The Calculational Design of a Generic Abstract Interpreter

P. Cousot

DMI - École Normale Supérieure - Paris
45, rue d'Ulm, 75230 Paris cedex 05, France
<cousot@dmi.ens.fr> <http://www.dmi.ens.fr/~cousot>

Introduction

- Explain the *calculation-based development of an abstract interpreter* for the automatic static analysis of a simple imperative language [1, 2];
- A static analyzer takes as input a program written in a given programming language (or a family thereof) and statically, automatically and in finite time [1] outputs an approximate description of all its possible runtime behaviors considered in all possible execution environments (e.g. for all possible input data).

[1] from a few seconds for small programs to a few hours for very large programs;

Values and their Properties

Machine integers

$$\begin{array}{lll} \texttt{max\_int} > 9, & & \text{greatest machine integer;} \\ \texttt{min\_int} \stackrel{\Delta}{=} -\texttt{max\_int} - 1, & & \text{smallest machine integer;} \\ z \in \mathbb{Z}, & & \text{mathematical integers;} \\ i \in \mathbb{I} \stackrel{\Delta}{=} [\texttt{min\_int}, \texttt{max\_int}], & & \text{bounded machine integers.} \end{array} \quad (1)$$

Errors

$$
\begin{array}{lll}
\Omega_{\mathtt{i}}, & & \text{initialization error;} \\
\Omega_{\mathtt{a}}, & & \text{arithmetic error;} \\
e \in \mathbb{E} & \triangleq \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\}, & \text{errors;} \\
v \in \mathbb{I}_\Omega & \triangleq \mathbb{I} \cup \mathbb{E}, & \text{machine values.}
\end{array} \tag{2}
$$

Concrete Properties of Values

- A value property is understood as the set of values which have this property;
- Exemples:
  - $[1, \mathtt{max\_int}] \in \wp(\mathbb{I}_\Omega)$: "is a positive machine integer"
  - $\{2n + 1 \in \mathbb{I} \mid n \in \mathbb{Z}\}$: "is an odd machine integer".
- $\langle \wp(\mathbb{I}_\Omega), \subseteq, \emptyset, \mathbb{I}_\Omega, \cup, \cap, \neg \rangle$ is a complete boolean lattice [2].

[2] $\subseteq$ is logical implication, $\emptyset$ is false, $\mathbb{I}_\Omega$ is true, $\cup$ is the disjunction, $\cap$ is the conjunction and $\neg$ is the negation.

Abstract Properties of Values

Galois connection based abstraction

- The set L of abstract properties is a machine encoding of a subset of all possible value properties;
- $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is a complete lattice for the approximation ordering $\sqsubseteq$ [3];
- Any abstract property $p \in L$ is the machine encoding of some value property $\gamma(p) \in \wp(\mathbb{I}_\Omega)$ specified by the concretization function $\gamma \in L \mapsto \wp(\mathbb{I}_\Omega)$;
- The concretization function is monotone [4]:

$$p \sqsubseteq q \implies \gamma(p) \subseteq \gamma(q) \; . \tag{3}$$

[3] The partial ordering $\sqsubseteq$ is understood as abstract logical implication, the infimum $\bot$ encodes false, the supremum $\top$ encodes true, the lub $\sqcup$ is the abstract disjunction and the glb $\sqcap$ is the abstract conjunction.

[4] This formalizes the fact that the approximation ordering $\sqsubseteq$ should encode logical implication on abstract properties

- The best approximation of concrete properties P in L is provided by the abstraction function $\alpha \in \wp(\mathbb{I}_\Omega) \mapsto L$:
  - α preserves implication:
$$P \subseteq Q \implies \alpha(P) \sqsubseteq \alpha(Q); \tag{4}$$
  - $\alpha(P)$ overapproximates P:
$$\forall P \in \wp(\mathbb{I}_\Omega) : P \subseteq \gamma(\alpha(P)); \tag{5}$$
  - γ introduces no loss of information:
$$\forall p \in \alpha(\gamma(p)) \sqsubseteq p. \tag{6}$$

Galois connections

The conjunction of properties (3) to (6) is equivalent to:

$$\forall P \in \wp(\mathbb{I}_\Omega), p \in L : \alpha(P) \sqsubseteq p \iff P \subseteq \gamma(p)\ . \tag{7}$$

denoted:

$$\langle \wp(\mathbb{I}_\Omega),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L,\ \sqsubseteq \rangle\ . \tag{8}$$

Componentwise abstraction of sets of pairs

$$\begin{aligned} \alpha^2(P) &\triangleq \langle \alpha(\{v_1 \mid \exists v_2 : \langle v_1,\ v_2\rangle \in P\}) \\ &\qquad \alpha(\{v_2 \mid \exists v_1 : \langle v_1,\ v_2\rangle \in P\})\rangle, \qquad (9)\\ \gamma^2(\langle p_1,\ p_2\rangle) &\triangleq \{\langle v_1,\ v_2\rangle \mid v_1 \in \gamma(p_1) \wedge v_2 \in \gamma(p_2)\} \qquad (10) \end{aligned}$$

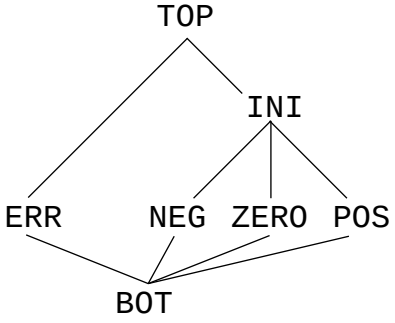
so that:

$$\langle \wp(\mathbb{I}_\Omega \times \mathbb{I}_\Omega),\ \subseteq \rangle \underset{\alpha^2}{\overset{\gamma^2}{\leftrightarrows}} \langle L \times L,\ \sqsubseteq^2 \rangle \tag{11}$$

with the componentwise ordering:

$$\langle p_1,\ p_2\rangle \sqsubseteq^2 \langle q_1,\ q_2\rangle \triangleq p_1 \sqsubseteq q_1 \wedge p_2 \sqsubseteq q_2\ .$$

The lattice of initialization and simple signs

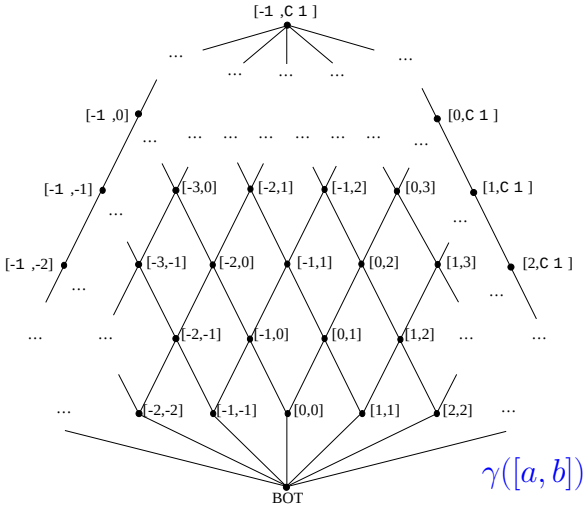


$$\begin{aligned} \gamma(\texttt{TOP}) &\triangleq \mathbb{I}_\Omega, \\ \gamma(\texttt{INI}) &\triangleq \mathbb{I} \cup \{\Omega_\texttt{a}\}, \\ \gamma(\texttt{ERR}) &\triangleq \{\Omega_\texttt{i}, \Omega_\texttt{a}\}, \\ \gamma(\texttt{ZERO}) &\triangleq \{0, \Omega_\texttt{a}\}, \\ \gamma(\texttt{NEG}) &\triangleq [\texttt{min\_int}, -1] \cup \{\Omega_\texttt{a}\}, \\ \gamma(\texttt{POS}) &\triangleq [1, \texttt{max\_int}] \cup \{\Omega_\texttt{a}\} \\ \gamma(\texttt{BOT}) &\triangleq \{\Omega_\texttt{a}\}\ . \end{aligned} \tag{12}$$

Initialization and simple sign abstraction

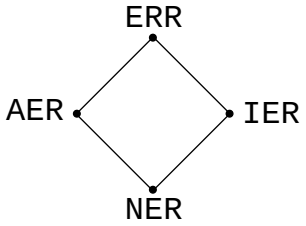
$$\begin{aligned}\alpha(P) \triangleq\ & (P \subseteq \{\Omega_{\mathtt{a}}\}\ ?\ \mathtt{BOT}\\ & \mid P \subseteq [\mathtt{min\_int}, -1] \cup \{\Omega_{\mathtt{a}}\}\ ?\ \mathtt{NEG}\\ & \mid P \subseteq \{0, \Omega_{\mathtt{a}}\}\ ?\ \mathtt{ZERO}\\ & \mid P \subseteq [1, \mathtt{max\_int}] \cup \{\Omega_{\mathtt{a}}\}\ ?\ \mathtt{POS}\\ & \mid P \subseteq \mathbb{I} \cup \{\Omega_{\mathtt{a}}\}\ ?\ \mathtt{INI}\\ & \mid P \subseteq \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\}\ ?\ \mathtt{ERR}\\ & \text{¿}\ \mathtt{TOP})\ .\end{aligned} \tag{13}$$

The lattice I of intervals



$$\gamma(\mathtt{BOT}) \triangleq \emptyset,$$
$$\gamma([a,b]) \triangleq \{x \in \mathbb{I} \mid a \leq x \leq b\}\ .$$

The lattice E of errors



$$\gamma(\mathtt{NER}) \triangleq \mathbb{I} \tag{14}$$
$$\gamma(\mathtt{IER}) \triangleq \mathbb{I} \cup \{\Omega_{\mathtt{i}}\} \tag{15}$$
$$\gamma(\mathtt{AER}) \triangleq \mathbb{I} \cup \{\Omega_{\mathtt{a}}\} \tag{16}$$
$$\gamma(\mathtt{ERR}) \triangleq \mathbb{I} \cup \{\Omega_{\mathtt{a}}, \Omega_{\mathtt{i}}\} \tag{17}$$

Combining interval and error information

The abstract domain is:

$$L \triangleq E \times I$$

with the following meaning:

$$\gamma(\langle e,\ i\rangle) \triangleq \gamma(e) \cap \gamma(i)\ .$$

Algebra of abstract properties of values [5, 6, 7]

```
module type Abstract_Lattice_Algebra_signature =
  sig
    type lat                              (* abstract properties    *)
    val bot         : unit -> lat         (* infimum                *)
    val isbotempty : unit -> bool         (* bottom is emptyset?    *)
    val initerr     : unit -> lat         (* uninitialization       *)
    val top         : unit -> lat         (* supremum               *)
    val join        : lat -> lat -> lat   (* least upper bound      *)
    val meet        : lat -> lat -> lat   (* greatest lower bound   *)
    val leq         : lat -> lat -> bool  (* approximation ordering *)
    val eq          : lat -> lat -> bool  (* equality               *)
    val in_errors   : lat -> bool         (* included in errors?    *)
    val print       : lat -> unit
    ...
  end;;
```


Environments

[5] The programming language is Objective CAML.
[6] `(isbotempty ())` is $\gamma(\bot) = \emptyset$.
[7] `(in_errors v)` implies $\gamma(\mathtt{v}) \subseteq \{\Omega_a, \Omega_1\}$.

Concrete environments

As usual, we use environments ρ to record the value $\rho(\mathtt{X})$ of program variables $\mathtt{X} \in \mathbb{V}$.

$$\rho \in \mathbb{R} \stackrel{\Delta}{=} \mathbb{V} \mapsto \mathbb{I}_\Omega, \qquad \text{environments.}$$

Since environments are functions, we can use the functional assignment/substitution notation defined as $(f \in D \mapsto E)$

$$\begin{aligned} f[d \leftarrow e](x) &\stackrel{\Delta}{=} f(x), \qquad \text{if } x \neq d\ ; \\ f[d \leftarrow e](d) &\stackrel{\Delta}{=} e\ ; \\ f[d_1 \leftarrow e_1; d_2 \leftarrow e_2; \ldots; d_n \leftarrow e_n] &\stackrel{\Delta}{=} \\ &(f[d_1 \leftarrow e_1])[d_2 \leftarrow e_2; \ldots; d_n \leftarrow e_n]\ . \end{aligned} \tag{18}$$

Properties of concrete environments

- Properties of environments are understood as sets of environments that is elements of $\wp(\mathbb{R})$ where $\subseteq$ is logical implication
- Such properties of environments are usually stated using predicates in some prescribed syntactic form;
- Environment properties are therefore their interpretations;
- Example: "$\mathtt{X} = \mathtt{Y}$" is interpreted as $\{\rho \in \mathbb{V} \mapsto \mathbb{I}_\Omega \mid \rho(\mathtt{X}) = \rho(\mathtt{Y})\}$.

Nonrelational abstraction of environment properties

Ignore relationships between the possible values of variables:

$$\langle \wp(\mathbb{V} \mapsto \mathbb{I}_\Omega),\ \subseteq \rangle \underset{\alpha_r}{\overset{\gamma_r}{\leftrightarrows}} \langle \mathbb{V} \mapsto \wp(\mathbb{I}_\Omega),\ \dot{\subseteq} \rangle$$

by defining:

$$\begin{aligned} \alpha_r(R) &= \lambda \mathtt{X} \in \mathbb{V} \cdot \{\rho(\mathtt{X}) \mid \rho \in R\}, \\ \gamma_r(r) &= \{\rho \mid \forall \mathtt{X} \in \mathbb{V} : \rho(\mathtt{X}) \in r(\mathtt{X})\} \end{aligned}$$

and the pointwise ordering which is denoted with the dot notation:

$$r \mathrel{\dot{\subseteq}} r' \triangleq \forall \mathtt{X} \in \mathbb{V} : r(\mathtt{X}) \subseteq r'(\mathtt{X})\ .$$

Example of nonrelational abstraction of environment properties

- If

  $$R = \{[\mathtt{X} \mapsto 1; \mathtt{Y} \mapsto 1], [\mathtt{X} \mapsto 2; \mathtt{Y} \mapsto 2]\}$$

  then

  $$\alpha_r(R) = [\mathtt{X} \mapsto \{1, 2\}; \mathtt{Y} \mapsto \{1, 2\}]$$

  so that the equality information $(\mathtt{X} = \mathtt{Y})$ is lost.
- Since all possible relationships between variables are lost in the nonrelational abstraction, such nonrelational analyzes often lack precision, but are rather efficient.

Environment properties abstraction

The Galois connection (8)

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

can be used to approximate the codomain

$$\langle \mathbb{V} \mapsto \wp(\mathbb{I}_\Omega),\ \dot{\subseteq} \rangle \underset{\alpha_c}{\overset{\gamma_c}{\leftrightarrows}} \langle \mathbb{V} \mapsto L,\ \dot{\sqsubseteq} \rangle$$

as follows

$$\begin{aligned} r \mathrel{\dot{\sqsubseteq}} r' &\triangleq \forall \mathtt{X} \in \mathbb{V} : r(\mathtt{X}) \sqsubseteq r'(\mathtt{X}), \\ \alpha_c(R) &\triangleq \alpha \circ R, \\ \gamma_c(r) &\triangleq \gamma \circ r, \end{aligned}$$

so that $\langle \mathbb{V} \mapsto L,\ \dot{\sqsubseteq},\ \dot{\bot},\ \dot{\top},\ \sqcup,\ \dot{\sqcap} \rangle$ is a complete lattice for the pointwise ordering $\dot{\sqsubseteq}$.

Composition of Galois connections

The composition of Galois connections

$$\langle L_1,\ \sqsubseteq_1 \rangle \underset{\alpha_{21}}{\overset{\gamma_{12}}{\leftrightarrows}} \langle L_2,\ \sqsubseteq_2 \rangle \qquad \text{and} \qquad \langle L_2,\ \sqsubseteq_2 \rangle \underset{\alpha_{32}}{\overset{\gamma_{23}}{\leftrightarrows}} \langle L_3,\ \sqsubseteq_3 \rangle$$

is a Galois connection

$$\langle L_1,\ \sqsubseteq_1 \rangle \underset{\alpha_{32} \circ \alpha_{21}}{\overset{\gamma_{12} \circ \gamma_{23}}{\leftrightarrows}} \langle L_3,\ \sqsubseteq_3 \rangle\ .$$

Composition of nonrelational and codomain abstractions

$$\langle \wp(\mathbb{V} \mapsto \mathbb{I}_\Omega),\ \subseteq \rangle \underset{\dot{\alpha}}{\overset{\dot{\gamma}}{\leftrightarrows}} \langle \mathbb{V} \mapsto L,\ \dot{\sqsubseteq} \rangle \tag{19}$$

where

$$\begin{aligned} \dot{\alpha}(R) &\stackrel{\Delta}{=} \alpha_c \circ \alpha_r(R) \\ &= \lambda \mathtt{X} \in \mathbb{V} \cdot \alpha(\{\rho(\mathtt{X}) \mid \rho \in R\}), \qquad (20) \\ \dot{\gamma}(r) &\stackrel{\Delta}{=} \gamma_r \circ \gamma_c(r) \\ &= \{\rho \mid \forall \mathtt{X} \in \mathbb{V} : \rho(\mathtt{X}) \in \gamma(r(\mathtt{X}))\}\ . \qquad (21) \end{aligned}$$

Reduced composition of nonrelational and codomain abstractions

- If L has an infimum $\bot$ such that $\gamma(\bot) = \emptyset$, we observe that if $r \in \mathbb{V} \mapsto L$ has $\rho(\mathtt{X}) = \bot$ then $\dot{\gamma}(r) = \emptyset$;
- It follows that the abstract environments with some bottom component all represent the same concrete information ($\emptyset$);
- The abstract lattice can then be reduced to eliminate equivalent abstract environments (i.e. which have the same meaning) [4, 6].

$$\langle \mathbb{V} \mapsto \mathbb{I}_\Omega,\ \dot{\subseteq} \rangle \underset{\dot{\alpha}}{\overset{\dot{\gamma}}{\leftrightarrows}} \langle \mathbb{V} \overset{\bot}{\longmapsto} L,\ \dot{\sqsubseteq} \rangle \tag{22}$$

where

$$\mathbb{V} \overset{\bot}{\longmapsto} L \stackrel{\Delta}{=} \{\rho \in \mathbb{V} \mapsto L \mid \forall \mathtt{X} \in \mathbb{V} : \rho(\mathtt{X}) \neq \bot\} \cup \{\lambda \mathtt{X} \in \mathbb{V} \cdot \bot\}\ .$$

Algebra of abstract environments

```
module type Abstract_Env_Algebra_signature =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  sig
    open Abstract_Syntax
    type env        (* complete lattice of abstract environments   *)
    type element = env
    val bot     : unit -> env            (* infimum                 *)
    val is_bot  : env  -> bool           (* check for infimum       *)
    val initerr : unit -> env            (* uninitialization        *)
    val top     : unit -> env            (* supremum                *)
    val join    : env  -> (env -> env)   (* least upper bound       *)
    val meet    : env  -> (env -> env)   (* greatest lower bound    *)
    val leq     : env  -> (env -> bool)  (* approximation ordering *)
    val eq      : env  -> (env -> bool)  (* equality                *)
    val print   : env  -> unit
    (* substitution *)
    val get     : env  -> variable -> L.lat        (* r(X)         *)
    val set     : env  -> variable -> L.lat -> env (* r[X <- v]    *)
  end;;
```


Semantics of Arithmetic Expressions

Abstract syntax of arithmetic expressions

Numbers

$\mathtt{d} \in \text{Digit} ::= 0 \mid 1 \mid \ldots \mid 9$ digits,

$\mathtt{n} \in \text{Nat} ::= \text{Digit} \mid \text{Nat}\ \text{Digit}$ numbers in decimal notation.

Variables

$\mathtt{X} \in \mathbb{V}$ variables/identifiers.

Arithmetic expressions

$A \in \text{Aexp} ::= \mathtt{n}$ numbers,

$\mid \mathtt{X}$ variables,

$\mid\ ?$ random machine integer,

$\mid\ +A \mid -A$ unary operators,

$\mid\ A_1 + A_2 \mid A_1 - A_2$ binary operators,

$\mid\ A_1 * A_2 \mid A_1 \,/\, A_2$

$\mid\ A_1 \,\mathtt{mod}\, A_2$.

Machine arithmetics

$$\begin{aligned}
\underline{\mathtt{d}} &\triangleq \mathtt{d}; \\
\underline{\mathtt{nd}} &\triangleq \Omega_{\mathtt{a}}, && \text{if } 10\,\underline{\mathtt{n}} + \mathtt{d} > \mathtt{max_int}; \\
\underline{\mathtt{nd}} &\triangleq 10\,\underline{\mathtt{n}} + \mathtt{d}, && \text{if } 10\,\underline{\mathtt{n}} + \mathtt{d} \leq \mathtt{max_int}\ ; \\
\\
\underline{\mathtt{u}}\,\Omega_e &\triangleq \Omega_e; \\
\underline{\mathtt{u}}\,i &\triangleq \mathtt{u}\,i, && \text{if } \mathtt{u}\,i \in \mathbb{I}; \\
\underline{\mathtt{u}}\,i &\triangleq \Omega_{\mathtt{a}}, && \text{if } \mathtt{u}\,i \notin \mathbb{I}\ .
\end{aligned} \tag{23}$$

Machine arithmetics (continued)

$$\begin{aligned}
\Omega_e\,\underline{\mathtt{b}}\,v &\triangleq \Omega_e; \\
i\,\underline{\mathtt{b}}\,\Omega_e &\triangleq \Omega_e; \\
i_1\,\underline{\mathtt{b}}\,i_2 &\triangleq i_1\,\mathtt{b}\,i_2, && \text{if } \mathtt{b} \in \{+,-,*\} \wedge i_1\,\mathtt{b}\,i_2 \in \mathbb{I}; \\
i_1\,\underline{\mathtt{b}}\,i_2 &\triangleq i_1\,\mathtt{b}\,i_2, && \text{if } \mathtt{b} \in \{/,\mathtt{mod}\} \wedge i_1 \in \mathbb{I} \cap \mathbb{N} \wedge \\
& && \quad i_2 \in \mathbb{I} \cap \mathbb{N}^+ \wedge i_1\,\mathtt{b}\,i_2 \in \mathbb{I}; \\
i_1\,\underline{\mathtt{b}}\,i_2 &\triangleq \Omega_{\mathtt{a}}, && \text{if } i_1\,\mathtt{b}\,i_2 \notin \mathbb{I} \vee (\mathtt{b} \in \{/,\mathtt{mod}\} \wedge \\
& && \quad (i_1 \notin \mathbb{I} \cap \mathbb{N} \vee i_2 \notin \mathbb{I} \cap \mathbb{N}^+))\ .
\end{aligned} \tag{24}$$

Operational semantics of arithmetic expressions

$$\rho \vdash \mathtt{n} \Mapsto \underline{\mathtt{n}}, \quad \text{decimal numbers;} \tag{25}$$

$$\rho \vdash \mathtt{X} \Mapsto \rho(\underline{\mathtt{X}}), \quad \text{variables;} \tag{26}$$

$$\frac{i \in \mathbb{I}}{\rho \vdash\ ? \Mapsto i}, \quad \text{random;} \tag{27}$$

$$\frac{\rho \vdash A \Mapsto v}{\rho \vdash \mathtt{u}\,A \Mapsto \underline{\mathtt{u}}\,v}, \quad \text{unary arithmetic operations;}^{8} \tag{28}$$

$$\frac{\rho \vdash A_1 \Mapsto v_1,\ \rho \vdash A_2 \Mapsto v_2}{\rho \vdash A_1\,\mathtt{b}\,A_2 \Mapsto v_1\,\underline{\mathtt{b}}\,v_2}, \quad \text{binary arithmetic operations .} \tag{29}$$

[8] Observe that if $\mathtt{m}$ and $\mathtt{M}$ are the strings of digits respectively representing the absolute value of `min_int` and `max_int` then $\underline{\mathtt{m}} > \mathtt{max\_int}$ so that $\rho \vdash \mathtt{m} \Mapsto \Omega_{\mathtt{a}}$ whence $\rho \vdash \mathtt{-\ m} \Mapsto \Omega_{\mathtt{a}}$. However $\rho \vdash \mathtt{(-\ M)\ -\ 1} \Mapsto \mathtt{min\_int}$.

Forward collecting semantics of arithmetic expressions

- Defines the possible values that the arithmetic expression can evaluate to in a given set of environments

$$\begin{aligned} &\mathrm{Faexp} \in \mathrm{Aexp} \mapsto \wp(\mathbb{R}) \stackrel{\text{cjm}}{\longmapsto} \wp(\mathbb{I}_\Omega), \\ &\mathrm{Faexp}[\![A]\!]R \stackrel{\Delta}{=} \{v \mid \exists \rho \in R : \rho \vdash A \mapsto v\} \,. \end{aligned} \tag{30}$$

- $\mathrm{Faexp}[\![A]\!]R$ specifies the strongest postcondition that values of the arithmetic expression A satisfy when this expression A is evaluated in an environment satisfying the precondition R [9].

Properties of the forward collecting semantics of arithmetic expressions

- $\mathrm{Faexp}[\![A]\!]R$ is a complete join morphism [10], that is ($\mathcal{S}$ is an arbitrary set)

$$\mathrm{Faexp}[\![A]\!]\left(\bigcup_{k\in\mathcal{S}} R_k\right) = \bigcup_{k\in\mathcal{S}} (\mathrm{Faexp}[\![A]\!]R_k) \,,$$

  which implies that
  - $\mathrm{Faexp}[\![A]\!]R$ is monotone (when $\mathcal{S} = \{1, 2\}$ and $R_1 \subseteq R_2$)
  - $\mathrm{Faexp}[\![A]\!]R$ is $\emptyset$-strict (when $\mathcal{S} = \emptyset$)

$$\mathrm{Faexp}[\![A]\!]\emptyset = \emptyset \,.$$

[9] The forward collecting semantics can therefore be understood as a predicate transformer [8].

[10] denoted with $\stackrel{\text{cjm}}{\longmapsto}$

Backward collecting semantics of arithmetic expressions

- The *backward/top-down collecting semantics* $\mathrm{Baexp}[\![A]\!](R)P$ of an arithmetic expression A defines the subset of possible environments R such that the arithmetic expression may evaluate, without producing a runtime error, to a value belonging to given set P;

$$\begin{aligned} &\mathrm{Baexp} \in \mathrm{Aexp} \mapsto \wp(\mathbb{R}) \stackrel{\text{cjm}}{\longmapsto} \wp(\mathbb{I}_\Omega) \stackrel{\text{cjm}}{\longmapsto} \wp(\mathbb{R}), \\ &\mathrm{Baexp}[\![A]\!](R)P \stackrel{\Delta}{=} \{\rho \in R \mid \exists i \in P \cap \mathbb{I} : \rho \vdash A \mapsto i\} \,. \end{aligned} \tag{31}$$

Abstract Interpretation of Arithmetic Expressions

Lifting Galois connections at higher-order

Functional abstraction [4]

$$\begin{aligned} \alpha^{\dot{}}(\Phi) &\stackrel{\Delta}{=} \alpha \circ \Phi \circ \dot{\gamma}, \\ \gamma^{\dot{}}(\varphi) &\stackrel{\Delta}{=} \gamma \circ \varphi \circ \dot{\alpha} \end{aligned} \tag{32}$$

so that

$$\langle \wp(\mathbb{V} \mapsto \mathbb{I}_\Omega) \stackrel{\text{mon}}{\longmapsto} \wp(\mathbb{I}_\Omega),\ \dot{\subseteq} \rangle \underset{\alpha^{\dot{}}}{\overset{\gamma^{\dot{}}}{\leftrightarrows}} \langle (\mathbb{V} \mapsto L) \stackrel{\text{mon}}{\longmapsto} L,\ \dot{\sqsubseteq} \rangle \ . \tag{33}$$

The best approximation

For any abstract precondition $p \in L$, or its concrete equivalent $\dot{\gamma}(p) \in \wp(\mathbb{V} \mapsto \mathbb{I}_\Omega)$, the abstract predicate transformer φ should provide an overestimate $\varphi(p)$ of the postcondition $\Phi(\gamma(p))$ defined by the concrete predicate transformer Φ:

$$\begin{array}{lll} & \forall p \in L : \gamma(\varphi(p)) \supseteq \Phi(\dot{\gamma}(p)) & \text{⟅soundness requirement⟆} \\ \iff & \forall p \in L : \Phi(\dot{\gamma}(p)) \subseteq \gamma(\varphi(p)) & \text{⟅def. inverse } \supseteq \text{ of } \subseteq\text{⟆} \\ \iff & \forall p \in L : \alpha(\Phi(\dot{\gamma}(p))) \sqsubseteq \varphi(p) & \text{⟅def. Galois connection⟆} \\ \iff & \alpha \circ \Phi \circ \dot{\gamma} \mathrel{\dot{\sqsubseteq}} \varphi & \text{⟅def. } \dot{\sqsubseteq}\text{⟆} \\ \iff & \alpha^{\dot{}}(\Phi) \mathrel{\dot{\sqsubseteq}} \varphi & \text{⟅def. } \alpha^{\dot{}}\text{⟆.} \end{array} \tag{34}$$

Choosing $\varphi \stackrel{\Delta}{=} \alpha^{\dot{}}(\Phi)$ is therefore the best of the possible sound choices since it always provides the strongest abstract postcondition, whence, by monotony, the strongest concrete one.

Generic forward/top-down abstract interpretation of arithmetic expressions

$$\begin{aligned} \text{Faexp}^{\dot{}} &\in \text{Aexp} \mapsto (\mathbb{V} \longmapsto L) \stackrel{\text{mon}}{\longmapsto} L, && \text{when } \gamma(\bot) \neq \emptyset; \\ \text{Faexp}^{\dot{}} &\in \text{Aexp} \mapsto (\mathbb{V} \stackrel{\bot}{\longmapsto} L) \stackrel{\text{mon}}{\longmapsto} L, && \text{when } \gamma(\bot) = \emptyset \end{aligned}$$

Design by calculus. Starting from the formal specification $\alpha^{\dot{}}(\text{Faexp}[\![A]\!])$, we derive an algorithm $\text{Faexp}^{\dot{}}[\![A]\!]$ satisfying:

$$\text{Faexp}^{\dot{}}[\![A]\!] \mathrel{\dot{\sqsupseteq}} \alpha^{\dot{}}(\text{Faexp}[\![A]\!]) \ . \tag{35}$$

$$\begin{array}{ll} & \alpha^{\dot{}}(\text{Faexp}[\![A]\!]) \\ = & \quad \text{⟅def. (32) of } \alpha^{\dot{}}\text{⟆} \\ & \alpha \circ \text{Faexp}[\![A]\!] \circ \dot{\gamma} \\ = & \quad \text{⟅def. of composition } \circ\text{⟆} \\ & \lambda r \cdot \alpha(\text{Faexp}[\![A]\!](\dot{\gamma}(r))) \\ = & \quad \text{⟅def. (30) of Faexp}[\![A]\!]\text{⟆} \\ & \lambda r \cdot \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash A \Rightarrow v\}) \ . \end{array}$$

- If r is the infimum $\lambda \mathtt{Y} \cdot \bot$ where the infimum $\bot$ of L is such that $\gamma(\bot) = \emptyset$, then $\dot{\gamma}(r) = \emptyset$ whence:

$$
\begin{array}{ll}
& \alpha^\flat(\mathrm{Faexp}[\![A]\!])(\lambda \mathtt{Y} \cdot \bot) \\
= & \qquad \wr \text{def. (21) of } \dot{\gamma} \int \\
& \alpha(\emptyset) \\
= & \qquad \wr \text{Galois connection (8) so that } \alpha(\emptyset) = \bot \int \\
& \bot \ .
\end{array}
$$

- When $r \neq \lambda \mathtt{Y} \cdot \bot$ or $\gamma(\bot) \neq \emptyset$, we have

$$
\begin{array}{ll}
& \alpha^\flat(\mathrm{Faexp}[\![A]\!])r \\
= & (\lambda r \cdot \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash A \mapsto v\}))r \\
= & \qquad \wr \text{def. lambda expression} \int \\
& \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash A \mapsto v\})
\end{array}
$$

and we proceed by induction on the arithmetic expression A.

- When $A = \mathtt{n} \in \mathrm{Nat}$ is a number, we have

$$
\begin{array}{ll}
& \alpha^\flat(\mathrm{Faexp}[\![\mathtt{n}]\!])r \\
= & \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash \mathtt{n} \mapsto v\}) \\
= & \qquad \wr \text{def. (25) of } \rho \vdash \mathtt{n} \mapsto v \int \\
& \alpha(\{\underline{\mathtt{n}}\}) \\
= & \qquad \wr \text{by defining } \mathtt{n} = \alpha(\{\underline{\mathtt{n}}\}) \int \\
& \mathtt{n} \\
= & \qquad \wr \text{by defining } \mathrm{Faexp}^\flat[\![\mathtt{n}]\!]r \stackrel{\Delta}{=} \mathtt{n} \int \\
& \mathrm{Faexp}^\flat[\![\mathtt{n}]\!]r \ .
\end{array}
$$

- When $A = \mathtt{X} \in \mathbb{V}$ is a variable, we have

$$
\begin{array}{ll}
& \alpha^\flat(\mathrm{Faexp}[\![\mathtt{X}]\!])r \\
= & \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash \mathtt{X} \mapsto v\}) \\
= & \qquad \wr \text{def. (26) of } \rho \vdash \mathtt{X} \mapsto v \int \\
& \alpha(\{\rho(\underline{\mathtt{X}}) \mid \rho \in \dot{\gamma}(r)\}) \\
= & \qquad \wr \text{def. (21) of } \dot{\gamma} \int \\
& \alpha(\gamma(r(\mathtt{X}))) \\
\sqsubseteq & \qquad \wr \text{Galois connection (8) so that } \alpha \circ \gamma \text{ is reductive} \int \\
& r(\mathtt{X}) \\
= & \qquad \wr \text{by defining } \mathrm{Faexp}^\flat[\![\mathtt{X}]\!]r \stackrel{\Delta}{=} r(\mathtt{X}) \int \\
& \mathrm{Faexp}^\flat[\![\mathtt{X}]\!]r \ .
\end{array}
$$

- When $A = ?$ is random, we have

$$
\begin{array}{ll}
& \alpha^\flat(\mathrm{Faexp}[\![?]\!])r \\
= & \alpha(\{v \mid \exists \rho \in \dot{\gamma}(r) : \rho \vdash ? \mapsto v\}) \\
= & \qquad \wr \text{def. (27) of } \rho \vdash ? \mapsto v \int \\
& \alpha(\mathbb{I}) \\
\sqsubseteq & \qquad \wr \text{by defining } ?^\flat \sqsupseteq \alpha(\mathbb{I}) \int \\
& ?^\flat \\
= & \qquad \wr \text{by defining } \mathrm{Faexp}^\flat[\![?]\!]r \stackrel{\Delta}{=} ?^\flat \int \\
& \mathrm{Faexp}^\flat[\![?]\!]r \ .
\end{array}
$$

- When $A = \mathtt{u}\,A'$ is a unary operation, we have

$\alpha^\sharp(\mathrm{Faexp}[\![\mathtt{u}\,A']\!])r$
$= \alpha(\{v \mid \exists \rho \in \dot\gamma(r) : \rho \vdash \mathtt{u}\,A' \Mapsto v\})$
$=$ ⟅def. (28) of $\rho \vdash \mathtt{u}\,A' \Mapsto v$⟆
$\alpha(\{\underline{\mathtt{u}}\,v \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A' \Mapsto v\})$
$\sqsubseteq$ ⟅$\gamma \circ \alpha$ is extensive (5), α is monotone (4)⟆
$\alpha(\{\underline{\mathtt{u}}\,v \mid v \in \gamma \circ \alpha(\{v' \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A' \Mapsto v'\})\})$
$\sqsubseteq$ ⟅induction hypothesis (35), γ (3) and α (4) are monotone⟆
$\alpha(\{\underline{\mathtt{u}}\,v \mid v \in \gamma(\mathrm{Faexp}^\sharp[\![A']\!]r)\})$
$\sqsubseteq$ ⟅by defining $\mathtt{u}^\sharp$ such that $\mathtt{u}^\sharp(p) \sqsupseteq \alpha(\{\underline{\mathtt{u}}\,v \mid v \in \gamma(p)\})$⟆
$\mathtt{u}^\sharp(\mathrm{Faexp}^\sharp[\![A']\!]r)$
$=$ ⟅by defining $\mathrm{Faexp}^\sharp[\![\mathtt{u}\,A']\!]r \triangleq \mathtt{u}^\sharp(\mathrm{Faexp}^\sharp[\![A']\!]r)$⟆
$\mathrm{Faexp}^\sharp[\![\mathtt{u}\,A']\!]r$.

- When $A = A_1\,\mathtt{b}\,A_2$ is a binary operation, we have

$\alpha^\sharp(\mathrm{Faexp}[\![A_1\,\mathtt{b}\,A_2]\!])r$
$= \alpha(\{v \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A_1\,\mathtt{b}\,A_2 \Mapsto v\})$
$=$ ⟅def. (29) of $\rho \vdash A_1\,\mathtt{b}\,A_2 \Mapsto v$⟆
$\alpha(\{v_1\,\underline{\mathtt{b}}\,v_2 \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A_1 \Mapsto v_1 \wedge \rho \vdash A_2 \Mapsto v_2\})$
$\sqsubseteq$ ⟅α monotone (4)⟆
$\alpha(\{v_1\,\underline{\mathtt{b}}\,v_2 \mid \exists \rho_1 \in \dot\gamma(r) : \rho_1 \vdash A_1 \Mapsto v_1 \wedge \exists \rho_2 \in \dot\gamma(r) : \rho_2 \vdash A_2 \Mapsto v_2\})$
$\sqsubseteq$ ⟅$\gamma \circ \alpha$ is extensive (5), α is monotone (4)⟆
$\alpha(\{v_1\,\underline{\mathtt{b}}\,v_2 \mid v_1 \in \gamma \circ \alpha(\{v \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A_1 \Mapsto v\}) \wedge$
$\qquad v_2 \in \gamma \circ \alpha(\{v \mid \exists \rho \in \dot\gamma(r) : \rho \vdash A_2 \Mapsto v\})\})$
$\sqsubseteq$ ⟅induction hypothesis (35), γ (3) and α (4) are monotone⟆
$\alpha(\{v_1\,\underline{\mathtt{b}}\,v_2 \mid v_1 \in \gamma(\mathrm{Faexp}^\sharp[\![A_1]\!]r) \wedge v_2 \in \gamma(\mathrm{Faexp}^\sharp[\![A_2]\!]r)\})$
$\sqsubseteq$ ⟅by defining $\mathtt{b}^\sharp$ such that $\mathtt{b}^\sharp(p_1, p_2) \sqsupseteq \alpha(\{v_1\,\underline{\mathtt{b}}\,v_2 \mid v_1 \in \gamma(p_1) \wedge v_2 \in \gamma(p_2)\})$⟆
$\mathtt{b}^\sharp(\mathrm{Faexp}^\sharp[\![A_1]\!]r, \mathrm{Faexp}^\sharp[\![A_2]\!]r)$
$=$ ⟅by defining $\mathrm{Faexp}^\sharp[\![A_1\,\mathtt{b}\,A_2]\!]r \triangleq \mathtt{b}^\sharp(\mathrm{Faexp}^\sharp[\![A_1]\!]r, \mathrm{Faexp}^\sharp[\![A_2]\!]r)$⟆
$\mathrm{Faexp}^\sharp[\![A_1\,\mathtt{b}\,A_2]\!]r$.

Generic forward abstract interpretation of arithmetic expressions

Forward abstract interpretation of arithmetic expressions

$$\mathrm{Faexp}^\sharp[\![A]\!](\lambda \mathtt{Y}\cdot\bot) \triangleq \bot \qquad \text{if } \gamma(\bot) = \emptyset \quad (36)$$
$$\mathrm{Faexp}^\sharp[\![\mathtt{n}]\!]r \triangleq \mathtt{n}^\sharp$$
$$\mathrm{Faexp}^\sharp[\![\mathtt{X}]\!]r \triangleq r(\mathtt{X})$$
$$\mathrm{Faexp}^\sharp[\![?]\!]r \triangleq ?^\sharp$$
$$\mathrm{Faexp}^\sharp[\![\mathtt{u}\,A']\!]r \triangleq \mathtt{u}^\sharp(\mathrm{Faexp}^\sharp[\![A']\!]r)$$
$$\mathrm{Faexp}^\sharp[\![A_1\,\mathtt{b}\,A_2]\!]r \triangleq \mathtt{b}^\sharp(\mathrm{Faexp}^\sharp[\![A_1]\!]r, \mathrm{Faexp}^\sharp[\![A_2]\!]r)$$

Forward abstract operations

$$\mathtt{n}^\sharp = \alpha(\{\underline{\mathtt{n}}\}) \tag{37}$$
$$\mathtt{u}^\sharp(p) \sqsupseteq \alpha(\{\underline{\mathtt{u}}\, v \mid v \in \gamma(p)\}) \tag{38}$$
$$?^\sharp \sqsupseteq \alpha(\mathbb{I}) \tag{39}$$
$$\mathtt{b}^\sharp(p_1, p_2) \sqsupseteq \alpha(\{v_1\, \underline{\mathtt{b}}\, v_2 \mid v_1 \in \gamma(p_1) \wedge v_2 \in \gamma(p_2)\}) \tag{40}$$

Monotony/continuity

- The abstract semantics $\mathtt{Faexp}^\sharp[\![A]\!]$ of A is monotonic if the abstract operations $\mathtt{u}^\sharp$ and $\mathtt{b}^\sharp$ are monotonic;
- idem for continuity;
- Proof by structural induction on the arithmetic expression A;
- Use the fact that the composition of monotonic (resp. continuous) functions is monotonic (resp. continuous).

Generic forward/top-down static analyzer of arithmetic expressions

```
module type Abstract_Lattice_Algebra_signature =
  sig
    (* complete lattice of abstract properties of values         *)
    type lat                              (* abstract properties    *)
    ...
    (* forward abstract interpretation of arithmetic expressions *)
    val f_INT    : string -> lat
    val f_RANDOM : unit -> lat
    val f_UMINUS : lat -> lat
    val f_UPLUS  : lat -> lat
    val f_PLUS   : lat -> lat -> lat
    val f_MINUS  : lat -> lat -> lat
    val f_TIMES  : lat -> lat -> lat
    val f_DIV    : lat -> lat -> lat
    val f_MOD    : lat -> lat -> lat
    ...
  end;;
```



```
module type Faexp_signature =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  functor (E: Abstract_Env_Algebra_signature) ->
  sig
    open Abstract_Syntax
    (* generic forward abstract interpretation of arithmetic operations *)
    val faexp : aexp ->  E(L).env -> L.lat
  end;;
```

```
module Faexp_implementation =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  functor (E: Abstract_Env_Algebra_signature) ->
  struct
    open Abstract_Syntax
    (* generic abstract environments *)
    module E'=E(L)
    (* generic forward abstract interpretation of arithmetic operations *)
    let rec faexp' a r =
      match a with
      | (INT i)          -> (L.f_INT i)
      | (VAR v)          -> (E'.get r v)
      | RANDOM           -> (L.f_RANDOM ())
      | (UMINUS a1)      -> (L.f_UMINUS (faexp' a1 r))
      | (UPLUS a1)       -> (L.f_UPLUS (faexp' a1 r))
      | (PLUS (a1, a2))  -> (L.f_PLUS (faexp' a1 r) (faexp' a2 r))
      | (MINUS (a1, a2)) -> (L.f_MINUS (faexp' a1 r) (faexp' a2 r))
      | (TIMES (a1, a2)) -> (L.f_TIMES (faexp' a1 r) (faexp' a2 r))
      | (DIV (a1, a2))   -> (L.f_DIV (faexp' a1 r) (faexp' a2 r))
      | (MOD (a1, a2))   -> (L.f_MOD (faexp' a1 r) (faexp' a2 r))
    let faexp a r =
      if (E'.is_bot r) & (L.isbotempty ()) then (L.bot ()) else faexp' a r
  end;;

module Faexp = (Faexp_implementation:Faexp_signature);;
```


Example: Initialization and simple sign abstract forward arithmetic operations

Initialization and simple sign abstract forward arithmetic operations

$$
\begin{array}{ll}
 & \alpha(\{\underline{\mathtt{n}}\}) \\
= & \quad \text{(13) and case analysis} \\
 & \begin{array}{lll} \mathtt{NEG} & \text{if} & \underline{\mathtt{n}} \in [\mathtt{min\_int}, -1] \\ \mathtt{ZERO} & \text{if} & \underline{\mathtt{n}} = 0 \\ \mathtt{POS} & \text{if} & \underline{\mathtt{n}} \in [1, \mathtt{max\_int}] \\ \mathtt{BOT} & \text{if} & \underline{\mathtt{n}} < \mathtt{min\_int} \quad \text{or} \quad \underline{\mathtt{n}} > \mathtt{max\_int} \end{array} \\
\triangleq & \mathtt{n}^\flat \ .
\end{array}
$$

$$
\begin{array}{ll}
 & \alpha(\mathbb{I}) \\
= & \quad \text{(13)} \\
 & \mathtt{INI} \\
\triangleq & ?^\flat \ .
\end{array}
$$

We design $-^\flat(p) \triangleq \alpha(\{-v \mid v \in \gamma(p)\})$ by case analysis:

$$
\begin{array}{lll}
-^\flat(\mathtt{BOT}) & = \alpha(\{-v \mid v \in \gamma(\mathtt{BOT})\}) & \text{(def. (38) of } -^\flat) \\
 & = \alpha(\{\underline{-}v \mid v \in \{\Omega_\mathtt{a}\}\}) & \text{(def. (12) of } \gamma) \\
 & = \alpha(\{\Omega_\mathtt{a}\}) & \text{(def. (23) of } \underline{-}) \\
 & = \mathtt{BOT} & \text{(def. (13) of } \alpha) \\
-^\flat(\mathtt{POS}) & = \alpha(\{-v \mid v \in \gamma(\mathtt{POS})\}) & \text{(def. (38) of } -^\flat) \\
 & = \alpha(\{\underline{-}v \mid v \in [1, \mathtt{max\_int}] \cup \{\Omega_\mathtt{a}\}\}) & \text{(def. (12) of } \gamma) \\
 & = \alpha([-\mathtt{max\_int}, -1] \cup \{\Omega_\mathtt{a}\}) & \text{(def. (23) of } \underline{-} \text{ and (1))} \\
 & = \mathtt{NEG} & \text{(def. (13) of } \alpha) \\
-^\flat(\mathtt{ERR}) & = \alpha(\{-v \mid v \in \gamma(\mathtt{ERR})\}) & \text{(def. (38) of } -^\flat) \\
 & = \alpha(\{\underline{-}v \mid v \in \{\Omega_\mathtt{i}, \Omega_\mathtt{a}\}\}) & \text{(def. (12) of } \gamma) \\
 & = \alpha(\{\Omega_\mathtt{i}, \Omega_\mathtt{a}\}) & \text{(def. (23) of } \underline{-}) \\
 & = \mathtt{ERR} & \text{(def. (13) of } \alpha)
\end{array}
$$

...

• The calculational design for the other cases of opposite $-^\flat$ and that of identity $+^\flat$ is similar and we get

p	BOT	NEG	ZERO	POS	INI	ERR	TOP
$+^\flat(p)$	BOT	NEG	ZERO	POS	INI	ERR	TOP
$-^\flat(p)$	BOT	POS	ZERO	NEG	INI	ERR	TOP

• The calculational design of the abstract binary operators is also similar and will not be fully detailed. For division, we get

	$/^\flat(p,q)$	q = BOT	NEG	ZERO	POS	INI	ERR	TOP
	BOT	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	NEG	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	ZERO	BOT	BOT	BOT	ZERO	POS	ERR	TOP
p	POS	BOT	BOT	BOT	INI	INI	ERR	TOP
	INI	BOT	BOT	BOT	INI	INI	ERR	TOP
	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
	TOP	ERR	ERR	ERR	TOP	TOP	ERR	TOP

• addition

	$+^\flat(p,q)$	q = BOT	NEG	ZERO	POS	INI	ERR	TOP
	BOT	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	NEG	BOT	NEG	NEG	INI	INI	ERR	TOP
	ZERO	BOT	NEG	ZERO	POS	INI	ERR	TOP
p	POS	BOT	INI	POS	POS	INI	ERR	TOP
	INI	BOT	INI	INI	INI	INI	ERR	TOP
	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
	TOP	ERR	TOP	TOP	TOP	TOP	ERR	TOP

• substraction

	$-^\flat(p,q)$	q = BOT	NEG	ZERO	POS	INI	ERR	TOP
	BOT	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	NEG	BOT	INI	NEG	NEG	INI	ERR	TOP
	ZERO	BOT	POS	ZERO	NEG	INI	ERR	TOP
p	POS	BOT	POS	POS	INI	INI	ERR	TOP
	INI	BOT	INI	INI	INI	INI	ERR	TOP
	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
	TOP	ERR	TOP	TOP	TOP	TOP	ERR	TOP

- multiplication

$*^\sharp(p,q)$		q						
		BOT	NEG	ZERO	POS	INI	ERR	TOP
p	BOT	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	NEG	BOT	POS	ZERO	NEG	INI	ERR	TOP
	ZERO	BOT	ZERO	ZERO	ZERO	ZERO	ERR	TOP
	POS	BOT	NEG	ZERO	POS	INI	ERR	TOP
	INI	BOT	INI	ZERO	INI	INI	ERR	TOP
	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
	TOP	ERR	TOP	TOP	TOP	TOP	ERR	TOP

- completeness

$\mathtt{mod}^\sharp(p,q)$		q						
		BOT	NEG	ZERO	POS	INI	ERR	TOP
p	BOT	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	NEG	BOT	BOT	BOT	BOT	BOT	BOT	BOT
	ZERO	BOT	BOT	BOT	ZERO	ZERO	ERR	TOP
	POS	BOT	BOT	BOT	INI	INI	ERR	TOP
	INI	BOT	BOT	BOT	INI	INI	ERR	TOP
	ERR	ERR	ERR	ERR	ERR	ERR	ERR	ERR
	TOP	ERR	ERR	ERR	TOP	TOP	ERR	TOP

Generic backward/bottom-up abstract interpretation of arithmetic expressions

Backward/bottom-up abstract semantics of arithmetic expressions

$$\mathrm{Baexp}^\sharp \in \mathrm{Aexp} \mapsto (\mathbb{V} \mapsto L) \stackrel{\mathrm{mon}}{\longmapsto} L \stackrel{\mathrm{mon}}{\longmapsto} (\mathbb{V} \mapsto L)\ .$$

- Up to abstraction, $\mathrm{Baexp}^\sharp[\![A]\!](r)p$ provides a necessary precondition on the values of the variables for the evaluation of expression A in an environment satisfying r to yield of value satisfying the predicate p.

For any possible approximation (8) of value properties, we approximate environment properties by the nonrelational abstraction (22) and apply the following functional abstraction

$$\langle \wp(\mathbb{R}) \overset{\text{mon}}{\longmapsto} \wp(\mathbb{I}_\Omega) \overset{\text{mon}}{\longmapsto} \wp(\mathbb{R}),\ \ddot{\subseteq} \rangle \underset{\alpha^\sharp}{\overset{\gamma^\sharp}{\leftrightarrows}} \langle (\mathbb{V} \mapsto L) \overset{\text{mon}}{\longmapsto} L \overset{\text{mon}}{\longmapsto} (\mathbb{V} \mapsto L),\ \ddot{\sqsubseteq} \rangle$$

where

$$\begin{aligned}
\Phi \mathrel{\ddot{\subseteq}} \Psi &\triangleq \forall R \in \wp(\mathbb{R}) : \forall P \in \wp(\mathbb{I}_\Omega) : \Phi(R)P \subseteq \Psi(R)P, \\
\varphi \mathrel{\ddot{\sqsubseteq}} \psi &\triangleq \forall r \in \mathbb{V} \mapsto L : \forall p \in L : \varphi(r)p \mathrel{\dot{\sqsubseteq}} \psi(r)p, \\
\alpha^\sharp(\Phi) &\triangleq \lambda r \in \mathbb{V} \mapsto L \cdot \lambda p \in L \cdot \dot{\alpha}(\Phi(\dot{\gamma}(r))\gamma(p)), \\
\gamma^\sharp(\varphi) &\triangleq \lambda R \in \wp(\mathbb{R}) \cdot \lambda P \in \wp(\mathbb{I}_\Omega) \cdot \dot{\gamma}(\varphi(\dot{\alpha}(R))\alpha(P)) \ .
\end{aligned} \tag{41}$$

Objective

Get an overapproximation of the backward collecting semantics (31) such that

$$\text{Baexp}^\sharp[\![A]\!] \mathrel{\ddot{\supseteq}} \alpha^\sharp(\text{Baexp}[\![A]\!]) \ . \tag{42}$$

We derive $\text{Baexp}^\sharp[\![A]\!]$ by calculus, as follows

$$\begin{array}{ll}
& \alpha^\sharp(\text{Baexp}[\![A]\!]) \\
= & \quad \wr \text{def. (41) of } \alpha^\sharp \int \\
& \lambda r \in \mathbb{V} \mapsto L \cdot \lambda p \in L \cdot \dot{\alpha}(\text{Baexp}[\![A]\!](\dot{\gamma}(r))\gamma(p)) \\
= & \quad \wr \text{def. (31) of Baexp}[\![A]\!] \int \\
& \lambda r \in \mathbb{V} \mapsto L \cdot \lambda p \in L \cdot \dot{\alpha}(\{\rho \in \dot{\gamma}(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash A \Rightarrow i\}) \ .
\end{array}$$

- If r is the infimum $\lambda \mathtt{Y} \cdot \bot$ where the infimum $\bot$ of L is such that $\gamma(\bot) = \emptyset$, then $\dot{\gamma}(r) = \emptyset$ whence

$$\begin{array}{ll}
& \alpha^\sharp(\text{Baexp}[\![A]\!])(\lambda \mathtt{Y} \cdot \bot)p \\
= & \quad \wr \text{def. (21) of } \dot{\gamma} \int \\
& \dot{\alpha}(\emptyset) \\
= & \quad \wr \text{def. (20) of } \dot{\alpha} \int \\
& \lambda \mathtt{Y} \cdot \bot \ .
\end{array}$$

- If $r \in \mathbb{V} \mapsto L$, $r \neq \lambda \mathtt{Y} \cdot \bot$ or $\gamma(\bot) \neq \emptyset$ and $p \in L$, we proceed by structural induction on the arithmetic expression A.

- When $A = \mathtt{n} \in \text{Nat}$ is a number, we have

$$\begin{array}{ll}
& \alpha^\sharp(\text{Baexp}[\![\mathtt{n}]\!])(r)p \\
= & \dot{\alpha}(\{\rho \in \dot{\gamma}(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash \mathtt{n} \Rightarrow i\}) \\
= & \quad \wr \text{def. (25) of } \rho \vdash \mathtt{n} \Rightarrow i \int \\
& \dot{\alpha}(\{\rho \in \dot{\gamma}(r) \mid \underline{\mathtt{n}} \in \gamma(p) \cap \mathbb{I}\}) \\
= & \quad \wr \text{def. conditional } (\ldots \ ? \ \ldots \ ¿ \ \ldots) \int \\
& (\underline{\mathtt{n}} \in \gamma(p) \cap \mathbb{I} \ ? \ \dot{\alpha}(\dot{\gamma}(r)) \ ¿ \ \dot{\alpha}(\emptyset)) \\
\mathrel{\dot{\sqsubseteq}} & \quad \wr \dot{\alpha} \circ \dot{\gamma} \text{ is reductive (6) and def. (20) of } \dot{\alpha} \int \\
& (\underline{\mathtt{n}} \in \gamma(p) \cap \mathbb{I} \ ? \ r \ ¿ \ \lambda \mathtt{Y} \cdot \bot) \\
= & \quad \wr \text{by defining } \mathtt{n}^\sharp(p) \triangleq (\underline{\mathtt{n}} \in \gamma(p) \cap \mathbb{I}) \int \\
& (\mathtt{n}^\sharp(p) \ ? \ r \ ¿ \ \lambda \mathtt{Y} \cdot \bot) \\
= & \quad \wr \text{by defining } \text{Baexp}^\sharp[\![\mathtt{n}]\!](r)p \triangleq (\mathtt{n}^\sharp(p) \ ? \ r \ ¿ \ \lambda \mathtt{Y} \cdot \bot) \int \\
& \text{Baexp}^\sharp[\![\mathtt{n}]\!](r)p \ .
\end{array}$$

- When $A = \mathtt{X} \in \mathbb{V}$ is a variable, we have

$\alpha^\sharp(\text{Baexp}[\![\mathtt{X}]\!])(r)p$
$= \dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash \mathtt{X} \mapsto i\})$
$= \quad$ ⟅def. (26) of $\rho \vdash \mathtt{X} \mapsto i$⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \rho(\mathtt{X}) \in \gamma(p) \cap \mathbb{I}\})$
$\dot\subseteq \quad$ ⟅$\gamma \circ \alpha$ is extensive (5) and $\dot\alpha$ is monotone (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \rho(\mathtt{X}) \in \gamma(p) \cap \gamma \circ \alpha(\mathbb{I})\})$
$= \quad$ ⟅def. (21) of $\dot\gamma$⟆
$\dot\alpha(\{\rho \mid \forall \mathtt{Y} \neq \mathtt{X} : \rho(\mathtt{Y}) \in \gamma(r(\mathtt{Y})) \wedge \rho(\mathtt{X}) \in \gamma(r(\mathtt{X})) \cap \gamma(p) \cap \gamma \circ \alpha(\mathbb{I})\})$
$= \quad$ ⟅γ is a complete meet morphism⟆
$\dot\alpha(\{\rho \mid \forall \mathtt{Y} \neq \mathtt{X} : \rho(\mathtt{Y}) \in \gamma(r(\mathtt{Y})) \wedge \rho(\mathtt{X}) \in \gamma(r(\mathtt{X}) \sqcap p \sqcap \alpha(\mathbb{I}))\})$
$= \quad$ ⟅def. (18) of environment assignment⟆
$\dot\alpha(\{\rho \mid \forall \mathtt{Y} \neq \mathtt{X} : \rho(\mathtt{Y}) \in \gamma(r[\mathtt{X} \leftarrow r(\mathtt{X}) \sqcap p \sqcap \alpha(\mathbb{I})](\mathtt{Y})) \wedge \rho(\mathtt{X}) \in \gamma(r[\mathtt{X} \leftarrow r(\mathtt{X}) \sqcap p \sqcap \alpha(\mathbb{I})](\mathtt{X}))\})$

...

- When $A = ?$ is random, we have

$\alpha^\sharp(\text{Baexp}[\![?]\!])(r)p$
$= \dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash ? \mapsto i\})$
$= \quad$ ⟅def. (27) of $\rho \vdash ? \mapsto i$⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \gamma(p) \cap \mathbb{I} \neq \emptyset\})$
$= \quad$ ⟅def. conditional $(\ldots ? \ldots ¿ \ldots)$⟆
$(\gamma(p) \cap \mathbb{I} = \emptyset \; ? \; \dot\alpha(\emptyset) \; ¿ \; \dot\alpha(\dot\gamma(r)))$
$\dot\subseteq \quad$ ⟅def. (20) of $\dot\alpha$ and $\dot\alpha \circ \dot\gamma$ reductive (6)⟆
$(\gamma(p) \cap \mathbb{I} = \emptyset \; ? \; \lambda \mathtt{Y} \cdot \bot \; ¿ \; r)$
$= \quad$ ⟅negation⟆
$(\gamma(p) \cap \mathbb{I} \neq \emptyset \; ? \; r \; ¿ \; \lambda \mathtt{Y} \cdot \bot)$
$= \quad$ ⟅by defining $?^\sharp(p) \stackrel{\Delta}{=} (\gamma(p) \cap \mathbb{I} \neq \emptyset)$⟆
$(?^\sharp(p) \; ? \; r \; ¿ \; \lambda \mathtt{Y} \cdot \bot)$
$= \quad$ ⟅by defining $\text{Baexp}^\sharp[\![?]\!] \stackrel{\Delta}{=} (?^\sharp(p) \; ? \; r \; ¿ \; \lambda \mathtt{Y} \cdot \bot)$⟆
$\text{Baexp}^\sharp[\![?]\!](r)p$.

- When $A = \mathtt{u}\, A'$ is a unary operation, we have

$\alpha^\sharp(\text{Baexp}[\![\mathtt{u}\, A']\!])(r)p$
$= \dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash \mathtt{u}\, A' \mapsto i\})$
$= \quad$ ⟅def. (28) of $\rho \vdash \mathtt{u}\, A' \mapsto i$⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' : \rho \vdash A' \mapsto i' \wedge \underline{\mathtt{u}}\, i' \in \gamma(p) \cap \mathbb{I}\})$
$= \quad$ ⟅set theory⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A' \mapsto v\} : \rho \vdash A' \mapsto i' \wedge \underline{\mathtt{u}}\, i' \in \gamma(p) \cap \mathbb{I}\})$
$\dot\subseteq \quad$ ⟅$\gamma \circ \alpha$ extensive (5) and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(\alpha(\{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A' \mapsto v\})) : \rho \vdash A' \mapsto i' \wedge \underline{\mathtt{u}}\, i' \in \gamma(p) \cap$
$\dot\subseteq \quad$ ⟅(35) implying $\text{Faexp}^\sharp[\![A']\!]r \sqsupseteq \alpha(\{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A' \mapsto v\})$,
γ and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(\text{Faexp}^\sharp[\![A']\!]r) : \rho \vdash A' \mapsto i' \wedge \underline{\mathtt{u}}\, i' \in \gamma(p) \cap \mathbb{I}\})$
$= \quad$ ⟅def. (23) of $\underline{\mathtt{u}}$ (such that $\underline{\mathtt{u}}\, i' \in \mathbb{I}$ only if $i' \in \mathbb{I}$)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(\text{Faexp}^\sharp[\![A']\!]r) \cap \mathbb{I} : \rho \vdash A' \mapsto i' \wedge \underline{\mathtt{u}}\, i' \in \gamma(p) \cap \mathbb{I}\})$

...

$=$ ⟅set theory⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \{i \in \gamma(\text{Faexp}^\sharp[\![A']\!]r) \mid \underline{\mathsf{u}}\, i \in \gamma(p) \cap \mathbb{I}\} \cap \mathbb{I} : \rho \vdash A' \Mapsto i'\})$
$\dot\subseteq$ ⟅$\gamma \circ \alpha$ extensive (5) and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(\alpha(\{i \in \gamma(\text{Faexp}^\sharp[\![A']\!]r) \mid \underline{\mathsf{u}}\, i \in \gamma(p) \cap \mathbb{I}\})) \cap \mathbb{I} : \rho \vdash A' \Mapsto i'\})$
$\dot\subseteq$ ⟅defining $\mathsf{u}^\sharp$ such that $\mathsf{u}^\sharp(q,p) \sqsupseteq \alpha(\{i \in \gamma(q) \mid \underline{\mathsf{u}}\, i \in \gamma(p) \cap \mathbb{I}\})$, γ and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(\mathsf{u}^\sharp(\text{Faexp}^\sharp[\![A']\!]r, p)) \cap \mathbb{I} : \rho \vdash A' \Mapsto i'\})$
$\dot\subseteq$ ⟅induction hypothesis (42) implying $\text{Baexp}^\sharp[\![A']\!](r)p \mathrel{\dot\sqsupseteq} \dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(p) \cap \mathbb{I} : \rho \vdash A' \Mapsto i'\})$⟆
$\text{Baexp}^\sharp[\![A']\!](r)(\mathsf{u}^\sharp(\text{Faexp}^\sharp[\![A']\!]r, p))$
$=$ ⟅defining $\text{Baexp}^\sharp[\![\mathsf{u}\, A']\!](r)p \stackrel{\Delta}{=} \text{Baexp}^\sharp[\![A']\!](r)(\mathsf{u}^\sharp(\text{Faexp}^\sharp[\![A']\!]r, p))$⟆
$\text{Baexp}^\sharp[\![\mathsf{u}\, A']\!](r)p$.

- When $A = A_1 \,\mathsf{b}\, A_2$ is a binary operation, we have

$\alpha^\sharp(\text{Baexp}[\![A_1 \,\mathsf{b}\, A_2]\!])(r)p$
$=$ $\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i \in \gamma(p) \cap \mathbb{I} : \rho \vdash A_1 \,\mathsf{b}\, A_2 \Mapsto i\})$
$=$ ⟅def. (29) of $\rho \vdash A_1 \,\mathsf{b}\, A_2 \Mapsto i$⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i_1, i_2 : \rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2 \wedge i_1 \,\underline{\mathsf{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\})$
$=$ ⟅set theory⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i_1 \in \{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A_1 \Mapsto v\} :$
$\exists i_2 \in \{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A_2 \Mapsto v\} :$
$\rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2 \wedge i_1 \,\underline{\mathsf{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\})$
$\dot\subseteq$ ⟅$\gamma \circ \alpha$ extensive (5) and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i_1 \in \gamma(\alpha(\{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A_1 \Mapsto v\})) :$
$\exists i_2 \in \gamma(\alpha(\{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash A_2 \Mapsto v\})) :$
$\rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2 \wedge i_1 \,\underline{\mathsf{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\})$
$\dot\subseteq$ ⟅(35) implying $\text{Faexp}^\sharp[\![A_i]\!]r \sqsupseteq \alpha(\{v \mid \exists \rho' \in \dot\gamma(r) : \rho' \vdash_i v\})$, $i = 1, 2$, γ and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i_1 \in \gamma(\text{Faexp}^\sharp[\![A_1]\!]r) : \exists i_2 \in \gamma(\text{Faexp}^\sharp[\![A_2]\!]r) :$
$\rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2 \wedge i_1 \,\underline{\mathsf{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\})$
. . .

$=$ ⟅def. (24) of $\underline{\mathsf{b}}$ (such that $i_1 \,\underline{\mathsf{b}}\, i_2 \in \mathbb{I}$ only if $i_1, i_2 \in \mathbb{I}$)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i_1 \in \gamma(\text{Faexp}^\sharp[\![A_1]\!]r) \cap \mathbb{I} : \exists i_2 \in \gamma(\text{Faexp}^\sharp[\![A_2]\!]r) \cap \mathbb{I} :$
$\rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2 \wedge i_1 \,\underline{\mathsf{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\})$
$=$ ⟅set theory⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists \langle i_1, i_2\rangle \in \{\langle i_1', i_2'\rangle \in \gamma(\text{Faexp}^\sharp[\![A_1]\!]r) \times \gamma(\text{Faexp}^\sharp[\![A_2]\!]r) \mid$
$i_1' \,\underline{\mathsf{b}}\, i_2' \in \gamma(p) \cap \mathbb{I}\} \cap (\mathbb{I} \times \mathbb{I}) : \rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2\})$
$\dot\subseteq$ ⟅$\gamma^2 \circ \alpha^2$ extensive (11), (5) and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists \langle i_1, i_2\rangle \in \gamma^2(\alpha^2(\{\langle i_1', i_2'\rangle \in \gamma(\text{Faexp}^\sharp[\![A_1]\!]r) \times \gamma(\text{Faexp}^\sharp[\![A_2]\!]r) \mid$
$i_1' \,\underline{\mathsf{b}}\, i_2' \in \gamma(p) \cap \mathbb{I}\})) \cap (\mathbb{I} \times \mathbb{I}) : \rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2\})$
$\dot\subseteq$ ⟅defining $\mathsf{b}^\sharp$ such that $\mathsf{b}^\sharp(q_1, q_2, p) \sqsupseteq^2 \alpha^2(\{\langle i_1', i_2'\rangle \in \gamma^2(\langle q_1, q_2\rangle) \mid i_1' \,\underline{\mathsf{b}}\, i_2' \in \gamma(p) \cap \mathbb{I}\})$, γ^2 and $\dot\alpha$ monotone (22), (4)⟆
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists \langle i_1, i_2\rangle \in \gamma^2(\mathsf{b}^\sharp(\text{Faexp}^\sharp[\![A_1]\!]r, \text{Faexp}^\sharp[\![A_2]\!]r, p)) \cap (\mathbb{I} \times \mathbb{I}) :$
$\rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2\})$
$=$ ⟅let notation⟆
. . .

let $\langle p_1, p_2\rangle = \mathsf{b}^\sharp(\text{Faexp}^\sharp[\![A_1]\!]r, \text{Faexp}^\sharp[\![A_2]\!]r, p)$ in
$\dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists \langle i_1, i_2\rangle \in \gamma^2(\langle p_1, p_2\rangle) \cap (\mathbb{I} \times \mathbb{I}) : \rho \vdash A_1 \Mapsto i_1 \wedge \rho \vdash A_2 \Mapsto i_2\})$
$=$ ⟅def. (10) of γ^2 and $\dot\alpha$ monotone (22), (4)⟆
let $\langle p_1, p_2\rangle = \mathsf{b}^\sharp(\text{Faexp}^\sharp[\![A_1]\!]r, \text{Faexp}^\sharp[\![A_2]\!]r, p)$ in
$\dot\alpha(\{\rho_1 \in \dot\gamma(r) \mid \exists i_1 \in \gamma p_1 \cap \mathbb{I} : \rho_1 \vdash A_1 \Mapsto i_1\} \cap$
$\{\rho_2 \in \dot\gamma(r) \mid \exists i_2 \in \gamma p_2 \cap \mathbb{I} : \rho_2 \vdash A_2 \Mapsto i_2\})$
$=$ ⟅$\dot\alpha$ complete join morphism⟆
let $\langle p_1, p_2\rangle = \mathsf{b}^\sharp(\text{Faexp}^\sharp[\![A_1]\!]r, \text{Faexp}^\sharp[\![A_2]\!]r, p)$ in
$\dot\alpha(\{\rho_1 \in \dot\gamma(r) \mid \exists i_1 \in \gamma p_1 \cap \mathbb{I} : \rho_1 \vdash A_1 \Mapsto i_1\})$
$\dot\sqcap\, \dot\alpha(\{\rho_2 \in \dot\gamma(r) \mid \exists i_2 \in \gamma p_2 \cap \mathbb{I} : \rho_2 \vdash A_2 \Mapsto i_2\})$
$\dot\subseteq$ ⟅induction hypothesis (42) implying $\text{Baexp}^\sharp[\![A']\!](r)p \mathrel{\dot\sqsupseteq} \dot\alpha(\{\rho \in \dot\gamma(r) \mid \exists i' \in \gamma(p) \cap \mathbb{I} : \rho \vdash A' \Mapsto i'\})$⟆
let $\langle p_1, p_2\rangle = \mathsf{b}^\sharp(\text{Faexp}^\sharp[\![A_1]\!]r, \text{Faexp}^\sharp[\![A_2]\!]r, p)$ in
$\text{Baexp}^\sharp[\![A_1]\!](r)p_1 \dot\sqcap \text{Baexp}^\sharp[\![A_2]\!](r)p_2$
. . .

$$
\begin{array}{rl}
= & \wr \text{defining } \mathrm{Baexp}^\sharp[\![A_1 \,\mathtt{b}\, A_2]\!](r)p \stackrel{\triangle}{=} \\
& \quad \text{let } \langle p_1,\ p_2\rangle = \mathtt{b}^\sharp(\mathrm{Faexp}^\sharp[\![A_1]\!]r, \mathrm{Faexp}^\sharp[\![A_2]\!]r, p) \text{ in} \\
& \quad\quad \mathrm{Baexp}^\sharp[\![A_1]\!](r)p_1 \mathbin{\dot\sqcap} \mathrm{Baexp}^\sharp[\![A_2]\!](r)p_2 \wr \\
& \mathrm{Baexp}^\sharp[\![A_1 \,\mathtt{b}\, A_2]\!](r)p\ .
\end{array}
$$

Backward abstract interpretation of arithmetic expressions

$$
\begin{array}{rcll}
\mathrm{Baexp}^\sharp[\![A]\!](\lambda \mathtt{Y}\cdot\bot)p & \stackrel{\triangle}{=} & \lambda \mathtt{Y}\cdot\bot \qquad \text{if } \gamma(\bot) = \emptyset & (43) \\
\mathrm{Baexp}^\sharp[\![\mathtt{n}]\!](r)p & \stackrel{\triangle}{=} & (\mathtt{n}^\sharp(p)\ ?\ r\ ¿\ \lambda \mathtt{Y}\cdot\bot) & \\
\mathrm{Baexp}^\sharp[\![\mathtt{X}]\!](r)p & \stackrel{\triangle}{=} & r[\mathtt{X} \leftarrow r(\mathtt{X}) \sqcap p \sqcap ?^\sharp] & (44) \\
\mathrm{Baexp}^\sharp[\![?]\!](r)p & \stackrel{\triangle}{=} & (?^\sharp(p)\ ?\ r\ ¿\ \lambda \mathtt{Y}\cdot\bot) & \\
\mathrm{Baexp}^\sharp[\![\mathtt{u}\, A']\!](r)p & \stackrel{\triangle}{=} & \mathrm{Baexp}^\sharp[\![A']\!](r)(\mathtt{u}^\sharp(\mathrm{Faexp}^\sharp[\![A']\!]r, p)) & \\
\mathrm{Baexp}^\sharp[\![A_1 \,\mathtt{b}\, A_2]\!](r)p & \stackrel{\triangle}{=} & \text{let } \langle p_1,\ p_2\rangle = \mathtt{b}^\sharp(\mathrm{Faexp}^\sharp[\![A_1]\!]r, \mathrm{Faexp}^\sharp[\![A_2]\!]r, p) \text{ in} & \\
 & & \quad \mathrm{Baexp}^\sharp[\![A_1]\!](r)p_1 \mathbin{\dot\sqcap} \mathrm{Baexp}^\sharp[\![A_2]\!](r)p_2 &
\end{array}
$$

parameterized by the following backward abstract operations on L

Backward abstract operations

$$
\begin{array}{rcll}
\mathtt{n}^\sharp(p) & \stackrel{\triangle}{=} & (\underline{\mathtt{n}} \in \gamma(p) \cap \mathbb{I}) & (45) \\
?^\sharp(p) & \stackrel{\triangle}{=} & (\gamma(p) \cap \mathbb{I} \neq \emptyset) & (46) \\
\mathtt{u}^\sharp(q, p) & \sqsupseteq & \alpha(\{i \in \gamma(q) \mid \underline{\mathtt{u}}\, i \in \gamma(p) \cap \mathbb{I}\}) & (47) \\
\mathtt{b}^\sharp(q_1, q_2, p) & \sqsupseteq^2 & \alpha^2(\{\langle i_1,\ i_2\rangle \in \gamma^2(\langle q_1,\ q_2\rangle) \mid i_1\, \underline{\mathtt{b}}\, i_2 \in \gamma(p) \cap \mathbb{I}\}) & (48)
\end{array}
$$

Monotony and reductivity

- For all $p \in L$ and by induction on A, the operator $\lambda r\cdot \mathrm{Baexp}^\sharp[\![A]\!](r)p$ on $\mathbb{V} \mapsto L$ is $\dot\sqsubseteq$-reductive and monotonic.

Generic backward/bottom-up static analyzer of arithmetic expressions

```
module type Baexp_signature =
 functor (L: Abstract_Lattice_Algebra_signature) ->
 functor (E: Abstract_Env_Algebra_signature) ->
 functor (Faexp: Faexp_signature) ->
 sig
  open Abstract_Syntax
  (* generic backward abstract interpretation of arithmetic operations *)
  val baexp : aexp ->  E (L).env -> L.lat -> E (L).env
 end;;
```



```
module Baexp_implementation =
 functor (L: Abstract_Lattice_Algebra_signature) ->
 functor (E: Abstract_Env_Algebra_signature) ->
 functor (Faexp: Faexp_signature) ->
struct
 open Abstract_Syntax
 (* generic abstract environments                                    *)
 module E' = E (L)
 (* generic forward abstract interpretation of arithmetic operations  *)
 module Faexp' = Faexp(L)(E)
 (* generic backward abstract interpretation of arithmetic operations *)
 let rec baexp' a r p =
   match a with
   | (INT i) -> if (L.b_INT i p) then r else (E'.bot ())
   | (VAR v) ->
       (E'.set r v (L.meet (L.meet (E'.get r v) p) (L.f_RANDOM ())))
   | RANDOM -> if (L.b_RANDOM p) then r else (E'.bot ())
   | (UMINUS a1) -> (baexp' a1 r (L.b_UMINUS (Faexp'.faexp a1 r) p))
   | (UPLUS a1)  -> (baexp' a1 r (L.b_UPLUS (Faexp'.faexp a1 r) p))

 ...
```



```
   | (PLUS (a1, a2)) ->
     let (p1,p2) = (L.b_PLUS (Faexp'.faexp a1 r) (Faexp'.faexp a2 r) p)
       in (E'.meet (baexp' a1 r p1) (baexp' a2 r p2))
   | (MINUS (a1, a2))  ->
     let (p1,p2) = (L.b_MINUS (Faexp'.faexp a1 r) (Faexp'.faexp a2 r) p)
       in (E'.meet (baexp' a1 r p1) (baexp' a2 r p2))
   | (TIMES (a1, a2))  ->
     let (p1,p2) = (L.b_TIMES (Faexp'.faexp a1 r) (Faexp'.faexp a2 r) p)
       in (E'.meet (baexp' a1 r p1) (baexp' a2 r p2))
   | (DIV (a1, a2))  ->
     let (p1,p2) = (L.b_DIV (Faexp'.faexp a1 r) (Faexp'.faexp a2 r) p)
       in (E'.meet (baexp' a1 r p1) (baexp' a2 r p2))
   | (MOD (a1, a2))  ->
     let (p1,p2) = (L.b_MOD (Faexp'.faexp a1 r) (Faexp'.faexp a2 r) p)
       in (E'.meet (baexp' a1 r p1) (baexp' a2 r p2))
 let baexp a r p =
  if (E'.is_bot r) & (L.isbotempty ()) then (E'.bot ()) else baexp' a r p
end;;

module Baexp = (Baexp_implementation:Baexp_signature);;
```


Abstract operations:

```
module type Abstract_Lattice_Algebra_signature =
  sig
    (* complete lattice of abstract properties of values          *)
    type lat                                (* abstract properties  *)
    ...
    (* forward abstract interpretation of arithmetic expressions  *)
    ...
    (* backward abstract interpretation of arithmetic expressions *)
    val b_INT    : string -> lat -> bool
    val b_RANDOM : lat -> bool
    val b_UMINUS : lat -> lat -> lat
    val b_UPLUS  : lat -> lat -> lat
    val b_PLUS   : lat -> lat -> lat -> lat * lat
    val b_MINUS  : lat -> lat -> lat -> lat * lat
    val b_TIMES  : lat -> lat -> lat -> lat * lat
    val b_DIV    : lat -> lat -> lat -> lat * lat
    val b_MOD    : lat -> lat -> lat -> lat * lat
    ...
  end;;
```

Example: Initialization and simple sign abstract backward arithmetic operations

Initialization and simple sign abstract backward arithmetic operations

In the abstract interpretation (44) of variables, we have

$$?^\sharp = \texttt{INI}$$

by definition (13) of α. From the definition (45) of $\mathtt{n}^\sharp$ and (12) of γ, we directly get by case analysis

$\mathtt{n}^\sharp(p)$	p						
	BOT	NEG	ZERO	POS	INI	ERR	TOP
$\underline{\mathtt{n}} \in [\mathtt{min\_int}, -1]$	ff	tt	ff	ff	tt	ff	tt
$\underline{\mathtt{n}} = 0$	ff	ff	tt	ff	tt	ff	tt
$\underline{\mathtt{n}} \in [1, \mathtt{max\_int}]$	ff	ff	ff	tt	tt	ff	tt
$\underline{\mathtt{n}} < \mathtt{min\_int} \vee \underline{\mathtt{n}} > \mathtt{max\_int}$	ff	ff	ff	ff	ff	ff	ff

- From the definition (46) of $?^\sharp$ and (12) of γ, we directly get by case analysis

p	BOT	NEG	ZERO	POS	INI	ERR	TOP
$?^\sharp(p)$	ff	tt	tt	tt	tt	ff	tt

- For the backward unary arithmetic operations (47), we have

p	BOT	NEG	ZERO	POS	INI	ERR	TOP
$+^\sharp(q,p)$	BOT	$q \sqcap$ NEG	$q \sqcap$ ZERO	$q \sqcap$ POS	$q \sqcap$ INI	BOT	$q \sqcap$ INI
$-^\sharp(q,p)$	BOT	$q \sqcap$ POS	$q \sqcap$ ZERO	$q \sqcap$ NEG	$q \sqcap$ INI	BOT	$q \sqcap$ INI

- For the backward binary arithmetic operations (48), we have

$$/^\sharp(q_1, q_2, p) \triangleq \mathtt{mod}^\sharp(q_1, q_2, p) \triangleq$$
$$(q_1 \in \{\mathtt{BOT}, \mathtt{NEG}, \mathtt{ERR}\} \vee q_2 \in \{\mathtt{BOT}, \mathtt{NEG}, \mathtt{ZERO}, \mathtt{ERR}\} \vee p \in \{\mathtt{BOT}, \mathtt{NEG}, \mathtt{ERR}\} \ ?$$
$$\langle \mathtt{BOT}, \mathtt{BOT} \rangle$$
$$¿ \ (p = \mathtt{POS} \ ?$$
$$smash(\langle q_1 \sqcap \mathtt{POS}, \ q_2 \sqcap \mathtt{POS} \rangle) \quad ¿ \ \langle q_1 \sqcap \mathtt{INI}, \ q_2 \sqcap \mathtt{POS} \rangle \quad))$$

$$smash(\langle x, \ y \rangle) \triangleq$$
$$(x = \mathtt{BOT} \vee y = \mathtt{BOT} \ ?$$
$$\langle \mathtt{BOT}, \mathtt{BOT} \rangle$$
$$¿$$
$$\langle x, \ y \rangle) \ .$$

- With the same reasoning, for addition $+^\sharp$, we have

$$+^\sharp(q_1, q_2, p) = \langle \mathtt{BOT}, \ \mathtt{BOT} \rangle \quad \text{if } q_1 \in \{\mathtt{BOT}, \mathtt{ERR}\} \vee q_2 \in \{\mathtt{BOT}, \mathtt{ERR}\} \vee p \in \{\mathtt{BOT}, \mathtt{ERR}\}$$
$$+^\sharp(q_1, q_2, p) = \langle q_1 \sqcap \mathtt{INI}, \ q_2 \sqcap \mathtt{INI} \rangle \ \text{if } p \in \{\mathtt{INI}, \mathtt{TOP}\} \ .$$

Otherwise

$+^\sharp(q_1, q_2, \mathtt{NEG})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨NEG, NEG⟩	⟨NEG, ZERO⟩	⟨NEG, POS⟩	⟨NEG, INI⟩
	ZERO	⟨ZERO, NEG⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨ZERO, NEG⟩
	POS	⟨POS, NEG⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨POS, NEG⟩
	INI, TOP	⟨INI, NEG⟩	⟨NEG, ZERO⟩	⟨NEG, POS⟩	⟨INI, INI⟩

$+^\sharp(q_1, q_2, \mathtt{ZERO})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨NEG, POS⟩	⟨NEG, POS⟩
	ZERO	⟨BOT, BOT⟩	⟨ZERO, ZERO⟩	⟨BOT, BOT⟩	⟨ZERO, ZERO⟩
	POS	⟨POS, NEG⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨POS, NEG⟩
	INI, TOP	⟨POS, NEG⟩	⟨ZERO, ZERO⟩	⟨NEG, POS⟩	⟨INI, INI⟩

$+^\sharp(q_1, q_2, \mathtt{POS})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨NEG, POS⟩	⟨NEG, POS⟩
	ZERO	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨ZERO, POS⟩	⟨ZERO, POS⟩
	POS	⟨POS, NEG⟩	⟨POS, ZERO⟩	⟨POS, POS⟩	⟨POS, INI⟩
	INI, TOP	⟨POS, NEG⟩	⟨POS, ZERO⟩	⟨INI, POS⟩	⟨INI, INI⟩

- The backward ternary substraction operation $-^\sharp$ is defined as

$$-^\sharp(q_1, q_2, p) \triangleq \text{let } (r_1, r_2) = +^\sharp(q_1, -^\sharp(q_2), p) \text{ in } (r_1, -^\sharp(r_2)) \ .$$

- The handling of the backward ternary multiplication operation $*^{\circ}$ is similar

$*^{\circ}(q_1, q_2, \texttt{NEG})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨NEG, POS⟩	⟨NEG, POS⟩
	ZERO	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩
	POS	⟨POS, NEG⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨POS, NEG⟩
	INI, TOP	⟨POS, NEG⟩	⟨BOT, BOT⟩	⟨NEG, POS⟩	⟨INI, INI⟩

$*^{\circ}(q_1, q_2, \texttt{ZERO})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨BOT, BOT⟩	⟨NEG, ZERO⟩	⟨BOT, BOT⟩	⟨NEG, ZERO⟩
	ZERO	⟨ZERO, NEG⟩	⟨ZERO, ZERO⟩	⟨ZERO, POS⟩	⟨ZERO, INI⟩
	POS	⟨BOT, BOT⟩	⟨POS, ZERO⟩	⟨BOT, BOT⟩	⟨POS, ZERO⟩
	INI, TOP	⟨ZERO, NEG⟩	⟨INI, ZERO⟩	⟨ZERO, POS⟩	⟨INI, INI⟩

$*^{\circ}(q_1, q_2, \texttt{POS})$		q_2			
		NEG	ZERO	POS	INI, TOP
q_1	NEG	⟨NEG, NEG⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨NEG, NEG⟩
	ZERO	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩
	POS	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨POS, POS⟩	⟨POS, POS⟩
	INI, TOP	⟨NEG, NEG⟩	⟨BOT, BOT⟩	⟨POS, POS⟩	⟨INI, INI⟩

Semantics of Boolean Expressions

Abstract syntax of boolean expressions

Arithmetic expressions

$A_1, A_2 \in \text{Aexp}$.

Boolean expressions

$$
\begin{array}{lll}
B, B_1, B_2 \in \text{Bexp} ::= & \texttt{true} & \text{truth,} \\
| & \texttt{false} & \text{falsity,} \\
| & A_1 = A_2 \mid A_1 < A_2 & \text{arithmetic comparison,} \\
| & B_1 \,\&\, B_2 & \text{conjunction,} \\
| & B_1 \mid B_2 & \text{disjunction.}
\end{array}
$$

Normalization of boolean expressions

$$
\begin{array}{rcl}
T(\texttt{true}) & \triangleq & \texttt{true}, \\
T(\texttt{false}) & \triangleq & \texttt{false}, \\
T(A_1 < A_2) & \triangleq & A_1 < A_2, \\
T(A_1 <= A_2) & \triangleq & (A_1 < A_2) \,|\, (A_1 = A_2), \\
T(A_1 = A_2) & \triangleq & A_1 = A_2, \\
T(A_1 <> A_2) & \triangleq & (A_1 < A_2) \,|\, (A_2 < A_1), \\
T(A_1 > A_2) & \triangleq & A_2 < A_1, \\
T(A_1 >= A_2) & \triangleq & (A_1 = A_2) \,|\, (A_2 < A_1), \\
T(B_1 \,|\, B_2) & \triangleq & T(B_1) \,|\, T(B_2) \\
T(B_1 \,\&\, B_2) & \triangleq & T(B_1) \,\&\, T(B_2),
\end{array}
\qquad
\begin{array}{rcl}
T(\neg\texttt{true}) & \triangleq & \texttt{false}, \\
T(\neg\texttt{false}) & \triangleq & \texttt{true}, \\
T(\neg(A_1 < A_2)) & \triangleq & T(A_1 >= A_2), \\
T(\neg(A_1 <= A_2)) & \triangleq & T(A_1 > A_2), \\
T(\neg(A_1 = A_2)) & \triangleq & T(A_1 <> A_2), \\
T(\neg(A_1 <> A_2)) & \triangleq & A_1 = A_2, \\
T(\neg(A_1 > A_2)) & \triangleq & T(A_1 <= A_2), \\
T(\neg(A_1 >= A_2)) & \triangleq & A_1 < A_2, \\
T(\neg(B_1 \,|\, B_2)) & \triangleq & T(\neg(B_1)) \,\&\, T(\neg(B_2)), \\
T(\neg(B_1 \,\&\, B_2)) & \triangleq & T(\neg(B_1)) \,|\, T(\neg(B_2)), \\
T(\neg(\neg(B))) & \triangleq & T(B).
\end{array}
$$

Machine booleans

- $\mathbb{B}$: logical boolean values;
- $\mathbb{B}_\Omega$: machine truth values (including errors $\mathbb{E} = \{\Omega_{\mathtt{i}}, \Omega_{\mathtt{a}}\}$)

$$\mathbb{B} \triangleq \{\mathtt{tt}, \mathtt{ff}\}, \qquad \mathbb{B}_\Omega \triangleq \mathbb{B} \cup \mathbb{E}\ ;$$

Boolean operations

- $\underline{\mathtt{c}} \in \mathbb{I}_\Omega \times \mathbb{I}_\Omega \mapsto \mathbb{B}_\Omega$: machine arithmetic comparison operation;
- $\mathtt{c} \in \mathbb{Z} \times \mathbb{Z} \mapsto \mathbb{B}$: mathematical arithmetic comparison operation corresponding to the language binary arithmetic comparison operators $\mathtt{c} \in \{<, <=, =, <>, >=, >\}$;
- $\underline{\mathtt{u}} \in \mathbb{B}_\Omega \mapsto \mathbb{B}_\Omega$: machine boolean operation corresponding to the language unary operators $\mathtt{u} \in \{\neg\}$;
- $\mathtt{u} \in \mathbb{B} \mapsto \mathbb{B}$: mathematical boolean operation corresponding to the language unary operators $\mathtt{u} \in \{\neg\}$;
- $\underline{\mathtt{b}} \in \mathbb{B}_\Omega \times \mathbb{B}_\Omega \mapsto \mathbb{B}_\Omega$: machine boolean operation corresponding to the language binary boolean operators $\mathtt{b} \in \{\&, |\}$;
- $\mathtt{b} \in \mathbb{B} \times \mathbb{B} \mapsto \mathbb{B}$: mathematical boolean operation corresponding to the language binary boolean operators $\mathtt{b} \in \{\&, |\}$;

Evaluation

- Evaluation of operands, whence error propagation is left to right;
- We have ($e \in \mathbb{E}$, $v \in \mathbb{I}_\Omega$, $i, i_1, i_2 \in \mathbb{I}$, $b, b_1, b_2 \in \mathbb{B}$, $w \in \mathbb{B}_\Omega$)

$$
\begin{array}{rcl}
\Omega_e \,\underline{\mathtt{c}}\, v & \triangleq & \Omega_e, \\
i \,\underline{\mathtt{c}}\, \Omega_e & \triangleq & \Omega_e, \\
i_1 \,\underline{\mathtt{c}}\, i_2 & \triangleq & i_1 \,\mathtt{c}\, i_2\ ; \\
\underline{\mathtt{u}}\, \Omega_e & \triangleq & \Omega_e, \\
\underline{\mathtt{u}}\, b & \triangleq & \mathtt{u}\, b\ ;
\end{array}
\tag{49}
$$

$$
\begin{array}{rcl}
\Omega_e \,\underline{\mathtt{b}}\, w & \triangleq & \Omega_e, \\
b \,\underline{\mathtt{b}}\, \Omega_e & \triangleq & \Omega_e, \\
b_1 \,\underline{\mathtt{b}}\, b_2 & \triangleq & b_1 \,\mathtt{b}\, i_2\ .
\end{array}
\tag{50}
$$

Operational semantics of boolean expressions

$$\rho \vdash \texttt{true} \mapsto \mathrm{tt}, \quad \text{truth;} \tag{51}$$

$$\rho \vdash \texttt{false} \mapsto \mathrm{ff}, \quad \text{falsity;} \tag{52}$$

$$\frac{\rho \vdash A_1 \mapsto v_1,\ \rho \vdash A_2 \mapsto v_2}{\rho \vdash A_1\ \texttt{c}\ A_2 \mapsto v_1\ \underline{\texttt{c}}\ v_2}, \quad \text{arithmetic comparisons;} \tag{53}$$

$$\frac{\rho \vdash B \mapsto w}{\rho \vdash \texttt{u}\ B \mapsto \underline{\texttt{u}}\ w}, \quad \text{unary boolean operations;}$$

$$\frac{\rho \vdash B_1 \mapsto w_1,\ \rho \vdash B_2 \mapsto w_2}{\rho \vdash B_1\ \texttt{b}\ B_2 \mapsto w_1\ \underline{\texttt{b}}\ w_2}, \quad \text{binary boolean operations.}$$

Equivalence of boolean expressions

- In general, the semantics of a boolean expression B is not the same as the semantics of its transformed form $T(B)$;
- Example: if $\rho(\texttt{X}) = \Omega_{\texttt{i}}$ then $\rho \vdash \texttt{X} > \texttt{(1 / 0)} \mapsto \Omega_{\texttt{i}}$ while $\rho \vdash \texttt{(1 / 0)} < \texttt{X} \mapsto \Omega_{\texttt{a}}$;
- We will consider that all boolean expressions have been normalized (i.e. $B = T(B)$);
- We have [11]:

$$\forall b \in \mathbb{B} : \rho \vdash B \mapsto b \iff \rho \vdash T(B) \mapsto b,$$
$$(\exists e \in \mathbb{E} : \rho \vdash B \mapsto e) \iff (\exists e' \in \mathbb{E} : \rho \vdash T(B) \mapsto e') .$$

[11] the respective evaluations of B and $T(B)$ either produce the same boolean values (in general there is more than one possible value, because of random choice) or both expressions produce errors (which may be different).

Collecting semantics of boolean expressions

The *collecting semantics* $\mathrm{Cbexp}[\![B]\!]R$ of a boolean expression B defines the subset of possible environments $\rho \in R$ for which the boolean expression may evaluate to true (hence without producing a runtime error)

$$\mathrm{Cbexp} \in \mathrm{Bexp} \mapsto \wp(\mathbb{R}) \stackrel{\mathrm{cjm}}{\longmapsto} \wp(\mathbb{R}),$$
$$\mathrm{Cbexp}[\![B]\!]R \stackrel{\Delta}{=} \{\rho \in R \mid \rho \vdash B \mapsto \mathrm{tt}\} . \tag{54}$$

Abstract Interpretation of Boolean Expressions

Generic abstract interpretation of boolean expression

$$\mathrm{Abexp} \in \mathrm{Bexp} \mapsto (\mathbb{V} \mapsto L) \stackrel{\mathrm{mon}}{\longmapsto} (\mathbb{V} \mapsto L) \ .$$

- $\mathrm{Abexp}[\![B]\!]r$ is the environment r' specifying a necessary condition for the evaluation of boolean expression B in environments satisfying r to be true.

For any possible approximation (8) of value properties, the abstraction consists in approximating environment properties by the nonrelational abstraction (22) and in applying the following functional abstraction to the collecting semantics (54):

$$\langle \wp(\mathbb{R}) \stackrel{\mathrm{cjm}}{\longmapsto} \wp(\mathbb{R}),\ \dot{\subseteq} \rangle \underset{\ddot{\alpha}}{\overset{\ddot{\gamma}}{\leftrightarrows}} \langle (\mathbb{V} \mapsto L) \stackrel{\mathrm{mon}}{\longmapsto} (\mathbb{V} \mapsto L),\ \ddot{\sqsubseteq} \rangle \tag{55}$$

where:

$$\begin{aligned} \Phi \mathrel{\dot{\subseteq}} \Psi &\stackrel{\Delta}{=} \forall R \in \wp(\mathbb{R}) : \Phi(R) \subseteq \Psi(R), \\ \varphi \mathrel{\ddot{\sqsubseteq}} \psi &\stackrel{\Delta}{=} \forall r \in \mathbb{V} \mapsto L : \varphi(r) \mathrel{\dot{\sqsubseteq}} \psi(r), \\ \ddot{\alpha}(\Phi) &\stackrel{\Delta}{=} \dot{\alpha} \circ \Phi \circ \dot{\gamma}, \\ \ddot{\gamma}(\varphi) &\stackrel{\Delta}{=} \dot{\gamma} \circ \varphi \circ \dot{\alpha} \ . \end{aligned} \tag{56}$$

We must get an overapproximation such that:

$$\mathrm{Abexp}[\![B]\!] \mathrel{\ddot{\sqsupseteq}} \ddot{\alpha}(\mathrm{Cbexp}[\![B]\!]) \ . \tag{57}$$

Abstract interpretation of boolean expression

$$\begin{aligned} \mathrm{Abexp}[\![B]\!]\,\lambda \mathtt{Y}\cdot\bot &\stackrel{\Delta}{=} \lambda \mathtt{Y}\cdot\bot \qquad \text{if } \gamma(\bot) = \emptyset \\ \mathrm{Abexp}[\![\mathtt{true}]\!]r &\stackrel{\Delta}{=} r \\ \mathrm{Abexp}[\![\mathtt{false}]\!]r &\stackrel{\Delta}{=} \lambda \mathtt{Y}\cdot\bot \\ \mathrm{Abexp}[\![A_1 \mathtt{\ c\ } A_2]\!]r &\stackrel{\Delta}{=} \text{let } \langle p_1,\ p_2\rangle = \check{\mathtt{c}}(\mathrm{Faexp}^{\circ}[\![A_1]\!]r, \mathrm{Faexp}^{\circ}[\![A_2]\!]r) \text{ in} \\ &\qquad \mathrm{Baexp}^{\circ}[\![A_1]\!](r)p_1 \ \dot{\sqcap}\ \mathrm{Baexp}^{\circ}[\![A_2]\!](r)p_2 \\ \mathrm{Abexp}[\![B_1 \,\&\, B_2]\!]r &\stackrel{\Delta}{=} \mathrm{Abexp}[\![B_1]\!]r \ \dot{\sqcap}\ \mathrm{Abexp}[\![B_2]\!]r \\ \mathrm{Abexp}[\![B_1 \mid B_2]\!]r &\stackrel{\Delta}{=} \mathrm{Abexp}[\![B_1]\!]r \ \dot{\sqcup}\ \mathrm{Abexp}[\![B_2]\!]r \end{aligned} \tag{58}$$

parameterized by the following abstract comparison operations $\check{\mathtt{c}}$, $\mathtt{c} \in \{<, =\}$ on L:

$$\check{\mathtt{c}}(p_1, p_2) \mathrel{\dot{\sqsupseteq}}^2 \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma(p_1) \cap \mathbb{I} \wedge i_2 \in \gamma(p_2) \cap \mathbb{I} \wedge i_1 \mathrel{\underline{\mathtt{c}}} i_2 = \mathrm{tt}\}) \tag{59}$$

Monotony and reductivity

By induction on B, the operator $\mathrm{Abexp}[\![B]\!]$ on $\mathbb{V} \mapsto L$ is $\dot{\sqsubseteq}$-reductive and monotonic.

Generic static analyzer of boolean expressions

```
module type Abstract_Lattice_Algebra_signature =
  sig
    (* complete lattice of abstract properties of values          *)
    type lat                                  (* abstract properties  *)
    ...
    (* forward abstract interpretation of arithmetic expressions  *)
    ...
    (* backward abstract interpretation of arithmetic expressions *)
    ...
    (* abstract interpretation of boolean expressions             *)
    val a_EQ : lat -> lat -> lat * lat
    val a_LT : lat -> lat -> lat * lat
  end;;
```



```
module Abexp_implementation =
 functor (L: Abstract_Lattice_Algebra_signature) ->
 functor (E: Abstract_Env_Algebra_signature) ->
 functor (Faexp: Faexp_signature) ->
 functor (Baexp: Baexp_signature) ->
 struct
  open Abstract_Syntax
  (* generic abstract environments                                  *)
  module E' = E (L)
  (* generic forward abstract interpretation of arithmetic operations  *)
  module Faexp' = Faexp(L)(E)
  (* generic backward abstract interpretation of arithmetic operations *)
  module Baexp' = Baexp(L)(E)(Faexp)
  (* generic abstract interpretation of boolean operations          *)
  let rec abexp' b r =
    match b with
    | TRUE          -> r
    | FALSE         -> (E'.bot ())

 ...
```



```
...
   | (EQ (a1, a2))  ->
       let (p1,p2) = (L.a_EQ (Faexp'.faexp a1 r) (Faexp'.faexp a2 r))
         in (E'.meet (Baexp'.baexp a1 r p1) (Baexp'.baexp a2 r p2))
   | (LT (a1, a2))  ->
       let (p1,p2) = (L.a_LT (Faexp'.faexp a1 r) (Faexp'.faexp a2 r))
         in (E'.meet (Baexp'.baexp a1 r p1) (Baexp'.baexp a2 r p2))
   | (AND (b1, b2)) -> (E'.meet (abexp' b1 r) (abexp' b2 r))
   | (OR (b1, b2))  -> (E'.join (abexp' b1 r) (abexp' b2 r))
 let abexp b r =
   if (E'.is_bot r) & (L.isbotempty ()) then (E'.bot ()) else abexp' b r
end;;
```


Generic abstract boolean equality

$$p_1 \mathrel{\check{=}} p_2 \stackrel{\Delta}{=} \text{let } p = p_1 \sqcap p_2 \sqcap ?^* \text{ in } \langle p,\ p\rangle\ .$$

$$
\begin{array}{ll}
 & \alpha^2(\{\langle i_1,\ i_2\rangle \mid i_1 \in \gamma(p_1) \cap \mathbb{I} \wedge i_2 \in \gamma(p_2) \cap \mathbb{I} \wedge i_1 \underline{=} i_2 = \mathrm{tt}\}) \\
= & \quad \wr \text{def. (49) of } \underline{=} \int \\
 & \alpha^2(\{\langle i,\ i\rangle \mid i \in \gamma(p_1) \cap \gamma(p_2) \cap \mathbb{I}\}) \\
\sqsubseteq^2 & \quad \wr \gamma \circ \alpha \text{ is extensive (5) and } \alpha^2 \text{ is monotone} \int \\
 & \alpha^2(\{\langle i,\ i\rangle \mid i \in \gamma(p_1) \cap \gamma(p_2) \cap \gamma(\alpha(\mathbb{I}))\}) \\
= & \quad \wr \gamma \text{ preserves meets} \int \\
 & \alpha^2(\{\langle i,\ i\rangle \mid i \in \gamma(p_1 \sqcap p_2 \sqcap \alpha(\mathbb{I}))\}) \\
= & \quad \wr \text{def. (10) of } \gamma^2 \int \\
 & \alpha^2(\gamma^2(\langle p_1 \sqcap p_2 \sqcap \alpha(\mathbb{I}),\ p_1 \sqcap p_2 \sqcap \alpha(\mathbb{I})\rangle)) \\
\sqsubseteq^2 = & \quad \wr \alpha^2 \circ \gamma^2 \text{ is reductive and let notation} \int \\
 & \text{let } p = p_1 \sqcap p_2 \sqcap \alpha(\mathbb{I}) \text{ in } \langle p,\ p\rangle \\
\sqsubseteq^2 & \quad \wr \text{def. (40) of } ?^\circ \int \\
 & \text{let } p = p_1 \sqcap p_2 \sqcap ?^\circ \text{ in } \langle p,\ p\rangle \\
= & \quad \wr \text{by defining } \mathbin{\check{=}} \stackrel{\Delta}{=} \text{let } p = p_1 \sqcap p_2 \sqcap ?^\circ \text{ in } \langle p,\ p\rangle \int \\
 & \mathbin{\check{=}} .
\end{array}
$$

Initialization and simple sign abstract arithmetic comparison operations

$\check{<}(p_1, p_2)$		p_2				
		BOT, ERR	NEG	ZERO	POS	INI, TOP
p_1	BOT, ERR	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩
	NEG	⟨BOT, BOT⟩	⟨NEG, NEG⟩	⟨NEG, ZERO⟩	⟨NEG, POS⟩	⟨NEG, INI⟩
	ZERO	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨ZERO, POS⟩	⟨ZERO, POS⟩
	POS	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨BOT, BOT⟩	⟨POS, POS⟩	⟨POS, POS⟩
	INI, TOP	⟨BOT, BOT⟩	⟨NEG, NEG⟩	⟨NEG, ZERO⟩	⟨INI, POS⟩	⟨INI, INI⟩

Reductive Iteration

Iterating monotone and reductive abstract operators

If $\langle M,\ \preceq\rangle$ is poset, $f \in M \mapsto M$ is monotone and reductive,

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

is a Galois connection, $\langle L,\ \sqsubseteq,\ \sqcap\rangle$ is a dual dcpo, $g \in L \mapsto L$ is monotone and reductive and $\alpha \circ f \circ \gamma \mathbin{\dot{\sqsubseteq}} g$ then the lower closure operator $g^\star \stackrel{\Delta}{=} \lambda x \cdot \mathrm{gfp}_x^{\sqsubseteq}\, g$ is a better abstract interpretation of f than g:

$$\alpha \circ f \circ \gamma \mathbin{\dot{\sqsubseteq}} g^\star \mathbin{\dot{\sqsubseteq}} g\ .$$

Reductive iteration for boolean and arithmetic expressions

- Analysis of boolean expressions with reductive iteration:

$$\ddot{\alpha}(\mathrm{Cbexp}[\![B]\!]) \ddot{\sqsubseteq} \mathrm{Abexp}[\![B]\!]^{\star} \ddot{\sqsubseteq} \mathrm{Abexp}[\![B]\!] .$$

- Backward analysis of arithmetic expressions with reductive iteration:

$$\forall p \in L : \lambda r \cdot \alpha^{\circ}(\mathrm{Baexp}[\![A]\!])(r)p \ddot{\sqsubseteq} (\lambda r \cdot \mathrm{Baexp}^{\circ}[\![A]\!](r)p)^{\star} \ddot{\sqsubseteq} \lambda r \cdot \mathrm{Baexp}^{\circ}[\![A]\!](r)p .$$

Generic implementation of reductive iteration

```
module type Poset_signature =
  sig
    type element
    val leq : element -> element -> bool
  end;;

module type Fixpoint_signature =
  functor (P:Poset_signature) ->
  sig
    val lfp : (P.element -> P.element) -> P.element -> P.element
    val gfp : (P.element -> P.element) -> P.element -> P.element
  end;;

 ...
```



```
module Fixpoint_implementation =
  functor (P:Poset_signature) ->
  struct
    (* iterative computation of the least fixpoint of f greater *)
    (* than or equal to the prefixpoint x (f(x) >= x)           *)
    let rec lfp f x =
      let x' = (f x) in
        if (P.leq x' x) then x'
        else lfp f x'
    (* iterative computation of the greatest fixpoint of f less *)
    (* than or equal to the postfixpoint x (f(x) <= x)          *)
    let rec gfp f x =
      let x' = (f x) in
        if (P.leq x x') then x
else gfp f x'
  end;;

module Fixpoint = (Fixpoint_implementation:Fixpoint_signature);;
```



```
module Baexp_Reductive_Iteration_implementation =
 functor (Baexp: Baexp_signature) ->
 functor (L: Abstract_Lattice_Algebra_signature) ->
 functor (E: Abstract_Env_Algebra_signature) ->
 functor (Faexp: Faexp_signature) ->
 struct
  (* generic abstract elementironments *)
  module E' = E (L)
  (* iterative fixpoint computation *)
  module F = Fixpoint((E':Poset_signature with type element = E (L).env))
  (* generic backward abstract interpretation of arithmetic operations *)
  module Baexp' = Baexp(L)(E)(Faexp)
  (* generic reductive backward abstract int. of arithmetic operations *)
  let baexp a r p = let f x = Baexp'.baexp a x p in F.gfp f r
 end;;

module Baexp_Reductive_Iteration =
 (Baexp_Reductive_Iteration_implementation (Baexp):Baexp_signature);;
```

Example

without reductive iteration:

```
{ x:ERR; y:ERR; z:ERR }
  x := 0; y := ?; z := ?;
{ x:ZERO; y:INI; z:INI }
  if ((x=y)&(y=z)&((z+1)=x)) then
    { x:ZERO; y:ZERO; z:NEG }
      skip
  else
    { x:ZERO; y:INI; z:INI }
      skip
  fi
{ x:ZERO; y:INI; z:INI }
```

with reductive iteration:

```
{ x:ERR; y:ERR; z:ERR }
  x := 0; y := ?; z := ?;
{ x:ZERO; y:INI; z:INI }
  if ((x=y)&(y=z)&((z+1)=x)) then
    { x:BOT; y:BOT; z:BOT }
      skip
  else
    { x:ZERO; y:INI; z:INI }
      skip
  fi
{ x:ZERO; y:INI; z:INI }
```


Semantics of Imperative Programs

Abstract syntax of commands and programs

We have already defined:

Variables

$\mathtt{X} \in \mathbb{V}$.

Arithmetic expressions

$A \in \mathrm{Aexp}$.

Boolean expressions

$B \in \mathrm{Bexp}$.

so that we can define the abstract syntax of commands and programs:

Commands

$$
\begin{array}{rcll}
C \in \mathrm{Com} & ::= & \mathtt{skip} & \text{identity,} \\
& | & \mathtt{X} := A & \text{assignment,} \\
& | & \mathtt{if}\ B\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2\ \mathtt{fi} & \text{conditional,} \\
& | & \mathtt{while}\ B\ \mathtt{do}\ S\ \mathtt{od} & \text{iteration.}
\end{array}
$$

List of commands

$$
\begin{array}{rcll}
S, S_1, S_2 \in \mathrm{Seq} & ::= & C & \text{command,} \\
& | & C\ ;\ S & \text{sequence.}
\end{array}
$$

Programs

$$
P \in \mathrm{Prog} ::= S\ ;; \qquad \text{program.}
$$

Program components

$$
\begin{aligned}
\mathrm{Cmp}[\![S\ ;\ ;]\!] &\triangleq \{\lfloor S\ ;\ ;\rfloor_0\} \cup \mathrm{Cmp}^0[\![S]\!],\\
\mathrm{Cmp}^\pi[\![C_1\ ;\ \ldots\ ;\ C_n]\!] &\triangleq \{\lfloor C_1\ ;\ \ldots\ ;\ C_n\rfloor_\pi\} \cup \bigcup_{i=1}^{n} \mathrm{Cmp}^{\pi.i}[\![C_i]\!],\\
\mathrm{Cmp}^\pi[\![\texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}]\!] &\triangleq \{\lfloor \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \texttt{fi}\rfloor_\pi\} \cup\\
&\qquad \mathrm{Cmp}^{\pi.1}[\![S_1]\!] \cup \mathrm{Cmp}^{\pi.2}[\![S_2]\!],\\
\mathrm{Cmp}^\pi[\![\texttt{while}\ B\ \texttt{do}\ S_1\ \texttt{od}]\!] &\triangleq \{\lfloor \texttt{while}\ B\ \texttt{do}\ S_1\ \texttt{od}\rfloor_\pi\} \cup \mathrm{Cmp}^{\pi.1}[\![S_1]\!],\\
\mathrm{Cmp}^\pi[\![\texttt{X := } A]\!] &\triangleq \{\lfloor \texttt{X := } A\rfloor_\pi\},\\
\mathrm{Cmp}^\pi[\![\texttt{skip}]\!] &\triangleq \{\lfloor \texttt{skip}\rfloor_\pi\}\ .
\end{aligned}
$$

Example: $\mathrm{Cmp}[\![\texttt{skip ; skip ; ;}]\!] = \{\lfloor \texttt{skip ; skip ; ;}\rfloor_0, \lfloor \texttt{skip}\rfloor_{01}, \lfloor \texttt{skip}\rfloor_{02}\}$.

Program labelling

We prefer labels $\ell \in \mathrm{Lab}$ designating program points ($P \in \mathrm{Prog}$)

$$
\begin{aligned}
\mathrm{at}_P, \mathrm{after}_P &\in \mathrm{Cmp}[\![P]\!] \mapsto \mathrm{Lab},\\
\mathrm{in}_P &\in \mathrm{Cmp}[\![P]\!] \mapsto \wp(\mathrm{Lab})\ .
\end{aligned}
$$

Definition of program labelling

- If $C = \texttt{skip} \in \mathrm{Cmp}[\![P]\!]$ or $C = \texttt{X := } A \in \mathrm{Cmp}[\![P]\!]$ then

$$\mathrm{in}_P[\![C]\!] = \{\mathrm{at}_P[\![C]\!], \mathrm{after}_P[\![C]\!]\}\ . \tag{60}$$

- If $S = C_1; \ldots ; C_n \in \mathrm{Cmp}[\![P]\!]$ where $n \geq 1$ is a sequence of commands, then

$$
\begin{aligned}
\mathrm{at}_P[\![S]\!] &= \mathrm{at}_P[\![C_1]\!],\\
\mathrm{after}_P[\![S]\!] &= \mathrm{after}_P[\![C_n]\!],\\
\mathrm{in}_P[\![S]\!] &= \bigcup_{i=1}^{n} \mathrm{in}_P[\![C_i]\!],\\
\forall i \in [1, n[&: \mathrm{after}_P[\![C_i]\!] = \mathrm{at}_P[\![C_{i+1}]\!] = \mathrm{in}_P[\![C_i]\!] \cap \mathrm{in}_P[\![C_{i+1}]\!],\\
\forall i, j \in [1, n] &: (j \neq i-1 \wedge j \neq i+1) \implies (\mathrm{in}_P[\![C_i]\!] \cap \mathrm{in}_P[\![C_j]\!] = \emptyset)\ .
\end{aligned}
\tag{61}
$$

- If $C = \texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi} \in \mathrm{Cmp}[\![P]\!]$ is a conditional command, then

$$
\begin{aligned}
\mathrm{in}_P[\![C]\!] &= \{\mathrm{at}_P[\![C]\!], \mathrm{after}_P[\![C]\!]\} \cup \mathrm{in}_P[\![S_t]\!] \cup \mathrm{in}_P[\![S_f]\!],\\
\{\mathrm{at}_P[\![C]\!], \mathrm{after}_P[\![C]\!]\} &\cap (\mathrm{in}_P[\![S_t]\!] \cup \mathrm{in}_P[\![S_f]\!]) = \emptyset,\\
\mathrm{in}_P[\![S_t]\!] &\cap \mathrm{in}_P[\![S_f]\!] = \emptyset\ .
\end{aligned}
\tag{62}
$$

- If $C = \texttt{while } B \texttt{ do } S \texttt{ od} \in \mathrm{Cmp}[\![P]\!]$ is an iteration command, then

$$
\begin{aligned}
\mathrm{in}_P[\![C]\!] &= \{\mathrm{at}_P[\![C]\!], \mathrm{after}_P[\![C]\!]\} \cup \mathrm{in}_P[\![S]\!],\\
\{\mathrm{at}_P[\![C]\!], \mathrm{after}_P[\![C]\!]\} &\cap \mathrm{in}_P[\![S]\!] = \emptyset\ .
\end{aligned}
\tag{63}
$$

- If $P = S\ ;\ ; \in \mathrm{Cmp}[\![P]\!]$ is a program, then

$$\mathrm{at}_P[\![P]\!] = \mathrm{at}_P[\![S]\!], \qquad \mathrm{after}_P[\![P]\!] = \mathrm{after}_P[\![S]\!], \qquad \mathrm{in}_P[\![P]\!] = \mathrm{in}_P[\![S]\!]\ .$$

Program variables

The free variables

$$\mathrm{Var} \in (\mathrm{Prog} \cup \mathrm{Com} \cup \mathrm{Seq} \cup \mathrm{Aexp} \cup \mathrm{Bexp}) \mapsto \wp(\mathbb{V})$$

are defined as usual for:

- *programs* ($S \in \mathrm{Seq}$)

$$\mathrm{Var}[\![S\ ;;]\!] \stackrel{\Delta}{=} \mathrm{Var}[\![S]\!]\ ;$$

- *list of commands* ($C \in \mathrm{Com}$, $S \in \mathrm{Seq}$)

$$\mathrm{Var}[\![C\ ;\ S]\!] \stackrel{\Delta}{=} \mathrm{Var}[\![C]\!] \cup \mathrm{Var}[\![S]\!]\ ;$$

- *commands* ($\mathtt{X} \in \mathbb{V}$, $A \in \mathrm{Aexp}$, $B \in \mathrm{Bexp}$, S, S_t, $S_f \in \mathrm{Seq}$)

$$\begin{aligned}
\mathrm{Var}[\![\mathtt{skip}]\!] &\stackrel{\Delta}{=} \emptyset,\\
\mathrm{Var}[\![\mathtt{X := } A]\!] &\stackrel{\Delta}{=} \{\mathtt{X}\} \cup \mathrm{Var}[\![A]\!],\\
\mathrm{Var}[\![\mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![B]\!] \cup \mathrm{Var}[\![S_t]\!] \cup \mathrm{Var}[\![S_f]\!],\\
\mathrm{Var}[\![\mathtt{while}\ B\ \mathtt{do}\ S\ \mathtt{od}]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![B]\!] \cup \mathrm{Var}[\![S]\!]\ ;
\end{aligned}$$

- *arithmetic expressions* ($\mathtt{n} \in \mathrm{Nat}$, $\mathtt{X} \in \mathbb{V}$, $\mathtt{u} \in \{+, -\}$, A_1, $A_2 \in \mathrm{Aexp}$, $\mathtt{b} \in \{+, -, *, /, \mathtt{mod}\}$)

$$\begin{aligned}
\mathrm{Var}[\![\mathtt{n}]\!] &\stackrel{\Delta}{=} \emptyset, & \mathrm{Var}[\![\mathtt{u}\ A_1]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![A_1]\!],\\
\mathrm{Var}[\![\mathtt{X}]\!] &\stackrel{\Delta}{=} \{\mathtt{X}\}, & \mathrm{Var}[\![A_1\ \mathtt{b}\ A_2]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![A_1]\!] \cup \mathrm{Var}[\![A_2]\!],\\
\mathrm{Var}[\![?]\!] &\stackrel{\Delta}{=} \emptyset
\end{aligned}$$

- *boolean expressions* (A_1, $A_2 \in \mathrm{Aexp}$, $\mathtt{r} \in \{=, <\}$, B_1, $B_2 \in \mathrm{Bexp}$, $\mathtt{l} \in \{\&, |\}$)

$$\begin{aligned}
\mathrm{Var}[\![\mathtt{true}]\!] &\stackrel{\Delta}{=} \emptyset, & \mathrm{Var}[\![A_1\ \mathtt{r}\ A_2]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![A_1]\!] \cup \mathrm{Var}[\![A_2]\!],\\
\mathrm{Var}[\![\mathtt{false}]\!] &\stackrel{\Delta}{=} \emptyset, & \mathrm{Var}[\![B_1\ \mathtt{l}\ B_2]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![B_1]\!] \cup \mathrm{Var}[\![B_2]\!]\ .
\end{aligned}$$

Program environments

- During execution of program $P \in \mathrm{Prog}$, an environment $\rho \in \mathrm{Env}[\![P]\!] \subseteq \mathbb{R}$ maps program variables $\mathtt{X} \in \mathrm{Var}[\![P]\!]$ to their value $\rho(\mathtt{X})$. We define:

$$\begin{aligned}
\mathrm{Env} &\in \mathrm{Prog} \mapsto \wp(\mathbb{R}),\\
\mathrm{Env}[\![P]\!] &\stackrel{\Delta}{=} \mathrm{Var}[\![P]\!] \mapsto \mathbb{I}_\Omega\ .
\end{aligned}$$

Program states

- States $\langle \ell,\ \rho \rangle \in \Sigma[\![P]\!]$ record a program point $\ell \in \mathrm{in}_P[\![P]\!]$ and an environment $\rho \in \mathrm{Env}[\![P]\!]$ assigning values to variables. The definition of states is

$$\begin{aligned} \Sigma &\in \mathrm{Prog} \mapsto \wp(\mathbb{V} \times \mathbb{R}), \\ \Sigma[\![P]\!] &\triangleq \mathrm{in}_P[\![P]\!] \times \mathrm{Env}[\![P]\!] \ . \end{aligned} \tag{64}$$

Small-step operational semantics of commands

- The small-step operational semantics [11] of commands, sequences and programs $C \in \mathrm{Com} \cup \mathrm{Seq} \cup \mathrm{Prog}$ within a program $P \in \mathrm{Prog}$ involves transition judgements:

$$\langle \ell,\ \rho \rangle \longmapsto[\![C]\!]\Longrightarrow \langle \ell',\ \rho' \rangle \ .$$

- Such judgements mean that if execution is at control point $\ell \in \mathrm{in}_P[\![C]\!]$ in environment $\rho \in \mathrm{Env}[\![P]\!]$ then the next computation step within command C leads to program control point $\ell' \in \mathrm{in}_P[\![C]\!]$ in the new environment $\rho' \in \mathrm{Env}[\![P]\!]$.

Small-step operational semantics of commands

Identity $C = \texttt{skip}$ ($\mathrm{at}_P[\![C]\!] = \ell$ and $\mathrm{after}_P[\![C]\!] = \ell'$)

$$\langle \ell,\ \rho \rangle \longmapsto[\![\texttt{skip}]\!]\Longrightarrow \langle \ell',\ \rho \rangle \ . \tag{65}$$

Assignment $C = \texttt{X := } A$ ($\mathrm{at}_P[\![C]\!] = \ell$ and $\mathrm{after}_P[\![C]\!] = \ell'$)

$$\frac{\rho \vdash A \mapsto i}{\langle \ell,\ \rho \rangle \longmapsto[\![\texttt{X := } A]\!]\Longrightarrow \langle \ell',\ \rho[\texttt{X} \leftarrow i] \rangle},\ i \in \mathbb{I} \ . \tag{66}$$

Conditional $C = \texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}$ ($\mathrm{at}_P[\![C]\!] = \ell$ and $\mathrm{after}_P[\![C]\!] = \ell'$)

$$\frac{\rho \vdash B \mapsto \mathrm{tt}}{\langle \ell,\ \rho \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \mathrm{at}_P[\![S_t]\!],\ \rho \rangle} \ , \tag{67}$$

$$\frac{\rho \vdash T(\neg B) \mapsto \mathrm{tt}}{\langle \ell,\ \rho \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \mathrm{at}_P[\![S_f]\!],\ \rho \rangle} \ . \tag{68}$$

$$\frac{\langle \ell_1,\ \rho_1 \rangle \longmapsto[\![S_t]\!]\Longrightarrow \langle \ell_2,\ \rho_2 \rangle}{\langle \ell_1,\ \rho_1 \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \ell_2,\ \rho_2 \rangle} \ , \tag{69}$$

$$\frac{\langle \ell_1,\ \rho_1 \rangle \longmapsto[\![S_f]\!]\Longrightarrow \langle \ell_2,\ \rho_2 \rangle}{\langle \ell_1,\ \rho_1 \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \ell_2,\ \rho_2 \rangle} \ . \tag{70}$$

$$\langle \mathrm{after}_P[\![S_t]\!],\ \rho \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \ell',\ \rho \rangle \ , \tag{71}$$

$$\langle \mathrm{after}_P[\![S_f]\!],\ \rho \rangle \longmapsto[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!]\Longrightarrow \langle \ell',\ \rho \rangle \ . \tag{72}$$

Iteration $C = \texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}$ ($\mathrm{at}_P[\![C]\!] = \ell$, $\mathrm{after}_P[\![C]\!] = \ell'$ and $\ell_1, \ell_2 \in \mathrm{in}_P[\![S]\!]$)

$$\frac{\rho \vdash T(\neg B) \Rightarrow \mathrm{tt}}{\langle \ell,\ \rho\rangle \mapsto\!\!\!\!\Longrightarrow [\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!] \langle \ell',\ \rho\rangle}\ , \tag{73}$$

$$\frac{\rho \vdash B \Rightarrow \mathrm{tt}}{\langle \ell,\ \rho\rangle \mapsto\!\!\!\!\Longrightarrow [\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!] \langle \mathrm{at}_P[\![S]\!],\ \rho\rangle}\ , \tag{74}$$

$$\frac{\langle \ell_1,\ \rho_1\rangle \mapsto\!\!\!\!\Longrightarrow [\![S]\!] \langle \ell_2,\ \rho_2\rangle}{\langle \ell_1,\ \rho_1\rangle \mapsto\!\!\!\!\Longrightarrow [\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!] \langle \ell_2,\ \rho_2\rangle}\ , \tag{75}$$

$$\langle \mathrm{after}_P[\![S]\!],\ \rho\rangle \mapsto\!\!\!\!\Longrightarrow [\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!] \langle \ell,\ \rho\rangle\ . \tag{76}$$

Sequence $C_1\ ;\ \ldots\ ;\ C_n$, $n > 0$ ($\ell_i, \ell_{i+1} \in \mathrm{in}_P[\![C_i]\!]$ for all $i \in [1, n]$)

$$\frac{\langle \ell_i,\ \rho_i\rangle \mapsto\!\!\!\!\Longrightarrow [\![C_i]\!] \langle \ell_{i+1},\ \rho_{i+1}\rangle}{\langle \ell_i,\ \rho_i\rangle \mapsto\!\!\!\!\Longrightarrow [\![C_1\ ;\ \ldots\ ;\ C_n]\!] \langle \ell_{i+1},\ \rho_{i+1}\rangle}\ . \tag{77}$$

Program $P = S\ ;;$

$$\frac{\langle \ell,\ \rho\rangle \mapsto\!\!\!\!\Longrightarrow [\![S]\!] \rho}{\langle \ell',\ \rho'\rangle \mapsto\!\!\!\!\Longrightarrow [\![S\ ;;]\!] \langle \ell',\ \rho'\rangle}\ . \tag{78}$$

Transition system of a program

The transition system of a program $P = S\ ;;$ is

$$\langle \Sigma[\![P]\!],\ \tau[\![P]\!]\rangle$$

where $\Sigma[\![P]\!]$ is the set (64) of program states and $\tau[\![C]\!]$, $C \in \mathrm{Cmp}[\![P]\!]$ is the transition relation for component C of program P, defined by:

$$\tau[\![C]\!] \stackrel{\Delta}{=} \{\langle \langle \ell,\ \rho\rangle,\ \langle \ell',\ \rho'\rangle\rangle \mid \langle \ell,\ \rho\rangle \mapsto\!\!\!\!\Longrightarrow [\![C]\!] \langle \ell',\ \rho'\rangle\}\ . \tag{79}$$

Starting and stopping execution ...

- Execution starts at the program entry point with all variables uninitialized:

$$\mathrm{Entry}[\![P]\!] \stackrel{\Delta}{=} \{\langle \mathrm{at}_P[\![P]\!],\ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \Omega_{\mathtt{i}}\rangle\}\ . \tag{80}$$

- Execution ends without error when control reaches the program exit point:

$$\mathrm{Exit}[\![P]\!] \stackrel{\Delta}{=} \{\mathrm{after}_P[\![P]\!]\} \times \mathrm{Env}[\![P]\!]\ .$$

- When the evaluation of an arithmetic or boolean expression fails with a runtime error, the program execution is blocked so that no further transition is possible.

No jumps out of commands

- A basic result on the program transition relation is that it is not possible to jump into or out of program components ($C \in \mathrm{Cmp}[\![P]\!]$))

$$\langle\langle \ell,\ \rho\rangle,\ \langle \ell',\ \rho'\rangle\rangle \in \tau[\![C]\!] \implies \{\ell, \ell'\} \subseteq \mathrm{in}_P[\![C]\!]\ . \tag{81}$$

Reflexive transitive closure of the program transition relation

- The reflexive transitive closure of the transition relation $\tau[\![C]\!]$ of a program component $C \in \mathrm{Cmp}[\![P]\!]$ is $\tau^\star[\![C]\!] \triangleq (\tau[\![C]\!])^\star$.
- $\tau^\star[\![P]\!]$ can be expressed compositionally (by structural induction the the components $C \in \mathrm{Cmp}[\![P]\!]$ of program P).

Big step operational semantics

- $$\tau^\star[\![\texttt{skip}]\!] = 1_{\Sigma[\![P]\!]} \cup \tau[\![\texttt{skip}]\!] \tag{82}$$
- $$\tau^\star[\![\texttt{X := } A]\!] = 1_{\Sigma[\![P]\!]} \cup \tau[\![\texttt{X := } A]\!]$$

- $$\tau^\star[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!] = (1_{\Sigma[\![P]\!]} \cup \tau^B) \circ \tau^\star[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t) \ \cup \ (1_{\Sigma[\![P]\!]} \cup \tau^{\bar{B}}) \circ \tau^\star[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^f) \tag{83}$$

where:

$$\begin{aligned}
\tau^B &\triangleq \{\langle\langle \mathrm{at}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle,\ \langle \mathrm{at}_P[\![S_t]\!],\ \rho\rangle\rangle \mid \rho \vdash B \Mapsto \mathrm{tt}\}\\
\tau^{\bar{B}} &\triangleq \{\langle\langle \mathrm{at}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle,\ \langle \mathrm{at}_P[\![S_f]\!],\ \rho\rangle\rangle \mid \rho \vdash T(\neg B) \Mapsto \mathrm{tt}\}\\
\tau^t &\triangleq \{\langle\langle \mathrm{after}_P[\![S_t]\!],\ \rho\rangle,\ \langle \mathrm{after}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle\rangle \mid \rho \in \mathrm{Env}[\![P]\!]\}\\
\tau^f &\triangleq \{\langle\langle \mathrm{after}_P[\![S_f]\!],\ \rho\rangle,\ \langle \mathrm{after}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle\rangle \mid \rho \in \mathrm{Env}[\![P]\!]\}
\end{aligned}$$

- $\tau^\star[\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!] = (1_{\Sigma[\![P]\!]} \cup \tau^\star[\![S]\!] \circ \tau^R) \circ (\tau^B \circ \tau^\star[\![S]\!] \circ \tau^R)^\star \circ (1_{\Sigma[\![P]\!]} \cup \tau^B \circ \tau^\star[\![S]\!] \cup \tau^{\bar{B}})$ (84)

where:

$\tau^B \triangleq \{\langle\langle \mathrm{at}_P[\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!],\ \rho\rangle,\ \langle \mathrm{at}_P[\![S]\!],\ \rho\rangle\rangle \mid \rho \vdash B \Rightarrow \mathrm{tt}\}$

$\tau^{\bar{B}} \triangleq \{\langle\langle \mathrm{at}_P[\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!],\ \rho\rangle,\ \langle \mathrm{after}_P[\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!],\ \rho\rangle\rangle \mid \rho \vdash T(\neg B) \Rightarrow \mathrm{tt}\}$

$\tau^R \triangleq \{\langle\langle \mathrm{after}_P[\![S]\!],\ \rho\rangle,\ \langle \mathrm{at}_P[\![\texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}]\!],\ \rho\rangle\rangle \mid \rho \in \mathrm{Env}[\![P]\!]\}$

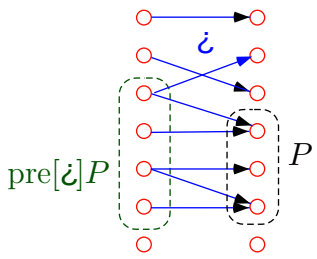
- $\tau^\star[\![C_1\ ;\ \ldots\ ;\ C_n]\!] = \tau^\star[\![C_1]\!] \circ \ldots \circ \tau^\star[\![C_n]\!]$ (85)
- $\tau^\star[\![S\ ;;]\!] = \tau^\star[\![S]\!]\ .$ (86)

Predicate transformers and fixpoints

Pre-image

- The *pre-image* $\mathrm{pre}[t]\,P$ of a set $P \subseteq S$ of states by a transition relation $t \subseteq S \times S$ is the set of states from which it is possible to reach a state in P by a transition t:

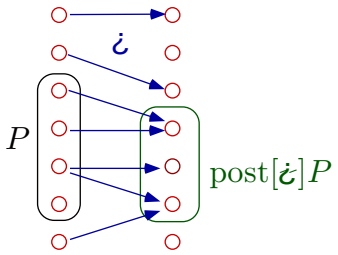
$$\mathrm{pre}[t]\,P \triangleq \{s \mid \exists s' : \langle s,\ s'\rangle \in t \wedge s' \in P\}\ .$$



Post-image

- The *post-image* $\mathrm{post}[t]\,P$ is the inverse pre-image, that is the set of states which are reachable from $P \subseteq S$ by a transition t:

$$\mathrm{post}[t]\,P \stackrel{\Delta}{=} \mathrm{pre}[t^{-1}]\,P = \{s' \mid \exists s : s \in P \wedge \langle s,\ s'\rangle \in t\}\ . \quad (87)$$



Dual pre-image

- The *dual pre-image* $\widetilde{\mathrm{pre}}[t]\,P$ is the set of states from which any transition, if any, must lead to a state in P:

$$\widetilde{\mathrm{pre}}[t]\,P \stackrel{\Delta}{=} \neg\,\mathrm{pre}[t](\neg P) = \{s \mid \forall s' : \langle s,\ s'\rangle \in t \implies s' \in P\}\ .$$

Dual post-image

- The *dual post-image* $\widetilde{\mathrm{post}}[t]\,P$ is the set of states which can only be reached, if ever possible, by a transition t from P:

$$\widetilde{\mathrm{post}}[t]\,P \stackrel{\Delta}{=} \neg\,\mathrm{post}[t](\neg P) = \{s' \mid \forall s : \langle s,\ s'\rangle \in t \implies s \in P\}\ .$$

Galois connections

We have the Galois connections ($t \subseteq S \times S$)

$$\langle \wp(S),\ \subseteq\rangle \underset{\mathrm{post}[t]}{\overset{\widetilde{\mathrm{pre}}[t]}{\leftrightarrows}} \langle \wp(S),\ \subseteq\rangle, \qquad \langle \wp(S),\ \subseteq\rangle \underset{\mathrm{pre}[t]}{\overset{\widetilde{\mathrm{post}}[t]}{\leftrightarrows}} \langle \wp(S),\ \subseteq\rangle$$

as well as ($P \subseteq S$, $\gamma_P(Y) \stackrel{\Delta}{=} \{\langle s,\ s'\rangle \mid s \in P \implies s' \in Y\}$)

$$\langle \wp(S \times S),\ \subseteq\rangle \underset{\lambda t\cdot\,\mathrm{post}[t]\,P}{\overset{\gamma_P}{\leftrightarrows}} \langle \wp(S),\ \subseteq\rangle, \quad (88)$$

$$\langle \wp(S \times S),\ \subseteq\rangle \underset{\lambda t\cdot\,\mathrm{pre}[t]\,P}{\overset{\gamma_P{}^{-1}}{\leftrightarrows}} \langle \wp(S),\ \subseteq\rangle\ . \quad (89)$$

Properties of predicate transformers

$$\begin{aligned}
\mathrm{post}[t_1 \circ t_2] &= \mathrm{post}[t_2] \circ \mathrm{post}[t_1], && (90)\\
\mathrm{pre}[t_1 \circ t_2] &= \mathrm{pre}[t_1] \circ \mathrm{pre}[t_2],\\
\mathrm{post}[1_S]\, P &= P && (91)\\
\mathrm{pre}[1_S]\, P &= P\ .
\end{aligned}$$

Reachable states collecting semantics

Fixpoint characterizations

$$\begin{aligned}
\mathrm{pre}[t^\star]\, F &= \mathrm{lfp}^{\subseteq}\, \lambda X \cdot F \cup \mathrm{pre}[t]\, X &&= \mathrm{lfp}^{\subseteq}_{F}\, \lambda X \cdot X \cup \mathrm{pre}[t]\, X,\\
\widetilde{\mathrm{pre}}[t^\star]\, F &= \mathrm{gfp}^{\subseteq}\, \lambda X \cdot F \cap \widetilde{\mathrm{pre}}[t]\, X &&= \mathrm{gfp}^{\subseteq}_{F}\, \lambda X \cdot X \cap \widetilde{\mathrm{pre}}[t]\, X,\\
\mathrm{post}[t^\star]\, I &= \mathrm{lfp}^{\subseteq}\, \lambda X \cdot I \cup \mathrm{post}[t]\, X &&= \mathrm{lfp}^{\subseteq}_{I}\, \lambda X \cdot X \cup \mathrm{post}[t]\, X, && (92)\\
\widetilde{\mathrm{post}}[t^\star]\, I &= \mathrm{gfp}^{\subseteq}\, \lambda X \cdot I \cap \widetilde{\mathrm{post}}[t]\, X &&= \mathrm{gfp}^{\subseteq}_{I}\, \lambda X \cdot X \cap \widetilde{\mathrm{post}}[t]\, X\ .
\end{aligned}$$

Reachable states collecting semantics

- The reachable states collecting semantics of a component $C \in \mathrm{Cmp}[\![P]\!]$ of a program $P \in \mathrm{Prog}$ is the set $\mathrm{post}[\tau^\star[\![C]\!]](In)$ of states which are reachable from a given set $In \in \wp(\Sigma[\![P]\!])$ of initial states;
- In particular when C is the program P, the initial states are the entry states $In = \mathrm{Entry}[\![P]\!]$.
- The program analysis problem we are interested in is to effectively compute a machine representable program invariant $J \in \wp(\Sigma[\![P]\!])$ such that:

$$\mathrm{post}[\tau^\star[\![C]\!]]\, In \subseteq J\ . \qquad (93)$$

- Using (92), the collecting semantics $\mathrm{post}[\tau^\star[\![C]\!]]\ In$ can be expressed in fixpoint form, as follows:

$$\begin{array}{rcl}
\mathrm{Post}[\![C]\!] & \in & \wp(\Sigma[\![P]\!]) \stackrel{\mathrm{cjm}}{\longmapsto} \wp(\Sigma[\![P]\!]), \\
\mathrm{Post}[\![C]\!]In & \stackrel{\Delta}{=} & \mathrm{post}[\tau^\star[\![C]\!]]\ In \qquad (94) \\
& = & \mathrm{lfp}^{\subseteq}\ \overrightarrow{\mathrm{Post}}[\![C]\!]In, \qquad (95) \\
\text{where} \quad \overrightarrow{\mathrm{Post}}[\![C]\!] & \in & \wp(\Sigma[\![P]\!]) \stackrel{\mathrm{cjm}}{\longmapsto} \left(\wp(\Sigma[\![P]\!]) \stackrel{\mathrm{cjm}}{\longmapsto} \wp(\Sigma[\![P]\!])\right), \\
\overrightarrow{\mathrm{Post}}[\![C]\!]In & \stackrel{\Delta}{=} & \lambda X \cdot In \cup \mathrm{post}[\tau[\![C]\!]]\ X\ .
\end{array}$$

- It follows that we have to effectively compute a machine representable approximation to the least solution of a fixpoint equation.

Abstract Interpretation of Imperative Programs

Fixpoint precise abstraction

If $\langle M, \preceq, 0, \vee\rangle$ is a cpo, the pair $\langle \alpha, \gamma\rangle$ is a Galois connection $\langle M, \preceq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L, \sqsubseteq\rangle$, $\mathcal{F} \in M \stackrel{\mathrm{mon}}{\longmapsto} M$ and $\mathcal{G} \in L \stackrel{\mathrm{mon}}{\longmapsto} L$ are monotonic and

$$\forall x \in M : x \preceq \mathrm{lfp}^{\preceq}\ \mathcal{F} \implies \alpha \circ \mathcal{F}(x) = \mathcal{G} \circ \alpha(x)$$

then

$$\alpha(\mathrm{lfp}^{\preceq}\ \mathcal{F}) = \mathrm{lfp}^{\sqsubseteq}\ \mathcal{G}$$

and the iteration order of $\mathcal{G}$ is less than or equal to that of $\mathcal{F}$.

Fixpoint precise abstraction

If $\langle M, \preceq, 0, \vee\rangle$ is a cpo, the pair $\langle \alpha, \gamma\rangle$ is a Galois connection $\langle M, \preceq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L, \sqsubseteq\rangle$, $\mathcal{F} \in M \stackrel{\mathrm{mon}}{\longmapsto} M$ and $\mathcal{G} \in L \stackrel{\mathrm{mon}}{\longmapsto} L$ are monotonic and

$$\forall y \in L : \gamma(y) \preceq \mathrm{lfp}^{\preceq}\ \mathcal{F} \Rightarrow \alpha \circ \mathcal{F} \circ \gamma(y) \sqsubseteq \mathcal{G}(y)$$

or equivalently $\forall x \in M : \gamma \circ \alpha(x) \preceq \mathrm{lfp}^{\preceq}\ \mathcal{F} \Rightarrow \alpha \circ \mathcal{F}(x) \sqsubseteq \mathcal{G} \circ \alpha(x)$

or equivalently $\forall y \in L : \gamma(y) \preceq \mathrm{lfp}^{\preceq}\ \mathcal{F} \Rightarrow \mathcal{F} \circ \gamma(y) \preceq \gamma \circ \mathcal{G}(y)$

then

$$\mathrm{lfp}^{\preceq}\ \mathcal{F} \preceq \gamma(\mathrm{lfp}^{\sqsubseteq}\ \mathcal{G})$$

and equivalently $\alpha(\mathrm{lfp}^{\preceq}\ \mathcal{F}) \sqsubseteq \mathrm{lfp}^{\sqsubseteq}\ \mathcal{G}$,

Abstract invariants

- Abstract invariants approximate program invariants in the form of abstract environments attached to program points.
- The abstract environments assign an abstract value in some abstract domain L to each program variable whence specify an overapproximation of its possible runtime values when execution reaches that point.
- The abstract domain is therefore ($P \in \mathrm{Prog}$):

$$\begin{array}{lll} \mathrm{AEnv}[\![P]\!] \stackrel{\Delta}{=} \mathrm{Var}[\![P]\!] \mapsto L, & & \text{abstract environments;} \\ \mathrm{ADom}[\![P]\!] \stackrel{\Delta}{=} \mathrm{in}_P[\![P]\!] \mapsto \mathrm{AEnv}[\![P]\!], & & \text{abstract invariants.} \end{array}$$

- The correspondence with program invariants is specified by the Galois connection

$$\langle \wp(\Sigma[\![P]\!]),\ \subseteq \rangle \underset{\ddot{\alpha}[\![P]\!]}{\overset{\ddot{\gamma}[\![P]\!]}{\leftrightarrows}} \langle \mathrm{ADom}[\![P]\!],\ \ddot{\sqsubseteq} \rangle \tag{96}$$

where (see def. (20) of $\dot{\alpha}$ and (21) of $\dot{\gamma}$ where $\mathbb{V}$ is $\mathrm{Var}[\![P]\!]$):

$$\ddot{\alpha}[\![P]\!]I \stackrel{\Delta}{=} \lambda \ell \in \mathrm{in}_P[\![P]\!] \cdot \dot{\alpha}(\{\rho \mid \langle \ell,\ \rho \rangle\} \in I\}), \tag{97}$$

$$\ddot{\gamma}[\![P]\!]J \stackrel{\Delta}{=} \{\langle \ell,\ \rho \rangle \mid \rho \in \dot{\gamma}(J_\ell)\}, \tag{98}$$

$$J \ddot{\sqsubseteq} J' \stackrel{\Delta}{=} \forall \ell \in \mathrm{in}_P[\![P]\!] : \forall \mathtt{X} \in \mathrm{Var}[\![P]\!] : J_\ell(\mathtt{X}) \sqsubseteq J'_\ell(\mathtt{X})\ .$$

- It follows that $\langle \mathrm{ADom}[\![P]\!],\ \ddot{\sqsubseteq},\ \ddot{\bot},\ \ddot{\top},\ \ddot{\sqcup},\ \ddot{\sqcap} \rangle$ is a complete lattice.

Generic implementation of nonrelational abstract invariants

```
module type Abstract_Dom_Algebra_signature =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  functor (E: Abstract_Env_Algebra_signature) ->
  sig
    open Abstract_Syntax
    open Labels
    type aDom        (* complete lattice of abstract invariants *)
    type element = aDom
    val bot  : unit -> aDom           (* infimum                *)
    val join : aDom -> (aDom -> aDom) (* least upper bound      *)
    val leq  : aDom -> (aDom -> bool) (* approximation ordering *)
    (* substitution *)
    val get  : aDom -> label -> E(L).env         (* j(l)        *)
    val set  : aDom -> label -> E(L).env -> aDom (* j[l <- r]   *)
  end;;
```


Abstract predicate transformers

This abstraction is extended to predicate transformers thanks to the functional abstraction

$$\langle \wp(\Sigma[\![P]\!]) \overset{\mathrm{cjm}}{\longmapsto} \wp(\Sigma[\![P]\!]),\ \dot{\subseteq} \rangle \underset{\ddot{\alpha}[\![P]\!]}{\overset{\dot{\gamma}[\![P]\!]}{\leftrightarrows}} \langle \mathrm{ADom}[\![P]\!] \overset{\mathrm{mon}}{\longmapsto} \mathrm{ADom}[\![P]\!],\ \dot{\ddot{\sqsubseteq}} \rangle \tag{99}$$

where

$$\dot{\ddot{\alpha}}[\![P]\!]F \stackrel{\Delta}{=} \ddot{\alpha}[\![P]\!] \circ F \circ \ddot{\gamma}[\![P]\!], \tag{100}$$

$$\dot{\ddot{\gamma}}[\![P]\!]G \stackrel{\Delta}{=} \ddot{\gamma}[\![P]\!] \circ G \circ \ddot{\alpha}[\![P]\!],$$

$$G \dot{\ddot{\sqsubseteq}} G' \stackrel{\Delta}{=} \forall J \in \mathrm{ADom}[\![P]\!] : \forall \ell \in \mathrm{in}_P[\![P]\!] : \forall \mathtt{X} \in \mathrm{Var}[\![P]\!] : G(J)_\ell(\mathtt{X}) \sqsubseteq G'(J)_\ell(\mathtt{X})\ . \tag{101}$$

Generic forward nonrelational abstract interpretation of programs

- The calculational design of the generic nonrelational abstract reachable states semantics of programs ($P \in \mathrm{Prog}$)

$$\mathrm{APost}[\![P]\!] \in \mathrm{ADom}[\![P]\!] \stackrel{\mathrm{mon}}{\longmapsto} \mathrm{ADom}[\![P]\!]$$

  can now be defined as an overapproximation of the forward collecting semantics (30)

$$\dot{\alpha}[\![P]\!](\mathrm{Post}[\![P]\!]) \mathrel{\dot{\sqsubseteq}} \mathrm{APost}[\![P]\!] \ . \tag{102}$$

- Starting from the formal specification $\dot{\alpha}[\![P]\!](\mathrm{Post}[\![P]\!])$, we derive an effectively computable function $\mathrm{APost}[\![P]\!]$ satisfying (102) by calculus.

We proceed by structural induction on the components $\mathrm{Cmp}[\![P]\!]$ of P, proving that for all $C \in \mathrm{Cmp}[\![P]\!]$

- *monotony*

$$\mathrm{APost}[\![C]\!] \in \mathrm{ADom}[\![P]\!] \stackrel{\mathrm{mon}}{\longmapsto} \mathrm{ADom}[\![P]\!] \tag{103}$$

- *soundness*

$$\dot{\alpha}[\![P]\!](\mathrm{Post}[\![C]\!]) \mathrel{\dot{\sqsubseteq}} \mathrm{APost}[\![C]\!], \tag{104}$$

- *locality*

$$\forall J \in \mathrm{ADom}[\![P]\!] : \forall \ell \in \mathrm{in}[\![P]\!] - \mathrm{in}_P[\![C]\!] : \\ J_\ell = (\mathrm{APost}[\![C]\!]J)_\ell, \tag{105}$$

- *dependence*

$$\forall J, J' \in \mathrm{ADom}[\![P]\!] : (\forall \ell \in \mathrm{in}_P[\![C]\!] : J_\ell = J'_\ell) \implies \\ (\forall \ell \in \mathrm{in}_P[\![C]\!] : (\mathrm{APost}[\![C]\!]J)_\ell = (\mathrm{APost}[\![C]\!]J')_\ell) \ . \tag{106}$$

- Intuitively the locality and dependence properties express that the postcondition of a command can only depend upon and affect the abstract local invariants attached to labels in the command.

Generic forward nonrelational reachability abstract semantics of programs

- $C = \mathtt{skip}$

$$\mathrm{APost}[\![\mathtt{skip}]\!] = \lambda J \cdot J[\mathrm{after}_P[\![\mathtt{skip}]\!] \leftarrow J_{\mathrm{after}_P[\![\mathtt{skip}]\!]} \dot{\sqcup} J_{\mathrm{at}_P[\![\mathtt{skip}]\!]}] \tag{107}$$

- $C = \mathtt{X} := A$

$$\mathrm{APost}[\![\mathtt{X} := A]\!] = \lambda J \cdot \mathrm{let}\ \ell = \mathrm{at}_P[\![\mathtt{X} := A]\!], \ell' = \mathrm{after}_P[\![\mathtt{X} := A]\!]\ \mathrm{in} \\ \mathrm{let}\ v = \mathrm{Faexp}^\sharp[\![A]\!](J_\ell)\ \mathrm{in} \\ (\mathfrak{V}(v)\ ?\ J\ ¿\ J[\ell' \leftarrow J_{\ell'} \dot{\sqcup} J_\ell[\mathtt{X} \leftarrow v \sqcap ?^\sharp]]) \tag{108}$$

  where:

$$\forall v \in L : \mathfrak{V}(v) \implies \gamma(v) \subseteq \mathbb{E}$$

- $C = \texttt{if}\ B\ \texttt{then}\ S_t\ \texttt{else}\ S_f\ \texttt{fi}$

$$\begin{array}{l}\mathrm{APost}[\![C]\!] = \\ \quad \lambda J\cdot \mathrm{let}\ J' = J[\mathrm{at}_P[\![S_t]\!] \leftarrow J_{\mathrm{at}_P[\![S_t]\!]} \dot\sqcup \mathrm{Abexp}[\![B]\!](J_{\mathrm{at}_P[\![C]\!]});\\ \qquad\qquad \mathrm{at}_P[\![S_f]\!] \leftarrow J_{\mathrm{at}_P[\![S_f]\!]} \dot\sqcup \mathrm{Abexp}[\![T(\neg B)]\!](J_{\mathrm{at}_P[\![C]\!]})]\ \mathrm{in}\\ \qquad \mathrm{let}\ J'' = \mathrm{APost}[\![S_t]\!] \circ \mathrm{APost}[\![S_f]\!](J')\ \mathrm{in}\\ \qquad\quad J''[\mathrm{after}_P[\![C]\!] \leftarrow J''_{\mathrm{after}_P[\![C]\!]} \dot\sqcup J''_{\mathrm{after}_P[\![S_t]\!]} \dot\sqcup J''_{\mathrm{after}_P[\![S_f]\!]}]\end{array} \tag{109}$$

- $C = \texttt{while}\ B\ \texttt{do}\ S\ \texttt{od}$

$$\begin{array}{l}\mathrm{APost}[\![C]\!] = \\ \quad \left(1_{\mathrm{ADom}[\![P]\!]^{\mathrm{mon}}_{\longmapsto}\mathrm{ADom}[\![P]\!]} \mathbin{\ddot\sqcup} (\mathrm{APost}[\![S]\!] \circ \mathrm{APost}^B[\![C]\!]) \mathbin{\ddot\sqcup} \mathrm{APost}^{\bar B}[\![C]\!]\right) \circ\\ \quad \left(\lambda J\cdot \mathrm{lfp}^{\ddot\sqsubseteq}\, \lambda X\cdot J \mathbin{\ddot\sqcup} \mathrm{APost}^R[\![C]\!] \circ \mathrm{APost}[\![S]\!] \circ \mathrm{APost}^B[\![C]\!](X)\right) \circ\\ \quad \left(1_{\mathrm{ADom}[\![P]\!]^{\mathrm{mon}}_{\longmapsto}\mathrm{ADom}[\![P]\!]} \mathbin{\ddot\sqcup} (\mathrm{APost}[\![S]\!] \circ \mathrm{APost}^R[\![C]\!])\right)\end{array} \tag{110}$$

where:

$$\begin{array}{l}\mathrm{APost}^B[\![C]\!] \stackrel{\Delta}{=} \lambda J\cdot \ddot\bot[\mathrm{at}_P[\![S]\!] \leftarrow \mathrm{Abexp}[\![B]\!] J_{\mathrm{at}_P[\![C]\!]}]\\ \mathrm{APost}^{\bar B}[\![C]\!] \stackrel{\Delta}{=} \lambda J\cdot \ddot\bot[\mathrm{after}_P[\![C]\!] \leftarrow \mathrm{Abexp}[\![T(\neg B)]\!] J_{\mathrm{at}_P[\![C]\!]}]\\ \mathrm{APost}^R[\![C]\!] \stackrel{\Delta}{=} \lambda J\cdot \ddot\bot[\mathrm{at}_P[\![C]\!] \leftarrow J_{\mathrm{after}_P[\![S]\!]}]\end{array}$$

- $$\mathrm{APost}[\![C_1\ ;\ \ldots\ ;\ C_n]\!] = \mathrm{APost}[\![C_n]\!] \circ \ldots \circ \mathrm{APost}[\![C_1]\!] \tag{111}$$
- $$\mathrm{APost}[\![S\ ;\ ;]\!] = \mathrm{APost}[\![S]\!]\ . \tag{112}$$

Descendants of initial states

- To effectively compute an overapproximation of the set $\mathrm{post}[\tau^\star[\![P]\!]]\ In$ of states which are reachable by repeated small steps of the program P from some given set In of initial states, we use an overapproximation of the initial states

$$\ddot\alpha[\![P]\!](In) \mathbin{\ddot\sqsubseteq} I \tag{113}$$

and compute $\mathrm{APost}[\![P]\!]I$.

- Elements of $\mathrm{ADom}[\![P]\!]$ must be machine representable;
- This is the case if the lattice L of abstract value properties is itself machine representable;
- The computation of $\mathrm{APost}[\![P]\!]I$ terminates if the complete lattice $\langle L, \sqsubseteq\rangle$ satisfies the ascending chain condition;
- Otherwise convergence must be accelerated using widening/narrowing techniques [3, 7].

Soundness

- The soundness of the approach is easily established

$$
\begin{array}{ll}
 & \mathrm{APost}[\![P]\!]I \\
\ddot{\sqsubseteq} & \quad \text{\{soundness (102)\}} \\
 & \dot{\ddot{\alpha}}[\![P]\!](\mathrm{Post}[\![P]\!])I \\
\ddot{\sqsubseteq} & \quad \text{\{abstraction (113) of the entry condition and monotony\}} \\
 & \dot{\ddot{\alpha}}[\![P]\!](\mathrm{Post}[\![P]\!])\,\ddot{\alpha}[\![P]\!](In) \\
\ddot{\sqsubseteq} & \quad \text{\{def. (100)\}} \\
 & \ddot{\alpha}[\![P]\!] \circ \mathrm{Post}[\![P]\!] \circ \ddot{\gamma}[\![P]\!] \circ \ddot{\alpha}[\![P]\!](In) \\
\ddot{\sqsubseteq} & \quad \text{\{Galois connection (96) so that } \ddot{\gamma}[\![P]\!] \circ \ddot{\alpha}[\![P]\!] \text{ is extensive and monotony\}} \\
 & \ddot{\alpha}[\![P]\!](\mathrm{Post}[\![P]\!](In))
\end{array}
$$

...

or equivalently, by the Galois connection (96)

$$\mathrm{post}[\tau^\star[\![P]\!]]\,In \subseteq \ddot{\gamma}[\![P]\!](\mathrm{APost}[\![P]\!]I)\ . \tag{114}$$

- Notice that the set $\ddot{\gamma}[\![P]\!](\mathrm{APost}[\![P]\!]I)$ is usually infinite so that its exploitation must be programmed using the encoding used for $\mathrm{ADom}[\![P]\!]$ (or some machine representable image).

The generic abstract interpreter for reachability analysis

Abstract syntax of commands

```
type com =
  | SKIP of label * label
  | ASSIGN of label * variable * aexp * label
  | SEQ of label * (com list) * label
  | IF of label * bexp * bexp * com * com * label
  | WHILE of label * bexp * bexp * com * label
```

- For a command C, the first label $\mathrm{at}_P[\![C]\!]$ (written `(at` C`)`) and the second $\mathrm{after}_P[\![C]\!]$ (written `(after` C`)`) satisfy the labelling conditions.
- The boolean expression B of conditional and iteration commands is recorded by $T(B)$ and $T(\neg(B))$.

Signature of the generic abstract interpreter

```
module type APost_signature =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  functor (E: Abstract_Env_Algebra_signature) ->
  functor (D: Abstract_Dom_Algebra_signature) ->
  functor (Faexp: Faexp_signature) ->
  functor (Baexp: Baexp_signature) ->
  functor (Abexp: Abexp_signature) ->
  sig
    open Abstract_Syntax
    (* generic forward nonrelational abstract reachability semantics of *)
 (* commands                                                            *)
    val aPost  : com -> D(L)(E).aDom -> D(L)(E).aDom
  end;;
```



```
module APost_implementation =
 functor (L: Abstract_Lattice_Algebra_signature) ->
 functor (E: Abstract_Env_Algebra_signature) ->
 functor (D: Abstract_Dom_Algebra_signature) ->
 functor (Faexp: Faexp_signature) ->
 functor (Baexp: Baexp_signature) ->
 functor (Abexp: Abexp_signature) ->
 struct
  open Abstract_Syntax
  open Labels
  (* generic abstract environments                                    *)
  module E' = E(L)
  (* generic abstract invariants                                      *)
  module D' = D(L)(E)
  (* generic forward abstract interpretation of arithmetic operations  *)
  module Faexp' = Faexp(L)(E)
  (* generic [reductive] abstract interpretation of boolean operations *)
  module Abexp' = Abexp(L)(E)(Faexp)(Baexp)
  (* iterative fixpoint computation *)
  module F = Fixpoint((D':Poset_signature with type element=D(L)(E).aDom))
```



```
  (* generic forward nonrelational abstract reachability semantics     *)
  exception Error_aPost of string
  let rec aPost c j = match c with
  | (SKIP (l, l')) -> (D'.set j l' (E'.join (D'.get j l') (D'.get j l)))
  | (ASSIGN (l,x,a,l')) ->
     let v = (Faexp'.faexp a (D'.get j l))  in
      if (L.in_errors v) then j
      else (D'.set j l' (E'.join (D'.get j l') (E'.set (D'.get j l) x
                                                (L.meet v (L.f_RANDOM ()))))))
  | (SEQ (l, s, l')) -> (aPostseq s j)
  | (IF (l, b, nb, t, f, l')) ->
     let j' = (D'.set j (at t) (E'.join (D'.get j (at t))
                                        (Abexp'.abexp b  (D'.get j l)))) in
     let j'' = (D'.set j' (at f) (E'.join (D'.get j'
                           (at f)) (Abexp'.abexp nb (D'.get j' l)))) in
     let j''' = (aPost t (aPost f j'')) in
       (D'.set j''' l' (E'.join (E'.join (D'.get j''' l')
     (D'.get j''' (after t))) (D'.get j''' (after f))))
```



```
  | (WHILE (l, b, nb, c', l')) ->
     let aPostB j = (D'.set (D'.bot ()) (at c')
                                  (Abexp'.abexp b (D'.get j l))) in
     let aPostnotB j = (D'.set (D'.bot ()) l'
                              (Abexp'.abexp nb (D'.get j l))) in
     let aPostR j = (D'.set (D'.bot ()) l (D'.get j (after c'))) in
     let j' = (D'.join j (aPost c' (aPostR j))) in
     let f x = (D'.join j' (aPostR (aPost c' (aPostB x)))) in
     let j'' = (F.lfp f (D'.bot ())) in
       (D'.join j'' (D'.join (aPost c' (aPostB j'')) (aPostnotB j'')))
  and aPostseq s j = match s with
  | []   -> raise (Error_aPost "empty sequence of commands")
  | [c]  -> (aPost c j)
  | h::t -> (aPostseq t (aPost h j))
 end;;

module APost = (APost_implementation:APost_signature);;
```

Abstract initial states

For short, we consider the simple case when In is the set $\mathrm{Entry}[\![P]\!]$ of program entry states:

$$
\begin{array}{ll}
 & \ddot{\alpha}[\![P]\!](\mathrm{Entry}[\![P]\!]) \\
= & \quad \wr \text{def. (97) of } \ddot{\alpha}[\![P]\!] \text{ and (80) of } \mathrm{Entry}[\![P]\!] \int \\
 & \lambda \ell \in \mathrm{in}_P[\![P]\!] \cdot \dot{\alpha}(\{\lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \Omega_{\mathtt{i}} \mid \ell = \mathrm{at}_P[\![P]\!]\}) \\
= & \quad \wr \text{def. (20) of } \dot{\alpha} \int \\
 & \lambda \ell \in \mathrm{in}_P[\![P]\!] \cdot (\ell = \mathrm{at}_P[\![P]\!] \ ? \ \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \alpha(\{\Omega_{\mathtt{i}}\}) \ ¿ \ \dot{\alpha}(\emptyset)) \\
= & \quad \wr \text{def. } \dot{\perp} \stackrel{\Delta}{=} \dot{\alpha}(\emptyset),\ \ddot{\perp} \stackrel{\Delta}{=} \lambda \ell \in \mathrm{in}_P[\![P]\!] \cdot \dot{\perp} \text{ and (18) of substitution} \int \\
 & \ddot{\perp}[\mathrm{at}_P[\![P]\!] \leftarrow \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \alpha(\{\Omega_{\mathtt{i}}\})] \\
= & \quad \wr \text{by defining } \mathrm{AEntry}[\![P]\!] \stackrel{\Delta}{=} \ddot{\perp}[\mathrm{at}_P[\![P]\!] \leftarrow \lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \alpha(\{\Omega_{\mathtt{i}}\})] \int \\
 & \mathrm{AEntry}[\![P]\!] \ .
\end{array}
$$

Implementation of the abstract entry states

```
module AEntry_implementation =
  functor (L: Abstract_Lattice_Algebra_signature) ->
  functor (E: Abstract_Env_Algebra_signature) ->
  functor (D: Abstract_Dom_Algebra_signature) ->
  struct
    open Abstract_Syntax
    open Labels
    (* generic abstract environments *)
    module E' = E(L)
    (* generic abstract invariants *)
    module D' = D(L)(E)
    (* abstraction of entry states *)
    exception Error_aEntry of string
    let aEntry c =
      if (at c) <> (entry ()) then
        raise (Error_aEntry "not the program entry point")
      else
        (D'.set (D'.bot ()) (at c) (E'.initerr ()))
  end;;
```


Instantiation for entry states

The generic abstract interpreter $\mathrm{APost}[\![P]\!](\mathrm{AEntry}[\![P]\!])$ can be partially instantiated with (or without) reductive iterations, as follows:

```
module  Analysis_Reductive_Iteration_implementation =
 functor (L: Abstract_Lattice_Algebra_signature) ->
struct
 open Program_To_Abstract_Syntax
 module ENTRY = AEntry(L)(Abstract_Env_Algebra)(Abstract_Dom_Algebra)
 module POST = APost(L)(Abstract_Env_Algebra)(Abstract_Dom_Algebra)(Faexp)
                    (Baexp_Reductive_Iteration)(Abexp_Reductive_Iteration)
 module PRN = Pretty_Print(L)(Abstract_Env_Algebra)(Abstract_Dom_Algebra)
 let analysis () =                                                    (115)
   print_string "type the  program to analyze..."; print_newline ();
   let p = abstract_syntax_of_program () in
     let j = (POST.aPost p (ENTRY.aEntry p)) in
    (PRN.pretty_print p j)
end;;
```


Instantiation for a particular value property abstract domain

```
module ISS' = Analysis_Reductive_Iteration(ISS_Lattice_Algebra);;
```

Examples

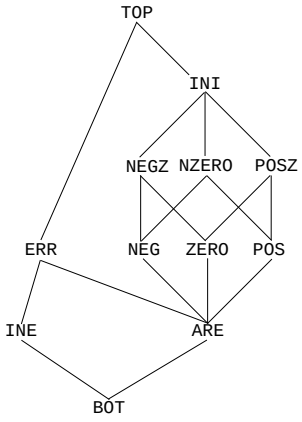
```
{ n:ERR; i:ERR }
  n := ?; i := 1;
{ n:INI; i:POS }
  while (i < n) do
    { n:POS; i:POS }
      i := (i + 1)
    { n:POS; i:POS }
  od
{ n:INI; i:POS }
```

```
{ n:ERR; i:ERR }
  n := ?; i := 0;
{ n:INI; i:INI }
  while (i < n) do
    { n:INI; i:INI }
      i := (i + 1)
    { n:INI; i:INI }
  od
{ n:INI; i:INI }
```

```
{ x:ERR }
  x := (1 / 0);
{ x:BOT }
  skip;
{ x:BOT }
  x := 1
{ x:POS }
```

The comparison of the first and second examples illustrates the loss of information due to the absence of an abstract value POSZ such that $\gamma(\texttt{POSZ}) \stackrel{\Delta}{=} [0, \texttt{max\_int}] \cup \{\Omega_\texttt{a}\}$. The third example shows the imprecision on reachability resulting from the choice to have $\gamma(\texttt{BOT}) \neq \emptyset$).

The lattice of errors and signs



$$
\begin{aligned}
\gamma(\texttt{BOT}) &\stackrel{\Delta}{=} \emptyset \\
\gamma(\texttt{INE}) &\stackrel{\Delta}{=} \{\Omega_\texttt{i}\} \\
\gamma(\texttt{ARE}) &\stackrel{\Delta}{=} \{\Omega_\texttt{a}\} \\
\gamma(\texttt{ERR}) &\stackrel{\Delta}{=} \{\Omega_\texttt{i}, \Omega_\texttt{a}\} \\
\gamma(\texttt{NEG}) &\stackrel{\Delta}{=} [\texttt{min\_int}, -1] \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{ZERO}) &\stackrel{\Delta}{=} \{0, \Omega_\texttt{a}\} \\
\gamma(\texttt{POS}) &\stackrel{\Delta}{=} [1, \texttt{max\_int}] \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{NEGZ}) &\stackrel{\Delta}{=} [\texttt{min\_int}, 0] \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{NZERO}) &\stackrel{\Delta}{=} [\texttt{min\_int}, -1] \cup [1, \texttt{max\_int}] \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{POSZ}) &\stackrel{\Delta}{=} [0, \texttt{max\_int}] \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{INI}) &\stackrel{\Delta}{=} \mathbb{I} \cup \{\Omega_\texttt{a}\} \\
\gamma(\texttt{TOP}) &\stackrel{\Delta}{=} \mathbb{I}_\Omega = \mathbb{I} \cup \{\Omega_\texttt{i}, \Omega_\texttt{a}\}
\end{aligned}
$$

Examples (lattice of errors and signs)

```
{ n:ERR; i:ERR }
  n := ?; i := 0;
{ n:INI; i:POSZ }
  while (i < n) do
    { n:POS; i:POSZ }
      i := (i + 1)
    { n:POS; i:POS }
  od
{ n:INI; i:POSZ }
```

```
{ x:ERR }
  x := (1 / 0);
{ x:BOT }
  skip;
{ x:BOT }
  x := 1
{ x:BOT }
```


Comparison with the handling of arithmetic or boolean expressions using assignments of simple monomials to auxiliary variables

```
{ x:ERR; y:ERR }
  x := 0; y := ?;
{ x:ZERO; y:INI }
  while (x = -y) do
    { x:ZERO; y:ZERO }
      skip
    { x:ZERO; y:ZERO }
  od
{ x:ZERO; y:INI }
```

```
{ x:ERR; y:ERR; i1:ERR }
  x := 0; y := ?; i1 := -y;
{ x:ZERO; y:INI; i1:INI }
  while (x = i1) do
    { x:ZERO; y:INI; i1:ZERO }
      skip; i1 := -y
    { x:ZERO; y:INI; i1:INI }
  od
{ x:ZERO; y:INI; i1:INI }
```

Comparison with the compilation of boolean expressions into short-circuit conditional code

```
{ x:ERR; y:ERR; z:ERR }
  x := 0; y := ?; z := ?;
{ x:ZERO; y:INI; z:INI }
  if ((x=y)&((z+1)=x)&(y=z)) then
    { x:BOT; y:BOT; z:BOT }
      skip
  else
    { x:ZERO; y:INI; z:INI }
      skip
  fi
{ x:ZERO; y:INI; z:INI }
```

```
{ x:ERR; y:ERR; z:ERR }
  x := 0; y := ?; z := ?;
{ x:ZERO; y:INI; z:INI }
  if ((x=y)&((z+1)=x)) then
    { x:ZERO; y:ZERO; z:NEG }
      if (y=z) then
        { x:ZERO; y:BOT; z:BOT }
          skip
      else
        { x:ZERO; y:ZERO; z:NEG }
          skip
      fi
    { x:ZERO; y:ZERO; z:NEG }
  else
    { x:ZERO; y:INI; z:INI }
      skip
  fi
{ x:ZERO; y:INI; z:INI }
```


Reachability analysis from the entry states

Specializing the abstract interpreter to reachability analysis from the entry states

We want to calculate:

$$\mathrm{APost}[\![P]\!](\mathrm{AEntry}[\![P]\!])$$

and more generally, for all program subcommands $C \in \mathrm{Cmp}[\![P]\!]$:

$$\lambda r \in \mathrm{AEnv}[\![P]\!] \cdot \mathrm{APost}[\![C]\!](\ddot{\bot}[\mathrm{at}_P[\![C]\!] \leftarrow r])$$

that is:

$$\mathrm{APostEn}_P[\![C]\!] \stackrel{\Delta}{=} \alpha^{\varepsilon}_P[\![C]\!](\mathrm{APost}[\![C]\!]) \tag{116}$$

...

where

$$\alpha^{\varepsilon}_P[\![C]\!] \stackrel{\Delta}{=} \lambda F \cdot \lambda r \cdot F(\ddot{\bot}[\mathrm{at}_P[\![C]\!] \leftarrow r]) \tag{117}$$

$$\gamma^{\varepsilon}_P[\![C]\!] \stackrel{\Delta}{=} \lambda f \cdot \lambda J \cdot (\forall l \neq \mathrm{at}_P[\![C]\!] : J_l = \dot{\bot} \ ? \ f(J_{\mathrm{at}_P[\![C]\!]}) \ ¿ \ \ddot{\top}) \tag{118}$$

is such that

$$\langle \mathrm{ADom}[\![P]\!] \stackrel{\mathrm{mon}}{\longmapsto} \mathrm{ADom}[\![P]\!], \dot{\ddot{\sqsubseteq}} \rangle \underset{\alpha^{\varepsilon}_P[\![C]\!]}{\overset{\gamma^{\varepsilon}_P[\![C]\!]}{\leftrightarrows}} \langle \mathrm{AEnv}[\![P]\!] \stackrel{\mathrm{mon}}{\longmapsto} \mathrm{ADom}[\![P]\!], \dot{\ddot{\sqsubseteq}} \rangle \ .$$

- We consider the simple situation where $\gamma(\bot) = \emptyset$ for which (36), (43) and (58) do hold;
- It follows by structural and fixpoint induction that for all $C \in \mathrm{Cmp}[\![P]\!]$

$$\mathrm{APost}[\![C]\!](\ddot{\bot}) = \ddot{\bot} \ . \tag{119}$$

- We calculate $\mathrm{APostEn}_P[\![C]\!]$ by structural induction and trivially prove simultaneously:

locality: $\forall r \in \mathrm{AEnv}[\![P]\!] : \forall l \in \mathrm{in}[\![P]\!] - \mathrm{in}_P[\![C]\!](\mathrm{APostEn}_P[\![C]\!]r)_l = \dot{\bot}$ (120)

extension: $\forall r \in \mathrm{AEnv}[\![P]\!] : r \sqsubseteq (\mathrm{APostEn}_P[\![C]\!]r)_{\mathrm{at}_P[\![C]\!]}$ (121)

Generic forward nonrelational reachability from entry states abstract semantics of programs

- $C =$ `skip`:

$$\mathrm{APostEn}_P[\![\texttt{skip}]\!]r = \ddot{\bot}[\mathrm{at}_P[\![\texttt{skip}]\!] \leftarrow r; \mathrm{after}_P[\![\texttt{skip}]\!] \leftarrow r]$$

- `X` $:=$ A:

$$\begin{array}{l}\mathrm{APostEn}_P[\![\texttt{X} := A]\!]r = \mathrm{let}\ v = \mathrm{Faexp}^{\flat}[\![A]\!]r\ \mathrm{in} \\ \quad (\mho(v)\ ?\ \ddot{\bot}[\mathrm{at}_P[\![\texttt{X} := A]\!] \leftarrow r] \\ \quad\quad ¿\ \ddot{\bot}[\mathrm{at}_P[\![\texttt{X} := A]\!] \leftarrow r; \mathrm{after}_P[\![\texttt{X} := A]\!] \leftarrow r[\texttt{X} \leftarrow v \sqcap ?^{\flat}]])\end{array}$$

- $C =$ `if` B `then` S_t `else` S_f `fi`:

$$\begin{array}{l}\mathrm{APostEn}_P[\![C]\!]r = \\ \quad \mathrm{let}\ J^{\mathrm{tt}} = \mathrm{APostEn}_P[\![S_t]\!](\mathrm{Abexp}[\![B]\!]r)\ \mathrm{in} \\ \quad\quad \mathrm{let}\ J^{\mathrm{ff}} = \mathrm{APostEn}_P[\![S_f]\!](\mathrm{Abexp}[\![T(\neg B)]\!]r)\ \mathrm{in} \\ \quad\quad\quad \ddot{\bot}[\mathrm{at}_P[\![C]\!] \leftarrow r; \mathrm{after}_P[\![C]\!] \leftarrow J^{\mathrm{tt}}_{\mathrm{after}_P[\![S_t]\!]} \dot{\cup} J^{\mathrm{ff}}_{\mathrm{after}_P[\![S_f]\!]}] \ddot{\cup} J^{\mathrm{tt}} \ddot{\cup} J^{\mathrm{ff}}\end{array} \tag{122}$$

- $C =$ `while` B `do` S `od`:

$$\begin{array}{l}\mathrm{APostEn}_P[\![C]\!]r = \\ \quad \mathrm{let}\ r' = \mathrm{lfp}^{\dot{\sqsubseteq}}\ \lambda x \cdot r \dot{\cup} (\mathrm{APostEn}_P[\![S]\!](\mathrm{Abexp}[\![B]\!]x))_{\mathrm{after}_P[\![S]\!]}\ \mathrm{in} \\ \quad\quad \ddot{\bot}[\mathrm{at}_P[\![C]\!] \leftarrow r'; \mathrm{after}_P[\![C]\!] \leftarrow \mathrm{Abexp}[\![T(\neg B)]\!]r'] \ddot{\cup} \\ \quad\quad\quad \mathrm{APostEn}_P[\![S]\!](\mathrm{Abexp}[\![B]\!]r')\end{array}$$

- $C = C_1\ ;\ \ldots\ ;\ C_n$:

$$\begin{array}{l}\mathrm{APostEn}_P[\![C]\!]r = \\ \quad \mathrm{let}\ J^1 = \mathrm{APostEn}_P[\![C_1]\!]r\ \mathrm{in} \\ \quad\quad \mathrm{let}\ J^2 = J^1 \ddot{\cup} \mathrm{APostEn}_P[\![C_2]\!](J^1)_{\mathrm{at}_P[\![C_2]\!]}\ \mathrm{in} \\ \quad\quad\quad \ldots \\ \quad\quad\quad\quad \mathrm{let}\ J^n = J^{n-1} \ddot{\cup} \mathrm{APostEn}_P[\![C_n]\!](J^{n-1})_{\mathrm{at}_P[\![C_n]\!]}\ \mathrm{in} \\ \quad\quad\quad\quad\quad J^n\end{array}$$

- $C = S\ ;;$:

$$\mathrm{APostEn}_P[\![S\ ;;]\!] = \mathrm{APostEn}_P[\![S]\!](\lambda \mathtt{X} \in \mathrm{Var}[\![P]\!] \cdot \alpha(\{\Omega_{\mathtt{i}}\}))\ .$$

Conclusion

Conclusion

- Calculational design of program static analyzers by abstract interpretation of a formal semantics;
- Scales up by small parts;
- This provides a thorough understanding of the abstraction process allowing for the later development of useful large scale analyzers [10].

References

[1] Patrick Cousot.
The Calculational Design of a Generic Abstract Interpreter. In *Calculational System Design*, M. Broy & R. Steinbrüggen, editors, NATO ASI Series F. IOS Press, Amsterdam, 1999. http://www.dmi.ens.fr/~cousot/COUSOTpapers/Marktoberdorf98.shtml, novembre 1998. 2

[2] Patrick Cousot.
The Marktoberdorf'98 generic abstract interpreter. http://www.dmi.ens.fr/~cousot/Marktoberdorf98.shtml, novembre 1998. 2

Bibliography

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In 4^{th} *POPL*, pp. 238–252, Los Angeles, Calif., 1977. ACM Press. 172, 172

[4] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In 6^{th} *POPL*, pp. 269–282, San Antonio, Texas, 1979. ACM Press. 26, 37, 26, 37

[5] P. Cousot. Semantic foundations of program analysis. In S.S. Muchnick and N.D. Jones, eds., *Program Flow Analysis: Theory and Applications*, ch. 10, pp. 303–342. Prentice-Hall, 1981.

[6] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *J. Logic Prog.*, 13(2–3):103–179, 1992. (The editor of JLP has mistakenly published the unreadable galley proof. For a correct version of this paper, see http://www.dmi.ens.fr/~cousot.). 26, 26

[7] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proc. Int. Work. PLILP '92,*, Leuven, Belgium, LNCS 631, pp. 269–295. Springer, 13–17 Aug. 1992. 172, 172

[8] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer, 1990. 33, 33

[9] P. Granger. Improving the results of static analyses of programs by local decreasing iterations. In *Proc. 12th FST & TCS*, pp. 68–79, New Delhi, India, LNCS 652. Springer, 18–20 Dec. 1992.

[10] P. Lacan, J.N. Monfort, Le Vinh Quy Ribal, A. Deutsch, and G. Gonthier. The software reliability verification process: The ARIANE 5 example. In *Proc. DASIA 98 – DAta Systems IN Aerospace*, Athens, Grece. ESA Publications, SP-422, May 25–28 1998. 198, 198

[11] G.D. Plotkin. A structural approach to operational semantics. Tech. rep. DAIMI FN-19, Aarhus University, Denmark, Sep. 1981. 134, 134