called monotonic (order preserving) if $f(a) \sqsubseteq f(b)$ for all $a \sqsubseteq b$.

A mapping $f : \mathbb{D}_1 \to \mathbb{D}_2$ is called monotonic (order preserving) if $f(a) \sqsubseteq f(b)$ for all $a \sqsubseteq b$.

Examples:

(1) $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$ for a set U and $f\,x = (x \cap a) \cup b$.

Obviously, every such f is monotonic

A mapping $f : \mathbb{D}_1 \to \mathbb{D}_2$ is called monotonic (order preserving) if $f(a) \sqsubseteq f(b)$ for all $a \sqsubseteq b$.

Examples:

(1) $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$ for a set U and $f\,x = (x \cap a) \cup b$.

Obviously, every such f is monotonic

(2) $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{Z}$ (with the ordering "$\leq$"). Then:

- $\mathsf{inc}\,x = x + 1$ is monotonic.
- $\mathsf{dec}\,x = x - 1$ is monotonic.

A mapping $f : \mathbb{D}_1 \to \mathbb{D}_2$ is called monotonic (order preserving) if $f(a) \sqsubseteq f(b)$ for all $a \sqsubseteq b$.

Examples:

(1) $\mathbb{D}_1 = \mathbb{D}_2 = 2^U$ for a set U and $f\,x = (x \cap a) \cup b$.

Obviously, every such f is monotonic

(2) $\mathbb{D}_1 = \mathbb{D}_2 = \mathbb{Z}$ (with the ordering "$\leq$"). Then:

- $\mathsf{inc}\,x = x + 1$ is monotonic.
- $\mathsf{dec}\,x = x - 1$ is monotonic.
- $\mathsf{inv}\,x = -x$ is not monotonic

Theorem:

If $f_1 : \mathbb{D}_1 \to \mathbb{D}_2$ and $f_2 : \mathbb{D}_2 \to \mathbb{D}_3$ are monotonic, then also $f_2 \circ f_1 : \mathbb{D}_1 \to \mathbb{D}_3$

Theorem:

If $f_1 : \mathbb{D}_1 \to \mathbb{D}_2$ and $f_2 : \mathbb{D}_2 \to \mathbb{D}_3$ are monotonic, then also $f_2 \circ f_1 : \mathbb{D}_1 \to \mathbb{D}_3$

Theorem:

If $f_1 : \mathbb{D}_1 \to \mathbb{D}_2$ and $f_2 : \mathbb{D}_2 \to \mathbb{D}_3$ are monotonic, then also $f_2 \circ f_1 : \mathbb{D}_1 \to \mathbb{D}_3$

Wanted: least solution for:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n \qquad (*)$$

where all $f_i : \mathbb{D}^n \to \mathbb{D}$ are monotonic.

Wanted: least solution for:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n \qquad (*)$$

where all $f_i : \mathbb{D}^n \to \mathbb{D}$ are monotonic.

Idea:

- Consider $F : \mathbb{D}^n \to \mathbb{D}^n$ where

  $F(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ with $y_i = f_i(x_1, \ldots, x_n)$.

Wanted: least solution for:

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n \qquad (*)$$

where all $f_i : \mathbb{D}^n \to \mathbb{D}$ are monotonic.

Idea:

- Consider $F : \mathbb{D}^n \to \mathbb{D}^n$ where

  $$F(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \quad \text{with} \quad y_i = f_i(x_1, \ldots, x_n).$$

- If all f_i are monotonic, then also F

Wanted: least solution for

$$x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n \qquad (*)$$

where all $f_i : \mathbb{D}^n \to \mathbb{D}$ are monotonic.

Idea:

- Consider $F : \mathbb{D}^n \to \mathbb{D}^n$ where

  $$F(x_1, \ldots, x_n) = (y_1, \ldots, y_n) \quad \text{with} \quad y_i = f_i(x_1, \ldots, x_n).$$

- If all f_i are monotonic, then also F
- We successively approximate a solution from below. We construct:

  $$\underline{\perp}, \quad F\,\underline{\perp}, \quad F^2\,\underline{\perp}, \quad F^3\,\underline{\perp}, \quad \ldots$$

Intuition: This iteration eliminates unjustified assumptions.

Hope: We eventually reach a solution!

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq \; = \; \subseteq$

$$\begin{aligned} x_1 &\supseteq \{a\} \cup x_3 \\ x_2 &\supseteq x_3 \cap \{a, b\} \\ x_3 &\supseteq x_1 \cup \{c\} \end{aligned}$$

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq = \subseteq$

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$				
x_2	$\emptyset$				
x_3	$\emptyset$				

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq \, = \, \subseteq$

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$			
x_2	$\emptyset$	$\emptyset$			
x_3	$\emptyset$	$\{c\}$			

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq \, = \, \subseteq$

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$		
x_2	$\emptyset$	$\emptyset$	$\emptyset$		
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$		

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq \, = \, \subseteq$

$$\begin{array}{lll} x_1 & \supseteq & \{a\} \cup x_3 \\ x_2 & \supseteq & x_3 \cap \{a, b\} \\ x_3 & \supseteq & x_1 \cup \{c\} \end{array}$$

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{a, c\}$	
x_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$	$\{a, c\}$	

Example: $\mathbb{D} = 2^{\{a,b,c\}}, \quad \sqsubseteq = \subseteq$

$$
\begin{aligned}
x_1 &\supseteq \{a\} \cup x_3 \\
x_2 &\supseteq x_3 \cap \{a, b\} \\
x_3 &\supseteq x_1 \cup \{c\}
\end{aligned}
$$

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{a, c\}$	dito
x_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	
x_3	$\emptyset$	$\{c\}$	$\{a, c\}$	$\{a, c\}$	

Theorem

- $\underline{\perp}, F\,\underline{\perp}, F^2\,\underline{\perp}, \ldots$ form an ascending chain :

$$\underline{\perp} \quad \sqsubseteq \quad F\,\underline{\perp} \quad \sqsubseteq \quad F^2\,\underline{\perp} \quad \sqsubseteq \quad \ldots$$

- If $F^k\,\underline{\perp} = F^{k+1}\,\underline{\perp}$, F^k is the least solution.
- If all ascending chains are finite, such a k always exists.

Theorem

- $\underline{\perp}, F\,\underline{\perp}, F^2\,\underline{\perp}, \ldots$ form an ascending chain :

$$\underline{\perp} \quad \sqsubseteq \quad F\,\underline{\perp} \quad \sqsubseteq \quad F^2\,\underline{\perp} \quad \sqsubseteq \quad \ldots$$

- If $F^k\,\underline{\perp} = F^{k+1}\,\underline{\perp}$, a solution is obtained, which is the least one.
- If all ascending chains are finite, such a k always exists.

If $\mathbb{D}$ is finite, a solution can be found that is definitely the least solution.

Question: What, if $\mathbb{D}$ is not finite?

Theorem　　　　Knaster – Tarski

Assume $\mathbb{D}$ is a complete lattice. Then every monotonic function $f : \mathbb{D} \to \mathbb{D}$ has a least fixed point $d_0 \in \mathbb{D}$.

Application:

Assume $$x_i \sqsupseteq f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n \qquad (*)$$

is a system of constraints where all $f_i : \mathbb{D}^n \to \mathbb{D}$ are monotonic.

$\Longrightarrow$ least solution of $(*)$ $==$ least fixed point of F

Example 1: $\mathbb{D} = 2^U, \quad f\, x = x \cap a \cup b$

Example 1: $\mathbb{D} = 2^U, \quad f\,x = x \cap a \cup b$

f	$f^k\,\bot$	$f^k\,\top$
0	$\emptyset$	U

Example 1: $\mathbb{D} = 2^U, \quad f\,x = x \cap a \cup b$

f	$f^k\,\bot$	$f^k\,\top$
0	$\emptyset$	U
1	b	$a \cup b$

Example 1: $\quad \mathbb{D} = 2^U, \quad f\,x = x \cap a \cup b$

f	$f^k\,\bot$	$f^k\,\top$
0	$\emptyset$	U
1	b	$a \cup b$
2	b	$a \cup b$

Example 1: $\mathbb{D} = 2^U, \quad f\,x = x \cap a \cup b$

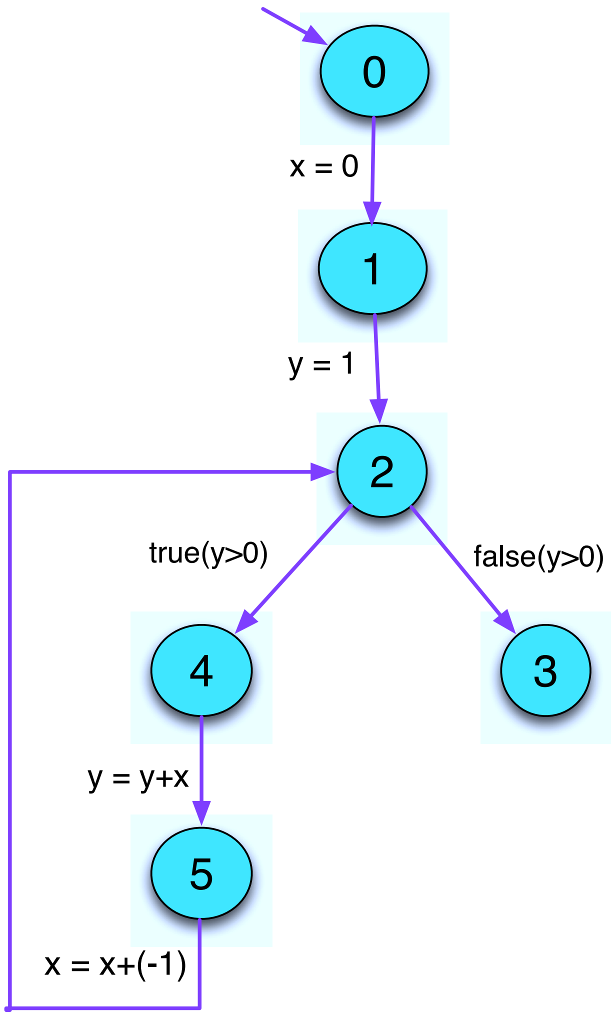
f	$f^k\,\bot$	$f^k\,\top$
0	$\emptyset$	U
1	b	$a \cup b$
2	b	$a \cup b$

Conclusion:

Systems of inequalities can be solved through fixed-point iteration, i.e., by repeated evaluation of right-hand sides

Caveat: Naive fixed-point iteration is rather inefficient

Example:

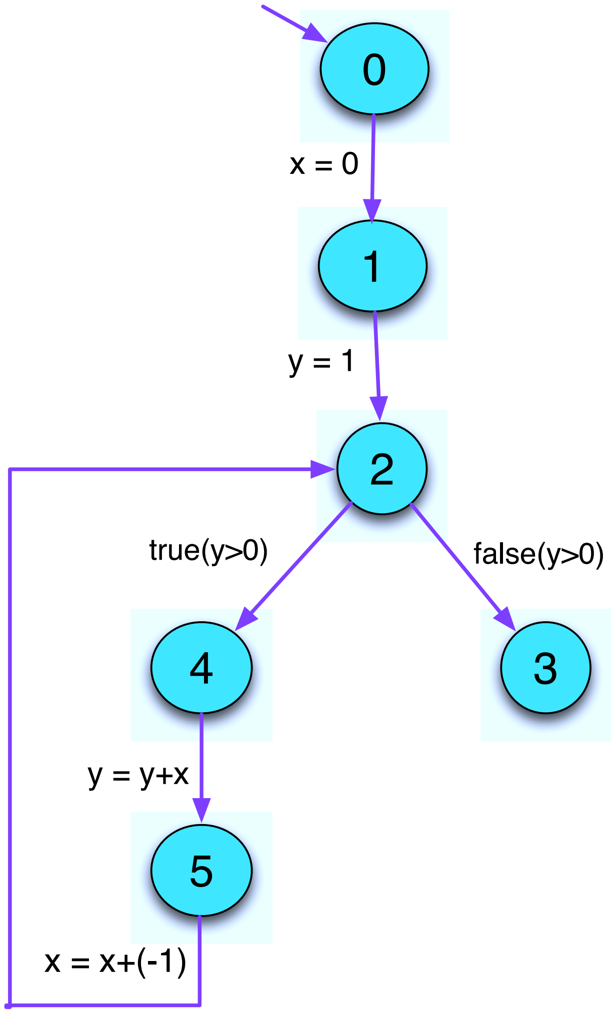


0		1		2		3		4		5	
x	y	x	y	x	y	x	y	x	y	x	y

Idea: Round Robin Iteration

Instead of accessing the values of the last iteration, always use the current values of unknowns

Example:



0		1		2		3		4		5	
x	y	x	y	x	y	x	y	x	y	x	y

The code for Round Robin Iteration in Java looks as follows:

$$
\begin{array}{l}
\textsf{for}\ (i = 1; i \leq n; i{+}{+})\ x_i = \bot; \\
\textsf{do}\ \{ \\
\qquad \mathit{finished} = \textsf{true}; \\
\qquad \textsf{for}\ (i = 1; i \leq n; i{+}{+})\ \{ \\
\qquad\qquad \mathit{new} = f_i(x_1, \ldots, x_n); \\
\qquad\qquad \textsf{if}\ (!(x_i \sqsupseteq \mathit{new}))\ \{ \\
\qquad\qquad\qquad \mathit{finished} = \textsf{false}; \\
\qquad\qquad\qquad x_i = x_i \sqcup \mathit{new}; \\
\qquad\qquad \} \\
\qquad \} \\
\}\ \textsf{while}\ (!\mathit{finished});
\end{array}
$$

What we have learned:

- The information derived by static program analysis is partially ordered in a complete lattice.
- the partial order represents information content/precision of the lattice elements.
- least upper-bound combines information in the best possible way.
- Monotone functions prevent loss of information.

For a complete lattice $\mathbb{D}$, consider systems:

$$\begin{array}{lll} \mathcal{I}[start] & \sqsupseteq & d_0 \\ \mathcal{I}[v] & \sqsupseteq & [\![k]\!]^\sharp\,(\mathcal{I}[u]) \qquad k = (u, \_, v) \quad \text{edge} \end{array}$$

where $d_0 \in \mathbb{D}$ and all $[\![k]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ are monotonic ...

Wanted: MOP (Merge Over all Paths)

$$\mathcal{I}^*[v] = \bigsqcup\{[\![\pi]\!]^\sharp\, d_0 \mid \pi : start \to^* v\}$$

Theorem Kam, Ullman 1975

Assume $\mathcal{I}$ is a solution of the constraint system. Then:

$$\mathcal{I}[v] \sqsupseteq \mathcal{I}^*[v] \qquad \text{for every} \quad v$$

In particular: $\mathcal{I}[v] \sqsupseteq [\![\pi]\!]^\sharp\, d_0$ for every $\pi : start \to^* v$

Disappointment: Are solutions of the constraint system just upper bounds?

Answer: In general: yes

Notable exception, all functions $[\![k]\!]^\sharp$ are distributive.

The function $f : \mathbb{D}_1 \to \mathbb{D}_2$ is called distributive, if
$f\,(\bigsqcup X) = \bigsqcup\{f\,x \mid x \in X\}$ for all $\emptyset \neq X \subseteq \mathbb{D}$;

Remark: If $f : \mathbb{D}_1 \to \mathbb{D}_2$ is distributive, then it is also monotonic

Theorem Kildall 1972

Assume all v are reachable from *start*.

Then: If all effects of edges $[\![k]\!]^\sharp$ are distributive, $\mathcal{I}^*[v] = \mathcal{I}[v]$ holds for all v.

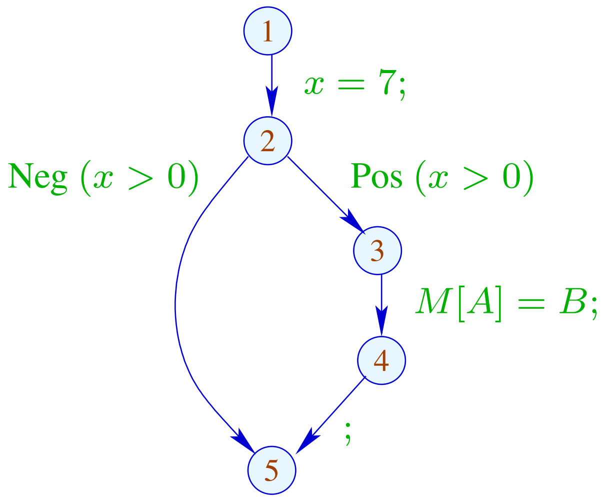
Question: Are the edge effects of the Rules-of-Sign analysis distributive?

5 Constant Propagation

Goal: Execute as much of the code at compile-time as possible!

Example:

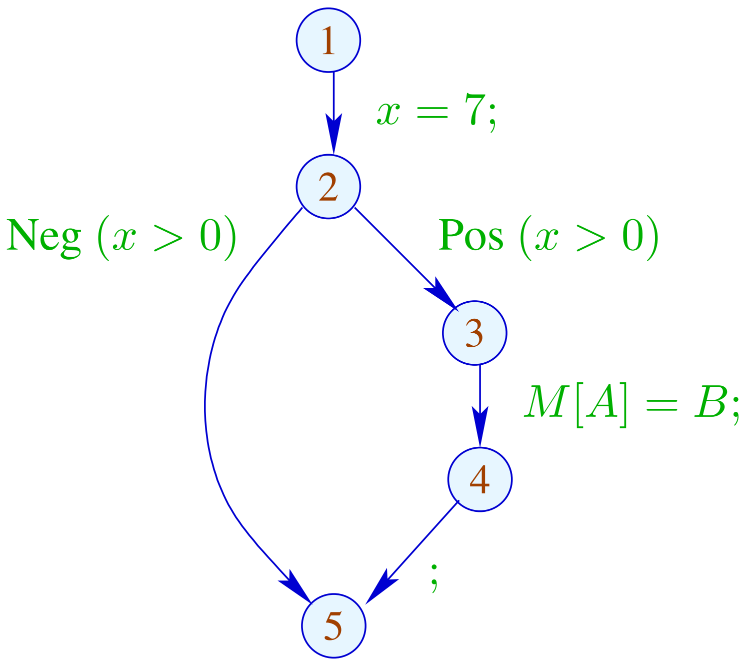
$$x = 7;$$
$$\text{if } (x > 0)$$
$$\quad M[A] = B;$$



Obviously, x has always the value 7

Thus, the memory access is always executed

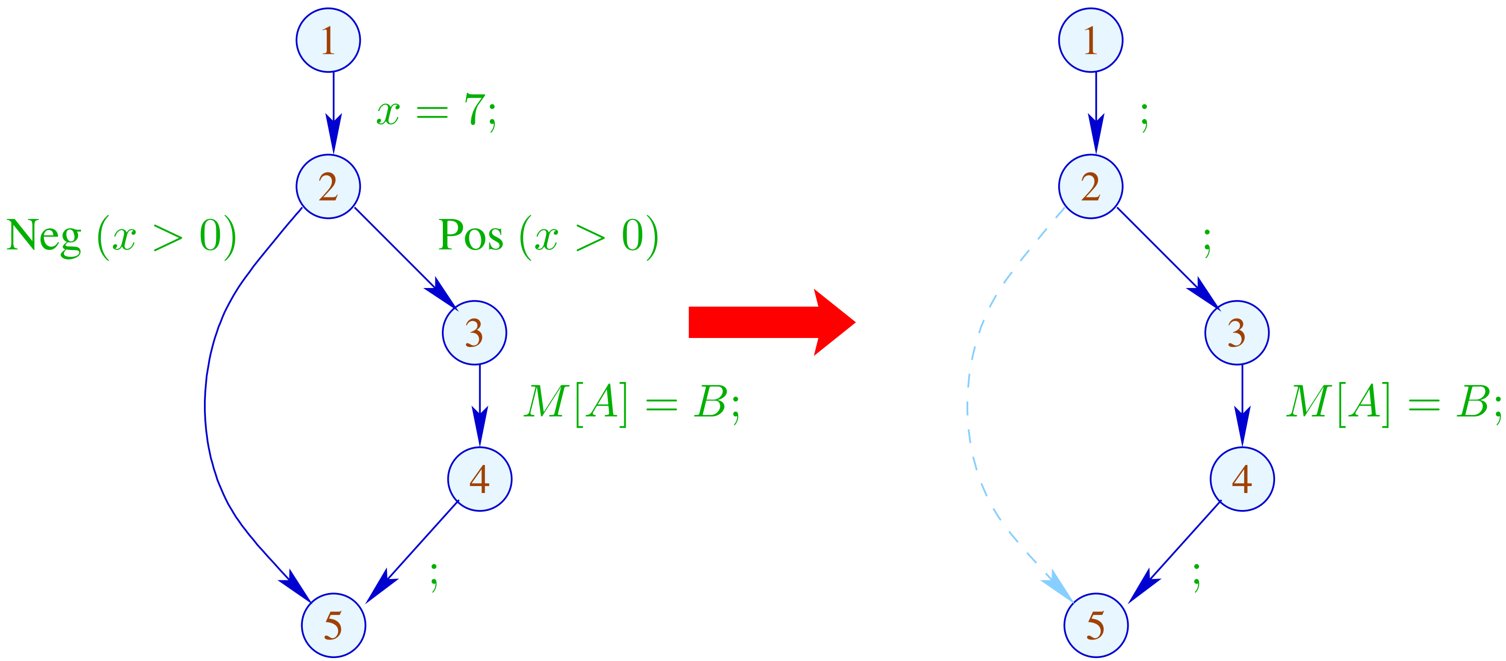
Goal:



Obviously, x has always the value 7

Thus, the memory access is always executed

Goal:



Idea:

Design an analysis that for every program point u, determines the values that variables definitely have at u;

As a side effect, it also tells whether u can be reached at all

Idea:

Design an analysis that for every program point u, determines the values that variables definitely have at u;

As a side effect, it also tells whether u can be reached at all

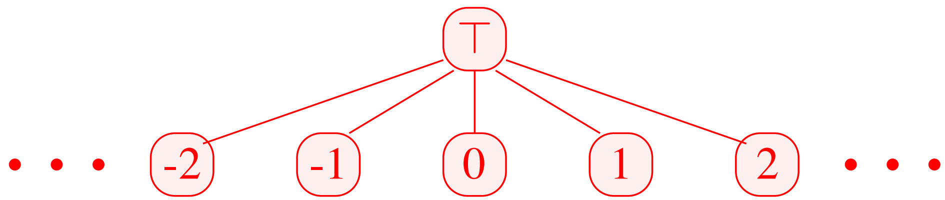
We need to design a complete lattice for this analysis.

It has a nice relation to the operational semantics of our tiny programming language.

As in the case of the Rules-of-Signs analysis the complete lattice is constructed in two steps.

(1) The potential values of variables:

$$\mathbb{Z}^{\top} = \mathbb{Z} \cup \{\top\} \qquad \text{with} \quad x \sqsubseteq y \quad \text{iff } y = \top \text{ or } x = y$$



Caveat: $\mathbb{Z}^\top$ is not a complete lattice in itself

(2) $\mathbb{D} = (\mathit{Vars} \to \mathbb{Z}^\top)_\perp = (\mathit{Vars} \to \mathbb{Z}^\top) \cup \{\perp\}$

// $\perp$ denotes: “not reachable”

with $D_1 \sqsubseteq D_2$ iff $\perp = D_1$ or

$D_1\, x \sqsubseteq D_2\, x \quad (x \in \mathit{Vars})$

Remark: $\mathbb{D}$ is a complete lattice

For every edge $k = (\_, lab, \_)$, construct an effect function $[\![k]\!]^\sharp = [\![lab]\!]^\sharp : \mathbb{D} \to \mathbb{D}$ which simulates the concrete computation.

Obviously, $[\![lab]\!]^\sharp \perp = \perp$ for all lab

Now let $\perp \neq D \in Vars \to \mathbb{Z}^\top$.

Idea:

- We use D to determine the values of expressions.

Idea:

- We use D to determine the values of expressions.
- For some sub-expressions, we obtain $\top$

Idea:

- We use D to determine the values of expressions.
- For some sub-expressions, we obtain $\top$

$\Longrightarrow$

We must replace the concrete operators $\Box$ by abstract operators $\Box^\sharp$ which can handle $\top$:

$$a \,\Box^\sharp\, b = \begin{cases} \top & \text{if} \quad a = \top \text{ or } b = \top \\ a \,\Box\, b & \text{otherwise} \end{cases}$$

Idea:

- We use D to determine the values of expressions.
- For some sub-expressions, we obtain $\top$

  $\Longrightarrow$

  We must replace the concrete operators $\Box$ by abstract operators $\Box^\sharp$ which can handle $\top$:

$$a \,\Box^\sharp\, b = \begin{cases} \top & \text{if} \quad a = \top \text{ or } b = \top \\ a \,\Box\, b & \text{otherwise} \end{cases}$$

- The abstract operators allow to define an abstract evaluation of expressions:

$$[\![e]\!]^\sharp \;:\; (\mathit{Vars} \to \mathbb{Z}^\top) \to \mathbb{Z}^\top$$

Abstract evaluation of expressions is like the concrete evaluation — but with abstract values and operators. Here:

$$
\begin{array}{lll}
[\![c]\!]^\sharp\, D & = & c \\
[\![e_1 \,\Box\, e_2]\!]^\sharp\, D & = & [\![e_1]\!]^\sharp\, D \,\Box^\sharp\, [\![e_2]\!]^\sharp\, D
\end{array}
$$

... analogously for unary operators

Abstract evaluation of expressions is like the concrete evaluation — but with abstract values and operators. Here:

$$
\begin{array}{lll}
[\![c]\!]^\sharp\, D & = & c \\
[\![e_1 \,\Box\, e_2]\!]^\sharp\, D & = & [\![e_1]\!]^\sharp\, D \,\Box^\sharp\, [\![e_2]\!]^\sharp\, D
\end{array}
$$

... analogously for unary operators

Example: $D = \{x \mapsto 2, y \mapsto \top\}$

$$
\begin{array}{lll}
[\![x+7]\!]^\sharp\, D & = & [\![x]\!]^\sharp\, D \,+^\sharp\, [\![7]\!]^\sharp\, D \\
 & = & 2 \,+^\sharp\, 7 \\
 & = & 9 \\
[\![x-y]\!]^\sharp\, D & = & 2 \,-^\sharp\, \top \\
 & = & \top
\end{array}
$$

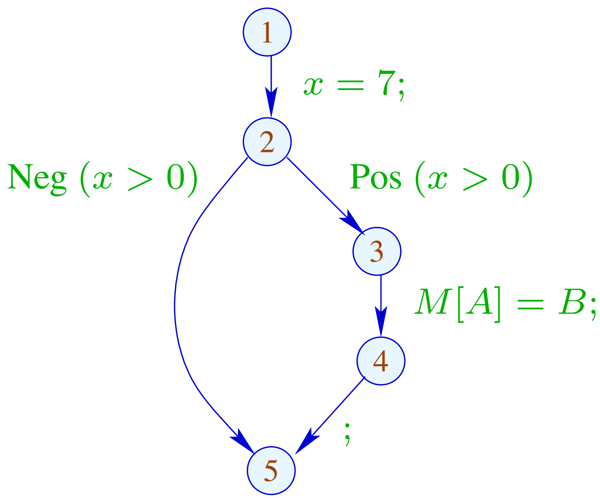
Thus, we obtain the following abstract edge effects $[\![lab]\!]^\sharp$:

$$
\begin{array}{lll}
[\![;]\!]^\sharp\, D & = & D \\
[\![\text{true}\,(e)]\!]^\sharp\, D & = & \left\{\begin{array}{lll} \bot & \text{if} \quad 0 = [\![e]\!]^\sharp\, D & \text{definitely false} \\ D & \text{otherwise} & \text{possibly true} \end{array}\right. \\
[\![\text{false}\,(e)]\!]^\sharp\, D & = & \left\{\begin{array}{lll} D & \text{if} \quad 0 \sqsubseteq [\![e]\!]^\sharp\, D & \text{possibly false} \\ \bot & \text{otherwise} & \text{definitely true} \end{array}\right. \\
[\![x = e;]\!]^\sharp\, D & = & D \oplus \{x \mapsto [\![e]\!]^\sharp\, D\} \\
[\![x = M[e];]\!]^\sharp\, D & = & D \oplus \{x \mapsto \top\} \\
[\![M[e_1] = e_2;]\!]^\sharp\, D & = & D
\end{array}
$$

... whenever $D \neq \bot$

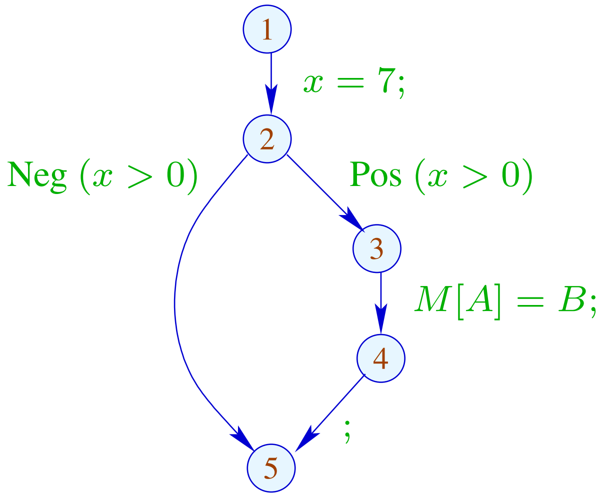
At *start*, we have $D_\top = \{x \mapsto \top \mid x \in \mathit{Vars}\}$.

Example:



At *start*, we have $D_\top = \{x \mapsto \top \mid x \in \mathit{Vars}\}$.

Example:



1	$\{x \mapsto \top\}$
2	$\{x \mapsto 7\}$
3	$\{x \mapsto 7\}$
4	$\{x \mapsto 7\}$
5	$\bot \sqcup \{x \mapsto 7\} = \{x \mapsto 7\}$

The abstract effects of edges $[\![k]\!]^\sharp$ are again composed to form the effects of paths $\pi = k_1 \ldots k_r$ by:

$$[\![\pi]\!]^\sharp = [\![k_r]\!]^\sharp \circ \ldots \circ [\![k_1]\!]^\sharp \quad : \mathbb{D} \to \mathbb{D}$$

Idea for Correctness: Abstract Interpretation

Cousot, Cousot 1977

Establish a description relation Δ between the concrete values and their descriptions with:

$$x \,\Delta\, a_1 \quad \wedge \quad a_1 \sqsubseteq a_2 \quad \Longrightarrow \quad x \,\Delta\, a_2$$

Concretization: $\gamma\, a = \{x \mid x \,\Delta\, a\}$

// returns the set of described values

(1) Values: $\Delta \subseteq \mathbb{Z} \times \mathbb{Z}^\top$

$$z \,\Delta\, a \quad \text{iff} \quad z = a \vee a = \top$$

Concretization:

$$\gamma\, a = \begin{cases} \{a\} & \text{if} \quad a \sqsubset \top \\ \mathbb{Z} & \text{if} \quad a = \top \end{cases}$$

(1) Values: $\Delta \subseteq \mathbb{Z} \times \mathbb{Z}^\top$

$$z \, \Delta \, a \quad \text{iff} \quad z = a \vee a = \top$$

Concretization:

$$\gamma \, a = \begin{cases} \{a\} & \text{if} \quad a \sqsubset \top \\ \mathbb{Z} & \text{if} \quad a = \top \end{cases}$$

(2) Variable Bindings: $\Delta \subseteq (\mathit{Vars} \to \mathbb{Z}) \times (\mathit{Vars} \to \mathbb{Z}^\top)_\perp$

$$\rho \, \Delta \, D \quad \text{iff} \quad D \neq \perp \wedge \rho \, x \sqsubseteq D \, x \quad (x \in \mathit{Vars})$$

Concretization:

$$\gamma \, D = \begin{cases} \emptyset & \text{if} \quad D = \perp \\ \{\rho \mid \forall \, x : \; (\rho \, x) \, \Delta \, (D \, x)\} & \text{otherwise} \end{cases}$$

Example: $\{x \mapsto 1, y \mapsto -7\} \;\Delta\; \{x \mapsto \top, y \mapsto -7\}$

(3) States:

$$\Delta \;\subseteq\; ((\mathit{Vars} \to \mathbb{Z}) \times (\mathbb{N} \to \mathbb{Z})) \times (\mathit{Vars} \to \mathbb{Z}^\top)_\perp$$

$$(\rho, \mu)\; \Delta\; D \qquad \text{iff} \qquad \rho\; \Delta\; D$$

Concretization:

$$\gamma\, D = \begin{cases} \emptyset & \text{if} \quad D = \perp \\ \{(\rho, \mu) \mid \forall\, x :\; (\rho\, x)\; \Delta\; (D\, x)\} & \text{otherwise} \end{cases}$$

We show correctness:

$(*)$ If $s \, \Delta \, D$ and $[\![\pi]\!]\, s$ is defined, then:

$$([\![\pi]\!]\, s) \;\; \Delta \;\; ([\![\pi]\!]^\sharp\, D)$$

$$\begin{tikzcd}
\boxed{s} \arrow[rrr, "{[\![\pi]\!]}"] \arrow[d, dash, "\Delta"'] & & & \boxed{s_1} \arrow[d, dash, dotted, "\Delta"] \\
\boxed{D} \arrow[rrr, "{[\![\pi]\!]^\sharp}"'] & & & \boxed{D_1}
\end{tikzcd}$$

The abstract semantics simulates the concrete semantics

In particular:

$$[\![\pi]\!]\ s \in \gamma\left([\![\pi]\!]^{\sharp}\ D\right)$$

The abstract semantics simulates the concrete semantics

In particular:

$$[\![\pi]\!]\, s \in \gamma\,([\![\pi]\!]^\sharp\, D)$$

In practice, this means for example that $D\, x = -7$ implies:

$$\begin{array}{lrcll} & \rho'\, x & = & -7 & \text{for all} \quad \rho' \in \gamma\, D \\ \Longrightarrow & \rho_1\, x & = & -7 & \text{for} \quad (\rho_1, \_) = [\![\pi]\!]\, s \end{array}$$

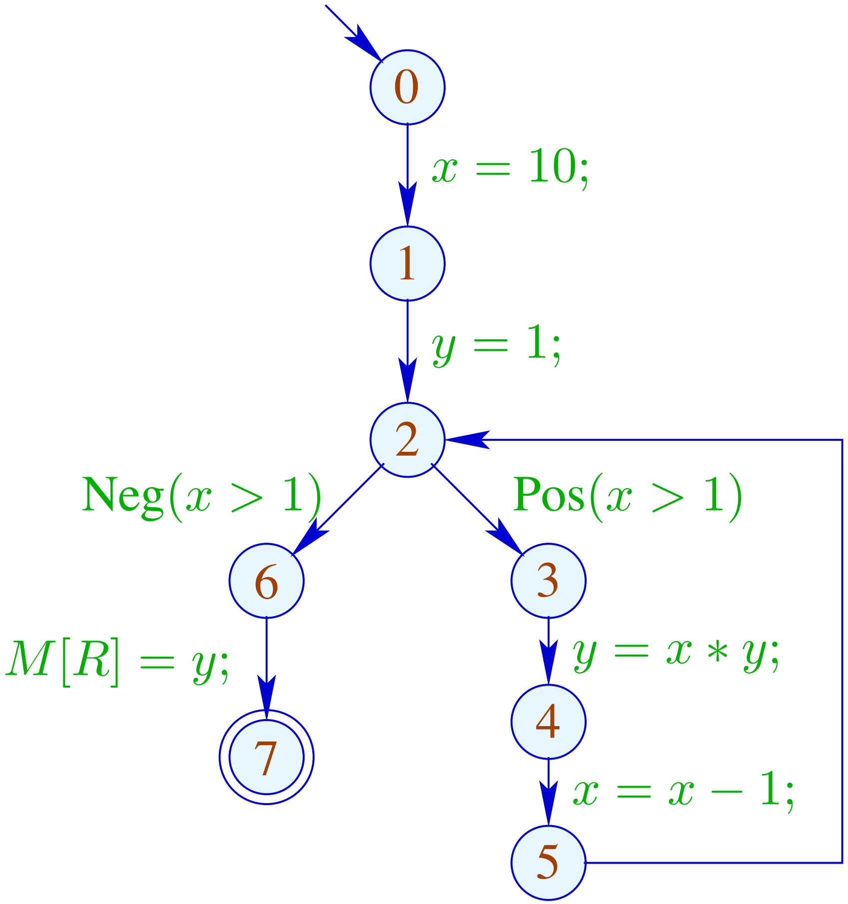
The MOP-Solution:

$$\mathcal{D}^*[v] \;=\; \bigsqcup\{[\![\pi]\!]^\sharp\, D_\top \mid \pi : start \to^* v\}$$

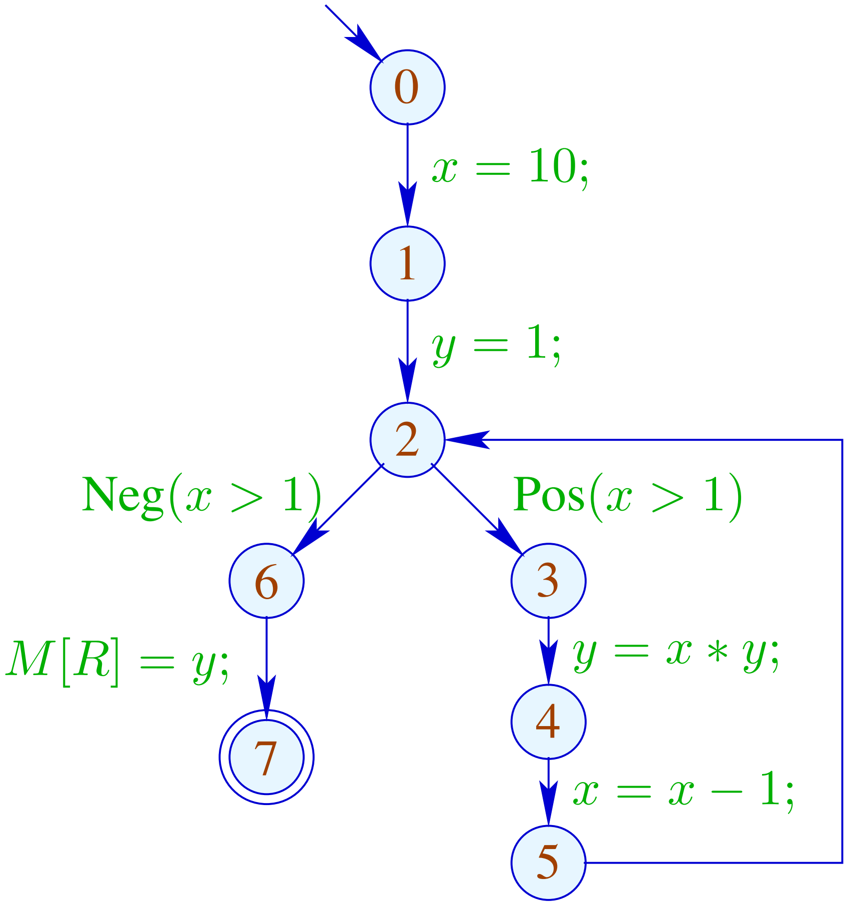
where $\qquad D_\top\, x = \top \qquad (x \in Vars)\;.$

In order to approximate the MOP, we use our constraint system

Example:

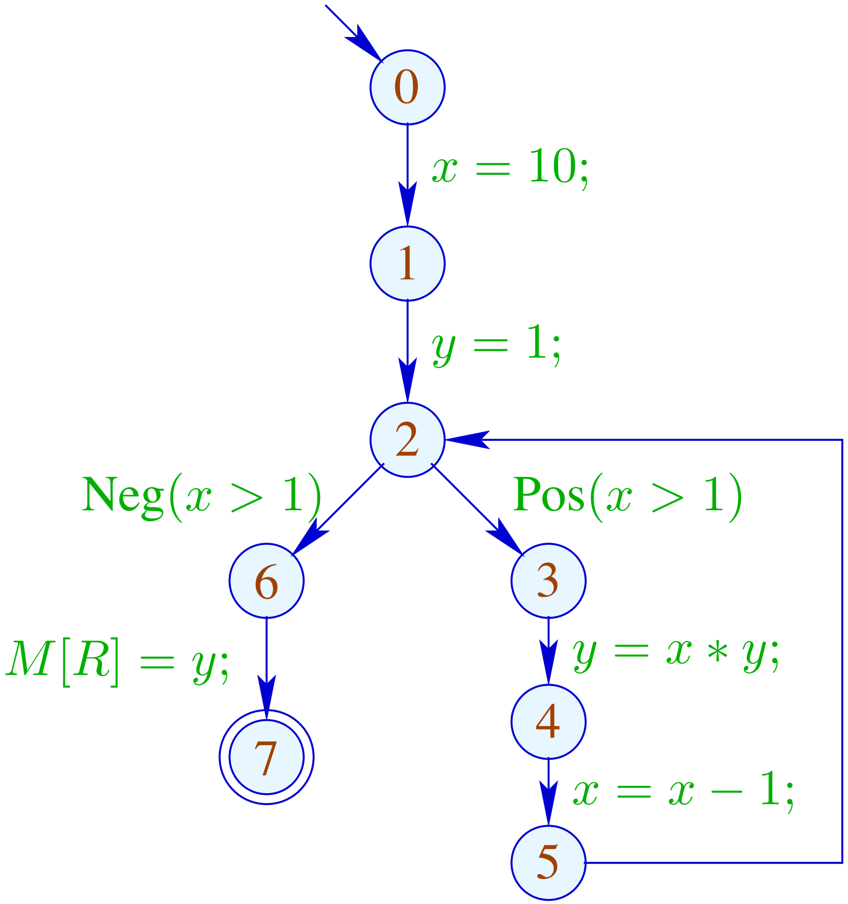


Example:



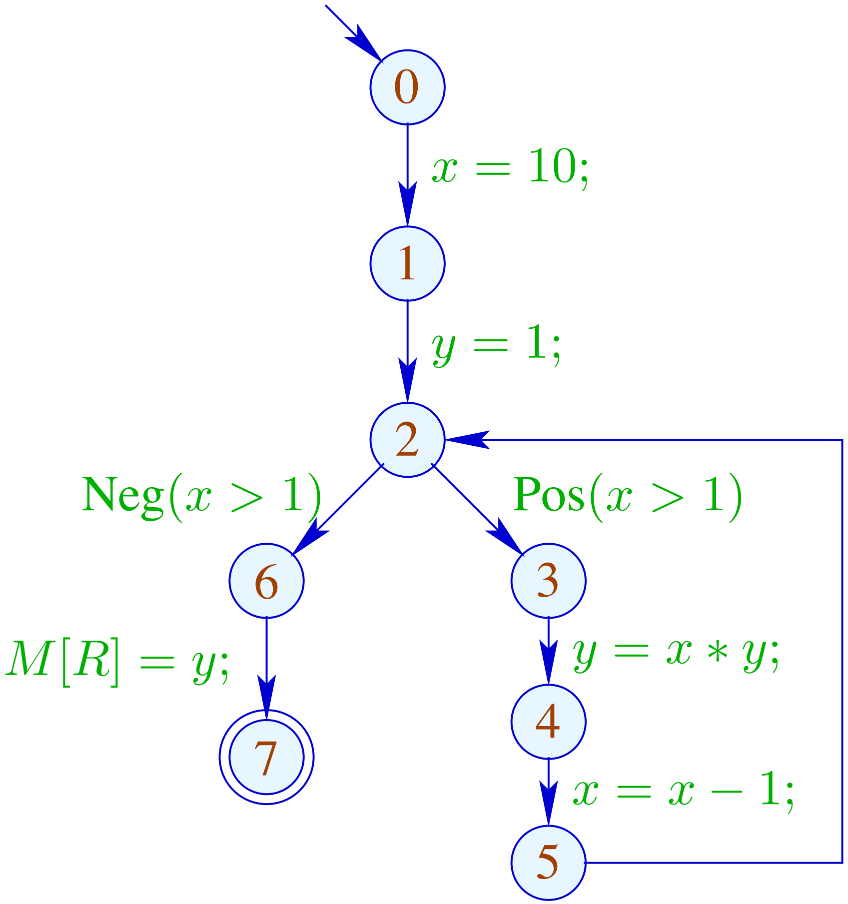
	1	
	x	y
0	$\top$	$\top$
1	10	$\top$
2	10	1
3	10	1
4	10	10
5	9	10
6	$\bot$	
7	$\bot$	

Example:



	1		2	
	x	y	x	y
0	$\top$	$\top$	$\top$	$\top$
1	10	$\top$	10	$\top$
2	10	1	$\top$	$\top$
3	10	1	$\top$	$\top$
4	10	10	$\top$	$\top$
5	9	10	$\top$	$\top$
6	$\perp$		$\top$	$\top$
7	$\perp$		$\top$	$\top$

Example:



	1		2		3	
	x	y	x	y	x	y
0	$\top$	$\top$	$\top$	$\top$		
1	10	$\top$	10	$\top$		
2	10	1	$\top$	$\top$		
3	10	1	$\top$	$\top$		
4	10	10	$\top$	$\top$	dito	
5	9	10	$\top$	$\top$		
6	$\perp$		$\top$	$\top$		
7	$\perp$		$\top$	$\top$		

Concrete vs. Abstract Execution:

Although program and all initial values are given, abstract execution does not compute the result!

On the other hand, fixed-point iteration is guaranteed to terminate:

For n program points and m variables, we maximally need:
$n \cdot (m + 1)$ rounds

Observation: The effects of edges are not distributive!

Counterexample: $f = [\![x = x + y;]\!]^\sharp$

$$\begin{array}{lrcl}
\text{Let} & D_1 & = & \{x \mapsto 2, y \mapsto 3\} \\
 & D_2 & = & \{x \mapsto 3, y \mapsto 2\} \\
\text{Then} & f\,D_1 \sqcup f\,D_2 & = & \{x \mapsto 5, y \mapsto 3\} \sqcup \{x \mapsto 5, y \mapsto 2\} \\
 & & = & \{x \mapsto 5, y \mapsto \top\} \\
 & & \neq & \{x \mapsto \top, y \mapsto \top\} \\
 & & = & f\,\{x \mapsto \top, y \mapsto \top\} \\
 & & = & f\,(D_1 \sqcup D_2)
\end{array}$$

We conclude:

The least solution $\mathcal{D}$ of the constraint system in general yields only an upper approximation of the MOP, i.e.,

$$\mathcal{D}^*[v] \quad \sqsubseteq \quad \mathcal{D}[v]$$

We conclude:

The least solution $\mathcal{D}$ of the constraint system in general yields only an upper approximation of the MOP, i.e.,

$$\mathcal{D}^*[v] \quad \sqsubseteq \quad \mathcal{D}[v]$$

As an upper approximation, $\mathcal{D}[v]$ nonetheless describes the result of every program execution π that reaches v :

$$(\llbracket\pi\rrbracket\,(\rho, \mu)) \;\; \Delta \;\; (\mathcal{D}[v])$$

whenever $\llbracket\pi\rrbracket\,(\rho, \mu)$ is defined

6 Interval Analysis

Constant propagation attempts to determine values of variables.

However, variables may take on several values during program execution.

So, *the value* of a variable will often be unknown.

Next attempt: determine an interval enclosing all possible values that a variable may take on during program execution at a program point.

Example:

$$
\begin{array}{l}
\textsf{for}\ (i = 0; i < 42; i{+}{+}) \\
\qquad \textsf{if}\ (0 \leq i \wedge i < 42)\{ \\
\qquad\qquad A_1 = A + i; \\
\qquad\qquad M[A_1] = i; \\
\qquad \}
\end{array}
$$

// A start address of an array

// if-statement does array-bound check

Obviously, the inner check is superfluous.

Idea 1:

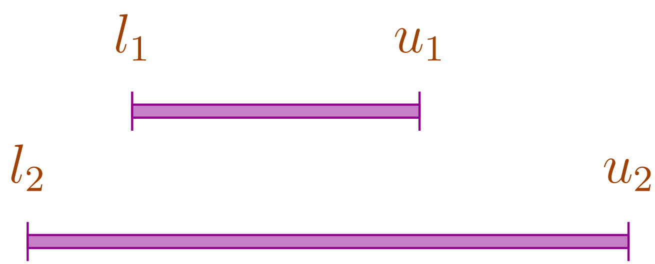
Determine for every variable x the tightest possible interval of potential values.

Abstract domain:

$$\mathbb{I} = \{[l, u] \mid l \in \mathbb{Z} \cup \{-\infty\}, u \in \mathbb{Z} \cup \{+\infty\}, l \leq u\}$$

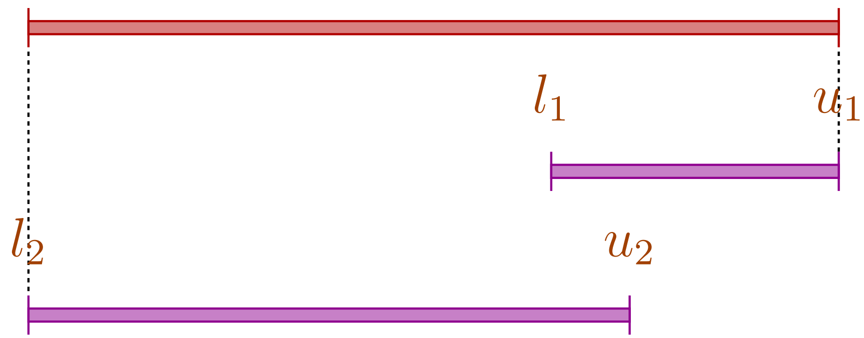
Partial order:

$$[l_1, u_1] \sqsubseteq [l_2, u_2] \qquad \text{iff} \qquad l_2 \leq l_1 \wedge u_1 \leq u_2$$



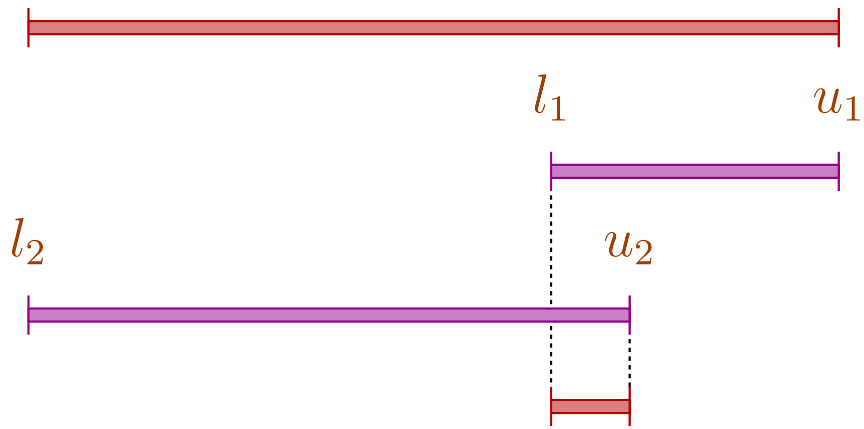
Thus:

$$[l_1, u_1] \sqcup [l_2, u_2] \quad = \quad [l_1 \sqcap l_2, u_1 \sqcup u_2]$$



Thus:

$$
\begin{array}{rcll}
[l_1, u_1] \sqcup [l_2, u_2] & = & [l_1 \sqcap l_2, u_1 \sqcup u_2] & \\
[l_1, u_1] \sqcap [l_2, u_2] & = & [l_1 \sqcup l_2, u_1 \sqcap u_2] & \quad \text{whenever } (l_1 \sqcup l_2) \leq (u_1 \sqcap u_2)
\end{array}
$$



Caveat:

$\rightarrow$ $\mathbb{I}$ is not a complete lattice,

$\rightarrow$ $\mathbb{I}$ has infinite ascending chains, e.g.,

$$[0,0] \sqsubset [0,1] \sqsubset [-1,1] \sqsubset [-1,2] \sqsubset \ldots$$

Caveat:

- $\rightarrow$ $\mathbb{I}$ is not a complete lattice,
- $\rightarrow$ $\mathbb{I}$ has infinite ascending chains, e.g.,

$$[0,0] \sqsubset [0,1] \sqsubset [-1,1] \sqsubset [-1,2] \sqsubset \ldots$$

Description Relation:

$$z \;\Delta\; [l,u] \qquad \text{iff} \qquad l \leq z \leq u$$

Concretization:

$$\gamma\,[l,u] = \{z \in \mathbb{Z} \mid l \leq z \leq u\}$$

Example:

$$\begin{aligned} \gamma\,[0,7] &= \{0,\ldots,7\} \\ \gamma\,[0,\infty] &= \{0,1,2,\ldots,\} \end{aligned}$$

Computing with intervals: Interval Arithmetic.

Addition:

$$\begin{aligned} [l_1,u_1] \;+^\sharp\; [l_2,u_2] &= [l_1+l_2, u_1+u_2] \qquad \text{where} \\ -\infty + \_ &= -\infty \\ +\infty + \_ &= +\infty \\ // \quad & -\infty + \infty \quad \text{cannot occur} \end{aligned}$$

Negation:

$$-^\sharp [l,u] \quad = \quad [-u,-l]$$

Multiplication:

$$\begin{array}{rcl} [l_1,u_1] \;*^\sharp\; [l_2,u_2] & = & [a,b] \qquad \text{where} \\ a & = & l_1 l_2 \sqcap l_1 u_2 \sqcap u_1 l_2 \sqcap u_1 u_2 \\ b & = & l_1 l_2 \sqcup l_1 u_2 \sqcup u_1 l_2 \sqcup u_1 u_2 \end{array}$$

Example:

$$\begin{array}{rcl} [0,2] \;*^\sharp\; [3,4] & = & [0,8] \\ [-1,2] \;*^\sharp\; [3,4] & = & [-4,8] \\ [-1,2] \;*^\sharp\; [-3,4] & = & [-6,8] \\ [-1,2] \;*^\sharp\; [-4,-3] & = & [-8,4] \end{array}$$

Division: $[l_1, u_1] \,/^\sharp\, [l_2, u_2] \;=\; [a, b]$

- If 0 is not contained in the interval of the denominator, then:

$$\begin{array}{lll} a & = & l_1/l_2 \sqcap l_1/u_2 \sqcap u_1/l_2 \sqcap u_1/u_2 \\ b & = & l_1/l_2 \sqcup l_1/u_2 \sqcup u_1/l_2 \sqcup u_1/u_2 \end{array}$$

- If: $l_2 \leq 0 \leq u_2$, we define:

$$[a, b] \;=\; [-\infty, +\infty]$$

Equality:

$$[l_1, u_1] ==^\sharp [l_2, u_2] \ = \ \begin{cases} \mathit{true} & \text{if} \quad l_1 = u_1 = l_2 = u_2 \\ \mathit{false} & \text{if} \quad u_1 < l_2 \lor u_2 < l_1 \\ \top & \text{otherwise} \end{cases}$$

Equality:

$$[l_1, u_1] ==^\sharp [l_2, u_2] \;=\; \begin{cases} \mathit{true} & \text{if} \quad l_1 = u_1 = l_2 = u_2 \\ \mathit{false} & \text{if} \quad u_1 < l_2 \vee u_2 < l_1 \\ \top & \text{otherwise} \end{cases}$$

Example:

$$\begin{aligned} [42, 42] ==^\sharp [42, 42] \quad &= \quad \mathit{true} \\ [0, 7] ==^\sharp [0, 7] \quad &= \quad \top \\ [1, 2] ==^\sharp [3, 4] \quad &= \quad \mathit{false} \end{aligned}$$

Less:

$$[l_1, u_1] \ <^\sharp \ [l_2, u_2] \ = \ \begin{cases} true & \text{if} \quad u_1 < l_2 \\ false & \text{if} \quad u_2 \leq l_1 \\ \top & \text{otherwise} \end{cases}$$

Less:

$$[l_1, u_1] \;<^\sharp\; [l_2, u_2] \;=\; \begin{cases} true & \text{if} \quad u_1 < l_2 \\ false & \text{if} \quad u_2 \leq l_1 \\ \top & \text{otherwise} \end{cases}$$

Example:

$$\begin{array}{rcl} [1,2] <^\sharp [9,42] & = & true \\ [0,7] \;<^\sharp\; [0,7] & = & \top \\ [3,4] \;<^\sharp\; [1,2] & = & false \end{array}$$

By means of $\mathbb{I}$ we construct the complete lattice:

$$\mathbb{D}_{\mathbb{I}} = (\mathit{Vars} \to \mathbb{I})_{\perp}$$

Description Relation:

$$\rho \ \Delta \ D \qquad \text{iff} \qquad D \neq \perp \quad \wedge \quad \forall\, x \in \mathit{Vars} : \ (\rho\, x) \ \Delta \ (D\, x)$$

The abstract evaluation of expressions is defined analogously to constant propagation. We have:

$$([\![e]\!]\ \rho) \ \Delta \ ([\![e]\!]^{\sharp}\ D) \qquad \text{whenever} \qquad \rho \ \Delta \ D$$

The Effects of Edges:

$$
\begin{array}{lcl}
[\![;]\!]^\sharp\, D & = & D \\
[\![x = e;]\!]^\sharp\, D & = & D \oplus \{x \mapsto [\![e]\!]^\sharp\, D\} \\
[\![x = M[e];]\!]^\sharp\, D & = & D \oplus \{x \mapsto \top\} \\
[\![M[e_1] = e_2;]\!]^\sharp\, D & = & D \\
[\![\mathrm{Pos}\,(e)]\!]^\sharp\, D & = & \begin{cases} \bot & \text{if} \quad \mathit{false} = [\![e]\!]^\sharp\, D \\ D & \text{otherwise} \end{cases} \\
[\![\mathrm{Neg}\,(e)]\!]^\sharp\, D & = & \begin{cases} D & \text{if} \quad \mathit{false} \sqsubseteq [\![e]\!]^\sharp\, D \\ \bot & \text{otherwise} \end{cases}
\end{array}
$$

... given that $\qquad D \neq \bot$

Better Exploitation of Conditions:

$$[\![\text{Pos}\,(e)]\!]^\sharp\, D \;=\; \begin{cases} \bot & \text{if} \quad \mathit{false} = [\![e]\!]^\sharp\, D \\ D_1 & \text{otherwise} \end{cases}$$

where :

$$D_1 \;=\; \begin{cases} D \oplus \{x \mapsto (D\,x) \sqcap ([\![e_1]\!]^\sharp\, D)\} & \text{if } e \equiv x == e_1 \\ D \oplus \{x \mapsto (D\,x) \sqcap [-\infty, u]\} & \text{if } e \equiv x \leq e_1, [\![e_1]\!]^\sharp\, D = [\_, u] \\ D \oplus \{x \mapsto (D\,x) \sqcap [l, \infty]\} & \text{if } e \equiv x \geq e_1, [\![e_1]\!]^\sharp\, D = [l, \_] \end{cases}$$

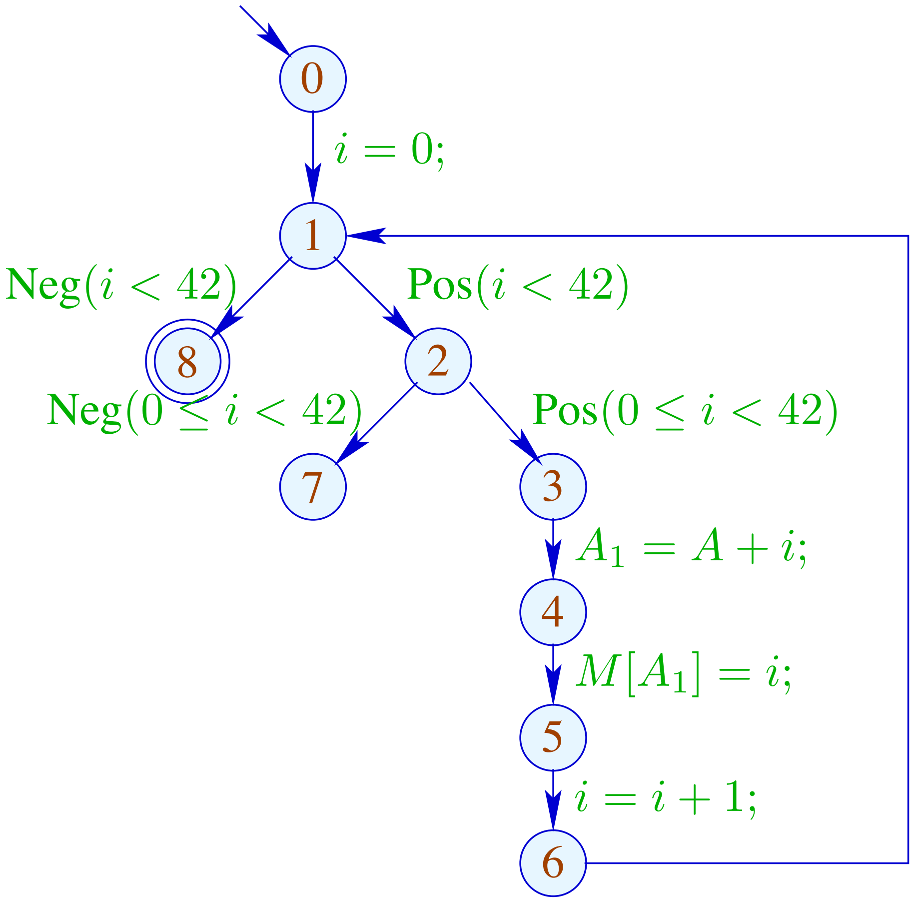
Better Exploitation of Conditions (cont.):

$$[\![\mathrm{Neg}\,(e)]\!]^\sharp\, D \quad = \quad \begin{cases} \bot & \text{if} \quad \mathit{false} \not\sqsubseteq [\![e]\!]^\sharp\, D \\ D_1 & \text{otherwise} \end{cases}$$

where :

$$D_1 \quad = \quad \begin{cases} D \oplus \{x \mapsto (D\, x) \sqcap ([\![e_1]\!]^\sharp\, D)\} & \text{if } e \equiv x \neq e_1 \\ D \oplus \{x \mapsto (D\, x) \sqcap [-\infty, u]\} & \text{if } e \equiv x > e_1,\, [\![e_1]\!]^\sharp\, D = [\_, u] \\ D \oplus \{x \mapsto (D\, x) \sqcap [l, \infty]\} & \text{if } e \equiv x < e_1,\, [\![e_1]\!]^\sharp\, D = [l, \_] \end{cases}$$

Example:



	i	
	l	u
0	$-\infty$	$+\infty$
1	0	42
2	0	41
3	0	41
4	0	41
5	0	41
6	1	42
7	$\bot$	
8	42	42

Problem:

- $\to$ The solution can be computed with RR-iteration — after about 42 rounds.
- $\to$ On some programs, iteration may never terminate.

Idea: Widening

Accelerate the iteration — at the cost of precision

Formalization of the Approach:

Let $\quad x_i \sqsupseteq f_i(x_1,\ldots,x_n)\,, \quad i = 1,\ldots,n \qquad (1)$

denote a system of constraints over $\quad \mathbb{D}$

Define an accumulating iteration:

$$x_i = x_i \sqcup f_i(x_1,\ldots,x_n)\,, \quad i = 1,\ldots,n \qquad (2)$$

We obviously have:

(a) $\underline{x}$ is a solution of (1) iff $\underline{x}$ is a solution of (2).

(b) The function $G : \mathbb{D}^n \to \mathbb{D}^n$ with
$G(x_1,\ldots,x_n) = (y_1,\ldots,y_n)\,, \quad y_i = x_i \sqcup f_i(x_1,\ldots,x_n)$
is increasing, i.e., $\quad \underline{x} \sqsubseteq G\,\underline{x}$ for all $\underline{x} \in \mathbb{D}^n$.

(c) The sequence $G^k \underline{\perp}$, $k \geq 0$, is an ascending chain:

$$\underline{\perp} \sqsubseteq G\,\underline{\perp} \sqsubseteq \ldots \sqsubseteq G^k \underline{\perp} \sqsubseteq \ldots$$

(d) If $G^k \underline{\perp} = G^{k+1} \underline{\perp} = \underline{y}$, then $\underline{y}$ is a solution of (1).

(e) If $\mathbb{D}$ has infinite strictly ascending chains, then (d) is not yet sufficient ...

but: we could consider the modified system of equations:

$$x_i = x_i \sqcup f_i(x_1, \ldots, x_n)\,, \quad i = 1, \ldots, n \qquad (3)$$

for a binary operation widening:

$$\sqcup : \mathbb{D}^2 \to \mathbb{D} \qquad \text{with} \qquad v_1 \sqcup v_2 \sqsubseteq v_1 \sqcup v_2$$

(RR)-iteration for (3) still will compute a solution of (1)

... for Interval Analysis:

- The complete lattice is: $\mathbb{D}_{\mathbb{I}} \;=\; (\mathit{Vars} \to \mathbb{I})_\bot$
- the widening $\sqcup\!\!\!\!-$ is defined by:

$$\bot \sqcup\!\!\!\!- D \;=\; D \sqcup\!\!\!\!- \bot \;=\; D \qquad \text{and for} \quad D_1 \neq \bot \neq D_2:$$

$$\begin{array}{rcl} (D_1 \sqcup\!\!\!\!- D_2)\, x & = & (D_1\, x) \sqcup\!\!\!\!- (D_2\, x) \qquad \text{where} \\ [l_1, u_1] \sqcup\!\!\!\!- [l_2, u_2] & = & [l, u] \qquad \text{with} \\ l & = & \begin{cases} l_1 & \text{if} \quad l_1 \leq l_2 \\ -\infty & \text{otherwise} \end{cases} \\ u & = & \begin{cases} u_1 & \text{if} \quad u_1 \geq u_2 \\ +\infty & \text{otherwise} \end{cases} \end{array}$$

$\Longrightarrow$ $\sqcup\!\!\!\!-$ is not commutative !!!

Example:

$$\begin{array}{rcl} [0,2] \sqcup [1,2] & = & [0,2] \\ [1,2] \sqcup [0,2] & = & [-\infty,2] \\ [1,5] \sqcup [3,7] & = & [1,+\infty] \end{array}$$

→ Widening returns larger values more quickly.

→ It should be constructed in such a way that termination of iteration is guaranteed.

→ For interval analysis, widening bounds the number of iterations by:

$$\#points \cdot (1 + 2 \cdot \#Vars)$$

Conclusion:

- In order to determine a solution of (1) over a complete lattice with infinite ascending chains, we define a suitable widening and then solve (3)

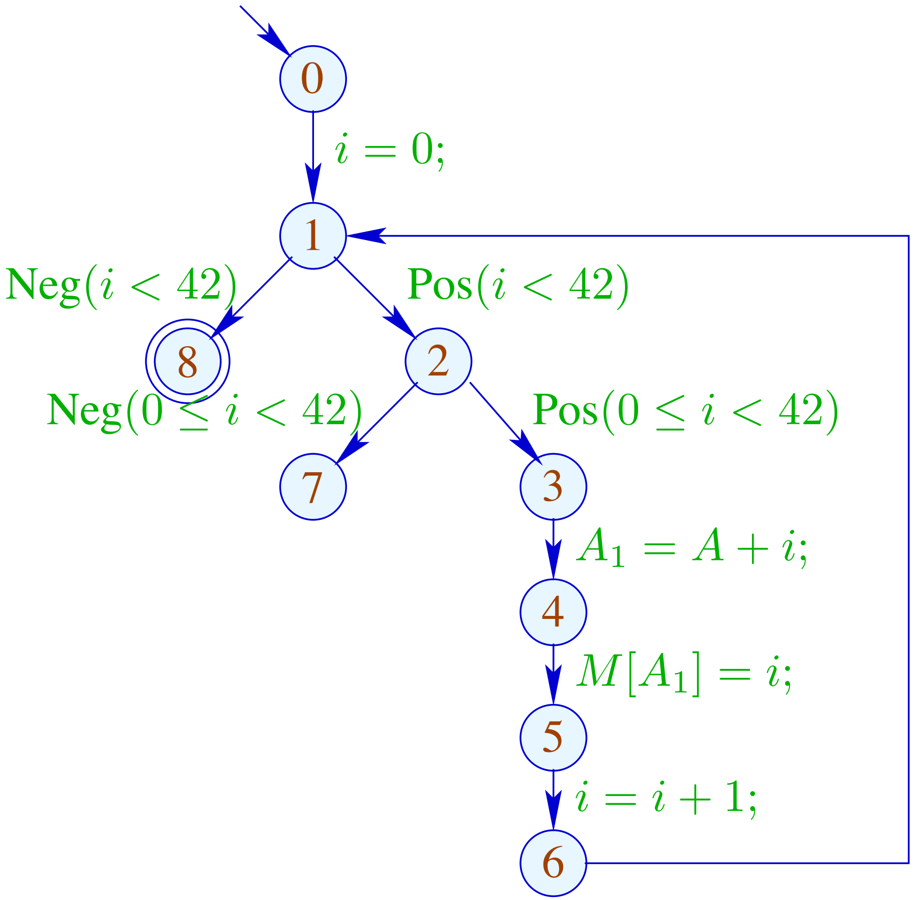
- Caveat: The construction of suitable widenings is a dark art !!!

  Often $\sqcup$ is chosen dynamically during iteration such that

  $\rightarrow$ the abstract values do not get too complicated;

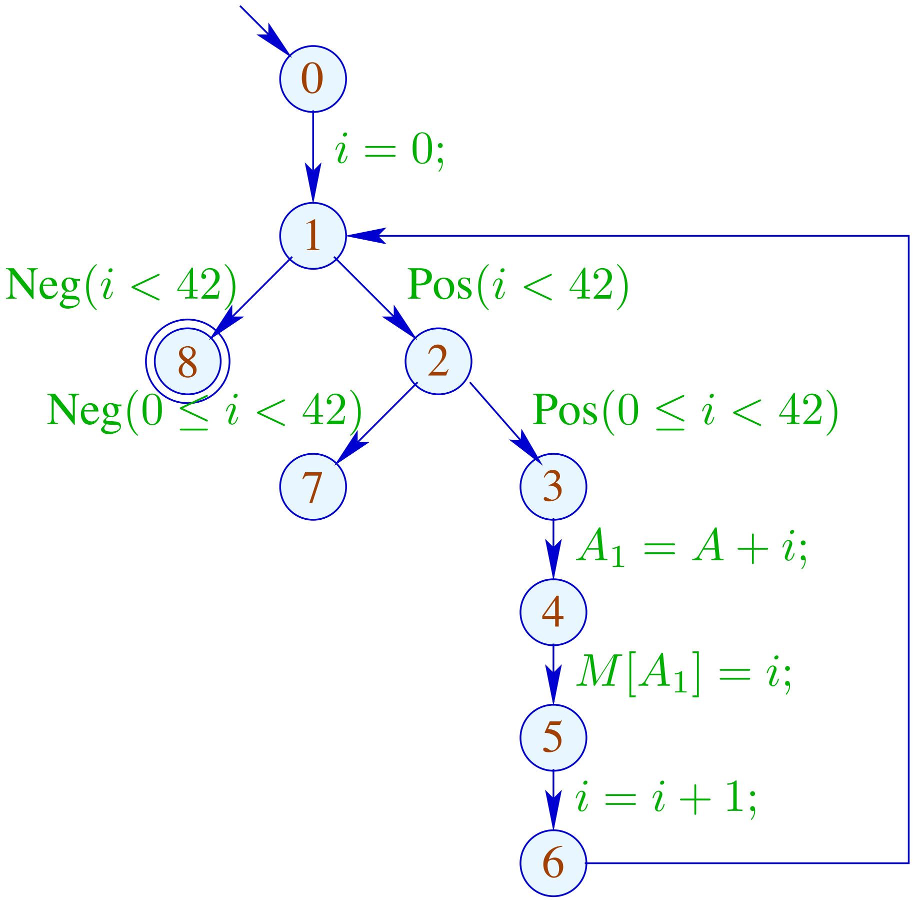
  $\rightarrow$ the number of updates remains bounded ...

Our Example:



	1	
	l	u
0	$-\infty$	$+\infty$
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	1	1
7	$\perp$	
8	$\perp$	

Our Example:



	1		2		3	
	l	u	l	u	l	u
0	$-\infty$	$+\infty$	$-\infty$	$+\infty$		
1	0	0	0	$+\infty$		
2	0	0	0	$+\infty$		
3	0	0	0	$+\infty$		
4	0	0	0	$+\infty$	dito	
5	0	0	0	$+\infty$		
6	1	1	1	$+\infty$		
7	$\perp$		42	$+\infty$		
8	$\perp$		42	$+\infty$		