

VMCAI 2019 Winter School

Abstract Interpretation

Semantics, Verification, and Analysis

Patrick Cousot

pcousot@cs.nyu.edu cs.nyu.edu/~pcousot

Friday, 01/11/2019, 09:00 – 12:30

Content

1. Semantics (45 mn)
2. Abstraction (45 mn)
break (30mn)
3. Verification and proofs (45 mn)
4. Analysis (45 mn)
 - Numerical abstraction: see the VMCAI invited talk by Sylvie Putot (École polytechnique, France) on “Zonotopic abstract domains for numerical program analysis”
 - Symbolic abstraction: dependency analysis

Part 1

Semantics

Syntax

Context-free syntax of expressions

$x, y, \dots \in \mathcal{V}$

variables (\mathcal{V} not empty)

$A \in \mathcal{A} ::= 1 \mid x \mid A_1 - A_2$

arithmetic expressions

$B \in \mathcal{B} ::= A_1 < A_2 \mid B_1 \text{ nand } B_2$

boolean expressions

$E \in \mathcal{E} ::= A \mid B$

expressions

Context-free syntax of program statements

$S ::=$	statement $S \in \mathcal{S}$
$x = A ;$	assignment
$ \quad ;$	skip
$ \quad \text{if } (B) \ S$	conditional
$ \quad \text{if } (B) \ S \ \text{else } S$	
$ \quad \text{while } (B) \ S$	iteration
$ \quad \text{break ;}$	iteration break
$ \quad \{ S_l \}$	compound statement
$S_l ::= S_l \ S \mid \epsilon$	statement list
$P ::= S_l$	program $P \in \mathcal{P}$

Program labels

- To designate program points of program components, not part of the language
- Labels are unique
- $\text{at}[[S]]$ label at entry of statement S
- $\text{after}[[S]]$ label after exit of statement S
- $\text{escape}[[S]]$ is it possible to break out of the statement S ?
- $\text{break-to}[[S]]$ where to break (exit label of enclosing loop)
- $\text{in}[[S]]$ labels in statement S (excluding $\text{after}[[S]]$ and $\text{break-to}[[S]]$)
- $\text{labs}[[S]] \triangleq \text{in}[[S]] \cup \{\text{after}[[S]]\}$
- $\text{labx}[[S]] \triangleq \text{labs}[[S]] \cup (\text{escape}[[S]] \text{ ? } \{\text{break-to}[[S]]\} \text{ : } \emptyset)$

Axiomatic definition of program labelling

- We never define labels, just the properties they must satisfy
- Example $S \triangleq \text{if } (B) S_t \text{ else } S_f$:

$$\text{in}[[S]] \triangleq \text{at}[[S]] \cup \text{in}[[S_t]] \cup \text{in}[[S_f]]$$

$$\text{at}[[S]] \not\subseteq \text{in}[[S_t]] \cup \text{in}[[S_f]]$$

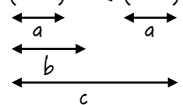
$$\text{in}[[S_t]] \cap \text{in}[[S_f]] = \emptyset$$

$$\text{after}[[S_t]] = \text{after}[[S_f]] = \text{after}[[S]]$$

Prefix trace semantics

Trace of a hand computation

Hand computation of

$$(1-1)-1 < (1-1)$$


is

$$a = 1 - 1$$

$$a = 0$$

$$b = a - 1$$

$$b = 0 - 1$$

$$b = -1$$

$$c = b < a$$

$$c = -1 < 0$$

$$c = tt$$

← read from

partial trace

finite trace

Prefix trace

- A **prefix trace** is a finite observation of the program execution from entry
- A trace is a finite sequence of **labels separated by actions** (no memory state)

$$\ell_1 \xrightarrow{a_1} \ell_2 \xrightarrow{a_3} \ell_3 \xrightarrow{a_3} \ell_4 \xrightarrow{a_4} \ell_3 \xrightarrow{a_5} \ell_6 \dots$$

- **labels** ℓ_i : next action to be executed
- **actions** a_i : records the computation done by a program step

Example of prefix trace

- default initialization to 0

ℓ_1 $x = x + 1$; (4.4)
while ℓ_2 (tt) {
 ℓ_3 $x = x + 1$;
 if ℓ_4 ($x > 2$) ℓ_5 break ; } ℓ_6 ; ℓ_7

- $\ell_1 \xrightarrow{x = x + 1 = 1} \ell_2 \xrightarrow{\text{tt}} \ell_3 \xrightarrow{x = x + 1 = 2} \ell_4 \xrightarrow{\neg(x > 2)} \ell_2 \xrightarrow{\text{tt}} \ell_3$ (6.1)
- $\ell_1 \xrightarrow{x = x + 1 = 1} \ell_2 \xrightarrow{\text{tt}} \ell_3 \xrightarrow{x = x + 1 = 2} \ell_4 \xrightarrow{\neg(x > 2)} \ell_2 \xrightarrow{\text{tt}} \ell_3 \xrightarrow{x = x + 1 = 3}$
 $\ell_4 \xrightarrow{x > 2} \ell_5 \xrightarrow{\text{break}} \ell_6 \xrightarrow{\text{skip}} \ell_7$

Values of variables

- Go back in the past to look for the last recorded assigned value (or 0 at initialization)

$$\begin{aligned}\rho(\pi^\ell \xrightarrow{x = E = v} \ell')_x &\triangleq v \\ \rho(\pi^\ell \xrightarrow{\dots} \ell')_x &\triangleq \rho(\pi^\ell) \quad \text{otherwise} \\ \rho(\ell)_x &\triangleq 0\end{aligned}\tag{6.2}$$

Prefix trace semantics

- Given a trace π_0 arriving at $\llbracket S \rrbracket$,
the prefix trace semantics $\mathcal{S}^* \llbracket S \rrbracket$ of S specifies
the trace π_1 of the execution of S from $\text{at} \llbracket S \rrbracket$ with initial values defined by π_0

$$\xrightarrow{\pi_0} \text{at} \llbracket S \rrbracket \xrightarrow{\pi_1} \ell$$

$\underbrace{\text{at} \llbracket S \rrbracket \xrightarrow{\pi_1} \ell}_{\in \mathcal{S}^* \llbracket S \rrbracket (\pi_0 \text{at} \llbracket S \rrbracket)}$

Structural rule-based definition of the prefix trace semantics

Structural prefix trace semantics at a statement

Prefix trace at a statement S

$$\blacksquare \quad \frac{}{\text{at}[\![S]\!] \in \widehat{\mathcal{S}}^*[\![S]\!](\pi_1 \text{at}[\![S]\!])} \quad (6.7)$$

A prefix continuation of the traces $\pi_1 \text{at}[\![S]\!]$ arriving at a program, statement or statement list S can be reduced to the program point $\text{at}[\![S]\!]$ at this program, statement or statement list S .

Semantics of arithmetic expressions

- An environment $\rho \in \mathbb{E}_V$ where $\mathbb{E}_V \triangleq V \rightarrow \mathbb{Z}$ is a function ρ mapping a variable x to its value $\rho(x)$ in the set \mathbb{Z} of all mathematical integers.
- Semantics of arithmetic expressions:

$$\begin{aligned}\mathcal{A} \llbracket 1 \rrbracket \rho &\triangleq 1 \\ \mathcal{A} \llbracket x \rrbracket \rho &\triangleq \rho(x) \\ \mathcal{A} \llbracket A_1 - A_2 \rrbracket \rho &\triangleq \mathcal{A} \llbracket A_1 \rrbracket \rho - \mathcal{A} \llbracket A_2 \rrbracket \rho\end{aligned}\tag{3.4}$$

Structural prefix trace semantics of an assignment statement

Prefix traces of an assignment statement $S ::= \ell \ x = A \ ;$

$$\blacksquare \quad \frac{v = \mathcal{A} \llbracket A \rrbracket \rho(\pi^\ell)}{\ell \xrightarrow{x = A = v} \text{after} \llbracket S \rrbracket \in \widehat{\mathcal{S}}^* \llbracket S \rrbracket (\pi^\ell)}$$

A prefix finite trace of an assignment $\ell \ x = E \ ;$ continuing some trace π^ℓ is ℓ followed by the event $x = v$ where v is the last value of x previously assigned to x on π^ℓ (otherwise initialized to 0) and finishing at the label $\text{after} \llbracket S \rrbracket$ after the assignment.

Structural prefix trace semantics of a conditional statement

Prefix traces of a conditional statement $S ::= \text{if } \ell \text{ (B)} S_t$

$$\blacksquare \frac{\mathcal{R}[\![B]\!] \rho(\pi_1^\ell) = \text{ff}}{\ell \xrightarrow{\neg(B)} \text{after}[\![S]\!] \in \widehat{\mathcal{S}}^*[\![S]\!](\pi_1^\ell)} \quad (6.14)$$

$$\blacksquare \frac{\mathcal{R}[\![B]\!] \rho(\pi_1^\ell) = \text{tt}, \quad \pi_2 \in \widehat{\mathcal{S}}^*[\![S_t]\!](\pi_1^\ell \xrightarrow{B} \text{at}[\![S_t]\!])}{\ell \xrightarrow{B} \text{at}[\![S_t]\!] \frown \pi_2 \in \widehat{\mathcal{S}}^*[\![S]\!](\pi_1^\ell)} \quad (6.15)$$

\frown is trace concatenation

Structural prefix trace semantics of an empty statement list

Prefix traces of an empty statement list $sl ::= \epsilon$

$$\blacksquare \frac{}{\text{at}[\![sl]\!] \in \widehat{\mathcal{F}}^*[\![sl]\!](\pi \text{at}[\![sl]\!])} \quad (6.11)$$

- A prefix/maximal trace π of the empty statement list ϵ continuing some trace is reduced to the program label $\text{at}[\![sl]\!]$ at that empty statement.
- This case is redundant and already covered by (6.7).

Structural prefix trace semantics of a statement list

Prefix traces of a statement list $sl ::= sl' \ S$

$$\begin{array}{c} \blacksquare \quad \frac{\pi_2 \in \widehat{\mathcal{F}}^* \llbracket sl' \rrbracket (\pi_1)}{\pi_2 \in \widehat{\mathcal{F}}^* \llbracket sl \rrbracket (\pi_1)} \end{array} \quad (6.9)$$

$$\begin{array}{c} \blacksquare \quad \frac{\pi_2 \in \widehat{\mathcal{F}}^+ \llbracket sl' \rrbracket (\pi_1), \quad \pi_3 \in \widehat{\mathcal{F}}^* \llbracket S \rrbracket (\pi_1 \frown \pi_2)}{\pi_2 \frown \pi_3 \in \widehat{\mathcal{F}}^* \llbracket sl \rrbracket (\pi_1)} \end{array} \quad (6.10)$$

A prefix trace of $sl' \ S$ continuing an initial trace π_1 can be a prefix trace of sl' or a finite maximal trace of sl' followed by a prefix trace of S .

Structural prefix trace semantics of an iteration statement

Prefix traces of an iteration statement $S ::= \text{while}^\ell(B) S_b$

$$\blacksquare \frac{}{\ell \in \widehat{\mathcal{S}}^*[[S]](\pi_1^\ell)} \quad (6.20)$$

$$\blacksquare \frac{\ell\pi_2^\ell \in \widehat{\mathcal{S}}^*[[S]](\pi_1^\ell), \quad \mathcal{B}[[B]]\rho(\pi_1^\ell\pi_2^\ell) = \text{ff}}{\ell\pi_2^\ell \xrightarrow{\neg(B)} \text{after}[[S]] \in \widehat{\mathcal{S}}^*[[S]](\pi_1^\ell)} \quad (6.21)$$

$$\blacksquare \frac{\begin{array}{l} \ell\pi_2^\ell \in \widehat{\mathcal{S}}^*[[S]](\pi_1^\ell), \quad \mathcal{B}[[B]]\rho(\pi_1^\ell\pi_2^\ell) = \text{tt}, \\ \pi_3 \in \widehat{\mathcal{S}}^*[[S_b]](\pi_1^\ell\pi_2^\ell \xrightarrow{B} \text{at}[[S_b]]) \end{array}}{\ell\pi_2^\ell \xrightarrow{B} \text{at}[[S_b]] \frown \pi_3 \in \widehat{\mathcal{S}}^*[[S]](\pi_1^\ell)} \quad (6.22)$$

This is a forward, left recursive definition where $n + 1$ iterations are n iterations followed by one more iteration.

Structural prefix trace semantics of an iteration statement

Prefix traces of an iteration statement $S ::= \text{while } \ell(B) S_b$

$$\frac{}{\ell \in \widehat{\mathcal{S}}^*[[S]](\pi_1 \ell)} \quad (6.20)$$

$$\frac{\ell \pi_2 \ell \in \boxed{\widehat{\mathcal{S}}^*[[S]](\pi_1 \ell)}, \quad \mathcal{B}[[B]]\rho(\pi_1 \ell \pi_2 \ell) = \text{ff}}{\ell \pi_2 \ell \xrightarrow{\neg(B)} \text{after}[[S]] \in \boxed{\widehat{\mathcal{S}}^*[[S]](\pi_1 \ell)}} \quad (6.21)$$

$$\frac{\begin{array}{l} \ell \pi_2 \ell \in \boxed{\widehat{\mathcal{S}}^*[[S]](\pi_1 \ell)}, \quad \mathcal{B}[[B]]\rho(\pi_1 \ell \pi_2 \ell) = \text{tt}, \\ \pi_3 \in \widehat{\mathcal{S}}^*[[S_b]](\pi_1 \ell \pi_2 \ell \xrightarrow{B} \text{at}[[S_b]]) \end{array}}{\ell \pi_2 \ell \xrightarrow{B} \text{at}[[S_b]] \cdot \pi_3 \in \boxed{\widehat{\mathcal{S}}^*[[S]](\pi_1 \ell)}} \quad (6.22)$$

The definition is **structural** (depends on the already defined semantics of sub-components) and **recursive** (depends on itself) \rightarrow might not be well-defined.

Structural prefix trace semantics of a break statement

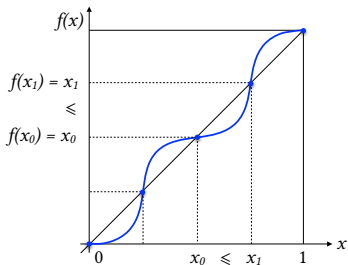
Prefix traces of a break statement $S ::= \ell \text{ break ;}$

$$\blacksquare \quad \frac{}{\ell \xrightarrow{\text{break}} \text{break-to}[[S]] \in \widehat{\mathcal{S}}^*[[S]](\pi^\ell)} \quad (6.25)$$

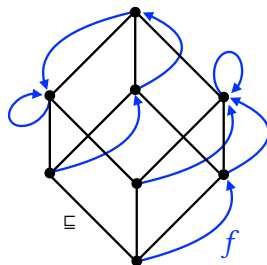
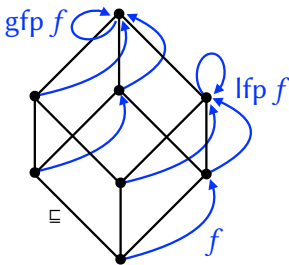
A prefix finite trace of a break $\ell \text{ break ;}$ continuing some initial trace π^ℓ is the trace ℓ followed by the **break ;** event and ending at the break label $\text{break-to}[[S]]$ (which is the exit label of the closest enclosing iteration loop or else the program exit).

Structural fixpoint definition of the prefix trace semantics

Examples of fixpoints x such that $f(x) = x$



increasing function f



non-increasing function f

- As shown by Alfred Tarski, an increasing function on a complete lattice has at least one fixpoint and has a least one.

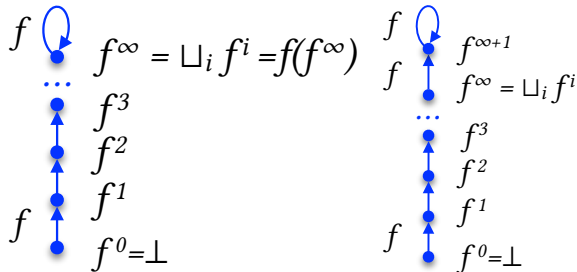
Tarski fixpoint theorem

Theorem (13.5, Tarski fixpoint theorem) An increasing function $f \in L \xrightarrow{\sqsubseteq} L$ on a complete lattice $\langle L, \sqsubseteq, \perp, \top, \sqcap, \sqcup \rangle$ has a least fixpoint $\text{lfp}^{\sqsubseteq} f = \bigsqcap \{x \in L \mid f(x) \sqsubseteq x\}$.

Tarski iterative fixpoint theorem

Theorem (13.14, Tarski iterative fixpoint)

- Let $f \in P \rightarrow P$ be an increasing function on a poset $\langle P, \sqsubseteq, \perp \rangle$ with infimum \perp .
- Define the iterates of f to be the sequence $f^0 = \perp$ and $f^{n+1} = f(f^n)$ for $n \in \mathbb{N}$.
- Assume that the least upper bound $\bigsqcup \{f^n \mid n \in \mathbb{N}\}$ exists and $f(\bigsqcup \{f^n \mid n \in \mathbb{N}\}) = \bigsqcup \{f(f^n) \mid n \in \mathbb{N}\}$.
- Then f has a least fixpoint $\text{lfp}^\sqsubseteq f = \bigsqcup \{f^n \mid n \in \mathbb{N}\}$.



Fixpoint prefix trace semantics of an assignment statement

Fixpoint prefix trace semantics of an assignment statement $S ::= {}^{\ell} x = E ;$

$$\widehat{\mathcal{S}}^* \llbracket S \rrbracket (\pi^{\ell}) = \{\ell\} \cup \{\ell \xrightarrow{x = E = v} \text{after} \llbracket S \rrbracket \mid v = \mathcal{G} \llbracket E \rrbracket \rho(\pi^{\ell})\}$$

- Example of basic case

Fixpoint prefix trace semantics of a statement list

Prefix traces of a statement list $sl ::= sl' \ S$

$$\widehat{\mathcal{F}}^*[[sl]](\pi_1) = \widehat{\mathcal{F}}^*[[sl']](\pi_1) \cup \{\pi_2 \frown \pi_3 \mid \pi_2 \in \widehat{\mathcal{F}}^+[[sl']](\pi_1) \wedge \pi_3 \in \widehat{\mathcal{F}}^*[[S]](\pi_1 \frown \pi_2)\} \quad (15.2)$$

- $\widehat{\mathcal{F}}^+[[sl']]$ contains the **finite maximal traces** of $\widehat{\mathcal{F}}^*[[sl']]$
- Example of inductive case ($\widehat{\mathcal{F}}^*[[sl]]$ defined in terms of $\widehat{\mathcal{F}}^+[[sl']]$ and $\widehat{\mathcal{F}}^*[[S]]$ with $sl' \triangleleft sl$ and $S \triangleleft sl$ where \triangleleft is the strict component relation)

Fixpoint prefix trace semantics of an iteration

Prefix traces of an iteration statement $S ::= \text{while } \ell \text{ (B)} S_b$

$$\mathcal{S}^*[\text{while } \ell \text{ (B)} S_b] = \text{lfp}^{\subseteq} \mathcal{F}^*[\text{while } \ell \text{ (B)} S_b] \quad (15.3)$$

$$\mathcal{F}^*[\text{while } \ell \text{ (B)} S_b](X)(\pi_1^{\ell'}) \triangleq \emptyset \quad \text{when } \ell' \neq \ell$$

$$\mathcal{F}^*[\text{while } \ell \text{ (B)} S_b](X)(\pi_1^{\ell}) \triangleq \{\ell\} \quad (a)$$

$$\begin{aligned} \cup \{ \ell' \pi_2^{\ell'} \xrightarrow{\neg(B)} \text{after}[S] \mid \ell' \pi_2^{\ell'} \in X(\pi_1^{\ell'}) \wedge \\ \mathcal{B}[B] \rho(\pi_1^{\ell'} \pi_2^{\ell'}) = \text{ff} \wedge \ell' = \ell \} \end{aligned} \quad (b)$$

$$\begin{aligned} \cup \{ \ell' \pi_2^{\ell'} \xrightarrow{B} \text{at}[S_b] \frown \pi_3 \mid \ell' \pi_2^{\ell'} \in X(\pi_1^{\ell'}) \wedge \mathcal{B}[B] \rho(\pi_1^{\ell'} \pi_2^{\ell'}) = \text{tt} \\ \wedge \pi_3 \in \mathcal{S}^*[S_b](\pi_1^{\ell'} \pi_2^{\ell'} \xrightarrow{B} \text{at}[S_b]) \wedge \ell' = \ell \} \end{aligned} \quad (c)$$

- Example of inductive fixpoint case
 - inductive: $\mathcal{S}^*[\text{while}^\ell(B) S_b]$ defined in terms of $\mathcal{S}^*[S_b]$ with $S_b \triangleleft \text{while}^\ell(B) S_b$
 - fixpoint: $\mathcal{S}^*[\text{while}^\ell(B) S_b]$ recursively defined in terms of itself ($n + 1$ iterations are 1 iteration plus n iterations)

Maximal trace semantics

Maximal trace semantics, informally

- The maximal trace semantics $\mathcal{S}^{+\infty}[[s]] = \mathcal{S}^+[[s]] \cup \mathcal{S}^\infty[[s]]$ is derived from the prefix trace semantics $\mathcal{S}^*[[s]]$ by
 - keeping the longest finite traces $\mathcal{S}^+[[s]]$, and
 - passing to the limit $\mathcal{S}^\infty[[s]]$ of prefix-closed traces for infinite traces.

Finite maximal trace semantics

- $\mathcal{S}^+[\![S]\!](\pi_1 \text{at}[\![S]\!]) \triangleq \{\pi_2^\ell \in \mathcal{S}^*[\![S]\!](\pi_1 \text{at}[\![S]\!]) \mid \ell = \text{after}[\![S]\!]\}$
- $\mathcal{S}^+[\![S]\!](\pi_1^\ell) = \emptyset$ when $\ell \neq \text{at}[\![S]\!]$
- $\mathcal{S}^+[\![S]\!](\pi_1 \text{at}[\![S]\!])$ is the set of maximal finite traces $\text{at}[\![S]\!]\pi_2 \text{after}[\![S]\!]$ of S continuing the trace $\pi_1 \text{at}[\![S]\!]$ and reaching $\text{after}[\![S]\!]$.

Prefixes of a trace

- If $\pi = \ell_0 \xrightarrow{e_0} \dots \ell_i \xrightarrow{e_i} \dots \ell_n$ is a finite trace then its prefix $\pi[0..p]$ at p is
 - π when $p \geq n$
 - $\ell_0 \xrightarrow{e_0} \dots \ell_j \xrightarrow{e_j} \dots \ell_p$ when $0 \leq p \leq n$.
- If $\pi = \ell_0 \xrightarrow{e_0} \dots \ell_i \xrightarrow{e_i} \dots$ is an infinite trace then its prefix $\pi[0..p]$ at p is $\ell_0 \xrightarrow{e_0} \dots \ell_j \xrightarrow{e_j} \dots \ell_p$.

Limit of prefix traces

- The limit $\lim \mathcal{T}$ of a set of traces \mathcal{T} is the set of infinite traces which prefixes can be extended to a trace in \mathcal{T} .

$$\lim \mathcal{T} \triangleq \{ \pi \in \mathbb{T}^\infty \mid \forall n \in \mathbb{N} . \exists p \geq n . \pi[0..p] \in \mathcal{T} \} .$$

- Let S be an iteration. $\langle \pi, \pi' \rangle \in \lim \mathcal{S}^* \llbracket S \rrbracket$ where π' is infinite if and only if, whenever we take a prefix $\pi'[0..n]$ of π' , it is a possible finite observation of the execution of S and so belongs to the prefix trace semantics $\langle \pi, \pi'[0..n] \rangle \in \mathcal{S}^* \llbracket S \rrbracket$.

Infinite maximal trace semantics

$$\mathcal{S}^\infty \llbracket s \rrbracket \triangleq \lim(\mathcal{S}^* \llbracket s \rrbracket)$$

Memory abstraction

Memory abstraction

- Abstraction from traces $\pi \in \mathbb{T}^+$ to environments $\rho \in \mathbb{E}_{\mathbb{V}} \triangleq \mathbb{V} \rightarrow \mathbb{V}$ mapping variables $x \in \mathbb{V}$ to their value $\rho(x) \in \mathbb{V}$
- $\alpha(\pi) = \rho(\pi)$

where

$$\begin{aligned} \rho(\pi^\ell \xrightarrow{x = E = v} \ell')x &\triangleq v \\ \rho(\pi^\ell \xrightarrow{\dots} \ell')x &\triangleq \rho(\pi^\ell) \quad \text{otherwise} \\ \rho(\ell)x &\triangleq 0 \end{aligned} \tag{6.2}$$

Properties

Formal property

- A property is the set of elements that satisfy this property.
- Examples:
 - $\{2k + 1 \mid k \in \mathbb{N}\}$ is the property “to be an odd natural”
 - $\{2k \mid k \in \mathbb{Z}\}$ is the property “to be an even integer”
- Formally:
 - \mathcal{E} is a set of entities
 - A property of these entities is an element of $\wp(\mathcal{E})$
 - Examples:
 - \emptyset is false (ff)
 - \mathcal{E} is true (tt)
 - $e \in P, P \in \wp(\mathcal{E})$ means “ e has property P ”
 - $P \subseteq P'$ is implication \Rightarrow (P is *stronger* than P' , P' is *weaker* than P)

Program property

- **Syntactic point of view**: a program property is the set of all programs which have this property (e.g. Rice theorem)
- **Semantic point of view**: a program property is the set of all semantic of programs which have this property.
- By **[program] property**, we mean the semantic point of view.
- A program semantics is a set of traces (in $\wp(\mathbb{T}^+)$) so a program property is a set of sets of traces (in $\wp(\wp(\mathbb{T}^+))$)¹

¹sometimes called “hyperproperties”

The complete (boolean) lattice of formal properties

$$\langle \wp(\mathcal{E}), \subseteq, \emptyset, \mathcal{E}, \cup, \cap, \neg \rangle$$

- $\wp(\mathcal{E})$ properties of entities belonging to \mathcal{E}
- \subseteq implication
- \emptyset false
- \mathcal{E} true
- \cup disjunction, or
- \cap conjunction, and
- \neg negation, $\neg P \triangleq \mathcal{E} \setminus P$

(the definition of “complete lattice” is forthcoming)

Posets and complete lattices

- A *poset* $\langle \mathbb{P}, \sqsubseteq \rangle$ is a set equipped with a binary relation \sqsubseteq which is (forall $x, y, z \in \mathbb{P}$)
 - *reflexive*: $x \sqsubseteq x$
 - *antisymmetric*: $x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$
 - *transitive*: $x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$
- A subset $S \in \wp(\mathbb{P})$ has a *least upper bound* (denoted $\sqcup S$) if and only if
 - $\sqcup S \in \mathbb{P}$
 - $\forall x \in S . x \sqsubseteq \sqcup S$
 - $\forall x \in S . x \sqsubseteq u \Rightarrow \sqcup S \sqsubseteq u$
- A *complete lattice* is a poset $\langle \mathbb{P}, \sqsubseteq \rangle$ in which any subset $S \in \wp(\mathbb{P})$ has a lub/join $\sqcup S$ (not only the finite ones).

Collecting semantics

Collecting semantics

- The strongest semantic property of program P

$$\mathcal{S}^c[P] \triangleq \{\mathcal{S}^*[P]\} . \quad (8.5)$$

- Program P has property $P \in \wp(\wp(\mathbb{T}^{+\infty}))$ is
 - $\mathcal{S}^*[P] \in P$, or equivalently
 - $\{\mathcal{S}^*[P]\} \subseteq P$ i.e. P is implied by the collecting semantics of program P .
- So we can use implication $\subseteq (\Rightarrow)$ instead of \in (with no direct equivalent for predicates in logic).
- Program verification $\{\mathcal{S}^*[P]\} \subseteq P$ is **undecidable** (Rice theorem)

Bibliography on semantics

References I

- Cousot, Patrick (2002). “Constructive design of a hierarchy of semantics of a transition system by abstract interpretation”. *Theor. Comput. Sci.* 277.1-2, pp. 47–103 (5, 10, 3, 16).
- Cousot, Patrick and Radhia Cousot (1979). “Constructive Versions of Tarski’s Fixed Point Theorems”. *Pacific Journal of Mathematics* 81.1, pp. 43–57 (7, 5, 3).
- (1992). “Inductive Definitions, Semantics and Abstract Interpretation”. In: *POPL*. ACM Press, pp. 83–94 (5, 10).
 - (1995). “Compositional and Inductive Semantic Definitions in Fixpoint, Equational, Constraint, Closure-condition, Rule-based and Game-Theoretic Form”. In: *CAV*. Vol. 939. *Lecture Notes in Computer Science*. Springer, pp. 293–308 (10, 29).
 - (2009). “Bi-inductive structural semantics”. *Inf. Comput.* 207.2, pp. 258–283 (5, 10, 3, 8).
- Plotkin, Gordon D. (2004). “A structural approach to operational semantics”. *J. Log. Algebr. Program.* 60-61, pp. 17–139 (10).

The End of Part 1

Part 2

Abstraction

Abstraction

- We formalize the *abstraction and approximation of program properties*
- We show how a structural rule-based/fixpoint *abstract semantics* can be derived from the collecting semantics by *calculational design*.

Informal introduction to abstraction

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$
- The decoding function $\gamma \in A \rightarrow \mathcal{A}$ is called the *concretization function*

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$
- The decoding function $\gamma \in A \rightarrow \mathcal{A}$ is called the *concretization function*
- Make proofs using abstract properties only

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$
- The decoding function $\gamma \in A \rightarrow \mathcal{A}$ is called the *concretization function*
- Make proofs using abstract properties only
- So *any concrete property must be over-approximated by a abstract property in*
 $\mathcal{A} = \gamma(A)$

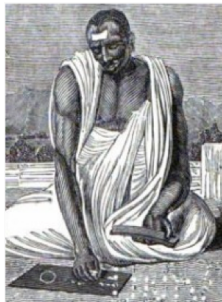
Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$
- The decoding function $\gamma \in A \rightarrow \mathcal{A}$ is called the *concretization function*
- Make proofs using abstract properties only
- So *any concrete property must be over-approximated by a abstract property in*
 $\mathcal{A} = \gamma(A)$
- If the abstract proof succeeds, it is valid in the concrete (*soundness*)

Abstraction, informally

- Let be $\langle \wp(\mathbb{C}), \subseteq \rangle$ be properties of entities (so called the *concrete domain*)
- Consider a subset $\mathcal{A} \subseteq \wp(\mathbb{C})$ of *properties of interest* $\langle \mathcal{A}, \subseteq \rangle$
- Encode these properties of interest in an *abstract domain* $\langle A, \sqsubseteq \rangle$
- The decoding function $\gamma \in A \rightarrow \mathcal{A}$ is called the *concretization function*
- Make proofs using abstract properties only
- So *any concrete property must be over-approximated by a abstract property in*
 $\mathcal{A} = \gamma(A)$
- If the abstract proof succeeds, it is valid in the concrete (*soundness*)
- If the abstract proof fails, you missed some property in $\wp(\mathbb{C}) \setminus \mathcal{A}$ which is essential in the concrete proof (*incompleteness*)

Brahmagupta



- Brahmagupta (born c. 598, died after 665) was an Indian mathematician and astronomer;
- Invented the rule of signs (including to compute with zero);
- We explain his rule of sign as an abstract interpretation;
- Probably the very first example of abstract interpretation.

Structural collecting semantics

- Semantics

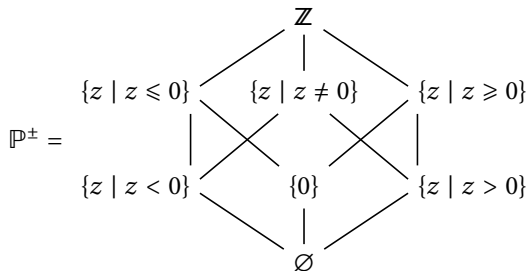
$$\begin{aligned}\mathcal{A} \llbracket A \rrbracket &\in (\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z} \\ \mathcal{A} \llbracket 1 \rrbracket \rho &\triangleq 1 \\ \mathcal{A} \llbracket x \rrbracket \rho &\triangleq \rho(x) \\ \mathcal{A} \llbracket A_1 - A_2 \rrbracket \rho &\triangleq \mathcal{A} \llbracket A_1 \rrbracket \rho - \mathcal{A} \llbracket A_2 \rrbracket \rho\end{aligned}$$

- Collecting semantics

$$\begin{aligned}\mathcal{S}^c \llbracket A \rrbracket &\in \wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z}) \\ \mathcal{S}^c \llbracket 1 \rrbracket &= \{\lambda \rho \in \cdot. 1\} \\ \mathcal{S}^c \llbracket x \rrbracket &= \{\lambda \rho \in (\mathcal{V} \rightarrow \mathbb{Z}). \rho(x)\} \\ \mathcal{S}^c \llbracket A_1 - A_2 \rrbracket &= \{\lambda \rho \in (\mathcal{V} \rightarrow \mathbb{Z}). f_1(\rho) - f_2(\rho) \mid f_1 \in \mathcal{S}^c \llbracket A_1 \rrbracket \wedge f_2 \in \mathcal{S}^c \llbracket A_2 \rrbracket\}\end{aligned}$$

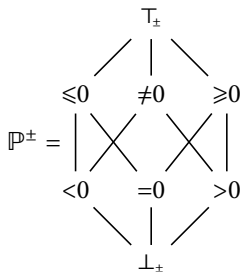
Sign abstraction

Sign property (of an individual variable)



Example of **Hasse diagram**.

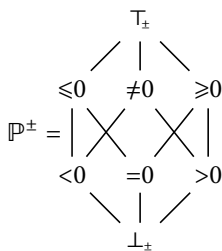
Encoding of sign properties (of an individual variable)



Concretization function:

$$\begin{array}{ll}
 \gamma_{\pm}(\perp_{\pm}) & \triangleq \emptyset \\
 \gamma_{\pm}(<0) & \triangleq \{z \mid z < 0\} \\
 \gamma_{\pm}(=0) & \triangleq \{0\} \\
 \gamma_{\pm}(>0) & \triangleq \{z \mid z > 0\} \\
 \gamma_{\pm}(\leq 0) & \triangleq \{z \mid z \leq 0\} \\
 \gamma_{\pm}(\neq 0) & \triangleq \{z \mid z \neq 0\} \\
 \gamma_{\pm}(\geq 0) & \triangleq \{z \mid z \geq 0\} \\
 \gamma_{\pm}(\top_{\pm}) & \triangleq \mathbb{Z}
 \end{array}$$

Encoding of sign properties (of an individual variable)



\sqsubseteq is the partial order in \mathbb{P}^\pm

\sqcup is the least upper bound in \mathbb{P}^\pm

e.g. $\sqcup\{\leq 0, \neq 0\} = \top_\pm$, $\sqcup \emptyset = \perp_\pm$

\sqcap is the greatest lower bound in \mathbb{P}^\pm

e.g. $\sqcap\{\leq 0, \neq 0\} = < 0$, $\sqcap \emptyset = \top_\pm$

Abstraction function: $\alpha_\pm(P) \triangleq \left(\begin{array}{l} P \subseteq \emptyset \text{ ? } \perp_\pm \\ P \subseteq \{z \mid z < 0\} \text{ ? } < 0 \\ P \subseteq \{0\} \text{ ? } = 0 \\ P \subseteq \{z \mid z > 0\} \text{ ? } > 0 \\ P \subseteq \{z \mid z \leq 0\} \text{ ? } \leq 0 \\ P \subseteq \{z \mid z \neq 0\} \text{ ? } \neq 0 \\ P \subseteq \{z \mid z \geq 0\} \text{ ? } \geq 0 \\ \text{? } \top_\pm \end{array} \right) \quad (3.28)$

Galois connection

- The pair $\langle \alpha_{\pm}, \gamma_{\pm} \rangle$ of functions satisfies $\alpha_{\pm}(P) \sqsubseteq Q \Leftrightarrow P \subseteq \gamma_{\pm}(Q)$

$$\alpha_{\pm}(P) \sqsubseteq Q$$

$$\Leftrightarrow \alpha_{\pm}(P) \sqsubseteq \neq 0$$

{in case $Q = \neq 0$, other cases are similar}

$$\Leftrightarrow \alpha_{\pm}(P) \in \{\perp_{\pm}, <0, \neq 0, >0\}$$

{def. }

$$\Leftrightarrow P \subseteq \emptyset \vee P \subseteq \{z \mid z < 0\} \vee P \subseteq \{z \mid z > 0\} \vee P \subseteq \{z \mid z \neq 0\}$$

{def. α_{\pm} }

$$\Leftrightarrow P \subseteq \{z \mid z \neq 0\}$$

{def. \subseteq }

$$\Leftrightarrow P \subseteq \gamma_{\pm}(\neq 0)$$

{def. γ_{\pm} }

$$\Leftrightarrow P \subseteq \gamma_{\pm}(Q)$$

{case $Q = \neq 0$ }

- This is the definition of a **Galois connection**

- We write $\langle \wp(\mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{P}^{\pm}, \sqsubseteq \rangle$

- This will be further generalized.

Sign abstract semantics

$$\mathcal{S}[\mathbf{A}] \in (\mathcal{V} \rightarrow \mathbb{P}^\pm) \rightarrow \mathbb{P}^\pm$$

$$\mathcal{S}[\mathbf{1}]P \triangleq >0$$

$$\mathcal{S}[\mathbf{x}]P \triangleq P(\mathbf{x})$$

$$\mathcal{S}[\mathbf{A}_1 - \mathbf{A}_2]P \triangleq \mathcal{S}[\mathbf{A}_1]P \dot{-}_\pm \mathcal{S}[\mathbf{A}_2]P$$

$x \dot{-}_\pm y$		y							
		\perp_\pm	<0	$=0$	>0	≤ 0	$\neq 0$	≥ 0	\top_\pm
x	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm	\perp_\pm
	<0	\perp_\pm	\top_\pm	<0	<0	\top_\pm	\top_\pm	<0	\top_\pm
	$=0$	\perp_\pm	>0	$=0$	<0	≥ 0	$\neq 0$	≤ 0	\top_\pm
	>0	\perp_\pm	>0	>0	\top_\pm	>0	\top_\pm	\top_\pm	\top_\pm
	≤ 0	\perp_\pm	>0	≤ 0	\top_\pm	\top_\pm	\top_\pm	≤ 0	\top_\pm
	$\neq 0$	\perp_\pm	\top_\pm	$\neq 0$	\top_\pm	\top_\pm	\top_\pm	\top_\pm	\top_\pm
	≥ 0	\perp_\pm	>0	≥ 0	\top_\pm	≥ 0	\top_\pm	\top_\pm	\top_\pm
	\top_\pm	\perp_\pm	\top_\pm	\top_\pm	\top_\pm	\top_\pm	\top_\pm	\top_\pm	\top_\pm

Computational design of the rule of signs

$$\begin{aligned} & >0 \text{ }_{\pm} \leq 0 \\ \triangleq & \alpha_{\pm}(\{x - y \mid x \in \gamma_{\pm}(>0) \wedge y \in \gamma_{\pm}(\leq 0)\}) \\ = & \alpha_{\pm}(\{x - y \mid x > 0 \wedge y \leq 0\}) \\ = & \alpha_{\pm}(\{x - y \mid x > 0 \wedge -y \geq 0\}) \\ \subseteq & \alpha_{\pm}(\{x - y \mid x - y > 0\}) \\ = & \alpha_{\pm}(\{z \mid z > 0\}) \\ = & >0 \end{aligned}$$

Same calculus for all other cases (can be automated with a theorem prover, so called *predicate abstraction*).

Sign abstract semantics (revisited)

- If a variable y has sign \perp_{\pm} , then $\gamma_{\pm}(\perp_{\pm}) = \emptyset$ so the expression is not evaluated hence returns no value
- Define $\Downarrow^{\pm}[P]s \triangleq (\exists y \in \mathcal{V} . P(y) = \perp_{\pm} \text{ ? } \perp_{\pm} \circ s)$ to force returning \perp_{\pm} if a variable has abstract value \perp_{\pm}
- The following sign abstract semantics is more precise:

$$\begin{aligned}\mathcal{S}^{\pm}[\mathbf{1}]P &= \Downarrow^{\pm}[P](>0) \\ \mathcal{S}^{\pm}[\mathbf{x}]P &= \Downarrow^{\pm}[P](P(\mathbf{x})) \\ \mathcal{S}^{\pm}[\mathbf{A}_1 - \mathbf{A}_2]P &= (\mathcal{S}^{\pm}[\mathbf{A}_1]P) \neg_{\pm} (\mathcal{S}^{\pm}[\mathbf{A}_2]P)\end{aligned}\tag{3.19}$$

- It follows that $\exists x \in \mathcal{V} . P(x) = \perp_{\pm}$ implies $\mathcal{S}^{\pm}[\mathbf{A}]P = \perp_{\pm}$.

Soundness

Sign concretization

- Sign

$$\begin{array}{ll} \gamma_{\pm}(\perp_{\pm}) \triangleq \emptyset & \gamma_{\pm}(\leq 0) \triangleq \{z \in \mathbb{Z} \mid z \leq 0\} \\ \gamma_{\pm}(< 0) \triangleq \{z \in \mathbb{Z} \mid z < 0\} & \gamma_{\pm}(\neq 0) \triangleq \{z \in \mathbb{Z} \mid z \neq 0\} \\ \gamma_{\pm}(= 0) \triangleq \{0\} & \gamma_{\pm}(\geq 0) \triangleq \{z \in \mathbb{Z} \mid z \geq 0\} \\ \gamma_{\pm}(> 0) \triangleq \{z \in \mathbb{Z} \mid z > 0\} & \gamma_{\pm}(\top_{\pm}) \triangleq \mathbb{Z} \end{array} \quad (3.21)$$

- Sign environment

$$\dot{\gamma}_{\pm}(\overset{\pm}{\rho}) \triangleq \{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \forall x \in \mathcal{V} . \rho(x) \in \gamma_{\pm}(\overset{\pm}{\rho}(x))\} \quad (3.22)$$

- Sign abstract property

$$\ddot{\gamma}_{\pm}(\overline{P}) \triangleq \{\mathcal{S} \in (\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z} \mid \forall \overset{\pm}{\rho} \in \mathcal{V} \rightarrow \mathbb{P}^{\pm} . \forall \rho \in \dot{\gamma}_{\pm}(\overset{\pm}{\rho}) . \mathcal{S}(\rho) \in \gamma_{\pm}(\overline{P}(\overset{\pm}{\rho}))\} \quad (3.23)$$

Sign abstraction

- Value property

$$\alpha_{\pm}(P) \triangleq \left(\begin{array}{l} P \subseteq \emptyset \text{ ? } \perp_{\pm} \\ P \subseteq \{z \mid z < 0\} \text{ ? } <0 \\ P \subseteq \{0\} \text{ ? } =0 \\ P \subseteq \{z \mid z > 0\} \text{ ? } >0 \\ P \subseteq \{z \mid z \leq 0\} \text{ ? } \leq 0 \\ P \subseteq \{z \mid z \neq 0\} \text{ ? } \neq 0 \\ P \subseteq \{z \mid z \geq 0\} \text{ ? } \geq 0 \\ \vdots \top_{\pm} \end{array} \right) \quad (3.28)$$

- Environment property

$$\dot{\alpha}_{\pm}(P) \triangleq \lambda x \in \mathcal{V} \cdot \alpha_{\pm}(\{\rho(x) \mid \rho \in P\}) \quad (3.31)$$

- Semantics property

$$\ddot{\alpha}_{\pm}(P) \triangleq \lambda \dot{\rho} \in \mathcal{V} \rightarrow \mathbb{P}^{\pm} \cdot \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in P \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) \quad (3.32)$$

Example of environment property abstraction

- The property of environments such that x is equal to 1:

$$\{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \rho(x) = 1\}$$

- Sign abstraction:

$$\dot{\alpha}_{\pm}(\{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \rho(x) = 1\})$$

$$\triangleq \lambda y \in \mathcal{V} . \alpha_{\pm}(\{\rho(y) \mid \rho \in \{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \rho(x) = 1\}\})$$

$$= \lambda y \in \mathcal{V} . \llbracket y = x \stackrel{?}{=} \alpha_{\pm}(\{1\}) \circ \alpha_{\pm}(\mathbb{Z}) \rrbracket$$

$$= \lambda y \in \mathcal{V} . \llbracket y = x \stackrel{?}{=} >0 \circ \top_{\pm} \rrbracket$$

- Sign concretization:

$$\dot{\gamma}_{\pm}(\lambda y \in \mathcal{V} . \llbracket y = x \stackrel{?}{=} >0 \circ \top_{\pm} \rrbracket)$$

$$\triangleq \{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \forall z \in \mathcal{V} . \rho(z) \in \dot{\gamma}_{\pm}(\lambda y \in \mathcal{V} . \llbracket y = x \stackrel{?}{=} >0 \circ \top_{\pm} \rrbracket)(z))\}$$

$$= \{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \rho(x) > 0\}$$

Galois connections

- Value to sign

$$\langle \wp(\mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{P}^{\pm}, \sqsubseteq \rangle$$

- Value environment to sign environment

$$\langle \wp(\mathcal{V} \rightarrow \mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\dot{\alpha}_{\pm}]{\dot{\gamma}_{\pm}} \langle \mathcal{V} \rightarrow \mathbb{P}^{\pm}, \dot{\sqsubseteq}_{\pm} \rangle$$

- Semantic to sign abstract semantic property

$$\langle \wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\ddot{\alpha}_{\pm}]{\ddot{\gamma}_{\pm}} \langle (\mathcal{V} \rightarrow \mathbb{P}^{\pm}) \rightarrow \mathbb{P}^{\pm}, \ddot{\sqsubseteq}_{\pm} \rangle$$

Soundness of the abstract sign semantics

- The abstract sign semantics is an abstraction of the collecting property

$$\begin{aligned} \mathcal{S}^c[A] &\subseteq \gamma_{\pm}(\mathcal{S}^{\pm}[A]) \\ \Leftrightarrow \alpha_{\pm}(\mathcal{S}^c[A]) &\sqsubseteq \mathcal{S}^{\pm}[A] \end{aligned}$$

- Precision loss: if the sign of x is ≤ 0 then the sign of $x - x$ is \top_{\pm} not $=0$
- The absolute value is abstracted away
- No precision loss for multiplication \times

Computational design of the sign semantics

Case when $\exists x \in \mathcal{V} . \dot{\rho}(x) = \perp_{\pm}$ so that $\dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset$

$$\begin{aligned}
 & - \ddot{\alpha}_{\pm}(\mathcal{S}^{\mathbb{C}}[\![A]\!])\dot{\rho} \\
 & = \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^{\mathbb{C}}[\![A]\!] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.32) of } \ddot{\alpha}_{\pm}\}} \\
 & = \alpha_{\pm}(\{\mathcal{A}[\![A]\!](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.11) of } \mathcal{S}^{\mathbb{C}}[\![A]\!]\}} \\
 & = \alpha_{\pm}(\emptyset) && \text{\{ } \exists x \in \mathcal{V} . \dot{\rho}(x) = \perp_{\pm} \text{ so that } \dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset \}} \\
 & = \perp_{\pm} && \text{\{def. (3.28) of } \alpha_{\pm}\}} \\
 & \triangleq \mathcal{S}^{\pm}[\![A]\!]\dot{\rho} \\
 & \text{\{in accordance with (3.19) such that } \exists x \in \mathcal{V} . \dot{\rho}(x) = \perp_{\pm} \text{ implies } \mathcal{S}^{\pm}[\![A]\!]\dot{\rho} = \perp_{\pm}.\}}
 \end{aligned}$$

Homework: Case of a variable x

$$\begin{aligned}
 & \ddot{\alpha}_{\pm}(\mathcal{S}^{\mathbb{C}}[\![x]\!])\dot{\rho} \\
 = & \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^{\mathbb{C}}[\![x]\!] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \{ \text{def. (3.32) of } \ddot{\alpha}_{\pm} \} \\
 = & \alpha_{\pm}(\{\mathcal{A}[\![x]\!](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \{ \text{def. (3.11) of } \mathcal{S}^{\mathbb{C}}[\![x]\!] \} \\
 = & \alpha_{\pm}(\{\rho(x) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \{ \text{def. (3.4) of } \mathcal{A}[\![x]\!] \} \\
 = & \alpha_{\pm}(\{\rho(x) \mid \forall y \in \mathbb{V} . \rho(y) \in \gamma_{\pm}(\dot{\rho}(y))\}) && \{ \text{def. (3.22) of } \dot{\gamma}_{\pm} \} \\
 = & \alpha_{\pm}(\{\rho(x) \mid \rho(x) \in \gamma_{\pm}(\dot{\rho}(x))\}) \\
 & \{ \text{since } \gamma_{\pm}(\dot{\rho}(y)) \text{ is not empty so for } y \neq x, \rho(y) \text{ can be chosen arbitrarily to} \\
 & \text{satisfy } \rho(y) \in \gamma_{\pm}(\dot{\rho}(y)) \} \\
 = & \alpha_{\pm}(\{x \mid x \in \gamma_{\pm}(\dot{\rho}(x))\}) && \{ \text{letting } x = \rho(x) \} \\
 = & \alpha_{\pm}(\gamma_{\pm}(\dot{\rho}(x))) && \{ \text{since } S = \{x \mid x \in S\} \text{ for any set } S \} \\
 = & \dot{\rho}(x) && \{ \text{by (3.35), } \alpha_{\pm} \circ \gamma_{\pm} \text{ is the identity} \} \\
 \triangleq & \mathcal{S}^{\pm}[\![x]\!]\dot{\rho} && \{ \text{in accordance with (3.19) when } \forall y \in \mathbb{V} . \dot{\rho}(y) \neq \perp_{\pm} \}
 \end{aligned}$$

Other cases

- similar for $\ddot{\alpha}_{\pm}(\mathcal{S}^c[\![1]\!])^{\pm}$
- by structural induction for $\ddot{\alpha}_{\pm}(\mathcal{S}^c[\![A_1 - A_2]\!])$
- See the course notes in the appendix.

Chapter 11

Galois Connections and Abstraction

Galois connections

- Given posets $\langle C, \sqsubseteq \rangle$ (the *concrete domain*) and $\langle \mathcal{A}, \preceq \rangle$ (the *abstract domain*), the pair $\langle \alpha, \gamma \rangle$ of functions $\alpha \in C \rightarrow \mathcal{A}$ (the *lower adjoint* or *abstraction*) and $\gamma \in \mathcal{A} \rightarrow C$ (the *upper-adjoint* or *concretization*) is a *Galois connection* (GC) if and only if

$$\forall P \in C . \forall \bar{P} \in \mathcal{A} . \alpha(P) \preceq \bar{P} \Leftrightarrow P \sqsubseteq \gamma(\bar{P}) \quad (11.1)$$

which we write

$$\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle .$$

Example: homomorphic/partition abstraction

- Let C and A be sets, $h \in C \rightarrow A$
- $\alpha_h(S) \triangleq \{h(e) \mid e \in S\}$
- $\gamma_h(\bar{S}) \triangleq \{e \in S \mid h(e) \in \bar{S}\}$
- $\langle \wp(C), \subseteq \rangle \xrightleftharpoons[\alpha_h]{\gamma_h} \langle \wp(A), \subseteq \rangle$

Proof

$$\alpha_h(S) \subseteq \bar{S}$$

$$\Leftrightarrow \{h(e) \mid e \in S\} \subseteq \bar{S}$$

{def. α_h }

$$\Leftrightarrow \forall e \in S . h(e) \in \bar{S}$$

{def. \subseteq }

$$\Leftrightarrow S \subseteq \{e \mid h(e) \in \bar{S}\}$$

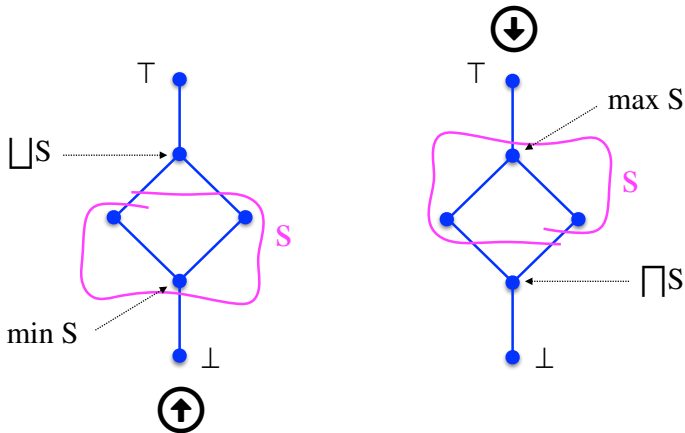
{def. \subseteq }

$$\Leftrightarrow S \subseteq \gamma_h(\bar{S})$$

{def. γ_h } \square

Duality in order theory

- The properties derived for \sqsubseteq , \perp , \top , \sqcup , \max , \sqcap , \min , etc. are valid for the dual \sqsupseteq , \top , \perp , \sqcap , \min , \sqcup , \max , etc.
- Intuition:



Dual of a Galois connection

- The **dual of** a Galois connection $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ is the Galois connection $\langle \mathcal{A}, \preceq \rangle \xrightleftharpoons[\gamma]{\alpha} \langle C, \sqsubseteq \rangle$

Proof $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$

$$\Leftrightarrow \alpha(x) \preceq y \Leftrightarrow x \sqsubseteq \gamma(y)$$

{def. Galois connection}

$$\alpha(x) \succeq y \Leftrightarrow x \sqsupseteq \gamma(y)$$

{dual statement}

$$\Leftrightarrow \gamma(y) \sqsubseteq x \Leftrightarrow y \preceq \alpha(x)$$

{inverse order $x \sqsupseteq y \Leftrightarrow y \sqsubseteq x$ }

$$\Leftrightarrow \gamma(x) \sqsubseteq y \Leftrightarrow x \preceq \alpha(y)$$

{dummy variable renaming}

$$\Leftrightarrow \langle \mathcal{A}, \preceq \rangle \xrightleftharpoons[\gamma]{\alpha} \langle C, \sqsubseteq \rangle$$

{def. Galois connection} \square

- Dualization of a statement involving Galois connections consists in exchanging the adjoints
- If an adjoint has a property, its adjoint has the dual property

Example of dualization

Lemma 1 If $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ then α is increasing. □

Proof Assume $P \sqsubseteq P'$. By $\alpha(P') \preceq \alpha(P')$ we have $P' \sqsubseteq \gamma(\alpha(P'))$ so $P \sqsubseteq \gamma(\alpha(P'))$ by transitivity hence $\alpha(P) \sqsubseteq \alpha(P')$ by definition of a GC, proving that α is increasing. □

Lemma 2 If $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ then γ is increasing. □

Proof By duality (increasing is self-dual so the dual of “ α is increasing” is “ γ is increasing”). □

Example of dualization

- In a Galois connection $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ we have $\alpha \circ \gamma \circ \alpha = \alpha$

Proof homework For all $x \in C$ and $y \in \mathcal{A}$,

$$\text{--- } \alpha(x) \preceq \alpha(x) \quad \quad \quad \{ \text{reflexivity} \}$$

$$\Rightarrow x \sqsubseteq \gamma(\alpha(x)) \quad \quad \quad \{ \text{def. GC} \}$$

$$\Rightarrow \alpha(x) \preceq \alpha(\gamma(\alpha(x))) \quad \quad \quad \{ \alpha \text{ increasing} \}$$

$$\text{--- } \gamma(y) \sqsubseteq \gamma(y) \quad \quad \quad \{ \text{reflexivity} \}$$

$$\Rightarrow \alpha(\gamma(y)) \preceq y \quad \quad \quad \{ \text{def. GC} \}$$

$$\Rightarrow \alpha(\gamma(\alpha(x))) \preceq \alpha(x) \quad \quad \quad \{ \text{for } y = \alpha(x) \}$$

$$\text{--- } \alpha(x) = \alpha(\gamma(\alpha(x))) \quad \quad \quad \{ \text{antisymmetry} \} \quad \square$$

- The dual is $\gamma \circ \alpha \circ \gamma = \gamma$.

Equivalent definition of Galois connections

- $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle A, \preceq \rangle$ if and only if $\alpha \in C \rightarrow A$ and $\gamma \in A \rightarrow C$ satisfy
 - (1) α is increasing;
 - (2) γ is increasing;
 - (3) $\forall x \in C . x \sqsubseteq \gamma \circ \alpha(x)$ (i.e. $\gamma \circ \alpha$ is extensive)
 - (4) $\forall y \in A . \alpha \circ \gamma(y) \preceq y$ (i.e. $\alpha \circ \gamma$ is reductive)

□

α preserves existing lubs

Lemma 3 If $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ then α preserves lubs that may exist in C . i.e. let \sqcup be the partially defined lub for \sqsubseteq in C and \vee be the partially defined lub for \preceq in \mathcal{A} . Let $S \in \wp(C)$ be any subset of C . If $\sqcup S$ exists in C then the upper bound $\vee \{\alpha(e) \mid e \in S\}$ exists in C and is equal to $\alpha(\sqcup S)$. \square

Proof By existence and definition of the lub $\sqcup S$, we have $\forall e \in S. e \sqsubseteq \sqcup S$ so $\alpha(e) \preceq \alpha(\sqcup S)$ since α is increasing. It follows that $\alpha(\sqcup S)$ is an upper bound of $\{\alpha(e) \mid e \in S\}$. Let u be any upper bound of this set $\{\alpha(e) \mid e \in S\}$ so that $\forall e \in S. \alpha(e) \preceq u$. By definition of a GC, $\forall e \in S. e \sqsubseteq \gamma(u)$. So $\gamma(u)$ is an upper bound of S . By existence and definition of the lub $\sqcup S$, $\sqcup S \sqsubseteq \gamma(u)$ so $\alpha(\sqcup S) \preceq u$ proving that $\alpha(\sqcup S)$, which exists since α is a total function, is the lub of $\{\alpha(e) \mid e \in S\}$ denoted $\vee \{\alpha(e) \mid e \in S\}$. \square

- By duality γ preserves existing meets.

lub-preserving α

Lemma 4 If α preserves existing lubs and $\gamma(y) \triangleq \bigsqcup \{x \in C \mid \alpha(x) \leq y\}$ is well-defined then $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \leq \rangle$. □

Proof $x \sqsubseteq \gamma(y)$

$\Rightarrow x \sqsubseteq \bigsqcup \{x' \in C \mid \alpha(x') \leq y\}$ {def. γ }

$\Rightarrow \alpha(x) \leq \alpha(\bigsqcup \{x' \in C \mid \alpha(x') \leq y\})$ { α preserves existing lubs so is increasing}

$\Rightarrow \alpha(x) \leq \bigvee \{\alpha(x') \mid x' \in C \wedge \alpha(x') \leq y\}$ { α preserves existing lubs}

$\Rightarrow \alpha(x) \leq y$

{since y is an upper bound of $\{\alpha(x') \mid \alpha(x') \leq y\}$ greater than or equal to the
lub $\bigvee \{\alpha(x') \mid \alpha(x') \leq y\}$ }

$\Rightarrow x \leq \bigsqcup \{x' \in C \mid \alpha(x') \leq y\}$ {since $x \in \{x' \in C \mid \alpha(x') \leq y\}$ }

$\Rightarrow x \leq \gamma(y)$ {def. γ } □

Uniqueness of adjoints

Lemma 5 In a Galois connection one adjoint uniquely determines the other. \square

Proof Observe that $\forall P \in C . \alpha(P) = \sqcap \{\bar{P} \mid \alpha(P) \preceq \bar{P}\}$ so, by definition of a GC, $\alpha(P) = \sqcap \{\bar{P} \mid P \sqsubseteq \gamma(\bar{P})\}$ i.e. γ uniquely determines α . Dually α uniquely determines γ since $\forall \bar{P} \in \mathcal{A} . \gamma(\bar{P}) = \sqcup \{P \mid \alpha(P) \preceq \bar{P}\}$. \square

- This lemma is useful in situations where only one adjoint is defined explicitly since then the other is also uniquely determined.
- Note: for given concrete and abstract partial orders

Galois retraction (surjection/insertion)

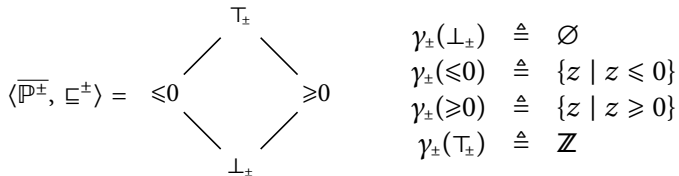
- If $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ then
 - α is surjective, if and only if
 - γ is injective, if and only if
 - $\forall \bar{P} \in \mathcal{A} . \alpha \circ \gamma(\bar{P}) = \bar{P}$.
- This is denoted $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ and called a Galois retraction (Galois surjection, insertion, etc.).

Abstraction

Sound abstraction

- Assume $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$
- We say that $\bar{P} \in \mathcal{A}$ is a *sound abstraction* of $P \in C$ if and only if
$$P \sqsubseteq \gamma(\bar{P})$$

Examples of sound abstractions



property	sound abstractions
$\{1, 42\}$	≥ 0 and \top_\pm
$\{0\}$	≤ 0 , ≥ 0 , and \top_\pm

Better abstraction

- Assume $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$
- Let $\bar{P}_1, \bar{P}_2 \in \mathcal{A}$ be sound abstractions of the concrete property $P \in C$.
- We say that \bar{P}_1 is **better/more precise/stronger/less abstract** than \bar{P}_2 if and only if $\bar{P}_1 \preceq \bar{P}_2$.

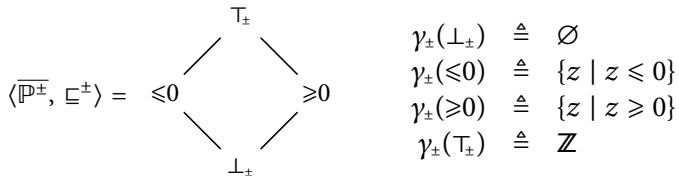
Best abstraction

- Assume $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$
- Then $\alpha(P)$ is the **best/most precise/strongest/least abstract** property which is a sound abstraction of the concrete property P .

Proof

- $\alpha(P)$ is a sound abstraction of P since $P \sqsubseteq \gamma(\alpha(P))$.
- $\alpha(P)$ is the least sound abstraction of P since $\alpha(P) = \bigsqcap \{\bar{P} \mid P \sqsubseteq \gamma(\bar{P})\}$. □

Examples of best abstractions



property	sound abstractions	best abstraction
$\{1, 42\}$	≥ 0 and \top_\pm	≥ 0
$\{0\}$	≤ 0 , ≥ 0 , and \top_\pm	none

- There is no Galois connection between $\langle \wp(\mathbb{Z}), \subseteq \rangle$ and $\langle \overline{\mathbb{P}^\pm}, \sqsubseteq^\pm \rangle$.

Combination of Galois connections

Composition of Galois connections

- The composition of Galois connections $\langle \mathcal{P}_1, \sqsubseteq \rangle \xrightleftharpoons[\alpha_1]{\gamma_1} \langle \mathcal{P}_2, \preceq \rangle$ and $\langle \mathcal{P}_2, \preceq \rangle \xrightleftharpoons[\alpha^2]{\gamma^2} \langle \mathcal{P}_3, \trianglelefteq \rangle$ is the Galois connection $\langle \mathcal{P}_1, \sqsubseteq \rangle \xrightleftharpoons[\alpha^2 \circ \alpha_1]{\gamma_1 \circ \gamma^2} \langle \mathcal{P}_3, \trianglelefteq \rangle$.

Galois connections pairs

- Let $\langle C_1, \sqsubseteq_1 \rangle \xleftrightarrow[\alpha_1]{\gamma_1} \langle \mathcal{A}_1, \preceq_1 \rangle$ and $\langle C_2, \sqsubseteq_2 \rangle \xleftrightarrow[\alpha_2]{\gamma_2} \langle \mathcal{A}_2, \preceq_2 \rangle$;
- $\langle C_1 \times C_2, \dot{\sqsubseteq} \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}_1 \times \mathcal{A}_2, \dot{\preceq} \rangle$, where
- $\alpha(\langle x, y \rangle) = \langle \alpha_1(x), \alpha_2(y) \rangle$,
- $\gamma(\langle \bar{x}, \bar{y} \rangle) = \langle \gamma_1(\bar{x}), \gamma_2(\bar{y}) \rangle$, and
- $\dot{\sqsubseteq}$ and $\dot{\preceq}$ are componentwise.

Higher-order Galois connections

- Let $\langle C_1, \sqsubseteq_1 \rangle \xleftrightarrow[\alpha_1]{\gamma_1} \langle \mathcal{A}_1, \preceq_1 \rangle$ and $\langle C_2, \sqsubseteq_2 \rangle \xleftrightarrow[\alpha_2]{\gamma_2} \langle \mathcal{A}_2, \preceq_2 \rangle$;
- $\langle C_1 \multimap C_2, \dot{\sqsubseteq}_2 \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}_1 \multimap \mathcal{A}_2, \dot{\preceq}_2 \rangle$, where
- $\alpha = \lambda f \cdot \alpha_2 \circ f \circ \gamma_1$, and
- $\gamma = \lambda \bar{f} \cdot \gamma_2 \circ \bar{f} \circ \alpha_1$.

$$\begin{array}{ccc}
 \mathcal{A}_1 & \xrightarrow{\quad \bar{f} \quad} & \mathcal{A}_2 \\
 \gamma_1 \left(\begin{array}{c} \uparrow \\ \downarrow \end{array} \right) \alpha_1 & & \gamma_2 \left(\begin{array}{c} \uparrow \\ \downarrow \end{array} \right) \alpha_2 \\
 C_1 & \xrightarrow{\quad f \quad} & C_2
 \end{array}$$

Conclusion on abstraction by Galois connections

- We can represent abstract program properties by posets and establish the correspondence with the concrete properties using a Galois connection.
- The concrete order structure is preserved in the abstract and inversely.
- Otherwise stated concrete and abstract implications coincide up to the Galois connection.
- So proofs in the abstract domain $\langle \mathcal{A}, \preceq \rangle$ using the abstract implication/order \preceq is valid in the concrete $\langle \mathcal{C}, \sqsubseteq \rangle$ for \sqsubseteq , up to this GC.

Bibliography on abstraction

References I

- Bertrane, Julien, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival (2015). “Static Analysis and Verification of Aerospace Software by Abstract Interpretation”. *Foundations and Trends in Programming Languages* 2.2-3, pp. 71–190.
- Blanchet, Bruno, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival (2003). “A static analyzer for large safety-critical software”. In: *PLDI*. ACM, pp. 196–207.
- Cousot, Patrick (Mar. 1978). “Méthodes itératives de construction et d’approximation de points fixes d’opérateurs monotones sur un treillis, analyse sémantique de programmes (in French)”. *Thèse d’État ès sciences mathématiques*. Grenoble, France: Université de Grenoble Alpes.
- (1981). “Semantic foundations of program analysis”. In: S.S. Muchnick and N.D. Jones, eds. *Program Flow Analysis: Theory and Applications*. Englewood Cliffs, New Jersey, USA: Prentice-Hall, Inc. Chap. 10, pp. 303–342.

References II

- Cousot, Patrick (1997). “Types as Abstract Interpretations”. In: *POPL*. ACM Press, pp. 316–331.
- (1999). “The Calculational Design of a Generic Abstract Interpreter”. In: M. Broy and R. Steinbrüggen, eds. *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam.
 - (2000). “Partial Completeness of Abstract Fixpoint Checking”. In: *SARA*. Vol. 1864. Lecture Notes in Computer Science. Springer, pp. 1–25.
 - (2015). “Abstracting Induction by Extrapolation and Interpolation”. In: *VMCAI*. Vol. 8931. Lecture Notes in Computer Science. Springer, pp. 19–42.
- Cousot, Patrick and Radhia Cousot (1976). “Static determination of dynamic properties of programs”. In: *Proceedings of the Second International Symposium on Programming*. Dunod, Paris, France, pp. 106–130.
- (1977a). “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *POPL*. ACM, pp. 238–252.

References III

- Cousot, Patrick and Radhia Cousot (1977b). “Static Determination of Dynamic Properties of Generalized Type Unions”. In: *Language Design for Reliable Software*, pp. 77–94.
- (1977c). “Static determination of dynamic properties of recursive procedures”. In: E.J. Neuhold, ed. *IFIP Conf. on Formal Description of Programming Concepts, St-Andrews, N.B., CA*. North-Holland Pub. Co., pp. 237–277.
 - (1979). “Systematic Design of Program Analysis Frameworks”. In: *POPL*. ACM Press, pp. 269–282.
 - (1992a). “Abstract Interpretation Frameworks”. *J. Log. Comput.* 2.4, pp. 511–547.
 - (1992b). “Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation”. In: *PLILP*. Vol. 631. Lecture Notes in Computer Science. Springer, pp. 269–295.
 - (1994). “Higher Order Abstract Interpretation (and Application to Comportment Analysis Generalizing Strictness, Termination, Projection, and PER Analysis”. In: *ICCL*. IEEE Computer Society, pp. 95–112.

References IV

- Cousot, Patrick and Radhia Cousot (1995). “Formal Language, Grammar and Set-Constraint-Based Program Analysis by Abstract Interpretation”. In: *FPCA*. ACM, pp. 170–181.
- (2000). “Temporal Abstract Interpretation”. In: *POPL*. ACM, pp. 12–25.
 - (2002). “Modular Static Program Analysis”. In: *CC*. Vol. 2304. Lecture Notes in Computer Science. Springer, pp. 159–178.
 - (2004). “Basic Concepts of Abstract Interpretation”. In: René Jacquard, ed. *Building the Information Society*. Springer, pp. 359–366.
 - (2012). “An abstract interpretation framework for termination”. In: *POPL*. ACM, pp. 245–258.
 - (2014). “A Galois connection calculus for abstract interpretation”. In: *POPL*. ACM, pp. 3–4.
- Cousot, Patrick, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival (2005). “The Astrée Analyzer”. In: *ESOP*. Vol. 3444. Lecture Notes in Computer Science. Springer, pp. 21–30.

References V

- Cousot, Patrick, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival (2006). “Combination of Abstractions in the Astrée Static Analyzer”. In: *ASIAC*. Vol. 4435. Lecture Notes in Computer Science. Springer, pp. 272–300.
- Cousot, Patrick, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival (2009). “Why does Astrée scale up?” *Formal Methods in System Design* 35.3, pp. 229–264.
- Cousot, Patrick, Roberto Giacobazzi, and Francesco Ranzato (2018). “Program Analysis Is Harder Than Verification: A Computability Perspective”. In: *CAV (2)*. Vol. 10982. Lecture Notes in Computer Science. Springer, pp. 75–95.
- (Jan. 2019). “A²I: Abstract² Interpretation”. *PACMPL (POPL conference)* 3, article 42. DOI: 10.1145/3290355.
- Cousot, Patrick and Nicolas Halbwachs (1978). “Automatic Discovery of Linear Constraints Among Variables of a Program”. In: *POPL*. ACM Press, pp. 84–96.

The End of Part 2, 30mn break

Part 3

Verification and proofs

Verification and proofs

- We show that **verification methods** and **program logics** are (non-computable) abstractions of the program collecting semantics.

Program properties

Program semantic properties

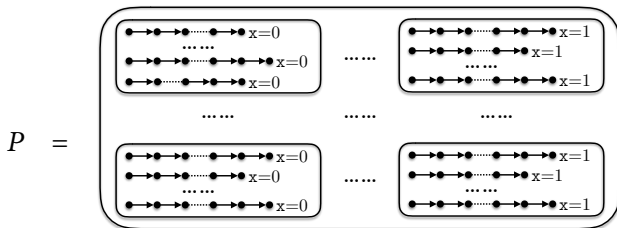
- The entities are semantics of program P *i.e.* sets of maximal traces $\mathcal{E} = \wp(\mathbb{T}^{+\infty})$
- The properties are sets of semantics of program P *i.e.* sets of sets of maximal traces
 $\wp(\mathcal{E}) = \wp(\wp(\mathbb{T}^{+\infty}))^2$

²also called “hyperproperties”

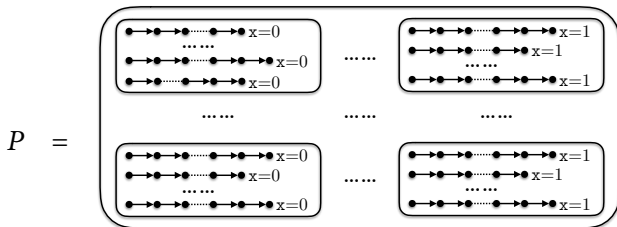
Example of program semantic property

$$P \triangleq \wp(\{\pi \in \mathbb{T}^+ \mid \rho(\pi)x = 0\}) \cup \wp(\{\pi \in \mathbb{T}^+ \mid \rho(\pi)x = 1\}) \in \wp(\wp(\mathbb{T}^{+\infty}))$$

- P means “all executions of P always terminate with $x = 0$ or all executions of P always terminate with $x = 1$ ”.



Example of program semantic property (Cont'd)



- Assume program P has this property P so $\mathcal{S}^{+\infty}[[P]] \in P$.
- Executing program P once, we know the result of all other executions.
- If the execution terminates with $x = 0$ (respectively $x = 1$) the property P implies that all other possible executions will always terminate with $x = 0$ (respectively $x = 1$).

Collecting semantics

Collecting semantics (for maximal traces)

- The strongest semantic property of program P

$$\mathcal{S}^c[P] \triangleq \{\mathcal{S}^{+\infty}[P]\} . \quad (8.5)$$

- Program P has property $P \in \wp(\wp(\mathbb{T}^{+\infty}))$ is
 - $\mathcal{S}^{+\infty}[P] \in P$, or equivalently
 - $\{\mathcal{S}^{+\infty}[P]\} \subseteq P$ i.e. P is implied by the collecting semantics of program P .
- So we can use implication $\subseteq (\Rightarrow)$ instead of \in (with no direct equivalent for predicates in logic).

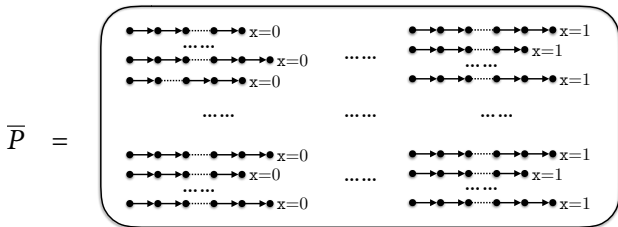
Trace properties

Trace properties

- By “program property” or “semantic property” most computer scientists refer to “trace properties”
- elements $\mathfrak{T} = \mathbb{T}^{+\infty}$, traces
- trace properties $\wp(\mathfrak{T}) = \wp(\mathbb{T}^{+\infty})$
- *safety* and *liveness* are trace properties

Example of trace properties

- the program trace semantics $\mathcal{S}^{+\infty}[\![P]\!] \in \wp(\mathbb{T}^{+\infty})$ is a trace property.
- $\{\pi \in \mathbb{T}^+ \mid \rho(\pi)x = 0\} \in \wp(\mathbb{T}^{+\infty})$ is the trace property of “terminating with $x=0$ ”.
- $\bar{P} = \{\pi \in \mathbb{T}^+ \mid \rho(\pi)x \in \{0, 1\}\} \in \wp(\mathbb{T}^{+\infty})$ is the trace property of “terminating with $x=0$ or $x=1$ ”.



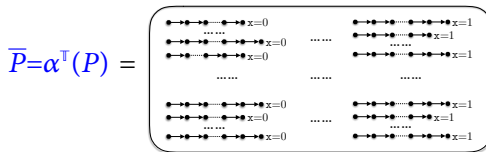
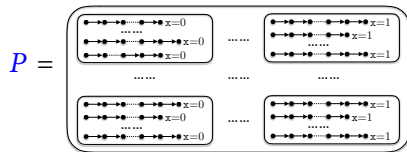
- Trace properties in $\wp(\mathbb{T}^{+\infty})$ are less expressive than semantic properties in $\wp(\wp(\mathbb{T}^{+\infty}))$

Abstraction of a semantic property into a trace property

- Any semantic property P can be abstracted into a less precise trace property $\alpha^{\top}(P)$ defined as

$$\begin{aligned}\alpha^{\top} &\in \wp(\wp(\mathbb{T}^{+\infty})) \rightarrow \wp(\mathbb{T}^{+\infty}) \\ \alpha^{\top}(P) &= \bigcup P\end{aligned}$$

$$\begin{aligned}\gamma^{\top} &\in \wp(\mathbb{T}^{+\infty}) \rightarrow \wp(\wp(\mathbb{T}^{+\infty})) \\ \gamma^{\top}(\bar{P}) &= \wp(\bar{P})\end{aligned}$$



- P and \bar{P} both express that program executions always terminate with a boolean value for x .
- P is stronger since it expresses that the result is always the same while \bar{P} doesn't.

Abstraction of a semantic property into a trace property (Cont'd)

- Galois connection $\langle \wp(\wp(\mathbb{T}^{+\infty})), \subseteq \rangle \xleftrightarrow[\alpha^\top]{\gamma^\top} \langle \wp(\mathbb{T}^{+\infty}), \subseteq \rangle$

- Proof:

$$\alpha^\top(P) \subseteq \overline{P}$$

$$\Leftrightarrow \bigcup P \subseteq \overline{P}$$

{def. α^\top }

$$\Leftrightarrow \{x \mid \exists X \in P . x \in X\} \subseteq \overline{P}$$

{def. \bigcup }

$$\Leftrightarrow \forall X \in P . \forall x \in X . x \in \overline{P}$$

{def. \subseteq }

$$\Leftrightarrow \forall X \in P . X \subseteq \overline{P}$$

{def. \subseteq }

$$\Leftrightarrow P \subseteq \{X \mid X \subseteq \overline{P}\}$$

{def. \subseteq }

$$\Leftrightarrow P \subseteq \wp(\overline{P})$$

{def. \wp }

$$\Leftrightarrow P \subseteq \gamma^\top(\overline{P})$$

{def. γ^\top .}

- α^\top is surjective (since $\alpha^\top(\{\overline{P}\}) = \overline{P}$).

Reachability properties

Reachability property

A relation $\mathcal{I}(\ell)$ between values of variables attached to each program point ℓ that holds whenever the program point ℓ is reached during execution

ℓ_1 */* x = 0 */*

 x = x + 1 ;

 while ℓ_2 (tt) */* 1 ≤ x ≤ 2 */* {

ℓ_3 */* 1 ≤ x ≤ 2 */*

 x = x + 1 ;

 if ℓ_4 (x > 2) */* 2 ≤ x ≤ 3 */*

ℓ_5 */* x = 3 */*

 break ;

 }

ℓ_6 */* x = 3 */*

 ;

ℓ_7 */* x = 3 */*

$$\mathcal{I}(\ell_1) \triangleq \{\rho \in \mathbb{E}\mathbb{V} \mid \forall y \in \mathbb{V} . \rho(y) = 0\}$$

$$\mathcal{I}(\ell_2) \triangleq \mathcal{I}(\ell_3) \triangleq \{\rho \in \mathbb{E}\mathbb{V} \mid 1 \leq \rho(x) \leq 2 \wedge \forall y \in \mathbb{V} \setminus \{x\} . \rho(y) = 0\}$$

$$\mathcal{I}(\ell_4) \triangleq \{\rho \in \mathbb{E}\mathbb{V} \mid 2 \leq \rho(x) \leq 3 \wedge \forall y \in \mathbb{V} \setminus \{x\} . \rho(y) = 0\}$$

$$\mathcal{I}(\ell_5) \triangleq \mathcal{I}(\ell_6) \triangleq \mathcal{I}(\ell_7) \triangleq \{\rho \in \mathbb{E}\mathbb{V} \mid \rho(x) = 3 \wedge \forall y \in \mathbb{V} \setminus \{x\} . \rho(y) = 0\}$$

Abstraction of a trace property into a reachability property

$$\begin{aligned}\alpha^\sharp &\in \wp(\mathbb{T}^{+\infty}) \rightarrow (\mathbb{L} \rightarrow \wp(\mathbb{E}\mathbf{v})) \\ \alpha^\sharp(\Pi) &\triangleq \lambda \ell \cdot \{\rho(\pi^\ell) \mid \exists \pi' . \pi^\ell \pi' \in \Pi\}\end{aligned}\tag{8.12}$$

collects at each program point ℓ of each trace the possible values of the variables at that point.

Abstraction of a trace property into a reachability property (Cont'd)

- Galois connection $\langle \wp(\mathbb{T}^{+\infty}), \subseteq \rangle \xrightleftharpoons[\alpha^!]{\gamma^!} \langle (\mathbb{L} \rightarrow \wp(\mathbb{E}\mathbb{V})), \dot{\subseteq} \rangle$

- Proof:

$$\alpha^!(\Pi) \dot{\subseteq} \mathcal{I}$$

$$\Leftrightarrow \lambda \ell . \{\rho(\pi^\ell) \mid \exists \pi' . \pi^\ell \pi' \in \Pi\} \dot{\subseteq} \mathcal{I} \quad \{\text{def. } \alpha^!\}$$

$$\Leftrightarrow \forall \ell . \{\rho(\pi^\ell) \mid \exists \pi' . \pi^\ell \pi' \in \Pi\} \subseteq \mathcal{I}(\ell) \quad \{\text{pointwise def. } \dot{\subseteq}\}$$

$$\Leftrightarrow \forall \ell . \{\rho(\pi^\ell) \mid \exists \bar{\pi} \in \Pi . \exists \pi' . \bar{\pi} = \pi^\ell \pi'\} \subseteq \mathcal{I}(\ell) \quad \{\text{def. } \in\}$$

$$\Leftrightarrow \forall \ell . \forall \bar{\pi} \in \Pi . \forall \pi' . \bar{\pi} = \pi^\ell \pi' \Rightarrow \rho(\pi^\ell) \in \mathcal{I}(\ell) \quad \{\text{def. } \subseteq\}$$

$$\Leftrightarrow \forall \bar{\pi} \in \Pi . \forall \pi' . \forall \ell . \bar{\pi} = \pi^\ell \pi' \Rightarrow \rho(\pi^\ell) \in \mathcal{I}(\ell) \quad \{\text{def. } \forall\}$$

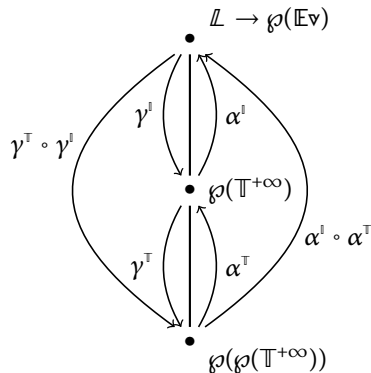
$$\Leftrightarrow \Pi \subseteq \{\bar{\pi} \mid \forall \pi' . \forall \ell . \bar{\pi} = \pi^\ell \pi' \Rightarrow \rho(\pi^\ell) \in \mathcal{I}(\ell)\} \quad \{\text{def. } \subseteq\}$$

$$\Leftrightarrow \Pi \subseteq \gamma^!(\mathcal{I})$$

by defining $\gamma^!(\mathcal{I}) \triangleq \{\bar{\pi} \mid \forall \pi' . \forall \ell . \bar{\pi} = \pi^\ell \pi' \Rightarrow \rho(\pi^\ell) \in \mathcal{I}(\ell)\}$.

Hierarchy of program properties

Hierarchy of program properties/semantics



$$\begin{aligned} \mathcal{S}^I[\mathbf{P}] &= \alpha^I(\mathcal{S}^T[\mathbf{P}]) \\ &= \alpha^I \circ \alpha^T(\mathcal{S}^C[\mathbf{P}]) \end{aligned} \quad \begin{array}{l} \text{invariance/} \\ \text{reachability} \\ \text{semantics} \end{array}$$

$$\begin{aligned} \mathcal{S}^T[\mathbf{P}] &= \mathcal{S}^{+\infty}[\mathbf{P}] \\ &= \alpha^T(\mathcal{S}^C[\mathbf{P}]) \end{aligned} \quad \text{trace semantics}$$

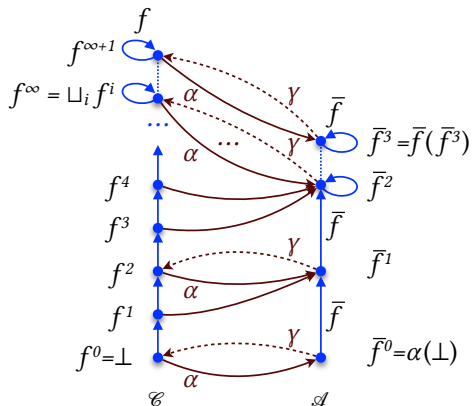
$$\mathcal{S}^C[\mathbf{P}] \triangleq \{\mathcal{S}^{+\infty}[\mathbf{P}]\}, \quad \text{collecting semantics}$$

Fixpoint abstraction

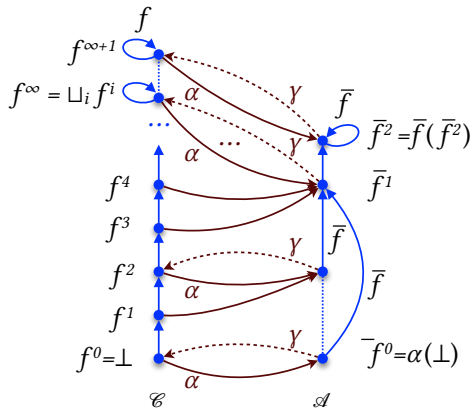
Fixpoint abstraction

- \mathcal{C} is a concrete domain
- $f \in \mathcal{C} \rightarrow \mathcal{C}$ is an increasing concrete transformer
- $\langle \mathcal{C}, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ is an abstraction into \mathcal{A}
- Problem: abstract $\text{lfp}^{\sqsubseteq} f$
 - first abstract the concrete transformer f into an abstract transformer $\bar{f} \in \mathcal{A} \rightarrow \mathcal{A}$
 - then abstract $\alpha(\text{lfp}^{\sqsubseteq} f)$ into $\text{lfp}^{\preceq} \bar{f}$.
 - This abstraction may be
 - *exact* i.e. $\alpha(\text{lfp}^{\sqsubseteq} f) = \text{lfp}^{\preceq} \bar{f}$
 - or *sound* but imprecise, in which case we get an overapproximation $\alpha(\text{lfp}^{\sqsubseteq} f) \preceq \text{lfp}^{\preceq} \bar{f}$.

Example of fixpoint abstraction



exact fixpoint abstraction

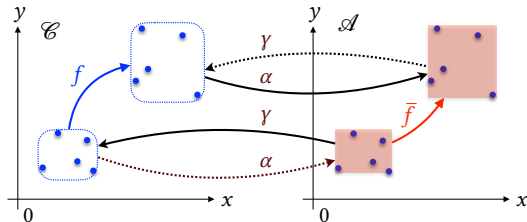


imprecise fixpoint abstraction

Transformer abstraction

Transformer abstraction

- To abstract a fixpoint $\alpha(\text{lfp}^\sqsubseteq f)$, we first abstract its transformer f .

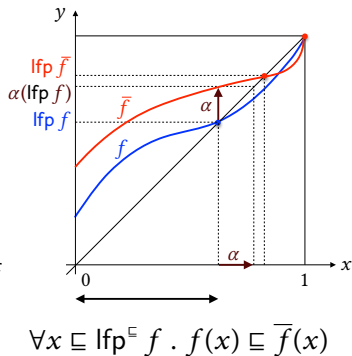
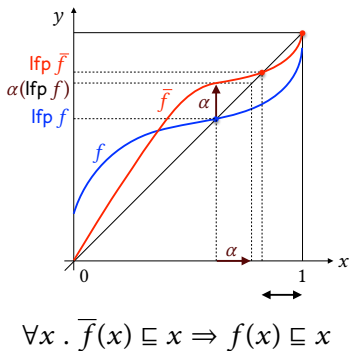
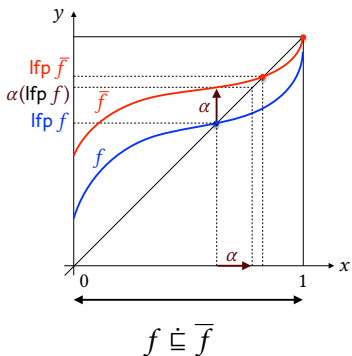


Theorem (16.1, transformer abstraction) If $\langle \mathcal{C}, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ then $\langle \mathcal{C} \multimap \mathcal{C}, \sqsubseteq \rangle \xleftrightarrow[\bar{\alpha}]{\bar{\gamma}} \langle \mathcal{A} \multimap \mathcal{A}, \preceq \rangle$ where \sqsubseteq and \preceq are pointwise (i.e. $f \sqsubseteq g$ if and only if $\forall x \in \mathcal{C}. f(x) \sqsubseteq g(x)$), $\bar{\alpha}(f) = \alpha \circ f \circ \gamma$, and $\bar{\gamma}(\bar{f}) = \gamma \circ \bar{f} \circ \alpha$.

Fixpoint over-approximation

Fixpoint over-approximation

- In general abstracting the fixpoint transformer by a larger one yields a fixpoint over-approximation.

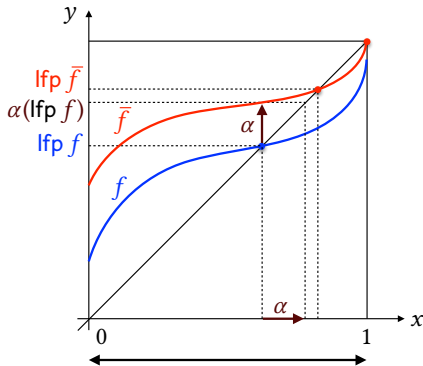


fixpoint over-approximation

Fixpoint over-approximation (cont'd)

Theorem (16.3, pointwise fixpoint over-approximation) Assume that $\langle C, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$ is a complete lattice, $f, g \in C \rightarrow C$ are increasing, and $f \sqsubseteq g$ then $\text{lfp}^\sqsubseteq f \sqsubseteq \text{lfp}^\sqsubseteq g$.

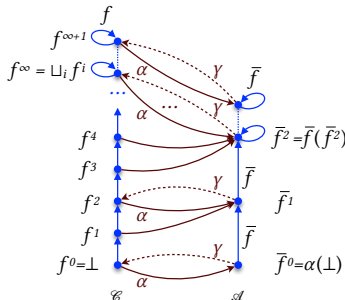
- Also valid for cpos.



Sound fixpoint abstraction

- An abstract fixpoint $\text{lfp}^{\leq} \bar{f}$ is a sound fixpoint abstraction of a concrete fixpoint $\text{lfp}^{\sqsubseteq} f$ whenever $\alpha(\text{lfp}^{\sqsubseteq} f) \leq \text{lfp}^{\leq} \bar{f}$.

Theorem (16.6, fixpoint over-approximation in a complete lattice) Assume that $\langle C, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$ and $\langle \mathcal{A}, \leq, 0, 1, \vee, \wedge \rangle$ are complete lattices, $\langle C, \sqsubseteq \rangle \xleftrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \leq \rangle$, and $f \in C \rightarrow C$ is increasing. Then $\text{lfp}^{\sqsubseteq} f \sqsubseteq \gamma(\text{lfp}^{\leq} \alpha \circ f \circ \gamma)$.

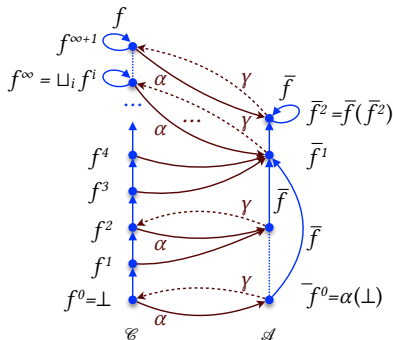


Sound fixpoint abstraction (cont'd)

Corollary (16.8, fixpoint approximation by transformer over-approximation)

Assume that $\langle C, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$ and $\langle \mathcal{A}, \leq, 0, 1, \vee, \wedge \rangle$ are complete lattices, $\langle C, \sqsubseteq \rangle \xrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \leq \rangle$, $f \in C \rightarrow C$ and $\bar{f} \in \mathcal{A} \rightarrow \mathcal{A}$ are increasing, and $\alpha \circ f \circ \gamma \leq \bar{f}$.

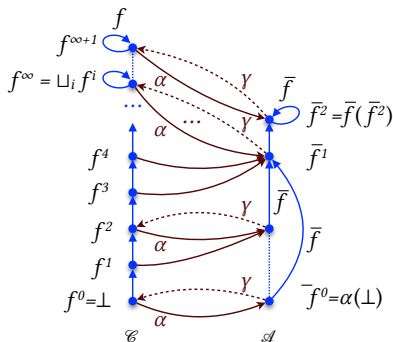
Then $\text{lfp}^\sqsubseteq f \sqsubseteq \gamma(\text{lfp}^{\leq} \bar{f})$.



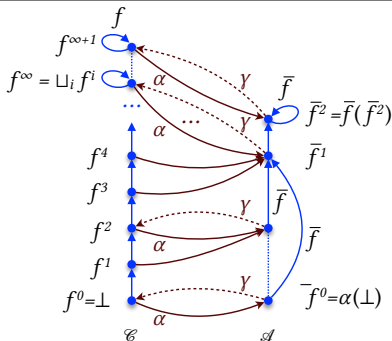
also in a cpo

Theorem (16.12, fixpoint over-approximation in a cpo) Assume that $\langle C, \sqsubseteq, \perp, \sqcup \rangle$ is a cpo and $\langle \mathcal{A}, \preceq, 0, \wedge \rangle$ are cpos, $\langle C, \sqsubseteq \rangle \xrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$, and $f \in C \xrightarrow{uc} C$ is upper continuous.

Then $\text{lfp}^{\sqsubseteq} f \sqsubseteq \gamma(\text{lfp}^{\preceq} \alpha \circ f \circ \gamma)$.



Under the hypotheses of Corollary 16.8 assume instead that $\alpha \circ f \preceq \bar{f} \circ \alpha$ (*semi-commutation*). Then $\text{lfp}^\sqsubseteq f \sqsubseteq \gamma(\text{lfp}^\preceq \bar{f})$.



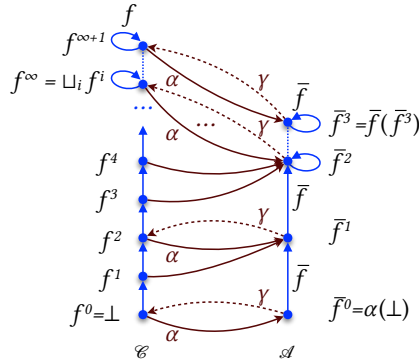
Exact fixpoint abstraction

Exact versus sound fixpoint abstraction

- A sound fixpoint abstraction $\alpha(\text{lfp}^{\sqsubseteq} f) \preceq \text{lfp}^{\preceq} \bar{f}$ is
 - *exact* when $\alpha(\text{lfp}^{\sqsubseteq} f) = \text{lfp}^{\preceq} \bar{f}$.
 - It is *sound but approximate (or imprecise)* when $\alpha(\text{lfp}^{\sqsubseteq} f) < \text{lfp}^{\preceq} \bar{f}$.

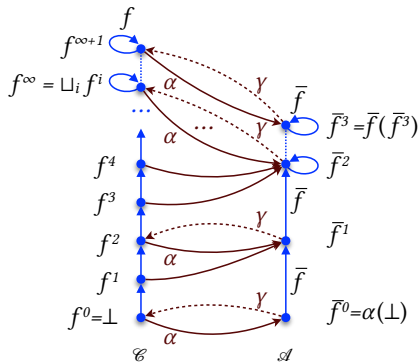
Exact fixpoint abstraction

Theorem (16.15, exact fixpoint abstraction in a complete lattice) Assume that $\langle C, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$ and $\langle \mathcal{A}, \preceq, 0, 1, \vee, \wedge \rangle$ are complete lattices, $f \in C \rightarrow C$ is increasing, $\langle C, \sqsubseteq \rangle \xrightarrow[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$, $\bar{f} \in \mathcal{A} \rightarrow \mathcal{A}$ is increasing, and $\alpha \circ f = \bar{f} \circ \alpha$ (commutation property). Then $\alpha(\text{lfp}^\sqsubseteq f) = \text{lfp}^\preceq \bar{f}$.



Exact fixpoint abstraction (cont'd)

Theorem (16.16, exact fixpoint abstraction in a cpo) Assume that $\langle C, \sqsubseteq \rangle$ is a cpo, $f \in C \xrightarrow{uc} C$ is upper continuous, $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle A, \preceq \rangle$ is a Galois retraction, and $\bar{f} \in A \rightarrow A$ satisfies the commutation property $\alpha \circ f = \bar{f} \circ \alpha$. Then $\bar{f} = \alpha \circ f \circ \gamma$ is increasing and $\alpha(\text{lfp}^\sqsubseteq f) = \text{lfp}^{\preceq} \bar{f} = \bigvee_{n \in \mathbb{N}} \bar{f}^n(\alpha(\perp))$.



Reachability semantics

Reachability abstraction

Assertional abstraction

$$\text{post}^{\vec{r}}(\mathcal{S}) \mathcal{R}_0^\ell \triangleq \{\rho(\pi_0^{\ell_0} \pi_1^{\ell'}) \mid \rho(\pi_0^{\ell_0}) \in \mathcal{R}_0 \wedge \ell_0 \pi_1^{\ell'} \in \mathcal{S}(\pi_0^{\ell_0}) \wedge \ell' = \ell\} \quad (18.1)$$

$$\begin{array}{c} \xrightarrow{\pi_0} \quad \xrightarrow{\ell_0} \quad \xrightarrow{\pi_1} \quad \xrightarrow{\ell} \quad \cdots \xrightarrow{\pi_2} \cdots \\ \hline \rho(\pi_0^{\ell_0}) \in \mathcal{R}_0 \quad \rho(\pi_0^{\ell_0} \pi_1^{\ell}) \in \text{post}^{\vec{r}}(\mathcal{S}) \mathcal{R}_0^\ell \end{array}$$

$\in \mathcal{S}(\pi_0^{\ell_0})$

$$\langle \mathbb{T}^+ \rightarrow \wp(\mathbb{T}^+), \dot{\subseteq} \rangle \xrightleftharpoons[\text{post}^{\vec{r}}]{\gamma^{\vec{r}}} \langle \wp(\mathbb{E}\mathbf{v}) \rightarrow \mathbb{L} \mapsto \wp(\mathbb{E}\mathbf{v}), \dot{\subseteq} \rangle$$

Assertional abstraction, Example

(4.4)

```

ℓ1 x = x + 1 ;
  while ℓ2 (tt) {
    ℓ3 x = x + 1 ;
    if ℓ4 (x > 2) ℓ5 break ; } ℓ6 ; ℓ7
    
```

We assume that all variables are initialized to 0. Maximal trace semantics

$$\mathcal{S} \triangleq \{ \ell_1 \xrightarrow{x=1} \ell_2 \xrightarrow{tt} \ell_3 \xrightarrow{x=2} \ell_4 \xrightarrow{\neg(x>2)} \ell_2 \xrightarrow{tt} \ell_3 \xrightarrow{x=3} \ell_4 \xrightarrow{x>2} \ell_5 \xrightarrow{\text{break}} \ell_6 \xrightarrow{\text{skip}} \ell_7 \} \quad (6.1)$$

The reachable states are

ℓ	$\text{post}^{\vec{r}}(\mathcal{S}) \mathcal{R}_0 \ell$
ℓ_1	$\mathcal{R}_0 = \{ \rho \in \mathbb{E}\mathbb{V} \mid \forall y \in \mathbb{V} . \rho(y) = 0 \}$
ℓ_2, ℓ_3	$\{ \rho[x \leftarrow i] \mid \rho \in \mathcal{R}_0 \wedge i \in [1, 2] \}$
ℓ_4	$\{ \rho[x \leftarrow i] \mid \rho \in \mathcal{R}_0 \wedge i \in [2, 3] \}$
ℓ_5, ℓ_6, ℓ_7	$\{ \rho[x \leftarrow 3] \mid \rho \in \mathcal{R}_0 \}$

□

Calculational design of the reachability semantics

Calculational design of the reachability semantics

- by structural induction
- by calculating the exact reachability transformer from the prefix trace transformer
- by applying the exact fixpoint abstraction 16.15 for the iteration

Reachability semantics of the assignment

Reachability of an assignment statement $S ::= x = A ;$

$$\begin{aligned}\widehat{\mathcal{S}}^{\vec{r}}[S] \mathcal{R}_0 \ell &= (\ell = \text{at}[S] \text{ ? } \mathcal{R}_0 \\ &\quad \parallel \ell = \text{after}[S] \text{ ? } \text{assign}^{\vec{r}}[x, A] \mathcal{R}_0 \\ &\quad : \emptyset) \\ \text{assign}^{\vec{r}}[x, A] \mathcal{R}_0 &\triangleq \{\rho[x \leftarrow \mathcal{A}[A]\rho] \mid \rho \in \mathcal{R}_0\}\end{aligned}\tag{17.10}$$

Reachability semantics of the conditional

Reachability of a conditional statement $S ::= \text{if } (B) S_t$

$$\begin{aligned} \widehat{\mathcal{S}}^{\vec{r}}[S] \mathcal{R}_0^\ell &= (\ell = \text{at}[S] \text{ ? } \mathcal{R}_0 \\ &\quad \mid \ell \in \text{in}[S_t] \text{ ? } \widehat{\mathcal{S}}^{\vec{r}}[S_t] (\text{test}^{\vec{r}}[B] \mathcal{R}_0)^\ell \\ &\quad \mid \ell = \text{after}[S] \text{ ? } \widehat{\mathcal{S}}^{\vec{r}}[S_t] (\text{test}^{\vec{r}}[B] \mathcal{R}_0)^\ell \cup (\overline{\text{test}^{\vec{r}}[B] \mathcal{R}_0}) \\ &\quad \text{: } \emptyset) \end{aligned} \tag{17.18}$$

$$\text{test}^{\vec{r}}[B] \mathcal{R}_0 \triangleq \{ \rho \in \mathcal{R}_0 \mid \mathcal{B}[B] \rho = \text{tt} \}$$

$$\overline{\text{test}^{\vec{r}}[B] \mathcal{R}_0} \triangleq \{ \rho \in \mathcal{R}_0 \mid \mathcal{B}[B] \rho = \text{ff} \}$$

Reachability semantics of the statement list

Reachability of a statement list $sl ::= sl' \ S$

$$\widehat{\mathcal{S}}^{\vec{r}}[sl]\mathcal{R}_0^\ell = \left(\ell \in \text{labs}[sl'] \setminus \{\text{at}[S]\} \ ? \ \widehat{\mathcal{S}}^{\vec{r}}[sl']\mathcal{R}_0^\ell \right. \\ \left. \parallel \ell \in \text{labs}[S] \ ? \ \widehat{\mathcal{S}}^{\vec{r}}[S](\widehat{\mathcal{S}}^{\vec{r}}[sl']\mathcal{R}_0^{\text{at}[S]})^\ell \right. \\ \left. \circ \emptyset \right) \quad (17.20)$$

Reachability semantics of the iteration

Reachability of an iteration statement $S ::= \text{while } \ell \text{ (B) } S_b$

$$\widehat{\mathcal{F}}^{\vec{r}}[\![S]\!] \mathcal{R}_0 \ell' = (\text{lfp}^{\leq} \mathcal{F}^{\vec{r}}[\![\text{while } \ell \text{ (B) } S_b]\!] \mathcal{R}_0) \ell' \quad (17.14)$$

$$\begin{aligned} \mathcal{F}^{\vec{r}}[\![\text{while } \ell \text{ (B) } S_b]\!] \mathcal{R}_0 X \ell' = & \\ & (\ell' = \ell \text{ ? } \mathcal{R}_0 \cup \widehat{\mathcal{F}}^{\vec{r}}[\![S_b]\!] (\text{test}^{\vec{r}}[\![B]\!] X(\ell)) \ell \\ & \mid \ell' \in \text{in}[\![S_b]\!] \setminus \{\ell\} \text{ ? } \widehat{\mathcal{F}}^{\vec{r}}[\![S_b]\!] (\text{test}^{\vec{r}}[\![B]\!] X(\ell)) \ell' \\ & \mid \ell' = \text{after}[\![S]\!] \text{ ? } \overline{\text{test}}^{\vec{r}}[\![B]\!](X(\ell)) \cup \bigcup_{\ell'' \in \text{breaks-of}[\![S_b]\!]} \widehat{\mathcal{F}}^{\vec{r}}[\![S_b]\!] (\text{test}^{\vec{r}}[\![B]\!] X(\ell)) \ell'' \\ & : \emptyset) \end{aligned}$$

Abstract domain and abstract interpreter

Abstract domain

The domain of properties, inclusion (*i.e.* logical implication), and the structural definitions of the semantics have the following common structure.

semantics	prefix trace $\widehat{\mathcal{S}}^*$	reachability $\widehat{\mathcal{S}}^{\vec{r}}$	abstract $\widehat{\mathcal{S}}^{\boxtimes}$
	$\wp(\mathbb{T}^+) \rightarrow (\mathbb{L} \rightarrow \wp(\mathbb{T}^+))$	$\wp(\mathbb{E}\mathbb{V}) \rightarrow (\mathbb{L} \rightarrow \wp(\mathbb{E}\mathbb{V}))$	$\mathbb{P}^{\boxtimes} \rightarrow (\mathbb{L} \rightarrow \mathbb{P}^{\boxtimes})$
domain	$\wp(\mathbb{T}^+)$	$\wp(\mathbb{E}\mathbb{V})$	\mathbb{P}^{\boxtimes}
inclusion	\subseteq	\subseteq	\sqsubseteq^{\boxtimes}
abstraction	$\mathbb{1}_{\wp(\mathbb{T}^+)}^3$	$\ddot{\alpha}_{\rho}$	α_{\boxtimes}
infimum	\emptyset	\emptyset	\perp^{\boxtimes}
join	\cup	\cup	\sqcup^{\boxtimes}
assignment	$\text{assign}^* \llbracket x, A \rrbracket$	$\text{assign}^{\vec{r}} \llbracket x, A \rrbracket$	$\text{assign}^{\boxtimes} \llbracket x, A \rrbracket$
test	$\text{test}^* \llbracket B \rrbracket$	$\text{test}^{\vec{r}} \llbracket B \rrbracket$	$\text{test}^{\boxtimes} \llbracket B \rrbracket$
	$\overline{\text{test}}^* \llbracket B \rrbracket$	$\overline{\text{test}}^{\vec{r}} \llbracket B \rrbracket$	$\overline{\text{test}}^{\boxtimes} \llbracket B \rrbracket$

³ $\mathbb{1}_S \triangleq \lambda x \in S. x$ is the identity function on the set S .

Definition (19.1, Domain well-definedness) We say that a domain

$$\mathbb{D}^\alpha \triangleq \langle \mathbb{P}^\alpha, \sqsubseteq^\alpha, \perp^\alpha, \sqcup^\alpha, \text{assign}^\alpha \llbracket x, A \rrbracket, \text{test}^\alpha \llbracket B \rrbracket, \overline{\text{test}}^\alpha \llbracket B \rrbracket \rangle$$

is *well-defined* when $\langle \mathbb{P}^\alpha, \sqsubseteq^\alpha \rangle$ is a poset of properties with infimum \perp^α , the lub \sqcup^α is well-defined for pairs of properties, and \sqsubseteq^α -increasing chains (so $\langle \mathbb{P}^\alpha, \sqsubseteq^\alpha \rangle$ is a join-lattice and a cpo), the assignment assign^α is well-defined in $(V \times E) \rightarrow \mathbb{P}^\alpha \rightarrow \mathbb{P}^\alpha$, and the tests $\text{test}^\alpha \llbracket B \rrbracket$ and $\overline{\text{test}}^\alpha \llbracket B \rrbracket$ are well-defined in $B \rightarrow \mathbb{P}^\alpha \rightarrow \mathbb{P}^\alpha$.

The abstract domain \mathbb{D}^α is an algebra while the domain of abstract properties \mathbb{P}^α is a set. So the mathematical structures are different. However, following mathematicians that call \mathbb{Z} the “ring of integers” where a ring is an algebraic structure and \mathbb{Z} is a set, we often say, by abuse of language, that \mathbb{P}^α an abstract domain.

Abstract structural semantics/interpreter

The semantics can be implemented as instances of a generic abstract interpreter defined below.

- *Abstract semantics of a statement list $sl ::= sl' s$*

$$\widehat{\mathcal{F}}^{\bowtie} \llbracket sl \rrbracket \mathcal{R}_0^\ell \triangleq \left(\ell \in \text{labs} \llbracket sl' \rrbracket \setminus \{\text{at} \llbracket s \rrbracket\} \text{ ? } \widehat{\mathcal{F}}^{\bowtie} \llbracket sl' \rrbracket \mathcal{R}_0^\ell \right. \\ \left. \parallel \ell \in \text{labs} \llbracket s \rrbracket \text{ ? } \widehat{\mathcal{F}}^{\bowtie} \llbracket s \rrbracket (\widehat{\mathcal{F}}^{\bowtie} \llbracket sl' \rrbracket \mathcal{R}_0^{\text{at} \llbracket s \rrbracket})^\ell \right. \\ \left. \circ \perp^{\bowtie} \right) \quad (19.5)$$

- *Abstract semantics of an empty statement list $sl ::= \epsilon$*

$$\widehat{\mathcal{F}}^{\bowtie} \llbracket sl \rrbracket \mathcal{R}_0^\ell \triangleq \left(\ell = \text{at} \llbracket sl \rrbracket \text{ ? } \mathcal{R}_0 \circ \perp^{\bowtie} \right) \quad (19.6)$$

- *Abstract semantics of an assignment statement $S ::= x = A ;$*

$$\widehat{\mathcal{S}}^{\bowtie} \llbracket S \rrbracket \mathcal{R}_0^{\ell} = \left(\begin{array}{l} \ell = \text{at} \llbracket S \rrbracket \text{ ? } \mathcal{R}_0 \\ \parallel \ell = \text{after} \llbracket S \rrbracket \text{ ? } \text{assign}^{\bowtie} \llbracket x, A \rrbracket \mathcal{R}_0 \\ \vdots \perp^{\bowtie} \end{array} \right) \quad (19.7)$$

where $\text{assign} \llbracket x, A \rrbracket \circ \gamma \sqsubseteq \gamma \circ \text{assign}^{\bowtie} \llbracket x, A \rrbracket$.

- *Abstract semantics of a conditional statement $S ::= \text{if } (B) S_t$*

$$\begin{aligned} \widehat{\mathcal{S}}^\bowtie \llbracket S \rrbracket \mathcal{R}_0^\ell = & \left(\ell = \text{at} \llbracket S \rrbracket \text{ ? } \mathcal{R}_0 \right. \\ & \left\| \ell \in \text{in} \llbracket S_t \rrbracket \text{ ? } \widehat{\mathcal{S}}^\bowtie \llbracket S_t \rrbracket (\text{test}^\bowtie \llbracket B \rrbracket \mathcal{R}_0)^\ell \right. \\ & \left\| \ell = \text{after} \llbracket S \rrbracket \text{ ? } \right. \\ & \quad \widehat{\mathcal{S}}^\bowtie \llbracket S_t \rrbracket (\text{test}^\bowtie \llbracket B \rrbracket \mathcal{R}_0)^\ell \sqcup^\bowtie \overline{\text{test}}^\bowtie \llbracket B \rrbracket \mathcal{R}_0 \\ & \left. : \perp^\bowtie \right) \end{aligned} \tag{19.9}$$

where $\text{test} \llbracket B \rrbracket \circ \gamma \sqsubseteq \gamma \circ \text{test}^\bowtie \llbracket B \rrbracket$ and $\overline{\text{test}} \llbracket B \rrbracket \circ \gamma \sqsubseteq \gamma \circ \overline{\text{test}}^\bowtie \llbracket B \rrbracket$.

- *Abstract semantics of an iteration statement $S ::= \text{while}^\ell(B) S_b$*

$$\widehat{\mathcal{F}}^\bowtie \llbracket S \rrbracket \mathcal{R}_0 \ell' = \text{lfp}^{\leq \bowtie} (\mathcal{F}^\bowtie \llbracket \text{while}^\ell(B) S_b \rrbracket \mathcal{R}_0) \ell' \quad (19.11)$$

$$\mathcal{F}^\bowtie \llbracket \text{while}^\ell(B) S_b \rrbracket \in \mathbb{P}^\bowtie \rightarrow ((\mathbb{L} \rightarrow \mathbb{P}^\bowtie) \rightarrow (\mathbb{L} \rightarrow \mathbb{P}^\bowtie))$$

$$\begin{aligned} \mathcal{F}^\bowtie \llbracket \text{while}^\ell(B) S_b \rrbracket \mathcal{R}_0 X \ell' = & \\ & (\ell' = \ell \text{ ? } \mathcal{R}_0 \sqcup^\bowtie \widehat{\mathcal{F}}^\bowtie \llbracket S_b \rrbracket (\text{test}^\bowtie \llbracket B \rrbracket X(\ell)) \ell \\ & \parallel \ell' \in \text{in} \llbracket S_b \rrbracket \setminus \{\ell\} \text{ ? } \widehat{\mathcal{F}}^\bowtie \llbracket S_b \rrbracket (\text{test}^\bowtie \llbracket B \rrbracket X(\ell)) \ell' \\ & \parallel \ell' = \text{after} \llbracket S \rrbracket \text{ ? } \overline{\text{test}}^\bowtie \llbracket B \rrbracket X(\ell) \sqcup^\bowtie \bigsqcup_{\ell'' \in \text{breaks-of} \llbracket S_b \rrbracket} \widehat{\mathcal{F}}^\bowtie \llbracket S_b \rrbracket (\text{test}^\bowtie \llbracket B \rrbracket X(\ell)) \ell'' \\ & \text{ : } \perp^\bowtie) \end{aligned}$$

- *Abstract semantics of a break statement* $S ::= \ell \text{ break } ;$

$$\widehat{\mathcal{S}}^{\bowtie} \llbracket S \rrbracket \mathcal{R}_0^{\ell} = (\ell = \text{at} \llbracket S \rrbracket \text{ ? } \mathcal{R}_0 \circ \perp^{\bowtie}) \quad (19.12)$$

Proof methods

Invariance proof methods

- Invariance proof methods derive from the reachability semantics
 - abstraction to verification conditions \rightarrow Turing/Floyd/Naur proof method
 - abstraction to Hoare triples \rightarrow Hoare logic
 - Fixpoints:

Theorem (22.1, Fixpoint induction) Let $f \in \mathcal{L} \xrightarrow{\sqsubseteq} \mathcal{L}$ be an increasing function on a complete lattice $\langle \mathcal{L}, \sqsubseteq, \perp, \top, \sqcap, \sqcup \rangle$ and $P \in \mathcal{L}$. We have $\text{lfp}^{\sqsubseteq} f \sqsubseteq P \Leftrightarrow \exists I \in \mathcal{L} . f(I) \sqsubseteq I \wedge I \sqsubseteq P$.

Bibliography on verification and proofs

References I

- Alglave, Jade and Patrick Cousot (2017). “Ogre and Pythia: an invariance proof method for weak consistency models”. In: *POPL. ACM*, pp. 3–18.
- Cousot, Patrick (1990). “Methods and Logics for Proving Programs”. In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*. MIT Press Cambridge, MA, USA ©1990, pp. 841–994.
- (2002). “Constructive design of a hierarchy of semantics of a transition system by abstract interpretation”. *Theor. Comput. Sci.* 277.1-2, pp. 47–103.
 - (2003). “Verification by Abstract Interpretation”. In: *Verification: Theory and Practice*. Vol. 2772. Lecture Notes in Computer Science. Springer, pp. 243–268.
- Cousot, Patrick, Roberto Giacobazzi, and Francesco Ranzato (2018). “Program Analysis Is Harder Than Verification: A Computability Perspective”. In: *CAV (2)*. Vol. 10982. Lecture Notes in Computer Science. Springer, pp. 75–95.
- Floyd, Robert W. (1967). “Assigning meaning to programs”. In: J.T. Schwartz, ed. *Proc. Symp. in Applied Math.* Vol. 19. Amer. Math. Soc., pp. 19–32.

References II

Hoare, C. A. R. (1978). “Some Properties of Predicate Transformers”. *J. ACM* 25.3, pp. 461–480.

Naur, Peter (1966). “Proofs of algorithms by general snapshots”. *BIT* 6, pp. 310–316.

The End of Part 3

Part 4

Symbolic abstraction: dependency analysis

Motivation

Dependency

Found in many reasonings on programs:

- Non-interference (confidentiality, integrity)
- Security, privacy
- Program slicing
- Temporal dependencies in synchronous languages (Esterelle, Lustre, Signal, ... called causality there)
- etc.

Dependency

The existing definitions

- are given a priori (e.g. Cheney, Ahmed, and Acar, 2011; D. E. Denning and P. J. Denning, 1977),
- without semantics justification (except Assaf, Naumann, Signoles, Totel, and Tronel, 2017 (“hyper-collecting semantics”), Urban and Müller, 2018)
- are dependencies on program exit only

Our objective is to study principles, not to get a new powerful dependency analysis

Dependency, informally

Functional dependency

- A function $f(\dots, x, \dots)$ depends on its parameter x if and only if changing only this parameter changes the result

$$\exists x_1, x_2 . f(\dots, x_1, \dots) \neq f(\dots, x_2, \dots)$$

- Example: $f(x, y) = x - (y - y)$ depends on x but not on y
- Definition:

$$\begin{aligned} \mathcal{F}d^{ni} &\triangleq \{ f \mid \exists x_1, \dots, x_n, x_i' . f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) \neq \\ &\quad f(x_1, \dots, x_{i-1}, x_i', x_{i+1}, \dots, x_n) \}. \end{aligned} \quad (44.1)$$
$$\mathcal{F}d \triangleq \bigcup_{n \in \mathbb{N}_*} \bigcup_{1 \leq i \leq n} \mathcal{F}d^{ni}$$

Non-interference

- Given low variables L (e.g. “public” respectively “untainted”) and high variables H (“private/conf” respectively “tainted”)
- Non-interference (Cohen, 1977; Goguen and Meseguer, 1982, 1984; Mantel, 2003) is defined as “if executions start with the same values of the low variables then, upon termination, if ever, the low variables are equal (so changing initial high variables cannot change final low variables)”
- The non-interference property is therefore

$$\begin{aligned} Ni(L, H) = \{ \Pi \in \wp(\mathbb{T}^+ \times \mathbb{T}^\infty) \mid \forall \langle \pi_0, \pi \rangle, \langle \pi'_0, \pi' \rangle \in \Pi \cap (\mathbb{T}^+ \times \mathbb{T}^+) . \\ (\forall x \in L . \rho(\pi_0)x = \rho(\pi'_0)x) \Rightarrow (\forall x \in L . \rho(\pi_0 \smallfrown \pi)x = \rho(\pi'_0 \smallfrown \pi')x) \} \end{aligned}$$

- Interference during the computation and non termination are not taken into account.

General idea of dependency

- y depends on the initial value x_0 of x at ℓ if and only if changing x_0 changes the future observations of y at ℓ
 - We consider dependency on initial values of variables
- More generally, changing an abstraction of the past at ℓ changes an abstraction of the future after ℓ

Dependency is local

- $\ell_1 \ y = 0 ; \ell_2 \ y = x ; \ell_3$
 - the value of y at ℓ_1 is the initial value y_0 of y at ℓ_1
Changing the initial value of x does not change the value of y at ℓ_1 so y does not depend on the initial value of x at ℓ_1
 - the value of y at ℓ_2 is 0.
Changing the initial value of x does not change the value of y at ℓ_2 so y does not depend on the initial value of x at ℓ_2
 - the value of y at ℓ_3 is the initial value x_0 of x .
Changing the initial value of x changes the value of y at ℓ_3 so y depends on the initial value of x at ℓ_3

⇒ dependency upon the initial value of variables is local (may be different at different program points).

Dependency depends on values of variables

`{if (x=0) y=x; else y=0;} ℓ`

- The value of y at ℓ is always 0, no dependency

`{if (x=0) y=x; else y=1;} ℓ`

- The value of y at ℓ is
 - if $x_0 = 0$ then “0”
 - if $x_0 \neq 0$ then “1”
- y at ℓ depends on x_0 (unless $(x_0 = 0 \wedge y_0 = 0) \vee (x_0 \neq 0 \wedge y_0 = 1)$)

⇒ dependency of y upon the initial value x_0 of x depends on the initial and current values of x and y

⇒ this is ignored in D. E. Denning and P. J. Denning, 1977's dataflow analysis

Dependency depends on sequences of observations of values of variables

$$P_u \triangleq \text{while } \ell \ (0 \neq 0) \ x = x + 1;$$

- One can observe $x_0 \cdot x_0 + 1 \cdot x_0 + 2 \cdot x_0 + 17 \cdot x_0 + 18 \cdot \dots x_0 + 42 \cdot x_0 + 43 \cdot \dots$ at ℓ
- changing the initial value x_0 of x changes this observation
- x at ℓ depends upon x_0

$$P_0 \triangleq x = 0; \text{ while } \ell \ (0 \neq 0) \ x = x + 1;$$

- One can observe $0 \cdot 1 \cdot 2 \cdot \dots 17 \cdot 18 \cdot \dots 42 \cdot 43 \cdot \dots$ at ℓ
- changing the initial value x_0 of x does not change this observation
- x at ℓ does not depend upon x_0

⇒ We must observe the maximal sequence of values successively taken by a variable at a program point

Counterfactual dependency: absence of observation

```
int x,y; if (x=0) { y=x; ℓ }
```

- Observation of y at ℓ :
 - if $x_0 = 0$ then “0”
 - if $x_0 \neq 0$ then “” (empty observations: no execution ever reaches ℓ)

⇒ Dependency if empty observations are taken into account

⇒ No dependency if empty observations are not taken into account

⇒ The choice is completely arbitrary!

Counterfactual value dependency: absence of observation

```
int x,y,z; if (x=0) { y=x; ℓ }
```

- Assume that empty observations are taken into account (so y depends on x_0)
 - Observation of z at ℓ :
 - if $x_0 = 0$ then “ z_0 ” (initial value of z)
 - if $x_0 \neq 0$ then “” (empty observations: no execution ever reaches ℓ)
 - Two different observations at ℓ !
 - Should z depends on x_0 at ℓ ?
- ⇒ The choice is completely arbitrary!
- No
 - Yes
 - Yes if the value of z at ℓ is different from z_0 (D. E. Denning and P. J. Denning, 1977)

Timing dependency

`whileℓ (x > 0) x = x - 1 ;`

- Does variable y (s.t. $y \neq x$) at ℓ depends on the initial value x_0 of x ?
 - The observation of y at ℓ is $y_0 \cdot y_0 \cdot \dots \cdot y_0$ repeated $x_0 + 1$ times.
 - So changing x_0 changes the observation of y at ℓ
- ⇒ This is a *covert/side channel* (Lampson, 1973; Mulder, Eisenbarth, and Schaumont, 2018), more precisely, a *timing channel* (Russo, Hughes, Naumann, and Sabelfeld, 2006; Sabelfeld and Myers, 2003)
- ⇒ The choice of ignoring timing channel is arbitrary
- ⇒ Ignored in the classical definition of dependency D. E. Denning and P. J. Denning, 1977
- ⇒ One way of ignoring timing channels is to require that observation sequences must differ by at least one data

Counterfactual timing dependency

```
/* x {0,1} */ while (x != 0) ℓ y = x ;
```

- If $x_0 = 1$, the infinite sequence of values of y observed at ℓ is $y_0 \cdot 1 \cdot 1 \dots$.
- If $x_0 = 0$, then the observation at ℓ is the empty sequence ϵ .
- Does y at ℓ depends on the initial value x_0 of x ?
- This depends on hypotheses on observables. Is an infinite sequence of values observable? Is the empty sequence ϵ of values observable?
- This is debatable and problem-specific
- For example if a program terminates it is easy to check on program termination that a program point is never reached. This may be considered impossible with non-termination.

Dependency, formally

Future observations

- initialisation trace $\pi_0 \in \mathbb{T}^+$
- (non empty) continuation trace $\pi \in \mathbb{T}^{+\infty}$
- $\text{future}[\![y]\!]^\ell(\pi_0, \pi)$ is the sequence of values of y successively observed at program point ℓ in the trace π continuing π_0 ⁴

$$\text{future}[\![y]\!]^\ell(\pi_0, \ell) \triangleq \rho(\pi_0)y$$

$$\text{future}[\![y]\!]^\ell(\pi_0, \ell') \triangleq \emptyset$$

$$\text{future}[\![y]\!]^\ell(\pi_0, \ell \xrightarrow{a} \ell''\pi) \triangleq \rho(\pi_0)y \cdot \text{future}[\![y]\!]^\ell(\pi_0 \smallfrown \ell \xrightarrow{a} \ell'', \ell''\pi)$$

$$\text{future}[\![y]\!]^\ell(\pi_0, \ell' \xrightarrow{a} \ell''\pi) \triangleq \text{future}[\![y]\!]^\ell(\pi_0 \smallfrown \ell' \xrightarrow{a} \ell'', \ell''\pi)$$

- $\text{future}[\![y]\!]^\ell(\pi_0, \pi)$ is the empty sequence \emptyset if ℓ does not appear in π

⁴this should be understood as a bi-inductive definition of P. Cousot and R. Cousot, 2009 to properly handle non-termination

Observations

- An observation $\langle \underline{v}, \omega \rangle$ of a variable at a program point is a pair of
 - an initial value \underline{v} of the variable
 - the future observation ω of this variable from that program point on

Differences between future observations $\langle \underline{v}, \omega \rangle$ and $\langle \underline{v}', \omega' \rangle$ (I)

(1) Counterfactual timing dependency:

$$\text{ctdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \triangleq \omega \neq \omega'$$

(empty observations are allowed)

(2) Timing dependency:

$$\text{tdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \triangleq \omega \neq \omega' \wedge \omega \neq \exists \wedge \omega' \neq \exists$$

(empty observations are disallowed)

Differences between future observations $\langle \underline{v}, \omega \rangle$ and $\langle \underline{v}', \omega' \rangle$ (II)

(3) Value dependency:

$$\text{vdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \triangleq \exists \omega_0, \omega_1, \omega'_1, v, v' . \\ \omega = \omega_0 \cdot v \cdot \omega_1 \wedge \omega' = \omega_0 \cdot v' \cdot \omega'_1 \wedge v \neq v'$$

(different values of the variable must be observed)

Example 6 `if ℓ_0 ($x == 1$) { ℓ_1 $y = x$; ℓ_2 } ℓ_3`

y does not depend on x at ℓ_0 , ℓ_1 , and ℓ_2 but y depends on x at ℓ_3 (unless $y = 1$ at ℓ_0).

□

Differences between future observations $\langle \underline{v}, \omega \rangle$ and $\langle \underline{v}', \omega' \rangle$ (III)

(4) counterfactual value dependency:

$$\text{cvdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \triangleq \text{vdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \vee (\omega = \exists \wedge \omega' \neq \exists) \vee (\omega \neq \exists \wedge \omega' = \exists)$$

(an empty observation is allowed)

Example 7 $\text{if } \ell_0 (x == 1) \{ \ell_1 y = x ; \ell_2 \} \ell_3$

y depends on x at ℓ_2 (unless $y = 1$ at ℓ_0).

Any variable depends on the initial value of x at ℓ_1 and ℓ_2 . □

Differences between future observations $\langle \underline{v}, \omega \rangle$ and $\langle \underline{v}', \omega' \rangle$ (IV)

(5) Counterfactual multi-values dependency:

$$\begin{aligned} \text{cmvdp}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) &\triangleq \text{vdep}(\langle \underline{v}, \omega \rangle, \langle \underline{v}', \omega' \rangle) \vee \\ &(\omega = \exists \wedge \exists \omega'_0, v', \omega'_1 . \omega' = \omega'_0 \cdot v' \cdot \omega'_1 \wedge \underline{v}' \neq v') \vee \\ &(\omega' = \exists \wedge \exists \omega_0, v, \omega_1 . \omega = \omega_0 \cdot v \cdot \omega_1 \wedge \underline{v} \neq v) \end{aligned}$$

(an empty observation is allowed for variables which value has changed)

Example 8 $\text{if } \ell_0 (x == 1) \{ \ell_1 y = x ; \ell_2 \} \ell_3$

No variable depends on the initial value of x at ℓ_1 and only y at ℓ_2 (unless y is initially 1).

This is D. E. Denning and P. J. Denning, 1977.

□

Formal definition of dependency

- Dependency property:

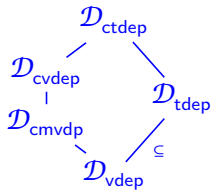
$$\mathcal{D}_{\text{dep}}^{\ell}\langle x, y \rangle \triangleq \{ \Pi \in \wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty}) \mid \exists \langle \pi_0, \pi_1 \rangle, \langle \pi'_0, \pi'_1 \rangle \in \Pi . \\ (\forall z \in \mathbb{V} \setminus \{x\} . \rho(\pi_0)z = \rho(\pi'_0)z) \wedge \\ \text{dep}(\langle \rho(\pi_0)y, \text{future}[[y]]^{\ell}(\pi_0, \pi_1) \rangle, \langle \rho(\pi'_0)y, \text{future}[[y]]^{\ell}(\pi'_0, \pi'_1) \rangle) \}$$

- choose $\text{dep} \in \{\text{vdep}, \text{cmvdp}, \text{cvdep}, \text{tdep}, \text{ctdep}\}$ to get 5 different definitions
- y depends on the initial value of x at point ℓ of program P is:

$$\widehat{\mathcal{S}}^{+\infty}[[P]] \in \mathcal{D}_{\text{dep}}^{\ell}\langle x, y \rangle$$

- No necessary distinction between explicit and implicit flows as in D. E. Denning and P. J. Denning, 1977

Dependency lattice



(??)

- The more differences between observed futures, the more dependencies;
- Not clear with postulated definitions (such as the hydraulic model where dependency depends on the rules to mix colors)

Why maximal traces?

- For prefix traces, if a trace is in the semantics, all of its prefixes are also in the semantics, which introduces artificial timing channels

Prefix traces for dependency on values

- For value dependencies, the maximal trace semantics can be replaced by the prefix trace semantics without problem:

Lemma $\mathcal{S}^{+\infty} \llbracket P \rrbracket \in \mathcal{D}_{\text{vdep}}^{\ell} \langle x, y \rangle \Leftrightarrow \mathcal{S}^* \llbracket P \rrbracket \in \mathcal{D}_{\text{vdep}}^{\ell} \langle x, y \rangle$

- Idem if we include empty observations (the prefixes of $\mathcal{S}^* \llbracket P \rrbracket \pi_0$ are never empty, so no possible confusion)

Dependency abstraction

Abstraction of data dependency

- The abstraction of a semantic property $S \in \wp(\wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty}))$ into a data dependency property $\alpha^{\text{vdep}}(S) \in \mathbb{L} \rightarrow \wp(\mathbb{V} \times \mathbb{V})$ is:

$$\alpha^{\text{vdep}}((S)^\ell \triangleq \{ \langle x, y \rangle \mid S \in \mathcal{D}_{\text{vdep}}^\ell \langle x, y \rangle \}$$

- This is a Galois connection:

Lemma 10 $\langle \wp(\wp(\mathbb{T}^+ \times \mathbb{T}^{+\infty})), \subseteq \rangle \xrightleftharpoons[\alpha^{\text{vdep}}]{\gamma^{\text{vdep}}} \langle \mathbb{L} \rightarrow \wp(\mathbb{V} \times \mathbb{V}), \supseteq^{\text{d}} \rangle$ where the concretization of a dependency property $\mathbf{D} \in \mathbb{L} \rightarrow \wp(\mathbb{V} \times \mathbb{V})$ is:

$$\gamma^{\text{vdep}}(\mathbf{D}) \triangleq \bigcap_{\ell \in \mathbb{L}} \bigcap_{\langle x, y \rangle \in \mathbf{D}(\ell)} \mathcal{D}_{\text{vdep}}^\ell \langle x, y \rangle$$

(the more semantics, the less dependencies)

Value dependency static analysis

Potential value dependency

- $\alpha^{\text{vdep}}(\{\mathcal{S}^{+\infty} \llbracket s \rrbracket\}) = \alpha^{\text{vdep}}(\{\mathcal{S}^* \llbracket s \rrbracket\})$ is not computable (Rice theorem)
- We design an over-approximation:

Potential value dependency semantics $\widehat{\mathcal{S}}_{\exists}^{\text{vdep}}$:

$$\alpha^{\text{vdep}}(\{\mathcal{S}^{+\infty} \llbracket s \rrbracket\}) \subseteq \widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket s \rrbracket$$

- The abstraction of D. E. Denning and P. J. Denning, 1977 is purely syntactic (in dataflow analysis style)
- We do slightly better, by taking values into account, in a very simple way

Example

$\text{if } \ell_0 \ (x == 1) \ \{ \ell_1 \ y = z \ ; \ell_2 \ } ; \ell_3$

- we have the potential value dependency:

ℓ	ℓ_0	ℓ_1	ℓ_2	ℓ_3
$\widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S \rrbracket \ell$	$\{\langle x, x \rangle, \langle y, y \rangle, \langle z, z \rangle\}$	$\{\langle y, y \rangle, \langle z, z \rangle\}$	$\{\langle z, y \rangle, \langle z, z \rangle\}$	$\{\langle x, x \rangle, \langle x, y \rangle, \langle y, y \rangle, \langle z, y \rangle, \langle z, z \rangle\}$

- this is an over-approximation since e.g. z flows to y at ℓ_3 only when $x = 1$ at ℓ_0 .

Computational design

- By calculus (in principle, can be checked with Coq like Jourdan, Laporte, Blazy, Leroy, and Pichardie, 2015)
- By structural induction on the program syntax
- By fixpoint over-approximation for iterations:

Theorem (over-approximation of fixpoints) If $\langle C, \sqsubseteq, \perp, \top, \sqcup, \sqcap \rangle$ and $\langle \mathcal{A}, \preceq, 0, 1, \vee, \wedge \rangle$ are complete lattices, $\langle C, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \mathcal{A}, \preceq \rangle$ is a Galois connection, $f \in C \xrightarrow{\gamma} C$ and $\bar{f} \in \mathcal{A} \xrightarrow{\alpha} \mathcal{A}$ are increasing and $\alpha \circ f \preceq \bar{f} \circ \alpha$ (*semi-commutation*) then $\text{lfp}^{\sqsubseteq} f \sqsubseteq \gamma(\text{lfp}^{\preceq} \bar{f})$.

- Finite domain, no widening needed

Potential dependency semantics of assignment $S ::= x = A ;$

$$\begin{aligned}\widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[[S]] \ell &= (\ell = \text{at}[[S]] \text{ ? } 1_v \\ &\quad \parallel \ell = \text{after}[[S]] \text{ ? } \{\langle y, x \rangle \mid y \in \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[[A]]\} \cup \\ &\quad \{\langle y, y \rangle \mid y \neq x\} \\ &\quad : \emptyset) \\ \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[[A]] &\triangleq \{y \mid \exists \rho \in \mathbb{E}v . \exists v \in \mathbb{V} . \mathcal{A}[[A]]\rho \neq \mathcal{A}[[A]]\rho[y \leftarrow v]\} \\ &\subseteq \text{vars}[[A]]\end{aligned}$$

Example:

- after $x = y - y ;$, x depends on y .
- after $x = y ; x = y - x ;$, x depends on the initial values of x and y
- To be more precise we would have to preserve information on the values of variables (eg. $x = y$)

Proof (don't read ☺) I

The cases $\ell = \text{at}[\![S]\!]$ was handled in (44.38) and $\ell \notin \text{labx}[\![S]\!]$ in (44.39). It remains the case $\ell = \text{after}[\![S]\!]$.

$$\begin{aligned}
 & \alpha^{\text{vdep}}(\{\mathcal{S}^{+\infty}[\![S]\!]\}) \text{ after}[\![S]\!] \\
 = & \alpha^{\text{vdep}}(\{\mathcal{S}^*[\![S]\!]\}) \text{ after}[\![S]\!] && \text{\{Lemma 44.25\}} \\
 = & \{\langle x', y \rangle \mid \mathcal{S}^*[\![S]\!] \in \mathcal{D}_{\text{vdep}}(\text{after}[\![S]\!]) \langle x', y \rangle\} && \text{\{def. (44.29) of } \alpha^{\text{vdep}} \text{ and def. } \subseteq \text{\}} \\
 = & \{\langle x', y \rangle \mid \exists \langle \pi_0, \pi_1 \rangle, \langle \pi'_0, \pi'_1 \rangle \in \mathcal{S}^*[\![S]\!] . \forall z \in \mathcal{V} \setminus \{x'\} . \boldsymbol{\rho}(\pi_0)z = \boldsymbol{\rho}(\pi'_0)z \wedge \text{vdep}(\langle \boldsymbol{\rho}(\pi_0)y, \\
 & \text{future}[\![y]\!](\text{after}[\![S]\!])(\pi_0, \pi_1) \rangle, \langle \boldsymbol{\rho}(\pi'_0)y, \text{future}[\![y]\!](\text{after}[\![S]\!])(\pi'_0, \pi'_1) \rangle) \rangle\} \\
 & && \text{\{def. } \in \text{ and (44.20) of } \mathcal{D}_{\text{vdep}}^\ell \langle x', y \rangle \text{\}} \\
 = & \{\langle x', y \rangle \mid \exists \langle \pi_0, \pi_1 \rangle, \langle \pi'_0, \pi'_1 \rangle \in \{\langle \pi_{\text{at}}[\![S]\!], \text{at}[\![S]\!] \xrightarrow{x=\mathcal{A}[\![A]\!]\boldsymbol{\rho}(\pi_{\text{at}}[\![S]\!])} \text{after}[\![S]\!]\} \mid \pi_{\text{at}}[\![S]\!] \in \\
 & \mathbb{T}^+ \} . \forall z \in \mathcal{V} \setminus \{x'\} . \boldsymbol{\rho}(\pi_0)z = \boldsymbol{\rho}(\pi'_0)z \wedge \text{vdep}(\langle \boldsymbol{\rho}(\pi_0)y, \text{future}[\![y]\!](\text{after}[\![S]\!])(\pi_0, \pi_1) \rangle, \langle \boldsymbol{\rho}(\pi'_0)y, \\
 & \text{future}[\![y]\!](\text{after}[\![S]\!])(\pi'_0, \pi'_1) \rangle) \rangle\} \\
 & && \text{\{def. (15.1) of the assignment prefix finite trace semantics\}}
 \end{aligned}$$

Proof (don't read ☺) II

$$\begin{aligned}
 = & \{ \langle x', y \rangle \mid \exists \langle \pi_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S] \rangle, \langle \pi'_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S] \rangle \\
 & \text{after}[S] \rangle . \quad \forall z \in \mathcal{V} \setminus \{x'\} . \quad \rho(\pi_0 \text{at}[S])z = \rho(\pi'_0 \text{at}[S])z \wedge \\
 & \text{vdep}(\langle \rho(\pi_0)y, \quad \text{future}[y](\text{after}[S])(\pi_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S]) \rangle, \langle \rho(\pi'_0)y, \\
 & \text{future}[y](\text{after}[S])(\pi'_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S]) \rangle) \} \quad \{ \text{def.} \in \}
 \end{aligned}$$

$$\begin{aligned}
 = & \{ \langle x', y \rangle \mid \exists \langle \pi_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S] \rangle, \langle \pi'_0 \text{at}[S], \\
 & \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S] \rangle . (\forall z \in \mathcal{V} \setminus \{x'\} . \rho(\pi_0 \text{at}[S])z = \rho(\pi'_0 \text{at}[S])z) \wedge \text{vdep}(\langle \rho(\pi_0)y, \\
 & \rho(\pi_0 \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S])y \rangle, \langle \rho(\pi'_0)y, \rho(\pi'_0 \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S])y \rangle) \} \\
 & \{ \text{def. (44.14) of the future future}[y] \}
 \end{aligned}$$

$$\begin{aligned}
 = & \{ \langle x', y \rangle \mid \exists \langle \pi_0 \text{at}[S], \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S] \rangle, \langle \pi'_0 \text{at}[S], \\
 & \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S] \rangle . (\forall z \in \mathcal{V} \setminus \{x'\} . \rho(\pi_0 \text{at}[S])z = \rho(\pi'_0 \text{at}[S])z) \wedge \\
 & ((\rho(\pi_0 \text{at}[S])y \neq \rho(\pi'_0 \text{at}[S])y) \vee (\rho(\pi_0 \text{at}[S])y = \rho(\pi'_0 \text{at}[S])y \wedge \rho(\pi_0 \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi_0 \text{at}[S])} \text{after}[S])y \\
 & \text{after}[S])y \neq \rho(\pi'_0 \text{at}[S] \xrightarrow{x=\mathcal{A}[A]\rho(\pi'_0 \text{at}[S])} \text{after}[S])y) \}
 \end{aligned}$$

Proof (don't read ☺) III

(44.18) so that $\text{vdep}(\langle x, a \cdot b \rangle, \langle y, c \cdot d \rangle)$ if and only if (1) $a \neq c$ or (2) $a = c \wedge b \neq d$.

$$\begin{aligned}
 &= \{ \langle x', y \rangle \mid \exists \langle \pi_0 \text{at}[\![S]\!], \text{at}[\![S]\!] \xrightarrow{x=\mathcal{A}[\![A]\!]\rho(\pi_0 \text{at}[\![S]\!])} \text{after}[\![S]\!]}, \langle \pi'_0 \text{at}[\![S]\!], \\
 &\quad \text{at}[\![S]\!] \xrightarrow{x=\mathcal{A}[\![A]\!]\rho(\pi'_0 \text{at}[\![S]\!])} \text{after}[\![S]\!] \rangle . (\forall z \in \mathcal{V} \setminus \{x'\} . \rho(\pi_0 \text{at}[\![S]\!])z = \rho(\pi'_0 \text{at}[\![S]\!])z \wedge ((y = \\
 &\quad x') \vee (y = x \wedge \mathcal{A}[\![A]\!]\rho(\pi_0 \text{at}[\![S]\!]) \neq \mathcal{A}[\![A]\!]\rho(\pi'_0 \text{at}[\![S]\!])))) \} \quad \text{(def. (6.2) of } \rho \text{)} \\
 &\subseteq \{ \langle x', y \rangle \mid ((y = x') \vee (y = x \wedge \exists \rho, v . \mathcal{A}[\![A]\!]\rho \neq \mathcal{A}[\![A]\!]\rho[x' \leftarrow v])) \} \quad (11)
 \end{aligned}$$

(letting $\rho = \rho(\pi_0 \text{at}[\![S]\!])$ and $v = \rho(\pi'_0 \text{at}[\![S]\!])(x')$ so that $\forall z \in \mathcal{V} \setminus \{x'\} . \rho(\pi_0 \text{at}[\![S]\!])z = \rho(\pi'_0 \text{at}[\![S]\!])z$ implies that $\rho(\pi'_0 \text{at}[\![S]\!]) = \rho[x' \leftarrow v]$.)

$$= \{ \langle x', x' \rangle \mid x' \neq x \} \cup \{ \langle x', x \rangle \mid \exists \rho, v . \mathcal{A}[\![A]\!]\rho \neq \mathcal{A}[\![A]\!]\rho[x' \leftarrow v] \} \quad \text{(case analysis)}$$

$$= \{ \langle x', x' \rangle \mid x' \neq x \} \cup \{ \langle x', x \rangle \mid x' \in \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[\![A]\!] \}$$

(by defining the functional dependency of an expression A as $\widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[\![A]\!] \triangleq \{x' \mid \exists \rho, v . \mathcal{A}[\![A]\!]\rho \neq \mathcal{A}[\![A]\!]\rho[x' \leftarrow v]\}$ in (44.41)) □

Potential dependency semantics of the conditional $S ::= \text{if } (B) S_t$

$$\widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S \rrbracket \ell = (\ell = \text{at} \llbracket S \rrbracket \text{ ? } 1_V \quad (a)$$

$$\parallel \ell \in \text{in} \llbracket S_t \rrbracket \text{ ? } \widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S_t \rrbracket \ell \parallel \text{nondet}(B, B) \quad (b)$$

$$\parallel \ell = \text{after} \llbracket S \rrbracket \text{ ? } \widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S_t \rrbracket \text{after} \llbracket S_t \rrbracket \parallel \text{nondet}(B, B) \quad (c.1)$$

$$\cup 1_V \parallel \text{nondet}(\neg B, \neg B) \quad (c.2)$$

$$\cup \text{nondet}(\neg B, \neg B) \times \text{mod} \llbracket S_t \rrbracket \quad (c.3)$$

$$: \emptyset) \quad (d)$$

$$\text{det}(B_1, B_2) \subseteq \{x \mid \forall \rho, \rho'. (\mathcal{B} \llbracket B_1 \rrbracket \rho \wedge \mathcal{B} \llbracket B_2 \rrbracket \rho') \Rightarrow (\rho(x) = \rho'(x))\} \quad \text{determinacy}$$

$$\text{nondet}(B_1, B_2) \supseteq V \setminus \text{det}(B_1, B_2) \quad \text{non-determinacy}$$

$$\text{mod} \llbracket x = E ; \rrbracket \triangleq \{x\} \quad \text{modified variables}$$

$$\text{mod} \llbracket ; \rrbracket \triangleq \text{mod} \llbracket \epsilon \rrbracket \triangleq \text{mod} \llbracket \text{break} ; \rrbracket \triangleq \emptyset$$

$$\text{mod} \llbracket \text{while } (B) S \rrbracket = \text{mod} \llbracket \text{if } (B) S \rrbracket \triangleq \text{mod} \llbracket S \rrbracket$$

$$\text{mod} \llbracket \text{if } (B) S_t \text{ else } S_f \rrbracket \triangleq \text{mod} \llbracket S_t \rrbracket \cup \text{mod} \llbracket S_f \rrbracket$$

$$\text{mod} \llbracket \{ S_l \} \rrbracket \triangleq \text{mod} \llbracket S_l \rrbracket$$

$$\text{mod} \llbracket S_l S \rrbracket \triangleq \text{mod} \llbracket S_l \rrbracket \cup \text{mod} \llbracket S \rrbracket$$

- On entry (a), variables in \mathcal{V} only depend upon themselves as specified by the identity relation $\mathbb{1}_{\mathcal{V}}$.
- The reasoning in (b) is that if a variable y depends at ℓ on the initial value of a variable x at $\text{at}[\llbracket S_t \rrbracket]$, it depends in the same way on that initial value of the variable x at $\text{at}[\llbracket S \rrbracket]$ since the test B has no side effect.
However, (b) also takes into account that if S_t can only be reached for a unique value of the variable x and the branch is not taken for all other values of x then the variable y does not depend on x in S_t since empty observations are disallowed by vdep .
- (c) determines dependencies after S so compare two possible executions of that statement. In case (c.1) both executions go through the true branch. In case (c.2) both executions go through the false branch, while in case (c.3) the executions take different branches.

- In case (c.1) when the test is true tt for both executions, the executions of the true branch S_t terminate and control after S_t reaches the program point after S (recall that $\text{after}[\llbracket S_t \rrbracket] = \text{after}[\llbracket S \rrbracket]$). The dependencies after S_t propagate after S but only in case of non-determinism, e.g. for variables that are not constant.
- The second case in (c.2) is for those executions for which the test B is false ff . Variables depend on themselves at $\llbracket S \rrbracket$ and control moves to $\text{after}[\llbracket S \rrbracket]$ so that dependencies are the same there, but only for variables that can reach $\text{after}[\llbracket S \rrbracket]$ with different values on different executions as indicated by the restriction to $\text{nondet}(\neg B, \neg B)$.
- The third case in (c.3) is for pairs of executions, one through the true branch and the other through the false branch. In that case y depends on x only if x does not force execution to always take the same branch, meaning that $x \in \text{nondet}(\neg B, \neg B)$. If y is not modified by the execution through S_t then its value after S is always the same as its value at $\llbracket S \rrbracket$ (since y is not modified on the false branch either). In that case changing y at $\llbracket S \rrbracket$ would not change y after S so that, in that situation, y does not depend on x . Therefore (c.3) requires that $y \in \text{mod}[\llbracket S_t \rrbracket]$.

Note on the potential dependency semantics of the conditional

$$S ::= \text{if } (B) S_t$$

- Empty observations are not taken into account
- $\ell_0 \text{ if } (x=0) \{ y=x; \ell_1 \} \ell_2$
 - y does not depend on x at ℓ_0 neither at ℓ_1
 - y depends on x at ℓ_2
- As already stated, this is different from D. E. Denning and P. J. Denning, 1977 implicitly allowing for counterfactual multi-values dependency [cmvdp](#).

Potential dependency semantics of the statement list $sl ::= sl' \ s$

$$\widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[sl] \ell \triangleq \left(\ell \in \text{labx}[sl'] \ ? \ \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[sl'] \ell \right. \quad (a)$$

$$\left. \begin{array}{l} \ell \in \text{labx}[s] \setminus \{\text{at}[s]\} \ ? \\ \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[sl'] \text{at}[s] \ ; \ \widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[s] \ell \\ \circ \emptyset \end{array} \right) \quad (b)$$

Potential dependency semantics of the iteration $S ::= \text{while}^\ell(B) S_b$

$$\widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S \rrbracket^{\ell'} = (\text{lfp}^{\subseteq} \mathcal{F}^{\text{vdep}} \llbracket \text{while}^\ell(B) S_b \rrbracket)^{\ell'}$$

$$\begin{aligned} \mathcal{F}^{\text{vdep}} \llbracket \text{while}^\ell(B) S_b \rrbracket X^{\ell'} = & \\ & \left(\begin{array}{l} \ell' = \ell \text{ ?} \\ \mathbb{1}_V \cup (X(\ell) \circ (\widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S_b \rrbracket^{\ell} \mid \text{nondet}(B, B))) \end{array} \right) \end{aligned} \quad (a)$$

$$\begin{aligned} & \left(\begin{array}{l} \ell' \in \text{in} \llbracket S_b \rrbracket \text{ ?} \\ X(\ell) \circ (\widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S_b \rrbracket^{\ell'} \mid \text{nondet}(B, B)) \end{array} \right) \end{aligned} \quad (b)$$

$$\begin{aligned} & \left(\begin{array}{l} \ell' = \text{after} \llbracket S \rrbracket \text{ ?} \\ X(\ell) \cup (X(\ell) \circ (V \times \text{mod} \llbracket S_b \rrbracket)) \cup \end{array} \right) \end{aligned} \quad (c)$$

$$\begin{aligned} & X(\ell) \circ \left(\left(\bigcup_{\ell'' \in \text{breaks-of} \llbracket S_b \rrbracket} \widehat{\mathcal{S}}_{\exists}^{\text{vdep}} \llbracket S_b \rrbracket^{\ell''} \right) \mid \text{nondet}(B, B) \right) \\ & \circ \emptyset \end{aligned} \quad (d)$$

Example

$S = \text{while } \ell_0 (\text{tt}) \{ \ell_1 y = z ; \ell_2 z = x ; \} \ell_3.$

The system of equations $X = \mathcal{F}^d[S](X)$ is

$$\begin{cases} X(\ell_0) &= \{ \langle v, v \rangle \mid v \in \mathbb{V} \} \cup (X(\ell_2) \circ \{ \langle x, x \rangle, \langle x, z \rangle, \langle y, y \rangle \}) \\ X(\ell_1) &= X(\ell_0) \\ X(\ell_2) &= X(\ell_2) \cup (X(\ell_1) \circ \{ \langle x, x \rangle, \langle z, y \rangle, \langle z, z \rangle \}) \\ X(\ell_3) &= \emptyset \end{cases}$$

The chaotic iterations are

ℓ	ℓ_0, ℓ_1	ℓ_2	ℓ_3
$X^0(\ell)$	\emptyset	\emptyset	\emptyset
$X^1(\ell)$	$\{ \langle x, x \rangle, \langle y, y \rangle, \langle z, z \rangle \}$	$\{ \langle x, x \rangle, \langle z, y \rangle, \langle z, z \rangle \}$	\emptyset
$X^2(\ell)$	$\{ \langle x, x \rangle, \langle x, z \rangle, \langle y, y \rangle, \langle z, y \rangle, \langle z, z \rangle \}$	$\{ \langle x, x \rangle, \langle x, y \rangle, \langle x, z \rangle, \langle z, y \rangle, \langle z, z \rangle \}$	\emptyset
$X^3(\ell)$	$\{ \langle x, x \rangle, \langle x, y \rangle, \langle x, z \rangle, \langle y, y \rangle, \langle z, y \rangle, \langle z, z \rangle \}$	$\{ \langle x, x \rangle, \langle x, y \rangle, \langle x, z \rangle, \langle z, y \rangle, \langle z, z \rangle \}$	\emptyset
$X^4(\ell)$	$X^3(\ell_0) = X^3(\ell_1)$	$X^3(\ell_2)$	\emptyset

- The initial value x_0 of x flows to x at ℓ_0 on iteration entry, to z after the first iteration and so to y after the first iteration.
- The initial value y_0 of y flows only to y at ℓ_0 on iteration entry.
- The initial value z_0 of z flows to z at ℓ_0 on iteration entry and then to y after the first iteration.

The potential dependency semantics is not purely structural ⁵

- Separate analysis of statements:

$\ell_0 \ y = x ;$ x and y at ℓ_1 depend on x at ℓ_0 .
 ℓ_1

$\ell_1 \ y = y - x ;$ x and y at ℓ_2 depend on x at ℓ_1 .
 ℓ_2

- Dependency analysis of the statement list:

$\ell_0 \ y = x ;$
 $\ell_1 \ y = y - x ;$ y at ℓ_2 depends on x at ℓ_1 which depends on x at ℓ_0 so,
 ℓ_2 by composition, y at ℓ_2 depends on x at ℓ_0 .

- Yet, $y = 0$ at ℓ_2 and so y at ℓ_2 do not depend on x at ℓ_0 .
- A purely syntactic structural definition of dependency like $\widehat{\mathcal{S}}_{\exists}^{\text{vdep}}[[S]]$ is necessarily imprecise (since values of variables are not taken into account)

⁵one would say compositional in denotational semantics.

Improving precision

- To be more precise, values of variables must be taken into account
- Reduced product with a reachability analysis (for example Cortesi, Ferrara, Halder, and Zanioli, 2018; Zanioli and Cortesi, 2011)

Examples of derived dependency semantics and analyzes

Dye instrumented semantics

Postulated definition of dependency (I)

- **dye-tracer tests in hydrology**: determine the possible origins of spring discharges or resurgences by water source coloring and flow tracing
- **dye instrumented semantics**: decorate the initial values of variables with labels such as color annotations and to track their diffusion and mixtures to determine dependencies Cheney, Ahmed, and Acar, 2011.

Postulated definition of dependency (II)

- This postulated definition of dependency can be proved sound by observing that the initial color of variables can be designated by the name of these variables and that the color mix at point ℓ for variable y is

$$\{x \mid \mathcal{S}^{+\infty}[[P]] \in \mathcal{D}_{\text{dep}}^{\ell}\langle x, y \rangle\}$$

- Note that in the postulated instrumented semantics, the choice of **dep** remains implicit as defined by the arbitrarily selected color mixing rules.
- Like all **instrumented semantics** Jones and Nielson, 1995, it must be semantically justified with respect to the non-instrumented semantics, in which case the non-instrumented semantics can be used as well to justify dependency, as we do.

Tracking analysis

- Assume the initial values of variables (more generally inputs) are partitioned into tracked \mathcal{T} and untracked \mathcal{U} variables,

$$\mathcal{V} = \mathcal{T} \cup \mathcal{U} \text{ and } \mathcal{T} \cap \mathcal{U} = \emptyset$$

- The tracking abstraction $\alpha^\tau(\mathbf{D})$ of a dependency property $\mathbf{D} \in \mathbb{L} \rightarrow \wp(\mathcal{V} \times \mathcal{V})$ attaches to each program point ℓ the set of variables y which, at that program point ℓ , depend upon the initial value of at least one tracked variable $x \in \mathcal{T}$.

$$\alpha^\tau(\mathbf{D})^\ell \triangleq \{y \mid \exists x \in \mathcal{T} . \langle x, y \rangle \in \mathbf{D}(\ell)\}$$

- A tracking analysis is an over-approximation of the abstract tracking semantics

$$\mathcal{S}^\tau \llbracket s \rrbracket \supseteq \alpha^\tau(\alpha^{\text{dep}}(\{\mathcal{S}^{+\infty} \llbracket s \rrbracket\}))$$

assigning the each program point ℓ , a set $\mathcal{S}^\tau \llbracket s \rrbracket^\ell \in \wp(\mathcal{V})$ of variables potentially depending on tracked variables.

Examples of tracking analyses

- **taint analysis** in privacy/security checks Ferrara, Olivieri, and Spoto, 2018; Li, Bissyandé, Papadakis, Rasthofer, Bartel, Octeau, Klein, and Traon, 2017 (tracked is tainted, untracked is untainted);
- **binding time analysis** in offline partial evaluation Hatcliff, 1998; Jones, Sestoft, and Søndergaard, 1989 (tracked is dynamic, untracked is static)
- **absence of interference** Bowman and Ahmed, 2015; Cohen, 1977; Goguen and Meseguer, 1982, 1984; Volpano, Irvine, and Smith, 1996 (tracked is high (private/untrusted), untracked is low (public/trusted)).

Conclusion

Dependency is an abstract interpretation of the program semantics

- Dependency analysis is an abstract interpretation of the program semantics
- This include non-interference, “taint” analysis, *etc.*
- Data dependency analysis to detect parallelism in sequential codes Padua and Wolfe, 1986 is also an abstract interpretation Tzolovski, 1997, Tzolovski, 2002, Ch. 5.
- We have considered particular cases of dependency.

Conjecture: all dependencies are abstract interpretations

- The semantics is a set of computations $\langle \pi^\ell, {}^\ell\pi' \rangle$ (where $\ell \notin \pi$).
- We define an abstraction of the past π^ℓ (the initial state in our case)
- We define an abstraction of the future (the sequence of values of a variable y observées dans ${}^\ell\pi'$ à each point ℓ dans ${}^\ell\pi'$).
- We define a difference on pasts (changing the value of only one variable in our case)
- We define a difference on futures ($tdep$, $ctdep$, $vdep$ or $cvdep$ in our case)
- Dependency is then the future abstraction depends on the past abstraction iff a change of the past changing its abstraction change the abstraction of the future.
- By varying abstractions and the difference we change the notions of dependency (and we should be able to recover the whole literature in that way).
- Good examples are Giacobazzi and Mastroeni, 2018 for non-interference and Barthe, Grégoire, and Laporte, 2017 for the protection against side channels attacks

Bibliography on dependency

References I

- Assaf, Mounir, David A. Naumann, Julien Signoles, Eric Totel, and Frédéric Tronel (2017). “Hypercollecting semantics and its application to static analysis of information flow”. In: *POPL*. ACM, pp. 874–887 (176).
- Barthe, Gilles, Benjamin Grégoire, and Vincent Laporte (2017). “Provably secure compilation of side-channel countermeasures”. *IACR Cryptology ePrint Archive* 2017, p. 1233 (227).
- Bowman, William J. and Amal Ahmed (2015). “Noninterference for free”. In: *ICFP*. ACM, pp. 101–113 (224).
- Cheney, James, Amal Ahmed, and Umut A. Acar (2011). “Provenance as dependency analysis”. *Mathematical Structures in Computer Science* 21.6, pp. 1301–1337 (176, 220).
- Cohen, Ellis S. (1977). “Information Transmission in Computational Systems”. In: *SOSP*. ACM, pp. 133–139 (179, 224).

References II

- Cortesi, Agostino, Pietro Ferrara, Raju Halder, and Matteo Zanioli (2018). “Combining Symbolic and Numerical Domains for Information Leakage Analysis”. *Trans. Computational Science* 31, pp. 98–135 (217).
- Cousot, Patrick and Radhia Cousot (2009). “Bi-inductive structural semantics”. *Inf. Comput.* 207.2, pp. 258–283 (189).
- Denning, Dorothy E. and Peter J. Denning (1977). “Certification of Programs for Secure Information Flow”. *Commun. ACM* 20.7, pp. 504–513 (176, 182, 185, 186, 194, 195, 202, 212).
- Ferrara, Pietro, Luca Olivieri, and Fausto Spoto (June 2018). “Tailoring Taint Analysis to GDPR”. In: *Privacy Technologies and Policy*. 6th Annual Privacy Forum, APF 2018, Barcelona, Spain, June 13-14, 2018, Revised Selected Papers. DOI: 10.1007/978-3-030-02547-2_4 (224).
- Giacobazzi, Roberto and Isabella Mastroeni (2018). “Abstract Non-Interference: A Unifying Framework for Weakening Information-flow”. *ACM Trans. Priv. Secur.* 21.2, 9:1–9:31 (227).

References III

- Goguen, Joseph A. and José Meseguer (1982). “Security Policies and Security Models”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, pp. 11–20 (179, 224).
- (1984). “Unwinding and Inference Control”. In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, pp. 75–87 (179, 224).
- Hatcliff, John (1998). “An Introduction to Online and Offline Partial Evaluation using a Simple Flowchart Language”. In: *Partial Evaluation*. Vol. 1706. Lecture Notes in Computer Science. Springer, pp. 20–82 (224).
- Jones, Neil D. and Flemming Nielson (1995). “Abstract Interpretation: a Semantics-Based Tool for Program Analysis”. In: Samson Abramsky and Dov M. Gabbay, eds. *Handbook of Logic in Computer Science*. Vol. 4, Semantic Modelling. Oxford University Press, pp. 527–636 (221).
- Jones, Neil D., Peter Sestoft, and Harald Søndergaard (1989). “Mix: A Self-Applicable Partial Evaluator for Experiments in Compiler Generation”. *Lisp and Symbolic Computation* 2.1, pp. 9–50 (224).

References IV

- Jourdan, Jacques-Henri, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie (2015). “A Formally-Verified C Static Analyzer”. In: *POPL. ACM*, pp. 247–259 (204).
- Lampson, Butler W. (1973). “A Note on the Confinement Problem”. *Commun. ACM* 16.10, pp. 613–615 (186).
- Li, Li, Tegawendé F. Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Ochteau, Jacques Klein, and Yves Le Traon (2017). “Static analysis of Android apps: A systematic literature review”. *Information & Software Technology* 88, pp. 67–95 (224).
- Mantel, Heiko (July 2003). “A Uniform Framework for the Formal Specification and Verification of Information Flow Security”. *Dr.-Ing. Thesis. Saarbrücken, Germany: Fakultät I der Universität des Saarlandes* (179).
- Mulder, Elke De, Thomas Eisenbarth, and Patrick Schaumont (2018). “Identifying and Eliminating Side-Channel Leaks in Programmable Systems”. *IEEE Design & Test* 35.1, pp. 74–89 (186).

References V

- Padua, David A. and Michael Wolfe (1986). “Advanced Compiler Optimizations for Supercomputers”. *Commun. ACM* 29.12, pp. 1184–1201 (226).
- Russo, Alejandro, John Hughes, David A. Naumann, and Andrei Sabelfeld (2006). “Closing Internal Timing Channels by Transformation”. In: *ASIAC*. Vol. 4435. Lecture Notes in Computer Science. Springer, pp. 120–135 (186).
- Sabelfeld, Andrei and Andrew C. Myers (2003). “Language-based information-flow security”. *IEEE Journal on Selected Areas in Communications* 21.1, pp. 5–19 (186).
- Tzolovski, Stanislav (1997). “Data Dependence as Abstract Interpretations”. In: *SAS*. Vol. 1302. Lecture Notes in Computer Science. Springer, p. 366 (226).
- (15 June 2002). “Raffinement d’analyses par interprétation abstraite”. *Thèse de doctorat*. Palaiseau, France: École polytechnique (226).
- Urban, Caterina and Peter Müller (2018). “An Abstract Interpretation Framework for Input Data Usage”. In: *ESOP*. Vol. 10801. Lecture Notes in Computer Science. Springer, pp. 683–710 (176).

References VI

- Volpano, Dennis M., Cynthia E. Irvine, and Geoffrey Smith (1996). “A Sound Type System for Secure Flow Analysis”. *Journal of Computer Security* 4.2/3, pp. 167–188 (224).
- Zanioli, Matteo and Agostino Cortesi (2011). “Information Leakage Analysis by Abstract Interpretation”. In: *SOFSEM*. Vol. 6543. Lecture Notes in Computer Science. Springer, pp. 545–557 (217).

The End, Thank you

Appendix

3 Syntax, semantics, properties, and static analysis of expressions

We introduce basic concepts of abstract interpretation using arithmetic and boolean expressions.

Contents

3.1	The rule of signs	1
3.2	Sign analysis of iterative programs	2
3.3	Sign abstraction, informally	3
3.4	Syntax of expressions	3
3.5	Structural definitions	4
3.6	Environments	4
3.7	Structural semantics of expressions	5
3.8	Proofs by structural induction	5
3.9	Semantic properties of expressions	6
3.10	Collecting semantics of expressions	6
3.11	Proving semantic properties of expressions by structural induction	7
3.12	Abstract sign properties	7
3.13	Structural sign semantics of expressions	8
3.14	Soundness	9
3.15	Sign concretization	9
3.16	Sign lattice	10
3.17	Sign abstraction, formally	11
3.18	Characteristic property of abstraction/concretization	12
3.19	Galois connection	12
3.20	Calculational design of the sign semantics of expressions	13
3.21	Calculational design of abstract interpretations	15
3.22	Conclusion	15
3.23	Exercises	15
3.24	Answers to selected exercises	16
3.25	Bibliography	20

3.1 The rule of signs

The Indian mathematician and astronomer **Brahmagupta** (born c. 598, died after 665) was the first to give rules to compute with zero and invented the rule of signs [14, page 151]. Verses 18.30–35 of his *Brāhma-sphuṭa-siddhānta* state

[The sum] of two positives is positive, of two negatives negative; of a positive and a negative [the sum] is their difference; if they are equal it is zero. The sum of a negative and zero is negative, [that] of a positive and zero positive, [and that] of two zeros zero.

...

A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.

The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.

A positive divided by a positive or a negative divided by a negative is positive; a zero divided by a zero is zero; a positive divided by a negative is negative; a negative divided by a positive is [also] negative.

A negative or a positive divided by zero has that [zero] as its divisor, or zero divided by a negative or a positive [has that negative or positive as its divisor]. The square of a negative or of a positive is positive; [the square] of zero is zero.

Exercise 3.1. What is the modern understanding of $\frac{0}{0}$? □

Exercise 3.2 (Erroneous sign analysis). Following the pseudo-evaluation idea of Peter Naur in compilation [12, 13], Michel Sintzoff [16] postulates the sign analysis in the following way:

“ $a \times a + b \times b$ yields always the object “pos” when a and b are the objects “pos” or “neg”, and when the valuation is defined as follows :

$$\begin{array}{llll}
 \text{pos} + \text{pos} & = & \text{pos} & \text{pos} \times \text{pos} & = & \text{pos} \\
 \text{pos} + \text{neg} & = & \text{pos, neg} & \text{pos} \times \text{neg} & = & \text{neg} \\
 \text{neg} + \text{pos} & = & \text{pos, neg} & \text{neg} \times \text{pos} & = & \text{neg} \\
 \text{neg} + \text{neg} & = & \text{neg} & \text{neg} \times \text{neg} & = & \text{pos} \\
 V(p+q) & = & V(p)+V(q) & V(p \times q) & = & V(p) \times V(q) \\
 V(0) & = & V(1) & = & \dots & = & \text{pos} \\
 V(-1) & = & V(-2) & = & \dots & = & \text{neg}
 \end{array}$$

The valuation of $a \times a + b \times b$ yields “pos” by the following computation :

$$\begin{array}{llll}
 V(a) & = & \text{pos, neg} & V(b) & = & \text{pos, neg} \\
 V(a \times a) & = & \text{pos} \times \text{pos, neg} \times \text{neg} & V(b \times b) & = & \text{pos} \times \text{pos, neg} \times \text{neg} \\
 & = & \text{pos, pos} & & = & \text{pos, pos} & = & \text{pos} \\
 V(a \times a + b \times b) & = & V(a \times a) + V(b \times b) & = & \text{pos} + \text{pos} & = & \text{pos}
 \end{array}$$

What is wrong about it? □

3.2 Sign analysis of iterative programs

The rule of signs generalizes to programs. For example the sign of x in

$x = 0; \text{ while } (...) \{ x = x+1 \}$

(where the iteration condition $(...)$ is ignored) can be determined as follows:

- After zero iteration, when entering the loop, if ever, $x = 0$;
- After one iteration, the sign of x is zero, 1 is positive, so the sum $x+1$ of zero and positive is positive;

3.3 Sign abstraction, informally

3

- For the basis, we have shown that after zero or one iteration, the sign of x is zero (at iteration 0) or positive (at iteration 1) that is positive after at most 1 iteration;
- For the induction step, if after at most $n \geq 0$ iterations, the sign of x is positive, then 1 is positive, so the sum $x+1$ of positive and positive is positive after the next iteration;
- After at most $n+1$ iterations, x is positive (at the previous $n \geq 0$ iterations) or positive (at the $n+1$ -th iteration) then x is positive after at most $n+1$ iterations;
- By recurrence on the number of iterations in the loop, x is positive in the loop.

3.3 Sign abstraction, informally

The abstraction is that you do not (always) need to know the absolute value of the arguments to know the sign of the result of an operation. This is sometimes precise (for example for the multiplication) but can be imprecise (for example the sign of the sum of a positive and a negative is unknown when ignoring the absolute value of the arguments). This is nevertheless useful in practice if you know what to do when you don't know the sign. For example, a compiler will not suppress the lower bound check when accessing an array with an index not known to be positive. Moreover, it is always possible to refine the abstraction to get more precise results. For example [Brahmagupta](#) states [14, page 151]

[If] a smaller [positive] is to be subtracted from a larger positive, [the result] is positive; [if] a smaller negative from a larger negative, [the result] is negative; [if] a larger [negative or positive is to be subtracted] from a smaller [positive or negative, the algebraic sign of] their difference is reversed—negative [becomes] positive and positive negative. ...

Knowing an interval of the possible values is more precise than just knowing the sign. Interval analysis is considered in Chapter 31 (Static Interval Analysis).

The objective of this Chapter 3 is to formalize abstract interpretations of arithmetic expressions (like the rule of signs) and to show how the abstraction can be formally calculated out of the semantics of arithmetic expressions.

3.4 Syntax of expressions

Let us consider the language of expressions.

x, y, \dots	\in	\mathcal{V}	variables (\mathcal{V} not empty)
A	\in	$\mathcal{A} ::= 1 \mid x \mid A_1 - A_2$	arithmetic expressions
B	\in	$\mathcal{B} ::= A_1 < A_2 \mid B_1 \text{ nand } B_2$	boolean expressions
E	\in	$\mathcal{E} ::= A \mid B$	expressions

This context-free grammar specifies sets of program syntactic entities, the set \mathcal{V} of variables, \mathcal{A} of arithmetic expressions, \mathcal{B} of boolean expressions, and \mathcal{E} of either arithmetic or boolean expressions. The mathematical variables x , y , A , B , and E denote arbitrary elements of these sets.

There syntax is defined by grammar rules such as $A ::= 1 \mid x \mid A_1 - A_2$ specifying that an arithmetic expression A is either the constant 1, a variable $x \in \mathcal{V}$, or the difference $A_1 - A_2$ of two arithmetic expressions A_1 and A_2 . The set \mathcal{V} of variables is left unspecified (usually it is an identifier starting with a letter followed by 0 or more letters or digits or special symbols like “_”).

This grammar is ambiguous since $1 - 1 - 1$ can either be understood as $(1 - 1) - 1$ or $1 - (1 - 1)$. We choose the first alternative so the binary operator is left-associative. In boolean expressions, **and** is left-associative and the arithmetic operators have priority over boolean operators (so $1 - 1 < 1 - 1 - 1$ is $((1 - 1) < ((1 - 1) - 1))$ i.e. false **ff**). The description of syntax by grammar dates back to Noam Chomsky [4].

3.5 Structural definitions

Structural definitions are generalizations of recursive definitions on naturals. Assume that we want to define a function $f \in \mathcal{E} \rightarrow S$ where S is a set. A structural definition is a recursive definition of the form

- $f(1)$ and $f(x)$ are defined to be constants (so $f(1) \triangleq c_1$ and $f(x) \triangleq c_x$ where $c_1, c_x \in S$);
- $f(A_1 - A_2)$ and $f(A_1 < A_2)$ are functions of $f(A_1)$ and $f(A_2)$ (so $f(A_1 - A_2) \triangleq F_-(f(A_1), f(A_2))$), $f(A_1 < A_2) \triangleq F_<(f(A_1), f(A_2))$ ¹;
- $f(B_1 \text{ and } B_2) \triangleq F_{\text{and}}(f(B_1), f(B_2))$ where $F_-, F_<, F_{\text{and}} \in S \times S \rightarrow S$.

Exercise 3.3. Define $\text{vars} \in \mathcal{E} \rightarrow \wp(\mathcal{V})$ such that $\text{vars}[\![E]\!]$ is the (possibly empty) set of variables occurring in expression E . □

Structural definitions are the basis of denotational semantics introduced by Dana Scott and Christopher Strachey [15] (and called compositional in this context).

3.6 Environments

In order to formally define the value of any expression e.g. $1 - 1 - 1 = -1$, we need to know the value of variables occurring in expressions e.g. $x - 1$ is 2 when $x = 3$, $x - 1$ is 42 when $x = 43$, etc. We cannot enumerate the infinitely many cases ..., $x = -1$, $x = 0$, $x = 1$, So we use an environment $\rho \in \mathcal{E}\mathcal{V}$ where $\mathcal{E}\mathcal{V} \triangleq \mathcal{V} \rightarrow \mathbb{Z}$ that is a function ρ mapping a variable x to its value $\rho(x)$ in the set \mathbb{Z} of all mathematical integers. By reasoning on the function ρ we can handle infinitely many cases at once. For example, in environment ρ , the value of $x - 1$ is $\rho(x) - 1$ where $\rho(x)$ is the value of variable x , 1 is the mathematical integer one and $-$ is the mathematical difference.

¹ \triangleq is “is defined as”.

3.7 Structural semantics of expressions

Given an environment $\rho \in \mathbb{E}_V \triangleq V \rightarrow \mathbb{Z}$ mapping variables $x \in V$ to their value $\rho(x) \in \mathbb{Z}$, the value $\mathcal{A}[\![A]\!]\rho \in \mathbb{Z}$ of an arithmetic expression $A \in \mathcal{A}$ and $\mathcal{B}[\![B]\!]\rho \in \mathbb{B}$ of a boolean expression $B \in \mathcal{B}$ is structurally defined as follows.

$$\begin{aligned}
 \mathcal{A}[\![1]\!]\rho &\triangleq 1 \\
 \mathcal{A}[\![x]\!]\rho &\triangleq \rho(x) \\
 \mathcal{A}[\![A_1 - A_2]\!]\rho &\triangleq \mathcal{A}[\![A_1]\!]\rho - \mathcal{A}[\![A_2]\!]\rho \\
 \mathcal{B}[\![A_1 < A_2]\!]\rho &\triangleq \mathcal{A}[\![A_1]\!]\rho < \mathcal{A}[\![A_2]\!]\rho \\
 \mathcal{B}[\![B_1 \text{ nand } B_2]\!]\rho &\triangleq \mathcal{B}[\![B_1]\!]\rho \uparrow \mathcal{B}[\![B_2]\!]\rho \\
 \mathcal{S}[\![E]\!]\rho &\triangleq \mathcal{A}[\![E]\!]\rho && \text{when } E \in \mathcal{A} \\
 \mathcal{S}[\![E]\!]\rho &\triangleq \mathcal{B}[\![E]\!]\rho && \text{when } E \in \mathcal{B}
 \end{aligned} \tag{3.4}$$

1, x , $-$, $<$, nand, \mathcal{A} , and \mathcal{B} are syntactic objects *e.g.* strings of characters. 1, ρ , $-$, $<$, and \uparrow are mathematical objects. The recursive definition is structural *i.e.* by induction on the syntax of expressions E (either arithmetic \mathcal{A} or boolean \mathcal{B}). The semantics of complex expressions $\mathcal{A}[\![A]\!]\rho$ or $\mathcal{B}[\![B]\!]\rho$ is defined in function of the semantics of simpler expressions until reaching basic cases $\mathcal{A}[\![1]\!]\rho \triangleq 1$ and $\mathcal{A}[\![x]\!]\rho \triangleq \rho(x)$ for which the value is constant. The “not and” or “nand” boolean operator \uparrow is defined by the following truth table

a	tt	tf	ft	ff
b	tt	ff	tt	ff
$a \uparrow b$	ff	tt	tt	tt

The functions \mathcal{A} and \mathcal{B} will be shown to be a total functions *i.e.* well-defined for all their arguments in Exercise 3.8. This shows that the recursion always terminates.

Exercise 3.5. Define the logical operators (negation \neg , implication \Rightarrow , conjunction \vee , disjunction \wedge) in terms of \uparrow . □

Exercise 3.6. Write a program in the language of your choice that inputs (an encoding of) an expression (with no variables) and returns the value of this expression. □

Exercise 3.7. Prove that any integer $z \in \mathbb{Z}$ has a finite denotation in the syntax \mathcal{A} of arithmetic expressions. □

3.8 Proofs by structural induction

Proofs by structural induction [2] generalize proofs by recurrence. They are well suited for proving properties of structural definitions. Rod Burstall [2] introduced them as follows “If for some set of structures a structure has a certain property whenever all its proper constituents have that property

then all the structures in the set have the property”. So, to prove that a property P holds for all expressions $E \in \mathcal{E}$, we prove that the property holds for the basic cases 1 and x . Then assuming that the property holds for A_1 and A_2 , we prove that it holds for $A_1 - A_2$ and $A_1 < A_2$. Moreover, assuming the property holds for boolean expressions B_1 and B_2 , we prove that it also holds for $B_1 \text{ nand } B_2$. We conclude that $\mathcal{E} \subseteq P$.

Exercise 3.8. Prove, by structural induction on the syntax of expressions, that \mathcal{B} is a total function i.e. $\forall B \in \mathcal{B} . \mathcal{B}[\![B]\!] \in (\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{B}$. (The property to be proved is therefore $P = \{B \in \mathcal{B} \mid \mathcal{B}[\![B]\!] \in (\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{B}\}$). (A similar property has to be stated and proved for arithmetic expressions.) \square

Exercise 3.9. Prove, by structural induction on the syntax of expressions, that if $x \notin \text{vars}[\![B]\!]$ and $\forall y \in \mathcal{V} \setminus \{x\} . \rho'(y) = \rho(y)$ then $\mathcal{B}[\![B]\!]\rho = \mathcal{B}[\![B]\!]\rho'$. (A similar property has to be stated and proved for arithmetic expressions.) \square

3.9 Semantic properties of expressions

By expression property we might mean a property of the syntax of the expression (such as A has 42 signs – more precisely A belongs to the set of expressions with 42 signs –). This is software metrics and metrology [17], of little interest to us.

Instead an expression property will be understood as a semantic property that is a property of the semantics of expressions.

The semantics $\mathcal{A}[\![A]\!]$ of an expression A maps environments $\rho \in \mathcal{V} \rightarrow \mathcal{Z}$ to a values in \mathcal{Z} , $\mathcal{A}[\![A]\!] \in (\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{Z}$. Following Section 2.3, a semantic property of an expression is a set of possible semantics hence belongs to $\wp((\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{Z})$. If $P \in \wp((\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{Z})$ is a semantic property, then $\mathcal{A}[\![A]\!] \in P$ means that “ A has property P ”.

Example 3.10 $P = \{b \mid \forall \rho \in \mathcal{V} \rightarrow \mathcal{Z} . b(\rho) = \text{tt}\} \cup \{b \mid \forall \rho \in \mathcal{V} \rightarrow \mathcal{Z} . b(\rho) = \text{ff}\}$ is the semantic property of a boolean expression “to always evaluate to tt ” or “to always evaluate to ff ”. For example $x * x + 1 > 0$ and $x * x < 0$ have this property but not $x * x > 0$ since $x * x > 0$ is sometimes true (when $|\rho(x)| > 0$) and sometimes false (when $|\rho(x)| = 0$). So $\mathcal{B}[\![x * x + 1 > 0]\!] \in P$ while $\mathcal{B}[\![x * x > 0]\!] \notin P$. \square

Notice that semantic properties P of expressions are just a particular case of property of expressions i.e. the property $\{E \in \mathcal{E} \mid \mathcal{S}[\![E]\!] \in P\}$.

3.10 Collecting semantics of expressions

The collecting semantics of expressions is the strongest property of an expression.

$$\mathcal{S}^c[\![A]\!] \triangleq \{\mathcal{A}[\![A]\!]\} \in \wp((\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{Z}) \quad (3.11)$$

Arithmetic expression A is said to have semantic property $P \in \wp((\mathcal{V} \rightarrow \mathcal{Z}) \rightarrow \mathcal{Z})$ if and only if $\mathcal{A}[\![A]\!] \in P$ or equivalently $\mathcal{S}^c[\![A]\!] \subseteq P$ so that $\mathcal{S}^c[\![A]\!]$ is the strongest property of A . The idea

3.11 Proving semantic properties of expressions by structural induction

7

of collecting semantics was introduced in [6] (under the qualifier “static semantics”) as a basis for proving the soundness of static analyzers. The concept of collecting semantics is further developed in Chapter 8.

The fact that $(\mathcal{A} \llbracket A \rrbracket \in P) \Leftrightarrow (\mathcal{S}^c \llbracket A \rrbracket \subseteq P)$ may suggest that the concept of collecting semantics is of poor interest. However, $x \in S \Leftrightarrow \{x\} \subseteq S$ is the basic idea for abstracting set theory into order/lattice theory [1]. It will later allow us to use order theory (which has the equivalent of \subseteq but not of \in), see Chapter 10 (Posets, lattices, and complete lattices).

Similarly, the collecting semantics of boolean expressions is

$$\mathcal{S}^c \llbracket B \rrbracket \triangleq \{\mathcal{B} \llbracket B \rrbracket\} \in \wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{B})$$

Again the collecting semantics $\mathcal{S}^c \llbracket E \rrbracket$ of expressions E is just a particular case of property of expressions *i.e.* the property $\{E' \in \mathcal{E} \mid \mathcal{S} \llbracket E' \rrbracket \in \mathcal{S}^c \llbracket E \rrbracket\}$ *i.e.* all expressions E' that have the same semantics as E .

3.11 Proving semantic properties of expressions by structural induction

Semantic properties can be proved by structural induction on expressions. For basic cases the proof is $\mathcal{S}^c \llbracket 1 \rrbracket \subseteq P$ and $\mathcal{S}^c \llbracket x \rrbracket \subseteq P$. Assuming $\mathcal{S}^c \llbracket A_1 \rrbracket \subseteq P$ and $\mathcal{S}^c \llbracket A_2 \rrbracket \subseteq P$, we prove $\mathcal{S}^c \llbracket A_1 - A_2 \rrbracket \subseteq P$ and $\mathcal{S}^c \llbracket A_1 < A_2 \rrbracket \subseteq P$. Assuming $\mathcal{S}^c \llbracket B_1 \rrbracket \subseteq P$ and $\mathcal{S}^c \llbracket B_2 \rrbracket \subseteq P$, we prove that for $\mathcal{S}^c \llbracket B_1 \text{ nand } B_2 \rrbracket \subseteq P$. By structural induction, we conclude that $\mathcal{E} \subseteq \{E \in \mathcal{E} \mid \mathcal{S}^c \llbracket E \rrbracket \subseteq P\}$ *i.e.* $\forall E \in \mathcal{E} . \mathcal{S}^c \llbracket E \rrbracket \subseteq P$.

Exercise 3.12. Prove by structural induction on expressions that

$$\begin{aligned} \mathcal{S}^c \llbracket 1 \rrbracket &= \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto 1\} \\ \mathcal{S}^c \llbracket x \rrbracket &= \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto \rho(x)\} \\ \mathcal{S}^c \llbracket A_1 - A_2 \rrbracket &= \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto f_1(\rho) - f_2(\rho) \mid f_1 \in \mathcal{S}^c \llbracket A_1 \rrbracket \wedge f_2 \in \mathcal{S}^c \llbracket A_2 \rrbracket\} \\ \mathcal{S}^c \llbracket A_1 < A_2 \rrbracket &= \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto f_1(\rho) < f_2(\rho) \mid f_1 \in \mathcal{S}^c \llbracket A_1 \rrbracket \wedge f_2 \in \mathcal{S}^c \llbracket A_2 \rrbracket\} \\ \mathcal{S}^c \llbracket B_1 \text{ nand } B_2 \rrbracket &= \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto f_1(\rho) \uparrow f_2(\rho) \mid f_1 \in \mathcal{S}^c \llbracket B_1 \rrbracket \wedge f_2 \in \mathcal{S}^c \llbracket B_2 \rrbracket\} \quad \square \end{aligned}$$

Exercise 3.13. Continuing Exercise 3.12, prove that $\mathcal{S}^c \llbracket x - x \rrbracket = \{\rho \in (\mathcal{V} \rightarrow \mathbb{Z}) \mapsto 0\}$. \square

Program proof methods are further studied in Chapters 22 () and 23 (Abstract verification semantics).

3.12 Abstract sign properties

We let $\mathbb{P}^\pm \triangleq \{\perp_\pm, <0, =0, >0, \leq 0, \neq 0, \geq 0, \top_\pm\}$ be the set of signs where <0 is “strictly negative”, ≥ 0 is “positive or zero”, *etc.*, $=0$ is “equal to zero”, $\neq 0$ is “different from zero” (*i.e.* “strictly negative or strictly positive”). \top_\pm (top) is “unknown sign” (*i.e.* \top that is “negative, zero, or positive”), \perp_\pm (bottom) is “no sign” (*i.e.* ff that is “neither negative, zero, nor positive”) be the abstract properties of the sign abstract domain \mathbb{D}_\pm .

Exercise 3.14. Continuing Section 3.2, provide an example of program where the sign of a variable x is \perp_{\pm} . \square

The sign minus operation $\neg_{\pm} \in \mathbb{P}^{\pm} \times \mathbb{P}^{\pm} \rightarrow \mathbb{P}^{\pm}$ defines the sign $s_1 \neg_{\pm} s_2$ of $x - y$ when x has sign s_1 and y has sign s_2 .

$s_1 \neg_{\pm} s_2$		s_2							
		\perp_{\pm}	<0	$=0$	>0	≤ 0	$\neq 0$	≥ 0	\top_{\pm}
s_1	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}	\perp_{\pm}
	<0	\perp_{\pm}	\top_{\pm}	<0	<0	\top_{\pm}	\top_{\pm}	<0	\top_{\pm}
	$=0$	\perp_{\pm}	>0	$=0$	<0	≥ 0	$\neq 0$	≤ 0	\top_{\pm}
	>0	\perp_{\pm}	>0	>0	\top_{\pm}	>0	\top_{\pm}	\top_{\pm}	\top_{\pm}
	≤ 0	\perp_{\pm}	\top_{\pm}	≤ 0	<0	\top_{\pm}	\top_{\pm}	≤ 0	\top_{\pm}
	$\neq 0$	\perp_{\pm}	\top_{\pm}	$\neq 0$	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}
	≥ 0	\perp_{\pm}	>0	≥ 0	\top_{\pm}	≥ 0	\top_{\pm}	\top_{\pm}	\top_{\pm}
	\top_{\pm}	\perp_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}	\top_{\pm}

Exercise 3.15. Show that sign operator \neg_{\pm} is imprecise for difference ($-$). \square

Exercise 3.16. What is wrong with the `int abs(int x) { return (x<0) ? -x : x; }` method in Java™? \square

Exercise 3.17. Prove that the sign minus operation \neg_{\pm} is incorrect with machine integers. \square

Exercise 3.18. Design a sign operator for multiplication of mathematical integers (\times). Prove that it is exact *i.e.* the sign of the result is exactly known from the sign of the parameters. \square

3.13 Structural sign semantics of expressions

The sign of an expression depends upon the sign of its free variables. We represent the sign of variables by a sign environment $\dot{\rho} \in \mathcal{V} \rightarrow \mathbb{P}^{\pm}$ such that $\dot{\rho}(x)$ is the sign of variable x .

The sign semantics $\mathcal{S}^{\pm} \llbracket A \rrbracket \dot{\rho}$ of an arithmetic expression A is the sign of the expression value when evaluated with variables which sign is given by the sign environment $\dot{\rho}$. For example, if $\dot{\rho}(x) = >0$ and $\dot{\rho}(y) = \leq 0$ then $\mathcal{S}^{\pm} \llbracket x - y \rrbracket \dot{\rho} = >0$.

The structural sign semantics $\mathcal{S}^{\pm} \llbracket A \rrbracket \in (\mathcal{V} \rightarrow \mathbb{P}^{\pm}) \rightarrow \mathbb{P}^{\pm}$ may be defined as follows.

$$\begin{aligned}
 \mathcal{S}^{\pm} \llbracket 1 \rrbracket \dot{\rho} &= >0 \\
 \mathcal{S}^{\pm} \llbracket x \rrbracket \dot{\rho} &= \dot{\rho}(x) \\
 \mathcal{S}^{\pm} \llbracket A_1 - A_2 \rrbracket \dot{\rho} &= (\mathcal{S}^{\pm} \llbracket A_1 \rrbracket \dot{\rho}) \neg_{\pm} (\mathcal{S}^{\pm} \llbracket A_2 \rrbracket \dot{\rho})
 \end{aligned}$$

3.14 Soundness

9

To be more precise, if any of the variables has sign \perp_+ , meaning “the expression is never evaluated” then the result is \perp_+ , meaning “no result is ever returned”. We say that signs are \perp_+ -strict and define \Downarrow^+ to enforce it.

$$\begin{aligned}
 \Downarrow^+[\dot{\rho}]s &\triangleq ((\exists y \in \mathcal{V} . \dot{\rho}(y) = \perp_+ \text{ ? } \perp_+ : s)) \\
 \mathcal{S}^+[\mathbf{1}]\dot{\rho} &= \Downarrow^+[\dot{\rho}](>0) \\
 \mathcal{S}^+[\mathbf{x}]\dot{\rho} &= \Downarrow^+[\dot{\rho}](\dot{\rho}(\mathbf{x})) \\
 \mathcal{S}^+[\mathbf{A}_1 - \mathbf{A}_2]\dot{\rho} &= (\mathcal{S}^+[\mathbf{A}_1]\dot{\rho})_{-} (\mathcal{S}^+[\mathbf{A}_2]\dot{\rho})
 \end{aligned} \tag{3.19}$$

Exercise 3.20. Prove by structural induction on \mathbf{A} that if $\exists \mathbf{x} \in \mathcal{V} . \dot{\rho}(\mathbf{x}) = \perp_+$ then $\mathcal{S}^+[\mathbf{A}]\dot{\rho} = \perp_+$. \square

3.14 Soundness

We would like to prove that the sign semantics $\mathcal{S}^+[\mathbf{A}]$ of an arithmetic expression \mathbf{A} is a weaker property than the collecting semantics $\mathcal{S}^c[\mathbf{A}]$. But $\mathcal{S}^+[\mathbf{A}] \in (\mathcal{V} \rightarrow \mathbb{P}^\pm) \rightarrow \mathbb{P}^\pm$ while $\mathcal{S}^c[\mathbf{A}] \in \wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ and the concrete semantic properties in $\wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ are hardly comparable to the abstract sign properties in $(\mathcal{V} \rightarrow \mathbb{P}^\pm) \rightarrow \mathbb{P}^\pm$.

The solution is to express abstract properties in $(\mathcal{V} \rightarrow \mathbb{P}^\pm) \rightarrow \mathbb{P}^\pm$ as a concrete property in $\wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$. For that purpose we will define a concretization function $\dot{\gamma}_\pm \in ((\mathcal{V} \rightarrow \mathbb{P}^\pm) \rightarrow \mathbb{P}^\pm) \rightarrow (\wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z}))$ mapping an abstract property to an “equivalent” concrete property.

Then the concrete semantics implies the abstract semantics up to concretization in that for all arithmetic expressions \mathbf{A} ,

$$\mathcal{S}^c[\mathbf{A}] \subseteq \dot{\gamma}_\pm(\mathcal{S}^+[\mathbf{A}]).$$

3.15 Sign concretization

We define the sign concretization function $\dot{\gamma}_\pm$ in several steps.

1. First we consider signs (in \mathbb{P}^\pm) as properties of integers (in $\wp(\mathbb{Z})$).

$$\begin{aligned}
 \gamma_\pm(\perp_+) &\triangleq \emptyset & \gamma_\pm(\leq 0) &\triangleq \{z \in \mathbb{Z} \mid z \leq 0\} \\
 \gamma_\pm(< 0) &\triangleq \{z \in \mathbb{Z} \mid z < 0\} & \gamma_\pm(\neq 0) &\triangleq \{z \in \mathbb{Z} \mid z \neq 0\} \\
 \gamma_\pm(= 0) &\triangleq \{0\} & \gamma_\pm(\geq 0) &\triangleq \{z \in \mathbb{Z} \mid z \geq 0\} \\
 \gamma_\pm(> 0) &\triangleq \{z \in \mathbb{Z} \mid z > 0\} & \gamma_\pm(\top_\pm) &\triangleq \mathbb{Z}
 \end{aligned} \tag{3.21}$$

2. Then we consider sign environments $\dot{\rho} \in \mathcal{V} \rightarrow \mathbb{P}^\pm$ as properties of environments (in $\wp(\mathcal{V} \rightarrow \mathbb{Z})$). $\dot{\rho}$ is the abstract property of all concrete environments ρ such that for all variables \mathbf{x} , the sign of $\rho(\mathbf{x})$ is $\dot{\rho}(\mathbf{x})$.

$$\dot{\gamma}_\pm(\dot{\rho}) \triangleq \{\rho \in \mathcal{V} \rightarrow \mathbb{Z} \mid \forall \mathbf{x} \in \mathcal{V} . \rho(\mathbf{x}) \in \gamma_\pm(\dot{\rho}(\mathbf{x}))\} \tag{3.22}$$

Observe that if $\dot{\rho}(x) = \perp_{\pm}$ for some $x \in \mathcal{V}$ then $\gamma_{\pm}(\dot{\rho}(x)) = \emptyset$ so $\forall x \in \mathcal{V} . \rho(x) \in \gamma_{\pm}(\dot{\rho}(x))$ is false proving that $\dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset$. So the abstraction of false ($\emptyset \in \wp(\mathcal{V} \rightarrow \mathbb{Z})$) is any abstract environment $\dot{\rho}$ with at least one variable x such that $\dot{\rho}(x) = \perp_{\pm}$.

3. Finally the concretization of abstract properties $\bar{P} \in (\mathcal{V} \rightarrow \mathbb{P}^{\pm}) \rightarrow \mathbb{P}^{\pm}$ is the concrete property $\dot{\gamma}_{\pm}(\bar{P}) \in \wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ defined as

$$\dot{\gamma}_{\pm}(\bar{P}) \triangleq \{ \mathcal{S} \in (\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z} \mid \forall \dot{\rho} \in \mathcal{V} \rightarrow \mathbb{P}^{\pm} . \forall \rho \in \dot{\gamma}_{\pm}(\dot{\rho}) . \mathcal{S}(\rho) \in \gamma_{\pm}(\bar{P}(\dot{\rho})) \} \quad (3.23)$$

i.e. \mathbf{A} has abstract property \bar{P} , that is $\mathcal{A}[\mathbf{A}] \in \dot{\gamma}_{\pm}(\bar{P})$, if and only if for all environments ρ with signs $\dot{\rho}$, the value $\mathcal{A}[\mathbf{A}]\rho$ of arithmetic expression \mathbf{A} has sign $\bar{P}(\dot{\rho})$.

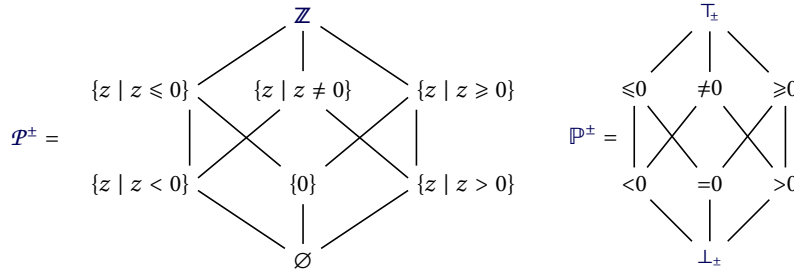
Exercise 3.24. Assume that $\gamma(\text{pos}) = \{z \in \mathbb{Z} \mid z \geq 0\}$ and $\gamma(\text{neg}) = \{z \in \mathbb{Z} \mid z < 0\}$ as in Exercise 3.2. Provide a sound definition of the rule of signs for multiplication \times with this interpretation of the rule of sign. \square

Exercise 3.25 (A posteriori soundness). Prove that for all $\mathbf{A} \in \mathcal{A}$, $\mathcal{S}^c[\mathbf{A}] \subseteq \dot{\gamma}_{\pm}(\mathcal{S}^{\pm}[\mathbf{A}])$. \square

3.16 Sign lattice

Sign properties $\mathcal{P}^{\pm} \triangleq \{\gamma_{\pm}(s) \mid s \in \mathbb{P}^{\pm}\}$ of integers can be partially ordered by \subseteq (i.e. implication) as represented by the Hasse diagram below where the nodes are the elements of \mathcal{P}^{\pm} and there is a bottom-up arrow from $P \in \mathcal{P}^{\pm}$ to $P' \in \mathcal{P}^{\pm}$ when $P \subsetneq P'$ and no $Q \in \mathcal{P}^{\pm}$ such that $P \subsetneq Q \subsetneq P'$. So $P \subseteq Q$ if and only if there is a path from P to Q in the Hasse diagram.

The abstract signs \mathbb{P}^{\pm} are an isomorphic representation of \mathcal{P}^{\pm} as shown on the right, where the isomorphism is $\gamma_{\pm} \in \mathbb{P}^{\pm} \rightarrow \mathcal{P}^{\pm}$.



Therefore, the abstract sign properties are partially ordered by \sqsubseteq_{\pm} defined by $s \sqsubseteq_{\pm} s'$ if and only if $\gamma_{\pm}(s) \subseteq \gamma_{\pm}(s')$.

Exercise 3.26. Specify algorithmically the inclusion \sqsubseteq_{\pm} on \mathbb{P}^{\pm} (mathematically defined above as $s \sqsubseteq_{\pm} s'$ if and only if $\gamma_{\pm}(s) \subseteq \gamma_{\pm}(s')$). \square

Exercise 3.27. Prove that \neg_{\pm} is increasing in each of its parameters *i.e.* if $s_1 \sqsubseteq_{\pm} s'_1$ then $s_1 \neg_{\pm} s_2 \sqsubseteq_{\pm} s'_1 \neg_{\pm} s_2$ and $s_2 \sqsubseteq_{\pm} s'_2$ then $s_1 \neg_{\pm} s_2 \sqsubseteq_{\pm} s_1 \neg_{\pm} s'_2$ so that if $s_1 \sqsubseteq_{\pm} s'_1$ and $s_2 \sqsubseteq_{\pm} s'_2$ then $s_1 \neg_{\pm} s_2 \sqsubseteq_{\pm} s'_1 \neg_{\pm} s'_2$.
□

3.17 Sign abstraction, formally

3.17.1 Abstraction of sign properties

An integer property like $2\mathbb{N} + 1$ (odd naturals) can be over-approximated in \mathbb{P}^{\pm} by sign properties $\{z \in \mathbb{Z} \mid z > 0\}$, $\{z \in \mathbb{Z} \mid z \geq 0\}$, and \mathbb{Z} . The best over-approximation of $2\mathbb{N} + 1$ in \mathbb{P}^{\pm} is $\{z \in \mathbb{Z} \mid z > 0\}$ since it is sound (in that $2\mathbb{N} + 1 \subseteq \{z \in \mathbb{Z} \mid z > 0\}$) and the most precise/strongest (in that $\{z \in \mathbb{Z} \mid z > 0\} \subseteq \{z \in \mathbb{Z} \mid z \geq 0\} \subseteq \mathbb{Z}$).

More generally, the best over-approximation of any integer property $P \in \wp(\mathbb{Z})$ in \mathbb{P}^{\pm} is given by the abstraction function

$$\alpha_{\pm}(P) \triangleq \left(\begin{array}{l} P \subseteq \emptyset \text{ ? } \perp_{\pm} \\ P \subseteq \{z \mid z < 0\} \text{ ? } <0 \\ P \subseteq \{0\} \text{ ? } =0 \\ P \subseteq \{z \mid z > 0\} \text{ ? } >0 \\ P \subseteq \{z \mid z \leq 0\} \text{ ? } \leq 0 \\ P \subseteq \{z \mid z \neq 0\} \text{ ? } \neq 0 \\ P \subseteq \{z \mid z \geq 0\} \text{ ? } \geq 0 \\ \vdots \end{array} \right) \quad (3.28)$$

$\alpha_{\pm}(P)$ is the best over-approximation of $P \in \wp(\mathbb{Z})$ in \mathbb{P}^{\pm} since

- $P \subseteq \gamma_{\pm}(\alpha_{\pm}(P))$ *i.e.* $\alpha_{\pm}(P)$ is an over-approximation/sound abstraction of P ;
- if $\bar{P} \in \mathbb{P}^{\pm}$ and $P \subseteq \gamma_{\pm}(\bar{P})$ then $\alpha_{\pm}(P) \sqsubseteq_{\pm} \bar{P}$ *i.e.* $\alpha_{\pm}(P)$ is more precise than any other over-approximation/sound abstraction of P .

Exercise 3.29. Prove that $s_1 \neg_{\pm} s_2 = \alpha_{\pm}(\{x - y \mid x \in \gamma_{\pm}(s_1) \wedge y \in \gamma_{\pm}(s_2)\})$. □

Exercise 3.30. Define the finite join \sqcup_{\pm} on \mathbb{P}^{\pm} such that $\sqcup_{\pm}\{s_i \mid i \in \Delta\} \triangleq \alpha_{\pm}(\bigcup\{\gamma_{\pm}(s_i) \mid i \in \Delta\})$. Prove that $s \sqcup_{\pm} s' = \{a \mid a \in \{<0, =0, >0\} \wedge (a \sqsubseteq_{\pm} s \vee a \sqsubseteq_{\pm} s')\}$. Specify the join \sqcup_{\pm} on \mathbb{P}^{\pm} algorithmically. □

3.17.2 Abstraction of environment properties

The best abstraction of an environment property $P \in \wp(\mathcal{V} \rightarrow \mathbb{Z})$ is

$$\dot{\alpha}_{\pm}(P) \triangleq x \in \mathcal{V} \mapsto \alpha_{\pm}(\{\rho(x) \mid \rho \in P\}) \quad (3.31)$$

i.e. for each variable x it is the sign of the set of values $\rho(x)$ in all environments ρ satisfying P .

Observe that $\dot{\alpha}_{\pm}(P) \triangleq \dot{\perp}_{\pm} \triangleq x \in \mathcal{V} \mapsto \perp_{\pm}$ while if $\dot{\rho}(x) = \perp_{\pm}$ then $\dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset$ so $\emptyset \in \wp(\mathcal{V} \rightarrow \mathbb{Z})$ has several possible abstractions in \mathbb{P}^{\pm} but $\dot{\perp}_{\pm}$ is the pointwise $\dot{\sqsubseteq}_{\pm}$ -smallest of them.

3.17.3 Abstraction of semantic properties

The best abstraction of a semantic property $P \in \wp((\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ is

$$\check{\alpha}_{\pm}(P) \triangleq \check{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} \mapsto \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in P \wedge \rho \in \check{\gamma}_{\pm}(\check{\rho})\}) \quad (3.32)$$

i.e. given a sign environment $\check{\rho}$, $\check{\alpha}_{\pm}(P)\check{\rho}$ is the sign of the possible results $\mathcal{S}(\rho)$ of the semantics $\mathcal{S} \in P$ with property P for all environments ρ with sign $\check{\rho}$.

3.18 Characteristic property of abstraction/concretization

The abstraction/concretization functions $\langle \alpha_{\pm}, \gamma_{\pm} \rangle$ are closely related in that for all $P \in \wp(\mathbb{Z})$ and $\bar{P} \in \mathbb{P}^{\pm}$, they satisfy

$$\alpha_{\pm}(P) \sqsubseteq_{\pm} \bar{P} \Leftrightarrow P \subseteq \gamma_{\pm}(\bar{P})$$

Proof By definition (3.21) of γ_{\pm} and (3.28) of α_{\pm} , we observe that

- γ_{\pm} is increasing i.e. if $s \sqsubseteq_{\pm} s'$ then $\gamma_{\pm}(s) \subseteq \gamma_{\pm}(s')$;
- α_{\pm} is increasing i.e. if $P \subseteq P'$ then $\alpha_{\pm}(P) \sqsubseteq_{\pm} \alpha_{\pm}(P')$; (3.33)

- if $\alpha_{\pm}(P) = s$ then $P \subseteq \gamma_{\pm}(s)$ so $\gamma_{\pm} \circ \alpha_{\pm}$ is extensive i.e. $P \subseteq \gamma_{\pm} \circ \alpha_{\pm}(P)$; (3.34)

- by case analysis, if $P = \gamma_{\pm}(s)$ then $\alpha_{\pm}(P) = s$ so $\alpha_{\pm} \circ \gamma_{\pm}$ is the identity hence reductive i.e. $\alpha_{\pm} \circ \gamma_{\pm}(s) \sqsubseteq_{\pm} s$ since \sqsubseteq_{\pm} is reflexive. (3.35)

It follows that

$$\begin{aligned} & \alpha_{\pm}(P) \sqsubseteq_{\pm} \bar{P} \\ \Rightarrow & \gamma_{\pm} \circ \alpha_{\pm}(P) \subseteq \gamma_{\pm}(\bar{P}) && \text{(\gamma_{\pm} is increasing and def. function composition \circ)} \\ \Rightarrow & P \subseteq \gamma_{\pm}(\bar{P}) && \text{(\gamma_{\pm} \circ \alpha_{\pm} is extensive and \subseteq transitive)} \\ \Rightarrow & \alpha_{\pm}(P) \sqsubseteq_{\pm} \alpha_{\pm} \circ \gamma_{\pm}(\bar{P}) && \text{(\alpha_{\pm} is increasing and def. function composition \circ)} \\ \Rightarrow & \alpha_{\pm}(P) \sqsubseteq_{\pm} \bar{P} && \text{(\alpha_{\pm} \circ \gamma_{\pm} is reductive and def. function composition \circ)} \quad \square \end{aligned}$$

Similar results hold for $\langle \check{\alpha}_{\pm}, \check{\gamma}_{\pm} \rangle$, and $\langle \check{\alpha}_{\pm}, \check{\gamma}_{\pm} \rangle$, see Exercise 3.37.

3.19 Galois connection

The abstraction/concretization functions $\langle \alpha_{\pm}, \gamma_{\pm} \rangle$ satisfy $\forall P \in \wp(\mathbb{Z}) . \forall \bar{P} \in \mathbb{P}^{\pm} . \alpha_{\pm}(P) \sqsubseteq_{\pm} \bar{P} \Leftrightarrow P \subseteq \gamma_{\pm}(\bar{P})$, which is the definition of a Galois connection, which we write $\langle \wp(\mathbb{Z}), \subseteq \rangle \xrightleftharpoons[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{P}^{\pm}, \sqsubseteq_{\pm} \rangle$.

More generally,

Definition 3.36 (Galois connection) a Galois connection $\langle \mathbb{P}, \sqsubseteq \rangle \xrightleftharpoons[\alpha]{\gamma} \langle \bar{\mathbb{P}}, \bar{\sqsubseteq} \rangle$ is such that the concrete domain $\langle \mathbb{P}, \sqsubseteq \rangle$ and the abstract domain $\langle \bar{\mathbb{P}}, \bar{\sqsubseteq} \rangle$ are partial orders, $\alpha \in \mathbb{P} \rightarrow \bar{\mathbb{P}}$ is the

abstraction function, $\gamma \in \overline{\mathbb{P}} \rightarrow \mathbb{P}$ is the concretization function, and $\forall P \in \mathbb{P} . \forall \overline{P} \in \overline{\mathbb{P}} . \alpha(P) \sqsubseteq \overline{P} \Leftrightarrow P \sqsubseteq \gamma(\overline{P})$. \square

Exercise 3.37. Prove that $\langle \wp(\mathbb{V} \rightarrow \mathbb{Z}), \sqsubseteq \rangle \xrightarrow[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{V} \rightarrow \mathbb{P}^{\pm}, \dot{\sqsubseteq}_{\pm} \rangle$, and $\langle \wp((\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z}), \sqsubseteq \rangle \xrightarrow[\alpha_{\pm}]{\gamma_{\pm}} \langle (\mathbb{V} \rightarrow \mathbb{P}^{\pm}) \rightarrow \mathbb{P}^{\pm}, \dot{\sqsubseteq}_{\pm} \rangle$. \square

3.20 Calculational design of the sign semantics of expressions

The soundness requirement in Section 3.15 is that $\forall A \in \mathcal{A} . \mathcal{S}^c[A] \subseteq \dot{\gamma}_{\pm}(\mathcal{S}^{\pm}[A])$. By the Galois connection of Exercise 3.37, this is equivalent to $\dot{\alpha}_{\pm}(\mathcal{S}^c[A]) \dot{\sqsubseteq}_{\pm} \mathcal{S}^{\pm}[A]$. Therefore the sign semantics is a sign abstraction of the collecting semantics. It follows that we can design $\mathcal{S}^{\pm}[A]$ by calculus, calculating $\dot{\alpha}_{\pm}(\mathcal{S}^c[A])$ using $\dot{\sqsubseteq}_{\pm}$ -over-approximation to avoid all computations made in the concrete domain.

- We first consider the case when $\exists x \in \mathbb{V} . \dot{\rho}(x) = \perp_{\pm}$ so that $\dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset$.

$$\begin{aligned}
 & - \dot{\alpha}_{\pm}(\mathcal{S}^c[A])\dot{\rho} \\
 &= \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[A] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{[def. (3.32) of } \dot{\alpha}_{\pm}] \\
 &= \alpha_{\pm}(\{\mathcal{A}[A](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{[def. (3.11) of } \mathcal{S}^c[A]] \\
 &= \alpha_{\pm}(\emptyset) && \text{[}\exists x \in \mathbb{V} . \dot{\rho}(x) = \perp_{\pm} \text{ so that } \dot{\gamma}_{\pm}(\dot{\rho}) = \emptyset] \\
 &= \perp_{\pm} && \text{[def. (3.28) of } \alpha_{\pm}] \\
 &\triangleq \mathcal{S}^{\pm}[A]\dot{\rho} \\
 &\quad \text{[in accordance with (3.19) such that, by Exercise 3.20, } \exists x \in \mathbb{V} . \dot{\rho}(x) = \perp_{\pm} \text{ implies} \\
 &\quad \mathcal{S}^{\pm}[A]\dot{\rho} = \perp_{\pm}.]
 \end{aligned}$$

- Then we consider the case when $\forall x \in \mathbb{V} . \dot{\rho}(x) \neq \perp_{\pm}$ so that $\dot{\gamma}_{\pm}(\dot{\rho}) \neq \emptyset$. We proceed by structural induction on A .

- For the basic case of a constant 1 , we just apply the definitions.

$$\begin{aligned}
 & - \dot{\alpha}_{\pm}(\mathcal{S}^c[1])\dot{\rho} \\
 &= \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[1] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{[def. (3.32) of } \dot{\alpha}_{\pm}] \\
 &= \alpha_{\pm}(\{\mathcal{A}[1](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{[def. (3.11) of } \mathcal{S}^c[1]] \\
 &= \alpha_{\pm}(\{1\}) && \text{[}\dot{\gamma}_{\pm}(\dot{\rho}) \text{ is not empty and def. (3.4) of } \mathcal{A}[1]] \\
 &= >0 && \text{[def. (3.28) of } \alpha_{\pm}] \\
 &\triangleq \mathcal{S}^{\pm}[1]\dot{\rho} && \text{[in accordance with (3.19) when } \forall y \in \mathbb{V} . \dot{\rho}(y) \neq \perp_{\pm}]
 \end{aligned}$$

— For the basic case of a variable x , we apply the definitions and then simplify.

$$\begin{aligned}
& \ddot{\alpha}_{\pm}(\mathcal{S}^c[x])^{\ddagger} \dot{\rho} \\
&= \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[x] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.32) of } \ddot{\alpha}_{\pm}\}} \\
&= \alpha_{\pm}(\{\mathcal{A}[x](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.11) of } \mathcal{S}^c[x]\}} \\
&= \alpha_{\pm}(\{\rho(x) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.4) of } \mathcal{A}[x]\}} \\
&= \alpha_{\pm}(\{\rho(x) \mid \forall y \in \mathbb{V}. \rho(y) \in \gamma_{\pm}(\dot{\rho}(y))\}) && \text{\{def. (3.22) of } \dot{\gamma}_{\pm}\}} \\
&= \alpha_{\pm}(\{\rho(x) \mid \rho(x) \in \gamma_{\pm}(\dot{\rho}(x))\}) \\
&\quad \text{\{since } \gamma_{\pm}(\dot{\rho}(y)) \text{ is not empty so for } y \neq x, \rho(y) \text{ can be chosen arbitrarily to satisfy } \\
&\quad \rho(y) \in \gamma_{\pm}(\dot{\rho}(y))\}} \\
&= \alpha_{\pm}(\{x \mid x \in \gamma_{\pm}(\dot{\rho}(x))\}) && \text{\{letting } x = \rho(x)\}} \\
&= \alpha_{\pm}(\gamma_{\pm}(\dot{\rho}(x))) && \text{\{since } S = \{x \mid x \in S\} \text{ for any set } S\}} \\
&= \dot{\rho}(x) && \text{\{by (3.35), } \alpha_{\pm} \circ \gamma_{\pm} \text{ is the identity}\}} \\
&\triangleq \mathcal{S}^{\pm}[x]^{\ddagger} \dot{\rho} && \text{\{in accordance with (3.19) when } \forall y \in \mathbb{V}. \dot{\rho}(y) \neq \perp_{\pm}\}}
\end{aligned}$$

— For the inductive case of $A_1 - A_2$, we assume, by structural induction hypothesis, that

$$\begin{aligned}
& \ddot{\alpha}_{\pm}(\mathcal{S}^c[A_1]) \sqsubseteq_{\pm} \mathcal{S}^{\pm}[A_1] \text{ and } \ddot{\alpha}_{\pm}(\mathcal{S}^c[A_2]) \sqsubseteq_{\pm} \mathcal{S}^{\pm}[A_2] \\
& \ddot{\alpha}_{\pm}(\mathcal{S}^c[A_1 - A_2])^{\ddagger} \dot{\rho} \\
&= \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[A_1 - A_2] \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.32) of } \ddot{\alpha}_{\pm}\}} \\
&= \alpha_{\pm}(\{\mathcal{A}[A_1 - A_2](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.11) of } \mathcal{S}^c[A_1 - A_2]\}} \\
&= \alpha_{\pm}(\{\mathcal{A}[A_1](\rho) - \mathcal{A}[A_2](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) && \text{\{def. (3.4) of } \mathcal{A}\}} \\
&\sqsubseteq_{\pm} \alpha_{\pm}(\{x - y \mid x \in \{\mathcal{A}[A_1](\rho') \mid \rho' \in \dot{\gamma}_{\pm}(\dot{\rho})\} \wedge y \in \{\mathcal{A}[A_2](\rho'') \mid \rho'' \in \dot{\gamma}_{\pm}(\dot{\rho})\}\} \\
&\quad \text{\{ } \{f(\rho) - g(\rho) \mid \rho \in R\} \subseteq \{x - y \mid x \in \{f(\rho') \mid \rho' \in R\} \wedge y \in \{g(\rho'') \mid \rho'' \in R\}\} \text{ and } \\
&\quad \alpha_{\pm} \text{ is increasing by (3.33).}\}}
\end{aligned}$$

This over-approximation allows for A_1 and A_2 to be evaluated in the concrete with different environments ρ' and ρ'' with the same sign of variables but possibly different values of variables. This accounts for the fact that the rule of signs does not take relationships between values of variables into account. For example the sign of $x - x$ is not $=0$ in general, see Exercise 3.15.)

$$\begin{aligned}
& \sqsubseteq_{\pm} \alpha_{\pm}(\{x - y \mid x \in \gamma_{\pm}(\alpha_{\pm}(\{\mathcal{A}[A_1](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\})) \wedge y \in \gamma_{\pm}(\alpha_{\pm}(\{\mathcal{A}[A_2](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}))\}) \\
&\quad \text{\{ } \{x - y \mid x \in P \wedge y \in Q\} \subseteq \{x - y \mid x \in \gamma_{\pm}(\alpha_{\pm}(P)) \wedge y \in \gamma_{\pm}(\alpha_{\pm}(Q))\} \text{ since } \gamma_{\pm} \circ \alpha_{\pm} \text{ is } \\
&\quad \text{extensive by (3.34) and } \alpha_{\pm} \text{ is increasing by (3.33).}\}}
\end{aligned}$$

This over-approximation allows for the evaluation of the sign to be performed in the abstract with \neg_{\pm} instead of the concrete.)

$$= \alpha_{\pm}(\{\mathcal{A}[A_1](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) \neg_{\pm} \alpha_{\pm}(\{\mathcal{A}[A_2](\rho) \mid \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\})$$

$$\begin{aligned}
& \{s_1 \dot{-}_\pm s_2 = \alpha_\pm(\{x - y \mid x \in \gamma_\pm(s_1) \wedge y \in \gamma_\pm(s_2)\})\} \text{ by Exercise 3.29} \\
& = \alpha_\pm(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[A_1] \wedge \rho \in \dot{\gamma}_\pm(\dot{\rho})\}) \dot{-}_\pm \alpha_\pm(\{\mathcal{S}(\rho) \mid \mathcal{S} \in \mathcal{S}^c[A_2] \wedge \rho \in \dot{\gamma}_\pm(\dot{\rho})\}) \text{ (def. (3.11))} \\
& \quad \text{of } \mathcal{S}^c \} \\
& = \ddot{\alpha}_\pm(\mathcal{S}^c[A_1])\dot{\rho} \dot{-}_\pm \ddot{\alpha}_\pm(\mathcal{S}^c[A_2])\dot{\rho} \text{ (def. (3.32) of } \ddot{\alpha}_\pm \text{)} \\
& = \ddot{\alpha}_\pm(\mathcal{S}^c[A_1])\dot{\rho} \dot{-}_\pm \ddot{\alpha}_\pm(\mathcal{S}^c[A_2])\dot{\rho} \text{ (def. (3.32) of } \ddot{\alpha}_\pm \text{)} \\
& \sqsubseteq_\pm (\mathcal{S}^+[A_1]\dot{\rho}) \dot{-}_\pm (\mathcal{S}^+[A_2]\dot{\rho}) \\
& \quad \{ \text{induction hypothesis and } \dot{-}_\pm \text{ is increasing in both parameters by Exercise 3.27} \} \\
& \triangleq \mathcal{S}^+[A_1 - A_2]\dot{\rho} \quad \{ \text{in accordance with (3.19) when } \forall y \in \mathcal{V} . \dot{\rho}(y) \neq \perp_\pm \} \quad \square
\end{aligned}$$

3.21 Calculational design of abstract interpretations

This concludes our formal design of the rule of signs for arithmetic expressions.

- We first define the semantics $\mathcal{A}[A]$ of arithmetic expressions A in (3.4);
- The strongest property of the semantics of arithmetic expressions A is their collecting semantics $\mathcal{S}^c[A]$ in (3.11);
- Among the semantic properties $\wp((\mathcal{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ of arithmetic expressions, we select a subset of properties of interest *i.e.* the sign properties and choose a computer representation, as defined by the abstraction function $\ddot{\alpha}_\pm$ in (3.32), which is the lower adjoint of the Galois connection (??);
- The rule of sign $\mathcal{S}^+[A]$ is then formally derived by calculational design in Section 3.20 by over-approximating the best abstraction $\ddot{\alpha}_\pm(\mathcal{S}^c[A])$ of the collecting semantics $\mathcal{S}^c[A]$.

It follows that $\mathcal{S}^+[A]$ is sound by construction.

3.22 Conclusion

We have defined the semantics of expressions, their properties, a proof method, and a sign analysis.

Instead of designing the rule of sign empirically and then proving its soundness (as proposed in Exercise 3.25), we used the soundness requirement as a guideline for designing the abstract sign semantics by calculus.

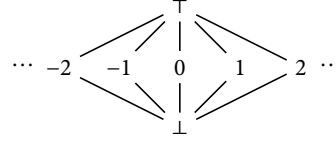
This sign analysis discovers an abstract property of an arithmetic expression by computing in the abstract only. This may involve some loss of precision, which was the case for the sign analysis.

The sign semantics is finite so it is an easily implementable static analysis (see Exercise 5.11) whereas exhaustive case analysis would not scale up (see Exercise 3.42).

3.23 Exercises

Exercise 3.38 (Parity). Design the parity analysis of expressions $E \in \mathcal{E}$. Show that the parity analysis is correct with machine integers. \square

Exercise 3.39 (Constancy analysis). Design the analysis of expressions $E \in \mathcal{E}$ using the opposite constancy properties such that $\gamma(\perp) = \emptyset$, $\gamma(i) = \{i\}$, $i \in \mathbb{Z}$, and $\gamma(\top) = \mathbb{Z}$ [9].



Exercise 3.40 (Typing). Extend the semantics of expressions with the real constant `1.0` and a generic minus operation with implicit conversion from `nat` to `int` to `real` where $\gamma(\text{nat}) \triangleq \mathbb{N}$, $\gamma(\text{int}) \triangleq \mathbb{Z}$, and $\gamma(\text{real}) \triangleq \mathbb{R}$. Design a type analysis for these extended expressions [11]. \square

Exercise 3.41. Define the cartesian set transformer $\mathcal{P}[\![A]\!]$ of an arithmetic expression A to be

$$\begin{aligned} \mathcal{P}[\![A]\!] &\in (\mathcal{V} \rightarrow \wp(\mathbb{Z})) \rightarrow \wp(\mathbb{Z}) \\ \mathcal{P}[\![A]\!] &\triangleq P \mapsto \{\mathcal{A}[\![A]\!]\rho \mid \rho \in \gamma_x(P)\}. \end{aligned}$$

For example if $P(x) = \{0, 1\}$ and $P(y) = \{-1, 0\}$ then $\mathcal{P}[\![x - y]\!]P = \{0, 1, 2\}$ and $\mathcal{P}[\![x - x]\!]P = \{-1, 0, 1\}$.

Prove that

$$\begin{aligned} \mathcal{P}[\![1]\!]P &\triangleq \{1\} \\ \mathcal{P}[\![x]\!]P &\triangleq P(x) \\ \mathcal{P}[\![A_1 - A_2]\!]P &\triangleq \{x - y \mid x \in \mathcal{P}[\![A_1]\!]P \wedge y \in \mathcal{P}[\![A_2]\!]P\} \\ \mathcal{P}[\![A_1 < A_2]\!]P &\triangleq \{x < y \mid x \in \mathcal{P}[\![A_1]\!]P \wedge y \in \mathcal{P}[\![A_2]\!]P\} \\ \mathcal{P}[\![B_1 \text{ nand } B_2]\!]P &\triangleq \{x \uparrow y \mid x \in \mathcal{P}[\![B_1]\!]P \wedge y \in \mathcal{P}[\![B_2]\!]P\} \end{aligned} \quad \square$$

Exercise 3.42 (Model checking). Implement the cartesian property transformer of Exercise 3.41 in the language of your choice. In order to prove that $x - y = 0$ when $x = y$, check that $\mathcal{P}[\![x - y]\!]P = \{0\}$ for all P such that $P(x) = \{i\}$ and $P(y) = \{i\}$, $i \in 2^p - 1$ where $p = 2, 3, \dots, 32$ is the bit size of integers. Observe the state explosion problem [5]. \square

3.24 Answers to selected exercises

Answer of exercise 3.1. $\frac{0}{0}$ and more generally $\frac{z}{0}$, $z \in \mathbb{Z}$ is undefined (computer scientists would say the computation has a “runtime error”). \square

Answer of exercise 3.2. 0 is pos, -1 is neg, the sign of $0 \times -1 = 0$ is pos, in contradiction with the rule $\text{pos} \times \text{neg} = \text{neg}$. \square

Answer of exercise 3.3.

$$\begin{aligned}
\text{vars}[1] &\triangleq \emptyset \\
\text{vars}[x] &\triangleq \{x\} \\
\text{vars}[A_1 - A_2] &\triangleq \text{vars}[A_1] \cup \text{vars}[A_2] \\
\text{vars}[A_1 < A_2] &\triangleq \text{vars}[A_1] \cup \text{vars}[A_2] \\
\text{vars}[B_1 \text{ nand } B_2] &\triangleq \text{vars}[B_1] \cup \text{vars}[B_2]
\end{aligned}$$

□

Answer of exercise 3.6. In OCaml [10], we have

```

# type aexpr = Num of int | Minus of aexpr * aexpr;;
# type bexpr = Lt of aexpr * aexpr | Nand of bexpr * bexpr;;

# let rec eval_aexpr a = match a with
  | Num i -> i
  | Minus (a1,a2) -> (eval_aexpr a1) - (eval_aexpr a2)
let rec eval_bexpr b = match b with
  | Lt (a1,a2) -> (eval_aexpr a1) < (eval_aexpr a2)
  | Nand (b1,b2) -> not((eval_bexpr b1) && (eval_bexpr b2));;

# eval_bexpr (Lt ((Minus (Num 1, Num 2)), (Minus (Num 0, Num 2))));;
- : bool = false

```

□

Answer of exercise 3.7. For all environments ρ , $\mathcal{A} \llbracket 1 - 1 \rrbracket \rho = 0$. If $\mathcal{A} \llbracket A \rrbracket \rho = n$ then $\mathcal{A} \llbracket A - ((1 - 1) - 1) \rrbracket \rho = n + 1$ and $\mathcal{A} \llbracket (1 - 1) - A \rrbracket \rho = -n$.

□

Answer of exercise 3.14. The loop body in $x = 0; \text{while } (0) \{ x = x+1 \}$ is never executed to the set of possible values of x in this loop body is empty so its sign is \perp_+ .

□

Answer of exercise 3.15. Consider $42 - 42 = 0$. The sign is $(>0) \neg_+ (>0) = \top_+$ not $=0$.

□

Answer of exercise 3.20. This is true for $A = 1$ and $A = x$ by definition of $\mathbb{I}^+[\rho]$. Assume, by structural induction that this is true for A_1 and A_2 . Then $\mathcal{S}^+ \llbracket A_1 - A_2 \rrbracket \rho = (\mathcal{S}^+ \llbracket A_1 \rrbracket \rho) \neg_+ (\mathcal{S}^+ \llbracket A_2 \rrbracket \rho) = \perp_+ \neg_+ \perp_+ = \perp_+$ by induction hypothesis and def. of \neg_+ .

□

Answer of exercise 3.29. We can use the soundness requirement as a definition of $s_1 \neg_+ s_2 \triangleq \alpha_+ (\{x - y \mid x \in \gamma_+(s_1) \wedge y \in \gamma_+(s_2)\})$ to design \neg_+ by calculus. We have to consider all possible cases for s_1 and s_2 . We show three cases $\top_+ \neg_+ \perp_+ = \perp_+$, $<0 \neg_+ \geq 0 = <0$, and $\geq 0 \neg_+ \geq 0 = \top_+$.

$$\begin{aligned}
& - \alpha_+ (\{x - y \mid x \in \gamma_+(\top_+) \wedge y \in \gamma_+(\perp_+)\}) \\
& = \alpha_+ (\{x - y \mid x \in \mathbb{Z} \wedge y \in \emptyset\}) && \{\text{def. } \gamma_+\} \\
& = \alpha_+ (\emptyset) = \perp_+ && \{\text{def. } \alpha_+\} \\
& - \alpha_+ (\{x - y \mid x \in \gamma_+(<0) \wedge y \in \gamma_+(\geq 0)\}) \\
& = \alpha_+ (\{x - y \mid x < 0 \wedge y \geq 0\}) && \{\text{def. } \gamma_+\}
\end{aligned}$$

$$\begin{aligned}
&= \alpha_{\pm}(\{x \mid x < 0\}) = <0 && \text{[def. } \alpha_{\pm} \text{]} \\
&= \alpha_{\pm}(\{x - y \mid x \in \gamma_{\pm}(\geq 0) \wedge y \in \gamma_{\pm}(\geq 0)\}) \\
&= \alpha_{\pm}(\{x - y \mid x \geq 0 \wedge y \geq 0\}) && \text{[def. } \gamma_{\pm} \text{]} \\
&= \alpha_{\pm}(\mathbb{Z}) = \top_{\pm} && \text{[def. } \alpha_{\pm} \text{]}
\end{aligned}$$

The calculations can be formally certified by a proof verifier [3, 8].

One can also consider all cases $s \in \mathbb{P}^{\pm}$ for $s_1 \dashv_{\pm} s_2$ for given s_1, s_2 when needed, using a theorem prover to make the proof that $\{x - y \mid x \in \gamma_{\pm}(s_1) \wedge y \in \gamma_{\pm}(s_2)\} \subseteq \gamma_{\pm}(s)$, and returning \top_{\pm} when the proof fails (e.g. times out). Among all possible answers s for which the theorem prover could make the proof, the \sqsubseteq_{\pm} -minimal one is chosen, if any. Otherwise, an arbitrary \sqsubseteq_{\pm} -minimal one has to be selected. This is called predicate abstraction [7].

□

Answer of exercise 3.37. For all $P \in \wp(\mathbb{V} \rightarrow \mathbb{Z})$ and $\dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm}$, we have

$$\begin{aligned}
&\dot{\alpha}_{\pm}(P) \sqsubseteq_{\pm} \dot{\rho} \\
&\Leftrightarrow \forall x \in \mathbb{V} . \dot{\alpha}_{\pm}(P)x \sqsubseteq_{\pm} \dot{\rho}(x) && \text{[pointwise def. of } \sqsubseteq_{\pm} \text{]} \\
&\Leftrightarrow \forall x \in \mathbb{V} . \alpha_{\pm}(\{\rho(x) \mid \rho \in P\}) \sqsubseteq_{\pm} \dot{\rho}(x) && \text{[def. (3.31) of } \dot{\alpha}_{\pm} \text{]} \\
&\Leftrightarrow \forall x \in \mathbb{V} . \{\rho(x) \mid \rho \in P\} \subseteq \gamma_{\pm}(\dot{\rho}(x)) && \text{[} \langle \wp(\mathbb{Z}), \subseteq \rangle \xleftrightarrow[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{P}^{\pm}, \sqsubseteq_{\pm} \rangle \text{]} \\
&\Leftrightarrow \forall x \in \mathbb{V} . \forall \rho \in P . \rho(x) \in \gamma_{\pm}(\dot{\rho}(x)) && \text{[def. } \in \text{]} \\
&\Leftrightarrow \forall \rho \in P . \forall x \in \mathbb{V} . \rho(x) \in \gamma_{\pm}(\dot{\rho}(x)) && \text{[def. } \forall \text{]} \\
&\Leftrightarrow P \subseteq \{\rho \in \mathbb{V} \rightarrow \mathbb{Z} \mid \forall x \in \mathbb{V} . \rho(x) \in \gamma_{\pm}(\dot{\rho}(x))\} && \text{[def. } \subseteq \text{]} \\
&\Leftrightarrow P \subseteq \dot{\gamma}_{\pm}(\dot{\rho}) && \text{[def. (3.22) of } \dot{\gamma}_{\pm} \text{, proving } \langle \wp(\mathbb{V} \rightarrow \mathbb{Z}), \subseteq \rangle \xleftrightarrow[\dot{\alpha}_{\pm}]{\dot{\gamma}_{\pm}} \langle \mathbb{V} \rightarrow \mathbb{P}^{\pm}, \sqsubseteq_{\pm} \rangle \text{]}
\end{aligned}$$

For all $P \in \wp((\mathbb{V} \rightarrow \mathbb{Z}) \rightarrow \mathbb{Z})$ and $\bar{P} \in (\mathbb{V} \rightarrow \mathbb{P}^{\pm}) \rightarrow \mathbb{P}^{\pm}$, we have

$$\begin{aligned}
&\ddot{\alpha}_{\pm}(P) \sqsubseteq_{\pm} \bar{P} \\
&\Leftrightarrow \forall \dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} . \ddot{\alpha}_{\pm}(P)\dot{\rho} \sqsubseteq_{\pm} \bar{P}(\dot{\rho}) && \text{[pointwise def. of } \sqsubseteq_{\pm} \text{]} \\
&\Leftrightarrow \forall \dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} . \alpha_{\pm}(\{\mathcal{S}(\rho) \mid \mathcal{S} \in P \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\}) \sqsubseteq_{\pm} \bar{P}(\dot{\rho}) && \text{[def. (3.32) of } \ddot{\alpha}_{\pm} \text{]} \\
&\Leftrightarrow \forall \dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} . \{\mathcal{S}(\rho) \mid \mathcal{S} \in P \wedge \rho \in \dot{\gamma}_{\pm}(\dot{\rho})\} \subseteq \gamma_{\pm}(\bar{P}(\dot{\rho})) && \text{[} \langle \wp(\mathbb{Z}), \subseteq \rangle \xleftrightarrow[\alpha_{\pm}]{\gamma_{\pm}} \langle \mathbb{P}^{\pm}, \sqsubseteq_{\pm} \rangle \text{]} \\
&\Leftrightarrow \forall \dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} . \forall \mathcal{S} \in P . \forall \rho \in \dot{\gamma}_{\pm}(\dot{\rho}) . \mathcal{S}(\rho) \in \gamma_{\pm}(\bar{P}(\dot{\rho})) && \text{[def. } \subseteq \text{]} \\
&\Leftrightarrow \forall \mathcal{S} \in P . \forall \dot{\rho} \in \mathbb{V} \rightarrow \mathbb{P}^{\pm} . \forall \rho \in \dot{\gamma}_{\pm}(\dot{\rho}) . \mathcal{S}(\rho) \in \gamma_{\pm}(\bar{P}(\dot{\rho})) && \text{[def. } \forall \text{]} \\
&\Leftrightarrow \forall \mathcal{S} \in P . \mathcal{S} \in \dot{\gamma}_{\pm}(\bar{P}) && \text{[def. } \in \text{ and (3.23) of } \dot{\gamma}_{\pm} \text{]} \\
&\Leftrightarrow P \subseteq \dot{\gamma}_{\pm}(\bar{P})
\end{aligned}$$

$$\{ \text{def. } \subseteq, \text{ proving } \langle \wp((V \rightarrow Z) \rightarrow Z), \subseteq \rangle \xrightarrow[\alpha_{\pm}]{\beta_{\pm}} \langle (V \rightarrow P^{\pm}) \rightarrow P^{\pm}, \subseteq_{\pm} \rangle. \}$$

Answer of exercise 3.42. The model checking time is exponential in the bit size of integers.

```
$ cat CartesianTransformer.py
import time
import matplotlib.pyplot as pyplot
from sets import *
```

```
# i integer
# 'x' variables
# e ::= ('INT', i) | ('VAR', 'x') |
# ('MINUS', e1, e2)
```

```
maxp = 32
natural = Set(range(0, 2 ** maxp))
```

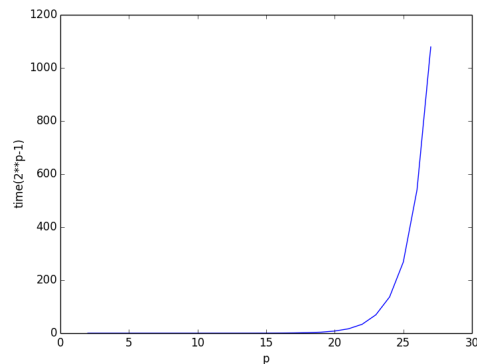
```
def eval (e, P):
    if e[0] == 'INT':
        return Set([e[1]])
    if e[0] == 'VAR':
        return P(e[1])
    if e[0] == 'MINUS':
        P1 = eval(e[1], P)
        P2 = eval(e[2], P)
        res = []
        for x in P1:
            for y in P2:
                res = res + [x - y]
        return Set(res)
    handle_error()
```

```
expr = ('MINUS', ('VAR', 'x'), ('VAR', 'y'))
```

```
def makeP(i):
    def P(x):
        if x == "x":
            return Set([i])
        if x == "y":
            return Set([i])
        return Set([0])
    return P
```

```
x = []
```

```
y = []
result = Set([])
for p in range (2, maxp):
    start_time = time.time()
    for i in range (0, ((2 ** p) - 1)):
        P = makeP(i)
        result = result | eval (expr, P)
    finish_time = time.time()
    x = x + [p]
    t = (finish_time - start_time)
    y = y + [t]
    print p, result, t, "seconds"
pyplot.plot(x,y)
pyplot.xlabel('p')
pyplot.ylabel('time(2**p-1)')
pyplot.savefig('CartesianTransformer.png')
$ /usr/bin/python CartesianTransformer.py
2 Set([0]) 0.000694036483765 seconds
3 Set([0]) 0.000127077102661 seconds...
25 Set([0]) 262.113693953 seconds
26 Set([0]) 537.459581137 seconds
27 Set([0]) 1062.62982702 seconds
$
```



□

3.25 Bibliography

- [1] Garrett Birkhoff. *Lattice Theory*. Third edition. American Mathematical Society, Colloquium publications, Volume XXV, 1973 (7, 2).
- [2] Rod M. Burstall. “Proving properties of programs by structural induction”. *Computer Journal* 12.1 (1969), pp. 41–48.
- [3] David Cachera and David Pichardie. “Programmation d’un interpréteur abstrait certifié en logique constructive”. *Technique et Science Informatiques* 30.4 (2011), pp. 381–408 (18).
- [4] Noam Chomsky. “Three models for the description of language”. *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. URL: <http://dx.doi.org/10.1109/TIT.1956.1056813> (4, 1).
- [5] Edmund M. Clarke, William Klieber, Milos Nováček, and Paolo Zuliani. “Model Checking and the State Explosion Problem”. In: *LASER Summer School*. Vol. 7682. Lecture Notes in Computer Science. Springer, 2011, pp. 1–30 (16).
- [6] Patrick Cousot and Radhia Cousot. “Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints”. In: *POPL*. ACM, 1977, pp. 238–252 (7, 4, 10, 9, 12, 15).
- [7] Susanne Graf and Hassen Saïdi. “Verifying Invariants Using Theorem Proving”. In: *CAV*. Vol. 1102. Lecture Notes in Computer Science. Springer, 1996, pp. 196–207 (18).
- [8] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. “A Formally-Verified C Static Analyzer”. In: *POPL*. ACM, 2015, pp. 247–259 (18, 15, 13, 16).
- [9] Gary A. Kildall. “A Unified Approach to Global Program Optimization”. In: *POPL*. ACM Press, 1973, pp. 194–206 (16, 4, 6, 1, 19, 12, 13, 22).
- [10] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. “The OCaml system, release 4.04, Documentation and user’s manual”. Copyright © 2013 Institut National de Recherche en Informatique et en Automatique. Nov. 2016 (17, 9, 14).
- [11] Peter Naur. “Generation of machine code in ALGOL compilers”. *BIT* 5.4 (1960), pp. 235–245 (16).
- [12] Peter Naur. “The design of the GIER ALGOL compiler”. *BIT Numerical Mathematics* 3 (June 1963), 124–140 and 145–166 (2).
- [13] Peter Naur. “Checking of operand types in ALGOL compilers”. *BIT Numerical Mathematics* 5 (Sept. 1965), pp. 151–163 (2, 6, 1, 19).
- [14] Kim Plofker. *Mathematics in India*. Princeton University Press, 2007 (1, 3).
- [15] Dana S. Scott and Christopher Strachey. *Towards a Mathematical Semantics for Computer Languages*. Technical report PRG-6. Oxford University Computer Laboratory, Aug. 1971, p. 49 (4, 1, 9, 3).

3.25 *Bibliography*

21

- [16] Michel Sintzoff. “Calculating Properties of Programs by Valuations on Specific Models”. In: *Proceedings of ACM Conference on Proving Assertions About Programs*. ACM, 1972, pp. 203–207 (2, 6, 10, 1, 19).
- [17] International Organization for Standardization. “ISO/IEC 19761: Software engineering – COSMIC: a functional size measurement method”. Mar. 2011. URL: <https://www.iso.org/standard/54849.html> (6).

Index of names

Brahmagupta, 1

Rod Burstall, 5

Noam Chomsky, 4

Peter Naur, 2

Dana Scott, 4

Michel Sintzoff, 2

Christopher Strachey, 4

Index of cited authors

Birkhoff, Garrett, 7	Graf, Susanne, 18	Plofker, Kim, 1, 3
Blazy, Sandrine, 18		
Burstall, Rod M., 5	Jourdan, Jacques-Henri, 18	Rémy, Didier, 17
Cachera, David, 18	Kildall, Gary A., 16	Saïdi, Hassen, 18
Chomsky, Noam, 4	Klieber, William, 16	Scott, Dana S., 4
Clarke, Edmund M., 16		Sintzoff, Michel, 2
Cousot, Patrick, 7	Laporte, Vincent, 18	Standardization, International
Cousot, Radhia, 7	Leroy, Xavier, 17, 18	Organization for, 6
		Strachey, Christopher, 4
Doligez, Damien, 17	Naur, Peter, 2, 16	
Frisch, Alain, 17	Nováček, Milos, 16	Vouillon, Jérôme, 17
Garrigue, Jacques, 17	Pichardie, David, 18	Zuliani, Paolo, 16

Index of concepts

abstract domain	environment, 4	sign _{_,} 7
sign _{_,} 7	expression	property
abstraction, 12	arithmetic _{_,} 3	expression _{_,} 6
best _{_,} 11, 12	semantics, 5	semantic _{_,} 6
sign _{_,} 11, 12	boolean _{_,} 3	syntactic _{_,} 6
analysis	semantics, 5	
constancy _{_,} 16	semantics of _{_,} s, 5	semantics
parity _{_,} 15	syntax of _{_,} s, 3	collecting _{_,} 6, 13
sign _{_,} 8		denotational _{_,} 4
typing _{_,} 16	Galois connection, 12, 13	sign _{_,} 8
associativity	grammar	sign, 1
left _{_,} 3	rule, 3	concretization, 9, 12
		lattice, 10
calculational design, 13	model checking, 16	analysis, 2
concretization, 12		rule of _{_,} 1
constancy, 16	OCaml programming	soundness, 9
constant, 3	language, 17	strictness
		bottom _{_,} 8
definition	proof	syntax
compositional _{_,} 4	by recurrence, 5	of expressions, 3
structural _{_,} 4, 5	by structural induction, 5, 7	
design	of program properties, 7	variable
empirical _{_,} 15	properties	free _{_,} 4

Index of symbols

- \mathbb{D}_\pm : sign abstract domain, 7
 \mathcal{A} : arithmetic expression semantics, 5
 \mathcal{A} : set of all arithmetic expressions, 3

 \mathcal{B} : boolean expression semantics, 5
 \mathcal{B} : set of all boolean expressions, 3

 \triangleq : defined as, 4

 $\xleftrightarrow[\alpha]{\gamma}$: Galois connection, 12

 \mathbb{Z} : mathematical integers, 4

 $-$: arithmetic difference, 3

 nand : not and, 3
 \uparrow : not and, 5

 $\alpha_\pm(\dot{\alpha}_\pm, \ddot{\alpha}_\pm)$: sign abstraction (pointwise), 11, 12

 \perp_\pm : bottom sign, 7
 $\gamma_\pm(\dot{\gamma}_\pm, \ddot{\gamma}_\pm)$: sign concretization (pointwise), 9, 12
 ρ^\pm : sign environment, 8
 \sqcup_\pm : sign join, 11
 $-\pm$: sign minus, 7
 ≤ 0 : negative sign, 7
 < 0 : strictly negative sign, 7
 $\sqsubseteq_\pm (\dot{\sqsubseteq}_\pm)$: sign partial order (pointwise), 10, 12
 ≥ 0 : positive sign, 7
 > 0 : strictly positive sign, 7
 \mathcal{P}^\pm : concrete sign properties, 10
 \mathcal{S}^\pm : sign semantics, 8, 13
 \downarrow^\pm : sign bottom strictness, 8
 \top_\pm : top sign, 7
 $= 0$: zero sign, 7
 $\neq 0$: non-zero sign, 7

 \mathcal{V} : set of all variables, 3
 vars : free variables of expressions, 4