« Forward Non-relational Infinitary Static Analysis »

Patrick Cousot

Jerome C. Hunsaker Visiting Professor
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics

cousot@mit.edu
www.mit.edu/~cousot

Course 16.399: "Abstract interpretation"

http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/

THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

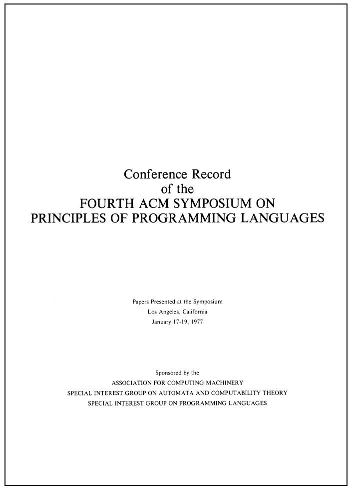
par

Patrick COUSOT

METHODES ITERATIVES DE CONSTRUCTION
ET D'APPROXIMATION DE PCINTS FIXES
D'OPERATEURS MONOTONES SUR UN TREILLIS,
ANALYSE SEMANTIQUE DES PROGRAMMES.

Thèse soutenue le 21 mars 1978 devant la Commission d'Examen :

Président : L. BOLLIET
Examinateurs : C. BENZAKEN
Ph. JORRAND
B. LORHO
C. PAIR
F. ROBERT
M. SINTZOFF



Conference Record
of the
FOURTH ACM SYMPOSIUM ON
PRINCIPLES OF PROGRAMMING LANGUAGES

Papers Presented at the Symposium
Los Angeles, California
January 17-19, 1977

Sponsored by the
ASSOCIATION FOR COMPUTING MACHINERY
SPECIAL INTEREST GROUP ON AUTOMATA AND COMPUTABILITY THEORY
SPECIAL INTEREST GROUP ON PROGRAMMING LANGUAGES

Proving the correctness of static analyzers

- The abstract interpretation theory provides a formal basis for proving the soundness (and sometimes the completeness) of static analyzers (abstract semantics)
- The principle is to proceed by induction on the syntax of programs, which yields a proof for the whole programming language
- This structural proof will be formalized independently of any particular programming language

- The proof relies on the use of a quite general form of collecting semantics, abstraction and abstract semantics as formalized by concretization functions (or abstract functions or Galois connections in case of existence of best abstractions)
- It is based on the use of fixpoint definitions for monotone operators on cpos (complete lattices)
- In absence of ACC, monotony is lost due to the use of widening/narrowing (and indeed monotony must be lost to enforce convergence)

- In this case, despite non-monotony, the structural argument remains valid, replacing fixpoints by convergent iterations with convergence acceleration through widening/narrowing
- Even if the abstract is non-monotone, soundness follows from monotony in the concrete

An abstract formalization of finitary structural analysis by abstract interpretation

An abstract definition of abstract syntax

- The abstract syntax defines a collection $\{\mathrm{Com}_i \mid i \in \Delta\}$ of syntactic categories and a well-founded relation $\prec$ meaning "to be an immediate subcomponent of"
- For example, $\{\mathrm{Com}_i \mid i \in \Delta\} = \{A, B, C\}$ where A are arithmetic expressions, B are boolean expressions, and C is a set of commands. Then $\prec$ is defined by the grammar defining A, B, C.

- In general we have:
  - Syntactic categories:

$$C_i \in \mathrm{Com}_i, \quad i \in \Delta$$

  - Immediate subcomponent relation:

$$\langle \bigcup_{i \in \Delta} \mathrm{Com}_i, \ \prec \rangle \text{ is well} - \text{founded}$$

- Example:

$$C = \texttt{while } B \texttt{ do } C' \texttt{ od}$$
$$B \prec C, \ T(\neg(B)) \prec C, \ C' \prec C$$

An abstract definition of the structural concrete (collecting) semantics

– Concrete domains: define the semantics information associated to each syntactic category Com_i, $i \in \Delta$. For each $i \in \Delta$ and $C_i \in \mathrm{Com}_i$:

$\langle \mathcal{D}_{C_i},\ \sqsubseteq_{C_i},\ \bot_{C_i},\ \sqcup_{C_i} \rangle$ is a poset (cpo, complete lattice, ...)

– Concrete (aka collecting) semantics:

$$\mathcal{C}_i \in [C_i \in \mathrm{Com}_i \mapsto \mathcal{D}_{C_i}]$$

is defined, by structural induction, as

$$\mathcal{C}_i[\![C_i]\!] \stackrel{\text{def}}{=} \mathcal{F}_i[\![C_i]\!]\Big(\prod_{C'_j \prec C_i} \mathcal{C}_j[\![C'_j]\!]\Big)$$

where

$$\mathcal{F}_i[\![C_i]\!] \in \Big(\prod_{C'_j \prec C_i} \mathcal{D}_{C'_j}\Big) \mapsto \mathcal{D}_{C_i}$$

is the collecting semantics transformer

The collecting semantics transformer is defined in the form:

$$\mathcal{F}_i[\![C_i]\!](S_1, \ldots, S_n) \stackrel{\text{def}}{=} e[\![\mathcal{D}_{C_i}]\!][S_1 : \mathcal{D}_{C'_1}, \ldots, S_n : \mathcal{D}_{C'_n}]()$$

where $\{C' \mid C' \prec C_i\} = \{C'_1, \ldots, C'_n\}$ and the right-hand side is an expression written according to the following attribute grammar, where we are given

- $S = S_1 : \mathcal{D}_{C'_1}, \ldots, S_n : \mathcal{D}_{C'_n}$: the collecting semantics of components
- $X = X_{n+1} : \mathcal{D}'_{n+1}, \ldots, X_m : \mathcal{D}'_m$: fixpoint variables
- $\langle \mathcal{D},\ \sqsubseteq,\ \bot,\ \sqcup \rangle$: the domain of the result

The attribute grammar of expressions is as follows:

$$\begin{array}{l} e[\![\mathcal{D}]\!][S](X) ::= \\ \quad \mid d \\ \quad \mid S_j \\ \quad \mid X_k \\ \quad \mid f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}}(e_1[\![\mathcal{D}_{j_1}]\!][S](X)), \ldots, e_\ell[\![\mathcal{D}_{j_\ell}]\!][S](X)) \\ \quad \mid \mathsf{lfp}^{\bot}_{\sqsubseteq}\ \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \end{array}$$ [1]

where

- $d \in \mathcal{D}$ is a constant
- S_j, $j \in [1, n]$ is the semantics of an immediate component of C_i such that $\mathcal{D}_j = \mathcal{D}$

[1] Y must be a new fresh variable

- X_k, $k \in [n+1, m]$ appears inside a fixpoint definition and $\mathcal{D}'_k = \mathcal{D}$
- $f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}} \in \left(\prod_{j=j_1}^{j_\ell} \mathcal{D}_j\right) \mapsto \mathcal{D}$ is a constant function such as $f(x, y) = x \sqcup y$; $x \sqcap y$, $x \circ y$, $x(y)$, etc).
- The existence of the fixpoint definition should be ensured (by def. of the poset $\langle \mathcal{D}, \sqsubseteq, \bot, \sqcup \rangle$ and properties of the function $\lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D})$ (such as monotony, continuity, extensivity, etc).
- In particular **lfp** need not be a fixpoint and can be defined as the limit of an iteration process, a solution of constraints, etc.

- $Y \notin \{X_{n+1}, \ldots, X_m\}$ is a fresh variable and $(X, Y : \mathcal{D})$ is short for $X_{n+1} : \mathcal{D}'_{n+1}, \ldots, X_m : \mathcal{D}'_m, Y : \mathcal{D}$.
- Note that for given S and X, we have $\lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \in \mathcal{D} \mapsto \mathcal{D}$

Notes on typing the structural concrete semantics

- language:
  - We have a set Δ of indexes i of syntactic categories Com_i
  - The language is $\langle \bigcup_{i \in \Delta} \mathrm{Com}_i, \prec \rangle$ where "$\prec$" is the well-founded "immediate component" relation
- types:
  - We can consider a set T of base types
  - The set $\mathbb{T}$ of types is then defined inductively as $\forall t \in T : t \in \mathbb{T}$ and if $t_i \in \mathbb{T}$, $i = 1, \ldots, n+1$ then $t_1 \times \ldots \times t_n \to t_{n+1} \in \mathbb{T}$

- abstract domains:
  - For each type $t \in T$, the (concrete or abstract) semantic domain is $\langle \mathcal{D}_t, \sqsubseteq)_t, \bot_t, \sqcup_t \rangle$
- typing program component:
  - A base type $t_i \in T$ is associated to each program component Com_i, which is written $\mathrm{Com}_i^{t_i}$
  - The intention is that the domain $\langle \mathcal{D}_{t_i}, \sqsubseteq)_{t_i}, \bot_{t_i}, \sqcup_{t_i} \rangle$ describes the possible behaviors of program component Com_i

– typing semantic expressions:

- All expressions are typed:

$$e^t[S_1^{t_1}, \ldots, S_n^{t_n}](X_{n+1}^{t_{n+1}}, \ldots, X_m^{t_m})$$

- These expressions are interpreted as functions:

$$\mathcal{D}_{t_1} \times \ldots \times \mathcal{D}_{t_n} \times \mathcal{D}_{t_{n+1}} \times \ldots \times \mathcal{D}_{t_m} \mapsto \mathcal{D}_t$$

– typing rules for semantic expressions:

- The typing rules for expressions are as follows:
  - · $d^t \in \mathcal{D}_t$, S^t and X^t have type t
  - · f has type $t_1 \times \ldots \times t_n \to t$ (written $f^{t_1 \times \ldots \times t_n \to t}$) if, whenever e_i has type t_i, $i = 1, \ldots, n$, then $f(e_1, \ldots, e_n)$ has type t

- · If F has type $t_1 \times \ldots \times t_n \to t \to t$ then $\mathsf{lfp}^{\perp_t}_{\sqsubseteq)_t} F$ has type $t_1 \times \ldots \times t_n \to t$

– typing rule for semantic expressions:

- It follows, by structural induction, that

$$e^t[S_1^{t_1}, \ldots, S_n^{t_n}](X_{n+1}^{t_{n+1}}, \ldots, X_m^{t_m})$$

belongs to

$$\mathcal{D}_{t_1} \times \ldots \times \mathcal{D}_{t_n} \times \mathcal{D}_{t_{n+1}} \times \ldots \times \mathcal{D}_{t_m} \mapsto \mathcal{D}_t$$

– type soundness for semantic equational definitions:

- In the definition $\mathcal{C}_i^t[\![C_i]\!] \stackrel{\text{def}}{=} F_{C_i}(\prod_{C'_j \prec C_i} \mathcal{C}_j[\![C'^{t'_j}_j]\!])$, it is required that F_{C_i} has type $\prod C'_j \prec C_i t'_j \to t$.

– type soundness for semantic equational definitions:

- It follows, by structural induction, that $\mathcal{C}[\![C_i^t]\!] \in \mathcal{D}_t$,

– If the implementation is in a typed functional language (such as OCaml), this typing is done by the compiler. Otherwise, that may have to be done by hand.

Example of structural concrete semantics: forward collecting semantics of arithmetic expressions

– $X \in \mathbb{V}$, variables

$A ::= \mathtt{n} \mid \mathtt{X} \mid \mathtt{u}A' \mid A_1\, \mathtt{b}\, A_2$, $A \in \text{Aexp}$, arithmetic expressions

– $\text{Faexp} \in \text{Aexp} \mapsto \mathcal{D}_{\text{Aexp}}$

$\text{Faexp}[\![A]\!] \stackrel{\text{def}}{=} \{v \mid \exists \rho \in R : \rho \vdash A \Rrightarrow v\}$ [2]

– $\mathbb{I} \stackrel{\text{def}}{=} [\mathtt{min\_int}, \mathtt{max\_int}]$, machine integers

[2] where $\rho \vdash A \Rrightarrow v$, as defined by the operational semantics, holds whenever evaluation of A in environment ρ may return value v

- $\mathbb{E} \stackrel{\text{def}}{=} \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}$, errors
- $\mathbb{I}_\Omega \stackrel{\text{def}}{=} \mathbb{I} \cup \mathbb{E}$, machine values
- $\mathbb{R} \stackrel{\text{def}}{=} \mathbb{V} \mapsto \mathbb{I}_\Omega$, environments
- $\mathcal{D}_{\mathrm{Aexp}} \stackrel{\text{def}}{=} \wp(\mathbb{R}) \stackrel{\sqcup}{\longmapsto} \wp(\mathbb{I}_\Omega)$ (same for all $A \in \mathrm{Aexp}$), properties
- $\mathrm{Faexp}\llbracket \mathtt{n} \rrbracket = \mathcal{F}_{\mathrm{Aexp}}\llbracket \mathtt{n} \rrbracket()$ (a)

  $\mathcal{F}_{\mathrm{Aexp}}\llbracket \mathtt{n} \rrbracket() \stackrel{\text{def}}{=} \lambda R \cdot \{\underline{\mathtt{n}}\}$ constant of $\mathcal{D}_{\mathrm{Aexp}}$
- $\mathrm{Faexp}\llbracket \mathtt{X} \rrbracket = \mathcal{F}_{\mathrm{Aexp}}\llbracket \mathtt{X} \rrbracket()$ (b)

  $\mathcal{F}_{\mathrm{Aexp}}\llbracket \mathtt{X} \rrbracket() \stackrel{\text{def}}{=} \lambda R \cdot R(\mathtt{X}) = \lambda R \cdot \{\rho(\mathtt{X}) \mid \rho \in R\}$ constant of $\mathcal{D}_{\mathrm{Aexp}}$

- $\mathrm{Faexp}\llbracket ? \rrbracket = \mathcal{F}_{\mathrm{Aexp}}\llbracket ? \rrbracket()$ (c)

  $\mathcal{F}_{\mathrm{Aexp}}\llbracket ? \rrbracket() \stackrel{\text{def}}{=} \lambda R \cdot R(\mathtt{X}) = \mathbb{I}$ constant of $\mathcal{D}_{\mathrm{Aexp}}$
- $\mathrm{Faexp}\llbracket \mathrm{u}A \rrbracket = \mathcal{F}_{\mathrm{Aexp}}\llbracket \mathrm{u}A \rrbracket(\mathrm{Faexp}\llbracket A \rrbracket)$ (d)

  $\mathcal{F}_{\mathrm{Aexp}}\llbracket \mathrm{u}A \rrbracket(S) = f_{\mathrm{u}A}(S) = \underline{\mathrm{u}}^{\mathcal{C}} \circ S$

  where $f_{\mathrm{u}A} \in \mathcal{D}_{\mathrm{Aexp}} \mapsto \mathcal{D}_{\mathrm{Aexp}}$ and $\underline{\mathrm{u}}^{\mathcal{C}}(V) \stackrel{\text{def}}{=} \{\mathrm{u}(v) \mid v \in V\}$

  so that:

  $\mathrm{Faexp}\llbracket \mathrm{u}A \rrbracket = \lambda R \cdot \underline{\mathrm{u}}^{\mathcal{C}}(\mathrm{Faexp}\llbracket A \rrbracket R)$
- $\mathrm{Faexp}\llbracket A_1 \mathrm{b} A_2 \rrbracket = \mathcal{F}_{\mathrm{Aexp}}\llbracket A_1 \mathrm{b} A_2 \rrbracket(\mathrm{Faexp}\llbracket A_1 \rrbracket, \mathrm{Faexp}\llbracket A_2 \rrbracket)$ (e)

  $\mathcal{F}_{\mathrm{Aexp}}\llbracket A_1 \mathrm{~b~} A_2 \rrbracket(S_1, S_2) \stackrel{\text{def}}{=} \underline{\mathrm{b}}^{\mathcal{C}}(S_1, S_2)$

where

$\underline{\mathrm{b}}^{\mathcal{C}} \in \mathcal{D}_{\mathrm{Aexp}} \times \mathcal{D}_{\mathrm{Aexp}} \mapsto \mathcal{D}_{\mathrm{Aexp}}$

and

$\underline{\mathrm{b}}^{\mathcal{C}}(S_1, S_2) \stackrel{\text{def}}{=} \lambda R \cdot \{v_1 \,\underline{\mathrm{b}}\, v_2 \mid \exists \rho \in R : v_1 \in S_1(\{\rho\}) \wedge v_2 \in S_2(\{\rho\})\}$

so that

$\mathrm{Faexp}\llbracket A_1 \mathrm{~b~} A_2 \rrbracket \stackrel{\text{def}}{=} \lambda R \cdot \underline{\mathrm{b}}^{\mathcal{C}}(\mathrm{Faexp}\llbracket A_1 \rrbracket, \mathrm{Faexp}\llbracket A_2 \rrbracket) R$

Example of structural concrete semantics: boolean expressions

- $B ::= \mathtt{true} \mid \mathtt{false} \mid A_1 \mathrm{~c~} A_2 \mid B_1 \,\&\, B_2 \mid B_1 \mid B_2$, $B \in \mathrm{Bexp}$, boolean expressions
- $\mathrm{Cbexp}\llbracket B \rrbracket \in \mathrm{Bexp} \mapsto \mathcal{D}_{\mathrm{Bexp}}$

  $\mathrm{Cbexp}\llbracket B \rrbracket \stackrel{\text{def}}{=} \lambda R \cdot \{\rho \in R \mid \rho \vdash B \Mapsto \mathtt{tt}\}$ [3]
- $\mathcal{D}_{\mathrm{Bexp}} \stackrel{\text{def}}{=} \wp(\mathbb{R}) \stackrel{\sqcup}{\longmapsto} \wp(\mathbb{R})$
- $\mathrm{Cbexp}\llbracket \mathtt{true} \rrbracket \stackrel{\text{def}}{=} \lambda R \cdot R$ constant of $\mathcal{D}_{\mathrm{Bexp}}$
- $\mathrm{Cbexp}\llbracket \mathtt{false} \rrbracket \stackrel{\text{def}}{=} \lambda R \cdot \emptyset$ constant of $\mathcal{D}_{\mathrm{Bexp}}$

[3] where $\rho \vdash B \Mapsto \mathtt{tt}$ is defined by the operational semantics as holding in environment ρ when B is true and without runtime error.

– $\mathrm{Cbexp}[\![A_1 \mathrel{\mathtt{c}} A_2]\!] \stackrel{\text{def}}{=} \mathcal{F}_{\mathrm{Bexp}}[\![A_1 \mathrel{\mathtt{c}} A_2]\!](\mathrm{Faexp}[\![A_1]\!], \mathrm{Faexp}[\![A_2]\!])$

$\mathcal{F}_{\mathrm{Bexp}}[\![A_1 \mathrel{\mathtt{c}} A_2]\!](S_1, S_2) \stackrel{\text{def}}{=} \underline{\mathrm{c}}^{\mathcal{C}}\ (S_1, S_2)$

$\underline{\mathrm{c}}^{\mathcal{C}} \in \mathcal{D}_{\mathrm{Bexp}} \times \mathcal{D}_{\mathrm{Bexp}} \mapsto \mathcal{D}_{\mathrm{Bexp}}$

$\underline{\mathrm{c}}^{\mathcal{C}}\ (S_1, S_2) \stackrel{\text{def}}{=} \lambda R \cdot \{\rho \in R \mid \exists v_1 \in S_1(\{\rho\}) \cap \mathbb{I} :$
$\exists v_2 \in S_2(\{\rho\}) \cap \mathbb{I} : v_1 \mathrel{\underline{\mathrm{c}}} v_2 = \mathtt{tt}\}$

so that:

$\mathrm{Cbexp}[\![A_1 \mathrel{\mathtt{c}} A_2]\!] = \lambda R \cdot \underline{\mathrm{c}}^{\mathcal{C}}\ (\mathrm{Faexp}[\![A_1]\!], \mathrm{Faexp}[\![A_2]\!])R$

– $\mathrm{Cbexp}[\![B_1 \& B_2]\!] \stackrel{\text{def}}{=} \mathcal{F}_{\mathrm{Bexp}}[\![B_1 \& B_2]\!](\mathrm{Cbexp}[\![B_1]\!], \mathrm{Cbexp}[\![B_2]\!])$

$\mathcal{F}_{\mathrm{Bexp}}[\![B_1 \ \&\ B_2]\!](S_1, S_2) = f_{B_1 \& B_2}(S_1, S_2)$

$f_{B_1 \& B_2} \mathcal{D}_{\mathrm{Bexp}} \times \mathcal{D}_{\mathrm{Bexp}} \mapsto \mathcal{D}_{\mathrm{Bexp}}$

$f_{B_1 \& B_2}(S_1, S_2) \stackrel{\text{def}}{=} \lambda R \cdot S_1(R) \cap S_2(R)$

so that:

$\mathrm{Cbexp}[\![B_1 \ \&\ B_2]\!] \stackrel{\text{def}}{=} \lambda R \cdot \mathrm{Cbexp}[\![B_1]\!]R \cap \mathrm{Cbexp}[\![B_2]\!]R$

– $\mathrm{Cbexp}[\![B_1 | B_2]\!] \stackrel{\text{def}}{=} \mathcal{F}_{\mathrm{Bexp}}[\![B_1 | B_2]\!](\mathrm{Cbexp}[\![B_1]\!], \mathrm{Cbexp}[\![B_2]\!])$

$\mathcal{F}_{\mathrm{Bexp}}[\![B_1 \mid B_2]\!](S_1, S_2) = f_{B_1 | B_2}(S_1, S_2)$

$f_{B_1 | B_2} \mathcal{D}_{\mathrm{Bexp}} \times \mathcal{D}_{\mathrm{Bexp}} \mapsto \mathcal{D}_{\mathrm{Bexp}}$

$f_{B_1 | B_2}(S_1, S_2) \stackrel{\text{def}}{=} \lambda R \cdot S_1(R) \cap S_2(R)$

so that:

$\mathrm{Cbexp}[\![B_1 \mid B_2]\!] \stackrel{\text{def}}{=} \lambda R \cdot \mathrm{Cbexp}[\![B_1]\!]R \cup \mathrm{Cbexp}[\![B_2]\!]R$

Example of structural concrete semantics: commands,sequences and programs

– $C ::= \mathtt{skip} \mid X := \mathrm{A}$ $\quad C \in \mathrm{Com}$, commands
$\mid \mathtt{if}\ B\ \mathtt{then}\ S\ \mathtt{else}\ S\ \mathtt{fi}$
$\mid \mathtt{while}\ B\ \mathtt{do}\ S\ \mathtt{od}$
$S ::= C\ ;\ S \mid C$ $\quad S \in \mathrm{Seq}$, sequences of commands
$P ::= S\ ;;$ $\quad P \in \mathrm{Prog}$, programs

– For all $I \in \mathrm{Com} \cup \mathrm{Seq} \cup \mathrm{Prog}$, we have $\mathrm{Rcom}[\![I]\!] \in \mathcal{D}_{\mathrm{Com}}[\![I]\!]$ where

$$\mathcal{D}_{\mathrm{Com}}[\![I]\!] \stackrel{\text{def}}{=} \wp(\mathbb{R}) \stackrel{\sqcup}{\longmapsto} (\mathrm{in}_P[\![I]\!] \to \wp(\mathbb{R}))$$

– $\mathrm{Rcom}[\![I]\!]R\ell \stackrel{\text{def}}{=} \lambda R \cdot \lambda \ell \cdot \{\rho \mid \exists \rho' \in R : \langle\langle \mathrm{at}_P[\![C]\!],\ \rho'\rangle,\ \langle \ell,\ \rho\rangle\rangle \in \tau^\star[\![C]\!]\}$

reachable states (according to the operational semantics defined in lecture 5)

– $\mathrm{Rcom}[\![\mathtt{skip}]\!] = \lambda R \cdot \lambda \ell \cdot R$ $\quad$ (constant of $\mathcal{D}_{\mathrm{Com}}[\![\mathtt{skip}]\!]$)

– $\mathrm{Rcom}[\![\mathtt{X} := A]\!] = \mathcal{F}_{\mathrm{Com}}[\![\mathtt{X} := A]\!](\mathrm{Faexp}[\![A]\!])$

$\mathcal{F}_{\mathrm{Com}}[\![\mathtt{X} := A]\!](S) = f_{\mathtt{X} := A}(S)$

$f_{\mathtt{X} := A} \in \mathcal{D}_{\mathrm{Aexp}} \mapsto \mathcal{D}_{\mathrm{Com}}[\![\mathtt{X} := A]\!]$

$f_{\mathtt{X} := A}(S) \stackrel{\text{def}}{=} \lambda R \cdot \lambda \ell \cdot = \mathsf{match}\ \ell\ \mathsf{with}$
$\mid \mathrm{at}_P[\![\mathtt{X} := A]\!] \to R$
$\mid \mathrm{after}_P[\![\mathtt{X} := A]\!] \to \{\rho[\mathtt{X} := i] \mid \rho \in R \wedge i \in (S(\{\rho\})) \cap \mathbb{I}\}$

so that

$$\begin{array}{l}\text{Rcom}[\![\texttt{X} := A]\!]R\ell = \text{match } \ell \text{ with}\\ \quad \mid \text{at}_P[\![\texttt{X} := A]\!] \to R\\ \quad \mid \text{after}_P[\![\texttt{X} := A]\!] \to \{\rho[\texttt{X} := i] \mid \rho \in R \wedge i \in (\text{Faexp}[\![A]\!]\{\rho\}) \cap \mathbb{I}\}\end{array}$$

– $\text{Rcom}[\![C]\!]$ where $C = \texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}$

$$= \mathcal{F}_{\text{Com}}[\![C]\!](\text{Cbexp}[\![B]\!], \text{Cbexp}[\![T(\neg(B))]\!], \text{Rcom}[\![S_t]\!], \text{Rcom}[\![S_f]\!])$$

$$\mathcal{F}_{\text{Com}}[\![C]\!](B_1, B_2, S_1, S_2) \stackrel{\text{def}}{=} f_C(B_1, B_2, S_1, S_2)$$

where

$$f_C \in \mathcal{D}_{\text{Bexp}} \times \mathcal{D}_{\text{Bexp}} \times \mathcal{D}_{\text{Com}}[\![S_t]\!] \times \mathcal{D}_{\text{Com}}[\![S_f]\!] \mapsto \mathcal{D}_{\text{Com}}[\![C]\!]$$

$$\begin{array}{l}f_C(B_1, B_2, S_1, S_2) \stackrel{\text{def}}{=} \lambda R \cdot \lambda \ell \cdot \text{match } \ell \text{ with}\\ \quad \mid \text{at}_P[\![C]\!] \to R\\ \quad \mid \text{in}_P[\![S_t]\!] \to S_1(B_1(R))\ell\\ \quad \mid \text{in}_P[\![S_f]\!] \to S_2(B_2(R))\ell\\ \quad \mid \text{after}_P[\![C]\!] \to S_1(B_1(R))(\text{after}_P[\![S_t]\!]) \cup\\ \qquad\qquad\qquad S_2(B_2(R))(\text{after}_P[\![S_f]\!])\end{array}$$

so that:

$$\begin{array}{l}\text{Rcom}[\![C]\!]R\ell \text{ where } C = \texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi} =\\ \quad \text{match } \ell \text{ with}\\ \qquad \mid \text{at}_P[\![C]\!] \to R\\ \qquad \mid \text{in}_P[\![S_t]\!] \to \text{Rcom}[\![S_t]\!](\text{Cbexp}[\![B]\!]R)\ell\\ \qquad \mid \text{in}_P[\![S_f]\!] \to \text{Rcom}[\![S_f]\!](\text{Cbexp}[\![T(\neg(B))]\!]R)\ell\\ \qquad \mid \text{after}_P[\![C]\!] \to \text{Rcom}[\![S_t]\!](\text{Cbexp}[\![B]\!]R)(\text{after}_P[\![S_t]\!])\\ \qquad\qquad \cup \text{Rcom}[\![S_f]\!](\text{Cbexp}[\![T(\neg(B))]\!]R)(\text{after}_P[\![S_f]\!])\end{array}$$

– $\text{Rcom}[\![C]\!]$ where $C = \texttt{while } B \texttt{ do } S \texttt{ od}$

$$= \mathcal{F}_{\text{Com}}[\![C]\!](\text{Cbexp}[\![B]\!], \text{Cbexp}[\![T(\neg(B))]\!], \text{Rcom}[\![S]\!])$$

$$\mathcal{F}_{\text{Com}}[\![C]\!](B_1, B_2, S_1) \stackrel{\text{def}}{=} f_C(B_1, B_2, S_1, \text{lfp}_{\emptyset}^{\subseteq} \lambda X \cdot f_S(B_1, S_1, X))$$

where

$$f_S \in \mathcal{D}_{\text{Bexp}} \times \mathcal{D}_{\text{Com}}[\![S]\!] \times \mathcal{D}_{\text{Com}}[\![C]\!] \mapsto \mathcal{D}_{\text{Com}}[\![C]\!]$$

$$f_S(B_1, S_1, X) \stackrel{\text{def}}{=} \lambda R \cdot R \cup S_1(B_1(X))(\text{after}_P[\![S]\!])$$

$$f_C \in \mathcal{D}_{\text{Bexp}} \times \mathcal{D}_{\text{Com}}[\![S]\!] \times \mathcal{D}_{\text{Com}}[\![C]\!] \mapsto \mathcal{D}_{\text{Com}}[\![C]\!]$$

$$\begin{array}{l}f_C(B_1, B_2, S_1, F_1) \stackrel{\text{def}}{=} \lambda R \cdot \lambda \ell \cdot \text{match } \ell \text{ with}\\ \qquad\qquad \mid \text{at}_P[\![C]\!] \to I\\ \qquad\qquad \mid \text{in}_P[\![S]\!] \to S_1(B_1(F_1(R)))(\ell)\\ \qquad\qquad \mid \text{after}_P[\![C]\!] \to B_2(F_1(R))\end{array}$$

so that

$$\begin{array}{l}\text{Rcom}[\![C]\!]R\ell \text{ where } C = \texttt{while } B \texttt{ do } S \texttt{ od} =\\ \quad \text{let } I = \text{lfp}_{\emptyset}^{\subseteq} \lambda X \cdot R \cup \text{Rcom}[\![S]\!](\text{Cbexp}[\![B]\!]X)(\text{after}_P[\![S]\!]) \text{ in}\\ \qquad \text{match } \ell \text{ with}\\ \qquad \mid \text{at}_P[\![C]\!] \to I\\ \qquad \mid \text{in}_P[\![S]\!] \to \text{Rcom}[\![S]\!](\text{Cbexp}[\![B]\!]I)(\ell)\\ \qquad \mid \text{after}_P[\![C]\!] \to \text{Cbexp}[\![T(\neg(B))]\!]R)I\end{array}$$

– Note that to prove equivalence, we need the following result:

Theorem. Let $\langle L, \sqsubseteq, \bot, \sqcup\rangle$ be a cpo, $F \in E \mapsto (L \stackrel{\text{m}}{\longmapsto} L)$. Then $\forall R \in E$:

$$\mathsf{lfp}^{\sqsubseteq}_{\bot}\, \lambda X \cdot F(R, X) = \left(\mathsf{lfp}^{\dot\sqsubseteq}_{\dot\bot}\, \lambda Y \cdot \lambda R \cdot F(R, Y(R))\right)(R)$$

$\blacksquare$

Proof. – Let X^δ, $\delta \in \mathbb{O}$ be the iterates of $\mathsf{lfp}^{\sqsubseteq}_{\bot}\, \lambda X \cdot F(R, X)$ with rank ϵ

– Let Y^δ, $\delta \in \mathbb{O}$ be the iterates of $\mathsf{lfp}^{\dot\sqsubseteq}_{\dot\bot}\, \lambda Y \cdot \lambda R \cdot F(R, Y(R))$ with rank ϵ'

– We prove by transfinite induction that $X^\delta = Y^\delta(R)$.

- $X^0 = \bot = \dot\bot(R) = Y^0(R)$
- If $X^\delta = Y^\delta(R)$ then

$$\begin{array}{lll}
X^{\delta+1} & = F(R, X^\delta) & \wr\text{def. iterates}\wr \\
= & F(R, Y^\delta(R)) & \wr\text{ind. hyp.}\wr \\
= & \lambda R' \cdot F(R', Y^\delta(R'))(R) & \\
= & \lambda Y \cdot (\lambda R' \cdot F(R', Y(R'))(R))(Y^\delta) & \\
= & Y^{\delta+1}(R) & \wr\text{def. iterates}\wr
\end{array}$$

- It λ is a limit ordinal and $\forall \beta < \lambda : X^\beta = Y^\beta(R)$ then

$$\begin{array}{lll}
X^\lambda & = \bigsqcup\limits_{\beta<\lambda} X^\beta & \wr\text{def. iterates}\wr \\
= & \bigsqcup\limits_{\beta<\lambda} Y^\beta(R) & \wr\text{ind. hyp.}\wr \\
= & (\lambda R' \cdot \bigsqcup\limits_{\beta<\lambda} Y^\beta(R'))(R) &
\end{array}$$

$$\begin{array}{lll}
= & \dot{\bigsqcup\limits_{\beta<\lambda}} Y^\beta)(R) & \\
= & Y^\lambda(R) & \wr\text{def. iterates}\wr
\end{array}$$

- It follows that $\mathsf{lfp}^{\sqsubseteq}_{\bot}\, \lambda X \cdot F(R, X) = X^\epsilon = X^{\max(\epsilon,\epsilon')} = Y^{\max(\epsilon,\epsilon')}(R)$ $= Y^{\epsilon'}(R) = \left(\mathsf{lfp}^{\dot\sqsubseteq}_{\dot\bot}\, \lambda Y \cdot \lambda R \cdot F(R, Y(R))\right)(R)$

$\square$

– $\mathrm{Rcom}[\![C\ ;\ S]\!] = \mathcal{F}_{\mathrm{Com}}[\![C\ ;\ S]\!](\mathrm{Rcom}[\![C]\!], \mathrm{Rcom}[\![S]\!])$
$\mathcal{F}_{\mathrm{Com}}[\![C\ ;\ S]\!](C_1, S_1) \stackrel{\text{def}}{=} f_{C\,;\,S}(C_1, S_1)$
$f_{C\,;\,S} \in \mathcal{D}_{\mathrm{Com}}[\![C]\!] \times \mathcal{D}_{\mathrm{Com}}[\![S]\!] \mapsto \mathcal{D}_{\mathrm{Com}}[\![C\ ;\ S]\!]$

$f_{C\,;\,S}(C_1, S_1) \stackrel{\text{def}}{=}$
$\lambda R \cdot \lambda \ell \cdot \mathsf{match}\ \ell\ \mathsf{with}$
$\quad \mid \mathrm{in}_P[\![C]\!] \to C_1(R)\ell$
$\quad \mid \mathrm{in}_P[\![S]\!] \to S_1(C_1(R)(\mathrm{after}_P[\![C]\!]))\ell$
so that we get

$\mathrm{Rcom}[\![C\ ;\ S]\!]R\ell = \mathsf{match}\ \ell\ \mathsf{with}$
$\quad \mid \mathrm{in}_P[\![C]\!] \to \mathrm{Rcom}[\![C]\!]R\ell$
$\quad \mid \mathrm{in}_P[\![S]\!] \to \mathrm{Rcom}[\![S]\!](\mathrm{Rcom}[\![C]\!]R(\mathrm{after}_P[\![C]\!]))\ell$

– $\mathrm{Rcom}[\![S\ ;\!;]\!] = \mathcal{F}_{\mathrm{Com}}[\![S\ ;\!;]\!](\mathrm{Rcom}[\![S]\!])$
$\mathcal{F}_{\mathrm{Com}}[\![S\ ;\!;]\!](S_1) = S_1$
so that we get

$$\text{Rcom}[\![S\ ;;]\!]R\ell = \text{Rcom}[\![S]\!]R\ell$$

– This concludes the proof that the forward collecting semantics of a command (as introduced in lecture 16) is of the general form on which we reason afterwards.

Fixpoint existence

In the structural definition of the semantics, all elements are well-defined but, may be, the fixpoints.

Definition (fixpoint existence).

– We say that *the fixpoints exist* if and only if all fixpoints appearing in the structural definition exist.

– A fixpoint $\mathsf{lfp}_a^{\sqsubseteq} F$ (where $F \in L \mapsto L$, $\langle L, \sqsubseteq, \bot, \sqcup \rangle$ is a poset) *exists* whenever the transfinite iteration sequence $X^0 = a$, $X^{\delta+1} = F(X^\delta)$, $X^\lambda = \bigsqcup_{\beta<\lambda} X^\beta$ for limit ordinals is well-defined (i.e. the lub $\bigsqcup$ does exist), ultimately stationary at rank ϵ (so that $\forall \delta \geq \epsilon : X^\delta = X^\epsilon$ in which case we let $\mathsf{lfp}_a^{\sqsubseteq} F \stackrel{\text{def}}{=} X^\epsilon$.

■

Well-definedness of structural semantics

Theorem. If fixpoints exist then the structural semantic definition is well-defined. ■

Proof. By structural induction.

– For expressions $e[\![\mathcal{D}]\!][S](X) \in \mathcal{D}$, by cases:
  - Basis
    · $d \in \mathcal{D}$, by hypothesis
    · $S_j \in \mathcal{D}_j = \mathcal{D}$ by hypothesis
    · $X_k \in \mathcal{D}_k = \mathcal{D}$ by hypothesis
  - Induction step
    · For all $k = 1, \ldots, \ell$, $e_k[\![\mathcal{D}_{j_k}]\!][S](X) \in \mathcal{D}_{j_k}$ by induction hypothesis and $f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}} \in \left(\prod_{j=j_1}^{j_\ell} \mathcal{D}_j\right) \mapsto \mathcal{D}$ by hypothesis, proving that
    $f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}}(e_1[\![\mathcal{D}_{j_1}]\!][S](X)), \ldots, e_\ell[\![\mathcal{D}_{j_\ell}]\!][S](X))$

is well-defined and belongs to $\mathcal{D}$
    · By induction hypothesis $\lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \in \mathcal{D} \mapsto \mathcal{D}$, and fixpoints exist. By transfinite induction, all iterates belong to $\mathcal{D}$, whence for the fixpoint $\mathsf{lfp}_{\sqsubseteq}^{\bot}\ \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \in \mathcal{D}$

– For semantics $\mathcal{C}_i \in [C_i \in \text{Com}_i \mapsto \mathcal{D}_{C_i}]$, we proceed by structural induction
  - For the basis, C_i has no C'_j such that $C'_j \prec C_i$ whence $\mathcal{C}_i[\![C_i]\!] = \mathcal{F}_i[\![C_i]\!]() = e[\![\mathcal{D}_{C_i}]\!][S_1 : \mathcal{D}_{C'_1}, \ldots, S_n : \mathcal{D}_{C'_n}]()$ is well-defined and belongs to $\mathcal{D}_{C_i}$
  - For the induction step, $\mathcal{C}_j[\![C'_j]\!] \in \mathcal{D}_{C'_j}$ by induction hypothesis and so $\mathcal{C}_i[\![C_i]\!] = \mathcal{F}_i[\![C_i]\!]\left(\prod_{C'_j \prec C_i} \mathcal{C}_j[\![C'_j]\!]\right) = e[\![\mathcal{D}_{C_i}]\!][\mathcal{C}_1[\![C'_1]\!] : \mathcal{D}_{C'_1}, \ldots, \mathcal{C}_n[\![C'_n]\!] : \mathcal{D}_{C'_n}]()$ where $\{C' \mid C' \prec C_i\} = \{C'_1, \ldots, C'_n\}$ is well-defined and belongs to $\mathcal{D}_{C_i}$

□

Monotonic structural semantics

DEFINITION. The structural semantics is said to be *monotonic* whenever all $f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}} \in \left(\prod_{j=j_1}^{j_\ell} \mathcal{D}_j\right) \mapsto \mathcal{D}$ are monotonic on the poset $\langle \mathcal{D},\ \sqsubseteq\rangle$, that is: $\forall k = 1, \ldots, \ell : \forall X_{j_k}, X'_{j_k} \in \langle \mathcal{D}_{j_k},\ \sqsubseteq_{j_k}\rangle$:

$$
\begin{array}{l}
X_{j_k} \sqsubseteq_{j_k} X'_{j_k} \\
\Longrightarrow f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}}(\prod_{k=1}^{\ell} X_{j_k}) \sqsubseteq f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}}(\prod_{k=1}^{\ell} X'_{j_k})
\end{array}
$$

■

Well-definedness of monotonic structural semantics

THEOREM. In a monotonic structural semantics, all expressions are monotonic, whence the fixpoints exist on cpos, so the semantics is well-defined on cpos. ■

PROOF. – For expressions if $\langle \mathcal{D}_i,\ \sqsubseteq_i,\ \bot_i,\ \sqcup_i\rangle$ and $\langle \mathcal{D},\ \sqsubseteq,\ \bot,\ \sqcup\rangle$ are cpos then $\prod_{i=1}^{n} \mathcal{D}_i$ and $\prod_{i=n+1}^{m} \mathcal{D}_i$ are cpos for the componentwise orderings $\sqsubseteq_{1,n}$ and $\sqsubseteq_{n+1,m}$. Assume $S \sqsubseteq_{1,n} S'$, $X \sqsubseteq_{n+1,m} X'$. We prove by structural induction on expression e that $e[\![\mathcal{D}]\!][S](X) \sqsubseteq e[\![\mathcal{D}]\!][S'](X')$ and the expression is well-defined in $\mathcal{D}$

- Basis
  - $d \sqsubseteq d$ by reflexivity and $d \in \mathcal{D}$, by hypothesis
  - $S_j \sqsubseteq_j S'_j$ by def. componentwise ordering and $\sqsubseteq_j\ =\ \sqsubseteq$ with $S_j, S'_j \in \mathcal{D}_j = \mathcal{D}$ by hypothesis
  - $X_k \sqsubseteq_k X'_k$ by def. componentwise ordering and $\sqsubseteq_k\ =\ \sqsubseteq$ with $X_k \in \mathcal{D}_k = \mathcal{D}$ by hypothesis

- Induction step
  - For all $k = 1, \ldots, \ell$, $e_k[\![\mathcal{D}_{j_k}]\!][S](X) \in \mathcal{D}_{j_k}$ is well-defined and monotone by induction hypothesis and $f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}} \in \left(\prod_{j=j_1}^{j_\ell} \mathcal{D}_j\right) \stackrel{\text{m}}{\longmapsto} \mathcal{D}$ by hypothesis, proving that if $Y \sqsubseteq Y'$ then $e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \sqsubseteq e[\![\mathcal{D}]\!][S'](X', Y' : \mathcal{D})$ so that the function $\mathcal{F} \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \in \mathcal{D} \stackrel{\text{m}}{\longmapsto} \mathcal{D}$ is monotonic in its Y parameter. By the constructive version of Tarski's theorem, $\mathsf{lfp}_{\bot}^{\sqsubseteq} \mathcal{F}$ does exist on the cpo $\langle \mathcal{D},\ \sqsubseteq,\ \bot,\ \sqcup\rangle$ and is an element of $\mathcal{D}$.

    Moreover if we let

$$
\begin{array}{rl}
\mathcal{F} & \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) \\
\text{and } \mathcal{F}' & \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S'](X', Y : \mathcal{D})
\end{array}
$$

    then $\mathcal{F} \mathrel{\dot{\sqsubseteq}} \mathcal{F}'$ pointwise and to $\mathsf{lfp}_{\bot}^{\sqsubseteq} \mathcal{F} \sqsubseteq \mathsf{lfp}_{\bot}^{\sqsubseteq} \mathcal{F}'$, proving monotony.
- An immediate consequence is that the functions $\mathcal{F}_i[\![C_i]\!](S_1, \ldots, S_n) = e[\![\mathcal{D}_{C_i}]\!][S_1 : \mathcal{D}_{C'_1}, \ldots, S_n : \mathcal{D}_{C'_n}]()$ are monotonic in $S_1, \ldots, S_n$:

$$
\mathcal{F}_i[\![C_i]\!] \in \big(\prod_{C'_j \prec C_i} \mathcal{D}_{C'_j} \stackrel{\text{m}}{\longmapsto} \mathcal{D}_{C_i}\big)
$$

and well-defined
- Since fixpoints exist, the structural semantics is well-defined

□

Structural abstract semantics

The abstract semantics is in the same structural form as the collecting semantics. More precisely:

– Abstract domains: define the abstract information associated to each syntactic category Com_i, $i \in \Delta$. For each $i \in \Delta$ and $C_i \in \mathrm{Com}_i$:

  $\langle \overline{\mathcal{D}}_{C_i}, \overline{\sqsubseteq}_{C_i}, \overline{\bot}_{C_i}, \overline{\sqcup}_{C_i} \rangle$ is a poset (cpo, complete lattice, ...)

  (The nature of the correspondence between the abstract domains and the corresponding concrete ones will be considered later).

– Abstract semantics:

$$\overline{\mathcal{C}}_i \in [C_i \in \mathrm{Com}_i \mapsto \overline{\mathcal{D}}_{C_i}]$$

is defined, by structural induction, as

$$\overline{\mathcal{C}}_i[\![C_i]\!] \stackrel{\text{def}}{=} \overline{\mathcal{F}}_i[\![C_i]\!]\Big(\prod_{C'_j \prec C_i} \overline{\mathcal{C}}_j[\![C'_j]\!]\Big)$$

where

$$\overline{\mathcal{F}}_i[\![C_i]\!] \in \Big(\prod_{C'_j \prec C_i} \overline{\mathcal{D}}_{C'_j}\Big) \mapsto \overline{\mathcal{D}}_{C_i}$$

is the abstract transformer

The abstract transformer is defined in the form:

$$\overline{\mathcal{F}}_i[\![C_i]\!](S_1, \ldots, S_n) \stackrel{\text{def}}{=} \overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][\overline{S}_1 : \overline{\mathcal{D}}_{C'_1}, \ldots, \overline{S}_n : \overline{\mathcal{D}}_{C'_n}]()$$

where $\{C' \mid C' \prec C_i\} = \{C'_1, \ldots, C'_n\}$ and the right-hand side is an expression written according to the following attribute grammar, where we are given

- $\overline{S} = \overline{S}_1 : \overline{\mathcal{D}}_{C'_1}, \ldots, \overline{S}_n : \overline{\mathcal{D}}_{C'_n}$: the abstract semantics of components
- $\overline{X} = \overline{X}_{n+1} : \overline{\mathcal{D}}'_{n+1}, \ldots, \overline{X}_m : \overline{\mathcal{D}}'_m$: fixpoint variables
- $\langle \overline{\mathcal{D}}, \overline{\sqsubseteq}, \overline{\bot}, \overline{\sqcup} \rangle$: the abstract domain of the result

The attribute grammar of expressions is as follows:

$$\begin{aligned}
&\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}) ::= \\
&\quad \mid \overline{d} \\
&\quad \mid \overline{S}_j \\
&\quad \mid \overline{X}_k \\
&\quad \mid \overline{f}_{\overline{\mathcal{D}}_{j_1} \ldots \overline{\mathcal{D}}_{j_\ell} \overline{\mathcal{D}}}(e_1[\![\overline{\mathcal{D}}_{j_1}]\!][\overline{S}](\overline{X})), \ldots, e_\ell[\![\overline{\mathcal{D}}_{j_\ell}]\!][\overline{S}](\overline{X})) \\
&\quad \mid \mathbf{lfp}^{\overline{\sqsubseteq}}_{\overline{\bot}}\ \lambda \overline{Y} \cdot \overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})\ {}^{4}
\end{aligned}$$

where, by hypothesis:

- $\overline{d} \in \overline{\mathcal{D}}$ is a constant

[4] $\overline{Y} \notin \{\overline{X}_{n+1}, \ldots, \overline{X}_m\}$ must be a new fresh variable

- $\overline{S}_j$, $j \in [1, n]$ is the abstract semantics of an immediate component of C_i such that $\overline{\mathcal{D}}_j = \overline{\mathcal{D}}$
- $\overline{X}_k$, $k \in [n+1, m]$ appears inside a fixpoint definition and $\overline{\mathcal{D}}'_k = \overline{\mathcal{D}}$
- $\overline{f}_{\overline{\mathcal{D}}_{j_1} \ldots \overline{\mathcal{D}}_{j_\ell} \overline{\mathcal{D}}} \in \left(\prod_{j=j_1}^{j_\ell} \overline{\mathcal{D}}_j\right) \mapsto \overline{\mathcal{D}}$ is a constant function
- The fixpoints exist.

Local abstraction hypotheses on the correspondence between concrete and abstract semantics

– Observe that the concrete and abstract semantics have the same structural form (and so are stated to be *structurally identical*)

– So, intuitively, if the ingredients of the abstract semantics are upper-upproximations of their concrete counterparts, then the abstract semantics should be an upper-approximation of the concrete semantics

– This is made precise and proved in what follows

Local abstraction

DEFINITION. We say that the abstract domains

$$\langle \overline{\mathcal{D}}_{C_i}, \overline{\sqsubseteq}_{C_i}, \overline{\bot}_{C_i}, \overline{\sqcup}_{C_i} \rangle,\ i \in \Delta \text{ and } C_i \in \mathrm{Com}_i$$

are *local abstractions* of the concrete domains

$$\langle \mathcal{D}_{C_i}, \sqsubseteq_{C_i}, \bot_{C_i}, \sqcup_{C_i} \rangle,\ i \in \Delta \text{ and } C_i \in \mathrm{Com}_i$$

whenever there exists a concretization function γ_{C_i} which is monotone:

$$\gamma_{C_i} \in \overline{\mathcal{D}}_{C_i} \stackrel{\mathrm{m}}{\longmapsto} \mathcal{D}_{C_i} \qquad \text{(LA1)}$$

such that in the definitions of the corresponding structurally identical expressions $e[\![\mathcal{D}]\!][S](X)$ and $\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X})$, we have

– $d \sqsubseteq \overline{d}$ when $d \in \mathcal{D}_{C_i}$ and $\overline{d} \in \overline{\mathcal{D}}_{C_i}$ (LA2)

– If $\forall k = 1, \ldots, \ell$: $X_k \sqsubseteq_{j_k} \gamma_{j_k}(\overline{X}_k)$ then

$$f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}}(X_1, \ldots, X_\ell) \sqsubseteq \gamma(\overline{f}_{\overline{\mathcal{D}}_{j_1} \ldots \overline{\mathcal{D}}_{j_\ell} \overline{\mathcal{D}}}(\overline{X}_1, \ldots, \overline{X}_\ell)) \qquad \text{(LA3)}$$

■

– Remark 1: $\bot_{C_i} \sqsubseteq_{C_i} \gamma_{C_i}(\overline{\bot}_{C_i})$ by def. infimum (otherwise this should be assumed as an additional hypothesis)

– Remark 2: when lubs exist $\bigsqcup_{C_i}\{\gamma_{C_i}(X_i) \mid i \in \Delta\} \sqsubseteq_{C_i} \gamma_{C_i}(\overline{\bigsqcup}_{C_i}\{X_i \mid i \in \Delta\})$ by monotony of γ_{C_i}

– Remark 3: in case of a Galois connection based abstractions

$$\langle \mathcal{D}_{C_i}, \sqsubseteq_{C_i} \rangle \underset{\alpha_{C_i}}{\overset{\gamma_{C_i}}{\leftrightarrows}} \langle \overline{\mathcal{D}}_{C_i}, \overline{\sqsubseteq}_{C_i} \rangle$$

the usual soundness requirement that

$$\alpha \circ f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}}(\gamma_{j_1}(\overline{X}_1), \gamma_{j_\ell}(\ldots, \overline{X}_\ell)) \mathrel{\overline{\sqsubseteq}} \overline{f}_{\overline{\mathcal{D}}_{j_1} \ldots \overline{\mathcal{D}}_{j_\ell} \overline{\mathcal{D}}}(\overline{X}_1, \ldots, \overline{X}_\ell)$$

implies (LA3) when $f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}}$ is monotone

A soundness theorem on the correspondence between a concrete semantics and its local abstraction

– Given an abstract semantics which is a local abstraction of a structurally identical concrete semantics, we prove that this abstract semantics is a sound upper-approximation of the concrete semantics

– We proceed by structural induction on the considered programming language

A soundness of the correspondence between expressions

THEOREM. If e and $\overline{e}$ are structurally identical, (LA1), (LA2) and (LA3) hold, concrete and abstract fixpoints exist, and

– $S_j \sqsubseteq_j \gamma_i(\overline{S}_j)$, $j = 1, \ldots, n$ (a)

– $X_k \sqsubseteq_{ki} \gamma_i(\overline{X}_k)$, $k = n+1, \ldots, m$ (b)

then

$$e[\![\mathcal{D}]\!][S](X) \sqsubseteq \gamma(\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}))$$

■

PROOF. By structural induction on expressions.

– $d \sqsubseteq (\overline{d})$ by (LA2)

– $S_j \sqsubseteq_j \gamma_i(\overline{S}_j) = \gamma(\overline{S}_j)$ by (a) and $\mathcal{D} = \mathcal{D}_j$, $\overline{\mathcal{D}} = \overline{\mathcal{D}}_j$, $j = 1, \ldots, n$

– $X_{kj} \sqsubseteq_k \gamma_k(\overline{X}_k) = \gamma(\overline{X}_k)$ by (b) and $\mathcal{D} = \mathcal{D}_k$, $\overline{\mathcal{D}} = \overline{\mathcal{D}}_k$, $k = n+1, \ldots, m$

– By induction hypothesis, we have:

$$e_k[\![\mathcal{D}_{j_k}]\!][S](X) \ \sqsubseteq_{jk} \ \gamma_{jk}(\overline{e}_k[\![\overline{\mathcal{D}}_{j_k}]\!][S](X)), \quad k = 1, \ldots, \ell$$

and so by (LA3):

$$f_{\mathcal{D}_{j_1} \ldots \mathcal{D}_{j_\ell} \mathcal{D}}(\prod_{k=1}^{\ell} e_k[\![\mathcal{D}_{j_k}]\!][S](X)) \sqsubseteq \gamma(\overline{f}_{\overline{\mathcal{D}}_{j_1} \ldots \overline{\mathcal{D}}_{j_\ell} \overline{\mathcal{D}}}(\prod_{k=1}^{\ell} \overline{e}_k[\![\overline{\mathcal{D}}_{j_k}]\!][S](X)))$$

– let

$$\mathcal{F} \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D})$$
$$\text{and } \overline{\mathcal{F}} \stackrel{\text{def}}{=} \lambda Y \cdot \overline{e}[\![\mathcal{D}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})$$

If $Y \sqsubseteq \gamma(\overline{Y})$ then, by induction hypothesis on the identical structures of e and $\overline{e}$, we have,

$$\mathcal{F}(Y) \sqsubseteq \gamma(\overline{\mathcal{F}}(\overline{Y})) \qquad \text{(c)}$$

for all $Y \in \mathcal{D}$ and $\overline{Y} \in \overline{\mathcal{D}}$

– Let us now consider the iterates $\langle X^\delta,\ \delta \in \mathbb{O}\rangle$ of $\mathcal{F}$ and $\langle Y^\delta,\ \delta \in \mathbb{O}\rangle$ of $\overline{\mathcal{F}}$ which are respectively stationary at ϵ and ϵ', by fixpoint existence hypothesis
  - $X^0 \stackrel{\text{def}}{=} \bot \sqsubseteq \gamma(\overline{\bot}) \stackrel{\text{def}}{=} Y^0$
  - If $X^\delta \sqsubseteq \gamma(Y^\delta)$ by induction hypothesis, then $X^{\delta+1} = \mathcal{F}(X^\delta) \sqsubseteq \gamma(\overline{\mathcal{F}}(Y^\delta))$ by (c) proving that $X^{\delta+1} \sqsubseteq \gamma(Y^{\delta+1})$ since $Y^{\delta+1} = \overline{\mathcal{F}}(Y^\delta)$
  - If λ is a limit ordinal and $\forall \beta < \lambda : X^\beta \sqsubseteq \gamma(Y^\beta)$ then $X^\lambda = \bigsqcup_{\beta<\lambda} X^\beta \sqsubseteq \bigsqcup_{\beta<\lambda} \gamma(Y^\beta) \sqsubseteq \gamma(\overline{\bigsqcup_{\beta<\lambda}} Y^\beta) = \gamma(Y^\lambda)$ (which are well-defined by fixpoint existence)
  - It follows that $\mathsf{lfp}_\bot^\sqsubseteq \mathcal{F} = X^\epsilon = X^{\max(\epsilon,\epsilon')} \sqsubseteq \gamma(Y^{\max(\epsilon,\epsilon')}) = \gamma(Y^{\epsilon'}) = \gamma(\mathsf{lfp}_{\overline{\bot}}^{\overline{\sqsubseteq}} \overline{\mathcal{F}})$

□

Soundness of the correspondence between a concrete and abstract semantics

THEOREM. If (LA1), (LA2) and (LA3) do hold, concrete and abstract fixpoints exist, then for all $i \in \varDelta$ and $C_i \in \mathrm{Com}_i$, we have

$$\mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \gamma_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])$$

■

PROOF. By structural induction on the well-founded relation $\langle \bigcup_{i\in\varDelta} \mathrm{Com}_i,\ \prec\rangle$. Given any $i \in \varDelta$ and $C_i \in \mathrm{Com}_i$, assume by induction hypothesis that

$$\forall C'_j \prec C_i : \mathcal{C}[\![C'_j]\!] \sqsubseteq_{C'_j} \gamma_{C_i}(\overline{\mathcal{C}}[\![C'_j]\!])$$

then

$$\begin{array}{ll}
 & \mathcal{C}[\![C_i]\!] \\
= & \mathcal{F}[\![C_i]\!](\prod_{C'_j \prec C_i} \mathcal{C}[\![C'_j]\!]) \\
= & e[\![\mathcal{D}_{C_i}]\!][\prod_{C'_j \prec C_i} \mathcal{C}[\![C'_j]\!] : \mathcal{D}_{C'_j}]() \\
\sqsubseteq_{C_i} & \gamma_{C_i}(\overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][\prod_{C'_j \prec C_i} \overline{\mathcal{C}}[\![C'_j]\!] : \overline{\mathcal{D}}_{C'_j}]()) \\
= & \gamma_{C_i}(\overline{\mathcal{F}}[\![C_i]\!](\prod_{C'_j \prec C_i} \overline{\mathcal{C}}[\![C'_j]\!])) \\
= & \gamma_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])
\end{array}$$

□

An abstract formalization of infinitary structural analysis by abstract interpretation

Hypotheses on widenings

Given a poset $\langle L, \sqsubseteq \rangle$, a widening operator on L is $\nabla \in L \times L \mapsto L$ satisfying

(W1) $y \sqsubseteq x \nabla y$

(W2) For all sequences $x^0, x^1, \ldots$ in L^ω, the sequence defined by

$$\begin{array}{rcll} y^0 & \stackrel{\text{def}}{=} & x^0 & \\ y^{n+1} & \stackrel{\text{def}}{=} & y^\ell & \text{if } \exists \ell \leq n : x^\ell \sqsubseteq y^\ell \\ & \stackrel{\text{def}}{=} & y^n \nabla x^n & \text{otherwise} \end{array}$$

is not strictly increasing.

THEOREM. The sequence $\langle y^k,\ k \in \mathbb{N} \rangle$ is strictly increasing up to a least $\ell \in \mathbb{N}$ such that $x^\ell \sqsubseteq y^\ell$ and the sequence is stationary at ℓ onwards. ■

PROOF. The sequence $\langle y^k,\ k \in \mathbb{N} \rangle$ cannot by strictly increasing by (W2). So there is a least ℓ such that $y^\ell \not\sqsubset y^{\ell+1}$. We cannot have $y^{\ell+1} = x^\ell \nabla y^\ell$ since by (W1), thus would imply that $y^\ell \sqsubseteq x^\ell \nabla y^\ell = y^{\ell+1}$. Hence, by definition of the sequence $\langle y^k,\ k \in \mathbb{N} \rangle$, we must have $y^{\ell+1} \stackrel{\text{def}}{=} y^k$ where $k \leq \ell$ and $x^k \sqsubseteq y^k$. We cannot have $k < \ell$ since for the smallest such k we would have $x^k \sqsubseteq y^k$ whence $y^{k+1} = y^k$ whence, by reflexivity, $y^k \sqsubseteq y^{k+1}$ in contradiction with the hypothesis that ℓ is the smallest natural with that property. It follows that $k = \ell$ and so by (W2): $y^{\ell+1} = y^\ell$ and $x^\ell \sqsubseteq y^\ell$. For all $n \geq \ell$, we have $\exists \ell \leq n : x^\ell \sqsubseteq y^\ell$ whence $y^{n+1} \stackrel{\text{def}}{=} y^\ell$ proving that the sequence is stationary at ℓ. □

Note: $\langle x^k,\ k \in \mathbb{N} \rangle$ not assumed to be increasing.

Structural abstract semantics with widening

The abstract semantics with widening is in the same structural form as the collecting semantics. More precisely:

– Abstract domains: For each $i \in \Delta$ and $C_i \in \mathrm{Com}_i$:

  $\langle \overline{\mathcal{D}}_{C_i}, \overline{\sqsubseteq}_{C_i}, \overline{\bot}_{C_i}, \overline{\sqcup}_{C_i} \rangle$ is a poset

– Widenings: For each $i \in \Delta$ and $C_i \in \mathrm{Com}_i$:

  $\nabla_{C_i} \in \overline{\mathcal{D}}_{C_i} \times \overline{\mathcal{D}}_{C_i} \mapsto \overline{\mathcal{D}}_{C_i}$ is a widening satisfying (W1) and (W2)

– Abstract semantics with widening:

$$\overline{\mathcal{C}}_i \in [C_i \in \mathrm{Com}_i \mapsto \overline{\mathcal{D}}_{C_i}]$$

is defined, by structural induction, as

$$\overline{\mathcal{C}}_i[\![C_i]\!] \stackrel{\text{def}}{=} \overline{\mathcal{F}}_i[\![C_i]\!]\Big(\prod_{C'_j \prec C_i} \overline{\mathcal{C}}_j[\![C'_j]\!]\Big)$$

where

$$\overline{\mathcal{F}}_i[\![C_i]\!] \in \Big(\prod_{C'_j \prec C_i} \overline{\mathcal{D}}_{C'_j}\Big) \mapsto \overline{\mathcal{D}}_{C_i}$$

is the abstract transformer

The abstract transformer is defined in the form:

$$\overline{\mathcal{F}}_i[\![C_i]\!](\overline{S}_1,\ldots,\overline{S}_n) \stackrel{\text{def}}{=} \overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][\overline{S}_1 : \overline{\mathcal{D}}_{C'_1},\ldots,\overline{S}_n : \overline{\mathcal{D}}_{C'_n}]()$$

where $\{C' \mid C' \prec C_i\} = \{C'_1,\ldots,C'_n\}$ and the right-hand side is an expression written according to the following attribute grammar, where we are given

- $\overline{S} = \overline{S}_1 : \overline{\mathcal{D}}_{C'_1},\ldots,\overline{S}_n : \overline{\mathcal{D}}_{C'_n}$: the abstract semantics of components
- $\overline{X} = \overline{X}_{n+1} : \overline{\mathcal{D}}'_{n+1},\ldots,\overline{X}_m : \overline{\mathcal{D}}'_m$: fixpoint variables
- $\langle\overline{\mathcal{D}},\ \overline{\sqsubseteq},\ \overline{\bot},\ \overline{\sqcup}\rangle$: the abstract domain of the result

The attribute grammar of expressions is as follows:

$$
\begin{array}{l}
\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}) ::= \\
\quad \mid \overline{d} \\
\quad \mid \overline{S}_j \\
\quad \mid \overline{X}_k \\
\quad \mid \overline{f}_{\overline{\mathcal{D}}_{j_1}\ldots\overline{\mathcal{D}}_{j_\ell}\overline{\mathcal{D}}}(e_1[\![\overline{\mathcal{D}}_{j_1}]\!][\overline{S}](\overline{X})),\ldots,e_\ell[\![\overline{\mathcal{D}}_{j_\ell}]\!][\overline{S}](\overline{X})) \\
\quad \mid \text{let} \\
\qquad\quad \overline{\mathcal{F}} = \lambda\overline{Y}\cdot\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X},\overline{Y}:\overline{\mathcal{D}}) \text{ and} \\
\qquad\quad \overline{\mathcal{G}} = \lambda X\cdot \text{let } Y = \overline{\mathcal{F}}(X) \text{ in } (Y \mathrel{\overline{\sqsubseteq}} X \mathrel{?} X \mathrel{¿} X \mathrel{\overline{\nabla}} Y) \\
\quad\ \ \text{in} \\
\qquad\quad \mathsf{lfp}_{\overline{\bot}}^{\overline{\sqsubseteq}}\ \overline{\mathcal{G}}
\end{array}
$$

where, by hypothesis:

- $\overline{d} \in \overline{\mathcal{D}}$ is a constant
- $\overline{S}_j$, $j \in [1,n]$ is the abstract semantics of an immediate component of C_i such that $\overline{\mathcal{D}}_j = \overline{\mathcal{D}}$
- $\overline{X}_k$, $k \in [n{+}1,m]$ appears inside a fixpoint definition and $\overline{\mathcal{D}}'_k = \overline{\mathcal{D}}$
- $\overline{f}_{\overline{\mathcal{D}}_{j_1}\ldots\overline{\mathcal{D}}_{j_\ell}\overline{\mathcal{D}}} \in \left(\prod_{j=j_1}^{j_\ell}\overline{\mathcal{D}}_j\right) \mapsto \overline{\mathcal{D}}$ is a constant function
- $\overline{\nabla} \in \overline{\mathcal{D}} \times \overline{\mathcal{D}} \mapsto \overline{\mathcal{D}}$ is a widening
- $\mathsf{lfp}_{\bot}^{\sqsubseteq}\ F$ is a shorthand for the limit X^ϵ of the transfinite iteration sequence $X^0 = \bot$, $X^{\delta+1} = F(X^\delta)$ and $X^\lambda = \bigsqcup_{\beta<\lambda} X^\beta$, λ limit ordinal, $\forall\delta \geq \epsilon : X^\delta = X^\epsilon$, whenever it exists.

Well-definedness of the structural abstract semantics with widening

THEOREM. *Any structural abstract semantics with widening satisfying (W1) and (W2) is well-defined.* ■

Proof. By structural induction on the inductive definition of the abstract semantics.

– For expressions $\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}) \in \overline{\mathcal{D}}$, by cases:
  - Basis
    · $\overline{d} \in \overline{\mathcal{D}}$, by hypothesis
    · $\overline{S}_j \in \overline{\mathcal{D}}_j = \overline{\mathcal{D}}$ by hypothesis
    · $\overline{X}_k \in \overline{\mathcal{D}}_k = \overline{\mathcal{D}}$ by hypothesis
  - Induction step
    · For all $k = 1, \ldots, \ell$, $\overline{e}_k[\![\overline{\mathcal{D}}_{j_k}]\!][\overline{S}](\overline{X}) \in \overline{\mathcal{D}}_{j_k}$ by induction hypothesis and $\overline{f}_{\overline{\mathcal{D}}_{j_1}\ldots\overline{\mathcal{D}}_{j_\ell}\overline{\mathcal{D}}} \in \left(\prod_{j=j_1}^{j_\ell} \overline{\mathcal{D}}_j\right) \mapsto \overline{\mathcal{D}}$ by hypothesis, proving that
    $\overline{f}_{\overline{\mathcal{D}}_{j_1}\ldots\overline{\mathcal{D}}_{j_\ell}\overline{\mathcal{D}}}(\overline{e}_1[\![\overline{\mathcal{D}}_{j_1}]\!][\overline{S}](\overline{X})), \ldots, \overline{e}_\ell[\![\overline{\mathcal{D}}_{j_\ell}]\!][\overline{S}](\overline{X}))$

is well-defined and belongs to $\overline{\mathcal{D}}$

· If $Y \in \overline{\mathcal{D}}$ then, by induction hypothesis, $\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})$ is well-defined and belongs to $\overline{\mathcal{D}}$, so that the locally defined function $\overline{\mathcal{F}} = \lambda\overline{Y} \cdot \overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})$ is well-defined and belongs that $\overline{\mathcal{D}} \mapsto \overline{\mathcal{D}}$. Since, by hypothesis, $\overline{\nabla} \in \overline{\mathcal{D}} \times \overline{\mathcal{D}} \mapsto \overline{\mathcal{D}}$, it follows that for all $X \in \overline{\mathcal{D}}$, $\overline{\mathcal{G}}(X)$ is well-defined and belongs to $\overline{\mathcal{D}} \mapsto \overline{\mathcal{D}}$. Let us now consider the iterates $\langle Z^n,\ \delta \in \mathbb{N}\rangle$ of $\overline{\mathcal{G}}$ starting from $\overline{\bot} \in \overline{\mathcal{D}}$. They are defined as:

$$
\begin{aligned}
Z^0 &\stackrel{\text{def}}{=} \overline{\bot} \\
Z^{n+1} &\stackrel{\text{def}}{=} Z^n && \text{if } \overline{\mathcal{F}}(Z^n) \mathrel{\overline{\sqsubseteq}} Z^n \\
&\stackrel{\text{def}}{=} Z^n \mathbin{\overline{\nabla}} \overline{\mathcal{F}}(Z^n) && \text{otherwise}
\end{aligned}
$$

Let $\ell \in \mathbb{N}$ be the smallest n, if any, such that $\overline{\mathcal{F}}(Z^\ell) \mathrel{\overline{\sqsubseteq}} Z^\ell$ then by recurrence, $\forall k \geq \ell : \overline{\mathcal{F}}(Z^k) \mathrel{\overline{\sqsubseteq}} Z^k$ and so the above $\langle Z^n,\ n \in \mathbb{N}\rangle$ can be defined in the equivalent form

$$
\begin{aligned}
Z^0 &\stackrel{\text{def}}{=} \overline{\bot} \\
Z^{n+1} &\stackrel{\text{def}}{=} y^\ell && \text{if } \exists \ell \leq n : \overline{\mathcal{F}}(Z^\ell) \mathrel{\overline{\sqsubseteq}} Z^\ell \\
&\stackrel{\text{def}}{=} Z^\ell \mathbin{\overline{\nabla}} \overline{\mathcal{F}}(Z^\ell) && \text{otherwise}
\end{aligned}
$$

By the theorem on widening (on page 62), The sequence $\langle Z^n,\ n \in \mathbb{N}\rangle$ is strictly increasing up to a least $\ell \in \mathbb{N}$ such that $\overline{\mathcal{F}}(Z^\ell) \mathrel{\overline{\sqsubseteq}} Z^\ell$ and the sequence is stationary at ℓ onwards. By definition of lubs, the transfinite extension of the sequence is well-defined and stationary at ℓ. By transfinite induction, all iterates belong to $\mathcal{D}$, whence for the fixpoint $\mathsf{lfp}_{\sqsubseteq}^{\bot}\ \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D}) = Z^\ell \in \mathcal{D}$

– For the abstract semantics $\overline{C}_i \in [C_i \in \mathrm{Com}_i \mapsto \overline{\mathcal{D}}_{C_i}]$, we proceed by structural induction
  - For the basis, C_i has no C'_j such that $C'_j \prec C_i$ whence $\overline{C}_i[\![C_i]\!] = \overline{\mathcal{F}}_i[\![C_i]\!]()$ $= \overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][S_1 : \overline{\mathcal{D}}_{C'_1}, \ldots, S_n : \overline{\mathcal{D}}_{C'_n}]()$ is well-defined and belongs to $\overline{\mathcal{D}}_{C_i}$

  - For the induction step, $\overline{C}_j[\![C'_j]\!] \in \overline{\mathcal{D}}_{C'_j}$ by induction hypothesis and so $\overline{C}_i[\![C_i]\!] = \overline{\mathcal{F}}_i[\![C_i]\!]\left(\prod_{C'_j \prec C_i} \overline{C}_j[\![C'_j]\!]\right) = \overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][\overline{C}_1[\![C'_1]\!] : \overline{\mathcal{D}}_{C'_1}, \ldots, \overline{C}_n[\![C'_n]\!] : \overline{\mathcal{D}}_{C'_n}]()$ where $\{C' \mid C' \prec C_i\} = \{C'_1, \ldots, C'_n\}$ is well-defined and belongs to $\overline{\mathcal{D}}_{C_i}$

$\square$

A soundness theorem on the correspondence between concrete semantics and its local abstraction by a structural abstract semantics with widening

Theorem. If e and $\overline{e}$ are structurally identical, (LA1), (LA2) and (LA3) hold, concrete fixpoints exist, (W1) and (W2) hold, and

– $S_j \sqsubseteq_j \gamma_i(\overline{S}_j)$, $j = 1, \ldots, n$ (a)

– $X_k \sqsubseteq_{ki} \gamma_i(\overline{X}_k)$, $k = n+1, \ldots, m$ (b)

then

$$e[\![\mathcal{D}]\!][S](X) \sqsubseteq \gamma(\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}))$$

■

Proof. By structural induction on expressions.

– $d \sqsubseteq (\overline{d})$ by (LA2)

– $S_j \sqsubseteq_j \gamma_i(\overline{S}_j) = \gamma(\overline{S}_j)$ by (a) and $\mathcal{D} = \mathcal{D}_j$, $\overline{\mathcal{D}} = \overline{\mathcal{D}}_j$, $j = 1, \ldots, n$

– $X_{kj} \sqsubseteq_k \gamma_k(\overline{X}_k) = \gamma(\overline{X}_k)$ by (b) and $\mathcal{D} = \mathcal{D}_k$, $\overline{\mathcal{D}} = \overline{\mathcal{D}}_k$, $k = n+1, \ldots, m$

– By induction hypothesis, we have:

$$e_k[\![\mathcal{D}_{j_k}]\!][S](X) \ \sqsubseteq_{jk} \ \gamma_{jk}(\overline{e}_k[\![\overline{\mathcal{D}}_{j_k}]\!][S](X)), \quad k = 1, \ldots, \ell$$

and so by (LA3):

$$f_{\mathcal{D}_{j_1}\ldots\mathcal{D}_{j_\ell}\mathcal{D}}(\prod_{k=1}^{\ell} e_k[\![\mathcal{D}_{j_k}]\!][S](X)) \mathrel{\overline{\sqsubseteq}} \gamma(\overline{f}_{\overline{\mathcal{D}}_{j_1}\ldots\overline{\mathcal{D}}_{j_\ell}\overline{\mathcal{D}}}(\prod_{k=1}^{\ell} \overline{e}_k[\![\overline{\mathcal{D}}_{j_k}]\!][S](X)))$$

– In the case of a fixpoint definition with widening, we let

$$\mathcal{F} \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D})$$

If $Y \sqsubseteq \gamma(\overline{Y})$ then, by induction hypothesis on the identical structures of e and $\overline{e}$, we have

$$\mathcal{F}(Y) \sqsubseteq \gamma(\overline{\mathcal{F}}(\overline{Y})) \qquad \text{(a)}$$

for all $Y \in \mathcal{D}$ and $\overline{Y} \in \overline{\mathcal{D}}$

– Since the concrete fixpoint $\mathsf{lfp}_{\perp}^{\sqsubseteq} \mathcal{F}$ is well-defined, the corresponding iterates $\langle X^\delta, \delta \in \mathbb{O} \rangle$ of $\mathcal{F}$ are stationary at rank $\epsilon \in \mathbb{O}$

– We have seen in the well-defined theorem proof that the iterates for $\mathsf{lfp}_{\overline{\perp}}^{\overline{\sqsubseteq}} \mathcal{G}$ are defined as:

$$\begin{array}{rcll} Z^0 & \stackrel{\text{def}}{=} & \overline{\perp} & \\ Z^{n+1} & \stackrel{\text{def}}{=} & Z^n & \text{if } \overline{\mathcal{F}}(Z^n) \mathrel{\overline{\sqsubseteq}} Z^n \quad \text{(b)} \\ & \stackrel{\text{def}}{=} & Z^n \mathbin{\overline{\nabla}} \overline{\mathcal{F}}(Z^n) & \text{otherwise} \quad \text{(c)} \end{array}$$

and proved using (W1), (W2) that they are ultimately stationery at rank $\epsilon' < \omega$ and that $\forall \delta \geq \epsilon' : Z^\delta = Z^\epsilon = \mathsf{lfp}_{\overline{\perp}}^{\overline{\sqsubseteq}} \mathcal{G}$. We have:

- $X^0 \stackrel{\text{def}}{=} \perp \sqsubseteq \gamma(\overline{\perp}) \stackrel{\text{def}}{=} Z^0$

- Assume that $X^\delta \sqsubseteq \gamma(Z^\delta)$ by induction hypothesis.
  - In case (b), we have

$$
\begin{aligned}
& \overline{\mathcal{F}}(Z^\delta) \,\overline{\sqsubseteq}\, Z^\delta && \wr\text{by (b)}\wr \\
\Longrightarrow\ & \gamma(\overline{\mathcal{F}}(Z^\delta)) \sqsubseteq \gamma(Z^\delta) && \wr\gamma\text{ monotone}\wr \\
\Longrightarrow\ & \mathcal{F}(X^\delta) \sqsubseteq \gamma(Z^\delta) && \wr\text{by (a)}\wr \\
\Longrightarrow\ & X^{\delta+1} \sqsubseteq \gamma(Z^\delta) && \wr\text{by def. iterates}\wr \\
\Longrightarrow\ & X^{\delta+1} \sqsubseteq \gamma(Z^{\delta+1}) && \wr\text{by (b)}\wr
\end{aligned}
$$

  - In case (c), we have:

$$
\begin{aligned}
& Z^{\delta+1} = Z^\delta \,\overline{\nabla}\, \overline{\mathcal{F}}(Z^\delta) && \wr\text{by (c)}\wr \\
\Longrightarrow\ & \overline{\mathcal{F}}(Z^\delta) \,\overline{\sqsubseteq}\, Z^{\delta+1} && \wr\text{by (W1)}\wr \\
\Longrightarrow\ & \mathcal{F}(X^\delta) \sqsubseteq \gamma(Z^\delta) && \wr\gamma\text{ is monotone}\wr \\
\Longrightarrow\ & X^{\delta+1} \sqsubseteq \gamma(Z^\delta) && \wr\text{by induction hypothesis, (a) and transitivity}\wr
\end{aligned}
$$

$$
\Longrightarrow\ X^{\delta+1} \sqsubseteq \gamma(Z^{\delta+1}) \qquad \wr\text{by def. iterates}\wr
$$

.

.

- If λ is a limit ordinal and $\forall \beta < \lambda : X^\beta \sqsubseteq \gamma(Z^\beta)$ then $X^\lambda = \bigsqcup_{\beta<\lambda} X^\beta \sqsubseteq \bigsqcup_{\beta<\lambda} \gamma(Z^\beta) = \gamma(\overline{\bigsqcup_{\beta<\lambda}} Z^\beta) = \gamma(Z^\lambda)$ since $\langle Z^\delta,\ \delta \in \mathbb{O}\rangle$ is stationary at rank $\epsilon' < \omega \leq \lambda$
- By transfinite induction, it follows that $\mathsf{lfp}_\bot^\sqsubseteq \mathcal{F} = X^\epsilon = X^{\max(\epsilon,\epsilon')} \sqsubseteq \gamma(Z^{\max(\epsilon,\epsilon')}) = \gamma(Z^{\epsilon'}) = \gamma(\mathsf{lfp}_{\overline{\bot}}^{\overline{\sqsubseteq}} \overline{\mathcal{G}})$

□

THEOREM. If (LA1), (LA2) and (LA3) do hold, concrete fixpoints exist, (W1) and (W2) hold, then for all $i \in \Delta$ and $C_i \in \mathrm{Com}_i$, we have

$$
\mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \gamma_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])
$$

■

PROOF. By structural induction on the well-founded relation $\langle \bigcup_{i\in\Delta} \mathrm{Com}_i,\ \prec\rangle$. Given any $i \in \Delta$ and $C_i \in \mathrm{Com}_i$, assume by induction hypothesis that

$$
\forall C'_j \prec C_i : \mathcal{C}[\![C'_j]\!] \sqsubseteq_{C'_j} \gamma_{C_i}(\overline{\mathcal{C}}[\![C'_j]\!])
$$

then

$$
\mathcal{C}[\![C_i]\!]
$$

$$
\begin{aligned}
= \ & \mathcal{F}[\![C_i]\!](\prod_{C'_j\prec C_i} \mathcal{C}[\![C'_j]\!]) \\
= \ & e[\![\mathcal{D}_{C_i}]\!][\prod_{C'_j\prec C_i} \mathcal{C}[\![C'_j]\!] : \mathcal{D}_{C'_j}]() \\
\sqsubseteq_{C_i} \ & \gamma_{C_i}(\overline{e}[\![\overline{\mathcal{D}}_{C_i}]\!][\prod_{C'_j\prec C_i} \overline{\mathcal{C}}[\![C'_j]\!] : \overline{\mathcal{D}}_{C'_j}]()) \\
= \ & \gamma_{C_i}(\overline{\mathcal{F}}[\![C_i]\!](\prod_{C'_j\prec C_i} \overline{\mathcal{C}}[\![C'_j]\!])) \\
= \ & \gamma_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])
\end{aligned}
$$

□

Hypotheses on narrowings

Given a poset $\langle L, \sqsubseteq \rangle$, a narrowing operator on L is $\Delta \in L \times L \mapsto L$ satisfying

(N1) $\forall y \sqsubset x : y \sqsubseteq x \,\Delta\, y \sqsubseteq x$

(N2) For all sequences $x^0, x^1, \ldots$ in L^ω, the sequence defined by

$$\begin{aligned} y^0 &\stackrel{\text{def}}{=} x^0 & \\ y^{n+1} &\stackrel{\text{def}}{=} y^n \,\Delta\, x^n & \text{if } x^n \sqsubset y^n \\ &\stackrel{\text{def}}{=} y^n & \text{otherwise} \end{aligned}$$

is not strictly increasing (although it is decreasing by (N1)).

Structural abstract semantics with widening/narrowing

– The structural definition is essential the same as the abstract semantics with widening (page 63), except for the use of a narrowing operator

– Narrowing: For each $i \in \Delta$ and $C_i \in \text{Com}_i$:

$\Delta_{C_i} \in \overline{\mathcal{D}}_{C_i} \times \overline{\mathcal{D}}_{C_i} \mapsto \overline{\mathcal{D}}_{C_i}$ is a narrowing satisfying (N1) and (N2)

– For fixpoints in the attribute grammar of expressions, we now have:

$\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}) ::= \ldots$
 | let
  $\overline{\mathcal{F}} = \lambda \overline{Y} \cdot \overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})$ and
  $\overline{\mathcal{G}} = \lambda X \cdot \text{let } Y = \overline{\mathcal{F}}(X) \text{ in } (\!| Y \sqsubseteq X \mathrel{?} X \mathrel{¿} X \,\nabla\, Y |\!)$ and
  $\overline{A} = \mathsf{lfp}^{\sqsubseteq}_{\perp} \overline{\mathcal{G}}$ and
  $\overline{\mathcal{H}} = \lambda X \cdot \text{let } Y = \overline{\mathcal{F}}(X) \text{ in } (\!| Y \sqsubset X \mathrel{?} X \,\Delta\, Y \mathrel{¿} X |\!)$
 in
  $\mathsf{gfp}^{\sqsubseteq}_{\overline{A}} \overline{\mathcal{H}}$ [5]

[5] where $\mathsf{gfp}^{\sqsubseteq}_{\top}$ is (partially) defined as the limit X^ϵ of the transfinite iteration sequence $X^0 = \top$, $X^{\delta+1} = F(X^\delta)$ and $X^\lambda = \prod_{\beta<\lambda} X^\beta$ when λ is a limit ordinal in case this sequence is well-defined and ultimately stationary at rank ϵ.

Well-definedness of the structural abstract semantics with widening/narrowing

THEOREM. A structural definition with widenings and narrowings respectively satisfying hypotheses (W1), (W2) and (N1), (N2) is well-defined. ∎

PROOF. – The proof, by structural induction, is essentially the same as in the previous case of "structural abstract semantics with widening", but for the case of fixpoints

– For fixpoints, we have already shown in this proof that $\overline{A} = \mathsf{lfp}^{\sqsubseteq}_{\perp} \overline{\mathcal{G}}$ is well-defined as the limit of an increasing chain stabilizing, in a finite number of steps, at a postfixpoint: $\overline{\mathcal{F}}(\overline{A}) \sqsubseteq \overline{A}$.

– If follows that the iterates $\langle X^\delta,\ \delta \in \mathbb{O}\rangle$ of $\mathsf{gfp}^{\overline{\sqsubseteq}}_{\overline{A}}\ \overline{\mathcal{H}}$ are of the following form:

(a) $X^0 = \overline{A}$, where $\overline{\mathcal{F}}(\overline{A}) \mathrel{\overline{\sqsubseteq}} \overline{A}$

(b) $X^{\delta+1} = X^\delta \mathbin{\overline{\Delta}} \overline{\mathcal{F}}(X^\delta)$, if $\overline{\mathcal{F}}(X^\delta) \mathrel{\overline{\sqsubset}} X^\delta$

(c) $X^{\delta+1} = X^\delta$, otherwise

(d) $X^\lambda = \overline{\bigsqcap}_{\beta<\lambda} X^\beta$ when λ is a limit ordinal

– Observe that by def. of $\overline{\mathcal{F}} = \lambda \overline{Y} \cdot \overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}, \overline{Y} : \overline{\mathcal{D}})$, $\overline{\mathcal{F}}$ is well-defined by induction hypothesis

– By its def., the sequence $\langle X^\delta,\ \delta < \omega\rangle$ is a decreasing chain, which is obvious in cases (c) and follow from (N1) in case (b)

– By (N2), the decreasing chain $\langle X^\delta,\ \delta < \omega\rangle$ is not strictly decreasing so its is ultimately stationary at some rank $\epsilon < \omega$

– Because $\epsilon < \omega$, the chain $\langle X^\delta,\ \delta < \mathbb{O}\rangle$ is well-defined since $\lambda \geq \epsilon$ in case (d) implies that $\overline{\bigsqcap}_{\beta<\lambda} X^\beta$ is well-defined and indeed equal to $X^\lambda = X^\epsilon$. So, by transfinite induction, $\langle X^\delta,\ \delta < \mathbb{O}\rangle$ is also well-defined and ultimately stationary at rank ϵ and so $\mathsf{gfp}^{\overline{\sqsubseteq}}_{\overline{A}}\ \overline{\mathcal{H}} = X^\epsilon$ is well-defined.

□

A soundness theorem on the correspondence between a concrete semantics and its local abstraction by a structural abstract semantics with widening/narrowing

THEOREM. If e and $\overline{e}$ are structurally identical, (LA1), (LA2) and (LA3) hold, concrete fixpoints exist, (W1), (W2), (N1) and (N2) hold, and

– $S_j \sqsubseteq_j \gamma_i(\overline{S}_j)$, $j = 1, \ldots, n$ (a)

– $X_k \sqsubseteq_{ki} \gamma_i(\overline{X}_k)$, $k = n+1, \ldots, m$ (b)

then

$$e[\![\mathcal{D}]\!][S](X) \sqsubseteq \gamma(\overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X}))$$

■

PROOF. – The proof is similar to the case of expressions with widenings (on page 73), except for the use of narrowings

– From this proof, we already know that by letting

$$\mathcal{F} \stackrel{\text{def}}{=} \lambda Y \cdot e[\![\mathcal{D}]\!][S](X, Y : \mathcal{D})$$

we have $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(\overline{A})$.

– Let $\langle X^\delta,\ \delta < \mathbb{O}\rangle$ be the iterates for $\overline{\mathcal{H}}$. We have shown that they are well-defined and ultimately stationary at rank ϵ such that $\mathsf{gfp}^{\overline{\sqsubseteq}}_{\overline{A}}\ \overline{\mathcal{H}} = X^\epsilon$.

– We have

$$\forall \delta \in \mathbb{O} : \mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^\delta)$$

The proof is by transfinite induction.

- We have $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(\overline{A})$ whence $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^0)$ since $X^0 = \overline{A}$

- If $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^\delta)$ by induction hypothesis, then
  - if $\overline{\mathcal{F}}(X^\delta) \mathrel{\overline{\sqsubset}} X^\delta$ then $X^{\delta+1} = X^\delta \mathbin{\overline{\Delta}} \overline{\mathcal{F}}(X^\delta)$, whence by (N1) $X^\delta \mathrel{\overline{\sqsubseteq}} X^{\delta+1}$ whence $\gamma(X^\delta) \sqsubseteq \gamma(X^{\delta+1})$ and so $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^{\delta+1})$ by transitivity
  - otherwise, $X^{\delta+1} = X^\delta$ and so $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^{\delta+1})$
- If λ is a limit ordinal then we know that $X^\lambda = X^\epsilon$ where $\epsilon < \omega \leq \lambda$ and so $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^\lambda)$

– We conclude that $\mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(X^\epsilon) = \gamma(\mathsf{gfp}^{\overline{\sqsubseteq}}_{\overline{A}}\ \overline{\mathcal{H}})$ whence $e[\![\mathcal{D}]\!][S](\overline{X}) = \mathsf{lfp}^{\sqsubseteq}_{\perp}\ \mathcal{F} \sqsubseteq \gamma(\mathsf{gfp}^{\overline{\sqsubseteq}}_{\overline{A}}\ \overline{\mathcal{H}}) = \overline{e}[\![\overline{\mathcal{D}}]\!][\overline{S}](\overline{X})$ in that case.

□

THEOREM. If (LA1), (LA2) and (LA3) do hold, concrete fixpoints exist, (W1), (W2), (N1) and (N2) hold, then for all $i \in \Delta$ and $C_i \in \mathrm{Com}_i$, we have

$$\mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \gamma_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])$$

■

PROOF. Same as in the "structural abstract semantics with widening" case. □

On monotony

– The abstract structural definitions are not assumed to be monotone (because of the presence of widenings which are essentially not monotone)
– Nevertheless, they have been shown to be
  - well-defined
  - sound abstractions

  using "local abstraction conditions" only
– The proof is by structural induction on the programming language syntax, but formulated independently of any particular programming language

On the use of widening/narrowing

– In lattices satisfying the ACC, one can chose $x \nabla y = x \cup y$ and $x \Delta y = x \cap y$
– In case of monotony and iteration form a pre/postfixpoint, one prefers $x \nabla y = y$ and $x \Delta y = y$

An abstract formalization of structural verification by abstract interpretation

Structural safety specification

- We consider a language $\langle \mathcal{L} = \bigcup_{i \in \Delta} \mathrm{Com}_i,\ \prec\rangle$ with syntactic components $C_i \in \mathrm{Com}_i$ and well-founded "immediate subcomponent relation" $\prec$
- The concrete semantics is given for all $i \in \Delta$, $C_i \in \mathrm{Com}_i$ by
  - $\langle \mathcal{D}_{C_i},\ \sqsubseteq_{C_i},\ \perp_{C_i},\ \sqcup_{C_i}\rangle$ concrete semantic domain (a)
  - $\mathcal{C}[\![C_i]\!] \in \mathcal{D}_{C_i}$ concrete semantics (b)
- A safety specification is
  $\mathcal{S} : C_i \in \mathrm{Com}_i \mapsto \mathcal{D}_{C_i},\ i \in \Delta$ (c)

Structural safety proof

- A safety proof is the proof that
  $\forall i \in \Delta : \forall C_i \in \mathrm{Com}_i : \mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \mathcal{S}[\![C_i]\!]$ (d)
- Informally: the semantics of commands satisfies their specification

Structural abstract safety specification and proof

- An abstract safety specification is
  - $\langle \widehat{\mathcal{D}}_{C_i},\ \widehat{\sqsubseteq}_{C_i},\ \widehat{\perp}_{C_i},\ \widehat{\sqcup}_{C_i}\rangle$ abstract domains (e)
  - $\widehat{\mathcal{S}}[\![C_i]\!] \in \widehat{\mathcal{D}}_{C_i}$ abstract spec. (f)
  - $\widehat{\gamma}_{C_i} \in \widehat{\mathcal{D}}_{C_i} \stackrel{\mathrm{m}}{\longmapsto} \mathcal{D}_{C_i}$ spec. concretization (g)
- An abstract safety proof is the proof that
  $\forall i \in \Delta : \forall C_i \in \mathrm{Com}_i : \mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \widehat{\gamma}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!])$ (d)

Abstract semantics

- An abstract safety verification by abstract interpretation consists in designing an abstract semantics for all $i \in \Delta$, $C_i \in \mathrm{Com}_i$
  - $\langle \overline{\mathcal{D}}_{C_i},\ \overline{\sqsubseteq}_{C_i},\ \overline{\perp}_{C_i},\ \overline{\sqcup}_{C_i}\rangle$ abstract semantic domain (i)
  - $\overline{\mathcal{C}}[\![C_i]\!] \in \overline{\mathcal{D}}_{C_i}$ abstract semantics (j)
  - $\overline{\gamma}_{C_i} \in \overline{\mathcal{D}}_{C_i} \stackrel{\mathrm{m}}{\longmapsto} \mathcal{D}_{C_i}$ concretization (k)

  which are sound, in that
  $\forall i \in \Delta : \forall C_i \in \mathrm{Com}_i : \mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \overline{\gamma}_{C_i}(\overline{\mathcal{C}}[\![C_i]\!])$ (ℓ)
  and effectively computable (thanks to the choice of computer representable abstract domains, transfer functions and widening/narrowing)

Choice of the abstractions

– The abstract domains $\langle \overline{\mathcal{D}}_{C_i}, \overline{\sqsubseteq}_{C_i}, \overline{\bot}_{C_i}, \overline{\sqcup}_{C_i}\rangle$ are chosen to be *more precise* than the abstract specification domains $\langle \widehat{\mathcal{D}}_{C_i}, \widehat{\sqsubseteq}_{C_i}, \widehat{\bot}_{C_i}, \widehat{\sqcup}_{C_i}\rangle$
– This can be formalized by the existence of concretizations:

$$\widehat{\overline{\gamma}}_{C_i} \in \widehat{\mathcal{D}}_{C_i} \xmapsto{\text{m}} \overline{\mathcal{D}}_{C_i} \qquad \text{(m)}$$

satisfying

$$\widehat{\gamma}_{C_i} \mathrel{\dot{\sqsupseteq}}_{C_i} \overline{\gamma}_{C_i} \circ \widehat{\overline{\gamma}}_{C_i} \qquad \text{(n)}$$

Abstract structural safety verification

– The abstract safety verification consists in checking that:

$$\overline{\mathcal{C}}[\![C_i]\!] \mathrel{\overline{\sqsubseteq}}_{C_i} \widehat{\overline{\gamma}}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!]) \qquad \text{(o)}$$

Soundness of the abstract safety specification

THEOREM. An abstract structural safety verification is sound. ■

PROOF. By the abstract check (o), we have $\overline{\mathcal{C}}[\![C_i]\!] \mathrel{\overline{\sqsubseteq}}_{C_i} \widehat{\overline{\gamma}}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!])$ whence by monotony (m) $\overline{\gamma}_{C_i}(\overline{\mathcal{C}}[\![C_i]\!]) \mathrel{\overline{\sqsubseteq}}_{C_i} \overline{\gamma}_{C_i} \circ \widehat{\overline{\gamma}}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!])$ and so by (n) and transitivity, $\overline{\gamma}_{C_i}(\overline{\mathcal{C}}[\![C_i]\!]) \mathrel{\overline{\sqsubseteq}}_{C_i} \widehat{\gamma}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!])$ whence, by soundness (ℓ) of the abstraction, we conclude $\mathcal{C}[\![C_i]\!] \sqsubseteq_{C_i} \widehat{\gamma}_{C_i}(\widehat{\mathcal{S}}[\![C_i]\!])$, proving soundness. □

Example of structural safety specification for arithmetic expressions: absence of runtime errors

– The execution of an arithmetic expression A in any environment $\rho \in R \subseteq (\mathrm{Var}[\![P]\!] \mapsto \mathbb{I}_\Omega)$ is without any runtime error if and only if

$$\begin{array}{rl} & \mathrm{Faexp}[\![A]\!]R \cap \mathbb{E} = \emptyset \\ \Longleftrightarrow & \mathrm{Faexp}[\![A]\!]R \subseteq \mathbb{I} \end{array}$$

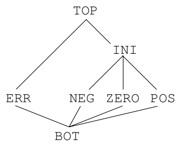
where Faexp is the forward collecting semantics of arithmetic expressions.

– If we define $\widehat{\mathcal{D}}_A \stackrel{\text{def}}{=} \begin{array}{c}\dot{\mathbb{I}}_\Omega \\ | \\ \dot{\mathbb{I}}\end{array}$ with $\widehat{\gamma}_A(\dot{\mathbb{I}}_\Omega) = \lambda R \cdot \mathbb{I}_\Omega$ and $\widehat{\gamma}_A(\dot{\mathbb{I}}) = \lambda R \cdot \mathbb{I}$ then the abstract safety proof is (c), (h):

$$\begin{array}{c}\widehat{\mathcal{S}}[\![A]\!] = \dot{\mathbb{I}} \\ \mathsf{Faexp}[\![A]\!] \mathrel{\dot{\subseteq}} \widehat{\gamma}_A(\widehat{\mathcal{S}}[\![A]\!])\end{array}$$

Example of too imprecise abstraction

– Observe that this cannot be checked with the initialization and simple sign abstraction:



$$\begin{array}{llll}\gamma(\mathtt{BOT}) \stackrel{\text{def}}{=} \{\Omega_{\mathrm{a}}\} & & \gamma(\mathtt{INI}) \stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\mathrm{a}}\}, \\ \gamma(\mathtt{NEG}) \stackrel{\text{def}}{=} [\mathtt{min\_int}, -1] \cup \{\Omega_{\mathrm{a}}\} & & \gamma(\mathtt{ERR}) \stackrel{\text{def}}{=} \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \\ \gamma(\mathtt{ZERO}) \stackrel{\text{def}}{=} \{0, \Omega_{\mathrm{a}}\} & & \gamma(\mathtt{TOP}) \stackrel{\text{def}}{=} \mathbb{I}_\Omega \\ \gamma(\mathtt{POS}) \stackrel{\text{def}}{=} [1, \mathtt{max\_int}] \cup \{\Omega_{\mathrm{a}}\} & & \end{array}$$

since defining $\widehat{\gamma}_A$ satisfying (m) and (n) is impossible since $\Omega_{\mathrm{a}} \notin \widehat{\gamma}_A(\dot{\mathbb{I}})(R)$

– We can only strengthen the analysis by refining the abstraction or weaken the specification

Example of concrete structural safety specification for arithmetic expressions: proper initialization

– The execution of an arithmetic expression A in any environment $\rho \in R \subseteq (\mathrm{Var}[\![P]\!] \mapsto \mathbb{I}_\Omega)$ is without any initialization error if and only if

$$\begin{array}{rl} & \mathsf{Faexp}[\![A]\!]R \cap \{\Omega_{\mathrm{i}}\} = \emptyset \\ \Longleftrightarrow & \mathsf{Faexp}[\![A]\!]R \subseteq \mathbb{I} \cup \{\Omega_{\mathrm{a}}\}\end{array}$$

where Faexp is the forward collecting semantics of arithmetic expressions, so we define in that case the concrete specification

$$\begin{array}{rll}\mathcal{S}[\![A]\!] & \stackrel{\text{def}}{=} \lambda R \cdot \mathbb{I} \cup \{\Omega_{\mathrm{a}}\} & \text{or} \\ & \stackrel{\text{def}}{=} \lambda R \cdot (\mathcal{P}(R) \mathrel{?} \mathbb{I} \cup \{\Omega_{\mathrm{a}}\} \mathrel{¿} \mathbb{I}_\Omega) & \end{array}$$

if we want to check absence initialization error under the hypothesis that some condition $\mathcal{P}(R)$ holds on the precondition R

Example of abstract structural safety specification for arithmetic expressions: proper initialization

– An abstract safety specification is

- $\widehat{\mathcal{D}}_A = \begin{array}{c}\mathrm{T\dot{O}P}\\ | \\ \mathrm{I\dot{N}I}\end{array}$
- $\widehat{\gamma}_A(\mathrm{T\dot{O}P}) = \lambda R \cdot \mathbb{I}_\Omega$, $\widehat{\gamma}_A(\mathrm{I\dot{N}I}) = \lambda R \cdot \mathbb{I} \cup \{\Omega_a\}$
- $\widehat{\mathcal{S}}[\![A]\!] = \mathrm{I\dot{N}I}$

– We have shown the abstract interpretation of arithmetic expressions to be sound $\mathrm{Faexp}^{\triangleright}[\![A]\!] \,\dot{\sqsupseteq}\, \alpha^{\triangleright}(\mathrm{Faexp}[\![A]\!])$ or equivalently $\mathrm{Faexp}[\![A]\!] \,\dot{\subseteq}\, \gamma^{\triangleright}(\mathrm{Faexp}^{\triangleright}[\![A]\!])$

– We define

$$\widehat{\gamma}^{\triangleright}_A(\mathrm{T\dot{O}P}) \stackrel{\mathrm{def}}{=} \lambda R \cdot \mathrm{TOP}$$
$$\widehat{\gamma}^{\triangleright}_A(\mathrm{I\dot{N}I}) \stackrel{\mathrm{def}}{=} \lambda R \cdot \mathrm{INI}$$

so that

$$\widehat{\gamma}_A = \gamma^{\triangleright} \circ \widehat{\gamma}^{\triangleright}_A$$

– The abstract safety verification condition is:

$$\mathrm{Faexp}^{\triangleright}[\![A]\!] \,\dot{\sqsubseteq}\, \widehat{\gamma}^{\triangleright}_A(\widehat{\mathcal{S}}[\![A]\!])$$
$$\Longleftrightarrow \quad \mathrm{Faexp}^{\triangleright}[\![A]\!] \,\dot{\sqsubseteq}\, \lambda R \cdot \mathrm{INI}$$

which implies

$$\gamma^{\triangleright}(\mathrm{Faexp}^{\triangleright}[\![A]\!]) \,\dot{\subseteq}\, \gamma^{\triangleright}(\lambda R \cdot \mathrm{INI})$$

whence

$$\forall R : \mathrm{Faexp}[\![A]\!]R \subseteq \mathbb{I} \cup \{\Omega_a\}$$

as required

Why choosing abstract specifications?

– The objective is to check the conformance of a semantics to a specification:

$$\mathrm{Sem} \sqsubseteq \mathrm{Spec} \qquad \text{(a)}$$

– We want to perform the check in the abstract:

$$\mathrm{Sem}^{\sharp} \,\overline{\sqsubseteq}\, \mathrm{Spec}^{\flat} \qquad \text{(b)}$$

so that it implies in the concrete:

$$\overline{\gamma}(\mathrm{Sem}^{\sharp}) \,\overline{\sqsubseteq}\, \overline{\gamma}(\mathrm{Spec}^{\flat}) \qquad \text{(c)}$$

– For (c) to imply (a) we need both:

$$\text{Sem} \sqsubseteq \overline{\gamma}(\text{Sem}^\sharp) \quad \text{and} \quad \overline{\gamma}(\text{Spec}^\flat) \sqsubseteq \text{Spec} \qquad (1)$$

– $\text{Sem} \sqsubseteq \overline{\gamma}(\text{Sem}^\sharp)$ is an *approximation from above*, which is pretty well studied
– $\overline{\gamma}(\text{Spec}^\flat) \sqsubseteq \text{Spec}$ is an *approximation from below* for which only finite abstractions are known to be automatizable, while specifications are most often infinite!

– By choosing *abstract specifications only*, we solve the problem by choosing

$$\text{Spec} = \overline{\gamma}(\text{Spec}^\flat)$$

but we are left with the problem of finding adequate machine representations of the specifications as abstract domain
– Progress is necessary in the abstraction of specifications from below!

Why choosing an abstract semantics more refined than an abstract specifications?

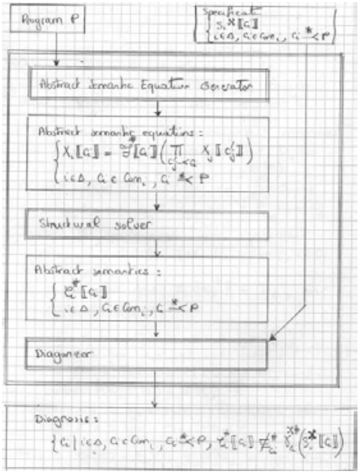
– The fact that the abstract semantics should be more refined than the abstract specification is similar the proof of theorem requiring stringer arguments in the proof
– For example, with Floyd's method

$$\begin{aligned} & \mathsf{lfp}_{\perp}^{\sqsubseteq} F \sqsubseteq P \\ \iff & \exists I : F(I) \sqsubseteq I \wedge I \sqsubseteq P \end{aligned}$$

P is *invariant* while the proof requires to find a stronger *inductive invariant* (while, in general $F(P) \not\sqsubseteq P$)

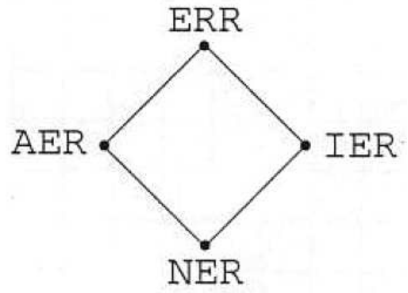
– Similarly we can always choose the abstraction $\overline{\mathcal{D}}_A = \widehat{\mathcal{D}}_A$ as a starting point, but in general refinements are needed
– While in Floyd's method or abstract model checking this refinement is done for a particular program, the difficulty in this refinement must be done for a language

Principle of a structural static analyzer/verifier



Example of abstract domain: Error analysis

Error abstraction



– Takes initialization and arithmetic errors into account

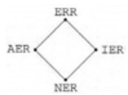
– The abstract properties are $\langle E, \sqsubseteq_E\rangle$ where $E \stackrel{\text{def}}{=} \{\texttt{NER}, \texttt{AER}, \texttt{IER}, \texttt{ERR}\}$ and the partial order is defined by the Hasse diagram

– Concretization:

$$\gamma_E(\texttt{NER}) \stackrel{\text{def}}{=} \mathbb{I}$$
$$\gamma_E(\texttt{AER}) \stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\texttt{a}}\}$$
$$\gamma_E(\texttt{IER}) \stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\texttt{i}}\}$$
$$\gamma_E(\texttt{ERR}) \stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\texttt{i}}, \Omega_{\texttt{a}}\} = \mathbb{I}_\Omega$$

The error complete lattice

The finite lattice



is obviously a complete lattice, with

– Partial ordering: $\mathtt{NER} \sqsubseteq_E \mathtt{NER}$ $\mathtt{AER} \sqsubseteq_E \mathtt{AER}$ $\mathtt{ERR} \sqsubseteq_E \mathtt{ERR}$ and $\mathtt{NER} \sqsubseteq_E \mathtt{IER} \sqsubseteq_E \mathtt{IER} \sqsubseteq_E \mathtt{ERR}$

– lub: $x \sqcup_E x = x$ $\quad \mathtt{AER} \sqcup_E \mathtt{IER} = \mathtt{ERR}$
$\mathtt{NER} \sqcup_E x = x$ $\quad x \sqcup_E y = y \sqcup_E x$
$\mathtt{ERR} \sqcup_E x = \mathtt{ERR}$

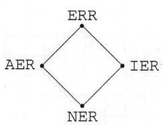
– glb: $x \sqcap_E x = x$ $\quad \mathtt{AER} \sqcap_E \mathtt{IER} = \mathtt{NER}$
$\mathtt{NER} \sqcap_E x = \mathtt{NER}$ $\quad x \sqcap_E y = y \sqcap_E x$
$\mathtt{ERR} \sqcap_E x = x$

– infimum: NER

– supremum: ERR

The error abstraction

We have defined



$$\begin{aligned}\gamma_E(\mathtt{NER}) &\stackrel{\text{def}}{=} \mathbb{I}\\ \gamma_E(\mathtt{AER}) &\stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\mathrm{a}}\}\\ \gamma_E(\mathtt{IER}) &\stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\mathrm{i}}\}\\ \gamma_E(\mathtt{ERR}) &\stackrel{\text{def}}{=} \mathbb{I} \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} = \mathbb{I}_\Omega\end{aligned}$$

so we let

$$\begin{aligned}\alpha_E(X) \stackrel{\text{def}}{=} (\,& X \subseteq \mathbb{I} \mathrel{?} \mathtt{NER}\\ \parallel\, & X \subseteq \mathbb{I} \cup \{\Omega_{\mathrm{a}}\} \mathrel{?} \mathtt{AER}\\ \parallel\, & X \subseteq \mathbb{I} \cup \{\Omega_{\mathrm{i}}\} \mathrel{?} \mathtt{IER}\\ \mathrel{¿}\, & \mathtt{ERR})\end{aligned}$$

and we have

$$\langle \wp(\mathbb{I}_\Omega), \subseteq \rangle \xleftarrow[\alpha_E]{\gamma_E} \langle E, \sqsubseteq_E \rangle$$

Proof. To prove $\alpha(x) \sqsubseteq_E y \iff x \subseteq \gamma_E(y)$, we consider 4 cases for $y = \mathtt{NER}$, $y = \mathtt{AER}$, $y = \mathtt{IER}$ and $y = \mathtt{ERR}$. Since all cases are very similar and the proof is tedious, we consider only the case $y = \mathtt{AER}$ and prove $\alpha(x) \sqsubseteq_E \mathtt{AER} \iff x \subseteq \gamma_E(\mathtt{AER})$.

– If $\alpha(x) \sqsubseteq_E \mathtt{AER}$ then either $\alpha(x) = \mathtt{NER}$ or $\alpha(x) = \mathtt{AER}$
  - If $\alpha(x) = \mathtt{NER}$ then $x \subseteq \mathbb{I} = \gamma_E(\mathtt{NER})$
  - Else $\alpha(x) = \mathtt{AER}$ and then $x \subseteq \mathbb{I} \cup \{\Omega_{\mathrm{a}}\} = \gamma_E(\mathtt{AER})$

– Reciprocally, if $x \subseteq \gamma_E(\mathtt{AER})$ then $x \subseteq \mathbb{I} \cup \{\Omega_{\mathrm{a}}\}$.
  - If $x \subseteq \mathbb{I}$ then $\alpha(x) = \mathtt{NER} \sqsubseteq_E \mathtt{AER}$
  - Otherwise $\alpha(x) = \mathtt{AER} \sqsubseteq_E \mathtt{AER}$

$\square$

The error analysis abstract domain

```
(* avalues.ml *)
open Values
(* abstraction of sets of machine integers by errors *)
(* complete lattice *)
type t = NER | AER | IER | ERR
(* gamma(NER) = [min_int,max_int]                     *)
(* gamma(AER) = [min_int,max_int] U {_O_(a)}          *)
(* gamma(IER) = [min_int,max_int] U {_O_(i)}          *)
(* gamma(ERR) = [min_int,max_int] U {_O_(a), _O_(i)} *)
(* infimum *)
let bot () = NER
(* bottom is emptyset? *)
let isbotempty () = false
(* uninitialization *)
```

```
let initerr () = IER
(* supremum *)
let top () =  ERR
(* least upper bound *)
let nat_of_lat u =
      match u with
      | NER -> 0
      | AER -> 1
      | IER -> 2
      | ERR -> 3
let select t u v = t.(nat_of_lat u).(nat_of_lat v)
let join_table =
(*          NER   AER   IER   ERR    *)
(*NER*)[|[| NER ; AER ; IER ; ERR ; |];
(*AER*)  [| AER ; AER ; ERR ; ERR ; |];
(*IER*)  [| IER ; ERR ; IER ; ERR ; |];
(*ERR*)  [| ERR ; ERR ; ERR ; ERR ; |]|]
let join u v = select join_table u v
```



```
(* greatest lower bound *)
let meet_table =
(*          NER   AER   IER   ERR    *)
(*NER*)[|[| NER ; NER ; NER ; NER ; |];
(*AER*)  [| NER ; AER ; NER ; AER ; |];
(*IER*)  [| NER ; NER ; IER ; IER ; |];
(*ERR*)  [| NER ; AER ; IER ; ERR ; |]|]
let  meet u v = select join_table u v
(* approximation ordering *)
let leq_table =
(*          NER     AER     IER     ERR    *)
(*NER*)[|[| true ;  true ;  true ;  true ; |];
(*AER*)  [| false ; true ;  false ; true ; |];
(*IER*)  [| false ; false ; true ;  true ; |];
(*ERR*)  [| false ; false ; false ; true ; |]|]
let leq u v = select leq_table u v
(* equality *)
let eq u v = (u = v)
```



```
(* included in errors? *)
let in_errors v = (leq v ERR)
(* printing *)
let print u = match u with
| NER -> print_string "{}"
| AER -> print_string "{_O_a}"
| IER -> print_string "{_O_i}"
| ERR -> print_string "{_O_a,_O_i}"
(* forward abstract semantics of arithmetic expressions *)
(* f_NAT s = \alpha({(machine_int_of_string s)})        *)
let f_NAT s =
   match (machine_int_of_string s) with
   | (ERROR_NAT INITIALIZATION) -> IER
   | (ERROR_NAT ARITHMETIC) -> AER
   | (NAT i) -> NER
(* f_RANDOM () = alpha([min_int, max_int]) *)
let f_RANDOM () = NER
(* f_UMINUS a = alpha({ (machine_unary_minus x) | x \in gamma(a)} }) *)
```



```
let f_UMINUS a =
   match a with
   | NER -> AER (* a can be min_int *)
   | AER -> AER
   | IER -> IER
   | ERR -> ERR
(* f_UPLUS a = alpha(gamma(a)) *)
let f_UPLUS a = a
(* f_BINARITH a b = alpha({ (machine_binary_binarith i j)| }) |  *)
(*                              i in gamma(a) /\ j \in gamma(b)} *)
let f_BINARITH a b =
   match a with
   | NER -> (match b with
              | NER -> AER
              | AER -> AER
              | IER -> IER
              | ERR -> ERR)
   | AER -> AER
```

```
    | IER -> IER
    | ERR -> ERR
let f_PLUS  = f_BINARITH
let f_MINUS = f_BINARITH
let f_TIMES = f_BINARITH
let f_DIV   = f_BINARITH
let f_MOD   = f_BINARITH
(* forward abstract semantics of boolean expressions *)
(* Are there integer values in gamma(u) equal to values in gamma(v)? *)
let f_EQ u v = true
(* Are there integer values in gamma(u) less than or equal to (<=) *)
(* integer values in gamma(v)?                                      *)
let f_LT u v = true
(* widening *)
let widen v w = w
(* narrowing *)
let narrow v w = w
(* backward abstract semantics of arithmetic expressions *)
```



```
(* b_NAT s v = (machine_int_of_string s) in gamma(v) cap I? *)
let b_NAT s p =
   match (machine_int_of_string s) with
   | (ERROR_NAT INITIALIZATION) -> false
   | (ERROR_NAT ARITHMETIC) -> false
   | (NAT i) -> true
(* b_RANDOM p = gamma(p) cap I <> emptyset *)
let b_RANDOM p = true
(* b_UOP q p = alpha({i in gamma(q) | UOP(i) \in gamma(p) cap   *)
(*                                          [min_int, max_int]}) *)
let b_UMINUS q p = NER
let b_UPLUS q p = NER
(* b_BOP q1 q2 p = alpha2({<i1,i2>  in gamma2(<q1,q2>) |        *)
(*             BOP(i1, i2) \in gamma(p) cap [min_int, max_int]}) *)
let b_PLUS q1 q2 p = NER, NER
let b_MINUS q1 q2 p = NER, NER
let b_TIMES q1 q2 p = NER, NER
let b_DIV q1 q2 p = NER, NER
```



```
let b_MOD q1 q2 p = NER, NER
(* backward abstract interpretation of boolean expressions         *)
(* a_EQ p1 p2 = let p = p1 cap p2 cap [min_int, max_int] in <p, p> *)
let a_EQ p1 p2 = NER, NER
(* a_LT p1 p2 = alpha({<i1, i2> |                                  *)
(*              i1 in gamma(p1) cap [min_int, max_int] /\          *)
(*              i2 in gamma(p1) cap [min_int, max_int] /\ i1 <= i2}) *)
let a_LT p1 p2 = NER, NER
```


Example of abstract domain: Parity analysis

The parity analysis abstract domain

```
(* avalues.ml *)
open Values
(* abstraction of sets of machine integers by parity *)
(* complete lattice *)
type t = BOT | ODD | EVEN | TOP
(*                 TOP                                 *)
(*                 /\                                  *)
(*                /  \                                 *)
(*               ODD EVEN                              *)
(*                \  /                                 *)
(*                 \/                                   *)
(*                 BOT                                 *)
(* \gamma(BOT)  = {_O_(a)}                             *)
(* \gamma(ODD)  = { 2n+1\in[min_int,max_int] | n\in Z } U {_O_(a)} *)
```



```
(* \gamma(EVEN) = { 2n\in[min_int,max_int] | n\in Z } U {_O_(a)}   *)
(* \gamma(TOP)  = [min_int,max_int] U {_O_(a),_O_(i)}              *)
let bot () = BOT
(* bottom is emptyset? *)
let isbotempty () = false   (* \gamma(BOT) = {_O_(a)} <> \emptyset *)
(* uninitialization *)
let initerr () = TOP
(* supremum *)
let top () = TOP
(* least upper bound *)
let join v w =
  if (v = BOT) then w
  else if (w = BOT) then v
  else if (v = w) then w
  else TOP
(* greatest lower bound *)
let meet v w =
  if (v = TOP) then w
```



```
  else if (w = TOP) then v
  else if (v = w) then w
  else BOT
(* approximation ordering *)
let leq v w =
  if (v = BOT) then true
  else if( w = TOP) then true
  else v = w
(* equality *)
let eq u v = (u = v)
(* included in errors? *)
let in_errors u = (u = BOT)
(* printing *)
let print u =
    match u with
    | BOT  -> print_string "_|_"
    | ODD  -> print_string "o"
    | EVEN -> print_string "e"
```



```
    | TOP  -> print_string "T"
(* forward abstract semantics of arithmetic expressions *)
(* f_NAT s = \alpha({(machine_int_of_string s)})       *)
let rec pry_of_intstring i s =
  let l = (String.length s) in
    if l = 0 then
      (if (i mod 2) = 0 then EVEN else ODD)
    else
      let v = (10 * i) + (int_of_string (String.sub s 0 1)) in
        if v < i then (* overflow *)
          BOT (* = \alpha({_O_(a)}) *)
        else
          pry_of_intstring v (String.sub s 1 (l-1))
let parity_of_intstring i = pry_of_intstring 0 i
let f_NAT i = parity_of_intstring i
(* f_RANDOM () = alpha([min_int, max_int]) *)
let f_RANDOM () = TOP
(* f_UMINUS a = alpha({ (machine_unary_minus x) | x \in gamma(a)} }) *)
```

```
let f_UMINUS u = u
(* f_UPLUS a = alpha(gamma(a)) *)
let f_UPLUS a = a
(* f_BINARITH a b = alpha({ (machine_binary_binarith i j)| }) |  *)
(*                              i in gamma(a) /\ j \in gamma(b)} *)
let nat_of_lat u =
  match u with
  | BOT  -> 0
  | ODD  -> 1
  | EVEN -> 2
  | TOP  -> 3
let select t u v = t.(nat_of_lat u). (nat_of_lat v)
let f_PLUS_table =
(*     +    BOT   ODD    EVEN   TOP  *)
(*BOT*)[|[| BOT ; BOT  ; BOT  ; BOT |];
(*ODD*)  [| BOT ; EVEN ; ODD  ; TOP |];
(*EVEN*) [| BOT ; ODD  ; EVEN ; TOP |];
(*TOP*)  [| BOT ; TOP  ; TOP  ; TOP |]|]
```



```
let f_PLUS  u v = select f_PLUS_table u v
let f_MINUS = f_PLUS
let f_TIMES_table =
(*     *    BOT   ODD    EVEN   TOP  *)
(*BOT*)[|[| BOT ; BOT ;  BOT  ; BOT |];
(*ODD*)  [| BOT ; ODD ;  EVEN ; TOP |];
(*EVEN*) [| BOT ; EVEN ; EVEN ; TOP |];
(*TOP*)  [| BOT ; TOP ;  TOP  ; TOP |]|]
let f_TIMES u v = select f_TIMES_table u v
let f_DIV_table =
(*     /    BOT   ODD   EVEN  TOP  *)
(*BOT*)[|[| BOT ; BOT ; BOT ; BOT |];
(*ODD*)  [| BOT ; TOP ; TOP ; TOP |];
(*EVEN*) [| BOT ; TOP ; TOP ; TOP |];
(*TOP*)  [| BOT ; TOP ; TOP ; TOP |]|]
let f_DIV   u v = select f_DIV_table u v
let f_MOD = f_DIV
(* forward abstract semantics of boolean expressions *)
```



```
(* Are there integer values in gamma(u) equal to values in gamma(v)? *)
let f_EQ u v = (u = TOP) || (v = TOP) || ((u = v) & (u != BOT))
(* Are there integer values in gamma(u) less than or equal to (<=) *)
(* integer values in gamma(v)?                                     *)
let f_LT u v = ((u != BOT) & (v != BOT))
(* widening *)
let widen v w = w
(* narrowing *)
let narrow v w = w
(* backward abstract semantics of arithmetic expressions *)
(* b_NAT s v = (machine_int_of_string s) in gamma(v) cap I? *)
exception Error_b_NAT of string
let b_NAT n p =
  match (String.get n (String.length n - 1)) with
  | '0'  -> leq EVEN p
  | '1'  -> leq ODD p
  | '2'  -> leq EVEN p
  | '3'  -> leq ODD p
```



```
  | '4'  -> leq EVEN p
  | '5'  -> leq ODD p
  | '6'  -> leq EVEN p
  | '7'  -> leq ODD p
  | '8'  -> leq EVEN p
  | '9'  -> leq ODD p
  | _    -> raise (Error_b_NAT "not a digit")
(* b_RANDOM p = gamma(p) cap I <> emptyset *)
let b_RANDOM p =
  match p with
  | BOT -> false
  | _   -> true
(* backward abstract semantics of arithmetic expressions      *)
(* b_NAT s v = (machine_int_of_string s) in gamma(v) cap      *)
(*                                         [min_int, max_int]? *)
let b_UMINUS q p = meet q p
let b_UPLUS q p = meet q p
(* b_BOP q1 q2 p = alpha2({<i1,i2>  in gamma2(<q1,q2>) |      *)
```

```
(*                         BOP(i1, i2) \in gamma(p) cap        *)
(*                                        [min_int, max_int]}) *)
exception Error_b_PLUS of string
let nat_of_lat' u =
  match u with
  | ODD  -> 0
  | EVEN -> 1
  | TOP  -> 2
  | _    -> raise (Error_b_PLUS "impossible selection")
let select' t u v = t.(nat_of_lat' u).(nat_of_lat' v)
let b_PLUS_ODD_table =
(*             ODD           EVEN        TOP       *)
(*ODD*)[|[| (BOT,BOT)  ; (ODD,EVEN) ; (ODD,EVEN) |];
(*EVEN*) [| (EVEN,ODD) ; (BOT,BOT)  ; (EVEN,ODD) |];
(*TOP*)  [| (EVEN,ODD) ; (ODD,EVEN) ; (TOP,TOP)  |]|]
let b_PLUS_EVEN_table =
(*             ODD          EVEN          TOP       *)
(*ODD*)[|[| (ODD,ODD) ; (BOT,BOT)   ; (ODD,ODD)   |];
```



```
(*EVEN*) [| (BOT,BOT) ; (EVEN,EVEN) ; (EVEN,EVEN) |];
(*TOP*)  [| (ODD,ODD) ; (EVEN,EVEN) ; (TOP,TOP)   |]|]
let b_PLUS q1 q2 p =
  if (q1=BOT)||(q2=BOT)||(p=BOT) then
    (BOT,BOT)
  else if (p=TOP) then
    (q1,q2)
  else if p = ODD then  select' b_PLUS_ODD_table q1 q2
  else if p = EVEN then select' b_PLUS_EVEN_table q1 q2
  else raise (Error_b_PLUS "impossible case")
let b_MINUS = b_PLUS
let b_TIMES_ODD_table =
(*             ODD           EVEN        TOP       *)
(*ODD*)[|[| (ODD,ODD) ; (BOT,BOT) ; (ODD,ODD) |];
(*EVEN*) [| (BOT,BOT) ; (BOT,BOT) ; (BOT,BOT) |];
(*TOP*)  [| (ODD,ODD) ; (BOT,BOT) ; (ODD,ODD) |]|]
let b_TIMES_EVEN_table =
(*             ODD          EVEN          TOP       *)
```



```
(*ODD*)[|[| (BOT,BOT)  ; (ODD,EVEN)  ; (ODD,EVEN) |];
(*EVEN*) [| (EVEN,ODD) ; (EVEN,EVEN) ; (EVEN,TOP) |];
(*TOP*)  [| (EVEN,ODD) ; (TOP,EVEN)  ; (TOP,TOP)  |]|]
exception Error_b_TIMES of string
let b_TIMES q1 q2 p =
  if (q1=BOT)||(q2=BOT)||(p=BOT) then
    (BOT,BOT)
  else if (p=TOP) then
    (q1,q2)
  else if p = ODD then  select' b_TIMES_ODD_table q1 q2
  else if p = EVEN then select' b_TIMES_EVEN_table q1 q2
  else raise (Error_b_TIMES "impossible case")
let b_DIV q1 q2 p =
  if (q1=BOT)||(q2=BOT)||(p=BOT) then
(BOT,BOT)
  else
     (q1,q2)
let b_MOD = b_DIV
```



```
(* backward abstract interpretation of boolean expressions        *)
(* a_EQ p1 p2 = let p = p1 cap p2 cap [min_int, max_int] in <p, p> *)
let a_EQ p1 p2 =
  let p = (meet p1 (meet p2 (f_RANDOM ()))) in
    (p,p)
(* a_LT p1 p2 = alpha({<i1, i2> | i1 in gamma(p1) cap [min_int,     *)
(*  max_int] /\ i2 in gamma(p1) cap [min_int, max_int] /\ i1 <= i2}) *)
let a_LT_table =
(*     <        BOT        ODD        EVEN          TOP     *)
(*BOT*)[|[| (BOT,BOT); (BOT,BOT) ; (BOT,BOT)   ; (BOT,BOT)  |];
(*ODD*)   [| (BOT,BOT); (ODD,ODD) ; (ODD,EVEN)  ; (ODD,TOP)  |];
(*EVEN*)  [| (BOT,BOT); (EVEN,ODD); (EVEN,EVEN); (EVEN,TOP) |];
(*TOP*)   [| (BOT,BOT); (TOP,ODD) ; (TOP,EVEN)  ; (TOP,TOP)  |]|]
let a_LT   u v = select a_LT_table u v
```

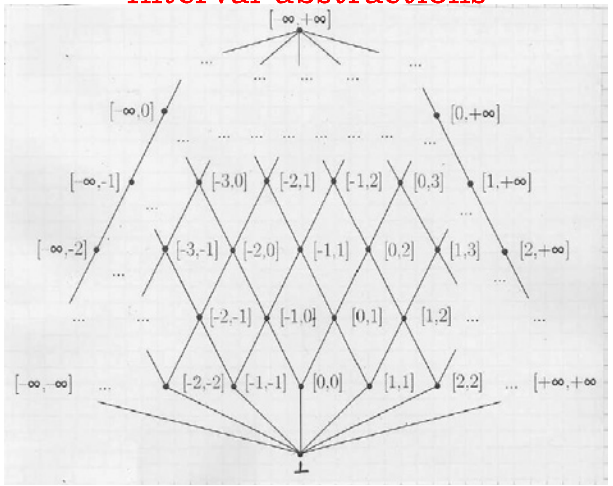
Parity analysis example

```
** Input file:
% example02.sil %
x := -1073741823 -1;
y := x - 1;;
{ x:T; y:T }
0:
  x := (-1073741823 - 1);
1:
  y := (x - 1)
2:
{ x:e; y:o }
```


Example of abstract domain: Interval analysis

Design of the abstract properties

Interval abstractions



- In the traditional lattice for interval analysis [1], a supremum $+\infty$ and an infimum $-\infty$ are added to reason on the complete lattice $\langle \mathbb{Z} \cup \{-\infty, +\infty\}, \leq\rangle$ [1]
- This is appropriate for mathematical, machine-independent reasoning only
- In practice we have $+\infty = \texttt{max\_int}$ and $-\infty = \texttt{min\_int}$ to take the finite machine representation of integers into account: $\langle\{z \in \mathbb{Z} \mid -\infty \leq z \leq +\infty\}, \leq\rangle$ that is $\langle \mathbb{I}, \leq\rangle$
- The abstract properties are $I \stackrel{\text{def}}{=} \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{I} \wedge -\infty \leq a \leq b \leq +\infty\}$

- The meaning of abstract properties is

$$\gamma_i(\bot) \stackrel{\text{def}}{=} \emptyset$$
$$\gamma_i([a, b]) \stackrel{\text{def}}{=} \{z \in \mathbb{I} \mid a \leq z \leq b\}$$

- This also works for floating points (care must be taken to over-estimate the bounds in case of roundings [2, 3])

Reference

[1] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conf. Rec. Fourth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[2] Antoine Miné. "Relational abstract domains for the detection of floating-point run-time errors". In ESOP 2004 — European Symposium on Programming, D. Schmidt (editor), Mar. 27 — Apr. 4, 2004, Barcelona, Lecture Notes in Computer Science 2986, pp. 3—17, © Springer.

[3] Antoine Miné. "Weakly relational numerical abstract domains". PhD thesis, École polytechnique, 6 December 2004.

Error and interval abstraction

- Combine interval and error information
- The lattice of program properties is $I \times E$
- The concretization is
$$\gamma(\langle i,\ e\rangle) \stackrel{\text{def}}{=} (\gamma_i(i) \cup \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\}) \cap \gamma_E(e)$$
- Intervals bring no information of errors
- Errors bring no range information
- The combination provide both range and error information
- This is an example of reduced product

The partial order of intervals

- For intervals we let $+\infty = \texttt{max\_int}$ and $-\infty = \texttt{min\_int}$ so that $\forall i \in \mathbb{I} : -\infty \leq i \leq +\infty$
- The Hasse diagram defines the interval abstract properties is $I \stackrel{\text{def}}{=} \{\bot\} \cup \{[a, b] \mid a, b \in \mathbb{I} \wedge -\infty \leq a \leq b \leq +\infty\}$
- The Hasse diagram defines the interval partial order $\sqsubseteq_I$ as

$$\forall x \in I : \bot \sqsubseteq_I i$$
$$\forall [a, b], [c, d] \in I : ([a, b] \sqsubseteq_I [c, d]) \iff (a \leq c \leq d \leq b)$$

THEOREM. $\langle I,\ \sqsubseteq_I \rangle$ is a partial order. ■

PROOF. – By def. of $\sqsubseteq_I$ and reflexivity of $\leq$, $\sqsubseteq_I$ is reflexive

– If $i \sqsubseteq_I j$ and $j \sqsubseteq_I k$ then
  - If $i = \bot$ then $i = \bot \sqsubseteq_I k$
  - Else $i = [a, b]$ so $j = [c, d]$ so $k = [e, f]$. By def. of $\sqsubseteq_I$, $i \sqsubseteq_I j$ implies $c \leq a \leq b \leq d$ and $j \sqsubseteq_I k$ implies $e \leq c \leq d \leq f$ so that by transitivity of $\leq$, we get $e \leq a \leq b \leq f$ proving $i \sqsubseteq_I k$

  In both cases $i \sqsubseteq_I k$ proving transitivity

– If $i \sqsubseteq_I j$ and $j \sqsubseteq_I i$ then
  - if $i = \bot$ then $j \sqsubseteq_I \bot$ implies $j = \bot$ by def. of $\sqsubseteq_I$ so $i = j$
  - if $i = [a, b]$ then $j = [c, d]$ since $i \sqsubseteq_I j$ by def. of $\sqsubseteq_I$. This implies $c \leq a \leq b \leq d$. $j \sqsubseteq_I i$ implies $a \leq b \leq d \leq b$ so by antisymmetry of $\leq$, we get $a = c \sqsubseteq_I b = d$ so $i = j$.

  In both cases $i = j$ proving antisymmetry.

– We conclude that $\sqsubseteq_I$ is a partial order on I

□

The complete lattice of intervals

– $\langle I, \sqsubseteq_I, \bot, [-\infty, \infty], \sqcup_I, \sqcap_I \rangle$ is a complete lattice, where:
  - $\bigsqcup_{I\, j \in \Delta} i_j \stackrel{\text{def}}{=} \bigsqcup_I \{i_j \mid j \in \Delta \wedge i_j \neq \bot\}$
  - $\bigsqcup_I \emptyset = \bot$
  - $\bigsqcup_{I\, j \in \Delta} [a_j, b_j] \stackrel{\text{def}}{=} [\min_{I\, j \in \Delta} a_j, \max_{I\, j \in \Delta} b_j]$ where min and max are extended on $\mathbb{I}$ to $-\infty$ and $+\infty$ in the natural way

THEOREM. $\sqcup_I$ is the lub. ■

PROOF. – Let S be any subset of I and $\ell = \bigsqcup_I S$. Then $\ell \in I$.

– To prove that it is an upper bound, either S is empty so $\ell = \bot$ is the infimum whence the lub of the empty set or, there exists $s \in S$. If $s = \bot$ then ℓ is an upper bound. Otherwise $s = [a, b]$. Let $S' = \{s \in S \mid s \neq \bot\}$. It is of the form $S' = \{[a_j, b_j] \mid j \in \Delta\}$ and $s \in S'$. By def. of $\bigsqcup_I$, we have $\bigsqcup_I S = \bigsqcup_I S' = [\min_{I\, j \in \Delta} a_j, \max_{I\, j \in \Delta} b_j]$ and so $\min_{I\, j \in \Delta} a_j \leq a \leq b \leq \max_{I\, j \in \Delta} b_j$ proving that $s \bigsqcup_I S$ by def. $\sqsubseteq_I$

– Let u be any other upper bound of S. Let $S' = \{s \in S \mid s \neq \bot\}$ so that $\bigsqcup_I S = \bigsqcup_I S'$ and $\bot \notin S'$.
  - If S' is empty, then $\bigsqcup_I S = \bigsqcup_I S' = \bot \sqsubseteq_I u$, by def. $\sqsubseteq_I$
  - Otherwise, $S' = \{[a_j, b_j] \mid j \in \Delta\}$. By def. $\bigsqcup_I$, we have $\bigsqcup_I S = \bigsqcup_I S' = [\min_{I\, j \in \Delta} a_j, \max_{I\, j \in \Delta} b_j]$. Since u is an upper bound of S whence S', we have $\forall j \in \Delta : [a_j, b_j] \sqsubseteq_I u = [a_u, b_u]$, whence $a_u \leq a_j \leq b_j \leq b_u$, by def. $\sqsubseteq_I$. It follows, that $a_u \leq \min_{I\, j \in \Delta} a_j \leq \max_{I\, j \in \Delta} b_j \leq b_u$ proving that $\bigsqcup_I S = \bigsqcup_I S' \sqsubseteq_I u$, by def. $\sqsubseteq_I$

  In both cases, we conclude that $\bigsqcup_I S$ is the lub of $\bigsqcup_I S$

□

By existence of lubs, it follows that the poset $\langle I,\ \sqsubseteq_I \rangle$ is a complete lattice $\langle I, \sqsubseteq_I, \bot, [-\infty, \infty], \sqcup_I, \sqcap_I \rangle$. We have:

– $[-\infty, +\infty]$ is the top

– The glb $\sqcap_I$ is defined as follows:
  - if $\bot \in S$ then $\bigsqcap_I S = \bot$
  - If $\bot \notin S$ then $S = \{[a_j, b_j] \mid j \in \Delta\}$ and then
    · $\bigsqcap_I S = \bot$ if $\min_{I\, j \in \Delta} a_j < \max_{I\, j \in \Delta} b_j$
    · $\bigsqcap_I S = [\max_{I\, j \in \Delta} a_j, \min_{I\, j \in \Delta} b_j]$ if $\max_{I\, j \in \Delta} a_j \leq \min_{I\, j \in \Delta} b_j$

PROOF. – By def. $\sqsubseteq_I$, we have $\bot \sqsubseteq_I [-\infty, +\infty]$ and for all $a, b \in \mathbb{I} : -\infty \leq a \leq b \leq +\infty$ and so $[a, b] \sqsubseteq_I [-\infty, +\infty]$, proving $[-\infty, +\infty]$ to be the supremum

- If $\bigsqcap_I S = \bot$, the obviously $\bigsqcap_I S$ is a lower bound of S
- Otherwise, $S = \{[a_j, b_j] \mid j \in \Delta\}$ and $\max_{I\, j\in\Delta} a_j \leq \min_{I\, j\in\Delta} b_j$. The for all elements of S, i.e. $i \in \Delta$, we have $a_i \leq \max_{I\, j\in\Delta} a_j \leq \min_{I\, j\in\Delta} b_j \leq b_i$ whence $\bigsqcap_I S = [\max_{I\, j\in\Delta} a_j, \min_{I\, j\in\Delta} b_j] \sqsubseteq_I [a_i, b_i]$, proving $\bigsqcap_I S$ to be a lower bound of S
- Let ℓ be another lower bound of S. If $\ell = \bot$ then immediately $\ell \sqsubseteq_I \bigsqcap_I S$. Otherwise $\ell = [a_\ell, b_\ell]$ and $\forall i \in \Delta : [a_\ell, b_\ell] \sqsubseteq_I [a_i, b_i]$ so $a_i \leq a_\ell \leq b_\ell \leq b_i$ proving $\max_{I\, j\in\Delta} a_j \leq a_\ell \leq b_\ell \leq \min_{I\, j\in\Delta} b_j$ whence $[a_\ell, b_\ell] \sqsubseteq_I [\max_{I\, j\in\Delta} a_j, \min_{I\, j\in\Delta} b_j] = \bigsqcap_I S$, proving $\bigsqcap_I S$ to be the glb of S.

$\square$

Interval abstraction

- We have defined $\gamma_i \in I \mapsto \wp(\mathbb{I})$ as

$$\gamma_i(\bot) \stackrel{\text{def}}{=} \emptyset$$
$$\gamma_i([a, b]) \stackrel{\text{def}}{=} \{z \in \mathbb{I} \mid a \leq z \leq b\}$$

- Given $X \subseteq \mathbb{I}$, we define

$$\alpha_i(\emptyset) \stackrel{\text{def}}{=} \bot$$
$$\alpha_i(X) \stackrel{\text{def}}{=} [\min_{I\, x\in X} x, \min_{I\, x\in X} x], \qquad X \neq \emptyset$$

- We have the Galois connection:

$$\langle \wp(\mathbb{I}), \subseteq \rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle I, \sqsubseteq_I \rangle$$

PROOF. We prove that $\alpha_i(x) \sqsubseteq_I y \iff x \subseteq \gamma_i(y)$.

- If x is $\emptyset$ then $\alpha_i(\emptyset) \stackrel{\text{def}}{=} \bot \sqsubseteq_I y$ and $x = \emptyset \subseteq \gamma(y)$ is true for all $y \in I$.
- If y is $\bot$ then $\alpha_i(x) \sqsubseteq_I y$ implies $\alpha_i(x) = \bot$ whence $x = \emptyset$ by def. α_i and so $x = \emptyset \subseteq \emptyset = \gamma_i(\bot)$.
  Reciprocally, if $x \subseteq \gamma_i(y)$ then $x \subseteq \emptyset$ so $x = \emptyset$ proving $\alpha_i(x) = \bot \sqsubseteq_I y$ by def. $\sqsubseteq_I$
- If x is not $\emptyset$ and y is not $\bot$ then $y = [a, b]$ with $-\infty \leq a \leq b \leq +\infty$ by def. I. If $\alpha_i(x) \sqsubseteq_I y$ then $[\min_I x, \max_I x] \sqsubseteq_I [a, b]$ so $a \leq \min_I x \leq \max_I x \leq b$ by def. $\sqsubseteq_I$, probing $x \subseteq \{z \in \mathbb{I} \mid a \leq z \leq b\} = \gamma_i([a, b]) = \gamma_i(y)$.
  Reciprocally, $x \sqsubseteq_I y$ implies $x \subseteq \{z \in \mathbb{I} \mid a \leq z \leq b\}$ which implies $a \leq \min_I x \leq \max_I x \leq b$ that is $[\min_I x, \max_I x] \sqsubseteq_I [a, b]$ i.e. $\alpha(x) \sqsubseteq_I y$.

$\square$

The interval abstraction revisited (to ignore errors)

- Define $\alpha_e \stackrel{\text{def}}{=} \wp(\mathbb{I}_\Omega) \mapsto \wp(\mathbb{I})$ by $\alpha_e(x) = x \cap \mathbb{I}$. We have shown that $\langle \wp(\mathbb{I}_\Omega), \subseteq \rangle \underset{\alpha_e}{\overset{\gamma_e}{\leftrightarrows}} \langle \wp(\mathbb{I}_\Omega), \subseteq \rangle$ where $\gamma_e(y) = y \cup \{\Omega_a, \Omega_i\}$
- We have shown that $\langle \wp(\mathbb{I}), \subseteq \rangle \underset{\alpha_i}{\overset{\gamma_i}{\leftrightarrows}} \langle I, \sqsubseteq_I \rangle$
- By composition $\alpha_I = \alpha_i \circ \alpha_e$ and $\gamma_I = \gamma_e \circ \gamma_i$, we get:

$$\langle \wp(\mathbb{I}_\Omega), \sqsubseteq \rangle \underset{\alpha_I}{\overset{\gamma_I}{\leftrightarrows}} \langle I, \sqsubseteq_I \rangle$$

– By definition, we have immediately:

$$\gamma_I(\bot) = \{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\}$$
$$\gamma_I([a,b]) = \{x \in I \mid a \leq x \leq b\} \cup \{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\}$$
$$\alpha_I(X) = (\!|\; X \subseteq \{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\} \;?\; \bot \;:\; [\min_I x \setminus \{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\}, \max_I x \setminus \{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\}]$$

The reduced product of abstractions

If

– $\langle L,\ \sqsubseteq,\ \sqcup\rangle$ is a meet semilattice,

– $\langle L,\ M\rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle M_1,\ \leq_1\rangle$

– $\langle L,\ M\rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle M_2,\ \leq_2\rangle$

then their reduced product [4] is

$$\langle L,\ M\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M,\ \leq\rangle$$

where

– $\gamma(\langle x,\ y\rangle) \stackrel{\text{def}}{=} \gamma_1(x) \sqcap \gamma_2(y)$

– $\langle x,\ y\rangle \equiv \langle x',\ y'\rangle \iff \gamma(\langle x,\ y\rangle) = \gamma(\langle x',\ y'\rangle)$

– $M \stackrel{\text{def}}{=} (M_1 \times M_2)/_{\equiv}$ quotient

– $\alpha(x) \stackrel{\text{def}}{=} [\langle \alpha_1(x),\ \alpha_2(x)\rangle]_{\equiv}$ equivalence class

– $[\langle x,\ y\rangle]_{\equiv} \leq [\langle x',\ y'\rangle]_{\equiv} \iff \exists \langle x_1,\ y_1\rangle \in [\langle x,\ y\rangle]_{\equiv} : \exists \langle x_2,\ y_2\rangle \in [\langle x',\ y'\rangle]_{\equiv} : x_1 \leq_1 x_2 \wedge y_1 \leq_2 y_2$

– $\gamma([\langle x,\ y\rangle]_{\equiv}) = \gamma(\langle x,\ y\rangle) = \gamma_1(x) \sqcap \gamma_2(y)$

PROOF. – the definition of $\gamma([\langle x,\ y\rangle]_{\equiv})$ is obviously independent of the choice of the representant $\langle x,\ y\rangle$ of the equivalence class $[\langle x,\ y\rangle]_{\equiv} \stackrel{\text{def}}{=} \{\langle x_1,\ x_2\rangle \mid \gamma(\langle x,\ y\rangle) = \gamma(\langle x_1,\ x_2\rangle)\}$. This remark is also valid for the definition of $\leq$.

$\alpha(X) \leq [\langle x,\ y\rangle]_{\equiv}$

$\Longrightarrow [\langle \alpha_1(X),\ \alpha_2(X)\rangle]_{\equiv} \leq [\langle x,\ y\rangle]_{\equiv}$ ⟨def. α⟩

$\Longrightarrow \exists \langle x_1,\ y_1\rangle \in [\langle \alpha_1(X),\ \alpha_2(X)\rangle]_{\equiv} : \exists \langle x_2,\ y_2\rangle \in [\langle x,\ y\rangle]_{\equiv} : x_1 \leq_1 x_2 \wedge y_1 \leq_2 y_2$ ⟨def. $\leq$⟩

$\Longrightarrow \exists \langle x_1,\ y_1\rangle, \langle x_2,\ y_2\rangle : \gamma(\langle x_1,\ y_1\rangle) = \gamma(\langle \alpha_1(X),\ \alpha_2(X)\rangle) \wedge \gamma(\langle x_2,\ y_2\rangle) = \gamma(\langle x,\ y\rangle) \wedge x_1 \leq_1 x_2 \wedge y_1 \leq_2 y_2$ ⟨def. $[\langle x,\ y\rangle]_{\equiv}$⟩

$\Longrightarrow \exists \langle x_1,\ y_1\rangle, \langle x_2,\ y_2\rangle : \gamma_1(x_1) \sqcap \gamma_2(y_1) = \gamma_1 \circ \alpha_1(X) \sqcap \gamma_2 \circ \alpha_2(X) \wedge \gamma_1(x_2) = \gamma_1(x) \wedge \gamma_2(y_2) = \gamma_2(y) \wedge x_1 \leq_1 x_2 \wedge y_1 \leq_2 y_2$ ⟨def. γ⟩

$\Longrightarrow$ ⟨by monotony of γ so that $\gamma_1(x_1) \sqsubseteq \gamma_1(x_2) = \gamma_1(x)$ and $\gamma_2(y_1) \sqsubseteq \gamma_2(y_2) = \gamma_2(y)$ so that $\gamma_1(x_1) \sqcap \gamma_2(y_1) \sqsubseteq \gamma_1(x) \sqcap \gamma_2(y)$⟩

$\gamma_1 \circ \alpha_1(X) \sqcap \gamma_2 \circ \alpha_2(X) \sqsubseteq \gamma_1(x) \sqcap \gamma_2(y)$

$$\Longrightarrow \quad \wr \gamma_1 \circ \alpha_1 \text{ and } \gamma_2 \circ \alpha_2 \text{ are extensive so that } X \sqsubseteq \gamma_1 \circ \alpha_1(X) \sqcap \gamma_2 \circ \alpha_2(X) \text{ and transitivity} \wr$$

$$X \sqsubseteq \gamma_1(x) \sqcap \gamma_2(y)$$

$$\Longrightarrow X \sqsubseteq \gamma([\langle x,\ y\rangle]_\equiv) \qquad \wr \text{def. } \gamma \quad \text{Q.E.D.} \wr$$

Reciprocally:

$$X \sqsubseteq \gamma([\langle x,\ y\rangle]_\equiv)$$

$$\Longrightarrow X \sqsubseteq \gamma_1(x) \sqcap \gamma_2(y) \qquad \wr \text{def. } \gamma \wr$$

$$\Longrightarrow \alpha_1(X) \leq_1 x \wedge \alpha_2(X) \leq_2 y \qquad \wr \text{def. Galois connection} \wr$$

$$\Longrightarrow [\langle \alpha_1(X),\ \alpha_2(X)\rangle]_\equiv \leq [\langle x,\ y\rangle]_\equiv \qquad \wr \text{def. } \leq \wr$$

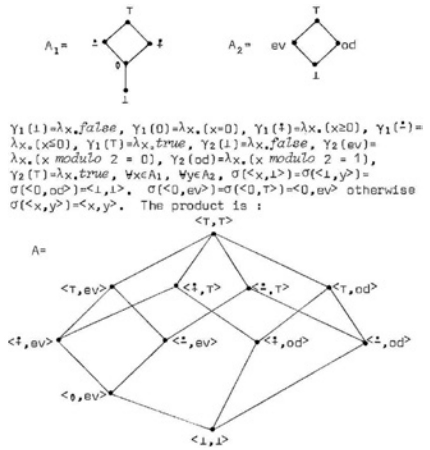
$$\Longrightarrow \alpha(X) \leq [\langle x,\ y\rangle]_\equiv \qquad \wr \text{def. } \alpha \quad \text{Q.E.D.} \wr$$

$\square$

Reference

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY, U.S.A.

Sign and parity



The error/interval abstraction

We define the interval and error abstraction as the reduced product of the interval and error abstractions:

$$\langle \wp(\mathbb{I}_\Omega),\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle I \times E,\ \sqsubseteq\rangle$$

where

- $\gamma(\langle i,\ e\rangle) \stackrel{\text{def}}{=} \gamma_I(i) \cup \gamma_E(e)$
- $\alpha(X) \stackrel{\text{def}}{=} \langle \alpha_I(X),\ \alpha_E(X)\rangle$
- $\langle i_1,\ e_1\rangle \sqsubseteq \langle i_2,\ e_2\rangle \stackrel{\text{def}}{=} i_1 \sqsubseteq_I i_2 \wedge e_1 \sqsubseteq_E e_2$

PROOF.

$$\langle i,\ e\rangle \equiv \langle i',\ e'\rangle$$

$$\Longleftrightarrow \gamma(\langle i,\ e\rangle) = \gamma(\langle i',\ e'\rangle) \qquad \wr \text{def. } \equiv \wr$$

$$\Longleftrightarrow \gamma_I(i) \cap \gamma_E(e) = \gamma_I(i') \cap \gamma_E(e') \qquad \wr \text{def. } \gamma \wr$$

$$\Longleftrightarrow \gamma_i(i) = \gamma_i(i') \wedge \gamma_E(e) = \gamma_E(e') \qquad \wr \text{def. } \gamma_I,\ \gamma_i \text{ and } \gamma_E \wr$$

$$\Longleftrightarrow i = i' \wedge e = e' \qquad \wr \gamma_i \text{ and } \gamma_E \text{ injective} \wr$$

$$\Longleftrightarrow \langle i,\ e\rangle = v\langle i',\ e'\rangle \qquad \wr \text{def. pairs} \wr$$

It follows that $\equiv$ is equality and so $[\langle i,\ e\rangle]_\equiv = \{\langle i,\ e\rangle\}$ whence $(I \times E)/_\equiv$ is $I \times E$ up to the isomorphism $\{\langle i,\ e\rangle\} \mapsto \langle i,\ e\rangle$. The definition of α and of the ordering $\sqsubseteq$ follows immediately from this remark. $\square$

The interval abstraction as the reduced product of the minimum and maximum abstractions

– We have seen that if $\langle S,\ \leq,\ -\infty,\ +\infty,\ \max,\ \min\rangle$ is a complete lattice, $\langle\wp(S),\ \subseteq\rangle \xleftrightarrow[\alpha_M]{\gamma_M} \langle S,\ \leq\rangle$ where $\alpha_M(X) = \max X$ and $\gamma_M(s) = \{x \in S \mid x \leq s\}$

– By duality, $\langle\wp(S),\ \subseteq\rangle \xleftrightarrow[\alpha_m]{\gamma_m} \langle S,\ \geq\rangle$ where $\alpha_m(X) = \min X$ and $\gamma_m(s) = \{x \in S \mid x \geq s\}$

– Let us consider the reduced product of these two abstractions

- The classes $[\langle a,\ b\rangle]_\equiv$ where $a \leq b$ is $\{\langle a,\ b\rangle\}$ whence can be represented as $\langle a,\ b\rangle \in S \times S$
- The classes $[\langle a,\ b\rangle]_\equiv$ where $a > b$ are $\{\emptyset\}$ whence can be represented by some new element $\bot \notin S \times S$

– The reduced product is now, up to an isomorphism:

$$\langle\wp(S),\ \subseteq\rangle \xleftrightarrow[\alpha]{\gamma} \langle\{\langle a,\ b\rangle \mid a \leq b\} \cup \{\bot\},\ \sqsubseteq\rangle$$

Proof. Trivial. □

Implementation of the complete lattice of intervals — (1) Abstract properties

```
(* avalues.ml *)
open Values
(* abstraction of sets of machine integers by intervals *)
(* complete lattice *)
(*                 *)
(* ABSTRACT VALUES  *)
(*                 *)
type t = int * int
(* gamma (a,b)                                                  *)
(*   = [a,b] U {_O_(a), _O_(i)} when min_int <= a <= b <= max_int *)
(*   = {_O_(a), _O_(i)}         when a = max_int > min_int = b    *)
(* infimum: alpha({})                                           *)
let bottom = (max_int, min_int)
```



```
(* infimum: bot () = alpha({}) *)
let bot () = bottom
(* isbottom a = (a = bot ()) *)
let isbottom (x, y) = y < x
(* isbotempty () = gamma(bot ()) = {}?                            *)
let isbotempty () = false (* gamma([max_int, min_int]) =          *)
                          (* {_O_(a), _O_(i)} <> emptyset         *)
(* uninitialization: initerr () = alpha({_O_i}) *)
let initerr () = bottom
(* supremum: top () = alpha({_O_i, _O_a} U [min_int,max_int]) *)
let top () =  (min_int, max_int)
(* least upper bound join: p q = alpha(gamma(p) U gamma(q)) *)
let min x y = if (x <= y) then x else y
let max x y = if (x < y) then y else x
let join (v,w) (x,y) = ((min v x), (max w y))
(* greatest lower bound meet p q = alpha(gamma(p) cap gamma(q)) *)
let meet (v,w) (x,y) = ((max v x), (min w y))
```

```
(* approximation ordering: leq p q = gamma(p) subseteq gamma(q) *)
let leq (v,w) (x,y) = (isbottom (v, w)) || ((x <= v) && (w <= y))
(* equality: eq p q = gamma(p) = gamma(q) *)
let eq u v = (u = v)
(* errors = alpha({_O_i, _O_a}) *)
let errors = bottom
(* included in errors?: in_errors p = gamma(p) subseteq {_O_i, _O_a} *)
let in_errors (x, y) = isbottom (x, y)
(* printing *)
let print_int x =
   if x = min_int then print_string "min_int"
else if x = - max_int then print_string "-max_int"
else if x = max_int then print_string "max_int"
else print_int x
let print (x, y) = if (isbottom (x, y)) then print_string "[]" else
   (print_string "["; print_int x; print_string ","; print_int y;
    print_string "]")
```


Design of the abstract transformers (samples)

Design of the abstract transformers: forward integer constant

f_NAT $s = \alpha(\{$machine_int_of_string$s)\})$ where:

```
(* values.ml *)
type error_type = INITIALIZATION | ARITHMETIC
type machine_int = ERROR_NAT of error_type | NAT of int
type machine_bool = ERROR_BOOL of error_type | BOOLEAN of bool
exception Incorrect_Nat of string
let rec int_of_intstring i s =
  let l = (String.length s) in
     if l = 0 then (NAT i)
     else let v = (10 * i) + (int_of_string (String.sub s 0 1)) in
            if v<i then (* overflow *)
              (ERROR_NAT ARITHMETIC)
            else
              int_of_intstring v (String.sub s 1 (l-1))
```



```
let machine_int_of_string s =
  int_of_intstring 0 s
```

The lexer (lexer.mll) is:

```
rule token = parse
  [' ' '\t' '\n' '\r'] { token lexbuf }
| ( '%' [^'%']* '%')   { token lexbuf }
| ['0'-'9']+           { (T_NAT (Lexing.lexeme lexbuf)) }
| '('                  { T_LPAR }
...
```

The parser (parser.mly) is:

```
...
%token <string> T_NAT
...
```

```
n_Aexp:
  T_RANDOM                { Abstract_Syntax.RANDOM                }
| T_NAT                   { (Abstract_Syntax.NAT $1)              }
| T_VAR { (Abstract_Syntax.VAR (Symbol_Table.add_symb_table $1)) }
| n_Aexp  T_PLUS   n_Aexp  { (Abstract_Syntax.PLUS  ($1, $3))     }
| n_Aexp  T_MINUS  n_Aexp  { (Abstract_Syntax.MINUS ($1, $3))     }
| n_Aexp  T_TIMES  n_Aexp  { (Abstract_Syntax.TIMES ($1, $3))     }
| n_Aexp  T_DIV    n_Aexp  { (Abstract_Syntax.DIV   ($1, $3))     }
| n_Aexp  T_MOD    n_Aexp  { (Abstract_Syntax.MOD   ($1, $3))     }
| T_PLUS  n_Aexp %prec T_UPLUS  { (Abstract_Syntax.UPLUS  $2)     }
| T_MINUS n_Aexp %prec T_UMINUS { (Abstract_Syntax.UMINUS $2)     }
| T_LPAR  n_Aexp  T_RPAR       { $2              }
;
...
```

This ensures that s is a finite non-empty string of digits:

$s = d_n d_{n-1} \ldots d_1 d_0$ where $n \geq 0$, $d_i \in [0, 9]$, $i = 1, \ldots, n$

Recall that we have defined (in decimal notation):

$$\begin{aligned} \underline{s} &\stackrel{\text{def}}{=} d_n.10^n + d_{n-1}.10^{n-1} + \ldots + d_1.10^1 + d_0.10^0 \\ &= d_n.10^n + d_{n-1}.10^{n-1} + \ldots + d_1.10 + d_0 \end{aligned}$$

Lemma. If i is a non-negative integer and s a string of digits, "$d_n \ldots d_0$" (which may be empty) then

$$\begin{aligned} &\texttt{int\_of\_intstring}\ i\ s \\ &= (\texttt{NAT}\ i) && \text{if } n = -1\ (|s| = 0) \\ &= (\texttt{NAT}\ 10^{n+1} \times i + \underline{s}) && \text{if } 10^{n+1} \times i + \underline{s} \leq \texttt{max\_int} \\ &= (\texttt{ERROR\_NAT ARITHMETIC}) && \text{otherwise} \end{aligned}$$

■

Proof. By recurrence on n:

- if $n = -1$ that is $|s| = 0$ whence $(\texttt{String.length s}) = 0$, then by symbolic execution, we get $(\texttt{NAT}\ i)$ as requested
- if $n = 0$ then, by symbolic execution

$$\begin{aligned} &\texttt{int\_of\_intstring}\ i\ \text{"}d_0\text{"} \\ = \ &\text{let } v = (10 \otimes i) \oplus (\texttt{int\_of\_string}\ \text{"}d_0\text{"}) \text{ in} \\ &\llparenthesis\, v < i \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ \texttt{int\_of\_intstring}\ v\ \text{""} \rrparenthesis \\ = \ &\llparenthesis\, ((10 \otimes i) \oplus d_0) < i \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ (\texttt{NAT}\ (10 \otimes i) \oplus d_0) \rrparenthesis \end{aligned}$$

$\wr$Notice that $\otimes$ and $\oplus$ are modulo arithmetic in $[-\texttt{max\_int} - 1, \texttt{max\_int}]$ where $\texttt{max\_int} > 9$ and so $((10 \otimes i) \oplus d_0) < i \iff ((10 \otimes i) \oplus d_0) > \texttt{max\_int}$ since i, $d_0 > 0$. Moreover $(10 \otimes i) \oplus d_0) = (10 \times i) + d_0)$ when $((10 \times i) + d_0) \leq \texttt{max\_int}$. Finally $d_O = \underline{\text{"}d_0\text{"}} = \underline{s}$ so that we get:$\int$

$$\begin{aligned} = \ &\llparenthesis\, ((10^1 \times i) + \underline{s}) > \texttt{max\_int} \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ (\texttt{NAT}\ (10^1 \times i) + \underline{s}) \rrparenthesis \qquad \text{Q.E.D.} \end{aligned}$$

- if $n > 0$ then

$$\begin{aligned} &\texttt{int\_of\_intstring}\ i\ \text{"}d_n d_{n-1} \ldots d_0\text{"} \\ = \ &\text{let } v = (10 \otimes i) \oplus (\texttt{int\_of\_string}\ \text{"}d_n\text{"}) \text{ in} \\ &\llparenthesis\, v < i \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ \texttt{int\_of\_intstring}\ v\ \text{"}d_{n-1} \ldots d_0\text{"} \rrparenthesis \\ = \ &\text{let } v = (10 \otimes i) \oplus d_n \text{ in} \\ &\llparenthesis\, v < i \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ \texttt{int\_of\_intstring}\ v\ \text{"}d_{n-1} \ldots d_0\text{"} \rrparenthesis \end{aligned}$$

$\wr$Notice that $((10 \otimes i) \oplus d_n) < i$ iff $((10 \times i) + d_n) > \texttt{max\_int}$ since i, $d_n \geq 0$$\int$

$$\begin{aligned} = \ &\llparenthesis\, (10 \times i) + d_n > \texttt{max\_int} \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \ :\ \texttt{int\_of\_intstring}\ ((10 \times i) + d_n)\ \text{"}d_{n-1} \ldots d_0\text{"} \rrparenthesis \\ = \ &\quad \wr\text{by induction hypothesis}\int \\ = \ &\llparenthesis\, (10 \times i) + d_n > \texttt{max\_int} \ ?\ (\texttt{ERROR\_NAT ARITHMETIC}) \parallel 10^n \times ((10 \times i) + d_n) + \underline{\text{"}d_{n-1} \ldots d_0\text{"}} \leq \texttt{max\_int} \ ?\ (\texttt{NAT}\ 10^n \times ((10 \times i) + d_n) + \underline{\text{"}d_{n-1} \ldots d_0\text{"}}) \ :\ (\texttt{ERROR\_NAT ARITHMETIC}) \rrparenthesis \end{aligned}$$

$\wr$Notice that $10^n \times ((10 \times i) + d_n) + \underline{\text{"}d_{n-1} \ldots d_0\text{"}} = 10^{n+1}.i + 10^n.d_n + {} + d_{n-1}.10^{n-1} + \ldots + d_1.10 + d_0 = 10^{n+1}.i + \underline{\text{"}d_n d_{n-1} \ldots d_0\text{"}}$ $\int$

$= \llparenthesis (10 \times i) + d_n > \texttt{max\_int}\ ?\ (\texttt{ERROR\_NAT ARITHMETIC})\ \|\ 10^{n+1}.i + \underline{"d_n d_{n-1} \ldots d_0"} \leq \texttt{max\_int}\ ?\ (\texttt{NAT}\ \ 10^{n+1}.i + \underline{"d_n d_{n-1} \ldots d_0"})\ \vdots\ (\texttt{ERROR\_NAT ARITHMETIC})\rrparenthesis$

$\wr$Observe that $(10 \times i) + d_n > \texttt{max_int} \Longrightarrow 10^{n+1}.i + \underline{"d_n d_{n-1} \ldots d_0"} > \texttt{max_int}$$\wr$

$= \llparenthesis 10^{n+1}.i + \underline{"d_n d_{n-1} \ldots d_0"} \leq \texttt{max\_int}\ ?\ 10^{n+1}.i + \underline{"d_n d_{n-1} \ldots d_0"}\ \vdots\ (\texttt{ERROR\_NAT ARITHMETIC})\ \ \text{Q.E.D.}\rrparenthesis$

□

LEMMA. Let $s = "d_n \ldots d_0"$ where $n \geq 0$. Then

$$\begin{array}{ll} \texttt{machine\_int\_of\_string}\ s & \\ \quad = (\texttt{NAT}\ \underline{s}) & \text{if } \underline{s} \leq \texttt{max\_int} \\ \quad = (\texttt{ERROR\_NAT ARITHMETIC}) & \text{otherwise} \end{array}$$

■

PROOF. By symbolic execution:

$\texttt{f\_NAT}\ s$

$\stackrel{\text{def}}{=} \alpha_i(\{(\texttt{NAT}\ \underline{s})\})$

$= \langle \underline{s},\ \underline{s} \rangle$

Otherwise $\underline{s} > \texttt{max\_int}$ and then

$\texttt{f\_NAT}\ s$

$\stackrel{\text{def}}{=} \alpha_i(\{(\texttt{ERROR\_NAT ARITHMETIC})\})$

$= \perp$

□

$\texttt{machine\_int\_of\_string}\ s$

$= \texttt{int\_of\_intstring}\ 0\ s$

$\wr$by previous lemma$\wr$

$= (\texttt{NAT}\ \underline{s})$ $\qquad\wr$if $\underline{s} \leq \texttt{max_int}$$\wr$

$= (\texttt{ERROR\_NAT ARITHMETIC})$ $\qquad\wr$otherwise$\wr$

□

We now have:

THEOREM.

$$\begin{array}{lll} \texttt{f\_NAT}\ s & = [\underline{s}, ; \underline{s}] & \text{if } \underline{s} \leq \texttt{max\_int} \\ & = \perp & \text{otherwise} \end{array}$$

■

PROOF. If $\underline{s} \leq \texttt{max\_int}$ then

The implementation follows (the impossible case (ERROR_NAT INITIALIZATION) could have been signalled as a design error by the analyzer):

```
(* f_NAT s = alpha({(machine_int_of_string s)})        *)
let f_NAT s =
   match (machine_int_of_string s) with
   | (ERROR_NAT INITIALIZATION) -> bottom
   | (ERROR_NAT ARITHMETIC) -> bottom
   | (NAT i) -> (i,i)
```

Design of the abstract transformers: backward integer constant

– The backward collecting semantics of arithmetic expressions was defined in lecture (17) as:

$$\mathrm{Baexp}[\![A]\!](R)P \stackrel{\mathrm{def}}{=} \{\rho \in R \mid \exists i \in P \cap \mathbb{I} : \rho \vdash A \mapsto i\} \qquad (2)$$

and their backward abstract interpretation was defined as:

$$\mathrm{Baexp}^{\triangleleft}[\![A]\!] \ \ddot{\sqsupseteq}\ \alpha^{\triangleleft}(\mathrm{Baexp}[\![A]\!]) \qquad (3)$$

and we have proved that:

$$\mathrm{Baexp}^{\triangleleft}[\![A]\!](\lambda \mathrm{Y} \cdot \bot)p \stackrel{\mathrm{def}}{=} \lambda \mathrm{Y} \cdot \bot \qquad \text{if } \gamma(\bot) = \emptyset \qquad (4)$$

$$\mathrm{Baexp}^{\triangleleft}[\![\mathrm{n}]\!](r)p \stackrel{\mathrm{def}}{=} (\!|\, \mathrm{n}^{\triangleleft}(p) \mathrel{?} r \mathrel{\vdots} \lambda \mathrm{Y} \cdot \bot \,|\!) \qquad (5)$$

where:

$$\mathrm{n}^{\triangleleft}(p) \stackrel{\mathrm{def}}{=} (\underline{\mathrm{n}} \in \gamma(p) \cap \mathbb{I}) \qquad (6)$$

– Therefore, for the implementation, we define [6]

[6] For short, up to a machine representation (NAT i) for i, (ERROR_NAT INITIALIZATION) for Ω_{i} and (ERROR_NAT ARITHMETIC) for Ω_{a}.

b_NAT s v

$\stackrel{\mathrm{def}}{=}$ (machine_int_of_string s) $\in \gamma(v) \cap [$min_int, max_int$]$

$= \text{let } v = [a, b] \text{ in } (\!|\, \underline{s} > \texttt{max\_int} \mathrel{?} \mathit{ff} \mathrel{\vdots} a \leq \underline{s} \leq b \,|\!)$

PROOF. Assume that $v = [a, b]$ where $b < a$ for bottom. We have:

b_NAT s $[a, b]$

$=$ (machine_int_of_string s) $\in \gamma([a, b]) \cap [min_int, max_int]$

  ⟨by lemma on machine_int_of_string⟩

$= (\!|\, \underline{s} \leq \texttt{max\_int} \mathrel{?} (\texttt{INT}\ s) \mathrel{\vdots} (\texttt{ERROR\_NAT ARITHMETIC}) \,|\!) \in \gamma([a, b]) \cap [\texttt{min\_int}, \texttt{max\_int}]$

$= (\!|\, \underline{s} \leq \texttt{max\_int} \mathrel{?} (\texttt{INT}\ s) \mathrel{\vdots} (\texttt{ERROR\_NAT ARITHMETIC}) \,|\!) \in ((\!|\, a \leq b \mathrel{?} ([a, b] \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}) \cap [\texttt{min\_int}, \texttt{max\_int}]) \mathrel{\vdots} (\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}) \cap [\texttt{min\_int}, \texttt{max\_int}]) \,|\!))$

$= (\!|\, \underline{s} \leq \texttt{max\_int} \mathrel{?} (\texttt{INT}\ s) \mathrel{\vdots} (\texttt{ERROR\_NAT ARITHMETIC}) \,|\!) \in (\!|\, a \leq b \mathrel{?} [a, b] \mathrel{\vdots} \emptyset \,|\!)$

$= (\!|\, \underline{s} > \texttt{max\_int} \mathrel{?} \mathit{ff} \parallel a \leq b \mathrel{?} a \leq \underline{s} \leq b \mathrel{\vdots} \mathit{ff} \,|\!)$

$= (\!|\, \underline{s} > \texttt{max\_int} \mathrel{?} \mathit{ff} \mathrel{\vdots} a \leq \underline{s} \leq b \,|\!)$

□

which directly yields the implementation:

```
(* b_NAT s v = (machine_int_of_string s) in
(¨                              gamma(v) cap [min_int, max_int]? *)
let b_NAT s (a, b) =
match (machine_int_of_string s) with
| (ERROR_NAT INITIALIZATION) -> false
| (ERROR_NAT ARITHMETIC) -> false
| (NAT i) -> (a <= i) && (i <= b)
```

Design of the abstract transformers: forward integer addition

– The forward abstract semantics of a binary operator is (from lecture 16):

$$\mathrm{Faexp}^{\triangleright}[\![A_1 \ \mathtt{b} \ A_2]\!]r \stackrel{\mathrm{def}}{=} \mathtt{b}^{\triangleright}(\mathrm{Faexp}^{\triangleright}[\![A_1]\!]r, \mathrm{Faexp}^{\triangleright}[\![A_2]\!]r)$$

where:

$$\mathtt{b}^{\triangleright}(p_1, p_2) \sqsupseteq \alpha(\{v_1 \ \underline{\mathtt{b}} \ v_2 \mid v_1 \in \gamma(p_1) \wedge v_2 \in \gamma(p_2)\}) \qquad (7)$$

– Therefore, up to the computer representation

$$\begin{cases} \bot \longrightarrow (\mathtt{max\_int}, \mathtt{min\_int}) \\ \Omega_{\mathrm{i}} \longrightarrow (\mathtt{ERROR\_NAT\ INITIALIZATION}) \\ \Omega_{\mathrm{a}} \longrightarrow (\mathtt{ERROR\_NAT\ ARITHMETIC}) \\ [a, b] \longrightarrow (a, b) \end{cases}$$

we define

$$\mathtt{f\_PLUS}\ x\ y \sqsupseteq \alpha(\{\mathtt{machine\_binary\_plus}\ i\ j \mid i \in \gamma(x) \wedge j \in \gamma(y)\})$$

where

$$\begin{array}{ll} \gamma([u, v]) = \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} & \text{if } v < u \\ \gamma([u, v]) = [u, v] \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} & \text{if } v \geq u \end{array}$$

and

```
let add_int x y =
  if (x >= 0) & (y >= 0) then
     (if x <= (max_int - y) then (NAT (x+y)) else (ERROR_NAT ARITHMETIC))
  else if (x <= 0) & (y <= 0) then
     (if (min_int - x) <= y then (NAT (x+y)) else (ERROR_NAT ARITHMETIC))
  else (NAT (x+y))

let machine_binary_plus a b = match a with
| ERROR_NAT e -> (ERROR_NAT e)
| NAT a'  -> match b with
  | ERROR_NAT e' -> (ERROR_NAT e')
  | NAT b' -> (add_int a' b')
```


Lemma.

$$\begin{array}{ll} \mathtt{add\_int}\ x\ y = x + y & \text{if } \mathtt{min\_int} \leq x + y \leq \mathtt{max\_int} \\ \qquad\qquad = \Omega_{\mathrm{a}} & \text{otherwise} \end{array}$$

■

Proof. By cases

– if $x \geq 0 \wedge y \geq 0$ then, by symbolic execution, if $O \leq x + y \leq \mathtt{max\_int}$ (or equivalently $x \leq \mathtt{max\_int} - y$, which avoids overflows) then $\mathtt{add\_int}\ x\ y = x + y$ else $x + y > q\mathtt{max\_int}$ and then $\mathtt{add\_int}\ x\ y = \Omega_{\mathrm{a}}$

– if $x \leq 0 \wedge y \leq 0$ then, by symbolic execution, if $\mathtt{min\_int} \leq x + y \leq 0$ (or equivalently $\mathtt{min\_int} - x \leq y$, which avoids overflows) then $\mathtt{add\_int}\ x\ y = x + y$ else $x + y < \mathtt{min\_int}$ and $\mathtt{add\_int}\ x\ y = \Omega_{\mathrm{a}}$

– Otherwise x and y are of opposite signs. Assume $x \in [\mathtt{max\_int} - 1, 0]$ and $y \in [0, \mathtt{max\_int}]$ (the other case beign symmetric). We have $x + y \in [-\mathtt{max\_int} - 1, \mathtt{max\_int}] = [\mathtt{min\_int}, \mathtt{max\_int}]$ and $\mathtt{add\_int}\ x\ y = x + y$ as required.

□

Notice that the proof implies the absence of overflows when computing `add_int` x y and so the modulo arithmetic of OCaml can be used in place of the mathematical arithmetic operations

THEOREM.

$$
\begin{array}{rcll}
\mathtt{f\_PLUS}\ \bot\ b & = & \bot & \\
\mathtt{f\_PLUS}\ a\ \bot & = & \bot & \\
\mathtt{f\_PLUS}\ (u,v)\ (w,x) & = & \bot & \text{if } u+w > \mathtt{max\_int} \\
 & = & \bot & \text{if } v+x < \mathtt{min\_int} \\
 & = & (\mathtt{min\_int}, \mathtt{max\_int}) & \text{if } u+w < \mathtt{min\_int} \wedge v+x > \mathtt{max\_int} \\
 & = & (\mathtt{min\_int}, u+w) & \text{if } u+w < \mathtt{min\_int} \wedge v+x \leq \mathtt{max\_int} \\
 & = & (u+w, \mathtt{max\_int}) & \text{if } u+w \geq \mathtt{min\_int} \wedge v+x > \mathtt{max\_int} \\
 & = & (u+w, v+x) & \text{if } u+w \geq \mathtt{min\_int} \wedge v+x \leq \mathtt{max\_int}
\end{array}
$$

■

PROOF. For the definition of `f_PLUS` a b, we proceed by cases

– if a is bottom, that is $a = (u, v)$ with $v < u$ so that (`isbottom` (u, v)) holds, we have

$$
\begin{array}{rl}
 & \mathtt{f\_PLUS}\ \bot\ b \\
= & \alpha(\{\mathtt{machine\_binary\_plus}\ i\ j \mid i \in \gamma(\bot) \wedge j \in \gamma(b)\})
\end{array}
$$

$$
\begin{array}{rl}
= & \alpha(\{\mathtt{machine\_binary\_plus}\ i\ j \mid i \in \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \wedge j \in \gamma(b)\}) \\
= & \alpha(\{\Omega_{\mathrm{a}} \mid j \in \gamma(b)\} \cup \{\Omega_{\mathrm{i}} \mid j \in \gamma(b)\}) \\
 & \quad \wr \gamma(b) \text{ is not empty} \int \\
= & \alpha(\{\Omega_{\mathrm{a}}, \Omega_{\mathrm{i}}\}) \\
= & \bot
\end{array}
$$

– The case of $b = \bot$ is handled in the same way, so that `f_PLUS` $a\ \bot = \bot$.

– Otherwise $a = (u, v) \neq \bot \wedge b = (w, x) \neq \bot$ in which case, we have $u \leq v$ and $w \leq x$. We calculate

$$
\begin{array}{rl}
 & \mathtt{f\_PLUS}\ (u,v)\ (w,x) \\
= & \alpha(\{\mathtt{machine\_binary\_plus}\ i\ j \mid i \in \gamma((u,v)) \wedge j \in \gamma((w,x))\}) \\
= & \alpha(\{\mathtt{machine\_binary\_plus}\ i\ j \mid i \in \{i' \mid u \leq i' \leq v\} \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \wedge j \in \{j' \mid \\
 & w \leq j' \leq x\}\} \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\})
\end{array}
$$

$$
\begin{array}{rl}
 & \wr \text{by symbolic execution of } \mathtt{machine\_binary\_plus} \text{ in cases } i \in \\
 & \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\},\ j \in \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\},\ i+j \in [\mathtt{min\_int}, \mathtt{max\_int}] \wedge i+j \notin \\
 & [\mathtt{min\_int}, \mathtt{max\_int}] \int \\
= & \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i+j \mid \mathtt{min\_int} \leq i+j \leq \mathtt{max\_int} \wedge u \leq i \leq v \wedge w \leq j \leq x\}) \\
= & \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i+j \mid \max(\mathtt{min\_int}, u+w) \leq i+j \leq \min(\mathtt{max\_int}, v+x)\})
\end{array}
$$

We proceed by cases:

– If $u+w > \mathtt{max\_int}$, then $\max(\mathtt{min\_int}, u+w) = u+w$ and $\min(\mathtt{max\_int}, v+x) = \mathtt{max\_int}$ if $v + x \geq u + w$ so in this case

$$
\begin{array}{rl}
= & \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i+j \mid \mathtt{max\_int} < i+j \leq \mathtt{max\_int}\}) \\
= & \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}) \\
= & \bot
\end{array}
$$

– If $v + x < \mathtt{min\_int}$, then $u + w \leq v + x < \mathtt{min\_int} < \mathtt{max\_int}$ so $\max(\mathtt{min\_int}, u+w) = \mathtt{min\_int}$ and $\min(\mathtt{max\_int}, v+x) = v+x$, so that in this case

$= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i + j \mid \mathtt{min\_int} < i + j \leq v + x < \mathtt{min\_int}\})$
$= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\})$
$= \perp$

– Otherwise, we have $u + w \leq v + x$, $u + w \leq \mathtt{max\_int}$ and $\mathtt{min\_int} \leq v + x$. There remain four cases:
– if $u + w < \mathtt{min\_int}$ then $\max(\mathtt{min\_int}, u + w) = \mathtt{min\_int}$ with two subcases:
  - if $v + x > \mathtt{max\_int}$ then $\min(\mathtt{max\_int}, v + x) = \mathtt{max\_int}$ so that in that case:

    $= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i + j \mid \mathtt{min\_int} \leq i + j \leq \mathtt{max\_int}\})$
    $= (\mathtt{min\_int}, \mathtt{max\_int})$
  - otherwise $v + x \leq \mathtt{max\_int}$ and then $\min(\mathtt{max\_int}, v + x) = v + x$ so that in that case:

    $= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i + j \mid \mathtt{min\_int} \leq i + j \leq u + w \leq \mathtt{max\_int}\})$

    $= (\mathtt{min\_int}, u + w)$

– otherwise $u + w \geq \mathtt{min\_int}$ and so $\max(\mathtt{min\_int}, u + w) = u + w$, with two subcases, as above:
  - if $v + x > \mathtt{max\_int}$ then $\min(\mathtt{max\_int}, v + x) = \mathtt{max\_int}$ so that in that case

    $= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i + j \mid \mathtt{min\_int} \leq u + w \leq i + j \leq \mathtt{max\_int}\})$
    $= (u + w, \mathtt{max\_int})$
  - otherwise $v + x \leq \mathtt{max\_int}$ then $\min(\mathtt{max\_int}, v + x) = v + x$ so that in that case

    $= \alpha(\{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\} \cup \{i + j \mid \mathtt{min\_int} \leq u + w \leq i + j \leq v + x \leq \mathtt{max\_int}\})$
    $= (u + w, v + x)$

$\square$

Observe that all sums are in the $[\mathtt{min\_int}, \mathtt{max\_int}]$ interval whence produce no overflow and can be computed with OCaml modulo arithmetic.

The only potential problem are the test $x + y < \mathtt{min\_int} \land x + y > \mathtt{max\_int}$ which can be easily proved to be equivalent to the following functions which produce no overflow whence can be implemented with modulo arithmetic:

```
let is_sum_lt_min_int x y =
(* x + y < min_int *)
if (x < 0) & (y < 0) then
(x < min_int - y)
else false

let is_sum_gt_max_int x y =
(* x + y > max_int *)
if (x > 0) && (y > 0) then
(x > max_int - y)
else false
```

From the calculational derivation of the definition of f_PLUS as shown above, we immediately obtain the following implementation, by just considering all possible cases:

```
(* f_BINARITH a b = alpha({ (machine_binary_binarith i j) |      *)
(*                          i in gamma(a) /\ j \in gamma(b)} *)
let f_PLUS (a, b) (c, d) =
  if (isbottom (a, b)) || (isbottom (c, d)) then bottom
  else if (is_sum_gt_max_int a c) then bottom
  else if (is_sum_lt_min_int b d) then bottom
  else let lb = if (is_sum_lt_min_int a c) then min_int else a + c
       and ub = if (is_sum_gt_max_int b d) then max_int else b + d
       in (lb, ub)
```


Design of the abstract transformers: backward integer addition

– The generic backward/bottom-up non-relational abstract semantics of arithmetic expressions was shown to be of the form

$$\mathrm{Baexp}^{\triangleleft}[\![A_1 \mathrm{~b~} A_2]\!](r)p \stackrel{\mathrm{def}}{=} \text{let } \langle p_1,\ p_2\rangle = \mathrm{b}^{\triangleleft}(\mathrm{Faexp}^{\triangleright}[\![A_1]\!]r, \mathrm{Faexp}^{\triangleright}[\![A_2]\!]r, p) \text{ in } \mathrm{Baexp}^{\triangleleft}[\![A_1]\!](r)p_1 \mathbin{\dot\sqcap} \mathrm{Baexp}^{\triangleleft}[\![A_2]\!](r)p_2 \quad (8)$$

where

$$\mathrm{b}^{\triangleleft}(q_1, q_2, p) \sqsupseteq^2 \alpha^2(\{\langle i_1,\ i_2\rangle \in \gamma^2(\langle q_1,\ q_2\rangle) \mid i_1 \underline{\mathrm{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\}) \quad (9)$$

– We consider the case of the binary addition $+$, up to the encoding

$$\begin{cases} \bot \to (\mathtt{max\_int}, \mathtt{min\_int}) \\ \Omega_{\mathrm{i}} \to (\mathtt{ERROR\_NAT\ INITIALIZATION}) \\ \Omega_{\mathrm{a}} \to (\mathtt{ERROR\_NAT\ ARITHMETIC}) \\ [a, b] \to (a, b) \end{cases}$$

– Recall that we have


```
(* gamma (a,b)                                                   *)
(*   = [a,b] U {_O_(a), _O_(i)} when min_int <= a <= b <= max_int *)
(*   = {_O_(a), _O_(i)}         when a = max_int > min_int = b    *)
(* infimum: alpha({})                                            *)
let bottom = (max_int, min_int)
(* infimum: bot () = alpha({}) *)
let bot () = bottom
(* isbottom a = (a = bot) *)
let isbottom (x, y) = y < x
(* isbotempty () = gamma(bot ()) = {}?                           *)
let isbotempty () = false (* gamma([max_int, min_int]) =         *)
                          (* {_O_(a), _O_(i)} <> emptyset        *)
(* errors = alpha({_O_i, _O_a}) *)
let errors = bottom
(* included in errors?: in_errors p = gamma(p) subseteq {_O_i, _O_a} *)
let in_errors (x, y) = isbottom (x, y)
```

– We define

$$\mathtt{b\_PLUS}\ q_1\ q_2\ p \stackrel{\text{def}}{=} \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma(q_1) \wedge i_2 \in \gamma(q_2) \wedge (\mathtt{machine\_binary\_plus}\ i_1\ i_2) \in \gamma(p) \cap \mathbb{I}\})$$

We have $q_1 = (a, b)$, $q_2 = (c, d)$ and $p = (e, f)$ with $(x, y) = \bot$ (bottom) whenever $y < x$.

THEOREM.

$$\mathtt{b\_PLUS}\ q_1\ q_2\ p = \bot \qquad \text{if } q_1,\ q_2 \text{ or } p = \bot$$

$$\begin{array}{l}\mathtt{b\_PLUS}\ (a, b)\ (c, d)\ (e, f) = \\ \quad \text{let } \ell_1 = \max(a, (\!|\, e - d < \mathtt{min\_int}\ ?\ \mathtt{min\_int} : e - d\,|\!))\ \text{in} \\ \quad \text{and } u_1 = \min(b, (\!|\, f - c > \mathtt{max\_int}\ ?\ \mathtt{max\_int} : f - c\,|\!))\ \text{in} \\ \quad \text{and } \ell_2 = \max(c, (\!|\, e - b < \mathtt{min\_int}\ ?\ \mathtt{min\_int} : e - b\,|\!))\ \text{in}) \\ \quad \text{and } u_2 = \min(d, (\!|\, f - a > \mathtt{max\_int}\ ?\ \mathtt{max\_int} : f - a\,|\!))\ \text{in} \\ \quad \text{and } q_1' = (\!|\, \ell_1 \le u_1\ ?\ [\ell_1, u_1] : \bot\,|\!)\ \text{in} \\ \quad \text{and } q_2' = (\!|\, \ell_2 \le u_2\ ?\ [\ell_2, u_2] : \bot\,|\!)\ \text{in}\ \langle q_1',\ q_2'\rangle \end{array}$$

■

PROOF. We first consider the cases of bottom arguments

– If $q_1 = \bot$, then $i_1 \in \gamma(q_1) = \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\}$ so, by definition of (machine_binary_plus (in values.ml):

```
let machine_binary_plus a b = match a with
| ERROR_NAT e -> (ERROR_NAT e)
| NAT a'  -> match b with
  | ERROR_NAT e' -> (ERROR_NAT e')
  | NAT b' -> (add_int a' b')
```

we have $(\mathtt{machine\_binary\_plus}\ i_1\ i_2 = i_1 \notin \mathbb{I}$ so $i_1 \notin \gamma(p) \cap \mathbb{I}$. In that case the result is therefore $\alpha^2(\emptyset) = \langle \bot,\ \bot\rangle$.

– If $q_2 = \bot$, then the same reasoning yields $\langle \bot,\ \bot\rangle$.

– If $p = \bot$ then $\gamma(p) \cap \mathbb{I} = \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\} \cap \mathbb{I} = \emptyset$ and so, once again, the result is $\alpha^2(\emptyset) = \langle \bot,\ \bot\rangle$.

– In the remaining cases, none of $q_1 = (a, b)$, $q_2 = (c, d)$ and $p = (e, f)$ is $\bot$ so that we can assume $a \le b$, $c \le d$ and $e \le f$. In that case we have:

$$\begin{array}{rl} & \mathtt{b\_PLUS}\ (a, b)\ (c, d)\ (e, f) \\ = & \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma((a, b)) \wedge i_2 \in \gamma((c, d)) \wedge (\mathtt{machine\_binary\_plus}\ i_1\ i_2) \in \gamma((e, f)) \cap \mathbb{I}\}) \\ = & \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \{i_1' \in \mathbb{I} \mid a \le i_1' \le b\} \cup \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\} \wedge i_2 \in \{i_2' \in \mathbb{I} \mid c \le i_2' \le d\} \cup \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\} \wedge (\mathtt{machine\_binary\_plus}\ i_1\ i_2) \in \{r \in \mathbb{I} \mid e \le r \le f\}\}) \\ & \quad \text{(by def. } \mathtt{machine\_binary\_plus}\text{, we cannot have } i_1 \in \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\} \text{ or } i_2 \in \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\} \text{ and } i_1 + i_2 \text{ cannot overflow since otherwise } (\mathtt{machine\_binary\_plus}\ i_1\ i_2) = \Omega_{\mathtt{a}}\} \notin \mathbb{I}\text{)} \\ = & \alpha^2(\{\langle i_1,\ i_2\rangle \mid a \le i_1 \le b \wedge c \le i_2 \le d \wedge \mathtt{min\_int} \le i_1 + i_2 \le \mathtt{max\_int} \wedge e \le i_1 + i_2 \le f\}) \\ = & \alpha^2(\{\langle i_1,\ i_2\rangle \mid a \le i_1 \le b \wedge c \le i_2 \le d \wedge \max(\mathtt{min\_int}, e) \le i_1 + i_2 \le \min(\mathtt{max\_int}, f)\}) \end{array}$$

Because of commutativity in absence of overflow, the cases of i_1 and i_2 are symmetric, and so we consider only i_1. We have:

$$\max(\mathtt{min\_int}, e) - i_2 \le i_1 \le \min(\mathtt{max\_int}, f) - i_2$$

$$\begin{array}{ll} \text{and} & -d \le -i_2 \le -c \\ \text{so} & \max(\mathtt{min\_int}, e) - d \le i_1 \le \min(\mathtt{max\_int}, f) - c \\ \text{and} & a \le i_1 \le b \\ \text{so} & \max(a, \max(\mathtt{min\_int}, e) - d) \le i_1 \le \min(b, \min(\mathtt{max\_int}, f) - c) \\ \text{i.e.} & \max(a, \mathtt{min\_int} - d, e - d) \le i_1 \le \min(b, \mathtt{max\_int} - c, f - c) \\ \text{but} & \mathtt{min\_int} - \le a \le i_1 \le b \le \mathtt{max\_int} \\ \text{so} & \end{array}$$

$$\max(a, \mathtt{min\_int}, \mathtt{min\_int} - d, e - d) \le i_1 \le \min(b, \mathtt{max\_int}, \mathtt{max\_int} - c, f - c)$$

If $d \ge 0$ then $\mathtt{min\_int} \ge \mathtt{min\_int} - d$ while if $d < 0$ then $\mathtt{min\_int} \le e \le d < 0$ so $\mathtt{min\_int} - d \le e - d$ whence $\max(a, \mathtt{min\_int}, \mathtt{min\_int} - d, e - d) = \max(a, \mathtt{min\_int}, e - d)$. Then same way we have $\min(b, \mathtt{max\_int}, \mathtt{max\_int} - c, f - c) = \min(b, \mathtt{max\_int}, f - c)$ and so

$$\max(a, \mathtt{min\_int}, e - d) \le i_1 \le \min(b, \mathtt{max\_int}, f - c)$$

which can also be written in the form:

$\max(a, (\!|\, e - d < \mathtt{min\_int}\ ?\ \mathtt{min\_int} : e - d\,|\!)) \le i_1 \le \min(b, (\!|\, f - c > \mathtt{max\_int}\ ?\ \mathtt{max\_int} : f - c\,|\!))$.

The case of i_2 is symmetrical □

The test can be implemented using the following function which can easily be shown to be respectively equivalent to $(e - d <$ `min_int`$)$ and $f - c >$ `max_int` while avoiding overflows:

```
let is_difference_lt_min_int x y =
(* x - y < min_int *)
if (x < 0) && (y > 0) then
(x < min_int + y)
else false

let is_difference_gt_max_int x y =
(* x - y > max_int *)
if (x > 0) && (y < 0) then
(x > max_int + y)
else false
```


The symbolic execution of `b_PLUS` q_1 q_2 p yields the expected result as defined above:

```
(* b_BOP q1 q2 p = alpha2({<i1,i2> in gamma2(<q1,q2>) |          *)
(*            BOP(i1, i2) \in gamma(p) cap [min_int, max_int]}) *)
let b_PLUS  (a, b) (c, d) (e, f) =
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else let lq1 = max a (if (is_difference_lt_min_int e d)
                          then min_int  else (e - d))
        and uq1 = min b (if (is_difference_gt_max_int f c)
                  then max_int else (f - c))
    and lq2 = max c (if (is_difference_lt_min_int e b)
                    then min_int else (e - b))
    and uq2 = min d (if (is_difference_gt_max_int f a)
                    then max_int else (f - a))
```


Design of the abstract transformers: forward integer comparison

– For the generic forward/top-down nonrelational abstract semantics of boolean expressions, we have defined (in lecture 16):

$$\text{Abexp}[\![A_1 \mathrel{\mathsf{c}} A_2]\!]r = \check{\mathsf{c}}(\text{Faexp}^{\triangleright}[\![A_1]\!]r, \text{Faexp}^{\triangleright}[\![A_2]\!]r) \quad (10)$$

where

$$\check{\mathsf{c}}(p_1, p_2)r \mathrel{\dot{\sqsupseteq}} (\!| \exists v_1 \in \gamma(p_1) : \exists v_2 \in \gamma(p_2) \cap \mathbb{I} : v_1 \mathrel{\underline{\mathsf{c}}} v_2 = \mathsf{tt} \mathrel{?} r \mathrel{\$} \dot{\bot} |\!)$$

– Therefore, we define `f_LT` p q such that

$$(\exists i \in \gamma(p) \cap \mathbb{I} : \exists j \in \gamma(q) \cap \mathbb{I} : \texttt{machine\_lt}\ i\ j) \Longrightarrow (\texttt{f\_LT}\ p\ q)$$

where (from `values.ml`)

```
let machine_lt a b = match a with
| ERROR_NAT e -> (ERROR_BOOL e)
| NAT a'  -> match b with
  | ERROR_NAT e' -> (ERROR_BOOL e')
  | NAT b' -> (BOOLEAN (a' < b'))
```

THEOREM.

$$\texttt{f\_LT}\ \bot\ q = \bot$$
$$\texttt{f\_LT}\ p\ \bot = \bot$$
$$\texttt{f\_LT}\ (x, y)\ (x', y') = (x < y')$$

■

PROOF. – Observe that if $p = \bot$ or $q = \bot$ then $\gamma(p) \cap \mathbb{I} = \emptyset$ or $\gamma(q) \cap \mathbb{I} = \emptyset$ so that we have $\texttt{f\_LT}\ \bot\ q = \bot$ and $\texttt{f\_LT}\ p\ \bot = \bot$

– Otherwise we let $p = (x, y)$ and $q = (x', y')$ where $x \leq y$ and $x' \leq y'$. We have

$$
\begin{array}{ll}
& (\exists i \in \gamma((x,y)) \cap \mathbb{I} : \exists j \in \gamma((x',y')) \cap \mathbb{I} : \texttt{machine\_lt}\ i\ j) \\
= & (\exists i \in (\{i' \mid x \leq i' \leq y\} \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}) \cap \mathbb{I} : \exists j \in\in (\{j' \mid x' \leq j' \leq \\
& y'\} \cup \{\Omega_{\mathrm{i}}, \Omega_{\mathrm{a}}\}) \cap \mathbb{I} : \texttt{machine\_lt}\ i\ j) \\
= & (\exists i, j \in \mathbb{I} : x \leq i \leq y \wedge x' \leq j \leq y' \wedge \texttt{machine\_lt}\ i\ j) \\
= & (\exists i, j \in \mathbb{I} : x \leq i \leq y \wedge x' \leq j \leq y' \wedge i < j) \\
\Longrightarrow & (\exists i, j \in \mathbb{I} : x \leq i < j \leq y') \\
\Longrightarrow & (x < y')
\end{array}
$$

so when $p = (x, y) \neq \bot$ and $q = (x', y') \neq \bot$, we define $\texttt{f\_LT}\ (x, y)\ (x', y') = (x < y')$. □

This immediately leads to the following implementation:

```
(* Are there integer values in gamma(u) equal to values in gamma(v)?   *)
(* f_LT p q = exists i in gamma(p) cap [min_int,max_int]:              *)
(*           exists j in gamma(q) cap [min_int,max_int]: machine_eq i j *)
let f_EQ (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) then false
   else (min y y') <= (max x x')
```


Design of the abstract transformers: forward integer comparison, revisited version

– When considering the improved abstract interpretation of boolean expressions using the backward abstract interpretation of arithmetic subexpressions (course 17), we have defined:

$$
\begin{array}{l}
\mathrm{Abexp}[\![A_1 \mathrel{\mathsf{c}} A_2]\!]r \stackrel{\mathrm{def}}{=} \\
\quad \mathrm{let}\ \langle p_1,\ p_2\rangle = \check{\mathsf{c}}(\mathrm{Faexp}^{\triangleright}[\![A_1]\!]r, \mathrm{Faexp}^{\triangleright}[\![A_2]\!]r)\ \mathrm{in} \\
\qquad \mathrm{Baexp}^{\triangleleft}[\![A_1]\!](r)p_1\ \dot{\sqcap}\ \mathrm{Baexp}^{\triangleleft}[\![A_2]\!](r)p_2
\end{array}
$$

where

$$
\begin{array}{l}
\check{\mathsf{c}}(p_1, p_2) \sqsupseteq^2 \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma(p_1) \cap \mathbb{I} \\
\qquad \wedge\ i_2 \in \gamma(p_2) \cap \mathbb{I} \wedge i_1 \mathrel{\underline{\mathsf{c}}} i_2 = \mathtt{tt}\})
\end{array}
$$

– Up to the machine representation of abstract values, we define:

$$
\begin{array}{l}
\texttt{a\_LT}\ p\ q = \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma(p_1) \cap \mathbb{I} \wedge i_2 \in \gamma(p_2) \cap \mathbb{I} \\
\qquad \wedge\ i_1 < i_2\})
\end{array}
$$

THEOREM.

$$
\begin{aligned}
\texttt{a\_LT}\ \bot\ q &= \langle \bot,\ \bot \rangle \\
\texttt{a\_LT}\ p\ \bot &= \langle \bot,\ \bot \rangle \\
\texttt{a\_LT}\ (a,b)\ (c,d) &= \langle \bot,\ \bot \rangle \quad \text{if } a \geq d \\
&= \langle [a, \min(b, d-1)],\ [\max(a+1, c), d] \rangle \\
&\qquad \text{if } a < d
\end{aligned}
$$

■

PROOF. – If $p = \bot$ or $q = \bot$ then $\gamma(p) \cap \mathbb{I}$ or $\gamma(q) \cap \mathbb{I}$ is $\emptyset$ so $\texttt{a\_LT}\ \bot\ q = \langle \bot,\ \bot \rangle$ and $\texttt{a\_LT}\ p\ \bot = \langle \bot,\ \bot \rangle$

– Otherwise $p = [a, b)$ and $q = (c, d)$ with $\texttt{min\_int} \leq a \leq b \leq \texttt{max\_int}$ and $\texttt{min\_int} \leq c \leq d \leq \texttt{max\_int}$

– We have

$$
\begin{aligned}
&\texttt{a\_LT}\ (a,b)\ (c,d) \\
&\alpha^2(\{\langle i_1,\ i_2 \rangle \mid i_1 \in \gamma(a,b)) \cap \mathbb{I} \wedge i_2 \in \gamma((c,d)) \cap \mathbb{I} \wedge i_1 < i_2\}) \\
&\alpha^2(\{\langle i_1,\ i_2 \rangle \mid a \leq i_1 \leq b \wedge c \leq i_2 \leq d \wedge i_1 < i_2\})
\end{aligned}
$$

Now, we consider three cases:

– If $d \leq a$, then we get $\alpha^2(\emptyset) = \langle \bot,\ \bot \rangle$. In this case

- $(a, \min(b, d-1)) = (a, d-1)$ since $d \leq a \leq b$
  $= \bot$ since $d - 1 < a$
- $(\max((a+1), c), d) = (a+1, d)$ since $c \leq d \leq a < a+1$
  $= \bot$ since $d < a + 1$

– Otherwise $a < d$, in which case:

$$
\begin{aligned}
&\alpha^2(\{\langle i_1,\ i_2 \rangle \mid a \leq i_1 \leq b \wedge c \leq i_2 \leq d \wedge i_1 < i_2\}) \\
= \ &\alpha^2(\{\langle i_1,\ i_2 \rangle \mid a \leq i_1 \leq b \wedge i_1 < d \wedge c \leq i_2 \leq d \wedge a < i_2 \wedge i_1 < i_2\}) \\
&\quad \wr \alpha^2 \text{ is monotone} \int \\
\sqsubseteq_2 \ &\alpha^2(\{\langle i_1,\ i_2 \rangle \mid a \leq i_1 \leq \min(b, d-1) \wedge \max(a+1, c) \leq i_2 \leq d \wedge i_1 < i_2\})
\end{aligned}
$$

$= \ ((a, \min(b, d-1)), (\max(a+1, c), d))$

which is of the same form than in the previous bottom case. □

The two cases can be grouped together in the implementation:

```
(* a_LT p1 p2 = alpha2({<i1, i2> |                                  *)
(*                      i1 in gamma(p1) cap [min_int, max_int] /\ *)
(*                      i2 in gamma(p1) cap [min_int, max_int] /\ *)
(*                      i1 < i2})                                   *)
let a_LT (a, b) (c, d) =
if (isbottom (a, b)) || (isbottom (c, d)) || (a >= d) then
  (bottom, bottom) else ((a, min b (d - 1)), (max (a + 1) c, d))
```


Implementation of the abstract transformers

```
(* avalues.ml *)
open Values
(* abstraction of sets of machine integers by intervals *)
(* complete lattice *)
...
(*                         *)
(* ABSTRACT TRANSFORMERS *)
(*                         *)
(* forward abstract semantics of arithmetic expressions *)
(* f_NAT s = alpha({(machine_int_of_string s)})       *)
let f_NAT s =
   match (machine_int_of_string s) with
   | (ERROR_NAT INITIALIZATION) -> bottom
   | (ERROR_NAT ARITHMETIC) -> bottom
```

```
   | (NAT i) -> (i,i)
(* f_RANDOM () = alpha([min_int, max_int]) *)
let f_RANDOM () = (min_int, max_int)
(* f_UMINUS a = alpha({ (machine_unary_minus x) | x \in gamma(a)} }) *)
let f_UMINUS (x, y) = if (isbottom (x, y)) then bottom
   else if (x = min_int) then (-y, max_int)
   else (-y, -x)
(* f_UPLUS a = alpha(gamma(a)) *)
let f_UPLUS x = x
(* f_BINARITH a b = alpha({ (machine_binary_binarith i j) |      *)
(*                              i in gamma(a) /\ j \in gamma(b)} *)
let is_sum_lt_min_int x y =
   (* x + y < min_int *)
   if (x < 0) & (y < 0) then
      (x < min_int - y)
   else false
let is_sum_gt_max_int x y =
   (* x + y > max_int *)
```



```
      if (x > 0) && (y > 0) then
         (x > max_int - y)
      else false
let f_PLUS (a, b) (c, d) =
  if (isbottom (a, b)) || (isbottom (c, d)) then bottom
  else if (is_sum_gt_max_int a c) then bottom
  else if (is_sum_lt_min_int b d) then bottom
  else let lb = if (is_sum_lt_min_int a c) then min_int else a + c
       and ub = if (is_sum_gt_max_int b d) then max_int else b + d
       in (lb, ub)
let is_difference_lt_min_int x y =
   (* x - y < min_int *)
   if (x < 0) && (y > 0) then
      (x < min_int + y)
   else false
let is_difference_gt_max_int x y =
   (* x - y > max_int *)
   if (x > 0) && (y < 0) then
```



```
      (x > max_int + y)
   else false
let f_MINUS (a, b) (c, d) =
   if (isbottom (a, b)) || (isbottom (c, d)) then bottom
   else if (is_difference_gt_max_int a d) then bottom
   else if (is_difference_lt_min_int b c) then bottom
   else let lb = if (is_difference_lt_min_int a d) then min_int
                 else a - d
        and ub = if (is_difference_gt_max_int b c) then max_int
                    else b - c
        in (lb, ub)
let sign x = if (x >= 0) then 1 else -1

exception Error_abs of string
let abs x = if (x >= 0) then x
            else if (x = min_int) then
               raise (Error_abs "Incoherence: abs(min_int)")
            else (- x)
```



```

let times_int x y =
  if (x = 0) or (y = 0) then 0
  else if x = min_int then
    (if y = 1 then min_int else if y < 0 then max_int else min_int)
  else if y = min_int then
    (if x = 1 then min_int else if x < 0 then max_int else min_int)
  else if (sign x) * (sign y) > 0 then
    (if (abs x) <= (max_int/(abs y)) then (x*y) else max_int)
  else if (abs x) = 1 then (x*y)
  else
    (if (abs y) <= (min_int/(-(abs x))) then (x*y) else min_int)

let f_TIMES (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) then bottom
   else let a = times_int x x'
        and b = times_int x y'
        and c = times_int y x'
```

```
       and d = times_int y y' in
          ((min (min a b) (min c d)), (max (max a b) (max c d)))

let rec f_DIV (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) || ((x' = 0) && (y' = 0))
   then bottom
   else if x' = 0 then f_DIV (x, y) (1, y')
   else if y' = 0 then f_DIV (x, y) (x', 1)
   else let a = x/x'
        and b = x/y'
        and c = y/x'
        and d = y/y' in
           ((min (min a b) (min c d)), (max (max a b) (max c d)))
let rec f_MOD (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) || (y < 0) || (y' < 1)
   then bottom
   else if x' < 0 then f_MOD (x, y) (0, y')
   else if y' <= 0 then f_MOD (x, y) (x', 1)
```



```
   else let a = x mod x'
     and b = x mod y'
     and c = y mod x'
     and d = y mod y' in
        ((min (min a b) (min c d)), (max (max a b) (max c d)))
(* forward abstract semantics of boolean expressions                    *)
(* Are there integer values in gamma(u) equal to values in gamma(v)? *)
(* f_LT p q = exists i in gamma(p) cap [min_int,max_int]:             *)
(*        exists j in gamma(q) cap [min_int,max_int]: machine_eq i j *)
let f_EQ (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) then false
   else (min y y') <= (max x x')
(* Are there integer values in gamma(u) strictly less than (<)        *)
(* integer values in gamma(v)?                                        *)
(* f_LT p q = exists i in gamma(p) cap [min_int,max_int]:             *)
(*        exists j in gamma(q) cap [min_int,max_int]: machine_lt i j *)
let f_LT (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) then false
```



```
   else (y' > x)
...
(* backward abstract semantics of arithmetic expressions             *)
(* b_NAT s v = (machine_int_of_string s) in
(*                                  gamma(v) cap [min_int, max_int]? *)
let b_NAT s (a, b) =
   match (machine_int_of_string s) with
   | (ERROR_NAT INITIALIZATION) -> false
   | (ERROR_NAT ARITHMETIC) -> false
   | (NAT i) -> (a <= i) && (i <= b)
(* b_RANDOM p = gamma(p) cap [min_int, max_int]  <> emptyset *)
let b_RANDOM p = not (isbottom p)
(* b_UOP q p = alpha({i in gamma(q) |                            *)
(*                 UOP(i) \in gamma(p) cap [min_int, max_int]}) *)
let b_UMINUS q (a, b) = meet q (-b, -a)
let b_UPLUS q p = meet q p
(* b_BOP q1 q2 p = alpha2({<i1,i2> in gamma2(<q1,q2>) |           *)
(*              BOP(i1, i2) \in gamma(p) cap [min_int, max_int]}) *)
```



```
let b_PLUS  (a, b) (c, d) (e, f) =
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else let lq1 = max a (if (is_difference_lt_min_int e d)
                          then min_int  else (e - d))
        and uq1 = min b (if (is_difference_gt_max_int f c)
                       then max_int else (f - c))
        and lq2 = max c (if (is_difference_lt_min_int e b)
                          then min_int else (e - b))
        and uq2 = min d (if (is_difference_gt_max_int f a)
                          then max_int else (f - a))
        in (if (lq1 <= uq1) then (lq1, uq1) else bottom),
           (if (lq2 <= uq2) then (lq2, uq2) else bottom)
let b_MINUS (a, b) (c, d) (e, f) =
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else b_PLUS  (a, b) (-d, -c) (e, f)
let b_TIMES (a, b) (c, d) (e, f) =
```

```
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else (a, b), (c, d)
let b_DIV (a, b) (c, d) (e, f) =
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else (a, b), (c, d)
let b_MOD (a, b) (c, d) (e, f) =
   if (in_errors (a, b)) || (in_errors (c, d)) then errors, errors
   else if (in_errors (e, f)) then bottom, bottom
   else (a, b), (c, d)
(* backward abstract interpretation of boolean expressions *)
(* a_EQ p1 p2 = let p = p1 cap p2 cap [min_int, max_int]I in <p, p> *)
let a_EQ p1 p2 = let p = meet p1 p2 in (p, p)
(* a_LT p1 p2 = alpha2({<i1, i2> |                           *)
(*                      i1 in gamma(p1) cap [min_int, max_int] /\ *)
(*                      i2 in gamma(p1) cap [min_int, max_int] /\ *)
(*                      i1 < i2})                            *)
```



```
let a_LT (a, b) (c, d) =
if (isbottom (a, b)) || (isbottom (c, d)) || (a >= d) then
  (bottom, bottom) else ((a, min b (d - 1)), (max (a + 1) c, d))
```


Design of the abstract convergence accelerators

Widening (with thresholds)

– The widening is defined with thresholds (including `min_int` and `max_int` by default):

```
(* widening *)
(* let thresholds = [| |] (* only min_int and max_int *) *)
(* widening with  thresholds                              *)
let cmp i j = if i < j then -1 else if i = j then 0 else 1
let thresholds = let data = [| -1; 0; 1; |] in
                     (Array.sort cmp data; data)
let widen (x, y) (x', y') =
  if (isbottom (x, y)) then (x', y')
  else if (isbottom (x', y')) then (x, y)
  else let lastindex = (Array.length thresholds) - 1 in
    let a = if x' >= x then x
       else let i = ref lastindex in
```

```
          (while (!i >= 0) & (x' < thresholds.(!i)) do
             i := !i - 1
           done;
           if (!i < 0) then min_int else thresholds.(!i))
  and b = if y' <= y then y
     else let j = ref 0 in
          (while (!j <= lastindex) & (y' > thresholds.(!j)) do
              j := !j + 1
           done;
           if (!j > lastindex) then max_int else thresholds.(!j))
  in a, b
```


THEOREM. `widen` is a widening operator. ■

PROOF. – We first prove that $p \sqsubseteq p \mathbin{\nabla} q$

- if $p = \bot$ then $p = \bot \sqsubseteq q = p \mathbin{\nabla} q$
- if $q = \bot$ then $p \sqsubseteq p = p \mathbin{\nabla} q$
- Otherwise $p = (x, y)$, $q = (x', y')$ such that $x \le y$ and $x' \le y'$. Then $p \mathbin{\nabla} q = (a, b)$. We must show that $a \le \min(x, x')$.
  - if $x' \ge x$ then $a = \min(x, x') = x$
  - otherwise $x' < x$ in which case we must prove that $a \le x' = \min(x, x')$. We consider two cases
    - if `thresholds` $=$ `[||]` is empty then `Array.length thesholds)` $= 0$ so `lastindex` $=$ -1 whence `!i` $=$ -1 < 0 and $a =$ `min_int` which satisfies $a \le x'$
    - Otherwise `thresholds` $=$ `[|`$t_0; \ldots; t_n$`|]` with $n \ge 0$ is not empty. So `lastindex` $=$ texttttArray.length thesholds) - 1 = n. Then the following loop is executed.


```
let i = ref lastindex in
   while (!i >= 0) & (x' < thresholds.(!i)) do
      i := !i - 1
   done;
```

A simple Floyd-Naur Hoare invariance argument shows that there are two cases:

- if $x' < t_0 \le t_1 \le \ldots \le t_n$ then `!i` $=$ -1 in which case $a =$ `min_int`
- otherwise $t_0 \le \ldots \le t_{!i} \le x' < t_{!i+1} \le \ldots \le t_n$ in which case $a = t_{!i} \le x'$

- The condition $\max(y, y') \le b$ follows by duality on $\le$

– The proof that $q \sqsubseteq p \mathbin{\nabla} q$ can be handled by a very similar argument which is left to the reader

– Finally, we must prove that for all infinite sequences $x^0, x^1, \ldots$, the sequence defined by

$$
\begin{aligned}
y^0 &= x^0 && && \text{(a)}\\
y^{\delta+1} &= y^\delta && \text{if } x^\delta \sqsubseteq y^\delta && \text{(b)}\\
&= y^\delta \mathbin{\nabla} x^\delta && \text{otherwise} && \text{(c)}
\end{aligned}
$$

is not strictly increasing. The proof is by reductio ad absurdum.

Assume that $y^0 \sqsubset y^1 \sqsubset \ldots \sqsubset y^\delta \sqsubset \ldots$ then (b) is never used. It follows that $\forall \delta \ge 0 : y^{\delta+1} = y^\delta \mathbin{\nabla} x^\delta$. The only $\bot$ element can be y^0, which can be eliminated by considering $x^1, x^2, \ldots$ and $y_1, y_2, \ldots$ with all $y^i \ne \bot$ and without changing the final result. Moreoever the x^i cannot be $\bot$ since in that case $y^{\delta+1} = y^\delta \mathbin{\nabla} \bot = y^\delta$ in contradiction with $y^\delta \sqsubset y^{\delta+1}$. Therefore we have $x^\delta = (a^\delta, b^\delta)$, $\delta \ge 1$ with `min_int` $\le a^\delta \le b^\delta \le$ `max_int` such that the sequence $y^\delta = (c^\delta, d^\delta)$, $\delta \ge 1$ with `min_int` $\le c^\delta \le d^\delta \le$ `max_int` is strictly increasing, with

$$(c^{\delta+1}, d^{\delta+1}) = (c^\delta, d^\delta) \mathbin{\nabla} (a^\delta, b^\delta),\ \delta \ge 1$$

– Because

$$(c^\delta, d^\delta) \sqsubset (c^{\delta+1}, d^{\delta+1})$$

we have

$$c^{\delta+1} < c^{\delta}) \vee (d^{\delta} < d^{\delta}+1)$$

by definition of $\sqsubseteq$

– In case $c^{\delta+1} < c^{\delta}$, we have by definition of ∇ that

$$\begin{aligned} c^{\delta+1} &= a^{\delta} && \text{if } a^{\delta} \geq c^{\delta} \\ &= t_i, 0 \leq i \leq n && \text{when } \texttt{thresholds} = [\,|t_0;\ldots;t_n|\,] \end{aligned}$$

Observe that the first case is indeed impossible since $c^{\delta+1} < c^{\delta}$ implies $\neg(c^{\delta+1} = a^{\delta} \geq c^{\delta})$ so $c^{\delta+1} = t_i$

– In case $d^{\delta} < d^{\delta+1}$, a similar reasoning shows that $d^{\delta+1} = t^j$, $j \in [0, n]$

– So we have a decreasing chain of elements of "thesholds" for $\langle c^{\delta},\ \delta \geq 2\rangle$ and an increasing chain of elements of "thesholds" for $\langle d^{\delta},\ \delta \geq 2\rangle$, one of them strictly increasing for each δ, which is impossible since "thesholds" is finite, which provides the desired contradiction.

$\square$

Narrowing

– The narrowing is defined as follows:

```
(* narrowing *)
let narrow (x, y) (x', y') =
   if (isbottom (x, y)) || (isbottom (x', y')) then bottom
   else ((if (x = min_int) then x' else x),
                    (if (y = max_int) then y' else y))
```

THEOREM. `narrow` *is a narrowing operator.* ■

PROOF. Let us show that this definition satisfies the hypotheses on narrowings.

– Assuming $q \sqsubset p$, we must show that $q \sqsubseteq p \,\Delta\, q \sqsubseteq p$.
  - the case $p = \perp$ is excluded by $q \sqsubset p$
  - the case $q = \perp$ yields $\perp \sqsubseteq \Delta q = \perp \sqsubseteq p$

- otherwise, $p = (x, y)$, $q = (x', y')$ with $\texttt{min\_int} \leq x \leq y \leq \texttt{max\_int}$ and $\texttt{min\_int} \leq x' \leq y' \leq \texttt{max\_int}$ and $(x', y') \sqsubseteq (x, y)$. By cases:
  · if $x = \texttt{min\_int}$ then
    - if $y = \texttt{max\_int}$ then, by symbolic execution, $(x', y') \sqsubseteq (x', y') = (x, y) \,\Delta\, (x', y') \sqsubseteq (x, y)$
    - else $(x', y') \sqsubseteq (x', y) = (x, y) \Delta (x', y') \sqsubseteq (x, y)$ since $x' \leq x$ by $(x', y') \sqsubseteq (x, y)$
  · otherwise $x \neq \texttt{min\_int}$, and then
    - if $y = \texttt{max\_int}$ then, by symbolic execution, $(x', y') \sqsubseteq (x', \texttt{max\_int}) = (x, y) \,\Delta\, (x', y') \sqsubseteq (x, y)$ since $x' \leq y$ by $(x', y') \sqsubseteq (x, y)$ and $y = \texttt{max\_int}$
    - otherwise $y \neq \texttt{max\_int}$ and then $(x', y') = (x, y) \,\Delta\, (x', y') \sqsubseteq (x, y)$

– We must also show that for all sequences p^0, p^1, ... the sequence defined by

$$\begin{aligned} q^0 &= p^0 \\ q^{\delta+1} &= q^{\delta} \,\Delta\, p^{\delta} && \text{if } p^{\delta} \sqsubset q^{\delta} \\ &= q^{\delta} && \text{otherwise} \end{aligned}$$

is not stritly increasing.

The proof is by reductio ad absurdum. Assume that $\langle q^{\delta},\ \delta \geq 0\rangle$ is strictly decreasing. The case (c) can never be chosen since we would have the contradiction that $q^{\delta+1} = q^{\delta}$ for some $\delta \geq 0$. So thye sequence $\langle q^{\delta},\ \delta \geq 0\rangle$ is defined using (a) and (b) only that is (b) only for $\langle q^{\delta},\ \delta \geq 1\rangle$. Let $q^{\delta} = (a^{\delta}, b^{\delta})$ and $p^{\delta} = (c^{\delta}, d^{\delta})$ for all $\delta \geq 1$. We have:

$$\begin{aligned} & (c^{\delta}, d^{\delta}) \sqsubset (a^{\delta}, b^{\delta}) \\ \text{and} \quad & (a^{\delta+1}, b^{\delta+1}) = (a^{\delta}, b^{\delta}) \,\Delta\, (c^{\delta}, d^{\delta}) \\ \text{and} \quad & (a^{\delta+1}, b^{\delta+1}) \sqsubset (a^{\delta}, b^{\delta}) \end{aligned}$$

After $\delta \geq 1$, all elements of $\langle q^\delta,\ \delta \geq 1\rangle$ are not $\perp$. We must have $a^{\delta+1} < a^\delta$ (or $b^{\delta+1} > b^\delta$ which is handled in the same way). By definition of Δ, if $a^\delta = \texttt{min\_int}$ then $a^{\delta+1} = c^\delta \leq a^\delta$ else $a^{\delta+1} = a^\delta$ which is impossible. So $a^\delta = \texttt{min\_int}$ at the next step $a^{\delta+2} < a^{\delta+1} = c^\delta \leq a^\delta = \texttt{min\_int}$ whic is impossible. This yields the contradiction proving that Δ enforces convergence.

$\square$

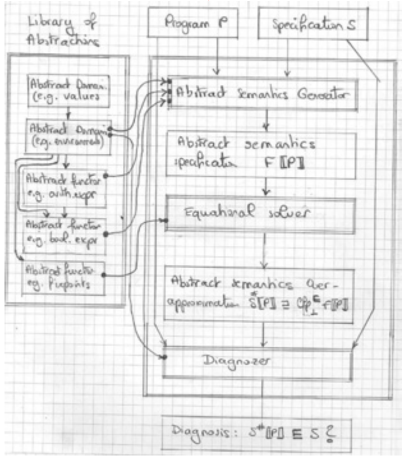
Generic abstract interpreter: A first implementation

Making the non-relational forward analyzer generic

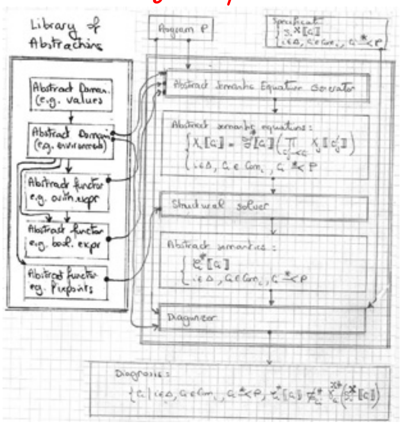
- The global structure of the analyzer is the same whichever is the abstract domain chosen to approximate sets of values;
- Up to the use of widening/narrowing when no convergence acceleration is needed (e.g. finite domains, domains satisfying the ACC with rapid convergence)
- For non-relational analyzes, the structure of the abstract domain approximating sets of environements only depends on the abstract doamin for sets of values

- The algebraic structure can be represented by the modular structure of OCaml programs (thanks to file aliases in this first implementation or better thanks to module functors)
- It is then easy to modify the static analyzer to perform experimentations on the abstract domains:
  - → by changing the abstract domain of values
  - → by changing the abstract interpretation of arithmetic/boolean expressions or commands
  - → without having to change the global structure of the analyzer

Principle of a generic equational static analyzer/verifier



Principle of a generic structural static analyzer/verifier



A first implementation using the modular structure of OCAML and aliases of module files

```
Generic-FW-Abstract-Interpreter % make
Forward non-relational static analysis:
make help          : this help
1) reset:
make reset         : erase all mode choices
2) choose tracing mode:
make trace         : tracing all
make traceaexp     : tracing arithmetic expressions
make tracebexp     : tracing boolean expressions
make tracecom      : tracing commands
make tracered      : tracing ternary reductions
make notrace       : no tracing
```



```
3) choose abstract interpreter mode:
3a) relational/non-relational analysis:
make r             : relational abstract interpretor (not implemented)
make nr            : non-relational abstract interpretor
3b) boolean expressions:
make fbool         : forward analysis
make fbbool        : forward/backward analysis
make fbrbool       : forward/backward reductive analysis
3c) arithmetic expressions:
make fassign       : forward analysis
make fbassign      : forward/backward analysis
4) choose static analysis and compile analyzer:
make err           : error analysis
make iss           : initialization and simple sign analysis
make int           : interval analysis
```

```
5) analyze:
./a.out           : analyze (the standard input)
./a.out file.sil : analyze (the file "file.sil")
make examples     : analyze all examples
6) clean:
make clean        : remove auxiliary files
```


File structure of the generic forward static analyzer

```
|- Examples
|   |- example00.sil
|   |- ...
|   |- example73.sil
|   `- makefile
|- Generic-FW-Abstract-Interpreter
|   |- Generic-FW-Abstract-Interpreter.tgz
|   |- aaexp.ml
|   |- aaexp.mli
|   |- abexp.ml -> abexp_fbr.ml
|   |- abexp.mli
|   |- abexp_f.ml
|   |- abexp_fb.ml
|   |- abexp_fbr.ml
```



```
|   |- abstract_Syntax.ml
|   |- acom.ml -> acom_fba.ml
|   |- acom.mli
|   |- acom_fa.ml
|   |- acom_fba.ml
|   |- aenv.ml -> ../Non-Relational/aenv.ml
|   |- aenv.mli
|   |- avalues.ml -> ../Non-Relational/03-Intervals/avalues.ml
|   |- avalues.mli -> ../Non-Relational/avalues.mli
|   |- baexp.ml
|   |- baexp.mli
|   |- concrete_To_Abstract_Syntax.ml
|   |- concrete_To_Abstract_Syntax.mli
|   |- fixpoint.ml
|   |- fixpoint.mli
|   |- labels.ml
|   |- labels.mli
```



```
|   |- lexer.mll
|   |- main.ml
|   |- makefile
|   |- parser.mly
|   |- pretty_Print.ml
|   |- pretty_Print.mli
|   |- program_To_Abstract_Syntax.ml
|   |- program_To_Abstract_Syntax.mli
|   |- red12.mli
|   |- red123.ml
|   |- red123.mli
|   |- red13.mli
|   |- red23.mli
|   |- symbol_Table.ml
|   |- symbol_Table.mli
|   |- trace.ml
|   |- trace.mli
```

```
|    |- typescript
|    |- values.ml
|    |- values.mli
|    |- variables.ml
|    `- variables.mli
|- Non-Relational
|    |- 01-Initialization-Simple-Sign
|    |    |- avalues.ml
|    |- 02-Errors
|    |    |- avalues.ml
|    |- 03-Intervals
|    |    |- avalues.ml
|    |    |- avalues.mli
|    |- 04-Parity
|    |    |- avalues.ml
|    |- aenv.ml
|    |- avalues.mli
|    `- laenv.ml
```


All files are shared by all instances, but:

– `aenv.ml` (and `avalues.mli` common to all non-relational abstractions)

– `avalues.ml` implementing each specific non-relational abstract domain (errors, intervals, ...)

Creating a specific instance of the generic analyzer

The creation of a specific instance of the analyzer consists in creating aliases of the specific instanciated files before recompiling.

```
% pwd
.../Generic-FW-Abstract-Interpreter
% make reset
Remove instanciated files
% make notrace
Tracing mode off
% make nr
"Non-relational" static analysis
% make fbrbool
Forward/backward analysis of boolean expressions with reduction
```



```
% make fbassign
Forward/backward analysis of assignments
% make int
ocamlyacc parser.mly
ocamllex lexer.mll
62 states, 3001 transitions, table size 12376 bytes
ocamlc trace.mli trace.ml symbol_Table.mli symbol_Table.ml
variables.mli variables.ml abstract_Syntax.ml
concrete_To_Abstract_Syntax.mli concrete_To_Abstract_Syntax.ml
labels.mli labels.ml parser.mli parser.ml lexer.ml
program_To_Abstract_Syntax.mli program_To_Abstract_Syntax.ml
pretty_Print.mli pretty_Print.ml values.mli values.ml avalues.mli
avalues.ml aenv.mli aenv.ml aaexp.mli aaexp.ml baexp.mli baexp.ml
fixpoint.mli fixpoint.ml abexp.mli abexp.ml acom.mli acom.ml main.ml
"Interval" static analysis
```

For example for interval analysis, the aliases will be created as follows:

```
 % tree
...
|- abexp.ml -> abexp_fbr.ml
...
|- acom.ml -> acom_fba.ml
...
|- aenv.ml -> ../Non-Relational/aenv.ml
...
|- avalues.ml -> ../Non-Relational/03-Intervals/avalues.ml
...
|- avalues.mli -> ../Non-Relational/avalues.mli
...
```


Generating an instance of the generic forward static analyzer

```
# makefile

SHELL = tcsh

EXAMPLES = ../Examples

SOURCES_SINGLE_DOMAIN = \
trace.mli \
trace.ml \
symbol_Table.mli \
symbol_Table.ml \
variables.mli \
variables.ml \
abstract_Syntax.ml \
concrete_To_Abstract_Syntax.mli \
```



```
concrete_To_Abstract_Syntax.ml \
labels.mli \
labels.ml \
parser.mli \
parser.ml \
lexer.ml \
program_To_Abstract_Syntax.mli \
program_To_Abstract_Syntax.ml \
pretty_Print.mli \
pretty_Print.ml \
values.mli \
values.ml \
avalues.mli \
avalues.ml \
aenv.mli \
aenv.ml \
aaexp.mli \
aaexp.ml \
baexp.mli \
baexp.ml \
```



```
fixpoint.mli \
fixpoint.ml \
abexp.mli \
abexp.ml \
acom.mli \
acom.ml \
main.ml

SOURCES_BINARY_REDUCED_PRODUCT = \
trace.mli \
trace.ml \
symbol_Table.mli \
symbol_Table.ml \
variables.mli \
variables.ml \
abstract_Syntax.ml \
concrete_To_Abstract_Syntax.mli \
concrete_To_Abstract_Syntax.ml \
labels.mli \
labels.ml \
```

```
parser.mli \
parser.ml \
lexer.ml \
program_To_Abstract_Syntax.mli \
program_To_Abstract_Syntax.ml \
pretty_Print.mli \
pretty_Print.ml \
values.mli \
values.ml \
avalues1.mli \
avalues1.ml \
avalues2.mli \
avalues2.ml \
red12.mli \
red12.ml \
avalues.mli \
avalues.ml \
aenv.mli \
aenv.ml \
aaexp.mli \
```



```
aaexp.ml \
baexp.mli \
baexp.ml \
fixpoint.mli \
fixpoint.ml \
abexp.mli \
abexp.ml \
acom.mli \
acom.ml \
main.ml

SOURCES_TERNARY_REDUCED_PRODUCT = \
trace.mli \
trace.ml \
symbol_Table.mli \
symbol_Table.ml \
variables.mli \
variables.ml \
abstract_Syntax.ml \
concrete_To_Abstract_Syntax.mli \
```



```
concrete_To_Abstract_Syntax.ml \
labels.mli \
labels.ml \
parser.mli \
parser.ml \
lexer.ml \
program_To_Abstract_Syntax.mli \
program_To_Abstract_Syntax.ml \
pretty_Print.mli \
pretty_Print.ml \
values.mli \
values.ml \
avalues1.mli \
avalues1.ml \
avalues2.mli \
avalues2.ml \
avalues3.mli \
avalues3.ml \
red12.mli \
red12.ml \
```



```
red23.mli \
red23.ml \
red13.mli \
red13.ml \
red123.mli \
red123.ml \
avalues.mli \
avalues.ml \
aenv.mli \
aenv.ml \
aaexp.mli \
aaexp.ml \
baexp.mli \
baexp.ml \
fixpoint.mli \
fixpoint.ml \
abexp.mli \
abexp.ml \
acom.mli \
acom.ml \
```

```
main.ml

.PHONY : help
help :
	@echo ""
	@echo "Forward non-relational static analysis:"
	@echo "make help        : this help"
	@echo "(1) reset:"
	@echo "make reset       : erase all mode choices"
	@echo "(2) choose tracing mode:"
	@echo "make trace       : tracing all"
	@echo "make traceaexp   : tracing arithmetic expressions"
	@echo "make tracebexp   : tracing boolean expressions"
	@echo "make tracecom    : tracing commands"
	@echo "make tracered    : tracing ternary reductions"
	@echo "make notrace     : no tracing"
	@echo "(3) choose abstract interpreter mode:"
	@echo "(3a) relational/non-relational analysis:"
	@echo "make r           : relational abstract interpretor"
	@echo "make nr          : non-relational abstract interpretor"
```



```
	@echo "(3b) boolean expressions:"
	@echo "make fbool       : forward analysis"
	@echo "make fbbool      : forward/backward analysis"
	@echo "make fbrbool     : forward/backward reductive analysis"
	@echo "(3c) arithmetic expressions:"
	@echo "make fassign     : forward analysis"
	@echo "make fbassign    : forward/backward analysis"
	@echo "(4) choose static analysis and compile analyzer:"
	@echo "make err         : error analysis"
	@echo "make iss         : initialization and simple sign analysis"
	@echo "make int         : interval analysis"
	@echo "make par         : parity analysis"
	@echo "make err-int     : error x interval analysis"
	@echo "make iss-int     : initialization and simple sign x interval analysis"
	@echo "make par-int     : parity x interval analysis"
	@echo "make par-iss     : parity x initialization and simple sign analysis"
	@echo "make par-iss-int : parity x initialization and simple sign analysis x"
	@echo "                   interval"
	@echo "(5) analyze:"
	@echo "./a.out          : analyze (the standard input)"
```



```
	@echo './a.out file.sil : analyze (the file "file.sil")'
	@echo "make examples    : analyze all examples"
	@echo "(6) clean:"
	@echo "make clean       : remove auxiliary files"
	@echo ""

.PHONY : trace
trace : traceaexp tracebexp tracecom tracered

.PHONY : traceaexp
traceaexp :
	-@/bin/rm -f trace_aexp  || true
	@echo "" > trace_aexp
	@echo "Tracing arithmetic expressions"

.PHONY : tracebexp
tracebexp :
	-@/bin/rm -f trace_bexp  || true
	@echo "" > trace_bexp
	@echo "Tracing boolean expressions"
```



```

.PHONY : tracecom
tracecom :
	-@/bin/rm -f trace_com  || true
	@echo "" > trace_com
	@echo "Tracing commands"

.PHONY : tracered
tracered :
	-@/bin/rm -f trace_red  || true
	@echo "" > trace_red
	@echo "Tracing ternary reductions"


.PHONY : notrace
notrace :
	-@/bin/rm -f trace_* || true
	@echo "Tracing mode off"

.PHONY : fbool
```

```
fbool :
	@/bin/rm -f abexp.ml || true
	@ln -s abexp_f.ml abexp.ml
	@echo "Forward analysis of boolean expressions"

.PHONY : fbbool
fbbool :
	@/bin/rm -f abexp.ml || true
	@ln -s abexp_fb.ml abexp.ml
	@echo "Forward/backward analysis of boolean expressions"

.PHONY : fbrbool
fbrbool :
	@/bin/rm -f abexp.ml || true
	@ln -s abexp_fbr.ml abexp.ml
	@echo "Forward/backward analysis of boolean expressions with reduction"

.PHONY : fassign
fassign :
	@/bin/rm -f acom.ml || true
```



```
	@ln -s acom_fa.ml acom.ml
	@echo "Forward analysis of assignments"

.PHONY : fbassign
fbassign :
	@/bin/rm -f acom.ml || true
	@ln -s acom_fba.ml acom.ml
	@echo "Forward/backward analysis of assignments"

.PHONY : r
r : nr
	@echo '"Relational" static analysis not implemented'

.PHONY : nr
nr :
	@/bin/rm -f aenv.ml || true
	@ln -s ../Non-Relational/aenv.ml aenv.ml
	@/bin/rm -f avalues.mli || true
	@ln -s ../Non-Relational/avalues.mli avalues.mli
	@echo '"Non-relational" static analysis'
```



```

.PHONY : err
err :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/02-Errors/avalues.ml avalues.ml
#	ocamlc -i ${SOURCES_SINGLE_DOMAIN} # to print types
	ocamlc ${SOURCES_SINGLE_DOMAIN}
	@echo '"Error" static analysis'

.PHONY : iss
iss :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/01-Initialization-Simple-Sign/avalues.ml avalues.ml
#	ocamlc -i ${SOURCES_SINGLE_DOMAIN} # to print types
	ocamlc ${SOURCES_SINGLE_DOMAIN}
	@echo '"Initialization and simple sign" static analysis'
```



```

.PHONY : int
int :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/03-Intervals/avalues.ml avalues.ml
#	ocamlc -i ${SOURCES_SINGLE_DOMAIN} # to print types
	ocamlc ${SOURCES_SINGLE_DOMAIN}
	@echo '"Interval" static analysis'

.PHONY : par
par :
	ocamlyacc parser.mly
	ocamllex lexer.mll || true
	@/bin/rm -f avalues.ml
	@ln -s ../Non-Relational/04-Parity/avalues.ml avalues.ml
#	ocamlc -i ${SOURCES_SINGLE_DOMAIN} # to print types
	ocamlc ${SOURCES_SINGLE_DOMAIN}
	@echo '"Parity" static analysis'
```

```
.PHONY :  err-int
err-int :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues1.mli || true
	@ln -s avalues.mli avalues1.mli
	@/bin/rm -f avalues1.ml || true
	@ln -s ../Non-Relational/02-Errors/avalues.ml avalues1.ml
	@/bin/rm -f avalues2.mli || true
	@ln -s avalues.mli avalues2.mli
	@/bin/rm -f avalues2.ml || true
	@ln -s ../Non-Relational/03-Intervals/avalues.ml avalues2.ml
	@/bin/rm -f red12.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-Errors-Intervals12.ml red12.ml
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/avalues12.ml avalues.ml
#	ocamlc -i ${SOURCES_BINARY_REDUCED_PRODUCT} # to print types
	ocamlc ${SOURCES_BINARY_REDUCED_PRODUCT}
```



```
	@echo 'Reduced "error" and "interval" static analysis'

.PHONY :  iss-int
iss-int :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues1.mli || true
	@ln -s avalues.mli avalues1.mli
	@/bin/rm -f avalues1.ml || true
	@ln -s ../Non-Relational/01-Initialization-Simple-Sign/avalues.ml avalues1.ml
	@/bin/rm -f avalues2.mli || true
	@ln -s ../Non-Relational/03-Intervals/avalues.mli avalues2.mli
	@/bin/rm -f avalues2.ml || true
	@ln -s ../Non-Relational/03-Intervals/avalues.ml avalues2.ml
	@/bin/rm -f red12.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-ISS-Intervals12.ml red12.ml
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/avalues12.ml avalues.ml
#	ocamlc -i ${SOURCES_BINARY_REDUCED_PRODUCT} # to print types
	ocamlc ${SOURCES_BINARY_REDUCED_PRODUCT}
```



```
	@echo 'Reduced "initialization and simple sign" and "interval" static analysis'

.PHONY :  par-int
par-int :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues1.mli || true
	@ln -s avalues.mli avalues1.mli
	@/bin/rm -f avalues1.ml || true
	@ln -s ../Non-Relational/04-Parity/avalues.ml avalues1.ml
	@/bin/rm -f avalues2.mli || true
	@ln -s ../Non-Relational/03-Intervals/avalues.mli avalues2.mli
	@/bin/rm -f avalues2.ml || true
	@ln -s ../Non-Relational/03-Intervals/avalues.ml avalues2.ml
	@/bin/rm -f red12.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-Parity-Intervals12.ml red12.ml
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/avalues12.ml avalues.ml
#	ocamlc -i ${SOURCES_BINARY_REDUCED_PRODUCT} # to print types
	ocamlc ${SOURCES_BINARY_REDUCED_PRODUCT}
```



```
	@echo 'Reduced "parity" and "interval" static analysis'

.PHONY :  par-iss
par-iss :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues1.mli || true
	@ln -s avalues.mli avalues1.mli
	@/bin/rm -f avalues1.ml || true
	@ln -s ../Non-Relational/04-Parity/avalues.ml avalues1.ml
	@/bin/rm -f avalues2.mli || true
	@ln -s avalues.mli avalues2.mli
	@/bin/rm -f avalues2.ml || true
	@ln -s ../Non-Relational/01-Initialization-Simple-Sign/avalues.ml avalues2.ml
	@/bin/rm -f red12.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-Parity-ISS12.ml red12.ml
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/avalues12.ml avalues.ml
#	ocamlc -i ${SOURCES_BINARY_REDUCED_PRODUCT} # to print types
	ocamlc ${SOURCES_BINARY_REDUCED_PRODUCT}
```

```
	@echo 'Reduced "parity" and "Initialization and simple sign" static analysis'

.PHONY :  par-iss-int
par-iss-int :
	ocamlyacc parser.mly
	ocamllex lexer.mll
	@/bin/rm -f avalues1.mli || true
	@ln -s avalues.mli avalues1.mli
	@/bin/rm -f avalues1.ml || true
	@ln -s ../Non-Relational/04-Parity/avalues.ml avalues1.ml
	@/bin/rm -f avalues2.mli || true
	@ln -s avalues.mli avalues2.mli
	@/bin/rm -f avalues2.ml || true
	@ln -s ../Non-Relational/01-Initialization-Simple-Sign/avalues.ml avalues2.ml
	@/bin/rm -f avalues3.mli || true
	@ln -s ../Non-Relational/03-Intervals/avalues.mli avalues3.mli
	@/bin/rm -f avalues3.ml || true
	@ln -s ../Non-Relational/03-Intervals/avalues.ml avalues3.ml
	@/bin/rm -f red12.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-Parity-ISS12.ml red12.ml
```



```
	@/bin/rm -f red23.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-ISS-Intervals23.ml red23.ml
	@/bin/rm -f red13.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/red-Parity-Intervals13.ml red13.ml
	@/bin/rm -f avalues.ml || true
	@ln -s ../Non-Relational/05-Prod-Red/avalues123.ml avalues.ml
#	ocamlc -i ${SOURCES_TERNARY_REDUCED_PRODUCT} # to print types
	ocamlc ${SOURCES_TERNARY_REDUCED_PRODUCT}
	@echo 'Reduced "parity", "initialization and simple sign" and "interval" static ana

include ${EXAMPLES}/makefile

.PHONY : clean
clean :
	-@/bin/rm -f *.cmi *.cmo *~ a.out lexer.ml parser.mli parser.ml || true
	-@/bin/rm -f examples/*~ ../Non-Relational/*~ trace_* || true
	-@/bin/rm -f ../Non-Relational/*/*~ || true
	@echo "Remove auxiliary files"

.PHONY : reset
```



```
reset :
	-@/bin/rm -f abexp.ml acom.ml aenv.ml avalues.ml avalues.mli avalues1.ml avalues1.m
	@echo "Remove instanciated files"
```


Examples of instantiation of the generic forward static analyzer

```
 % pwd
.../Generic-FW-Abstract-Interpreter
 % make reset
Remove instanciated files
 % make notrace
Tracing mode off
 % make nr
"Non-relational" static analysis
 % make fbrbool
Forward/backward analysis of boolean expressions with reduction
 % make fbassign
Forward/backward analysis of assignments
```

```
 % make err
"Error" static analysis
 % ./a.out ../Examples/example25.sil
{ x:{_O_i}; y:{_O_i}; z:{_O_i} }
0:
  x := 0;
  y := ?;
  if ((x + y) = 0) then
      z := (x + y)
  else {(((x + y) < 0) | (0 < (x + y)))}
      z := 0
  fi
7:
{ x:{}; y:{}; z:{_O_a} }
```



```
 % make int
"Interval" static analysis
 % ./a.out ../Examples/example25.sil
{ x:[]; y:[]; z:[] }
0:
  x := 0;
  y := ?;
  if ((x + y) = 0) then
      z := (x + y)
  else {(((x + y) < 0) | (0 < (x + y)))}
      z := 0
  fi
7:
{ x:[0,0]; y:[min_int,max_int]; z:[0,0] }
 % make clean
 %
```


Bibliography

[5] P. Cousot. "The Calculational Design of a Generic Abstract Interpreter". In M. Broy and R. Steinbrüggen (eds.): *Calculational System Design*. NATO ASI Series F. Amsterdam: IOS Press, 1999.

[6] P. Cousot. "The Marktoberdorf'98 Generic Abstract Interpreter". http://www.di.ens.fr/~cousot/Marktoberdorf98.shtml.

THE END

My MIT web site is http://www.mit.edu/~cousot/

The course web site is http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/.