« Iterated Forward/Backward Relational Infinitary Static Analysis »

Patrick Cousot

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

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

Course 16.399: "Abstract interpretation"

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

THESE

présentée à

Université Scientifique et Médicale de Grenoble
Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR ES SCIENCES MATHEMATIQUES

par

Patrick COUSOT

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

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

Président : L. BOLLIET

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

Recalls on operational and collecting semantics

Programs states (recall from lecture 5)

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:

$$\Sigma \in \mathrm{Prog} \mapsto \wp(\mathbb{V} \times \mathbb{R}),$$
$$\Sigma[\![P]\!] \stackrel{\mathrm{def}}{=} \mathrm{in}_P[\![P]\!] \times \mathrm{Env}[\![P]\!] . \qquad (1)$$

Program transition relation (recall from lecture 5)

The small-step operational semantics 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 [1]

$$\langle \ell,\, \rho\rangle \models\!\!\llbracket C \rrbracket\!\!\Longrightarrow \langle \ell',\, \rho'\rangle\ .$$

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

Transition system of a program (recall from lecture 5)

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

$$\langle \Sigma\llbracket P\rrbracket,\, \tau\llbracket P\rrbracket\rangle$$

where $\Sigma\llbracket P\rrbracket$ is the set (1) of program states and $\tau\llbracket C\rrbracket$, $C \in \mathrm{Cmp}\llbracket P\rrbracket$ is the transition relation for component C of program P, defined by

$$\tau\llbracket C\rrbracket \stackrel{\mathrm{def}}{=} \{\langle\langle \ell,\, \rho\rangle,\, \langle \ell',\, \rho'\rangle\rangle \mid \langle \ell,\, \rho\rangle \models\!\!\llbracket C \rrbracket\!\!\Longrightarrow \langle \ell',\, \rho'\rangle\} \quad (2)$$

Initial States (recall from lecture 5)

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

$$\mathrm{Entry}\llbracket P\rrbracket \stackrel{\mathrm{def}}{=} \{\langle \mathrm{at}_P\llbracket P\rrbracket,\, \lambda \mathtt{X} \in \mathrm{Var}\llbracket P\rrbracket \cdot \Omega_{\mathrm{i}}\rangle\}\ . \quad (3)$$

Final States (recall from lecture 5)

Execution ends without error when control reaches the program exit point

$$\mathrm{Exit}\llbracket P\rrbracket \stackrel{\mathrm{def}}{=} \{\mathrm{after}_P\llbracket P\rrbracket\} \times \mathrm{Env}\llbracket P\rrbracket\ .$$

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.

Big-step operational semantics of a program (recall from lecture 5)

- The big-step operational semantics of a program P is

$$\langle \Sigma[\![P]\!],\ t^*[\![P]\!] \rangle$$

  where $t^*[\![P]\!] \stackrel{\text{def}}{=} (t[\![P]\!])^*$ is the reflexive transitive closure of the transition relation $t[\![P]\!]$
- Infinite executions are not considered with this semantics

Structural big-step operational semantics (recall from lecture 8)

$$\tau^\star[\![\texttt{skip}]\!] = 1_{\Sigma[\![P]\!]} \cup \tau[\![\texttt{skip}]\!] \tag{4}$$

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

$$\begin{aligned}\tau^\star[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!] = {} & (1_{\Sigma[\![P]\!]} \cup \tau^B) \circ \tau^\star[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t) \ \cup \\ & (1_{\Sigma[\![P]\!]} \cup \tau^{\bar{B}}) \circ \tau^\star[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^f)\end{aligned} \tag{5}$$

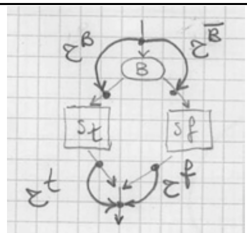
where:

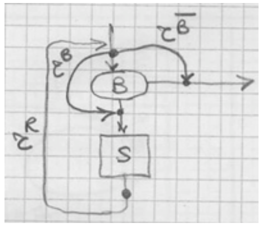
$$\tau^B \stackrel{\text{def}}{=} \{\langle\langle \mathsf{at}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle,\ \langle \mathsf{at}_P[\![S_t]\!],\ \rho\rangle\rangle \mid \rho \vdash B \Rightarrow \mathtt{tt}\}$$

$$\tau^{\bar{B}} \stackrel{\text{def}}{=} \{\langle\langle \mathsf{at}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle,\ \langle \mathsf{at}_P[\![S_f]\!],\ \rho\rangle\rangle \mid \rho \vdash T(\neg B) \Rightarrow \mathtt{tt}\}$$

$$\tau^t \stackrel{\text{def}}{=} \{\langle\langle \mathsf{after}_P[\![S_t]\!],\ \rho\rangle,\ \langle \mathsf{after}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle\rangle \mid \rho \in \mathrm{Env}[\![P]\!]\}$$

$$\tau^f \stackrel{\text{def}}{=} \{\langle\langle \mathsf{after}_P[\![S_f]\!],\ \rho\rangle,\ \langle \mathsf{after}_P[\![\texttt{if } B \texttt{ then } S_t \texttt{ else } S_f \texttt{ fi}]\!],\ \rho\rangle\rangle \mid \rho \in \mathrm{Env}[\![P]\!]\}$$





$$\begin{aligned}\tau^\star[\![\texttt{while } B \texttt{ do } S \texttt{ od}]\!] = {} & ((1_{\Sigma[\![P]\!]} \cup \tau^\star[\![S]\!] \circ \tau^R) \circ (\tau^B \circ \tau^\star[\![S]\!] \circ \tau^R)^\star \circ \\ & (1_{\Sigma[\![P]\!]} \cup \tau^B \circ \tau^\star[\![S]\!] \cup \tau^{\bar{B}})) \cup \tau[\![S]\!]^\star\end{aligned} \tag{6}$$

where:

$$\tau^B \stackrel{\text{def}}{=} \{\langle\langle \mathsf{at}_P[\![\texttt{while } B \texttt{ do } S \texttt{ od}]\!],\ \rho\rangle,\ \langle \mathsf{at}_P[\![S]\!],\ \rho\rangle\rangle \mid \rho \vdash B \Rightarrow \mathtt{tt}\}$$

$$\tau^{\bar{B}} \stackrel{\text{def}}{=} \{\langle\langle \mathsf{at}_P[\![\texttt{while } B \texttt{ do } S \texttt{ od}]\!],\ \rho\rangle,\ \langle \mathsf{after}_P[\![\texttt{while } B \texttt{ do } S \texttt{ od}]\!],\ \rho\rangle\rangle \mid \rho \vdash T(\neg B) \Rightarrow \mathtt{tt}\}$$

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

Definition of the forward collecting semantics of boolean expressions (recall from lecture 8)

Recall the *collecting semantics* $\mathrm{Cbexp}[\![B]\!]R$ of a boolean expression B from course 8:

$$\mathrm{Cbexp} \in \mathrm{Bexp} \mapsto \wp(\mathrm{Env}[\![P]\!]) \stackrel{\sqcup}{\longmapsto} \wp(\mathrm{Env}[\![P]\!]),$$
$$\mathrm{Cbexp}[\![B]\!]R \stackrel{\mathrm{def}}{=} \{\rho \in R \mid \rho \vdash B \Mapsto \mathtt{tt}\} . \qquad (7)$$

such that:

$$\mathrm{Cbexp}[\![B]\!]\Big(\bigcup_{k\in\mathcal{S}} R_k\Big) = \bigcup_{k\in\mathcal{S}} (\mathrm{Cbexp}[\![B]\!]R_k)$$
$$\mathrm{Cbexp}[\![B]\!]\emptyset = \emptyset .$$

Structural specification of the forward collecting semantics of boolean expressions (recall from lecture 8)

$$\mathrm{Cbexp}[\![\mathtt{true}]\!]R \stackrel{\mathrm{def}}{=} R$$
$$\mathrm{Cbexp}[\![\mathtt{false}]\!]R \stackrel{\mathrm{def}}{=} \emptyset$$
$$\mathrm{Cbexp}[\![A_1 \ \mathsf{c}\ A_2]\!] \stackrel{\mathrm{def}}{=} \underline{\mathsf{c}}^{C}\ (\mathrm{Faexp}[\![A_1]\!], \mathrm{Faexp}[\![A_2]\!])R$$
$$\text{where } \underline{\mathsf{c}}^{C}\ (F,G)R \stackrel{\mathrm{def}}{=} \{\rho \in R \mid \exists v_1 \in F(\{\rho\}) \cap \mathbb{I} : \exists v_2 \in G(\{\rho\}) \cap \mathbb{I} : v_1\ \underline{\mathsf{c}}\ v_2 = \mathtt{tt}\}$$
$$\mathrm{Cbexp}[\![B_1 \ \&\ B_2]\!]R \stackrel{\mathrm{def}}{=} \mathrm{Cbexp}[\![B_1]\!]R \cap \mathrm{Cbexp}[\![B_2]\!]R$$
$$\mathrm{Cbexp}[\![B_1 \mid B_2]\!]R \stackrel{\mathrm{def}}{=} \mathrm{Cbexp}[\![B_1]\!]R \cup \mathrm{Cbexp}[\![B_2]\!]R$$

Backward collecting semantics of arithmetic expressions (recall)

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:

$$\mathrm{Baexp} \in \mathrm{Aexp} \mapsto \wp(\mathbb{R}) \stackrel{\sqcup}{\longmapsto} \wp(\mathbb{I}_\Omega) \stackrel{\sqcup}{\longmapsto} \wp(\mathbb{R}),$$
$$\mathrm{Baexp}[\![A]\!](R)P \stackrel{\mathrm{def}}{=} \{\rho \in R \mid \exists i \in P \cap \mathbb{I} : \rho \vdash A \Mapsto i\} . (8)$$

$\forall P \in \wp(\mathbb{R}) : \lambda R \cdot \mathrm{Baexp}[\![A]\!](R)P$ is a lower closure operator

Structural definition of the backward/bottom-up collecting semantics of arithmetic expressions

$$\mathrm{Baexp}[\![\mathtt{n}]\!](R)P = (\underline{\mathtt{n}} \in P \cap \mathbb{I}\ ?\ R : \emptyset) \qquad (9)$$
$$\mathrm{Baexp}[\![\mathtt{X}]\!](R)P = \{\rho \in R \mid \rho(\mathtt{X}) \in P \cap \mathbb{I}\} \qquad (10)$$
$$\mathrm{Baexp}[\![\mathtt{?}]\!](R)P = (P \cap \mathbb{I} = \emptyset\ ?\ \emptyset : R) \qquad (11)$$
$$\mathrm{Baexp}[\![\mathtt{u}A']\!](R)P = \mathrm{Baexp}[\![A']\!](R)(\underline{\mathtt{u}}_{C}^{\triangleleft}(\mathrm{Faexp}[\![A']\!]R, P)) \qquad (12)$$
$$\text{where } \underline{\mathtt{u}}_{C}^{\triangleleft}(Q,P) \stackrel{\mathrm{def}}{=} \{v \in Q \mid \underline{\mathtt{u}}\, v \in P \cap \mathbb{I}\}$$
$$\mathrm{Baexp}[\![A_1\ \mathtt{b}\ A_2]\!](R)P = \qquad (13)$$
$$\text{let } \langle P_1,\ P_2\rangle = \underline{\mathtt{b}}_{C}^{\triangleleft}(\mathrm{Faexp}[\![A_1]\!]R, \mathrm{Faexp}[\![A_2]\!]R, P) \text{ in}$$
$$\mathrm{Baexp}[\![A_1]\!](R)P_1 \cap \mathrm{Baexp}[\![A_2]\!](R)P_2$$
$$\text{where } \underline{\mathtt{b}}_{C}^{\triangleleft}(P_1, P_2, P) \stackrel{\mathrm{def}}{=} \{\langle v_1,\ v_2\rangle \in P_1 \times P_2 \mid v_1\ \underline{\mathtt{b}}\ v_2 \in P \cap \mathbb{I}\}$$

Definition of the postcondition semantics of commands (recall from lecture 14)

The postcondition semantics of a command $C \in \mathrm{Com}$ (within a given program P) specifies the strongest postcondition $\mathrm{FPcom}[\![C]\!]R$ satisfied by environments resulting from the execution of the command C starting in any of the environments satisfying the precondition R, if and when this execution terminates.

$$\mathrm{FPcom} \in \mathrm{Com} \mapsto \wp(\mathrm{Env}[\![P]\!]) \stackrel{\sqcup}{\longmapsto} \wp(\mathrm{Env}[\![P]\!]) \tag{14}$$
$$\mathrm{FPcom}[\![C]\!]R \stackrel{\mathrm{def}}{=} \{\rho' \mid \exists \rho \in R : \langle\langle \mathrm{at}_P[\![C]\!],\ \rho\rangle,\ \langle \mathrm{after}_P[\![C]\!],\ \rho'\rangle\rangle \in \tau^\star[\![C]\!]\}$$

The postcondition semantics of a command can be understood, up to an interpretation, as a predicate transformer.

Backward/precondition collecting semantics

Definition of the backward/precondition collecting semantics

– The *backward collecting semantics* of commands C is

$$\mathrm{BPcom} \in \mathrm{Com} \mapsto \wp(\mathrm{Env}[\![P]\!]) \stackrel{\sqcup}{\longmapsto} \wp(\mathrm{Env}[\![P]\!]) \tag{15}$$
$$\mathrm{BPcom}[\![C]\!]R \stackrel{\mathrm{def}}{=} \{\rho \mid \exists \rho' \in R : \langle\langle \mathrm{at}_P[\![C]\!],\ \rho\rangle,\ \langle \mathrm{after}_P[\![C]\!],\ \rho'\rangle\rangle \in \tau^\star[\![C]\!]\}$$

– $\mathrm{BPcom}[\![C]\!]R$ is the weakest precondition for execution of P from entry point of command C to have the possibility to reach a final state in R on exit of the command

– The condition is necessary but not sufficient, because of non-determinism. So non-termination or termination in a state not in R is left opened

Structural specification of the backward collecting semantics

$$\mathrm{BPcom}[\![\mathtt{skip}]\!]R = R$$
$$\mathrm{BPcom}[\![\mathtt{X} := A]\!]R = \{\rho \mid \exists i \in \mathbb{I} : \rho \vdash A \Mapsto i \wedge \rho[\mathtt{X} := i] \in R\}$$
$$\mathrm{BPcom}[\![\mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}]\!]R =$$
$$\mathrm{Cbexp}[\![B]\!](\mathrm{BPcom}[\![S_t]\!]R) \cup \mathrm{Cbexp}[\![T(\neg(B))]\!](\mathrm{BPcom}[\![S_f]\!]R)$$
$$\mathrm{BPcom}[\![\mathtt{while}\ B\ \mathtt{do}\ S\ \mathtt{od}]\!]R =$$
$$\mathsf{lfp}_{\emptyset}^{\subseteq}\ \lambda X \cdot \mathrm{Cbexp}[\![T(\neg(B))]\!]R \cup \mathrm{Cbexp}[\![B]\!](\mathrm{BPcom}[\![S]\!]X)$$
$$\mathrm{BPcom}[\![C\ ;\ S]\!]R = (\mathrm{BPcom}[\![C]\!] \circ \mathrm{BPcom}[\![S]\!])R$$
$$\mathrm{BPcom}[\![S\ ;;]\!]R = \mathrm{BPcom}[\![S]\!]$$

Note

If we define:

$\mathbb{R}_P = \wp(\mathrm{Env}[\![P]\!])$

$\alpha_P[\![C]\!] \in \wp(\Sigma[\![P]\!] \times \Sigma[\![P]\!]) \longrightarrow (\wp(\mathbb{R}) \longrightarrow \wp(\mathbb{R}))$

$\alpha_P[\![C]\!]X = \lambda R.\{\rho \mid \exists \rho' \in R : \langle\langle at_P[\![C]\!], \rho\rangle, \langle after_P[\![C]\!], \rho'\rangle\rangle \in X\}$

then

$\mathcal{B}com[\![C]\!]R = \alpha[\![C]\!](\tau^*[\![C]\!])$ is an abstract interpretation of the big-step operational semantics $t^*[\![C]\!]$

□

Proof of correctness of the structural specification of the backward collecting semantics

PROOF.

- We observe that $\alpha_P[\![C]\!]$ is a complete join morphism:

$\alpha_P[\![C]\!](\bigcup_{i\in\Delta} X_i)$

$= \lambda R.\{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![C]\!],\rho\rangle,\langle after_P[\![C]\!],\rho'\rangle\rangle \in \bigcup_{i\in\Delta} X_i\}$

$= \lambda R.\bigcup_{i\in\Delta}\{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![C]\!],\rho\rangle,\langle after_P[\![C]\!],\rho'\rangle\rangle \in X_i\}$

$= \dot{\bigcup_{i\in\Delta}} \alpha_P[\![C]\!](X_i)$

so that this is the lower-adjoint of a Galois connection:

$\langle \wp(\Sigma[\![P]\!]\times\Sigma[\![P]\!]), \subseteq\rangle \underset{\alpha[\![C]\!]}{\overset{\gamma_P[\![C]\!]}{\leftrightarrows}} \langle \wp(\mathbb{R}) \to \wp(\mathbb{R}), \dot\subseteq\rangle$

for the pointwise extension $\dot\subseteq$ of $\subseteq$.

- We proceed by structural induction on the structure of programs.
- $\mathcal{B}com[\![skip]\!]R$

$\triangleq \alpha_P[\![skip]\!](t^*[\![skip]\!])R$

$= \alpha_P[\![skip]\!](1_{\Sigma[\![P]\!]} \cup \tau[\![skip]\!])R$

$= \{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![skip]\!],\rho\rangle,\langle after_P[\![skip]\!],\rho'\rangle\rangle \in 1_{\Sigma[\![P]\!]}\cup\tau[\![skip]\!]\}$

$= \{\rho \mid \exists\rho'\in R : \rho'=\rho\}$ $\{at_P[\![C]\!] \neq after_P[\![C]\!]$ and def. $\tau[\![skip]\!]\}$

$= R$

- $\mathcal{B}com[\![X:=A]\!]R$

$\triangleq \alpha_P[\![X:=A]\!](t^*[\![X:=A]\!])R$

$= \alpha_P[\![X:=A]\!](1_{\Sigma[\![P]\!]}\cup\tau[\![X:=A]\!])R$

$= \{\rho\mid\exists\rho'\in R : \langle\langle at_P[\![X:=E]\!],\rho\rangle,\langle after_P[\![X:=E]\!],\rho'\rangle\rangle \in 1_{\Sigma[\![P]\!]}\cup\tau[\![X:=A]\!]\}$

$= \{\rho\mid\exists\rho'\in R : \langle\langle at_P[\![X:=A]\!],\rho\rangle,\langle after_P[\![X:=A]\!],\rho'\rangle\rangle \in \{\langle\langle at_P[\![X:=A]\!],\rho\rangle,\langle after_P[\![X:=A]\!],\rho[X\leftarrow i]\rangle\rangle \mid i\in\mathbb{I} \wedge \rho\vdash A \Rightarrow i\}\}$

$\{$since $at_P[\![C]\!]\neq after_P[\![C]\!]$ and def. $\tau[\![X:=A]\!]\}$

$= \{\rho\mid\exists\rho'\in R : \exists i\in\mathbb{I} : \rho'=\rho[X\leftarrow i]\wedge\rho\vdash A\Rightarrow i\}$

$= \{\rho\mid\exists i\in\mathbb{I} : \rho\vdash A\Rightarrow i \wedge \rho[X\leftarrow i]\in R\}$

- $\mathcal{B}\mathrm{com}[\![C]\!]R$ where $C = $ **if** B **then** S_t **else** S_f **fi**

$$\triangleq \alpha_p[\![C]\!]\,(t^*[\![C]\!])\,R$$

$$= \alpha_p[\![C]\!]\Big(\big((1_{\Sigma[\![P]\!]} \cup \tau^B) \circ t^*[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t)\big) \cup \big((1_{\Sigma[\![P]\!]} \cup \tau^{\overline{B}}) \circ t^*[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^f)\big)\Big)\,R$$

$$= \alpha_p[\![C]\!]\big((1_{\Sigma[\![P]\!]} \cup \tau^B) \circ t^*[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t)\big)\,R$$

$$\dot{\cup}\ \alpha_p[\![C]\!]\big((1_{\Sigma[\![P]\!]} \cup \tau^{\overline{B}}) \circ t^*[\![S_f]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^f)\big)\,R$$

We handle the case of the true alternative, the false alternative being handled in the same way.

$$\alpha_p[\![C]\!]\big((1_{\Sigma[\![P]\!]} \cup \tau^B) \circ t^*[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t)\big)\,R$$

$$= \{\rho \mid \exists \rho' \in R : \langle\!\langle \underline{\mathrm{at}}_p[\![C]\!], \rho\rangle, \langle \underline{\mathrm{after}}_p[\![C]\!], \rho'\rangle\!\rangle \in \big((1_{\Sigma[\![P]\!]} \cup \tau^B) \circ t^*[\![S_t]\!] \circ (1_{\Sigma[\![P]\!]} \cup \tau^t)\big)\}$$

⦃by def. $\alpha[\![C]\!]$⦄

$$= \{\rho \mid \exists \rho''' \in R : \exists \rho', \rho'' \in R : \exists \ell', \ell'' \in \underline{\mathrm{in}}_p[\![C]\!] :$$
$$\langle\!\langle \underline{\mathrm{at}}_p[\![C]\!], \rho\rangle, \langle \ell', \rho'\rangle\!\rangle \in (1_{\Sigma[\![P]\!]} \cup \tau^B)$$
$$\wedge\ \langle\!\langle \ell', \rho'\rangle, \langle \ell'', \rho''\rangle\!\rangle \in \tau^*[\![S_t]\!]$$
$$\wedge\ \langle\!\langle \ell'', \rho''\rangle, \langle \underline{\mathrm{after}}[\![C]\!], \rho'''\rangle\!\rangle \in (1_{\Sigma[\![P]\!]} \cup \tau^t)\}$$

⦃by def. of the composition $\circ$ of relations⦄

= ⦃We observe that $\underline{\mathrm{at}}_p[\![C]\!] = \ell'$ is impossible since $\ell' \in \underline{\mathrm{in}}_p[\![S_t]\!]$ would imply $\underline{\mathrm{at}}_p[\![C]\!] \in \underline{\mathrm{in}}_p[\![S_t]\!]$ in contradiction with (59) i.e. $\{\underline{\mathrm{at}}_p[\![C]\!], \underline{\mathrm{after}}_p[\![C]\!]\} \cap (\underline{\mathrm{in}}_p[\![S_t]\!] \cup \underline{\mathrm{in}}_p[\![S_f]\!]) = \emptyset$;
We observe that $\underline{\mathrm{after}}_p[\![C]\!] = \ell''$ is impossible for similar reasons.⦄

$$\{\rho \mid \exists \rho''' \in R : \exists \rho', \rho'' \in R : \exists \ell', \ell'' \in \underline{\mathrm{in}}_p[\![C]\!] :$$
$$\langle\!\langle \underline{\mathrm{at}}_p[\![C]\!], \rho\rangle, \langle \ell', \rho'\rangle\!\rangle \in \tau^B$$
$$\wedge\ \langle\!\langle \ell', \rho'\rangle, \langle \ell'', \rho''\rangle\!\rangle \in \tau^*[\![S_t]\!]$$
$$\wedge\ \langle\!\langle \ell'', \rho''\rangle, \langle \underline{\mathrm{after}}_p[\![C]\!], \rho'''\rangle\!\rangle \in \tau^t\}$$

= ⦃By def. of τ^B, $\rho' = \rho$ and $\ell' = \underline{\mathrm{at}}_p[\![S_t]\!]$. By def. of τ^t, $\rho'' = \rho'''$ and $\ell'' = \underline{\mathrm{after}}_p[\![S_t]\!]$⦄

$$\{\rho \mid \exists \rho''' \in R : \langle\!\langle \underline{\mathrm{at}}_p[\![C]\!], \rho\rangle, \langle \underline{\mathrm{at}}_p[\![S_t]\!], \rho\rangle\!\rangle \in \tau^B\ \wedge$$
$$\langle\!\langle \underline{\mathrm{at}}_p[\![S_t]\!], \rho\rangle, \langle \underline{\mathrm{after}}_p[\![S_t]\!], \rho'''\rangle\!\rangle \in \tau^*[\![S_t]\!]\ \wedge$$
$$\langle\!\langle \underline{\mathrm{after}}_p[\![S_t]\!], \rho'''\rangle, \langle \underline{\mathrm{after}}_p[\![C]\!], \rho'''\rangle\!\rangle \in \tau^t\}$$

= { by def. τ^B and τ^t }

$\{\rho \mid \exists \rho' \in R : \rho \vdash B \Rightarrow tt \wedge \langle\langle at_P[\![St]\!], \rho\rangle, \langle after[\![St]\!], \rho'\rangle\rangle \in \tau^*[\![St]\!]\}$

$= \{\rho \in \{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![St]\!],\rho\rangle, \langle after[\![St]\!],\rho'\rangle\rangle \in \tau^*[\![St]\!]\}$
$\mid \rho\vdash B \Rightarrow tt\}$

$= \{\rho \in \alpha_P[\![St]\!](\tau^*[\![St]\!])R \mid \rho \vdash B \Rightarrow tt\}$

= { by def. $\mathcal{C}bexp[\![B]\!]R$ }

$\mathcal{C}bexp[\![B]\!](\alpha_P[\![St]\!](\tau^*[\![St]\!])R)$

= { by induction hypothesis }

$\mathcal{C}bexp[\![B]\!](\mathcal{B}com[\![St]\!]R)$

The false alternative is similar with Sf for St and $T(\neg B)$ for B.

– $\mathcal{B}com[\![C]\!]R$ where C = while B do S od

$= \alpha_P[\![C]\!](t^*[\![C]\!])R$

$= \alpha_P[\![C]\!](((1_{\Sigma[\![P]\!]} \cup t^*[\![S]\!]\circ\tau^R)\circ(\tau^B\circ\tau^*[\![S]\!]\circ\tau^R)^*\circ$
$(1_{\Sigma[\![P]\!]}\cup\tau^B\circ t^*[\![S]\!]\cup\tau^{\bar B})) \cup t^*[\![S]\!])R$

$= \{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![C]\!],\rho\rangle, \langle after_P[\![C]\!],\rho'\rangle\rangle\in \tau^*[\![S]\!]\}$

$\cup \{\rho \mid \exists \rho'''\in R : \exists\rho',\rho''\in R : \exists \ell',\ell''\in in_P[\![C]\!] :$
$\langle\langle at_P[\![C]\!],\rho\rangle,\langle\ell',\rho'\rangle\rangle \in (1_{\Sigma[\![P]\!]}\cup t^*[\![S]\!]\circ\tau^R) \wedge$
$\langle\langle\ell',\rho'\rangle,\langle\ell'',\rho''\rangle\rangle\in(\tau^B\circ\tau^*[\![S]\!]\circ\tau^R)^*\wedge$
$\langle\langle\ell'',\rho''\rangle,\langle after_P[\![C]\!],\rho'''\rangle\rangle \in (1_{\Sigma[\![P]\!]}\cup\tau^B\circ t^*[\![S]\!]\cup\tau^{\bar B})\}$

= { The first term is $\emptyset$ since $\{at_P[\![C]\!], after_P[\![C]\!]\} \cap in_P[\![S]\!] = \emptyset$ by (60) and $\langle\langle\ell_1,\rho_1\rangle,\langle\ell_2,\rho_2\rangle\rangle \in \tau^*[\![S]\!]$ implies $\ell_1,\ell_2 \in in_P[\![S]\!]$.

. for the second term, $\ell' = at_P[\![C]\!]$ since otherwise $\exists \ell_2,\rho_2 :$ $\langle\langle at_P[\![C]\!],\rho\rangle,\langle\ell_2,\rho_2\rangle\rangle \in t^*[\![S]\!]\circ\tau^R$ which would imply $at_P[\![C]\!] \in in_P[\![S]\!]$ in contradiction with (60).

. for the second term, $\langle\langle\ell'',\rho''\rangle,\langle after_P[\![C]\!],\rho'''\rangle\rangle \in 1_{\Sigma[\![P]\!]}$ is impossible since then $\langle\langle\ell',\rho'\rangle,\langle after_P[\![C]\!],\rho''\rangle\rangle \in (\tau^B\circ\tau^*[\![S]\!]\circ\tau^R)^n$, with $n\geq 0$ which is itself impossible since either $n=0$ and $\ell' = after_P[\![C]\!]$ so $\ell' = at_P[\![C]\!] = after_P[\![C]\!]$ in contradiction with (56) or $n>0$ and $\exists \ell_1,\rho_1 : \langle\langle\ell_1,\rho_1\rangle,\langle after_P[\![C]\!],\rho''\rangle\rangle \in \tau^R$ in contradiction with the definition of τ^R and (56) }

$\{\rho \mid \exists\rho'''\in R : \langle\langle at_P[\![C]\!],\rho\rangle,\langle\ell'',\rho''\rangle\rangle \in (\tau^B\circ\tau^*[\![S]\!]\circ\tau^R)^*$
$\wedge \langle\langle\ell'',\rho''\rangle,\langle after_C[\![C]\!],\rho'''\rangle\rangle \in \tau^{\bar B}\}$

= { by def. $\tau^{\bar B}$ so that $\ell'' = at_P[\![C]\!]$ and $\rho''=\rho'''$ }

$\{\rho \mid \exists\rho'\in R : \langle\langle at_P[\![C]\!],\rho\rangle,\langle at_P[\![C]\!],\rho'\rangle\rangle \in (\tau^B\circ\tau^*[\![S]\!]\circ\tau^R)^*$
$\wedge\ \rho' \vdash T(\neg B) \Rightarrow tt\}$

$= \{\rho \mid \exists \rho' \in \{\rho'' \in \mathcal{R} \mid \rho'' \vdash T(\neg B) \Rightarrow tt\} : \langle\langle at_P[\![C]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in (t^B \circ \tau^*[\![S]\!] \circ \tau^R)^*\}$

$= (\text{def. } \mathcal{C}bexp\,)$

$\{\rho \mid \exists \rho' \in \mathcal{C}bexp[\![T(\neg B)]\!]R : \langle\langle at_P[\![C]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in (t^B \circ \tau^*[\![S]\!] \circ \tau^R)^*\}$

$= \alpha'((t^B \circ \tau^*[\![S]\!] \circ \tau^R)^*)$

by defining (for a given while command C and a given $R \in \wp(\mathcal{R})$):

$\overline{R} = \mathcal{C}bexp[\![T(\neg B)]\!]R$

$\alpha'(t) \triangleq \{\rho \mid \exists \rho' \in \overline{R} : \langle\langle at_P[\![C]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in t\}$

We have

$\alpha'(\bigcup_{i \in \Delta} t_i) = \bigcup_{i \in \Delta} \alpha'(t_i)$

whence a Galois connection

$\langle \wp(\Sigma[\![P]\!] \times \Sigma[\![P]\!]), \subseteq\rangle \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} \langle \wp(\mathcal{R}), \subseteq\rangle$

such that

$\alpha'(t^*)$ where $t = (\tau^B \circ \tau^*[\![S]\!] \circ \tau^R)$

$= \alpha'(\mathrm{lfp}\ \lambda X.\, 1 \cup t \circ X)$

$= \mathrm{lfp}\ F'$

where $\alpha'(1 \cup t \circ X) = F'(\alpha'(X))$.

i.e. $\alpha'(1) \cup \alpha'(t \circ X) = F'(\alpha'(X))$

since α' is a complete join morphism.

- $\alpha'(1)$

$= \{\rho' \mid \exists \rho \in \overline{R} : \langle\langle at_P[\![C]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in 1_{\Sigma[\![P]\!]}\}$

$= \{\rho' \mid \exists \rho \in \overline{R} : \rho' = \rho\}$

$= \overline{R}$

- $\alpha'(t \circ X)$

$= \{\rho \mid \exists \rho' \in \overline{R} : \langle\langle at_P[\![C]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in \tau^B \circ \tau^*[\![S]\!] \circ \tau^R \circ X\}$

$= (\text{def. composition } \circ\,)$

$\{\rho \mid \exists \rho'''' \in \overline{R} : \exists \rho', \rho'', \rho''' \in \mathcal{R} : \exists \ell', \ell'', \ell''', \ell'''' \in in_P[\![C]\!] :$

$\quad \langle\langle at_P[\![C]\!], \rho\rangle, \langle \ell', \rho'\rangle\rangle \in \tau^B$

$\quad \wedge \langle\langle \ell', \rho'\rangle, \langle \ell'', \rho''\rangle\rangle \in \tau^*[\![S]\!]$

$\quad \wedge \langle\langle \ell'', \rho''\rangle, \langle \ell''', \rho'''\rangle\rangle \in \tau^R$

$\quad \wedge \langle\langle \ell''', \rho'''\rangle, \langle at_P[\![C]\!], \rho''''\rangle\rangle \in X\}$

$=$ (by def of τ^B, we have $\ell' = at_P[\![S]\!]$, $\rho' = \rho$ and by definition of τ^R, we have $\ell'' = \underline{after}_P[\![S]\!]$, $\rho'' = \rho'''$ and $\ell''' = \underline{after}_P[\![C]\!]$)

$\{\rho \mid \exists \rho'''' \in \overline{R} : \exists \rho'' \in \mathcal{R} : \rho \vdash B \Rightarrow tt$

$\quad \wedge \langle\langle at_P[\![S]\!], \rho\rangle, \langle \underline{after}_P[\![S]\!], \rho''\rangle\rangle \in \tau^*[\![S]\!]$

$\quad \wedge \langle\langle \underline{after}_P[\![S]\!], \rho''\rangle, \langle at_P[\![C]\!], \rho''''\rangle\rangle \in X\}$

$$= \{\rho \in \{\rho \mid \exists \rho'''' \in \overline{R} : \exists \rho'' \in R : \langle\langle at_P[\![S]\!], \rho\rangle, \langle after_P[\![S]\!], \rho''\rangle \in \tau^*[\![S]\!] \wedge \langle\langle after_P[\![S]\!], \rho''\rangle, \langle at_P[\![C]\!], \rho''''\rangle \in X\} \mid \rho \vdash B \Rightarrow tt\}$$

$= \{$ by def of $\mathcal{C}bexp$ $\}$

$$\mathcal{C}bexp[\![B]\!](\{\rho \mid \exists \rho' \in \{\rho \mid \exists \rho' \in \overline{R} : \langle after[\![S]\!], \rho\rangle, \langle at_P[\![C]\!], \rho'\rangle\rangle \in X\} : \langle\langle at_P[\![S]\!], \rho\rangle, \langle after_P[\![S]\!], \rho'\rangle \in \tau^*[\![S]\!]\})$$

$= \{$ $after[\![S]\!] = at[\![C]\!]$ and def. α' $\}$

$$\mathcal{C}bexp[\![B]\!](\{\rho \mid \exists \rho' \in \alpha'(X) : \langle\langle at_P[\![S]\!], \rho\rangle, \langle after_P[\![S]\!], \rho'\rangle \in \tau^*[\![S]\!]\})$$

$= \{$ def. $\alpha_P[\![S]\!]$ $\}$

$$\mathcal{C}bexp[\![B]\!](\alpha_P[\![S]\!](\tau^*[\![S]\!])(\alpha'(X)))$$

$= \{$ structural induction hypothesis $\}$

$$\mathcal{C}bexp[\![B]\!](\mathcal{B}com[\![S]\!](\alpha'(X)))$$

So $F'(X) = \mathcal{C}bexp[\![T(\neg B)]\!]R \;\cup$
$\qquad \mathcal{C}bexp[\![B]\!](\mathcal{B}com[\![S]\!]X)$

and $\mathcal{B}com[\![C]\!]R = \mathrm{lfp}^{\subseteq}_{\emptyset} F'$

■ $\mathcal{B}com[\![C; S]\!]R$

$= \alpha[\![C;S]\!](t^*[\![C; S]\!])R$

$= \alpha[\![C;S]\!](t^*[\![C]\!] \circ t^*[\![S]\!])R$

$= \{\rho \mid \exists \rho' \in R : \langle\langle at_P[\![C;S]\!], \rho\rangle, \langle after_P[\![C;S]\!], \rho'\rangle\rangle \in t^*[\![C]\!] \circ t^*[\![S]\!]\}$

$= \{\rho \mid \exists \rho' \in R : \langle\langle at_P[\![C]\!], \rho\rangle, \langle after_P[\![S]\!], \rho'\rangle\rangle \in t^*[\![C]\!] \circ t^*[\![S]\!]\}$

$\{$ by (57) $\}$

$= \{\rho \mid \exists \rho'' \in R, \rho' \in R, \ell' \in in_P[\![C;S]\!] :$
$\quad \langle\langle at_P[\![C]\!], \rho\rangle, \langle \ell', \rho'\rangle\rangle \in t^*[\![C]\!]$
$\quad \wedge \langle\langle \ell', \rho'\rangle, \langle after_P[\![S]\!], \rho''\rangle\rangle \in t^*[\![S]\!]\}$

$= \{$ since $\ell' \in in_P[\![C]\!]$ and $\ell' \in in_P[\![S]\!]$, we must have $\ell' = after_P[\![C]\!] = at_P[\![S]\!]$ $\}$

$\{\rho \mid \exists \rho'' \in R : \rho' \in : \langle\langle at_P[\![C]\!], \rho\rangle, \langle after_P[\![C]\!], \rho'\rangle \in t^*[\![C]\!] \wedge$
$\quad \langle\langle at_P[\![S]\!], \rho'\rangle, \langle after_P[\![S]\!], \rho''\rangle \in t^*[\![S]\!]\}$

$= \{\rho \mid \exists \rho' \in \{\rho' \mid \exists \rho'' \in R : \langle\langle at_P[\![S]\!], \rho'\rangle, \langle after_P[\![S]\!], \rho''\rangle \in \tau^*[\![S]\!]\} :$
$\quad \langle\langle at_P[\![C]\!], \rho\rangle, \langle after_P[\![C]\!], \rho'\rangle\rangle \in t^*[\![C]\!]\}$

$= \mathcal{B}com[\![C]\!](\{\rho' \mid \exists \rho'' \in R : \langle\langle at_P[\![S]\!]\rho'\rangle, \langle after_P[\![S]\!], \rho''\rangle\rangle \in t^*[\![S]\!]\})$

$= \mathcal{B}com[\![C]\!](\mathcal{B}com[\![S]\!] R)$

$= (\mathcal{B}com[\![C]\!] \circ \mathcal{B}com[\![S]\!])(R)$

- $\mathcal{B}\mathrm{Com}[\![C\,;;]\!]R$
$= \{\rho \mid \exists \rho' \in R : \langle\langle \mathrm{at}_{P;;}[\![C;;]\!], \rho\rangle, \langle \underline{\mathrm{after}}_{P;;}[\![C;;]\!], \rho'\rangle\rangle \in t^\star[\![C;;]\!]\}$
$= \{\rho \mid \exists \rho' \in R : \langle\langle \mathrm{at}_{C;}[\![C]\!], \rho\rangle, \langle \mathrm{after}_{C;}[\![C]\!], \rho'\rangle\rangle \in t^\star[\![C]\!]\}$
$= \mathcal{B}\mathrm{Com}[\![C]\!]R$

$\square$

$\square$

Backward/precondition relational abstract semantics

Definition of the backward/precondition collecting semantics

- The *abstract backward/precondition semantics* $\mathrm{Abcom}[\![C]\!]R$ of command C in abstract environment R satisfies:

$$\mathrm{Abcom}[\![C]\!]R \mathrel{\dot{\sqsupseteq}} \alpha(\mathrm{BPcom}[\![C]\!](\gamma(R)))$$

<u>Note</u> : this is the case when

$$\langle(\wp(\mathbb{R}_P) \to \wp(\mathbb{R}_P)), \dot{\subseteq}\rangle \xrightleftharpoons[\alpha]{\gamma} \langle L, \sqsubseteq\rangle.$$

where $\mathbb{R}_P = \wp(\mathrm{Env}[\![P]\!])$. In absence of best approximation one would use a concretization function only where the soundness condition $\alpha \circ F \circ \gamma \sqsubseteq F^\#$ becomes $F \circ \gamma \sqsubseteq \gamma \circ F^\#$ for all concrete/abstract operations $F/F^\#$ $\square$

Structural specification of the backward collecting semantics

$$\mathrm{Abcom}[\![\mathtt{skip}]\!]R \;=\; R$$

$$\mathrm{Abcom}[\![\mathtt{X} := A]\!]R \;\dot{\sqsupseteq}\; \alpha(\{\rho \mid \exists i \in \mathbb{I} : \rho \vdash A \mapsto i \wedge \rho[\mathtt{X} := i] \in \gamma(R)\})$$

$$\mathrm{Abcom}[\![\mathtt{if}\ B\ \mathtt{then}\ S_t\ \mathtt{else}\ S_f\ \mathtt{fi}]\!]R =$$
$$\mathrm{Abexp}[\![B]\!](\mathrm{Abcom}[\![S_t]\!]R) \;\dot{\sqcup}\; \mathrm{Abexp}[\![T(\neg(B))]\!](\mathrm{Abcom}[\![S_f]\!]R)$$

$$\mathrm{Abcom}[\![\mathtt{while}\ B\ \mathtt{do}\ S\ \mathtt{od}]\!]R =$$
$$\mathsf{lfp}_{\emptyset}^{\subseteq}\ \lambda X \cdot \mathrm{Abexp}[\![T(\neg(B))]\!]R \;\dot{\sqcup}\; \mathrm{Abexp}[\![B]\!](\mathrm{Abcom}[\![S]\!]X)$$

$$\mathrm{Abcom}[\![C\ ;\ S]\!]R \;=\; (\mathrm{Abcom}[\![C]\!] \circ \mathrm{Abcom}[\![S]\!])R$$

$$\mathrm{Abcom}[\![S\ ;;]\!]R \;=\; \mathrm{Abcom}[\![S]\!]$$

PROOF. The soundness condition is:

$$\alpha \circ \mathrm{Bcom}[\![C]\!] \circ \gamma \sqsubseteq \mathrm{Abcom}[\![C]\!]$$

for Galois connections, or more generally

$$\mathrm{Bcom}[\![C]\!] \circ \gamma \sqsubseteq \mathrm{Abcom}[\![C]\!] \circ \gamma$$

in absence of best approximation and use of a concretization function only.

The proof is by structural induction on commands

- $\alpha \circ \mathrm{Bcom}[\![\mathtt{skip}]\!] \circ \gamma$
$= \alpha \circ \gamma$ {by def. $\mathrm{Bcom}[\![\mathtt{skip}]\!]$}
$\sqsubseteq 1$ {$\alpha \circ \gamma$ is reductive}.
$\triangleq \mathrm{Abcom}$

In case of concretization only we have
$\mathrm{Bcom}[\![\mathtt{skip}]\!] \circ \gamma \sqsubseteq \gamma \circ \mathrm{Abcom}[\![\mathtt{skip}]\!]$
$\Leftrightarrow 1 \circ \gamma \sqsubseteq \gamma \circ 1$ {where 1 is identity}
$\Leftrightarrow$ true

- For the assignment the soundness follows directly from its definition and that of Bcom.

- For the test $C =$ if Bexp then LC_1 else LC_2 fi

$(\alpha \circ \mathrm{Bcom}[\![C]\!] \circ \gamma)R$
$= \alpha(\mathrm{Cbexp}[\![\mathrm{Bexp}]\!](\mathrm{Bcom}[\![LC_1]\!]\gamma(R))$
$\cup\ \mathrm{Cbexp}[\![T(\neg \mathrm{Bexp})]\!](\mathrm{Bcom}[\![LC_2]\!]\gamma(R)))$
$=$ {$\gamma \circ \alpha$ is extensive, $\mathrm{Cbexp}[\![B]\!]$ is monotonic and α is a complete join morphism}

$$\alpha(\mathit{Cbexp}[\![\mathit{Bexp}]\!](\gamma(\alpha(\mathit{Bcom}[\![\mathit{LC}_1]\!]\,\gamma(R)))))$$
$$\sqcup\ \alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!](\gamma(\alpha(\mathit{Bcom}[\![\mathit{LC}_2]\!]\,\gamma(R)))))$$
$$\sqsubseteq \{\text{induction hypothesis}, \alpha, \mathit{Cbexp}[\![B]\!], \gamma \text{ monotone}\}$$
$$\alpha(\mathit{Cbexp}[\![\mathit{Bexp}]\!](\gamma(\mathit{Abcom}[\![\mathit{LC}_1]\!]R)))$$
$$\sqcup\ \alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!](\gamma(\mathit{Abcom}[\![\mathit{LC}_2]\!]R)))$$
$$\sqsubseteq \{\alpha\circ\mathit{Cbexp}[\![B]\!]\circ\gamma \mathrel{\dot{\sqsubseteq}} \mathit{Cbexp}^{\#}[\![B]\!] \text{ hypothesis}\}$$
$$\mathit{Cbexp}^{\#}[\![\mathit{Bexp}]\!](\mathit{Abcom}[\![\mathit{LC}_1]\!]R)$$
$$\sqcup\ \mathit{Cbexp}^{\#}[\![T(\neg\mathit{Bexp})]\!](\mathit{Abcom}[\![\mathit{LC}_2]\!]R)$$
$$\triangleq \mathit{Abcom}[\![C]\!]R.$$

In case of a γ-based abstract interpretation, we would have

$$(\mathit{Cbexp}[\![\mathit{Bexp}]\!](\mathit{Bcom}[\![\mathit{LC}_1]\!](\gamma(R))))$$
$$\cup\ (\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!](\mathit{Bcom}[\![\mathit{LC}_2]\!](\gamma(R))))$$
$$\subseteq \{\mathit{Cbexp}[\![B]\!] \text{ is monotone and induction hypothesis}\}$$
$$\mathit{Cbexp}[\![\mathit{Bexp}]\!](\gamma(\mathit{Abcom}[\![\mathit{LC}_1]\!]R))$$
$$\cup\ \mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!](\gamma(\mathit{Abcom}[\![\mathit{LC}_2]\!]R))$$
$$\subseteq \{\mathit{Cbexp}[\![B]\!]\circ\gamma \mathrel{\dot{\subseteq}} \gamma\circ\mathit{Cbexp}^{\#}[\![B]\!]\}$$
$$\gamma(\mathit{Cbexp}^{\#}[\![\mathit{Bexp}]\!](\mathit{Abcom}[\![\mathit{LC}_1]\!]R))$$
$$\cup\ \gamma(\mathit{Cbexp}^{\#}[\![T(\neg\mathit{Bexp})]\!](\mathit{Abcom}[\![\mathit{LC}_2]\!]R))$$

$$\subseteq \{\gamma \text{ is monotone so } \gamma(X)\cup\gamma(Y)\subseteq\gamma(X\sqcup Y) \text{ where } \sqcup \text{ is the abstract lub}\}$$
$$\gamma(\mathit{Cbexp}^{\#}[\![\mathit{Bexp}]\!](\mathit{Abcom}[\![\mathit{LC}_1]\!]R)\ \sqcup$$
$$\mathit{Cbexp}^{\#}[\![T(\neg\mathit{Bexp})]\!](\mathit{Abcom}[\![\mathit{LC}_2]\!]R))$$
$$= \gamma(\mathit{Abcom}[\![C]\!]R).$$

- For the while loop $C = \underline{\texttt{while}}\ \mathit{Bexp}\ \underline{\texttt{do}}\ \mathit{LC}\ \underline{\texttt{od}}$, we have:

$$\alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!]\gamma(R)\cup\mathit{Cbexp}[\![\mathit{Bexp}]\!](\mathit{Bcom}[\![\mathit{LC}]\!]X))$$
$$= \alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!]R)$$
$$\sqcup\ \alpha(\mathit{Cbexp}[\![\mathit{Bexp}]\!](\mathit{Bcom}[\![\mathit{LC}]\!]X))$$
$$\{\text{since } \alpha \text{ is a complete join morphism}\}$$
$$\sqsubseteq \alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!]\gamma(R))\ \sqcup$$
$$\alpha(\mathit{Cbexp}[\![\mathit{Bexp}]\!]\gamma(\alpha(\mathit{Bcom}[\![\mathit{LC}]\!]X)))$$
$$\{\text{since } \gamma\circ\alpha \text{ is extensive and } \mathit{Cbexp}[\![\mathit{Bexp}]\!] \text{ is monotone}\}$$
$$\sqsubseteq \alpha(\mathit{Cbexp}[\![T(\neg\mathit{Bexp})]\!]\gamma(R))\ \sqcup$$
$$\alpha(\mathit{Cbexp}[\![\mathit{Bexp}]\!]\gamma(\mathit{Abcom}[\![\mathit{LC}]\!](\alpha(X))))$$
$$\{\text{by induction hypothesis and monotony}\}$$
$$\sqsubseteq \mathit{Cbexp}^{\#}[\![T(\neg\mathit{Bexp})]\!]R\ \sqcup\ \mathit{Cbexp}^{\#}[\![\mathit{Bexp}]\!](\mathit{Abcom}[\![\mathit{LC}]\!](\alpha(X)))$$

so by the fixpoint approximation, theorem we conclude that

$$\alpha(\mathcal{B}com[\![\text{while } Bexp \text{ do } LCo \text{ od}]\!]\, \gamma(R))$$

$$= \alpha\Big(\mathrm{lfp}^{\subseteq}_{\emptyset}\, \lambda X.\, \mathcal{C}bexp[\![T(\neg Bexp)]\!]\, \gamma(R) \cup \mathcal{C}bexp[\![Bexp]\!](\mathcal{B}com[\![LCo]\!]\, X)\Big)$$

$$\sqsubseteq \mathrm{lfp}^{\sqsubseteq}_{\perp}\, \lambda X.\, \mathcal{C}bexp^{\#}[\![T(\neg Bexp)]\!]\, R \sqcup \mathcal{C}bexp^{\#}[\![Bexp]\!](\mathcal{A}bcom[\![LCo]\!]\, X)$$

The case of a γ-based abstraction is similar. We have:

$$\mathcal{C}bexp[\![T(\neg Bexp)]\!]\, \gamma(R) \cup \mathcal{C}bexp[\![Bexp]\!](\mathcal{B}com[\![LCo]\!]\, \gamma(X))$$

$\subseteq$ { $\mathcal{C}bexp[\![B]\!]$ is monotone, hypothesis on $\mathcal{C}bexp^{\#}$ and induction hypothesis }

$$\gamma(\mathcal{C}bexp^{\#}[\![Bexp]\!]\, R) \cup \mathcal{C}bexp[\![Bexp]\!](\gamma(\mathcal{A}com[\![LCo]\!]\, X))$$

$\subseteq$ { we use once again the hypothesis on $\mathcal{C}bexp^{\#}$ such that $\mathcal{C}bexp[\![B]\!] \circ \gamma \subseteq \gamma \circ \mathcal{C}bexp^{\#}[\![B]\!]$ }

$$\gamma(\mathcal{C}bexp^{\#}[\![Bexp]\!]\, R) \cup \gamma(\mathcal{C}bexp^{\#}[\![B]\!](\mathcal{A}com[\![LCo]\!]\, X))$$

$\subseteq$ { γ is monotone so $X \sqsubseteq X \sqcup Y$ implies $\gamma(X) \subseteq \gamma(X \sqcup Y)$ similarly $\gamma(Y) \subseteq \gamma(X \sqcup Y)$ whence $\gamma(X) \cup \gamma(Y) \subseteq \gamma(X \sqcup Y)$ }

$$\gamma(\mathcal{C}bexp^{\#}[\![Bexp]\!]\, R \sqcup \mathcal{C}bexp^{\#}[\![B]\!](\mathcal{A}com[\![LCo]\!]\, X))$$

Then we have $F \circ \gamma \subseteq \gamma \circ F^{\#}$ which implies $\mathrm{lfp}\, F \subseteq \gamma(\mathrm{lfp}\, F^{\#})$ since by transfinite induction $\perp \subseteq \gamma(\perp^{\#})$, if $X^{\delta} \subseteq \gamma(X^{\#\delta})$ then $F(X^{\delta}) \subseteq F \circ \gamma(X^{\#\delta}) \subseteq \gamma \circ F^{\#}(X^{\#\delta})$ so $X^{\delta+1} \subseteq \gamma(X^{\#\delta+1})$. For transfinite ordinals, $\forall X^{\beta} \subseteq \gamma(X^{\#\beta})$ for all $\beta < \lambda$, then $X^{\lambda} = \bigcup_{\beta<\lambda} X^{\beta} \subseteq \bigcup_{\beta<\lambda} \gamma(X^{\#\beta}) \subseteq \gamma(\bigsqcup_{\beta<\lambda} X^{\#\beta})$ since γ is monotonic whence $X^{\lambda} \subseteq \gamma(X^{\#\lambda})$. So $\mathrm{lfp}\, F = X^{\varepsilon} = X^{\max(\varepsilon,\varepsilon')} \subseteq X^{\#\max(\varepsilon,\varepsilon')} = X^{\#\varepsilon'} = \mathrm{lfp}^{\#} F$. We conclude that

$$\mathcal{B}com[\![\text{while } Bexp \text{ do } LCo \text{ od}]\!]\, \gamma(R)$$

$$= \mathrm{lfp}^{\subseteq}_{\emptyset}\, \lambda X.\, \mathcal{C}bexp[\![T(\neg Bexp)]\!]\, \gamma(R) \cup \mathcal{C}bexp[\![Bexp]\!](\mathcal{B}com[\![LCo]\!]\, X)$$

$$\subseteq \gamma\Big(\mathrm{lfp}^{\sqsubseteq}_{\perp}\, \lambda X.\, \mathcal{C}bexp^{\#}[\![T(\neg Bexp)]\!]\, R \sqcup \mathcal{C}bexp^{\#}[\![Bexp]\!](\mathcal{A}bcom[\![LCo]\!]\, X)\Big).$$

– We have

$$\alpha \circ \mathcal{B}com[\![Com\, ;\, LCo]\!] \circ \gamma$$

$$= \alpha \circ \mathcal{B}com[\![Com]\!] \circ \mathcal{B}com[\![LCo]\!] \circ \gamma$$

$\sqsubseteq$ { $\gamma \circ \alpha$ is extensive, α and $\mathcal{B}com[\![Com]\!]$ are monotonic }

$$\alpha \circ \mathcal{B}com[\![Com]\!] \circ \gamma \circ \alpha \circ \mathcal{B}com[\![LCo]\!] \circ \gamma$$

$\sqsubseteq$ { Induction hypothesis and monotony }

$\alpha \circ \mathrm{Bcom}[\![\mathrm{Com}]\!] \circ \gamma \circ \mathrm{Abcom}[\![\mathrm{LCo}]\!]$

$\sqsubseteq$ { induction hypothesis }

$\mathrm{Abcom}[\![\mathrm{Com}]\!] \circ \mathrm{Abcom}[\![\mathrm{LCo}]\!]$.

In the case of a γ-based abstraction, we have similarly,

$\mathrm{Bcom}[\![\mathrm{Com} ; \mathrm{LCo}]\!](\gamma(X))$

$= \mathrm{Bcom}[\![\mathrm{Com}]\!](\mathrm{Bcom}[\![\mathrm{LCo}]\!](\gamma(X)))$

$\sqsubseteq$ { monotony and ind. hyp. }

$\mathrm{Bcom}[\![\mathrm{Com}]\!](\gamma(\mathrm{Abcom}[\![\mathrm{CLo}]\!] X))$

$\sqsubseteq$ { ind. hyp. }

$\gamma(\mathrm{Abcom}[\![\mathrm{Com}]\!] \circ \mathrm{Abcom}[\![\mathrm{CLo}]\!](X))$

$= \gamma(\mathrm{Abcom}[\![\mathrm{Com} ; \mathrm{CLo}]\!] X)$.

- Finally $\alpha \circ \mathrm{Bcom}[\![\mathrm{LCo} ;;]\!] \circ \gamma = \alpha \circ \mathrm{Bcom}[\![\mathrm{LCo}]\!] \circ \gamma \sqsubseteq \mathrm{Abcom}[\![\mathrm{LCo}]\!] = \mathrm{Abcom}[\![\mathrm{LCo} ;;]\!]$. Similarly, $\mathrm{Bcom}[\![\mathrm{LCo} ;;]\!](\gamma(X)) = \mathrm{Bcom}[\![\mathrm{LCo}]\!]\gamma(X) \sqsubseteq \gamma(\mathrm{Abcom}[\![\mathrm{LCo}]\!]X)$ by induction hypothesis $= \gamma(\mathrm{Abcom}[\![\mathrm{LCo} ;;]\!]X)$.

$\square$

$\square$

- Note 1: The proof of the γ-based abstraction implies that of a Galois connection-based abstraction since $F \circ \gamma \sqsubseteq \gamma \circ F^\#$ implies $\alpha \circ F \circ \gamma \sqsubseteq \alpha \circ \gamma \circ F^\# \sqsubseteq F^\#$ (monotony, $\gamma \circ \alpha$ reductive)
- Note 2: We have never used the hypothesis that $\mathrm{Abcom}[\![B]\!]$ is monotonic so that the proof can be easily generalized to the widening/narrowing case.

Generic backward/precondition linear relational analyzer

Linear syntax

We just have to add a function `at` mapping linearized commands to their entry point:

```
(* linear_Syntax.mli *)
open Abstract_Syntax
(* A linear arithmetic expression a1.x1+...+an.xn+b, where n is the *)
(* number of program variables, is represented by a vector:         *)
(* LINEAR_AEXP a1 ... an b. A non-linear arithmetic expression is   *)
(* represented by RANDOM_AEXP.                                      *)
type laexp =
  | RANDOM_AEXP              (* random expression                   *)
  | LINEAR_AEXP of int array (* linear expression                   *)
and lbexp =
  | LTRUE | LFALSE     (* constant boolean expression               *)
  | RANDOM_BEXP        (* random boolean expression                 *)
  | LAND of lbexp list (* boolean conjunction                       *)
```



```
  | LOR  of lbexp list (* boolean disjunction                       *)
  | LGE  of int array  (* LGE a1 ... an b is a1.x1+...+an.xn >= b   *)
  | LEQ  of int array  (* LGE a1 ... an b is a1.x1+...+an.xn = b    *)
and label = Abstract_Syntax.label
and lcom =
  | LSKIP of label * label
  | LASSIGN of label * variable * laexp * label
  | LSEQ of label * (lcom list) * label
  | LIF of label * lbexp * lbexp * lcom * lcom * label
  | LWHILE of label * lbexp * lbexp * lcom * label
val at : lcom -> label            (* command entry label            *)
val after : lcom -> label         (* command exit label             *)
val incom : label -> lcom -> bool (* label in command               *)

(* linear_Syntax.mli *)
open Abstract_Syntax
(* A linear arithmetic expression a1.x1+...+an.xn+b, where n is the *)
(* number of program variables, is represented by a vector:         *)
```



```
(* LINEAR_AEXP a1 ... an b. A non-linear arithmetic expression is   *)
(* represented by RANDOM_AEXP.                                      *)
type laexp =
  | RANDOM_AEXP              (* random expression                   *)
  | LINEAR_AEXP of int array (* linear expression                   *)
and lbexp =
  | LTRUE | LFALSE     (* constant boolean expression               *)
  | RANDOM_BEXP        (* random boolean expression                 *)
  | LAND of lbexp list (* boolean conjunction                       *)
  | LOR  of lbexp list (* boolean disjunction                       *)
  | LGE  of int array  (* LGE a1 ... an b is a1.x1+...+an.xn >= b   *)
  | LEQ  of int array  (* LGE a1 ... an b is a1.x1+...+an.xn = b    *)
and label = Abstract_Syntax.label
and lcom =
  | LSKIP of label * label
  | LASSIGN of label * variable * laexp * label
  | LSEQ of label * (lcom list) * label
  | LIF of label * lbexp * lbexp * lcom * lcom * label
```



```
  | LWHILE of label * lbexp * lbexp * lcom * label
(* linearized command entry label *)
let at c = match c with
  | LSKIP (l,m)             -> l
  | LASSIGN (l,x,a,m)       -> l
  | LSEQ (l,s,m)            -> l
  | LIF (l,b,nb,st,sf,m)    -> l
  | LWHILE (l,b,nb,s,m)     -> l
(* linearized command exit label *)
let after c = match c with
  | LSKIP (l,m)             -> m
  | LASSIGN (l,x,a,m)       -> m
  | LSEQ (l,s,m)            -> m
  | LIF (l,b,nb,st,sf,m)    -> m
  | LWHILE (l,b,nb,s,m)     -> m
(* label in command     *)
let rec incom l c = match c with
  | LSKIP (l1,l2)           -> (l=l1) or (l=l2)
```

```
  | LASSIGN (l1,x,a,l2)    -> (l=l1) or (l=l2)
  | LSEQ (l1,s,l2)         -> (l=l1) or (l=l2) or (inseq l s)
  | LIF (l1,b,nb,st,sf,l2) -> (l=l1) or (l=l2) or (incom l st)
                                              or (incom l sf)
  | LWHILE (l1,b,nb,s,l2)  -> (l=l1) or (l=l2) or (incom l s)
and inseq l s = match s with
  | [] -> false
  | h::t -> (incom l h) or (inseq l t)
```


backward relational abstract interpretation of commands

```
(* bcom.mli *)
open Linear_Syntax
open Aenv
(* backward abstract interpretation of commands *)
val bcom : lcom -> Aenv.t -> label -> Aenv.t

(* bcom.ml *)
open Linear_Syntax
open Aenv
open Abexp
open Fixpoint
(* backward abstract semantics of commands *)
exception Error of string
```



```
let rec bcom c r l =
  match c with
  | (LSKIP (l', l'')) ->
      if (l = l') then r
      else if (l = l'') then r
      else (raise (Error "SKIP incoherence"))
  | (LASSIGN (l',x,a,l'')) ->
      if (l = l') then b_ASSIGN x a r
      else if (l = l'') then r
      else (raise (Error "ASSIGN incoherence"))
  | (LSEQ (l', s, l'')) ->
      (bcomseq s r l)
  | (LIF (l', b, nb, t, f, l'')) ->
      (if (l = l') then
         (join (a_bexp b (bcom t r (at t))) (a_bexp nb (bcom f r (at f))
      else if (incom l t) then
         (bcom t r l)
      else if (incom l f) then
```



```
         (bcom f r l)
      else if (l = l'') then
         r
      else (raise (Error "IF incoherence")))
  | (LWHILE (l', b, nb, c', l'')) ->
    let f x = (join (a_bexp b (bcom c' x (at c'))) (a_bexp nb r))
    in let i = lfp (bot ()) leq widen narrow f in
      (if (l = l') then i
       else if (incom l c') then (bcom c' i l)
       else if (l = l'') then r
       else (raise (Error "WHILE incoherence")))
and bcomseq s r l = match s with
  | []   -> raise (Error "empty SEQ incoherence")
  | [c]  -> if (incom l c) then (bcom c r l)
            else (raise (Error "SEQ incoherence"))
  | h::t -> if (incom l h) then (bcom h (bcomseq t r (at (List.hd t))) l
            else (bcomseq t r l)
```

The generic backward/precondition linear relational static analyzer

```
(* main.ml = main-bw.ml *)
open Program_To_Abstract_Syntax
open Labels
open Pretty_Print
open Lpretty_Print
open Abstract_To_Linear_Syntax
open Linear_Syntax
open Aenv
open Acom
open Bcom
let _ =
  let arg = if (Array.length  Sys.argv) = 1 then ""
            else Sys.argv.(1) in
```



```
     Random.self_init ();
    let p = (abstract_syntax_of_program arg) in
      (print_string "** Program:\n";
       pretty_print p;
       let p' = (linearize_com p) in
         print_string "** Linearized program:\n";
         lpretty_print p';
         init ();
         print_string "** Postcondition:\n";
         print (top ());
         print_string "** Precondition:\n";
         print (bcom p' (top ()) (at p'));
         quit ())
```


Polyhedral backward/precondition analysis

The polyhedral abstract environment for backward static analysis

– We just have to add the definition of `b_ASSIGN` to handle backward assignments:

```
(* aenv.mli *)
open Linear_Syntax
open Array
open Variables
(* set of environments *)
type t
(* relational library initialization *)
val init : unit -> unit
(* relational library exit *)
val quit : unit -> unit
(* infimum *)
```

```
val bot : unit -> t
(* check for infimum *)
val is_bot : t -> bool
(* uninitialization *)
val initerr : unit -> t
(* supremum *)
val top : unit -> t
(* least upper bound *)
val join : t -> t -> t
(* greatest lower bound *)
val meet : t -> t -> t
(* approximation ordering *)
val leq : t -> t -> bool
(* equality *)
val eq : t -> t -> bool
(* printing *)
val print : t -> unit
(* forward collecting semantics of assignment                   *)
```



```
(* f_ASSIGN x f r =  {e[x <- i] | e in r /\ i in f({e}) cap I } *)
val f_ASSIGN : variable -> laexp -> t -> t
(* backward collecting semantics of assignment                       *)
(* b_ASSIGN x f r =  {e | exists i in f({e}) cap I: e[x <- i] in r } *)
val b_ASSIGN : variable -> laexp -> t -> t
(* collecting semantics of boolean expressions                  *)
(* f_LGE a r = {e in r | a0.v0+...+an-1.vn-1 >= an             *)
val f_LGE : (int array) -> t -> t
(* f_LEQ a r = {e in r | a0.v0+...+an-1.vn-1 = an              *)
val f_LEQ : (int array) -> t -> t
(* convergence acceleration *)
(* widening *)
val widen : t -> t -> t
(* narrowing *)
val narrow : t -> t -> t

(* aenv.ml *)
open Linear_Syntax
```



```
open Variables
type lattice = BOT | TOP
type t =
  NULL of lattice  (* in absence of any variable, dimension = 0 *)
| POLY of Poly.t   (* must be of dimension > 0                  *)
exception PolyError of string
(* relational library initialization *)
let maxdims = 10000
let maxrows = 100
let init () = (Polka.initialize false maxdims maxrows;
               Polka.strict := false)
(* relational library exit (* print statistics *) *)
let quit () = Polka.finalize ()
(* infimum *)
let bot () = match (number_of_variables ()) with
 | 0 -> NULL BOT
 | n -> if (n < 0) then
     raise (PolyError "negative number of variables (bot)")
```



```
   else if (n > maxdims) then
     raise (PolyError "too many variables (bot)")
   else
     POLY (Poly.empty n) (* 1 <= n <= polka_maxcolumns-polka_dec *)
(* check for infimum *)
let is_bot r = match r with
 | NULL BOT -> true
 | NULL TOP -> false
 | POLY p -> (Poly.is_equal p (Poly.empty (number_of_variables ())))
(* uninitialization *)
let initerr () = match (number_of_variables ()) with
 | 0 -> NULL TOP
 | n -> if (n < 0) then
     raise (PolyError "negative number of variables (initerr)")
   else if (n > maxdims) then
     raise (PolyError "too many variables (initerr)")
   else
     POLY (Poly.universe n)
```

```
(* supremum *)
let top () =  match (number_of_variables ()) with
 | 0 -> NULL TOP
 | n -> if (n < 0) then
      raise (PolyError "negative number of variables (top)")
    else if (n > maxdims) then
      raise (PolyError "too many variables (top)")
    else
      POLY (Poly.universe n)
(* least upper bound *)
let ljoin l1 l2 = match (l1, l2) with
 | BOT, _ -> l2
 | _, BOT -> l1
 | _, _   -> TOP
let join r1 r2 = match (r1, r2) with
 | NULL l1, NULL l2 -> (NULL (ljoin l1 l2))
 | POLY p1, POLY p2 -> (POLY (Poly.union p1 p2))
 | _, _ -> raise (PolyError "join")
```



```
(* greatest lower bound *)
let lmeet l1 l2 = match (l1, l2) with
 | TOP, _ -> l2
 | _, TOP -> l1
 | _, _   -> BOT
let meet r1 r2 = match (r1, r2) with
 | NULL l1, NULL l2 -> (NULL (lmeet l1 l2))
 | POLY p1, POLY p2 -> (POLY (Poly.inter p1 p2))
 | _, _ -> raise (PolyError "meet")
(* approximation ordering *)
let lleq l1 l2 = match (l1, l2) with
 | BOT, _   -> true
 | _, TOP   -> true
 | TOP, BOT -> false
let leq r1 r2 = match (r1, r2) with
 | NULL l1, NULL l2 -> (lleq l1 l2)
 | POLY p1, POLY p2 -> (Poly.is_included_in p1 p2)
 | _, _ -> raise (PolyError "leq")
```



```
(* equality *)
let eq r1 r2 = match (r1, r2) with
 | NULL l1, NULL l2 -> (l1 = l2)
 | POLY p1, POLY p2 -> (Poly.is_equal p1 p2)
 | _, _ -> raise (PolyError "eq")
(* printing *)
let print r = match r with
 | NULL BOT -> (print_string "{ _|_ }\n")
 | NULL TOP -> (print_string "{ T }\n")
 | POLY p ->
   (Poly.minimize p; (* to get the constraints and generators of p *)
    Poly.print_constraints string_of_variable Format.std_formatter p;
    Format.pp_print_newline Format.std_formatter ())
(* convert a0.v0+...+an-1.vn-1+an where n = (number_of_variables ()) int
(* vector [1,an,a0,...,an-1].
let vector_of_lin_expr a =
  let v = Vector.make ((number_of_variables ()) + 2) in
    (Vector.set v 0 1;
```



```
     Vector.set v 1 (a.(number_of_variables ()));
     for i = 0 to ((number_of_variables ()) - 1) do
       Vector.set v (i+2) a.(i)
     done;
     (*
     Vector._print v;
     Vector.print_constraint string_of_variable Format.std_formatter v;
     Format.pp_print_newline Format.std_formatter ();
     *)
     v)
(* f_ASSIGN x f r =  {e[x <- i] | e in r /\ i in f({e}) cap I } *)
let f_ASSIGN x f r =
  match r with
 | NULL _ -> r
 | POLY p -> (match f with
 | RANDOM_AEXP ->
    let d = [|{ Polka.pos = x; Polka.nbdims = 1 }|] in
      (POLY (Poly.add_dims_and_embed_multi (Poly.del_dims_multi p d) d))
```

```
 | LINEAR_AEXP a ->
      (POLY (Poly.assign_var p x (vector_of_lin_expr a))))
(* b_ASSIGN x f r =  {e | exists i in f({e}) cap I: e[x <- i] in r } *)
let b_ASSIGN x f r =
  match r with
 | NULL _ -> r
 | POLY p -> POLY (match f with
 | RANDOM_AEXP ->
     let d = [|{ Polka.pos = x; Polka.nbdims = 1 }|] in
        (Poly.add_dims_and_embed_multi (Poly.del_dims_multi p d) d)
 | LINEAR_AEXP a ->
        (Poly.minimize p; (* <- to get around a bug of Polka 2.0.2   *)
                          (* in substitute_var                        *)
         Poly.substitute_var p x (vector_of_lin_expr a)))
(* f_LGE a r = {e in r | a0.v0+...+an-1.vn-1+an >= 0} *)
let f_LGE a r =
  match r with
 | NULL _ -> r
```



```
 | POLY p -> POLY (Poly.add_constraint p (vector_of_lin_expr a))
(* f_LEQ a r = {e in r | a0.v0+...+an-1.vn-1+an = 0} *)
let minus = Array.map (fun x -> (- x))
let f_LEQ a r = meet (f_LGE a r) (f_LGE (minus a) r)
(* widening *)
let widen r1 r2 = match (r1, r2) with
 | NULL l1, NULL l2 -> (NULL (ljoin l1 l2))
 | POLY p1, POLY p2 -> (POLY (Poly.widening p1 p2))
 | _, _ -> raise (PolyError "widen")
(* narrowing *)
(* let narrow a b = a (* does not ensure termination *) *)
let narrow a b = b (* less precise but does ensure termination *)
```


Examples of backward polyhedral static analysis

```
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out
x := 1;
while (x > 0) do
  x := x + 10000000
od;;
...
** Postcondition:
{1>=0}
** Precondition:
empty(1)
```

- The only way for the program execution to exit the loop is to *never start it* (because of arithmetic overflow when $\mathbb{I}$ is unbounded or non-termination if $\mathbb{N} \subseteq \mathbb{I}$)

The precondition/postcondition collecting semantics

Definition of the forward/backward semantics

- We assume that a transition system $\langle \Sigma,\ \tau,\ I,\ F\rangle$ is given such that
  - Σ is a set of states
  - $\tau \in \wp(\Sigma \times \Sigma)$ is the transition relation between a state and its potential successors
  - $I \subseteq \Sigma$ is the set of initial states
  - $F \subseteq \Sigma$ is the set of final states

- The forward or postcondition semantics is the set of states which are reachable from the initial states I

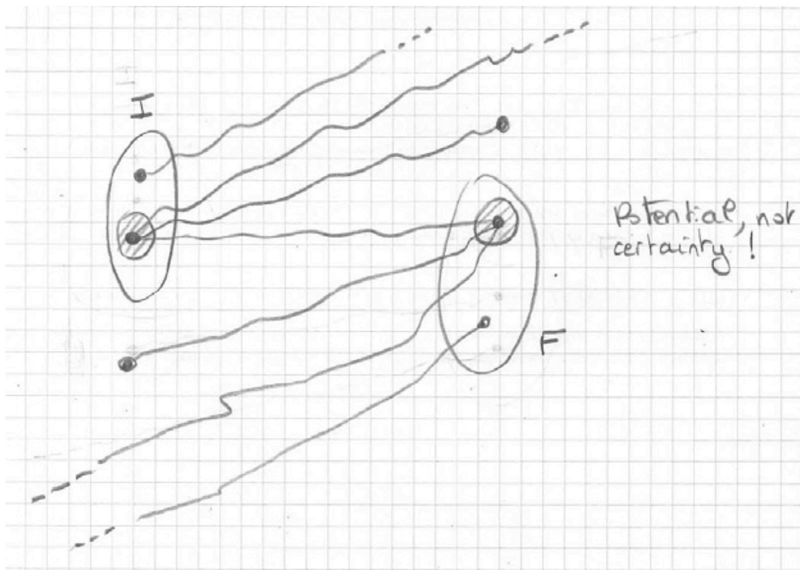
$$
\begin{aligned}
&\operatorname{post}[\tau^\star]I\\
&= \{s' \in \Sigma \mid \exists s \in I : \tau^\star(s, s')\}\\
&= \mathsf{lfp}^{\subseteq}_{\emptyset}\ \mathcal{F}
\end{aligned}
$$

$$
\begin{aligned}
\text{where}\quad & \mathcal{F}(X) = \ = I \cup \operatorname{post}[t]X\\
\text{and}\quad & \operatorname{post}[t]X = \{s' \in \Sigma \mid \exists s \in X : t(s, s')\}
\end{aligned}
$$

- The backward or precondition semantics is the set of states from which the final states F may be potentially reached

$$
\begin{aligned}
&\operatorname{pre}[\tau^\star]F\\
&= \{s \in \Sigma \mid \exists s' \in F : \tau^\star(s, s')\}\\
&= \mathsf{lfp}^{\subseteq}_{\emptyset}\ \mathcal{B}
\end{aligned}
$$

$$
\begin{aligned}
\text{where}\quad & \mathcal{B}(X) = F \cup \operatorname{pre}[t]X\\
\text{and}\quad & \operatorname{pre}[t]X = \{s \in \Sigma \mid \exists s' \in X : t(s, s')\}
\end{aligned}
$$



– The forward/backward or precondition/postcondition semantics is the pair

$$\langle I \cap \mathrm{pre}[\tau^\star]F,\ \mathrm{post}[\tau^\star]I \cap F\rangle$$

– In the concrete, there is nothing new

– However, in the abstract, we can do much better than separate overapproximations of $\mathrm{pre}[\tau^\star]F$ and $\mathrm{post}[\tau^\star]I$

Example of abstract forward/backward semantics

```
Generic-FW-BW-REL-Abstract-Interpreter % cat ../Examples/example48.sil
% example48.sil %
X := X + Y + 3;
while (X + Y <> 5)  do
  skip
od;;
Generic-FW-REL-Abstract-Interpreter % make pol
Polyhedral analysis
Generic-FW-REL-Abstract-Interpreter % ./a.out ../Examples/example48.sil
...
** Precondition:
{1>=0}
** Postcondition:
{X+Y=5}
```



```
Generic-FW-BW-REL-Abstract-Interpreter % make bw
Backward analysis
Generic-FW-BW-REL-Abstract-Interpreter % make pol
Polyhedral analysis
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example48.sil
** Postcondition:
{1>=0}
** Precondition:
{X+2Y=2}
Generic-FW-BW-REL-Abstract-Interpreter % make iter
Iterated forward/backward analysis
Generic-FW-BW-REL-Abstract-Interpreter % make pol
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example48.sil
** Precondition:
{X+2Y=2}
** Postcondition:
{X+Y=5}
```


Properties of the forward/backward collecting semantics

THEOREM. $\langle I \cap \mathrm{pre}[\tau^\star]F,\ F \cap \mathrm{post}[\tau^\star]I\rangle = \mathsf{gfp}^{\subseteq_2}\ \lambda\langle X,\ Y\rangle \cdot \langle I,\ F\rangle \cap_2 \langle X \cap \mathrm{pre}[\tau^\star]Y,\ Y \cap \mathrm{post}[\tau^\star]X\rangle$ ∎

PROOF. We prove that $\langle I \cap \mathrm{pre}[\tau^\star]F,\ F \cap \mathrm{post}[\tau^\star]I\rangle$ is a fixpoint of

$$\lambda\langle X,\ Y\rangle \cdot \langle I,\ F\rangle \cap_2 \langle X \cap \mathrm{pre}[\tau^\star]Y,\ Y \cap \mathrm{post}[\tau^\star]X\rangle$$

componentwise. For the first component, we have

$$\begin{aligned}
& I \cap (I \cap \mathrm{pre}[\tau^\star]F) \cap \mathrm{pre}[\tau^\star](F \cap \mathrm{post}[\tau^\star]I) \\
& \quad \text{(def. pre)} \\
= \ & I \cap \mathrm{pre}[\tau^\star]F \cap \{s \mid \exists s_2 : \tau^\star(s, s_2) \wedge s_2 \in F \wedge s_2 \in \mathrm{post}[\tau^\star]I\} \\
& \quad \text{(def. post)} \\
= \ & I \cap \mathrm{pre}[\tau^\star]F \cap \{s \mid \exists s_2 : \tau^\star(s, s_2) \wedge s_2 \in F \wedge \exists s_3 : s_3 \in I \wedge \tau^\star(s_3, s_2)\}
\end{aligned}$$

$\wr$def. pre$\wr$

$= I \cap \{s \mid \exists s_1 : s_1 \in F \wedge \tau^\star(s, s_A) \wedge \exists s_2 : \tau^\star(s, s_2) \wedge s_2 \in F \wedge \exists s_3 : s_3 \in I \wedge \tau^\star(s_3, s_2)\}$

$\wr$simplifying with $s_1 = s_2$ and $s_3 = s$$\wr$

$= I \cap \{s \mid \exists s_2 : \tau^\star(s, s_2) \wedge s_2 \in F\}$

$\wr$def. pre$\wr$

$= I \cap \mathrm{pre}[\tau^\star]F$

for I, F and τ.

To show that $\langle I \cap \mathrm{pre}[\tau^\star]F,\ F \cap \mathrm{post}[\tau^\star]I\rangle$ is the greatest fixpoint of

$$\lambda\langle X, Y\rangle \cdot \langle I, F\rangle \cap_2 \langle X \cap \mathrm{pre}[\tau^\star]Y,\ Y \cap \mathrm{post}[\tau^\star]X\rangle$$

we let $\langle X, Y\rangle$ be any such fixpoint. We have: $X = I \cap X \cap \mathrm{pre}[\tau^\star]Y$ so $X \subseteq I \cap \mathrm{pre}[\tau^\star]Y$. Moreover $Y = F \cap Y \cap \mathrm{post}[\tau^\star]X$ implies $Y \subseteq F$ so by monotony $X \subseteq I \cap \mathrm{pre}[\tau^\star]F$. Similarly, $Y \subseteq F \cap \mathrm{post}[\tau^\star]F$. $\square$

The precondition/postcondition abstract semantics and its implementation

Pre/postcondition abstract semantics

– We are given an abstraction $\langle \wp(\Sigma), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle L, \sqsubseteq\rangle$

– We want to compute an overapproximation of:

$$\begin{aligned} &\langle \alpha(I \cap \mathrm{pre}[\tau^\star]F),\ \alpha(F \cap \mathrm{post}[\tau^\star]I)\rangle \\ =\ &\langle \alpha(I \cap \mathsf{lfp}\,\lambda Z \cdot F \cup \mathrm{pre}[\tau]Z),\ \alpha(F \cap \mathsf{lfp}\,\lambda Z \cdot I \cup \mathrm{post}[\tau]Z)\rangle \end{aligned}$$

– One solution is to overapproximate

$$\begin{aligned} \langle &\alpha(I) \sqcap \mathsf{lfp}\,\lambda Z \cdot \alpha(F) \sqcup \alpha(\mathrm{pre}[\tau](\gamma(Z)), \\ &\alpha(F) \sqcap \mathsf{lfp}\,\lambda Z \cdot \alpha(I) \sqcup \alpha(\mathrm{post}[\tau]\gamma(Z))\rangle \end{aligned}$$

– A better solution is to use the greatest fixpoint characterization and to overapproximate:

$$\begin{aligned} &\alpha_2(\mathsf{gfp}^{\subseteq_2}\,\lambda\langle X, Y\rangle \cdot \langle I, F\rangle \cap_2 \\ &\quad\langle X \cap \mathrm{pre}[\tau^\star]Y,\ Y \cap \mathrm{post}[\tau^\star]X\rangle) \\ &\quad\wr\text{fixpoint overapproximation}\wr \\ \sqsubseteq_2\ &\mathsf{gfp}^{\sqsubseteq_2}\,\lambda\langle X, Y\rangle \cdot \langle \alpha(I), \alpha(F)\rangle \sqcap_2 \\ &\quad\langle \alpha(\gamma(X) \cap \mathrm{pre}[\tau^\star]\gamma(Y)),\ \alpha(\gamma(Y) \cap \mathrm{post}[\tau^\star]\gamma(X))\rangle \\ &\quad\wr\text{fixpoint definition of } \mathrm{pre}[\tau^\star] \text{ and } \mathrm{post}[\tau^\star]\wr \\ =\ &\mathsf{gfp}^{\sqsubseteq_2}\,\lambda\langle X, Y\rangle \cdot \langle \alpha(I), \alpha(F)\rangle \sqcap_2 \\ &\quad\langle \alpha(\gamma(X) \cap \mathsf{lfp}\,\lambda Z \cdot \gamma(Y) \cup \mathrm{pre}[\tau]Z), \\ &\quad\ \alpha(\gamma(Y) \cap \mathsf{lfp}\,\lambda Z \cdot \gamma(X) \cup \mathrm{post}[\tau]Z)\rangle \end{aligned}$$

$\wr \alpha$ monotone $\wr$

$$\sqsubseteq_2 \ \mathsf{gfp}^{\sqsubseteq_2}\ \lambda\langle X,\ Y\rangle \cdot \langle \alpha(I),\ \alpha(F)\rangle \sqcap_2 \langle \alpha(\gamma(X)) \sqcap \mathsf{lfp}\ \lambda Z \cdot \alpha(\gamma(Y)) \sqcup \alpha(\mathrm{pre}[\tau](\gamma(Z))), \alpha(\gamma(Y)) \sqcap \lambda Z \cdot \alpha(\gamma(X)) \sqcup \alpha(\mathrm{post}[\tau]\gamma(Z))\rangle$$

$\wr \alpha \circ \gamma$ reductive and monotony $\wr$

$$\sqsubseteq_2 \ \mathsf{gfp}^{\sqsubseteq_2}\ \lambda\langle X,\ Y\rangle \cdot \langle \alpha(I),\ \alpha(F)\rangle \sqcap_2 \langle X \sqcap \mathsf{lfp}\ \lambda Z \cdot Y \sqcup \alpha(\mathrm{pre}[\tau](\gamma(Z))), Y \sqcap \lambda Z \cdot X \sqcup \alpha(\mathrm{post}[\tau]\gamma(Z))\rangle$$

Pre/postcondition static analysis

Assuming $\alpha(I) \sqsubseteq I^\sharp$, $\alpha(F) \sqsubseteq F^\sharp$, $\alpha \circ \mathrm{pre}[\tau] \circ \gamma \mathrel{\dot\sqsubseteq} \mathcal{B}^\sharp$, $\alpha \circ \mathrm{post}[\tau] \circ \gamma \mathrel{\dot\sqsubseteq} \mathcal{F}^\sharp$, and using chaotic iterations, the algorithm is

$$\begin{array}{rcl} X^0 & = & I^\sharp \\ Y^0 & = & F^\sharp \\ \ldots & \ldots & \ldots \\ X^{n+1} & = & X^n \sqcap \mathsf{lfp}\ \lambda Z \cdot Y^n \sqcup \mathcal{B}^\sharp(Z) \\ Y^{n+1} & = & Y^n \sqcap \mathsf{lfp}\ \lambda Z \cdot X^{n+1} \sqcup \mathcal{F}^\sharp(Z) \\ \ldots & \ldots & \ldots \end{array}$$

Convergence may have to be enforced via a narrowing.

Implementation of the pre/postcondition static analysis

In the implementation the initial precondition is that variables are uninitializated and the postcondition is $\mathtt{tt}$, that is $\top$ is both cases.

```
(* main.ml = main-iter-fw-bw.ml *)
open Program_To_Abstract_Syntax
open Labels
open Pretty_Print
open Lpretty_Print
open Abstract_To_Linear_Syntax
open Linear_Syntax
open Aenv
open Acom
open Bcom
```



```
let _ =
 let narrowing_limit = 100 in
  let arg = if (Array.length  Sys.argv) = 1 then ""
            else Sys.argv.(1) in
     Random.self_init ();
    let p = (abstract_syntax_of_program arg) in
      (print_string "** Program:\n";
       pretty_print p;
       let p' = (linearize_com p) in
         print_string "** Linearized program:\n";
         lpretty_print p';
         init ();
         let rec iterate pre n =
           (print_string "** Precondition:\n";
            print pre;
            let post = (acom p' pre (after p')) in
              (print_string "** Postcondition:\n";
               print post;
```

```
          let pre' = (bcom p' post (at p')) in
            (if (Aenv.eq pre pre') then
               (print_string "stable precondition after ";
                print_int n;print_string " iteration(s).\n")
             else if (n < narrowing_limit) then
               (print_string "unstable precondition after ";
                print_int n;print_string " iteration(s).\n";
                iterate pre' (n+1))
             else
               (print_string "stable stopped "; print_int n;
                print_string " iterations (narrowing).\n"))))
    in iterate (initerr ()) 1;
    quit ())
```


Examples of pre/postcondition static analysis

In the examples, we enforce a postcondition B as:

while $\neg B$ do skip od

which, since termination is enforced by the backward analysis, requires B to hold just before the loop.

```
Generic-FW-BW-REL-Abstract-Interpreter % make iter
Iterated forward/backward analysis
Generic-FW-BW-REL-Abstract-Interpreter % make pol
...
Polyhedral analysis
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example49.sil
```



```
** Program:
0:
  x := y;
1:
  z := ((2 * x) + 1);
2:
  while ((y < 0) | (0 < y)) do
    3:
      skip
    4:
  od {(y = 0)}
5:
```



```
** Linearized program:
0:
  x := 1.y + 0.x + 0.z + 0;
1:
  z := 0.y + 2.x + 0.z + 1;
2:
  while (-1.y + 0.x + 0.z + -1 >= 0 | 1.y + 0.x + 0.z + -1 >= 0) do
    3:
      skip
    4:
  od {-1.y + 0.x + 0.z + 0 = 0}
5:
```

```
** Precondition:
{1>=0}
** Postcondition:
{z=1,x=0,y=0}
unstable precondition after 1 iteration(s).
** Precondition:
{y=0}
** Postcondition:
{z=1,x=0,y=0}
stable precondition after 2 iteration(s).
Generic-FW-BW-REL-Abstract-Interpreter %
```

Comments on this example:

– The initial precondition {1>=0} is tt whence states no hypothesis on the input data

– The forward analysis provides the first postcondition stating that on termination y=0 whence z=1 and x=0

– Since y is not modified in the program, the next backward analysis provides the second precondition that we must have y=0 initially

– The last forward iteration shows that the fixpoint is reached

The following program is proved to never terminate:

```
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out
x := 1;
while (x > 0) do
  x := x + 1000000
od;;
** Precondition:
{1>=0}
** Postcondition:
empty(1)
unstable precondition after 1 iteration(s).
** Precondition:
empty(1)
** Postcondition:
empty(1)
stable precondition after 2 iteration(s).
```



```
Generic-FW-BW-REL-Abstract-Interpreter % cat ../Examples/example45.sil
% example45.sil %
X := X + Y;
while (X + Y <> 0)  do
  skip
od;;
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example45.sil
** Precondition:
{1>=0}
** Postcondition:
{X+Y=0}
unstable precondition after 1 iteration(s).
** Precondition:
{X+2Y=0}
** Postcondition:
{X+Y=0}
stable precondition after 2 iteration(s).
```

```
Generic-FW-BW-REL-Abstract-Interpreter % cat ../Examples/example46.sil
% example46.sil %
X := Y;
while (X + Y + Z <> 0)  do
  skip
od;;
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example46.sil
** Precondition:
{1>=0}
** Postcondition:
{2Y+Z=0,Y=X}
unstable precondition after 1 iteration(s).
** Precondition:
{2Y+Z=0}
** Postcondition:
{2Y+Z=0,Y=X}
stable precondition after 2 iteration(s).
```



```
Generic-FW-BW-REL-Abstract-Interpreter % cat ../Examples/example47.sil
% example47.sil %
X := Y;
while (X + Y + Z > 0)  do
  skip
od;;
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example47.sil
** Precondition:
{1>=0}
** Postcondition:
{Y=X,2Y+Z<=0}
unstable precondition after 1 iteration(s).
** Precondition:
{2Y+Z<=0}
** Postcondition:
{Y=X,2Y+Z<=0}
stable precondition after 2 iteration(s).
```

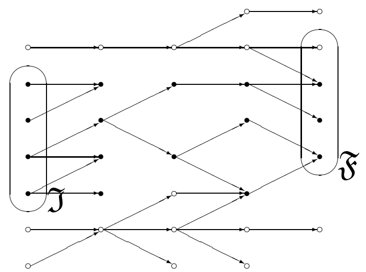

The forward/backward reachability collecting semantics

The forward reachability collecting semantics

– The forward reachability semantics

$$\text{post}[\tau^\star]I = \mathsf{lfp}^{\subseteq}\ \lambda X \cdot I \cup \text{post}[\tau]X$$

is the set of descendants of the initial states I by transitions τ



Recall the following fixpoint characterization of the forward reachability collecting semantics:

THEOREM. $\mathfrak{D} = \text{post}[\tau\star]\mathfrak{I} = \mathsf{lfp}\, F[\![\text{P}]\!]$ where $F[\![\text{P}]\!] \in \langle\wp(S),\, \cup\rangle \stackrel{\sqcup}{\longmapsto} \langle\wp(S),\, \cup\rangle$ is defined by $F[\![\text{P}]\!]X = \mathfrak{I} \cup \text{post}[\tau X$. ■

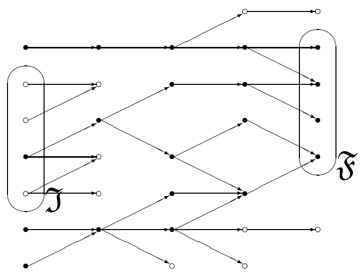
PROOF. Observe that $\wp(\mathfrak{S}\times\mathfrak{S})(\subseteq, \emptyset, \mathfrak{S}\times\mathfrak{S}, \cup, \cap)$ and $\wp(\mathfrak{S})(\subseteq, \emptyset, \mathfrak{S}, \cup, \cap)$ are complete lattices. Define $\alpha \in \wp(\mathfrak{S}\times\mathfrak{S}) \mapsto \wp(\mathfrak{S})$ by $\alpha(X) = \text{post}[X]\mathfrak{I}$. It is a complete $\cup$-morphism so that there exists γ such that $\wp(\mathfrak{S}\times\mathfrak{S})(\subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathfrak{S})(\subseteq)$. We have $\tau\star = \mathsf{lfp}\, T = \bigcup_{n\in\mathbb{N}} T^n(\emptyset)$ where $T(X) = 1 \cup X \circ \tau$, $\emptyset = \alpha(\emptyset)$ and $F[\![\text{P}]\!] \in \langle\wp(S),\, \cup\rangle \stackrel{\sqcup}{\longmapsto} \langle\wp(S),\, \cup\rangle$ is such that for all $X \in \wp(\mathfrak{S}\times\mathfrak{S})$, we have $\alpha \circ T(X) = \text{post}[T(X)]\mathfrak{I} = \{s \mid \exists s' \in \mathfrak{I} : \langle s',\, s\rangle \in T(X)\} = \{s \mid \exists s' \in \mathfrak{I} : \langle s',\, s\rangle \in 1 \cup X \circ \tau = \{s \mid \exists s' \in \mathfrak{I} : (\langle s',\, s\rangle \in 1) \vee (\langle s',\, s\rangle \in X \circ \tau\} = \{s \mid \exists s' \in \mathfrak{I} : (s' = s) \vee (\langle s',\, s\rangle \in X \circ \tau\} = \mathfrak{I} \cup \{s \mid \exists s' \in \mathfrak{I} : \langle s',\, s\rangle \in X \circ \tau = \mathfrak{I} \cup \{s \mid \exists s' \in \mathfrak{I} : \exists s'' \in \mathfrak{S} : \langle s',\, s''\rangle \in X \wedge s''\tau\} = \mathfrak{I} \cup \{s \mid \exists s'' \in \{s'' \mid \exists s' \in \mathfrak{I} : \langle s',\, s''\rangle \in X\} : s''\tau\} = \mathfrak{I} \cup \{s \mid \exists s'' \in \text{post}[X]\mathfrak{I} : s''\tau\} = \mathfrak{I} \cup \text{post}[\tau(\text{post}[X]\mathfrak{I}) = F[\![\text{P}]\!](\text{post}[X]\mathfrak{I}) = F[\![\text{P}]\!] \circ \alpha(X)$. By the fixpoint transfer theorem, we have $\alpha(\mathsf{lfp}\, T) = \mathsf{lfp}\, F[\![\text{P}]\!]$ so that $\mathfrak{D} = \text{post}[\tau\star]\mathfrak{I} = \alpha(\tau\star) = \alpha(\mathsf{lfp}\, T) = \mathsf{lfp}\, F[\![\text{P}]\!]$. □

The backward reachability collecting semantics

– The backward reachability semantics

$$\text{pre}[\tau^\star]F = \mathsf{lfp}^{\subseteq}\ \lambda Y \cdot F \cup \text{pre}[\tau]Y$$

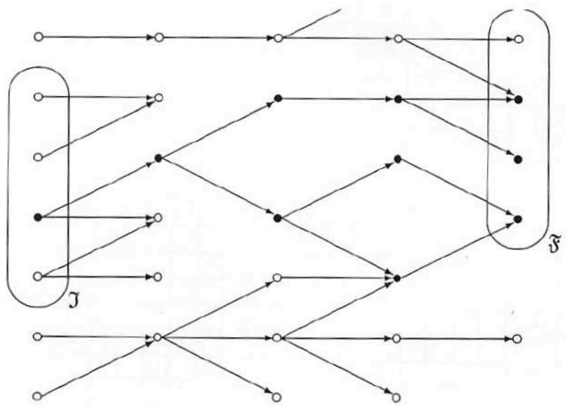
is the set of potential ascendants of the final states F by transitions τ



The forward/backward reachability collecting semantics

– We are interest in the forward/backward collecting semantics which is defined as $\text{post}[\tau^\star]I \cap \text{pre}[\tau^\star]F$ that is the intersection of two fixpoints $\mathsf{lfp}^{\subseteq}\ \lambda X \cdot I \cup \text{post}[\tau]X \cap \lambda Y \cdot F \cup \text{pre}[\tau]Y$

– We look for a more precise analysis than the mere intersection of the independent overapproximations of $\mathsf{lfp}^{\subseteq}\ \lambda X \cdot I \cup \text{post}[\tau]X$ and $\mathsf{lfp}^{\subseteq}\ \lambda Y \cdot F \cup \text{pre}[\tau]Y$



Descendant states (•) of the initial states (𝔍) which are ascendant states of the final states (𝔉).

Properties of the forward/backward collecting semantics

THEOREM. For all transition systems $\langle \Sigma, I, F, \tau\rangle$ where $\mathcal{F}(X) \stackrel{\text{def}}{=} I \cup \text{post}[\tau]X$, $\mathcal{B}(X) \stackrel{\text{def}}{=} I \cup \text{pre}[\tau]X$, and $X \subseteq \Sigma$, we have:

$$
\begin{aligned}
(\text{pre}[\tau]X) \cap \text{lfp}\,\mathcal{F} &\subseteq \text{pre}[\tau](X \cap \text{lfp}\,\mathcal{F}) &(16)\\
(\text{post}[\tau]X) \cap \text{lfp}\,B[\![\text{P}]\!] &\subseteq \text{post}[\tau](X \cap \text{lfp}\,\mathcal{B}) &(17)\\
\text{lfp}\,\mathcal{F} \cap \text{lfp}\,\mathcal{B} & &(18)\\
&= \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap \mathcal{B}(X)) &(19)\\
&= \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{B} \cap \mathcal{F}(X)) &(20)\\
&= \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap \text{lfp}\,\mathcal{B} \cap \mathcal{B}(X)) &(21)\\
&= \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap \text{lfp}\,\mathcal{B} \cap \mathcal{F}(X)) &(22)
\end{aligned}
$$

■

PROOF. — To prove (16), observe that the theorem on page 113 implies that $(\text{pre}[\tau]X) \cap \text{lfp}\,\mathcal{F} = \{s \mid \exists s' \in X : s\tau s'\} \cap \{s \mid \exists s'' \in I : s''\tau^\star s\} \subseteq \{s \mid \exists s' \in X : \exists s'' \in I : s''\tau^\star s' \wedge s\tau s'\}$ since $s''\tau^\star s$ and $s\tau s'$ imply $s''\tau^\star s'$. This is precisely $\text{pre}[\tau](X \cap \text{post}[\tau^\star]I) = \text{pre}[\tau](X \cap \text{lfp}\,\mathcal{F})$. The proof of (17) is similar to that of (16).

— To prove (19), let $X^n, n \in \mathbb{N}$ and $Y^n, n \in \mathbb{N}$ be the iteration sequences starting from the infimum $\emptyset$ for $\mathcal{B}$ and $\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap B[\![\,\text{P}]\!]X)$ respectively. We have $\text{lfp}\,\mathcal{F} \cap X^0 = \emptyset = Y^0$. Assume that $\text{lfp}\,\mathcal{F} \cap X^n \subseteq Y^n$ by induction hypothesis. Then $\text{lfp}\,\mathcal{F} \cap X^{n+1} = \text{lfp}\,\mathcal{F} \cap \mathcal{B}(X^n) = \text{lfp}\,\mathcal{F} \cap (\text{pre}[\tau]X^n \cup F) = \text{lfp}\,\mathcal{F} \cap \Big((\text{pre}[\tau]X^n \cap \text{lfp}\,\mathcal{F}) \cup F\Big)$, which, by (16), is included in $\text{lfp}\,\mathcal{F} \cap \Big(\text{pre}[\tau](X^n \cap \text{lfp}\,\mathcal{F}) \cup F\Big)$ which, by induction hypothesis and monotony, is included in $\text{lfp}\,\mathcal{F} \cap \big(\text{pre}[\tau](Y^n) \cup F\big) = \text{lfp}\,\mathcal{F} \cap \mathcal{B}Y^n = Y^{n+1}$. It follows that $\text{lfp}\,\mathcal{F} \cap \text{lfp}\,\mathcal{B} = (\cup_{n\in\mathbb{N}} X^n) \cap \text{lfp}\,\mathcal{F} = \cup_{n\in\mathbb{N}} (X^n \cap \text{lfp}\,\mathcal{F}) \subseteq \cup_{n\in\mathbb{N}} Y^n = \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap \mathcal{B}(X))$. But lfp is monotone so that $\text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F} \cap \mathcal{B}(X)) \subseteq \text{lfp}\,\lambda X \cdot (\text{lfp}\,\mathcal{F}) \cap \text{lfp}\,\lambda X \cdot (\mathcal{B}(X)) = \text{lfp}\,\mathcal{F} \cap \text{lfp}\,\mathcal{B}$. Equality follows by antisymmetry. The proofs of (20) to (22) are similar. □

The forward/backward reachability abstract semantics

Let us recall the following fixpoint approximation result:

THEOREM. If $P(\preceq, f, t, \wedge, \vee)$ is a complete lattice, F, $\bar{F} \in P(\preceq) \stackrel{m}{\longmapsto} P(\preceq)$, and $F \preceq \bar{F}$, then $\text{lfp}\,F \preceq \text{lfp}\,\bar{F}$. ■

PROOF. We have $F(\text{lfp}\,\bar{F}) \preceq \bar{F}(\text{lfp}\,\bar{F}) = \text{lfp}\,\bar{F}$ whence $\text{lfp}\,F \preceq \text{lfp}\,\bar{F}$ since $\text{lfp}\,F = \bigwedge\{X \mid F(X) \preceq X\}$ by Tarski's fixpoint theorem. □

as well as fixpoint abstraction:

THEOREM. If $P(\preceq, f, t, \wedge, \vee)$ and $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ are complete lattices, $P(\preceq) \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} P^\sharp(\preceq^\sharp)$ and $F \in P(\preceq) \stackrel{m}{\longmapsto} P(\preceq)$, then $\alpha(\text{lfp}\,F) \preceq^\sharp \text{lfp}\,\alpha \circ F \circ \gamma$. ■

Proof. By Tarski's fixpoint theorem, the least fixpoints exist. So let $p^\sharp = \mathsf{lfp}\, \alpha \circ F \circ \gamma$. We have $\alpha \circ F \circ \gamma(p^\sharp) = p^\sharp$ whence $F \circ \gamma(p^\sharp) \preceq \gamma(p^\sharp)$ by def. Galois connection. It follows that $\gamma(p^\sharp)$ is a postfixpoint of F whence $\mathsf{lfp}\, F \preceq \gamma(p^\sharp)$ by Tarski's fixpoint theorem or equivalently $\alpha(\mathsf{lfp}\, F) \preceq^\sharp p^\sharp = \mathsf{lfp}\, \alpha \circ F \circ \gamma$. □

- In order to approximate $\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B$ from above using abstract interpretations $F^\sharp$ of F and $B^\sharp$ of B, we can use the abstract upper approximation $\mathsf{lfp}\, F^\sharp \wedge^\sharp \mathsf{lfp}\, B^\sharp$.
- However, a better approximation suggested in [2] can be obtained as the limit of the decreasing chain

$$\begin{aligned}
\dot{X}^0 &= \mathsf{lfp}\, F^\sharp \\
\dot{X}^{2n+1} &= \mathsf{lfp}\, \lambda X \cdot \dot{X}^{2n} \wedge^\sharp F^\sharp(X) \\
\dot{X}^{2n+2} &= \mathsf{lfp}\, \lambda X \cdot \dot{X}^{2n+1} \wedge^\sharp B^\sharp(X)
\end{aligned}$$

for all $n \in \mathbb{N}$.

- Observe that by theorem on page **117** there is no improvement when considering the exact collecting semantics.
- However, when considering approximations of the collecting semantics, not all information can be collected in one pass and iteration can lead definite improvement

- If the abstract lattice does not satisfy the descending chain condition then [2] also suggests to use a narrowing operator Δ to enforce convergence of the downward iteration $\dot{X}^k$, $k \in \mathbb{N}$.
- The same way a widening/narrowing approach can be used to enforce convergence of the iterates for $\lambda X \cdot \dot{X}^{2n} \wedge^\sharp F^\sharp(X)$ and $\lambda X \cdot \dot{X}^{2n+1} \wedge^\sharp B^\sharp(X)$.
- The correctness of this approach follows from the following result on fixpoint meet approximation:

Theorem. If $P(\preceq, f, t, \wedge, \vee)$ and $P^\sharp(\preceq^\sharp, f^\sharp, t^\sharp, \wedge^\sharp, \vee^\sharp)$ are complete lattices, $P(\preceq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp(\preceq^\sharp)$, $F \in P(\preceq) \xrightarrow{m} P(\preceq)$ and $B \in P(\preceq) \xrightarrow{m} P(\preceq)$ satisfy hypotheses (21) and (22) of theorem , $F^\sharp \in P^\sharp(\preceq^\sharp) \xrightarrow{m} P^\sharp(\preceq^\sharp)$, $B^\sharp \in P^\sharp(\preceq^\sharp) \xrightarrow{m} P^\sharp(\preceq^\sharp)$, $\alpha \circ F \circ \gamma \preceq^\sharp F^\sharp$, $\alpha \circ B \circ \gamma \preceq^\sharp B^\sharp$, $\dot{X}^0$ is $\mathsf{lfp}\, F^\sharp$ or $\mathsf{lfp}\, B^\sharp$ and for all $n \in \mathbb{N}$, $\dot{X}^{2n+1} = \mathsf{lfp}\, \lambda X \cdot (\dot{X}^{2n} \wedge^\sharp B^\sharp(X))$ and $\dot{X}^{2n+2} = \mathsf{lfp}\, \lambda X \cdot (\dot{X}^{2n+1} \wedge^\sharp F^\sharp(X))$ then $\forall k \in \mathbb{N} : \alpha(\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B) \preceq^\sharp \dot{X}^{k+1} \preceq^\sharp \dot{X}^k$. ■

Proof. Observe that by the fixpoint property, $\dot{X}^{2n+1} = \dot{X}^{2n} \wedge^\sharp B^\sharp(\dot{X}^{2n+1})$ and $\dot{X}^{2n+2} = \dot{X}^{2n+1} \wedge^\sharp F^\sharp(\dot{X}^{2n+2})$, hence $\dot{X}^{2n} \preceq^\sharp \dot{X}^{2n+1} \preceq^\sharp \dot{X}^{2n+2}$ since $\wedge^\sharp$ is the greatest lower bound for $\preceq^\sharp$ so that $\dot{X}^k$, $k \in \mathbb{N}$ is a decreasing chain.

We have $\alpha(\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B) \preceq^\sharp \alpha(\mathsf{lfp}\, F)$ since α is monotone and $\alpha(\mathsf{lfp}\, F) \preceq^\sharp \mathsf{lfp}\, F^\sharp$ by theorems on pages 120 and 120, thus proving the proposition for $k = 0$. Let us observe that $\alpha \circ F \circ \gamma \preceq^\sharp F^\sharp$ implies $F \circ \gamma \preceq \gamma \circ F^\sharp$ by def. Galois connection so that in particular for an argument of the form $\alpha(X)$, $F \circ \gamma \circ \alpha \preceq \gamma \circ F^\sharp \circ \alpha$. $\gamma \circ \alpha$ is extensive so that by monotony and transitivity $F \preceq \gamma \circ F^\sharp \circ \alpha$. Assume now by induction hypothesis that $\alpha(\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B) \preceq^\sharp \dot{X}^{2n}$, whence $\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B \preceq^\sharp \gamma(\dot{X}^{2n})$ by def. Galois connection. Since $F \preceq \gamma \circ F^\sharp \circ \alpha$, it follows that $\lambda X \cdot \mathsf{lfp}\, F \wedge \mathsf{lfp}\, B \wedge F(X) \preceq^\sharp \lambda X \cdot \gamma(\dot{X}^{2n}) \wedge \gamma \circ F^\sharp \circ \alpha(X) = \lambda X \cdot \gamma(\dot{X}^{2n} \wedge F^\sharp \circ \alpha(X))$ since γ is a complete meet morphism. Now by hypothesis (21) of theorel , we have $\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B = \mathsf{lfp}\, \lambda X \cdot (\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B \wedge F(X)) \preceq^\sharp \mathsf{lfp}\, \lambda X \cdot \gamma(\dot{X}^{2n} \wedge F^\sharp \circ \alpha(X))$ by theorem on page 120. Let G be $\lambda X \cdot \dot{X}^{2n} \wedge F^\sharp(X)$. $\alpha \circ \gamma$ is reductive so that by monotony $G \circ \alpha \circ \gamma \preceq^\sharp G$ and $\alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp G \circ \alpha \circ \gamma$, whence, by transitivity, $\alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp G$. By the theorem on page 120 we have $\alpha(\mathsf{lfp}\, \gamma \circ G \circ \alpha) \preceq^\sharp \mathsf{lfp}\, \alpha \circ \gamma \circ G \circ \alpha \circ \gamma \preceq^\sharp \mathsf{lfp}\, G$ by theorem on page 120. Hence, $\mathsf{lfp}\, \lambda X \cdot \gamma(\dot{X}^{2n} \wedge F^\sharp \circ \alpha(X)) \preceq \gamma(\mathsf{lfp}\, \lambda X \cdot \dot{X}^{2n} \wedge F^\sharp(X))$ so that by transitivity we conclude that $\alpha(\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B) \preceq^\sharp \dot{X}^{2n+1}$. The proof that $\alpha(\mathsf{lfp}\, F \wedge \mathsf{lfp}\, B) \preceq^\sharp \dot{X}^{2n+2}$ is similar using hypothesis (22) of theorem . $\square$

Example

– Let us consider the original example given in [2], with interval analysis, so that a widening/narrowing is required:

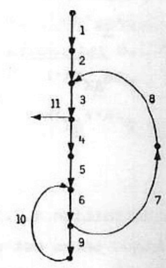
$$
\begin{array}{lll}
P^1 \sqsupseteq lfp(\overline{F}(\overline{\phi})) & & \\
P^2 = P^1 \underline{\Delta} X^2 & \text{où } X^2 & \sqsupseteq lfp(\lambda X . P^1 \sqcap \overline{B}(\overline{\Psi})(X)) \\
\ldots & & \\
P^{2k+1} = P^{2k} \underline{\Delta} X^{2k+1} & \text{où } X^{2k+1} & \sqsupseteq lfp(\lambda X . P^{2k} \sqcap \overline{F}(\overline{\phi})(X)) \\
P^{2k+2} = P^{2k+1} \underline{\Delta} X^{2k+2} & \text{où } X^{2k+2} & \sqsupseteq lfp(\lambda X . P^{2k+1} \sqcap \overline{B}(\overline{\phi})(X)) \\
\ldots & &
\end{array}
$$

– We illustrate on bubble sort [Z. Manna, 1974, p. 191]


```
{1}
        i:=n;
{2}
    L:
{3}
        si i≠0 alors
{4}
            j:=0;
{5}
    M:
{6}
            si j=i alors
{7}
                i:=i-1;
{8}
                allera L;
            finsi;
{9}
            j:=j+1;
{10}
            allera M;
        finsi;
{11}
```

– The analysis in [2] is not structural but equational and so the forward system of equations is:

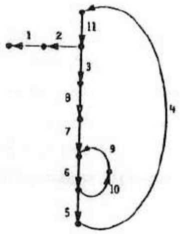
$$
\left\{
\begin{array}{lll}
P_1 & = & \overline{\phi} \\
P_2 & = & P_1(i \leftarrow n) \\
P_3 & = & P_2 \sqcup P_8 \\
P_4 & = & P_3(i \leftarrow i \neq 0) \\
P_5 & = & P_4(j \leftarrow [0,0]) \\
P_6 & = & P_5 \sqcup P_{10} \\
P_7 & = & P_6(i \leftarrow i \sqcap j, j \leftarrow i \sqcap j) \\
P_8 & = & P_7(i \leftarrow i-1) \\
P_9 & = & P_6(i \leftarrow i \neq j, j \leftarrow i \neq j) \\
P_{10} & = & P_9(j \leftarrow j+1) \\
P_{11} & = & P_3(i \leftarrow i \sqcap [0,0])
\end{array}
\right.
$$



Graphe de dépendance
(tête de cycle: 6)

– The backward system of equations is:

$$
\begin{cases}
P_1 = P_2(i \leftarrow [-\infty,+\infty], n \leftarrow i) \\
P_2 = P_3 \\
P_3 = P_{11} \sqcup P_4 \\
P_4 = P_5(j \leftarrow [-\infty,+\infty]) \\
P_5 = P_6 \\
P_6 = P_7 \sqcup P_9 \\
P_7 = P_8(i \leftarrow i+1) \\
P_8 = P_3 \\
P_9 = P_{10}(j \leftarrow j-1) \\
P_{10} = P_6 \\
P_{11} = \overline{\Psi}
\end{cases}
$$



Graphe de dépendance
(tête de cycle: 6)

– The considered program specification is:

$\overline{\Phi} = ((i=\bot),(j=\bot),(n=[-\infty,+\infty]))$
$\overline{\Psi} = ((i=[-\infty,+\infty]),(j=[-\infty,+\infty]),(n=[-\infty,+\infty]))$

– The first forward iteration is:

$P^1 \sqsupseteq \mathit{lfp}(\overline{F}(\overline{\phi}))$

	1	2	3	4	5	6	7	8	9	10	11
i	⊥	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[0,+∞]	[-1,+∞]	[-∞,+∞]	[-∞,+∞]	[0,0]
j	⊥	⊥	[0,+∞]	[0,+∞]	[0,0]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[1,+∞]	[0,+∞]
n	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]

It is discovered that j being initialized to 0 and incremented, it should be positive.

– The second iteration is:

$P^2 = P^1 \underline{\underline{\Delta}} X^2$ où $X^2 \sqsupseteq \mathit{lfp}(\lambda X. P^1 \sqcap \overline{B}(\overline{\Psi})(X))$

	1	2	3	4	5	6	7	8	9	10	11
i	⊥	[0,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[0,0]
j	⊥	⊥	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[1,+∞]	[0,+∞]
n	[0,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]	[-∞,+∞]

Since i is decremented by 1 in the loop which ends with i = 0, i must necessarily be positive on loop entry, so n must be initially positive

– The third iteration is:

$P^3 = P^2 \underline{\underline{\Delta}} X^3$ où $X^3 \sqsupseteq \mathit{lfp}(\lambda X. P^2 \sqcap \overline{B}(\overline{\Psi})(X))$

	1	2	3	4	5	6	7	8	9	10	11
i	⊥	[0,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[1,+∞]	[0,+∞]	[1,+∞]	[1,+∞]	[0,0]
j	⊥	⊥	[1,+∞]	[1,+∞]	[0,0]	[0,+∞]	[1,+∞]	[1,+∞]	[0,+∞]	[1,+∞]	[1,+∞]
n	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]	[0,+∞]

This propagates forward the information on n

– The next iteration shows that a fixpoint is reached

– The analysis proves that that if n is initially negative then the program exiecution either terminates by an error or does not terminate

– By propagating forward the invariants, one determines where execution tests have to be placed

– With polyhedral analysis, the result is the following:

```
Generic-FW-BW-REL-Abstract-Interpreter % cat ../Examples/example30.sil
i := n;
while (i <> 1) do
 j := 0;
 while(j <> i) do
   j := j + 1
 od;
 i := i - 1
od;;
```

```
Generic-FW-BW-REL-Abstract-Interpreter % ./a.out ../Examples/example30.sil
** Precondition:
{1>=0}
** Postcondition:
{i=1,n>=1}
unstable precondition after 1 iteration(s).
** Precondition:
{n>=1}
** Postcondition:
{i=1,n>=1}
stable precondition after 2 iteration(s).
```


Optimality

Lemma 1 *Let $P^\flat(\sqsubseteq^\flat, \bot^\flat, \top^\flat, \sqcap^\flat, \sqcup^\flat)$ and $P^\sharp(\sqsubseteq^\sharp, \bot^\sharp, \top^\sharp, \sqcap^\sharp, \sqcup^\sharp)$ be complete lattices with a Galois connection $P^\flat \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} P^\sharp$, $F^\flat \in P^\flat \to P^\flat$ and $B^\flat \in P^\flat \to P^\flat$ be two monotonic functions, and let*

$$L^\flat = \operatorname{gfp} \lambda Z.(Z \sqcap^\flat F^\flat(Z) \sqcap^\flat B^\flat(Z))$$

If $F^\sharp \in P^\sharp \to P^\sharp$ and $B^\sharp \in P^\sharp \to P^\sharp$ are monotonic and satisfy $\alpha \circ F^\flat \circ \gamma \sqsubseteq^\sharp F^\sharp$ and $\alpha \circ B^\flat \circ \gamma \sqsubseteq^\sharp B^\sharp$, then the sequence $(X_n)_{n\in\mathcal{N}}$ defined by $X_0 = \top^\sharp$, $X_{2n+1} = X_{2n} \sqcap^\sharp F^\sharp(X_{2n})$, and $X_{2n+2} = X_{2n+1} \sqcap^\sharp B^\sharp(X_{2n+1}), \forall n \geq 0$ is such that:

$$\forall n \geq 0,\ \alpha(L^\flat) \sqsubseteq^\sharp X_{n+1} \sqsubseteq^\sharp X_n$$

The optimality of this approach has been proved in [1]: it has been shown that $L^\sharp = \operatorname{gfp} \lambda Z.(Z \sqcap^\sharp F^\sharp(Z) \sqcap^\sharp B^\sharp(Z))$ is the greatest lower bound of the set $E^\sharp$ defined inductively as:

- $\top^\sharp \in E^\sharp$
- If Z is in $E^\sharp$ then so are $F^\sharp(Z)$ and $B^\sharp(Z)$.
- If Z_1 and Z_2 are in $E^\sharp$ then so are $Z_1 \sqcap^\sharp Z_2$ and $Z_1 \sqcup^\sharp Z_2$.

Therefore, $L^\sharp$ is the best upper approximation of $\alpha(L^\flat)$ that can be obtained using $F^\sharp$ and $B^\sharp$.

Standard backward-forward combination

The standard backward-forward combination [4] derives from an application of this lemma to the particular case of backward and forward collecting semantics:

$F^\flat$, $B^\flat$, $F^\sharp$ and $B^\flat$ are instantiated as follows:

$$\begin{aligned}
F^\flat &= \lambda Y.\operatorname{lgfp}_1 \lambda X.(Y \sqcap^\flat f^\flat(X)) \\
B^\flat &= \lambda Y.\operatorname{lgfp}_2 \lambda X.(Y \sqcap^\flat b^\flat(X)) \\
F^\sharp &= \lambda Y.\operatorname{lgfp}_1 \lambda X.(Y \sqcap^\sharp f^\sharp(X)) \\
B^\sharp &= \lambda Y.\operatorname{lgfp}_2 \lambda X.(Y \sqcap^\sharp b^\sharp(X))
\end{aligned}$$

where lgfp means either lfp or gfp, and $f^\flat$, $b^\flat \in P^\flat \to P^\flat$, $f^\sharp$, $b^\sharp \in P^\sharp \to P^\sharp$ are monotonic. When $\alpha \circ f^\flat \circ \gamma \sqsubseteq^\sharp f^\sharp$ and $\alpha \circ b^\flat \circ \gamma \sqsubseteq^\sharp b^\sharp$, the conditions of Lemma 1 are satisfied [7].

Now, let $P^\flat = \wp(\Sigma)$, Σ a set of states[3], and $\tau \in \wp(\Sigma \times \Sigma)$ a transition relation. As usual, we define pre, $\widetilde{pre}$, $post$, $\widetilde{post}$ as:

$$\begin{aligned}
post(X) &= \{s' \mid \exists s : \langle s, s'\rangle \in \tau \wedge s \in X\} \\
\widetilde{post}(X) &= \{s' \mid \forall s : \langle s, s'\rangle \in \tau \Rightarrow s \in X\} \\
pre(X) &= \{s \mid \exists s' : \langle s, s'\rangle \in \tau \wedge s' \in X\} \\
\widetilde{pre}(X) &= \{s \mid \forall s' : \langle s, s'\rangle \in \tau \Rightarrow s' \in X\}
\end{aligned}$$

Given $\mathcal{I}, \mathcal{F} \subseteq \Sigma$, sets of initial and final states, $f^\flat = \lambda X.(\mathcal{I} \cup post(X))$ and $b^\flat = \lambda X.(\mathcal{F} \cup pre(X))$ (and $\operatorname{lgfp}_1 = \operatorname{lgfp}_2 = \operatorname{lfp}$), we have [4]:

$$L^\flat = F^\flat(\top^\flat) \sqcap^\flat B^\flat(\top^\flat) = \operatorname{lfp} f^\flat \cap \operatorname{lfp} b^\flat \qquad (1)$$

By computing $\gamma(L^\sharp)$[4], we obtain a good upper approximation of $F^\flat(\Sigma) \cap B^\flat(\Sigma)$ (at least equal to $\gamma(F^\sharp(\top^\sharp) \sqcap^\sharp B^\sharp(\top^\sharp))$).

The ASTRÉE static analyzer

ASTRÉE: A Sound, Automatic, Specializable, Domain-Aware, Parametric, Modular, Efficient and Precise Static Program Analyzer

`www.astree.ens.fr`

- C programs:
  - with
    - pointers (including on functions), structures and arrays
    - floating point computations
    - tests, loops and function calls
    - limited branching (forward `goto`, `break`, `continue`)

- without
  - `union`
  - dynamic memory allocation
  - recursive function calls
  - backward branching
  - conflict side effects
  - C libraries
- Application Domain: safety critical embedded real-time synchronous software for non-linear control of very complex control/command systems.

Concrete Operational Semantics

- International norm of C (ISO/IEC 9899:1999)
- *restricted by* implementation-specific behaviors depending upon the machine and compiler (e.g. representation and size of integers, IEEE 754-1985 norm for floats and doubles)
- *restricted by* user-defined programming guidelines (such as no modular arithmetic for signed integers, even though this might be the hardware choice)
- *restricted by* program specific user requirements (e.g. `assert`)

Abstract Semantics

– Trace-based refinement of the reachable states for the concrete operational semantics
– Volatile environment is specified by a *trusted* configuration file.

Implicit Specification: Absence of Runtime Errors

– No violation of the norm of C (e.g. array index out of bounds)
– No implementation-specific undefined behaviors (e.g. maximum short integer is 32767)
– No violation of the programming guidelines (e.g. static variables cannot be assumed to be initialized to 0)
– No violation of the programmer assertions (must all be statically verified).

Example application

– Primary flight control software of the A340/A380 fly-by-wire system

– C program, automatically generated from a proprietary high-level specification (à la Simulink/SCADE)
– A340 family: 132,000 lines, 75,000 LOCs after preprocessing, 10,000 global variables, over 21,000 after expansion of small arrays
– A380: $\times$ 3

The Class of Considered Periodic Synchronous Programs

```
declare volatile input, state and output variables;
initialize state and output variables;
loop forever
   - read volatile input variables,
   - compute output and state variables,
   - write to volatile output variables;
  wait_for_clock ();
end loop
```

– Requirements: the only interrupts are clock ticks;
– Execution time of loop body less than a clock tick [1].

Reference

[1] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. *ESOP (2001)*, LNCS 2211, 469–485.

Characteristics of the ASTRÉE Analyzer

Static: compile time analysis (≠ run time analysis Rational Purify, Parasoft Insure++)

Program Analyzer: analyzes programs not micromodels of programs (≠ PROMELA in SPIN or Alloy in the Alloy Analyzer)

Automatic: no end-user intervention needed (≠ ESC Java, ESC Java 2)

Sound: covers the whole state space (≠ MAGIC, CBMC) so never omit potential errors (≠ UNO, CMC from coverity.com) or sort most probable ones (≠ Splint)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Multiabstraction: uses many numerical/symbolic abstract domains (≠ symbolic constraints in Bane or the canonical abstraction of TVLA)

Infinitary: all abstractions use infinite abstract domains with widening/narrowing (≠ model checking based analyzers such as VeriSoft, Bandera, Java PathFinder)

Efficient: always terminate (≠ counterexample-driven automatic abstraction refinement BLAST, SLAM)

Characteristics of the ASTRÉE Analyzer (Cont'd)

Specializable: can easily incorporate new abstractions (and reduction with already existing abstract domains) (≠ general-purpose analyzers PolySpace Verifier)

Domain-Aware: knows about control/command (e.g. digital filters) (as opposed to specialization to a mere programming style in C Global Surveyor)

Parametric: the precision/cost can be tailored to user needs by options and directives in the code

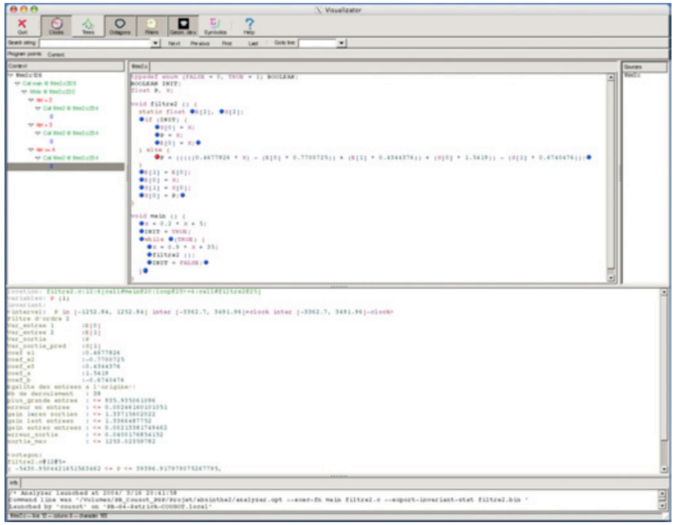
Characteristics of the ASTRÉE Analyzer (Cont'd)

Automatic Parametrization: the generation of parametric directives in the code can be programmed (to be specialized for a specific application domain)

Modular: an analyzer instance is built by selection of O-CAML modules from a collection each implementing an abstract domain

Precise: very few or no false alarm when adapted to an application domain ⟶ it is a VERIFIER!

Example of Analysis Session



Benchmarks (A340 Primary Flight Control Software)

- 132,000 lines, 75,000 LOCs after preprocessing
- Comparative results (commercial software):
  4,200 (false?) alarms,
  3.5 days;
- Our results, November 2003:
  0 alarms,
  40mn on 2.8 GHz PC,
  300 Megabytes
  ⟶ A world première!

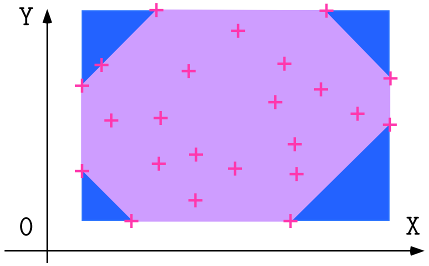
(A380 Primary Flight Control Software)

- 350,000 lines
- 0 alarms (mid-October 2004!),
  7h [2] on 2.8 GHz PC,
  1 Gigabyte
  ⟶ A world grand première!

[2] We are still in a phase where we favour precision rather than computation costs, and this should go down. For example, the A340 analysis went up to 5 h, before being reduced by requiring less precision while still getting no false alarm.

Examples of Abstractions

General-Purpose Abstract Domains: Intervals and Octagons



Intervals:

$$\begin{cases} 1 \leq x \leq 9 \\ 1 \leq y \leq 20 \end{cases}$$

Octagons:

$$\begin{cases} 1 \leq x \leq 9 \\ x + y \leq 77 \\ 1 \leq y \leq 20 \\ x - y \leq 04 \end{cases}$$

Difficulties: many global variables, arrays (smashed or not), IEEE 754 floating-point arithmetic (in program and analyzer)

Floating-Point Computations

– Code Sample:

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
} % gcc float-error.c
% ./a.out
0.000000
```

$(x+a) - (x-a) \neq 2a$

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951487.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
0.000000
```


Floating-Point Computations

– Code Sample:

```
/* float-error.c */
int main () {
  float x, y, z, r;
  x = 1.000000019e+38;
  y = x + 1.0e21;
  z = x - 1.0e21;
  r = y - z;
  printf("%f\n", r);
} % gcc float-error.c
% ./a.out
0.000000
```

$(x+a) - (x-a) \neq 2a$

```
/* double-error.c */
int main () {
double x; float y, z, r;
/* x = ldexp(1.,50)+ldexp(1.,26); */
x = 1125899973951488.0;
y = x + 1;
z = x - 1;
r = y - z;
printf("%f\n", r);
}
% gcc double-error.c
% ./a.out
134217728.000000
```


Symbolic abstract domain

- Interval analysis: if $x \in [a, b]$, $y \in [c, d]$ & $a, c \geq 0$ then $x - y \in [a - d, b - c]$ so if $x \in [0, 100]$ then $x - x \in [-100, 100]$!!!
- The symbolic abstract domain propagates the symbolic values of variables and performs simplifications;
- Must maintain the maximal possible rounding error for float computations (overestimated with intervals);

```
% cat -n x-x.c
     1  void main () { int X, Y;
     2          __ASTREE_known_fact(((0 <= X) && (X <= 100)));
     3          Y = (X - X);
     4          __ASTREE_log_vars((Y)); }
```

```
astree -exec-fn main -no-relational x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in [-100, 100]>
```

```
astree -exec-fn main x-x.c
Call main@x-x.c:1:5-x-x.c:1:9:
<interval: Y in {0}> <symbolic: Y = (X -i X)>
```

Clock Abstract Domain for Counters

– Code Sample:

```
R = 0;
while (1) {
  if (I)
    { R = R+1; }
  else
    { R = 0; }
  T = (R>=n);
  wait_for_clock ();
}
```

- Output T is true iff the volatile input I has been true for the last n clock ticks.
- The clock ticks every s seconds for at most h hours, thus R is bounded.
- To prove that R cannot overflow, we must prove that R cannot exceed the elapsed clock ticks *(impossible using only intervals)*.

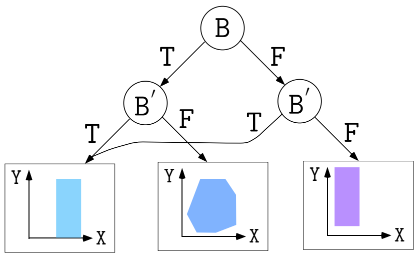
– Solution:

- We add a phantom variable `clock` in the concrete user semantics to track elapsed clock ticks.
- For each variable X, we abstract three intervals: X, X+clock, and X-clock.
- If X+clock or X-clock is bounded, so is X.

Boolean Relations for Boolean Control

– Code Sample:

```
/* boolean.c */
typedef enum {F=0,T=1} BOOL;
BOOL B;
void main () {
  unsigned int X, Y;
  while (1) {
    ...
    B = (X == 0);
    ...
    if (!B) {
      Y = 1 / X;
    }
    ...
  }
}
```



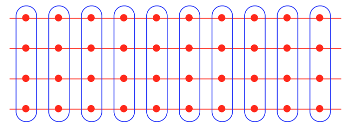
The boolean relation abstract domain is parameterized by the height of the decision tree (an analyzer option) and the abstract domain at the leafs

Control Partitionning for Case Analysis

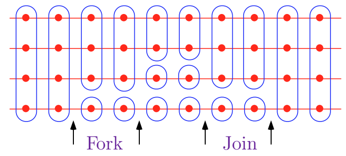
– Code Sample:

```
/* trace_partitionning.c */
void main() {
  float t[5] = {-10.0, -10.0, 0.0, 10.0, 10.0};
  float c[4] = {0.0, 2.0, 2.0, 0.0};
  float d[4] = {-20.0, -20.0, 0.0, 20.0};
  float x, r;
  int i = 0;

  ... found invariant -100 ≤ x ≤ 100 ...

  while ((i < 3) && (x >= t[i+1])) {
    i = i + 1;
  }
  r = (x - t[i]) * c[i] + d[i];
}
```

Control point partitionning:

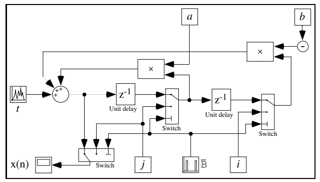


Trace partitionning:

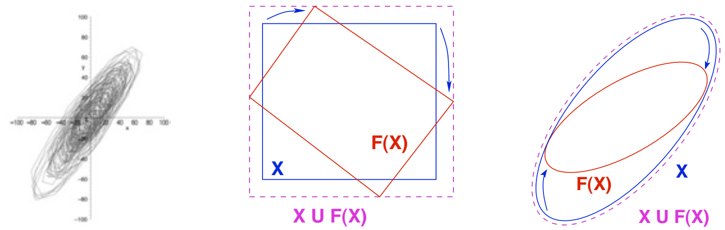


Delaying abstract unions in tests and loops is more precise for non-distributive abstract domains (and much less expensive than disjunctive completion).

2d Order Digital Filter: Ellipsoid Abstract Domain for Filters



- Computes $X_n = \begin{cases} \alpha X_{n-1} + \beta X_{n-2} + Y_n \\ I_n \end{cases}$
- The concrete computation is bounded, which must be proved in the abstract.
- There is no stable interval or octagon.
- The simplest stable surface is an ellipsoid.



execution trace — unstable interval — stable ellipsoid

Filter Example

```
typedef enum {FALSE = 0, TRUE = 1} BOOLEAN;
BOOLEAN INIT; float P, X;
void filter () {
  static float E[2], S[2];
  if (INIT) { S[0] = X; P = X; E[0] = X; }
  else { P = (((((0.5 * X) - (E[0] * 0.7)) + (E[1] * 0.4))
           + (S[0] * 1.5)) - (S[1] * 0.7)); }
  E[1] = E[0]; E[0] = X; S[1] = S[0]; S[0] = P;
  /* S[0], S[1] in [-1327.02698354, 1327.02698354] */
}
void main () { X = 0.2 * X + 5; INIT = TRUE;
  while (1) {
    X = 0.9 * X + 35; /* simulated filter input */
    filter (); INIT = FALSE; }
}
```


Arithmetic-Geometric Progressions

```
% cat retro.c
typedef enum {FALSE=0, TRUE=1} BOOL;
BOOL FIRST;
volatile BOOL SWITCH;
volatile float E;
float P, X, A, B;

void dev( )
{ X=E;
  if (FIRST) { P = X; }
  else
    { P = (P - ((((2.0 * P) - A) - B)
          * 4.491048e-03)); };
  B = A;
  if (SWITCH) {A = P;}
  else {A = X;}
}
```

```
void main()
{ FIRST = TRUE;
  while (TRUE) {
    dev( );
    FIRST = FALSE;
    __ASTREE_wait_for_clock(());
  }}
% cat retro.config
__ASTREE_volatile_input((E [-15.0, 15.0]));
__ASTREE_volatile_input((SWITCH [0,1]));
__ASTREE_max_clock((3600000));
|P| <= (15.  + 5.87747175411e-39
/ 1.19209290217e-07) * (1 +
1.19209290217e-07)^clock -
5.87747175411e-39 / 1.19209290217e-07
<= 23.0393526881
```


(Automatic) Parameterization

- All abstract domains of ASTRÉE are parameterized, e.g.
  - variable packing for octagones and decision trees,
  - partition/merge program points,
  - loop unrollings,
  - thresholds in widenings, ...;
- End-users can either parameterize by hand (analyzer options, directives in the code), or
- choose the automatic parameterization (default options, directives for pattern-matched predefined program schemata).

The main loop invariant for the A340

A textual file over 4.5 Mb with

- 6,900 boolean interval assertions ($x \in [0;1]$)
- 9,600 interval assertions ($x \in [a;b]$)
- 25,400 clock assertions ($x+\mathrm{clk} \in [a;b] \wedge x-\mathrm{clk} \in [a;b]$)
- 19,100 additive octagonal assertions ($a \le x+y \le b$)
- 19,200 subtractive octagonal assertions ($a \le x-y \le b$)
- 100 decision trees
- 60 ellipse invariants, etc ...

involving over 16,000 floating point constants (only 550 appearing in the program text) $\times$ 75,000 LOCs.

Possible origins of imprecision and how to fix it

In case of false alarm, the imprecision can come from:

- Abstract transformers (not best possible) $\longrightarrow$ improve algorithm;
- Automatized parametrization (e.g. variable packing) $\longrightarrow$ improve pattern-matched program schemata;
- Iteration strategy for fixpoints $\longrightarrow$ fix widening [3];
- Inexpressivity i.e. indispensable local inductive invariant are inexpressible in the abstract $\longrightarrow$ add a new abstract domain to the reduced product (e.g. filters).

[3] This can be very hard since at the limit only a precise infinite iteration might be able to compute the proper abstract invariant. In that case, it might be better to design a more refined abstract domain.

Bibliography

[2] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 March 1978.

[3] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. [4]

[4] www.astree.ens.fr/

[4] The editor of JLP has mistakenly published the unreadable galley proof. For a correct version of this paper, see `http://www.ens.fr/~cousot`.

THE END

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

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