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
   - $\rightarrow$ available expressions
   - $\rightarrow$ live variables
   - $\rightarrow$ 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!

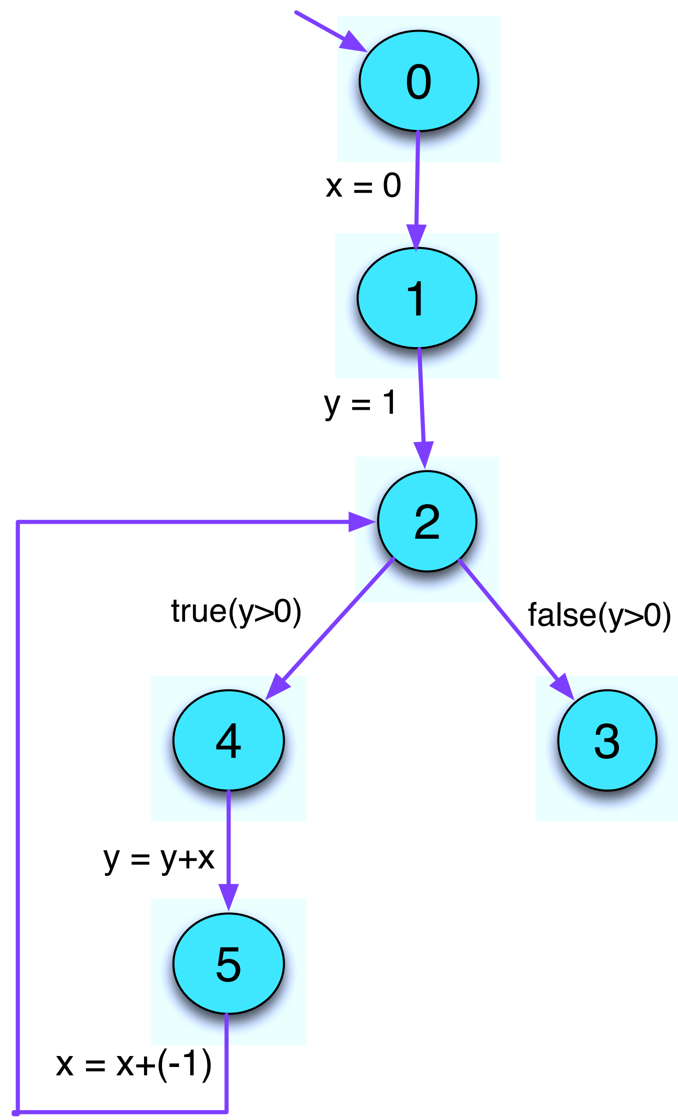
2 Example — Rules-of-Sign Analysis

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

Example program:

```
1: x = 0;
2: y = 1;
3: while (y > 0) do
4:     y = y + x;
5:     x = x + (-1);
```

Program representation as *control-flow graphs*



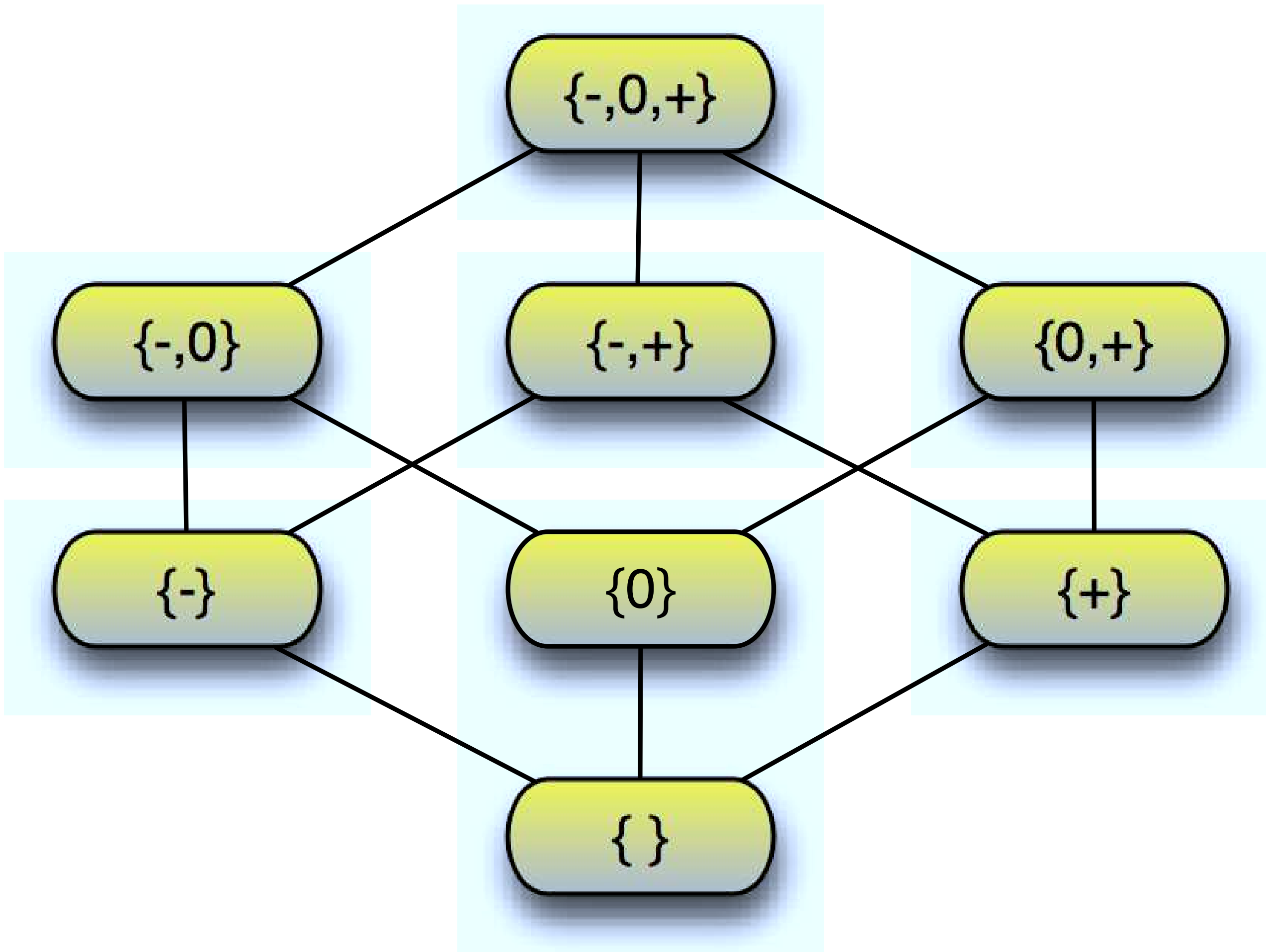
What are the ingredients that we need?

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 $\mathbb{D} = (\,Vars \to Signs)_\perp$, 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

$D_1 = \perp \qquad$ or

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

Intuition?

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

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

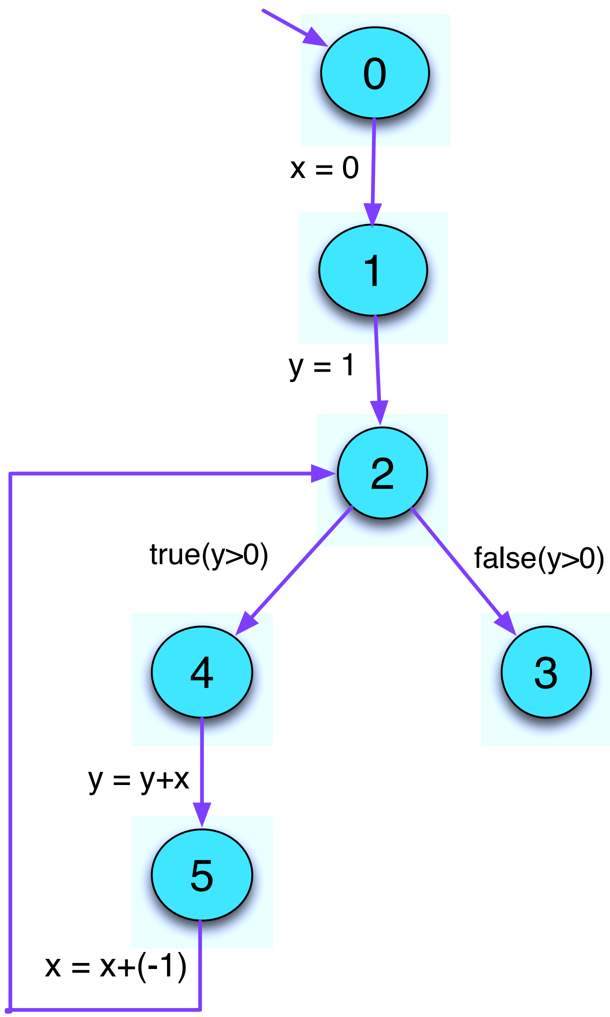
$D_1 = \perp \qquad$ or

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

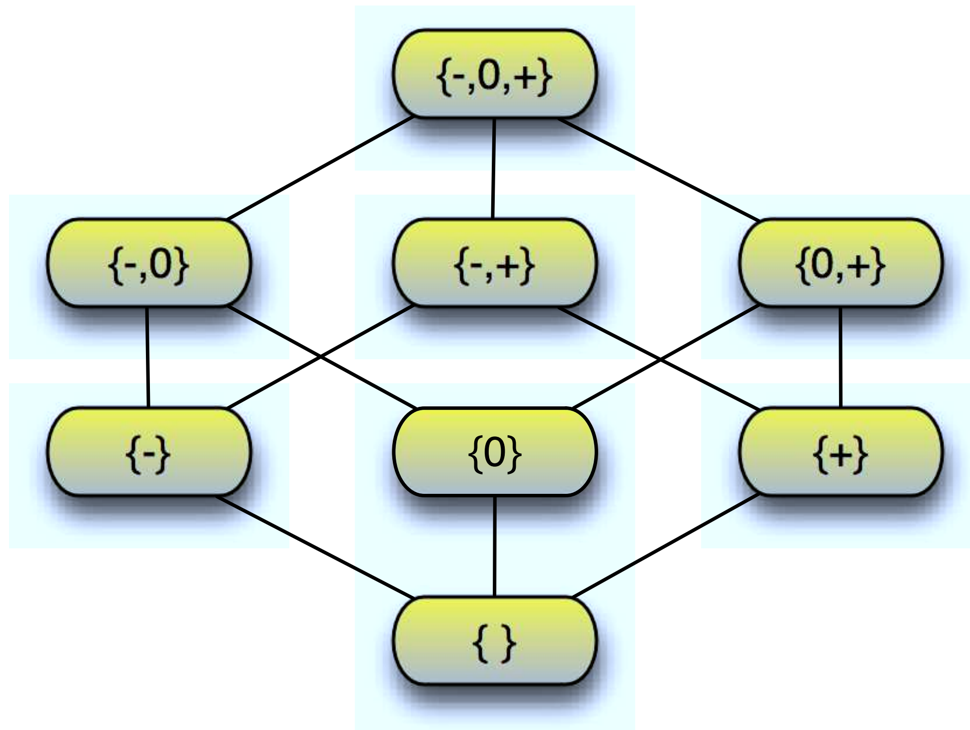
Intuition?

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?

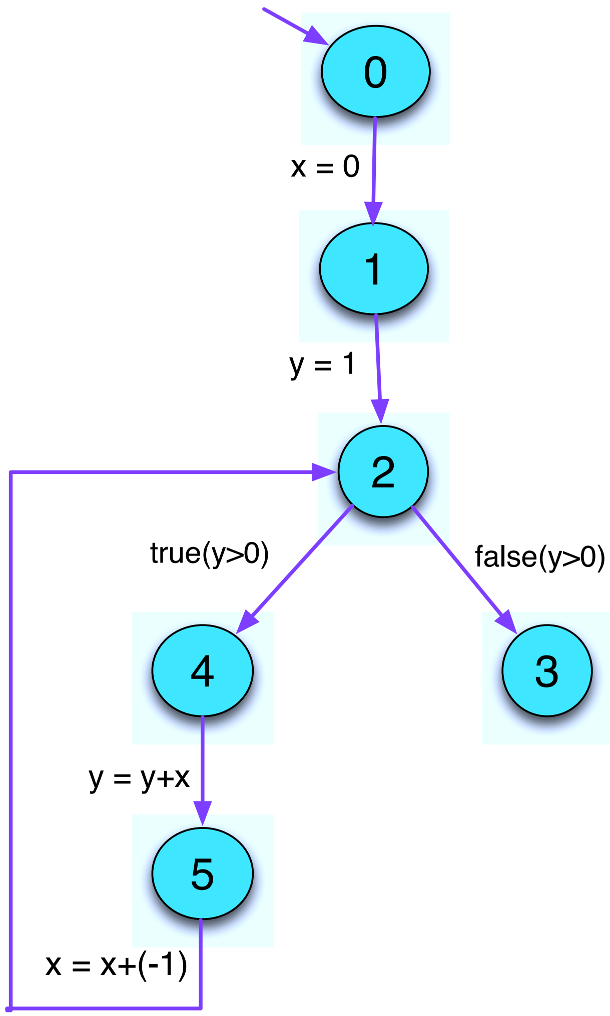


In particular, how did we walk the lattice for y at program point 5?



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 $\square$ working on values by abstract operators $\square^\sharp$ 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:

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

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*:

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

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

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

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

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

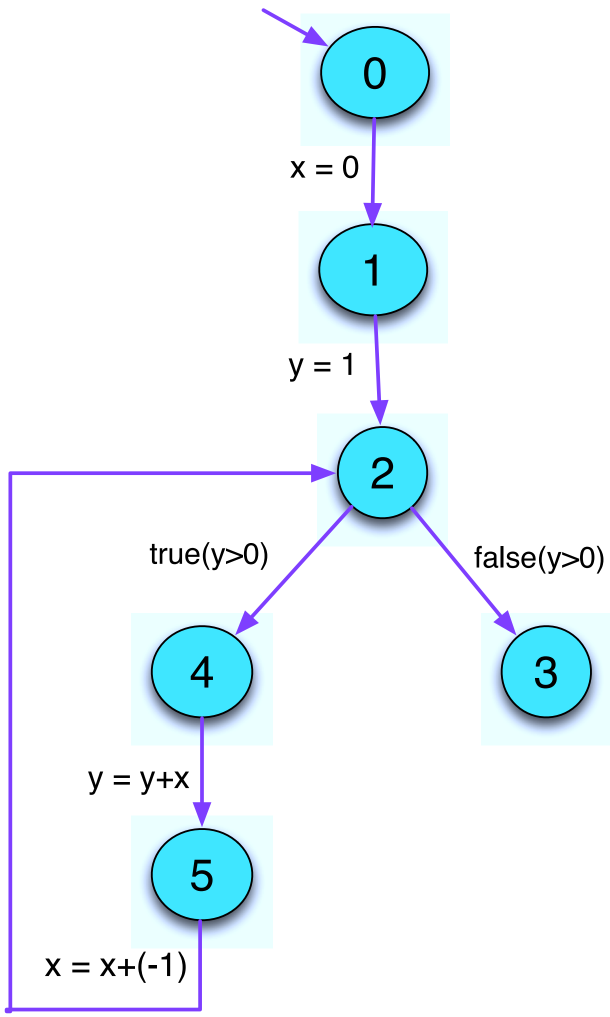
Collect all constraints on the values of $\mathcal{S}[u]$ into a system of constraints:

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

Wanted:

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

Example:



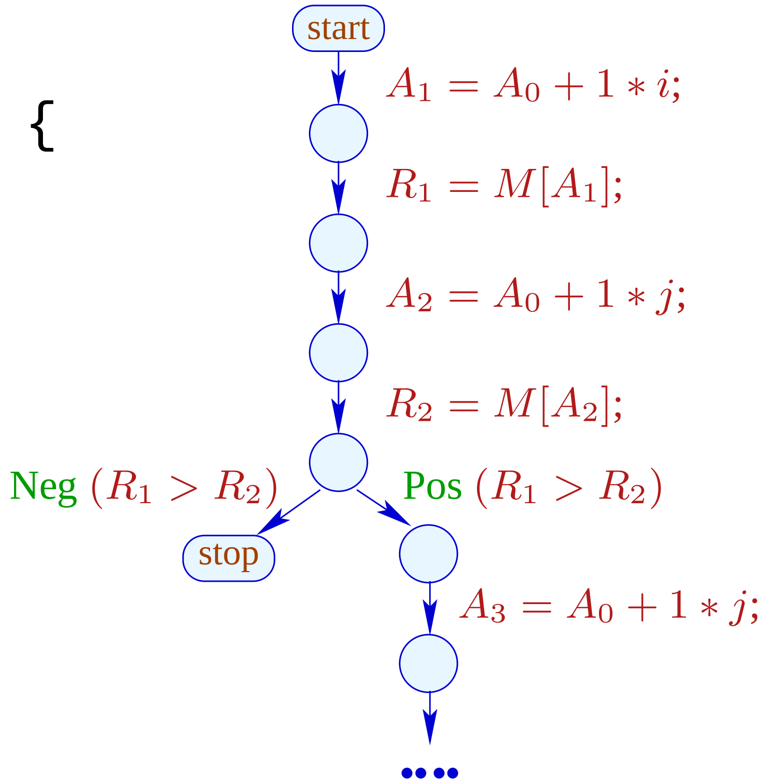
$$
\begin{aligned}
\mathcal{S}[0] &\supseteq \top \\
\mathcal{S}[1] &\supseteq \mathcal{S}[0] \oplus \{x \mapsto \{0\}\} \\
\mathcal{S}[2] &\supseteq \mathcal{S}[1] \oplus \{y \mapsto \{+\}\} \\
\mathcal{S}[2] &\supseteq \mathcal{S}[5] \oplus \{x \mapsto [\![x + (-1)]\!]^\sharp\, \mathcal{S}[5]\} \\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

Background   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 $k = (u, lab, v)$ defines a partial transformation

$$[\![k]\!] = [\![lab]\!]$$

of the state:

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

$$
\begin{array}{lcll}
[\![;]\!]\,(\rho,\mu) & = & (\rho,\mu) & \\[2ex]
[\![\mathrm{Pos}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho \neq 0 \\
[\![\mathrm{Neg}\,(e)]\!]\,(\rho,\mu) & = & (\rho,\mu) & \text{if } [\![e]\!]\,\rho = 0
\end{array}
$$

// $[\![e]\!]$: 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

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

$$\llbracket M[e_1] = e_2; \rrbracket\,(\rho,\mu) \quad = \quad (\rho, \boxed{\mu \oplus \{\llbracket e_1 \rrbracket\,\rho \mapsto \llbracket e_2 \rrbracket\,\rho\}})$$

Example:

$\llbracket x = x + 1; \rrbracket\,(\{x \mapsto 5\}, \mu) = (\rho, \mu)$ where

$$\begin{aligned} \rho &= \{x \mapsto 5\} \oplus \{x \mapsto \llbracket x+1 \rrbracket\,\{x \mapsto 5\}\} \\ &= \{x \mapsto 5\} \oplus \{x \mapsto 6\} \\ &= \{x \mapsto 6\} \end{aligned}$$

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

$$s \in \mathit{def}\,(\llbracket k_m \rrbracket \circ \ldots \circ \llbracket k_1 \rrbracket)$$

The result of the computation is $\llbracket \pi \rrbracket\, s = (\llbracket k_m \rrbracket \circ \ldots \circ \llbracket k_1 \rrbracket)\, s$

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.

Background Complete Lattices

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

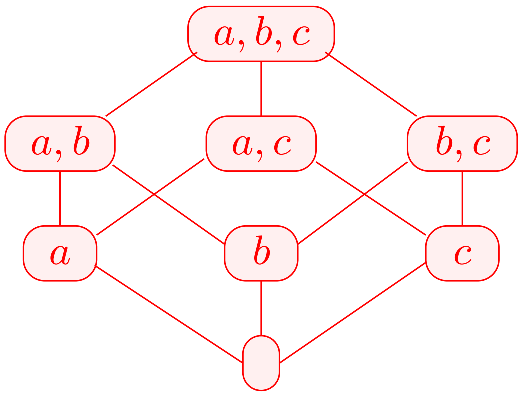
$$
\begin{array}{ll}
a \sqsubseteq a & \textit{reflexivity} \\
a \sqsubseteq b \wedge b \sqsubseteq a \implies a = b & \textit{anti-symmetry} \\
a \sqsubseteq b \wedge b \sqsubseteq c \implies a \sqsubseteq c & \textit{transitivity}
\end{array}
$$

Intuition: $\sqsubseteq$ represents precision.

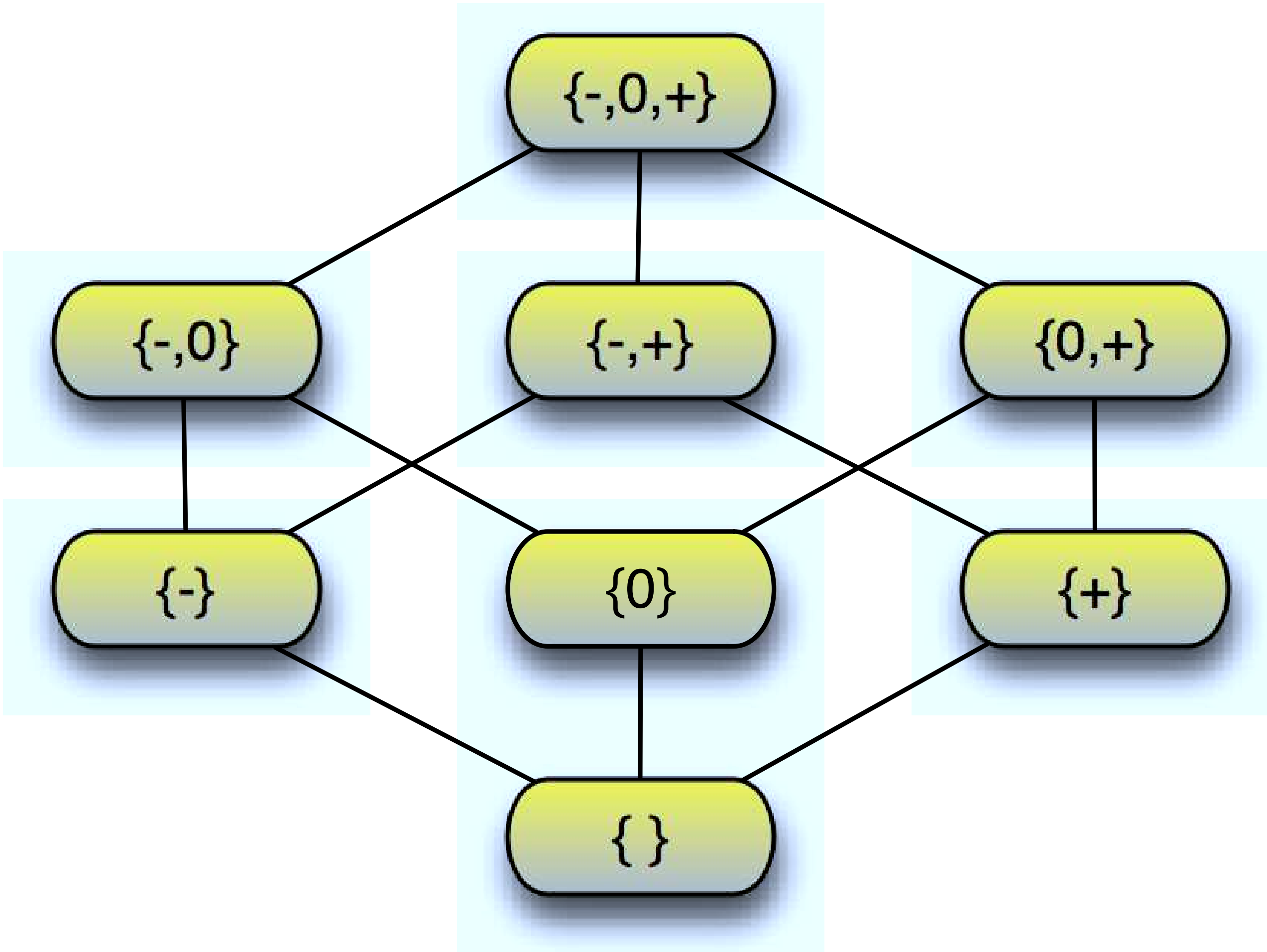
By convention: $a \sqsubseteq b$ means a is more precise than b.

Examples:

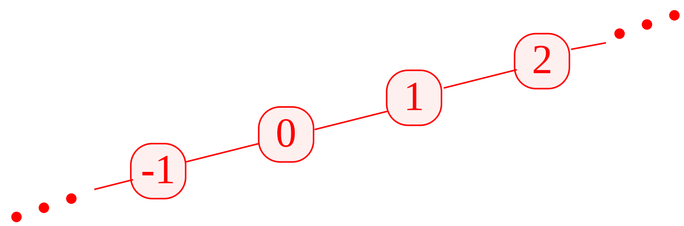
1. $\mathbb{D} = 2^{\{a,b,c\}}$ with the relation "$\subseteq$" :



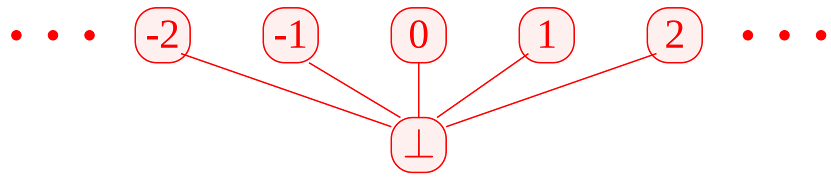
2. The rules-of-sign analysis uses the following lattice $\mathbb{D} = 2^{\{-,0,+\}}$ with the relation "$\subseteq$" :



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:

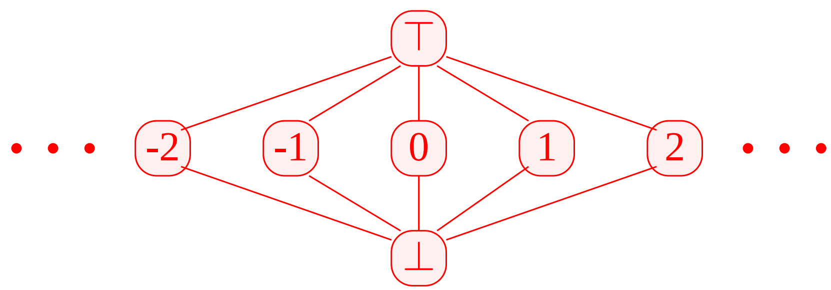
Every complete lattice has

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

$\rightarrow$ a greatest element $\top = \bigsqcup \mathbb{D} \in \mathbb{D}$.

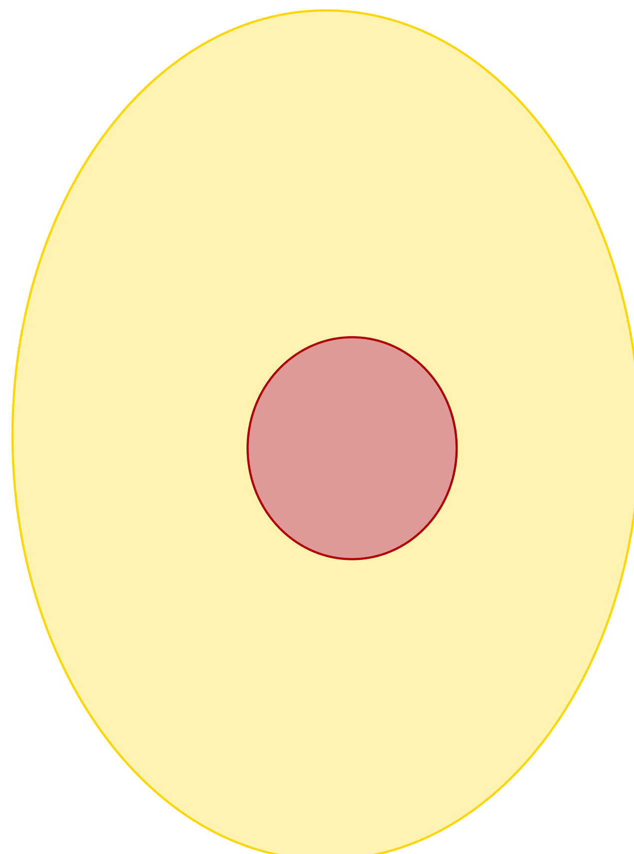
Examples:

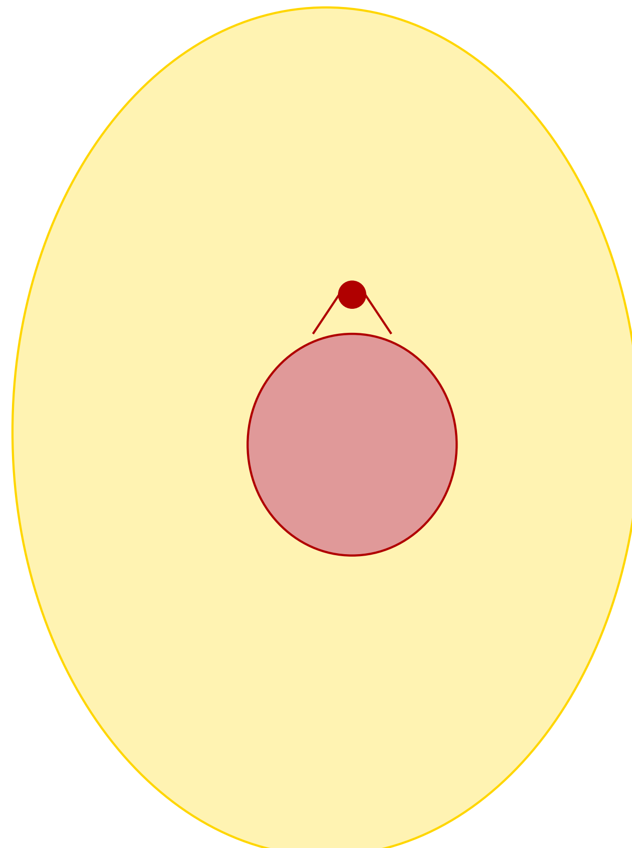
1. $\mathbb{D} = 2^{\{a,b,c\}}$ is a complete lattice
2. $\mathbb{D} = \mathbb{Z}$ with "$\leq$" is not a complete lattice.
3. $\mathbb{D} = \mathbb{Z}_\perp$ is also not a complete lattice
4. With an extra element $\top$, we obtain the flat lattice
   $\mathbb{Z}_\perp^\top = \mathbb{Z} \cup \{\perp,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

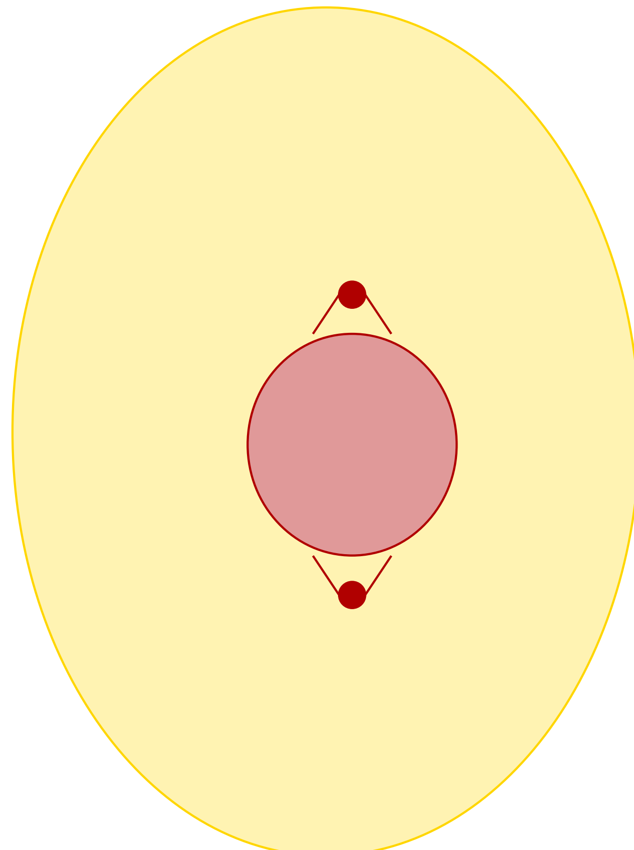


Theorem:

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!

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

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:	$Signs$
$\sqsubseteq \;\subseteq\; \mathbb{D} \times \mathbb{D}$	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

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

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

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

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

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

The Iteration:

	0	1	2	3	4
x_1	$\emptyset$	$\{a\}$	$\{a, c\}$	$\{a, c\}$	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: $\mathbb{D} = 2^U, \quad f\,x = x \cap a \cup b$

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

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

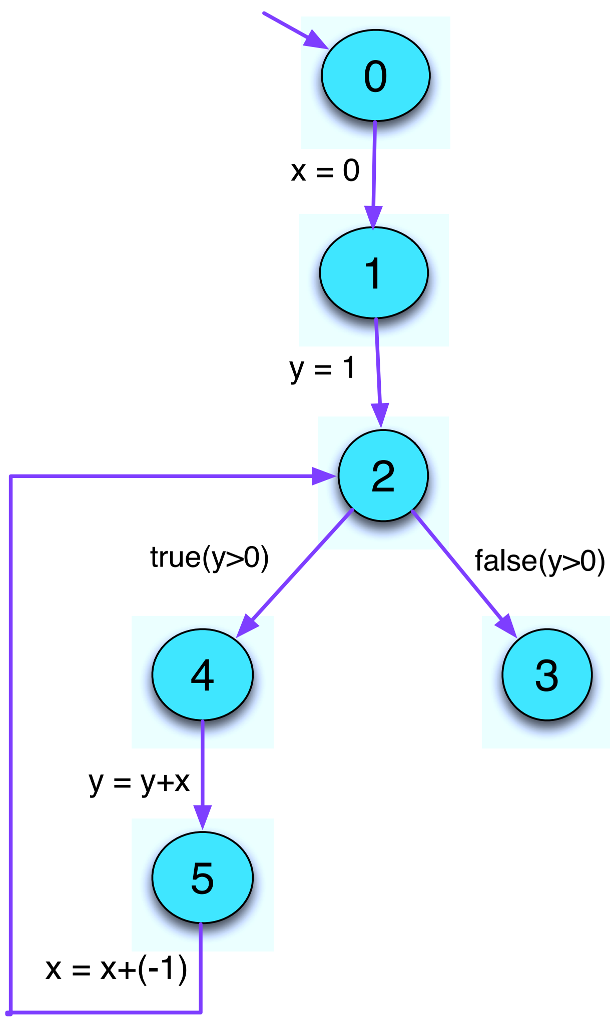
f	$f^k\,\bot$	$f^k\,\top$
0	$\emptyset$	U
1	b	$a \cup b$
2	b	$a \cup b$

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:

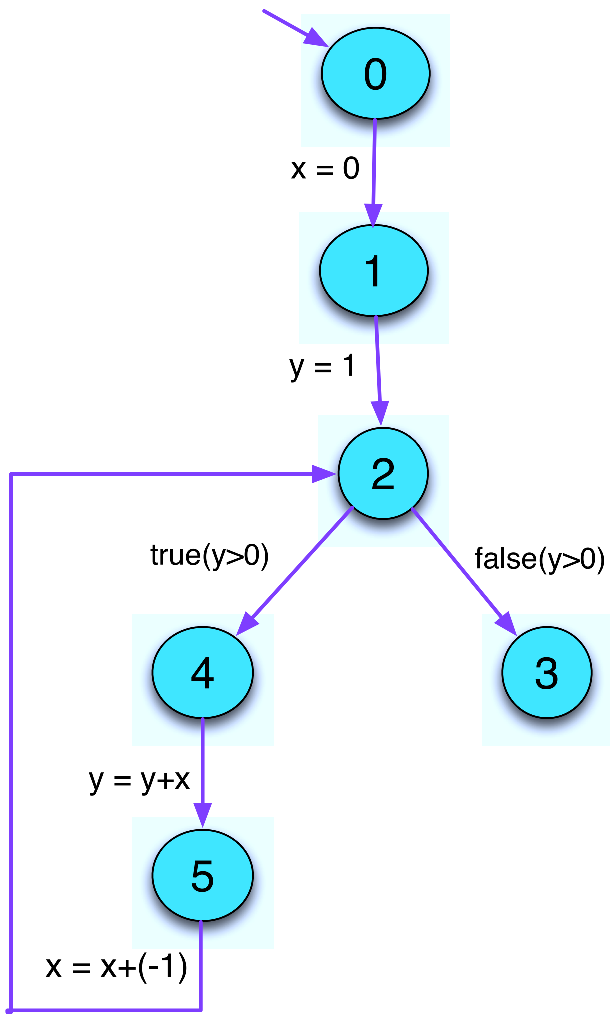


0		1		2		3		4		5	
x	y	x	y	x	y	x	y	x	y	x	y

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

With the notable exception when 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 also monotonic

Assumption: all v are reachable from *start* .
Then:

Theorem Kildall 1972

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

Summary and Application:

→ The abstract edge effects in the analysis of availability of expressions are distributive:

$$\begin{aligned} (a \cup (x_1 \cap x_2)) \backslash b &= ((a \cup x_1) \cap (a \cup x_2)) \backslash b \\ &= ((a \cup x_1) \backslash b) \cap ((a \cup x_2) \backslash b) \end{aligned}$$

→ If all effects of edges are distributive, then the MOP can be computed by means of the constraint system and RR-iteration.

Summary and Application:

→ The abstract edge effects in the analysis of availability of expressions are distributive:

$$\begin{array}{lll} (a \cup (x_1 \cap x_2))\backslash b & = & ((a \cup x_1) \cap (a \cup x_2))\backslash b \\ & = & ((a \cup x_1)\backslash b) \cap ((a \cup x_2)\backslash b) \end{array}$$

→ If all effects of edges are distributive, then the MOP can be computed by means of the constraint system and RR-iteration.

→ If not all effects of edges are distributive, then RR-iteration for the constraint system at least returns a safe upper bound to the MOP