Static program analysis

Thomas Jensen

Ecole Jeunes Chercheurs en Programmation
Rennes, 2012

Plan

Plan

1. Data flow analysis

2. Java byte code verification

Static program analysis

Program analysis

Goal: deduce mechanically properties about the program behaviour without executing it.

Application area: compilers, code optimisation, program verification, debugging...

3 rules:

1. The analyser must terminate;
2. The computed information must be correct;
3. It is allowed to return an approximative description of the program behaviour.

Static vs. dynamic

Static analysis:

- Work done at compile-time
- Characterizes all executions
- Conservative: approximates concrete program states

Dynamic analysis:

- Run-time overhead
- Characterizes one or a few executions
- Precise: knows the concrete program state
- Can't "look into the future"

Why abstraction?

The bad news: Rice's theorem:

> *For a Turing-complete programming language, for any non-trivial property, the question of whether the computation of a given program satisfies this property is undecidable.*

Solutions:

- verify a model of the program (model checking)
- verify the program interactively with the help of the user (deductive methods)
- computes only an approximation of the behavior of the program
  - Rice's theorem for static analyses:

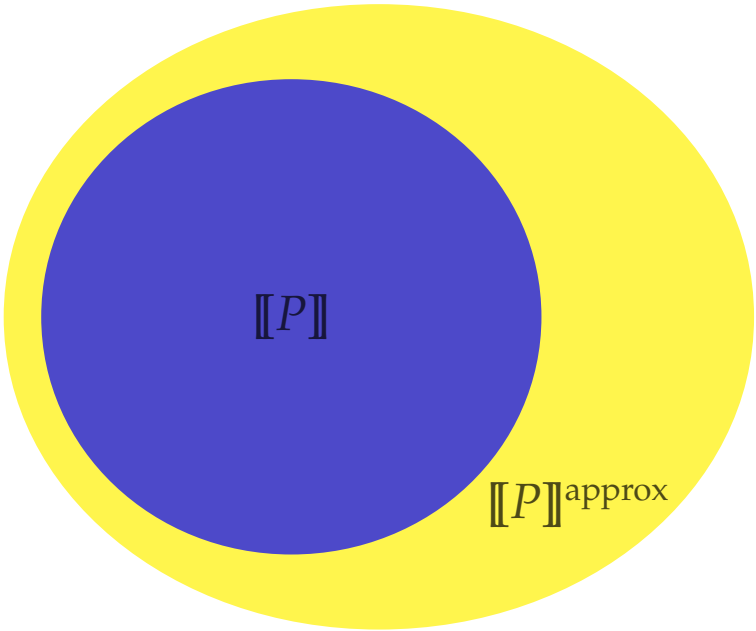
    No static analysis can prove a non-trivial property for any programs in a finite time.

  - It does not mean that it is impossible for *some* programs!

    ASTRÉE[1] analyses electric flight control codes of Airbus (~ 1 M loc)

[1] `http://www.astree.ens.fr/`

A static analysis computes an approximation[2]

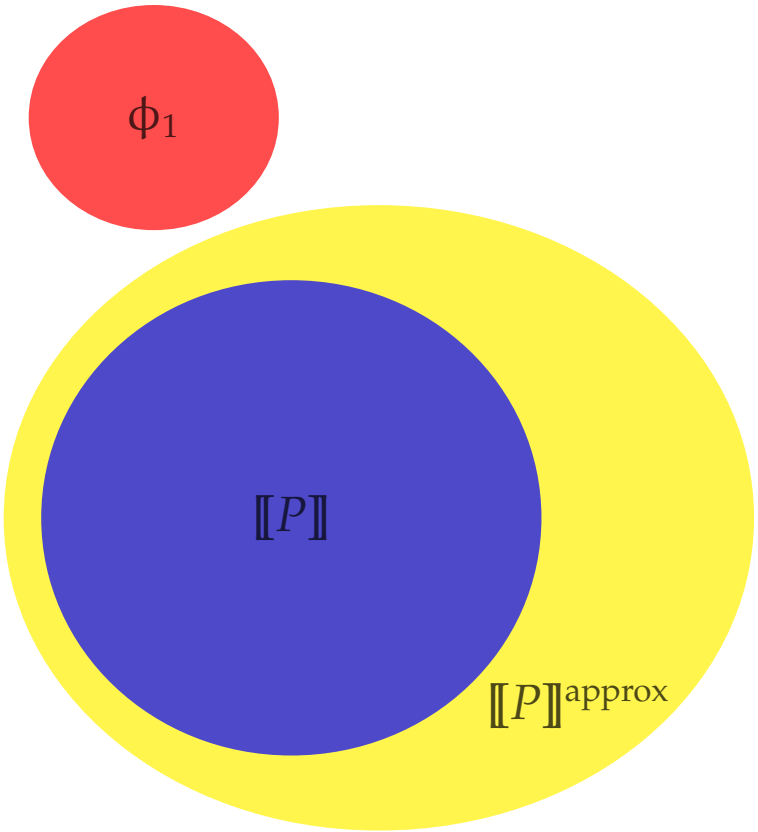


$[\![P]\!]$: concrete semantics (e.g. set of reachable states) (not computable)

$[\![P]\!]^{\text{approx}}$: analyser result (here over-approximation) (computable)

[2] cf `http://www.astree.ens.fr/IntroAbsInt.html`

A static analysis computes an approximation[2]



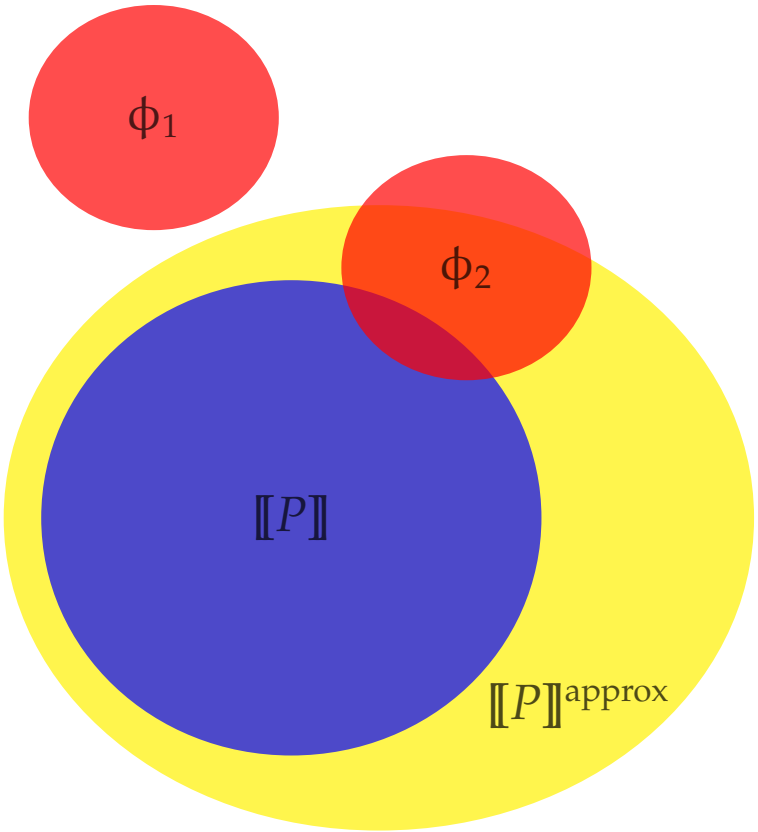
- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1:	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] cf `http://www.astree.ens.fr/IntroAbsInt.html`

A static analysis computes an approximation[2]



- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

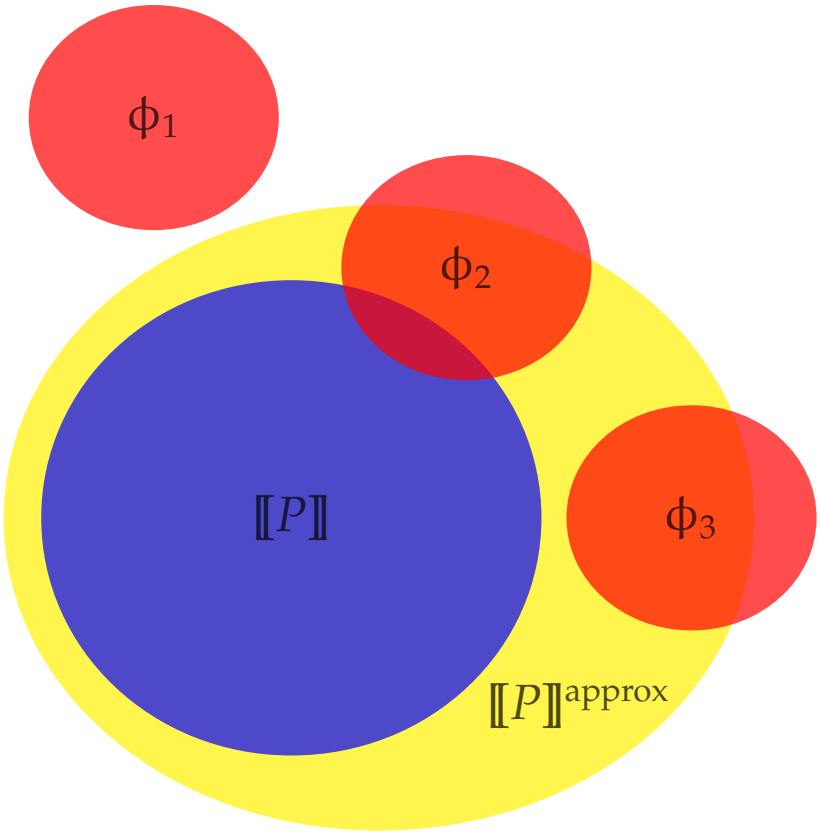
- P is unsafe w.r.t. ϕ_2 and the analyser warns about it

$$[\![P]\!] \cap \phi_2 \neq \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_2 \neq \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1, ϕ_2:	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] cf `http://www.astree.ens.fr/IntroAbsInt.html`

A static analysis computes an approximation[2]



- P is safe w.r.t. ϕ_1 and the analyser proves it

$$[\![P]\!] \cap \phi_1 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_1 = \emptyset$$

- P is unsafe w.r.t. ϕ_2 and the analyser warns about it

$$[\![P]\!] \cap \phi_2 \neq \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_2 \neq \emptyset$$

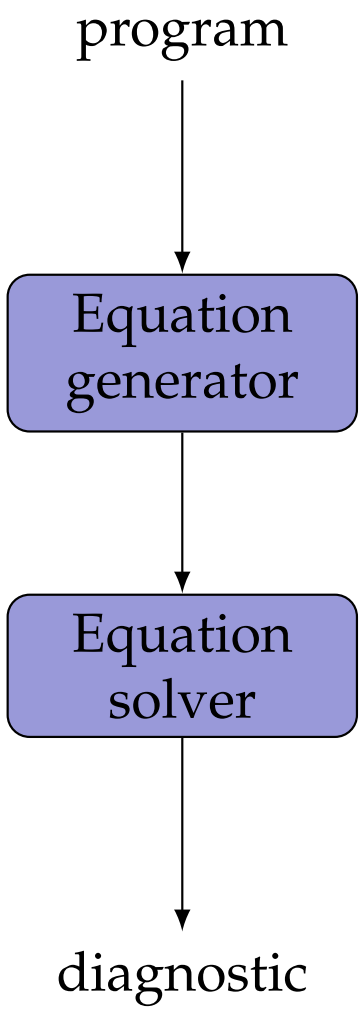
- but P is safe w.r.t. ϕ_3 and the analyser can't prove it (this is called a *false alarm*)

$$[\![P]\!] \cap \phi_3 = \emptyset \qquad [\![P]\!]^{\text{approx}} \cap \phi_3 \neq \emptyset$$

$[\![P]\!]$:	concrete semantics (e.g. set of reachable states)	(not computable)
ϕ_1, ϕ_2, ϕ_3:	erroneous/dangerous set of states	(computable)
$[\![P]\!]^{\text{approx}}$:	analyser result (here over-approximation)	(computable)

[2] cf `http://www.astree.ens.fr/IntroAbsInt.html`

Common structure of analyses



An analysis can be separated into two parts:

1. From a program description, producing an equation system (analysis specification)
   - the solutions of the system must be proved correct w.r.t. the program semantics
2. Solving the system
   - *fixpoint* iterations in *lattice* structures

Dataflow analysis: examples

Reachable definitions : May a definition reach a given point ?
(Dependency analysis between instructions)

Available expressions : What are the expressions already computed at a given point ?
(Re-use of expression computations)

Live variables : Is a variable used in the future ?
(Assignments deletion, Register allocation)

Reachable definition analysis

Determine the set of definitions (assignments) that may reach a program point

Factorial function :

```
1. y := x;
2. z := 1;
3. while y > 1 do
4.    z := z * y;
5.    y := y - 1;
   end
6. y := 0;
```

At point 4, the definition that occurs at labels 1, 2, 4 and 5 are reachable (not for label 6).

Reachable definition analysis

A definition is represented by a couple $(v, l) \in Var \times Lab^?$ with $Lab^? = Lab \cup \{?\}$.

(v, l) : "the variable v has been defined at program point l and has not been modified since"

$(v, ?)$: "the variable v is not initialised"

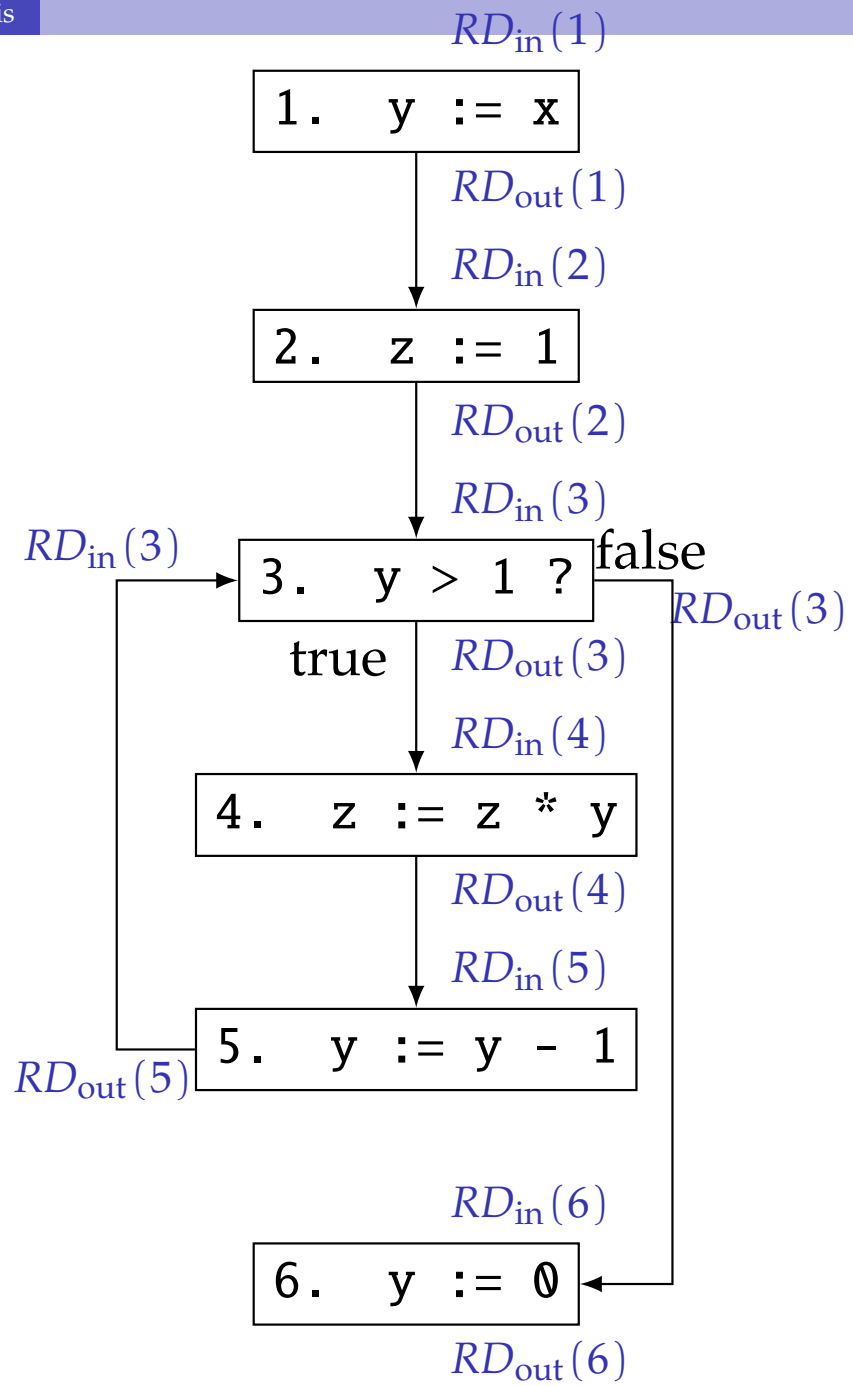
We compute two sets at each label (program point) l:

$RD_{\text{in}}(l)$ = the definitions that enter in l (*i.e.*reachable)

$RD_{\text{out}}(l)$ = the definitions that exit from l (auxiliary set)

Each instruction define some relations between theses set of definitions

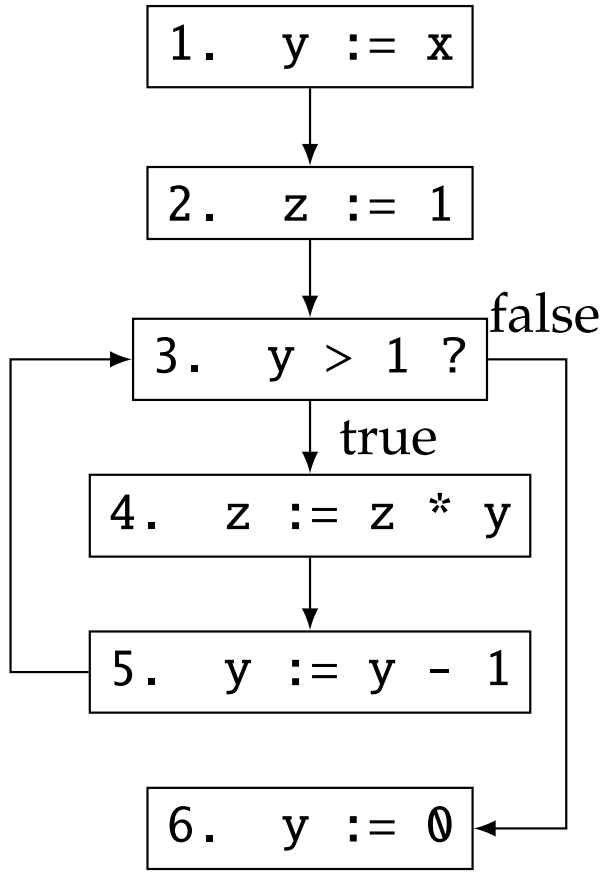
```
1. y := x;
2. z := 1;
3. while y > 1 do
4.     z := z * y;
5.     y := y - 1;
   end
6. y := 0;
```



Reachable definition analysis: equations (1)

An assignment deletes the previous definitions of the assigned variable.

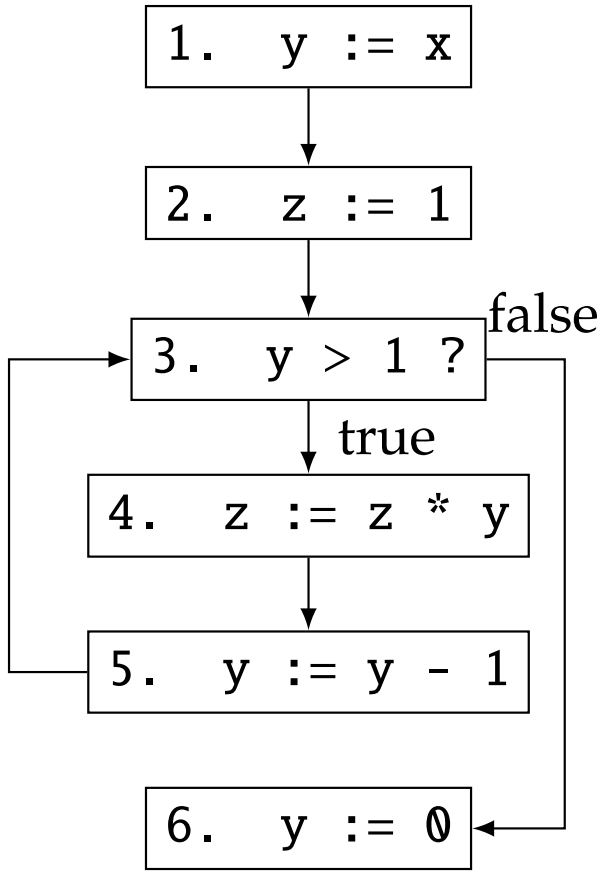
$$RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 1)\}$$
$$RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z}, l) \mid l \in Lab^?\} \cup \{(\mathbf{z}, 2)\}$$
$$RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z}, l) \mid l \in Lab^?\} \cup \{(\mathbf{z}, 4)\}$$
$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 5)\}$$
$$RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 6)\}$$



Reachable definition analysis: equations (2)

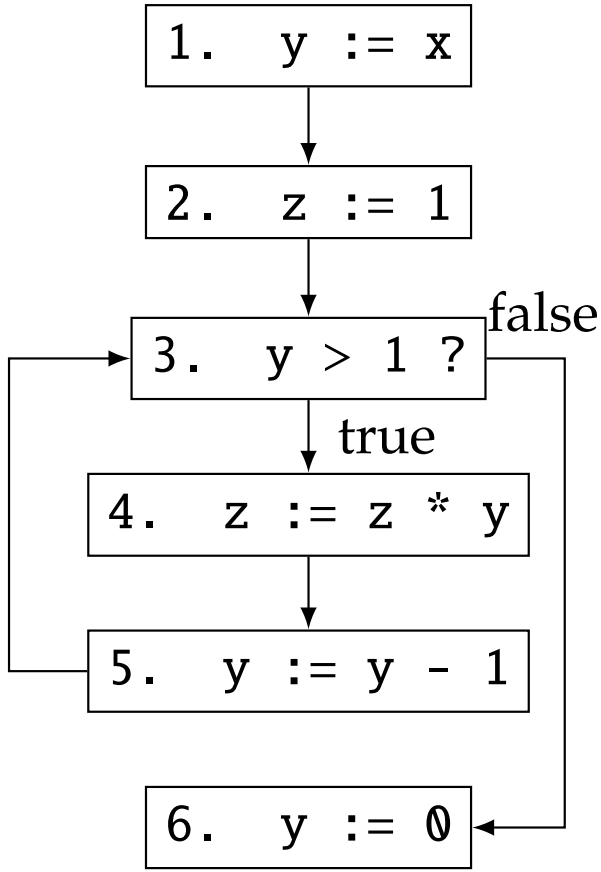
Definitions that are reachable after an instruction, are reachable before the next instruction.

$$RD_{\text{in}}(1) = \{(v, ?) \mid v \in Var\}$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3)$$



Reachable definition analysis : a solution

$$
\begin{aligned}
RD_{\text{in}}(1) &= \{(x,?),(y,?),(z,?)\}\\
RD_{\text{in}}(2) &= \{(x,?),(y,1),(z,?)\}\\
RD_{\text{in}}(3) &= \{(x,?),(y,1),(y,5),(z,2),(z,4)\}\\
RD_{\text{in}}(4) &= \{(x,?),(y,1),(y,5),(z,2),(z,4)\}\\
RD_{\text{in}}(5) &= \{(x,?),(y,1),(y,5),(z,4)\}\\
RD_{\text{in}}(6) &= \{(x,?),(y,1),(y,5),(z,2),(z,4)\}\\
RD_{\text{out}}(1) &= \{(x,?),(y,1),(z,?)\}\\
RD_{\text{out}}(2) &= \{(x,?),(y,1),(z,2)\}\\
RD_{\text{out}}(3) &= \{(x,?),(y,1),(y,5),(z,2),(z,4)\}\\
RD_{\text{out}}(4) &= \{(x,?),(y,1),(y,5),(z,4)\}\\
RD_{\text{out}}(5) &= \{(x,?),(y,5),(z,4)\}\\
RD_{\text{out}}(6) &= \{(x,?),(y,6),(z,2),(z,4)\}
\end{aligned}
$$



We observe that $(y,1),(y,5) \in RD_{\text{in}}(6)$.

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Iteration 0: $\vec{\emptyset}$

$$RD_{\text{in}}(1) = \emptyset \qquad RD_{\text{out}}(1) = \emptyset$$
$$RD_{\text{in}}(2) = \emptyset \qquad RD_{\text{out}}(2) = \emptyset$$
$$RD_{\text{in}}(3) = \emptyset \qquad RD_{\text{out}}(3) = \emptyset$$
$$RD_{\text{in}}(4) = \emptyset \qquad RD_{\text{out}}(4) = \emptyset$$
$$RD_{\text{in}}(5) = \emptyset \qquad RD_{\text{out}}(5) = \emptyset$$
$$RD_{\text{in}}(6) = \emptyset \qquad RD_{\text{out}}(6) = \emptyset$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 1: $F(\vec{\emptyset})$

$RD_{\text{in}}(1) =$	$\{(x,?),(y,?),(z,?)\}$	$RD_{\text{out}}(1) =$	$\{(y,1)\}$
$RD_{\text{in}}(2) =$	$\emptyset$	$RD_{\text{out}}(2) =$	$\{(z,2)\}$
$RD_{\text{in}}(3) =$	$\emptyset$	$RD_{\text{out}}(3) =$	$\emptyset$
$RD_{\text{in}}(4) =$	$\emptyset$	$RD_{\text{out}}(4) =$	$\{(z,4)\}$
$RD_{\text{in}}(5) =$	$\emptyset$	$RD_{\text{out}}(5) =$	$\{(y,5)\}$
$RD_{\text{in}}(6) =$	$\emptyset$	$RD_{\text{out}}(6) =$	$\{(y,6)\}$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 2: $F^2(\vec{\emptyset})$

$RD_{\text{in}}(1) =$	$\{(x,?),(y,?),(z,?)\}$	$RD_{\text{out}}(1) =$	$\{(x,?),(y,1),(z,?)\}$
$RD_{\text{in}}(2) =$	$\{(y,1)\}$	$RD_{\text{out}}(2) =$	$\{(z,2)\}$
$RD_{\text{in}}(3) =$	$\{(y,5),(z,2)\}$	$RD_{\text{out}}(3) =$	$\emptyset$
$RD_{\text{in}}(4) =$	$\emptyset$	$RD_{\text{out}}(4) =$	$\{(z,4)\}$
$RD_{\text{in}}(5) =$	$\{(z,4)\}$	$RD_{\text{out}}(5) =$	$\{(y,5)\}$
$RD_{\text{in}}(6) =$	$\emptyset$	$RD_{\text{out}}(6) =$	$\{(y,6)\}$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 3: $F^3(\vec{\emptyset})$

$RD_{\text{in}}(1) =$	$\{(x,?),(y,?),(z,?)\}$	$RD_{\text{out}}(1) =$	$\{(x,?),(y,1),(z,?)\}$
$RD_{\text{in}}(2) =$	$\{(x,?),(y,1),(z,?)\}$	$RD_{\text{out}}(2) =$	$\{(y,1),(z,2)\}$
$RD_{\text{in}}(3) =$	$\{(y,5),(z,2)\}$	$RD_{\text{out}}(3) =$	$\{(y,5),(z,2)\}$
$RD_{\text{in}}(4) =$	$\emptyset$	$RD_{\text{out}}(4) =$	$\{(z,4)\}$
$RD_{\text{in}}(5) =$	$\{(z,4)\}$	$RD_{\text{out}}(5) =$	$\{(y,5),(z,4)\}$
$RD_{\text{in}}(6) =$	$\emptyset$	$RD_{\text{out}}(6) =$	$\{(y,6)\}$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Iteration 4: $F^4(\vec{\emptyset})$

$$RD_{\text{in}}(1) = \{(\mathbf{x},?),(\mathbf{y},?),(\mathbf{z},?)\} \qquad RD_{\text{out}}(1) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\}$$
$$RD_{\text{in}}(2) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\} \qquad RD_{\text{out}}(2) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},2)\}$$
$$RD_{\text{in}}(3) = \{(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},2),(\mathbf{z},4)\} \qquad RD_{\text{out}}(3) = \{(\mathbf{y},5),(\mathbf{z},2)\}$$
$$RD_{\text{in}}(4) = \{(\mathbf{y},5),(\mathbf{z},2)\} \qquad RD_{\text{out}}(4) = \{(\mathbf{z},4)\}$$
$$RD_{\text{in}}(5) = \{(\mathbf{z},4)\} \qquad RD_{\text{out}}(5) = \{(\mathbf{y},5),(\mathbf{z},4)\}$$
$$RD_{\text{in}}(6) = \{(\mathbf{y},5),(\mathbf{z},2)\} \qquad RD_{\text{out}}(6) = \{(\mathbf{y},6)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 5: $F^5(\vec{\emptyset})$

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(y,5),(z,2)\} \qquad RD_{\text{out}}(4) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(z,4)\} \qquad RD_{\text{out}}(5) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(y,5),(z,2)\} \qquad RD_{\text{out}}(6) = \{(y,6),(z,2)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 6: $F^6(\vec{\emptyset})$

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(y,6),(z,2)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 7: $F^7(\vec{\emptyset})$

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration 8: $F^8(\vec{\emptyset})$

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(x,?),(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(y,1),(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(x,?),(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(x,?),(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(x,?),(y,1),(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(x,?),(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \quad (s_6)$$

Iteration

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(x,?),(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(x,?),(y,1),(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(x,?),(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(x,?),(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : iterative computation

The solution can be computed by iteration. $RD_{\text{in}}(l)$ and $RD_{\text{out}}(l)$ are initialised with $\emptyset$ and their values are recomputed until stabilisation.

Equations : $\vec{RD} = F(\vec{RD})$

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Iteration

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(x,?),(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(x,?),(y,1),(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(x,?),(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(x,?),(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : several solutions ?

The equation system admits several solutions.

Equations :

$$RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\}$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\}$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\}$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\}$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\}$$

Previous solution:

$$RD_{\text{in}}(1) = \{(x,?),(y,?),(z,?)\} \qquad RD_{\text{out}}(1) = \{(x,?),(y,1),(z,?)\}$$
$$RD_{\text{in}}(2) = \{(x,?),(y,1),(z,?)\} \qquad RD_{\text{out}}(2) = \{(x,?),(y,1),(z,2)\}$$
$$RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$$
$$RD_{\text{in}}(4) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(4) = \{(x,?),(y,1),(y,5),(z,4)\}$$
$$RD_{\text{in}}(5) = \{(x,?),(y,1),(y,5),(z,4)\} \qquad RD_{\text{out}}(5) = \{(x,?),(y,5),(z,4)\}$$
$$RD_{\text{in}}(6) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\} \qquad RD_{\text{out}}(6) = \{(x,?),(y,6),(z,2),(z,4)\}$$

Reachable definition analysis : several solutions ?

The equation system admits several solutions.

Equations :

$$
\begin{array}{ll}
RD_{\text{in}}(1) = \{(v,?) \mid v \in Var\} & RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\} \\
RD_{\text{in}}(2) = RD_{\text{out}}(1) & RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \\
RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) & RD_{\text{out}}(3) = RD_{\text{in}}(3) \\
RD_{\text{in}}(4) = RD_{\text{out}}(3) & RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \\
RD_{\text{in}}(5) = RD_{\text{out}}(4) & RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \\
RD_{\text{in}}(6) = RD_{\text{out}}(3) & RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\}
\end{array}
$$

An other solution:

$$
\begin{array}{ll}
RD'_{\text{in}}(1) = \{(\mathbf{x},?),(\mathbf{y},?),(\mathbf{z},?)\} & RD'_{\text{out}}(1) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\} \\
RD'_{\text{in}}(2) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\} & RD'_{\text{out}}(2) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},2)\} \\
RD'_{\text{in}}(3) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},2),(\mathbf{z},4)\} & RD'_{\text{out}}(3) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},2),(\mathbf{z},4)\} \\
RD'_{\text{in}}(4) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},2),(\mathbf{z},4)\} & RD'_{\text{out}}(4) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},4)\} \\
RD'_{\text{in}}(5) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},4)\} & RD'_{\text{out}}(5) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},5),(\mathbf{z},4)\} \\
RD'_{\text{in}}(6) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},1),(\mathbf{y},5),(\mathbf{z},2),(\mathbf{z},4)\} & RD'_{\text{out}}(6) = \{(\mathbf{x},?),(\mathbf{x},1),(\mathbf{y},6),(\mathbf{z},2),(\mathbf{z},4)\}
\end{array}
$$

Choosing the best solution

Remark :

$$RD_{\text{in}}(1) \subseteq RD'_{\text{in}}(1),\ \ RD_{\text{out}}(1) \subseteq RD'_{\text{out}}(1),\ \ \ldots,\ RD_{\text{out}}(6) \subseteq RD'_{\text{out}}(6)$$

RD gives an information more precise than RD' :

- $RD_{\text{in}}(3) = \{(x,?),(y,1),(y,5),(z,2),(z,4)\}$
  "the value of x at point 3 is not initialised"
- $RD'_{\text{in}}(3) = \{(x,?),(x,1),(y,1),(y,5),(z,2),(z,4)\}$
  "the value of x at point 3 is not initialised, or has been defined at point 1"

Between two comparables solutions (e.g. $\vec{RD} \subseteq \vec{RD}'$), we prefer the smallest.

Theoretical result: there always exists a smallest solution

Equation resolution

The previous analysis is a solution of an equation system of the form

$$\begin{cases} x_1 &= f_1(x_1, \ldots, x_n) \\ &\vdots \\ x_n &= f_n(x_1, \ldots, x_n) \end{cases} \qquad \text{or} \quad \vec{x} = \vec{f}(\vec{x})$$

called fixpoint equations.

It is a common mathematical problem that raises two questions:

1. Existence and uniqueness (in what sense ?) of the solution ?
2. Effective computation method ?

A few observations about the previous analysis:

- The x_i are sets, that can be ordered by set inclusion $\subseteq$
- The functions f_i are monotones (*croissantes*) for the partial order $\subseteq$

$\Rightarrow$ Suitable mathematical framework: lattice theory

Poset

Definition

A *partially ordered set (poset)* is a couple $(A, \sqsubseteq)$ with A a set, and $\sqsubseteq$ a partial order relation, *i.e.*:

$$\begin{array}{ll} \forall x \in A,\ x \sqsubseteq x & \text{(reflexivity)} \\ \forall x, y \in A,\ x \sqsubseteq y \ \wedge\ y \sqsubseteq x \ \Rightarrow\ x = y & \text{(antisymmetry)} \\ \forall x, y, z \in A,\ x \sqsubseteq y \ \wedge\ y \sqsubseteq z \ \Rightarrow\ x \sqsubseteq z & \text{(transitivity)} \end{array}$$

Examples

- $(\mathbb{N}, \leqslant)$ (total : $\forall x, y,\ x \sqsubseteq y \ \vee\ y \sqsubseteq x$)
- $(\mathbb{N}, \text{"}\textit{is a divisor of}\text{"})$ written $(\mathbb{N}, |)$
- $(\mathcal{P}(X), \subseteq)$ with X any set
- $(A^*, \text{"}\textit{to be a prefix of}\text{"})$ with A an alphabet

Exercice

Show that $(\mathbb{N}, \textit{"is a divisor of"})$ is a poset.

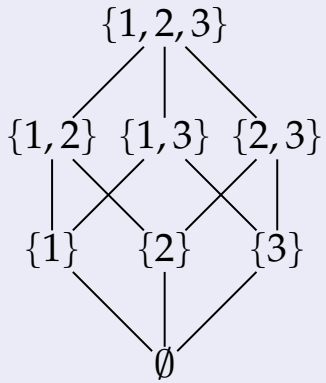
Hasse diagram

Definition

A 2D-drawing (set of points and segments) is *a Hasse diagram* of a poset $(A, \sqsubseteq)$ iff

- each element of A is associated with a point
- if $x \sqsubseteq y$ with $x \neq y$ and $\neg\exists z, x \sqsubseteq z \sqsubset y$ then
  - a segment connects the points p_x and p_y that are associated respectively with x and y
  - the ordinate (vertical scale) of p_x is lower than the ordinate of p_y

Example



is an Hasse diagram of the poset $(\mathcal{P}(\{1,2,3\}), \subseteq)$

Exercice

Give a Hasse diagram of the poset $(\{1, 2, 3, 4, 6, 8, 12\}, |)$

Lattice

Definition

A *lattice* is a 4-tuple $(A, \sqsubseteq, \sqcup, \sqcap)$ with

- $(A, \sqsubseteq)$ a poset,
- $\sqcup$ a binary least upper bound:

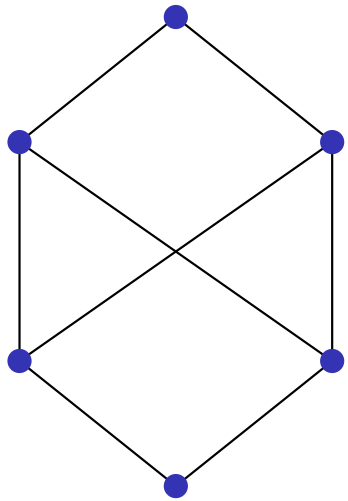
$$\forall x, y \in A,\ x \sqsubseteq x \sqcup y \ \wedge\ y \sqsubseteq x \sqcup y$$
$$\forall x, y, z \in A,\ x \sqsubseteq z \ \wedge\ y \sqsubseteq z \ \Rightarrow\ x \sqcup y \sqsubseteq z$$

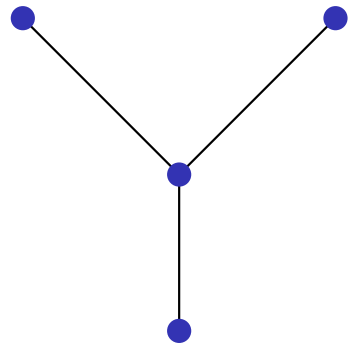
- $\sqcap$ a binary greatest lower bound:

$$\forall x, y \in A,\ x \sqcap y \sqsubseteq x \ \wedge\ x \sqcap y \sqsubseteq y$$
$$\forall x, y, z \in A,\ z \sqsubseteq x \ \wedge\ z \sqsubseteq y \ \Rightarrow\ z \sqsubseteq x \sqcap y$$

Exercice

Between the following diagrams, which represent lattices ?





Exercice

Give the lattice structure of $(\mathbb{N}, \leqslant)$ and $(\mathbb{N}, |)$.

Complete lattice

Definition

A complete lattice is a triple $(A, \sqsubseteq, \bigsqcup)$ with

- $(A, \sqsubseteq)$ a poset,
- $\bigsqcup$ a least upper bound : for all subsets S of A,
  - $\forall a \in S,\ a \sqsubseteq \bigsqcup S$
  - $\forall b \in A, (\forall a \in S,\ a \sqsubseteq b) \ \Rightarrow\ \bigsqcup S \sqsubseteq b$

A complete lattice necessarily posseses *a greatest lower bound* $\bigsqcap$

i.e. : for all subsets S of A,

- $\forall a \in S,\ \bigsqcap S \sqsubseteq a$
- $\forall b \in A, (\forall a \in S,\ b \sqsubseteq a) \ \Rightarrow\ b \sqsubseteq \bigsqcap S$

Just consider

$$\bigsqcap S = \bigsqcup \{\, y \mid \forall x \in S, y \sqsubseteq x \,\}$$

Examples

1. For all set X, $(\mathcal{P}(X), \subseteq, \bigcup)$ is a complete lattice for which $\bigcap$ is a greatest lower bound.
2. All finite lattice is complete.

Exercice

Show that any complete lattice admits

- *a greatest element* $\top$ $(\forall x, x \sqsubseteq \top)$
- *a least element* $\bot$ $(\forall x, \bot \sqsubseteq x)$

Fixpoints, post-fixpoints and pre-fixpoints

Definition

Consider $f \in A \to A$ with $(A, \sqsubseteq)$ a poset, an element $x \in A$

- is a *fixpoint of* f iff $f(x) = x$
- is a *greatest fixpoint of* f iff $f(x) = x$ and $\forall y,\ f(y) = y \ \Rightarrow y \sqsubseteq x$
- is a *least fixpoint of* f iff $f(x) = x$ and $\forall y,\ f(y) = y \ \Rightarrow x \sqsubseteq y$
- is a *post-fixpoint of* f iff $f(x) \sqsubseteq x$
- is a *pre-fixpoint of* f iff $x \sqsubseteq f(x)$

Definition

Let $f \in A \to A$, f is *monotone* iff

$$\forall x, y \in,\ x \sqsubseteq y \ \Rightarrow\ f(x) \sqsubseteq f(y)$$

Fixpoints, post-fixpoints and pre-fixpoints

Theorem (Knaster-Tarski)

In a complete lattice $(A, \sqsubseteq, \bigsqcup)$*, for all monotone functions* $f \in A \to A$*,*

- *the least fixpoint* $\mathrm{lfp}(f)$ *of* f *exists and is* $\bigsqcap \{x \in A \mid f(x) \sqsubseteq x\}$*,*
- *the greatest fixpoint* $\mathrm{gfp}(f)$ *of* f *exists and is* $\bigsqcup \{x \in A \mid x \sqsubseteq f(x)\}$*,*

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  –

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ a greatest lower bound of P
  -

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  -

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a lower bound of P

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a lower bound of P
  $\Rightarrow f(a) = a$ — antisymmetry

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a lower bound of P
  $\Rightarrow f(a) = a$ — antisymmetry

  hence $a \in Fix(f)$

Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a lower bound of P
  $\Rightarrow f(a) = a$ — antisymmetry

  hence $a \in Fix(f)$
- If $x \in Fix(f)$ then $x \in P$, hence $a \sqsubseteq x$ because a is a lower bound of P. Hence $a = \mathrm{lfp}(f)$.

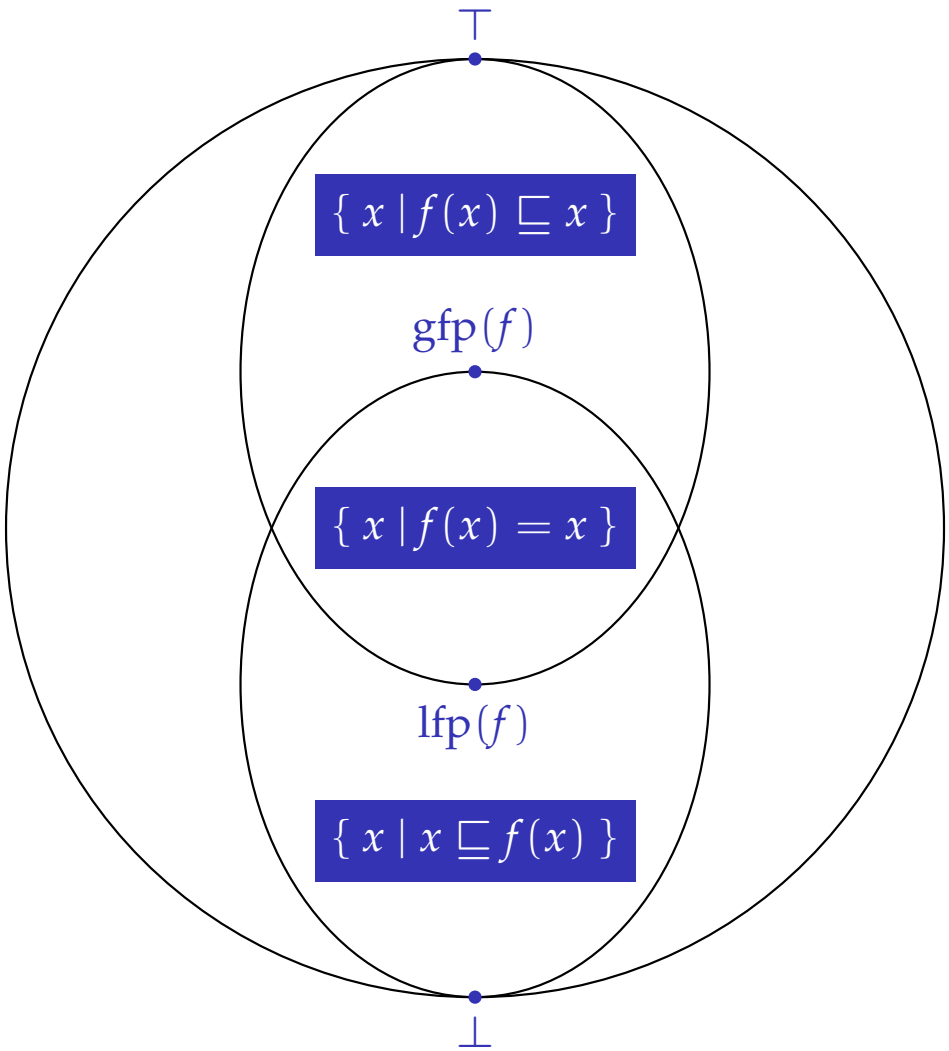
Proof of the Knaster-Tarski theorem

- Let us pose $a = \bigsqcap P$ with $P = PostFix(f) = \{x \mid f(x) \sqsubseteq x\}$.
- For all $x \in P$, we have
  - $a \sqsubseteq x$ — a greatest lower bound of P
  - $f(a) \sqsubseteq f(x)$ — f monotone
  - $f(a) \sqsubseteq x$ — def. P and transitivity

  hence $f(a) \sqsubseteq x$ for all $x \in P$
- $f(a) \sqsubseteq a$ — previous result and a greatest lower bound of P
  $\Rightarrow f(f(a)) \sqsubseteq f(a)$ — f monotone
  $\Rightarrow f(a) \in P$ — def. P
  $\Rightarrow a \sqsubseteq f(a)$ — a lower bound of P
  $\Rightarrow f(a) = a$ — antisymmetry

  hence $a \in Fix(f)$
- If $x \in Fix(f)$ then $x \in P$, hence $a \sqsubseteq x$ because a is a lower bound of P. Hence $a = \mathrm{lfp}(f)$.
- Proof of $\mathrm{gfp}(f) = \bigsqcup PreFix(f)$ by duality.

Fixpoints, post-fixpoints and pre-fixpoints



$$\top = \bigsqcup \{ x \mid f(x) \sqsubseteq x \}$$

$$\mathrm{gfp}(f) = \bigsqcup \{ x \mid x \sqsubseteq f(x) \}$$

$$\mathrm{lfp}(f) = \bigsqcap \{ x \mid f(x) \sqsubseteq x \}$$

$$\bot = \bigsqcap \{ x \mid x \sqsubseteq f(x) \}$$

Fixpoint computation

Theorem

Let $(A, \sqsubseteq)$ a poset with a least element $\bot$. Let f a monotone function. If the sequence $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ stabilises from a given rank k (i.e. $f^k(\bot) = f^{k+1}(\bot)$, then $f^k(\bot)$ is the least fixpoint of f.

Proof: Since $\bot \sqsubseteq f(\bot)$ and f is monotone, we can show by induction on $\mathbb{N}$ that $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ is an increasing sequence.

Let k such that $f^k(\bot) = f^{k+1}(\bot)$.

- Hence $f^k(\bot)$ is a fixpoint of f.
- If x is a fixpoint of f, we show by induction on $\mathbb{N}$ that $f^n(\bot) \sqsubseteq x \; \forall n \in \mathbb{N}$. It shows in particular that $f^k(\bot) \sqsubseteq x$.

□

Remark : $\top, f(\top), \ldots, f^n(\top), \ldots$ allows to compute the greatest fixpoint of f.

Fixpoint computation: ascending chain condition

Definition

A poset $(A, \sqsubseteq)$ verifies the ascending chain condition if for all ascending (increasing) sequence $x_0 \sqsubseteq x_1 \sqsubseteq \cdots \sqsubseteq x_n \sqsubseteq \cdots$ there exists an index k from which the sequence is stationary ($\forall n \geq k,\ x_k = x_n$) (*i.e.* the sequence eventually stabilises).

Corollary

Let $(A, \sqsubseteq)$ a poset that verifies the ascending chain condition and f a monotone function. The sequence $\bot, f(\bot), \ldots, f^n(\bot), \ldots$ eventually stabilises. Its limit is the least fixpoint of f.

Remarque : A finite poset verifies the ascending chain condition.

Fixpoint computation: Kleene fixpoint theorem

Definition

Let $(A, \sqsubseteq, \bigsqcup)$ a complete lattice. A function $f \in A \to A$ is continuous iff

$$\forall S \subseteq A,\ \bigsqcup f(S) = f(\bigsqcup S)$$

Remark : a continuous function is necessarily monotone.

Theorem (Kleene fixpoint theorem)

In a complete lattice $(A, \sqsubseteq, \bigsqcup)$, for all continuous function $f \in A \to A$, the least fixpoint $\mathbf{lfp}(f)$ of f is equal to $\bigsqcup \{ f^n(\bot) \mid n \in \mathbb{N} \}$.

Remark : the original theorem is stated for *complete partial order (CPO)*.

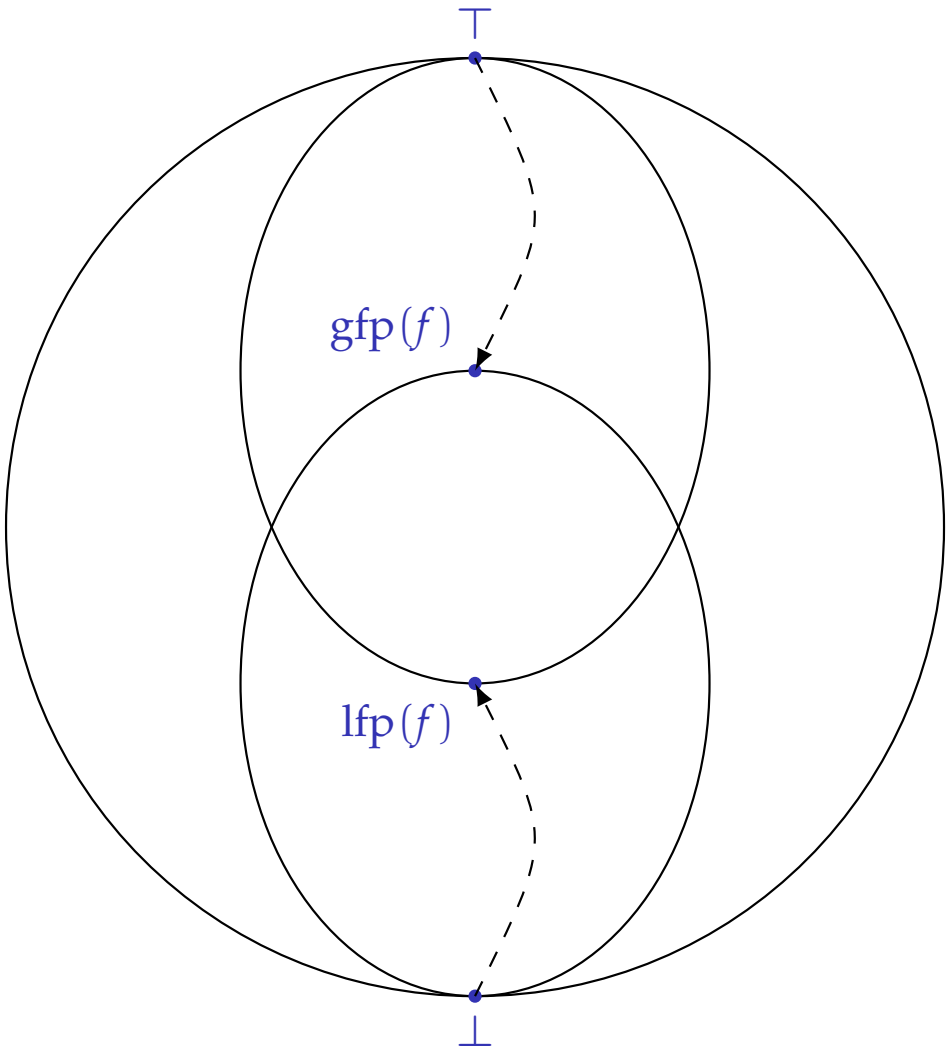
Proof of the Kleene fixpoint theorem

- We have already show that $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$
- $\bigsqcup_{n\geq 0} f^n(\bot)$ is a fixpoint of f:

$$
\begin{aligned}
& f(\bigsqcup_{n\geq 0} f^n(\bot)) \\
&= \bigsqcup_{n\geq 0} f(f^n(\bot)) && f \text{ is continuous} \\
&= f^0(\bot) \sqcup \bigsqcup_{n\geq 0} f^{n+1}(\bot) && (\bot \sqcup x = x) \text{ and def. } f^n \\
&= \bigsqcup_{n\geq 0} f^n(\bot)
\end{aligned}
$$

- It is the least fixpoint: consider $x \in Fix(f)$
  - $f^0(\bot) = \bot \sqsubseteq x$
  - $\forall n \geq 0 : f^n(\bot) \sqsubseteq x$ induction on n, because f monotone and $f(x) = x$
  - $\bigsqcup_{n\geq 0} f^n(\bot) \sqsubseteq x$ greater bound

Fixpoint computation



$\top, f(\top), \ldots, f^n(\top), \ldots, \mathbf{gfp}\, f$

$\bot, f(\bot), \ldots, f^n(\bot), \ldots, \mathbf{lfp}\, f$

The underlying lattice structure of the Reaching definitions analysis

$\left(\mathcal{P}(Var \times Lab^?), \subseteq, \bigcup\right)$ is a complete lattice.

Lattice product: if $(L_1, \sqsubseteq_1, \bigsqcup_1)$ and $(L_2, \sqsubseteq_2), \bigsqcup_1$ are complete lattices, their *product* $L_1 \times L_2$ is the complete lattice $(L_1 \times L_2, \sqsubseteq_{L_1 \times L_2}, \bigsqcup_{L_1 \times L_2})$ defined par:

$$\begin{array}{rcl} (x_1, x_2) \sqsubseteq (y_1, y_2) & \Leftrightarrow & x_1 \sqsubseteq_1 y_1 \wedge x_2 \sqsubseteq_2 y_2 \\ \bigsqcup_{L_1 \times L_2} S & = & (\bigsqcup_1 proj_1(S), \bigsqcup_2 proj_2(S)),\ \forall S \subseteq L_1 \times L_2 \end{array}$$

Conclusion :

$$(RD_s(1), RD_e(1), \ldots, RD_s(6), RD_e(6)) \in \mathcal{P}(Var \times Lab^?)^{12}$$

and $\left(\mathcal{P}(Var \times Lab^?)^{12}, \subseteq^{12}, \bigcup^{12}\right)$ is a complete lattice.

Exercise: Justify the termination of the analysis.

Accelerated iterations

Consider the system

$$\begin{cases} x_1 &= f_1(x_1, \ldots, x_n) \\ &\vdots \\ x_n &= f_n(x_1, \ldots, x_n) \end{cases}$$

Standard iterations:

$$\begin{aligned} x_1^{i+1} &= f_1(x_1^i, \ldots, x_n^i) \\ x_2^{i+1} &= f_2(x_1^i, \ldots, x_n^i) \\ &\vdots \\ x_n^{i+1} &= f_n(x_1^i, \ldots, x_n^i) \end{aligned}$$

Chaotic iterations: at each step, we only use selected equations, without forgetting any equation infinitely often. $L \in \mathbb{N} \to \mathcal{P}(\{1, \ldots, n\})$ gives the iteration strategy (*i.e.* at the i^{th} iteration, equations in L_i are used).

$$\begin{aligned} x_j^{i+1} = f_j(x_1^i, \ldots, x_n^i) \quad & \text{if } j \in L_{i+1} \\ x_j^{i+1} = x_j^i \quad & \text{if } j \notin L_{i+1} \end{aligned}$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(v, ?) \mid v \in Var\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 1)\} \quad (s_1)$$

$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z}, l) \mid l \in Lab^?\} \cup \{(\mathbf{z}, 2)\} \quad (s_2)$$

$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$

$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z}, l) \mid l \in Lab^?\} \cup \{(\mathbf{z}, 4)\} \quad (s_4)$$

$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 5)\} \quad (s_5)$$

$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y}, l) \mid l \in Lab^?\} \cup \{(\mathbf{y}, 6)\} \quad (s_6)$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(\mathbf{x},?),(\mathbf{y},?),(\mathbf{z},?)\} \quad (e_1) \qquad RD_{\text{out}}(1) = RD_{\text{in}}(1) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},1)\} \quad (s_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2) \qquad RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3) \qquad RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4) \qquad RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5) \qquad RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6) \qquad RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(\mathbf{x},?),(\mathbf{y},?),(\mathbf{z},?)\} \quad (e_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6)$$

$$RD_{\text{out}}(1) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\} \quad (s_1)$$
$$RD_{\text{out}}(2) = RD_{\text{in}}(2) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{out}}(3) = RD_{\text{in}}(3) \quad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{in}}(4) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{in}}(5) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{in}}(6) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(\mathtt{x},?),(\mathtt{y},?),(\mathtt{z},?)\} \quad (e_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6)$$

$$RD_{\text{out}}(1) = \{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},?)\} \quad (s_1)$$
$$RD_{\text{out}}(2) = RD_{\text{out}}(1) \setminus \{(\mathtt{z},l) \mid l \in Lab^?\} \cup \{(\mathtt{z},2)\} \quad (s_2)$$
$$RD_{\text{out}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(\mathtt{z},l) \mid l \in Lab^?\} \cup \{(\mathtt{z},4)\} \quad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(\mathtt{y},l) \mid l \in Lab^?\} \cup \{(\mathtt{y},5)\} \quad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(\mathtt{y},l) \mid l \in Lab^?\} \cup \{(\mathtt{y},6)\} \quad (s_6)$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(\mathbf{x},?),(\mathbf{y},?),(\mathbf{z},?)\} \quad (e_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6)$$

$$RD_{\text{out}}(1) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},?)\} \quad (s_1)$$
$$RD_{\text{out}}(2) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},2)\} \quad (s_2)$$
$$RD_{\text{out}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \quad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \quad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \quad (s_6)$$

Example

Remark: the equation system can be simplified (at least by hand).

$$RD_{\text{in}}(1) = \{(\mathtt{x},?),(\mathtt{y},?),(\mathtt{z},?)\} \quad (e_1)$$
$$RD_{\text{in}}(2) = RD_{\text{out}}(1) \quad (e_2)$$
$$RD_{\text{in}}(3) = RD_{\text{out}}(2) \cup RD_{\text{out}}(5) \quad (e_3)$$
$$RD_{\text{in}}(4) = RD_{\text{out}}(3) \quad (e_4)$$
$$RD_{\text{in}}(5) = RD_{\text{out}}(4) \quad (e_5)$$
$$RD_{\text{in}}(6) = RD_{\text{out}}(3) \quad (e_6)$$

$$RD_{\text{out}}(1) = \{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},?)\} \quad (s_1)$$
$$RD_{\text{out}}(2) = \{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},2)\} \quad (s_2)$$
$$RD_{\text{out}}(3) = \{(\mathtt{x},?),(\mathtt{y},1),(\mathtt{z},2)\} \cup RD_{\text{out}}(5) \quad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(\mathtt{z},l) \mid l \in Lab^?\} \cup \{(\mathtt{z},4)\} \quad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(\mathtt{y},l) \mid l \in Lab^?\} \cup \{(\mathtt{y},5)\} \quad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(\mathtt{y},l) \mid l \in Lab^?\} \cup \{(\mathtt{y},6)\} \quad (s_6)$$

Example

It is hence sufficient to solve the following system:

$$RD_{\text{out}}(3) = \{(\mathbf{x},?),(\mathbf{y},1),(\mathbf{z},2)\} \cup RD_{\text{out}}(5) \qquad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(\mathbf{z},l) \mid l \in Lab^?\} \cup \{(\mathbf{z},4)\} \qquad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},5)\} \qquad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(\mathbf{y},l) \mid l \in Lab^?\} \cup \{(\mathbf{y},6)\} \qquad (s_6)$$

Example

It is hence sufficient to solve the following system:

$$RD_{\text{out}}(3) = \{(x,?),(y,1),(z,2)\} \cup RD_{\text{out}}(5) \qquad (s_3)$$
$$RD_{\text{out}}(4) = RD_{\text{out}}(3) \setminus \{(z,l) \mid l \in Lab^?\} \cup \{(z,4)\} \qquad (s_4)$$
$$RD_{\text{out}}(5) = RD_{\text{out}}(4) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,5)\} \qquad (s_5)$$
$$RD_{\text{out}}(6) = RD_{\text{out}}(3) \setminus \{(y,l) \mid l \in Lab^?\} \cup \{(y,6)\} \qquad (s_6)$$

L_i		$\{s_3\}$	$\{s_4\}$	$\{s_5\}$
$RD_{\text{out}}(3)$	$\emptyset$	$\{(x,?),(y,1),(z,2)\}$	$\{(x,?),(y,1),(z,2)\}$	$\{(x,?),(y,1),(z,2)\}$
$RD_{\text{out}}(4)$	$\emptyset$	$\emptyset$	$\{(x,?),(y,1),(z,4)\}$	$\{(x,?),(y,1),(z,4)\}$
$RD_{\text{out}}(5)$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(x,?),(y,5),(z,4)\}$
$RD_{\text{out}}(6)$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$\{s_3\}$	$\{s_4, s_6\}$
$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$	$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$
$\{(x,?),(y,1),(z,4)\}$	$\{(x,?),(y,1),(y,5),(z,4)\}$
$\{(x,?),(y,5),(z,4)\}$	$\{(x,?),(y,5),(z,4)\}$
$\emptyset$	$\{(x,?),(y,6),(z,2),(z,4)\}$

The Work-set Algorithm

forall $i \in \{1, \ldots, n\}$ **do** $x_i := \bot$;
$W := \{1, \ldots, n\}$
repeat
  $i := choose(W)$;
  $tmp := f_i(x_1, \ldots, x_n)$;
  if $tmp \neq x_i$ **then begin** /* *the value of* x_i *has changed* */
    $x_i := tmp$;
    $W := W \cup dependences(x_i)$
until $W = \emptyset$

$choose(S)$: removes one element from a set S.
$dependences(x)$: returns the set of variables that depends on a variable x.

The chaotic iteration (*i.e.* the choice of a good order iteration) can be combined with the "work-set" technique (re-computation only when necessary).

Static program analysis

Plan

Semantics of Java byte code

An example of an operational semantics in the form of an abstract machine (the Java virtual machine).

References:

- Tim Lindholm , Frank Yellin, Java Virtual Machine Specification, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999
- S. Freund , J. Mitchell : A Type System for the Java Bytecode Language and Verifier, Journal of Automated Reasoning, Volume 30, Issue 3-4, Pages: 271 - 321, (2003)

Java

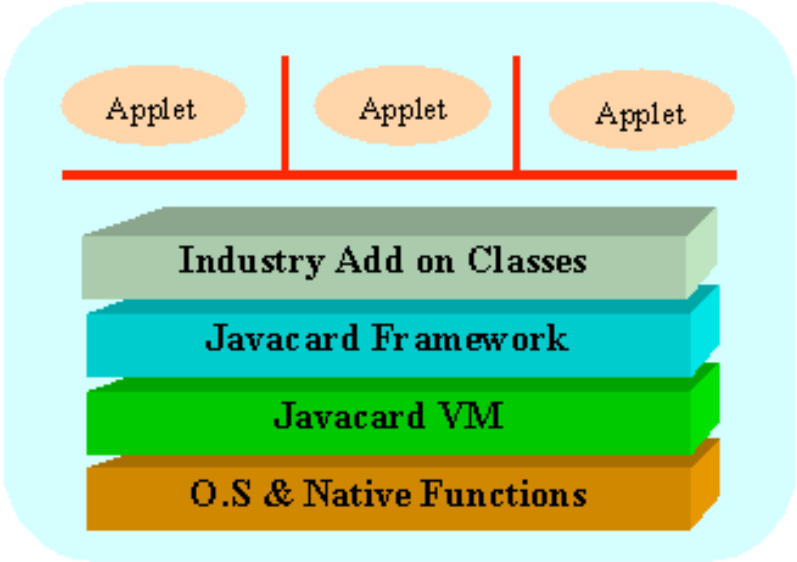
Java: a class-based, object-oriented programming language.

```
public class Bicycle{

    private int gear;

    private int id;

    private static int numberOfBicycles = 0;

    public Bicycle(int startCadence, int startSpeed, int startGear){
        gear = startGear;
        cadence = startCadence;
        id = ++numberOfBicycles;}

     public void setGear(int newValue){
        gear = newValue;}

public class MountainBike extends Bicycle {

    // the MountainBike subclass has one field
    public int seatHeight;
    ... }
```

Java byte code

To enhance portability, Java defines a byte code language that is interpreted by the Java virtual machine.

```
...
7 : ipush 0
8 : iload high
9 : iload low
10 : isub
11 : if_icmpge 56
...
```



Java byte code: factorial

Source code

```
static int factorial(int n) {
 int res;
 for (res = 1; n > 0; n--) res = res * n;
 return res;
}
```

Byte code

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 // push the integer constant 1
1: istore 1 // store it in register 1 (the res variable)
2: iload 0 // push register 0 (the n parameter)
3: ifle 14 // if negative or null, go to PC 14
6: iload 1 // push register 1 (res)
7: iload 0 // push register 0 (n)
8: imul // multiply the two integers at top of stack
9: istore 1 // pop result and store it in register 1
10: iinc 0, -1 // decrement register 0 (n) by 1
11: goto 2 // go to PC 2
14: iload 1 // load register 1 (res)
15: ireturn // return its value to caller
```

The Java virtual machine state

The Java virtual machine has several components:

A code component that contains the byte code of all methods of all classes loaded into the virtual machine. Our machine will have a program counter *pc* pointing to the next instruction to execute.

A heap (or memory) of objects that have been created during execution of the program.

A frame stack of methods under execution. Each frame contains an operand stack and variables local to a method call.

The Java virtual machine specification

iadd

Operation

Add `int`

Format

Forms

iadd = 96 (0x60)

Operand Stack

..., *value1*, *value2* ⇒ ..., *result*

Description

Both *value1* and *value2* must be of type `int`. The values are popped from the operand stack. The `int` *result* is *value1* + *value2*. The *result* is pushed onto the operand stack.

The result is the 32 low-order bits of the true mathematical result in a sufficiently wide two's-complement format, represented as a value of type `int`. If overflow occurs, then the sign of the result may not be the same as the sign of the mathematical sum of the two values.

Despite the fact that overflow may occur, execution of an *iadd* instruction never throws a runtime exception.

Idealized byte code syntax

Instruction ::= `nop`
`const` c	stack manipulation
`pop`	
`dup`	
`dup2`	
`swap`	
`numop` op	
`load` x	local variables manipulation
`store` x	

Some bytecodes are typed (`iconst`, `aload`) to indicate whether they operate on integers, references, arrays, etc.

We leave out the types here, and put them in again when considering byte code verificaiton.

Idealized byte code syntax (continued)

More instructions:

`if` *pc*	jump
`goto` *pc*	
`new` *cl*	heap manipulation
`putfield` *f*	
`getfield` *f*	

Real Java byte code contains many more "if"s: `if_eq`, `if_ge`, `if_nonnull`, `if_acmpeq`,...

The field in a `putfield` *f* instruction carries extra typing information of the form $C : f : t$ where C is the class declaring the field and t is the type of the field.

Idealized byte code syntax (continued)

More instructions

$$\left.\begin{array}{l}\texttt{invokevirtual}\ m_{id}\\ \texttt{return}\end{array}\right\}\ \text{method call and return}$$

The method identifier m_{id} is of form $C : m : sig$ where C is the class of declaration and sig the method signature.

Not included here: `invokespecial` and `invokeinterface`.

Program structure

A **program** P is a collection of classes, organized into a tree-structured class hierarchy, denoted by classes(P).

To each **class** is associated a class name nameClass(c) and a set of methods defined in the class.

A **method** consists of

- its code (sequence of byte codes)
- a signature that specifies its argument types and return type
- max stack size and max number of registers used

A function Lookup(M, cl) will locate the most recent (re-)definition of method M starting from class cl and going upwards in the class hierarchy.

Operational semantics

Semantic domains.

$$
\begin{array}{rcll}
\text{Value} & = & \text{num } n & n \in \mathbb{N} \\
 & & \text{ref } r & r \in \text{Reference} \\
 & & \text{null} & \\
\text{Stack} & = & \text{Value}^* & \\
\text{LocalVar} & = & \text{Var} \to \text{Value} & \\
\text{Frame} & = & \text{ProgCount} \times \text{nameMethod} \times \text{LocalVar} \times \text{Stack} & \\
\text{CallStack} & = & \text{Frame}^* & \\
\text{Object} & = & \text{nameClass} \times (\text{FieldName} \to \text{Value}) & \\
\text{Heap} & = & \text{Reference} \to \text{Object}_\perp & \\
\text{State} & = & \text{Heap} \times \text{CallStack} &
\end{array}
$$

Operational semantics

Basic stack operations

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{nop}}{\langle\langle h, \langle m, pc, l, s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, s\rangle :: sf\rangle\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{const}\ c}{\langle\langle h, \langle m, pc, l, s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, c :: s\rangle :: sf\rangle\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{pop}}{\langle\langle h, \langle m, pc, l, v :: s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, s\rangle :: sf\rangle\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{dup}}{\langle\langle h, \langle m, pc, l, v :: s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, v :: v :: s\rangle :: sf\rangle\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{dup2}}{\langle\langle h, \langle m, pc, l, v_1 :: v_2 :: s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, v_1 :: v_2 :: v_1 :: v_2 :: s\rangle :: sf\rangle\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{swap}}{\langle\langle h, \langle m, pc, l, v_1 :: v_2 :: s\rangle :: sf\rangle\rangle \to \langle\langle h, \langle m, pc+1, l, v_2 :: v_1 :: s\rangle :: sf\rangle\rangle}$$

Operational semantics

Arithmetic and local variables.

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{numop}\ op}{\langle\!\langle h, \langle m, pc, l, n_1 :: n_2 :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc+1, l, [\![op]\!](n_1, n_2) :: s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{load}\ x}{\langle\!\langle h, \langle m, pc, l, s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc+1, l, l[x] :: s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{store}\ x}{\langle\!\langle h, \langle m, pc, l, v :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc+1, l[x \mapsto v], s\rangle :: sf\rangle\!\rangle}$$

Operational semantics

Conditionals and control transfer

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{if}\ pc' \qquad n = 0}{\langle\!\langle h, \langle m, pc, l, n :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc', l, s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{if}\ pc' \qquad n \neq 0}{\langle\!\langle h, \langle m, pc, l, n :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc + 1, l, s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m, pc) = \texttt{goto}\ pc'}{\langle\!\langle h, \langle m, pc, l, s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc', l, s\rangle :: sf\rangle\!\rangle}$$

Real Java byte code contains many more "if"s: `if_eq`, `if_ge`, `if_nonnull`, `if_acmpeq`,...

Operational semantics

Objects

$$\frac{\begin{array}{c}\text{instructionAt}_P(m,pc) = \texttt{new}\ cl \\ \exists c \in \text{classes}(P) \text{ with } \text{nameClass}(c) = cl \qquad (h', loc) = \text{newObject}(cl, h)\end{array}}{\langle\!\langle h, \langle m, pc, l, s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h', \langle m, pc+1, l, loc :: s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m,pc) = \texttt{putfield}\, f \qquad h(loc) = o \qquad o' = o[f \mapsto v]}{\langle\!\langle h, \langle m, pc, l, v :: loc :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h[loc \mapsto o'], \langle m, pc+1, l, s\rangle :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m,pc) = \texttt{getfield}\, f \qquad h(loc) = o}{\langle\!\langle h, \langle m, pc, l, loc :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m, pc+1, l, \text{fieldValue}(o,f) :: s\rangle :: sf\rangle\!\rangle}$$

Operational semantics

Method calls

$$\frac{\begin{array}{c}\text{instructionAt}_P(m,pc) = \texttt{invokevirtual}\ M \\ h(loc) = o \qquad m' = \text{Lookup}(M, class(o)) \\ f' = \langle m', 1, V, \varepsilon\rangle \qquad f'' = \langle m, pc, l, s\rangle\end{array}}{\langle\!\langle h, \langle m, pc, l, loc :: V :: s\rangle :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, f' :: f'' :: sf\rangle\!\rangle}$$

$$\frac{\text{instructionAt}_P(m,pc) = \texttt{return} \qquad f' = \langle m', pc', l', s'\rangle}{\langle\!\langle h, \langle m, pc, l, v :: s\rangle :: f' :: sf\rangle\!\rangle \rightarrow \langle\!\langle h, \langle m', pc'+1, l', v :: s'\rangle :: sf\rangle\!\rangle}$$

Other language features

Aspects of the language not dealt with here include:

- Visibility modifiers (`public`, `private`,`protected`,...)
- Multi-threading (`monitorenter`, `monitorexit`)
- Exceptions
- Constant pool
- Arrays
- Long integers

Java byte code verification

Java byte code verification is done at class loading time.

What is verified:

- structural correctness of class file,
- no jumps out of methods,
- size of operand stack at a program point never changes,
- operands get arguments of correct type,
- no pointer arithmetic or forging of references,
- objects and local variables are **initialized** before being used.
- `final` methods are not re-defined.

What is not verified

The Java byte code verifier (BCV) have fewer types that the Java type system

- no booleans,
- no generic types,
- no inner classes,
- interfaces

The BCV leaves a certain number of properties to be checked dynamically:

- null pointer dereferencing,
- array stores and indexing,
- division by zero,
- class casting,
- interface method calls

Typing as a data flow problem

Verification checks that arguments to operands are of correct type.

eg., adding a reference and an integer is not correct

The byte code verifier will

- abstract objects by their class
- not distinguish between different method calls

Compute for each program point

- a type for each local variable (register) and
- a type for the operand stack

This can be seen as a **data flow analysis** where data is replaced by types.

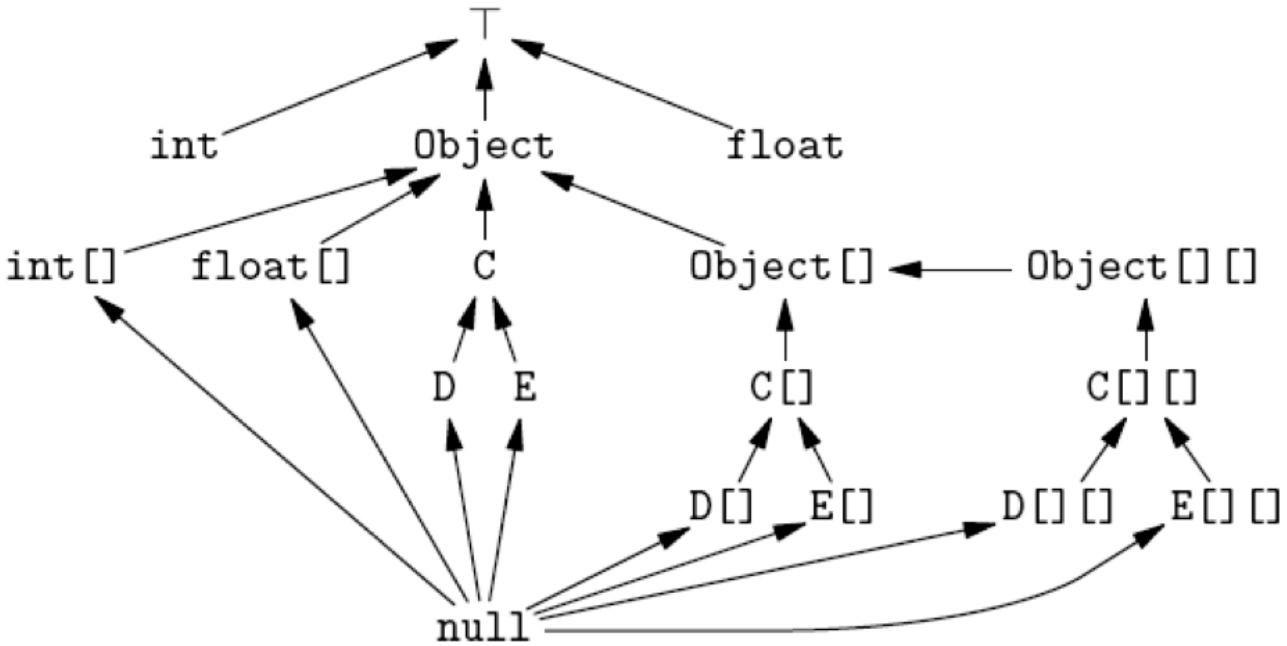
The abstract state at each program point is

$$R : Int \to Types \quad , \quad S : Types^*$$

The $\sqcup$-semi lattice of types

The class hierarchy is extended with an element $\top$ ("no type information"), and an element representing `null` values.

The sub-class relation `<:` is defined by the following Hasse diagram.



Least upper bound of two **stacks** is the pointwise extension of `<:` **provided** the stacks are of same size, otherwise $\top$.

Typing as data flow analysis

The **initial** state for the verification of a method:

At the first instruction,

- registers correspoding to paramters are given the type of the parameter,
- all other registers are set to type $\top$ (no information),
- and the stack is empty.

At all other instruction,

- the registers and stacks are undefined ($\bot$)

The abstract state is **propagated** from instruction to instruction following the control flow.

When control flows to an instruction from more than one program point, we take the **least upper bound** of the incoming states.

Typing as data flow analysis

```
iconst n : (S; R)  →  (int:S; R) if |S| < MaxStack
iadd : (int:int:S; R)  →  (int:S; R)
iload n : (S; R)  →  (int:S; R)
   if 0 < n < MaxReg and R(n) = int and |S|  < MaxStack
istore n : (int:S; R)  →  (S; R[n  →  int]) if 0 < n < MaxReg
aconst null : (S; R)  →  (null:S; R) if |S|  < MaxStack
aload n : (S; R)  →  (R(n):S; R)
   if 0  ⩽  n < MaxReg and R(n) <: Object and |S|  < MaxStack
astore n : (t:S; R)  →  (S; R[n  →  t])
   if 0 < n < MaxReg and t <: Object
getfield C:f:t : (t':S;R) →  (t:S; R) if t' <: C
putfield C:f:t : (t1:t2:S; R)  →  (S; R) if t1 <: t and t2
<: C
new C : (S;R)  →  (C:S;R)   if |S| < MaxStack
ifle j : (int:S;R)  →  (S;R)
goto j : (S;R)  →  (S;R)
invokevirtual C:m:sig : (tn : ... : t1 : t0 : S, R)  →  (s:S; R)
    if sig = s (s1, ... , sn) and
       t0 <: C,  ti <: si for i = 1 ... n
```

Here, <: is the sub-class relation.

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //
1: istore 1 //
2: iload 0 //
3: ifle 14 //
6: iload 1 //
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //
2: iload 0 //
3: ifle 14 //
6: iload 1 //
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //
3: ifle 14 //
6: iload 1 //
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //
6: iload 1 //
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //  R: int,int   S: [int]
10: iinc 0, -1 //
11: goto 2 //
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //  R: int,int   S: [int]
10: iinc 0, -1 //  R: int,int   S: []
11: goto 2 //
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //  R: int,int   S: [int]
10: iinc 0, -1 //  R: int,int   S: []
11: goto 2 //  R: int,int   S: []
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //    R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //  R: int,int   S: [int]
10: iinc 0, -1 //  R: int,int   S: []
11: goto 2 //  R: int,int   S: []
14: iload 1 //  R: int,int   S: []
15: ireturn //
```

Example: typing factorial

```
method static int factorial(int), 2 registers, 2 stack slots
0: iconst 1 //  R: int,⊤   S: []
1: istore 1 //  R: int,⊤   S: [int]
2: iload 0 //  R: int,int   S: []
3: ifle 14 //  R: int,int   S: [int]
6: iload 1 //  R: int,int   S: []
7: iload 0 //  R: int,int   S: [int]
8: imul //  R: int,int   S: [int :: int]
9: istore 1 //  R: int,int   S: [int]
10: iinc 0, -1 //  R: int,int   S: []
11: goto 2 //  R: int,int   S: []
14: iload 1 //  R: int,int   S: []
15: ireturn //  R: int,int   S: [int]
```

Exercise

Consider the following bytecode program

```
0:  ifle 6
1:  iload 1
2:  iconst 1
3:  iadd
4:  istore 2
5:  goto 9
6:  new Object
7:  astore 2
8:  goto 9
9:  iload 1
```

Show that this program is typable by the bytecode verifier with `MaxStack` $= 3$ and `MaxReg` $= 3$, starting with an operand stack [`int`] (one element of type `int`) and a register 1 of type `int`.

Correctness (informally)

Correctness of a program is expressed through a subject-reduction property.

Requires

- a well-typedness predicate on heaps h,
- a correctness relation between concrete and abstract states

$$\langle\langle h, \langle m, pc, l, s\rangle :: sf\rangle\rangle \sim (S, R).$$

Prove for each instruction i with semantic transition $\langle\langle h, s\rangle\rangle \to \langle\langle h', s'\rangle\rangle$ that

$$i : (S, R) \to (S', R') \text{ and } \langle\langle h, s\rangle\rangle \sim (S, R) \text{ implies } \langle\langle h', s'\rangle\rangle \sim (S', R').$$

Use this to conclude

- that each method respects its signature,
- and consequently, that all methods in a class hierarchy are well-typed.

Object creation and initialization

In Java byte code, objects of class `C` are created in two steps:

- they are *allocated* by `new`
- and *initialized* by a call to the constructor method `<init>` of the class `C`.

```
new C    // create uninitialized instance of class C
dup      // duplicate reference
....     // compute args to constructor
invokespecial C.<init>
```

The Java byte code language definition says:

> *It is an* **`error`** *to use an object (except assigning values to local fields) before all its constructors have been called.*

Similarly, it is an error to read from a local variable (a register) before it has been assigned a value (no `load` from a register of type $\top$).

Object initialization analysis

The byte code verifier performs two static analyses to check

1. that a constructor of a class always calls a constructor of the super-class.
2. that an object is not used before one of its constructors has been called,

For the first analysis:

- Add an extra element to the state, that is set to *initialized* when a constructor of the super class is called.

For the second analysis:

- Mark object types as "not yet completely initialized": $\overline{C}$
- Add an instruction number to distinguish between objects allocated at different program points: $\overline{C}_4$.
- Change status of all objects of a given type when exiting the constructor of that method.

Object initialization analysis

Example:

```
 0: new C      // stack type after: C̄0
 3: dup                          //    C̄0, C̄0
 4: new C                        //    C̄0, C̄0, C̄4
 7: dup                          //    C̄0, C̄0, C̄4, C̄4
 8: aconst_null                  //    C̄0, C̄0, C̄4, C̄4, null
 9: invokespecial C.<init>       //    C̄0, C̄0, C
12: invokespecial C.<init>       //    C
15: ...
```

Need restriction to get alias analysis right:

> *no stack element of type* $\overline{C}_p$ *when executing the instruction* `new C` *at program point* p.

Java interfaces

A Java interface specifes methods and fields but not their implementation.

Classes can be declared to implement one or more interfaces.

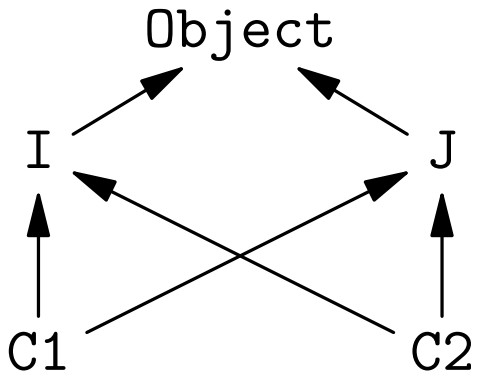
Interfaces can be used as declared types for variable.

```
interface PointInterface {
        void move(int dx, int dy);
}

public class C {
   PointInterface x;
...
   x.move(4,2);
}
```

The problem with interfaces

Classes can implement **several** interfaces

$\Rightarrow$ certain least upper bounds do not exist.

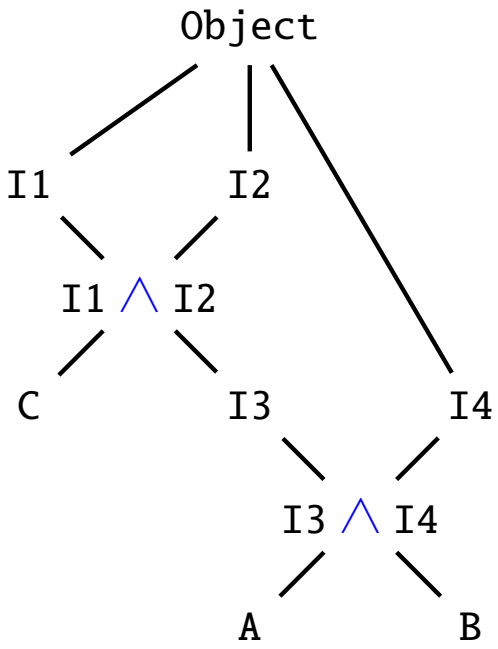


Sun's solution: treat interfaces as `Objects` and check at run-time whether an object implements an interfaces.

Another option: introduce intersection types to join several interfaces into one type.

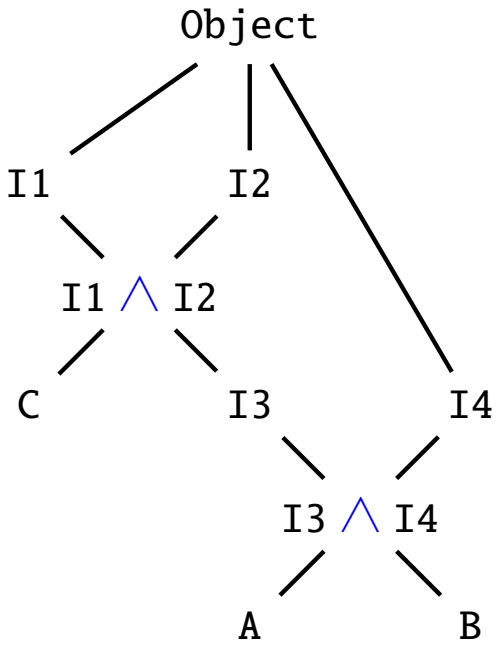
BCV with interfaces

- `C` implements `I1` and `I2`
- `I3` extends `I1` and `I2`
- `A` and `B` implements `I3` and `I4`
- Each `Ii` declares a method `mi`.



BCV with interfaces

- `C` implements `I1` and `I2`
- `I3` extends `I1` and `I2`
- `A` and `B` implements `I3` and `I4`
- Each `Ii` declares a method `mi`.



Exercise: find a type for `i`

```
void foo(boolean b) {
 if (b) {
   i:=new A() ;
 } else
   i:=new B();
...
i.m1();
i.m2();
}
```

BCV with interfaces

Standard BCV will not be able to infer a type for variable `i`.

Inference with intersection types will infer `I1` $\wedge$ `I2`

Notice: there is a weaker intersection-free type for `i`: `I3`

Prune typings to obtain stack maps without intersection types

- work for most (but not all!) byte codes
- Eclipse can be given stack maps w/o intersections

Verification of sub-routines

`jsr l` : jump to program point `l`, pushing the address of the follwing instruction

`ret n` : recover a return address from register `n` and jump to it.

Problems:
- sub-routine entries are *merge points*
- may limit precision

This complicates the byte code verification —
for a relatively **little gain**.

```
0: jsr 100 // register 0 undef
3: ...
50: iconst 0
51: istore 0
52: jsr 100  // register 0 int
55: iload 0
56: ireturn
...
100: astore 1
101: ...  // don't touch register 0
110: ret 1
```

Summary

Java byte code verification

- checks `.class` files when loaded into a Java virtual machine
- part of the secuuity architecture of Java
- verifies
  - well typing
  - object initialization
- formalized as a set of constraint rules

We have treated a subset of Java BC here

- basic sequential, procedural Java byte code
- interfaces and subroutines
- we did not deal with exceptions and arrays

Literature

Xavier Leroy: Bytecode verification: algorithms and formalizations, J. Automated Reasoning, 30(3–4), 2003.

Stephen N. Freund, John C. Mitchell: A Type System for the Java Bytecode Language and Verifier. J. Automated Reasoning 30(3-4), 2003.

Gerwin Klein, Tobias Nipkow: Verified bytecode verifiers, Theoretical Computer Science, 298(3), 2003.

G. Barthe, G. Dufay. A Tool-Assisted Framework for Certified Bytecode Verification. Proc. of FASE'04, LNCS 2984, 2004.