Automatic Verification by Abstract Interpretation (DRAFT)

Patrick Cousot [a,1]

[a] *Département d'informatique, École normale supérieure*
45 rue d'Ulm, 75230 Paris cedex 05, France

Abstract

Key words: Abstract Interpretation, Ground Predicate Abstraction, Generic Predicate Abstraction, Program Verification, Abstraction Refinement, Static Program Analysis.

.

1 Introduction

Abstract interpretation theory [12, 16, 19, 21] formalizes the idea of *abstraction* for mathematical constructs involved in the specification of properties of computer systems.

Applications of abstract interpretation range from *static program analysis* [16, 19, 29] (including *dataflow analysis* [19, 25], *set-based analysis* [24], etc), *syntax analysis* [28], *hierarchies of semantics* (including proofs) [13, 15, 22] *typing* [14], *temporal verification* and *abstract model checking* [25] and *program transformation* [27].

All these techniques involve *approximations* that can be formalized by abstract interpretation. Consequently, sound (and complete) abstracts semantics, including abstract models, algorithms, etc can be derived systematically in a mathematically constructive way by algebraic calculation. The abstract interpretation methodology will be illustrated on ground predicate abstraction [1, 2, 3, 9, 30, 31, 32, 34, 35, 36, 37, 39, 46, 48, 50, 53] and on generic predicate abstraction, a program-independent generalization.

[1] `cousot@ens.fr`, `www.di.ens.fr/~cousot`

2 A Short Introduction to Abstract Interpretation

Let us recall a few elements of abstract interpretation from [19].

2.1 Properties and their Abstraction

Given a set Σ of objects (such as program states, execution traces, etc), we represent properties P of objects $s \in \Sigma$ as sets of objects $P \in \wp(\Sigma)$ (which have the property in question).

Example 1. The property "*to be an even natural number*" is represented by the infinite set $\{0, 2, 4, 6, \ldots\}$. □

Consequently, the set of properties of objects in Σ is a complete boolean lattice $\langle \wp(\Sigma),\ \subseteq,\ \emptyset,\ \Sigma,\ \cup,\ \cap,\ \neg \rangle$ [6, 33].

By "*abstraction*", we understand a reasoning or (mechanical) computation on objects such that only some properties of these objects can be used. Let us call *concrete* the general properties in $\wp(\Sigma)$. Let us call *abstract* the properties that can be used in the reasoning or computation. So, abstraction consists in *approximating* the concrete properties by the abstract ones.

This "*property-based*" point of view on abstraction is more general than the "*object-based*" point of view where the reasonings on a complex object are approximated by reasonings on a simpler similar structure (so called a "*model*"), as later shown in Ex. 11.

There are two possible directions of approximation. In the *approximation from above*, P is over approximated by $\overline{P}$ such that $P \subseteq \overline{P}$. This is illustrated in Fig. 1. In the *approximation from below*, P is under approximated by $\underline{P}$ such that $\underline{P} \subseteq P$. Obviously these notions are dual [6] since an approximation from above/below for $\subseteq/\supseteq$ is an inverse approximation from below/above for $\supseteq/\subseteq$. Moreover, the complement dual of an approximation from above/below for P is an approximation from below/above for $\neg P$. Therefore, from a purely mathematical point of view, only approximation from above need to be studied.

We let $\overline{\mathcal{A}} \subsetneq \wp(\Sigma)$ be the set of *abstract properties* (the only one which can be used to over-approximate concrete properties). In Fig. 2 we have illustrated the approximation of the set of points considered in Fig. 1 by signs [19], intervals [19], octagons [45] and polyhedra [29].

If the abstract properties are chosen arbitrarily (e.g. $\overline{\mathcal{A}} = \emptyset$), some concrete properties may have no (computable) abstraction. In this case, the ver-

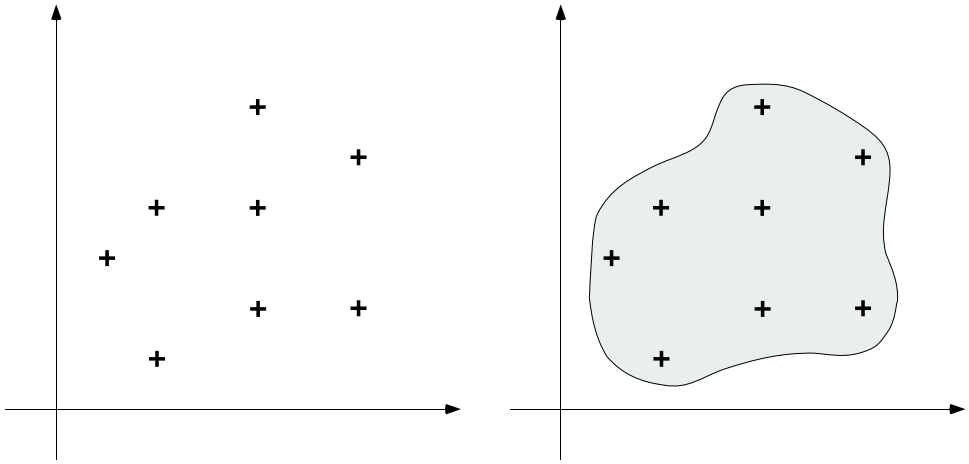


Fig. 1. A set of points and an upper approximation

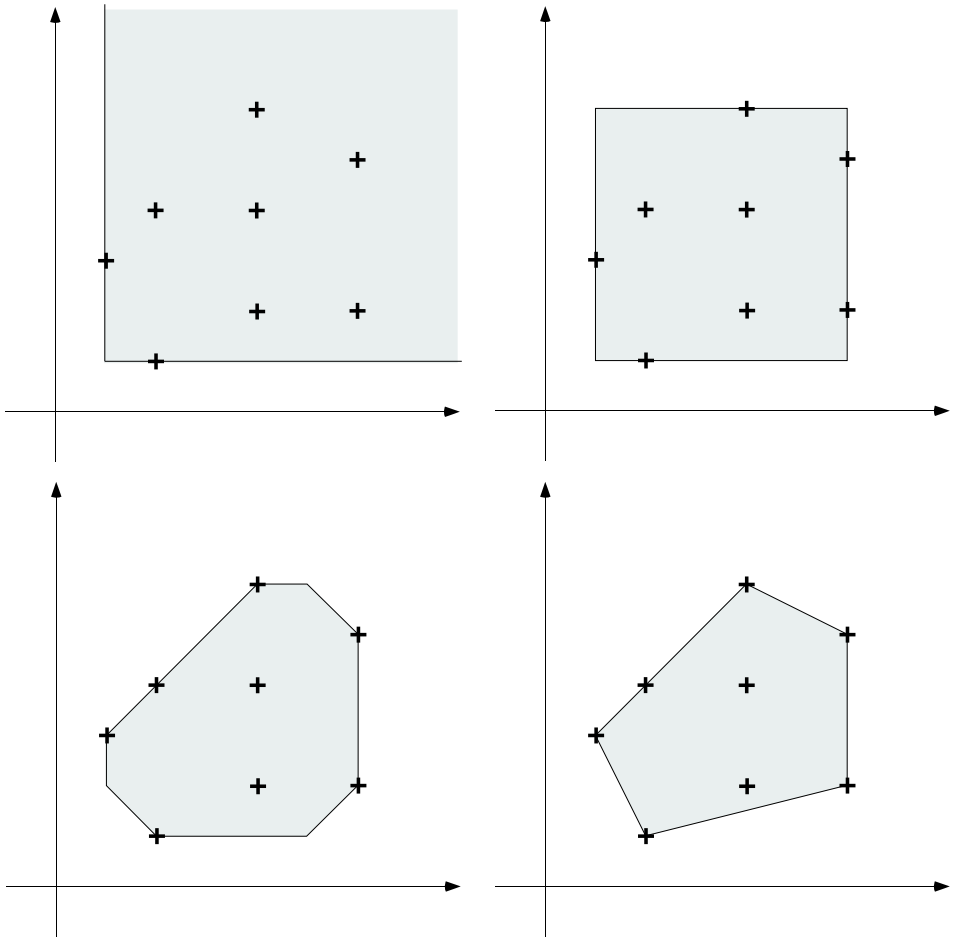


Fig. 2. Upper approximations by signs, intervals, octagons and polyhedra

ifier/prover/analyzer may loop forever, block (e.g. out of resources), ask for help or say something. We insist for always saying something in a finite amount

of time and therefore require that any property should be approximable from above by Σ (i.e. "true" or "*I don't know*").

In general a concrete property has several possible abstractions which may be more or less precise or even incomparable.

Example 2. Fig. 3 provides several interval approximations of a set of points by intervals (geometrically rectangles covering the cloud of points) including non-comparable ones (when the rectangles are not included into one another). □

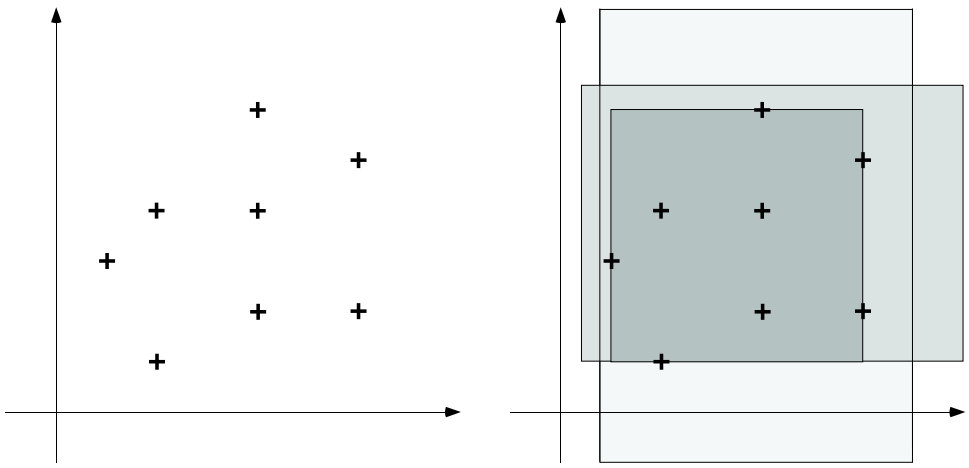


Fig. 3. Comparable and incomparable upper approximations by intervals

A concrete property $P \in \wp(\Sigma)$ is *most precisely abstracted* by any minimal upper approximation $\overline{P} \in \overline{\mathcal{A}}$:

$$P \subseteq \overline{P}$$
$$\nexists \overline{P'} \in \overline{\mathcal{A}} : P \subseteq \overline{P'} \subsetneq \overline{P}$$

In particular, an abstract property $\overline{P} \in \overline{\mathcal{A}}$ is best approximated by itself.

Example 3. In the classical rule of signs, 0 is better approximated by the minimal abstract properties *positive* $(+1)$ or *negative* (-1) (than by non-minimal ones such as e.g. *I don't know*). □

A first problem is that minimal abstract properties approximating a given concrete property may not exist at all.

Example 4. This is the case for the linear inequalities/polyhedra abstract domain of [29]. For example, a disk (i.e. the property of points inside a circle)

has no minimal approximation by polyhedra (i.e. the property of points inside a polyhedron), as illustrated in Fig. 4 [2]. □

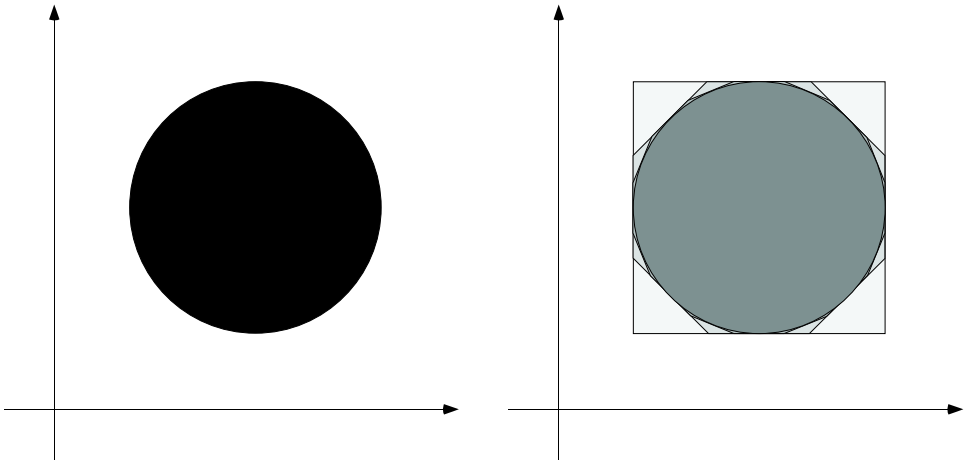


Fig. 4. Archimedes' upper approximations of a disk by polyhedra

A second problem is when minimal abstract properties approximating a given concrete property do exist but there are several incomparable such minimal abstract properties. Then the question of which minimal approximation is *most useful* depends upon the circumstances.

Example 5. In the rule of signs [19], 0 is better approximated as *positive* in "$3 + 0$" (since the sum of two positives is positive) while 0 is better approximated as *negative* in "$-3 + 0$" (since the sum of two negatives is negative and, in both cases, the sign of the sum of two numbers of opposite signs is unknown). □

We would like to avoid backtracking (that is to exhaustively try all minimal approximations one after the other) as well as parallelism (that is simultaneously try all minimal approximations) because of potential exponential costs when these strategies are applied recursively. Consequently, one may want to use only one of the minimal over approximations.

Since, for economy, we want to use only one of the minimal abstractions of a concrete property, we must, for best precision, choose one of the *minimal abstractions*. We can either make a *circumstantial choice* [3] (which may be

[2] Indeed Fig. 4 shows the first elements of Archimedes' infinite sequence of polyhedra which is decreasing (each polyhedron is strictly included in the previous one) and which limit is the disk, hence not a polyhedron, thus proving the absence of best upper approximation of a disk by a polyhedron.

[3] in that case, [21] uses a concretization function, an example of application being [29].

different each time an over approximation is needed), or make a definitive *arbitrary choice* [4] (with the risk that in some circumstances this arbitrary choice is not the best) [5] or require the existence of a *best choice* [6].

The requirement that all concrete property $P \in \wp(\Sigma)$ have a *best abstraction* $\overline{P} \in \overline{\mathcal{A}}$:

$$P \subseteq \overline{P}$$
$$\forall \overline{P'} \in \overline{\mathcal{A}} : (P \subseteq \overline{P'}) \;\Rightarrow\; (\overline{P} \subseteq \overline{P'})$$

is, by definition of the intersection/conjunction/greatest lower bound/meet $\cap$, equivalent to the fact that the intersection of abstract properties should be abstract:

$$\overline{P} = \bigcap \{\overline{P'} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P'}\} \in \overline{\mathcal{A}}$$

(since otherwise $\bigcap \{\overline{P'} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P'}\}$ would have no best abstraction).

2.2 Moore Family-Based Abstraction

So, the hypothesis that any concrete property $P \in \wp(\Sigma)$ has a *best abstraction* $\overline{P} \in \overline{\mathcal{A}}$ implies that [19, Sec. 5.1]:

$$\overline{\mathcal{A}} \text{ is a Moore family } [7]$$

that is, by definition, the set of abstract properties is closed under intersection $\cap$:

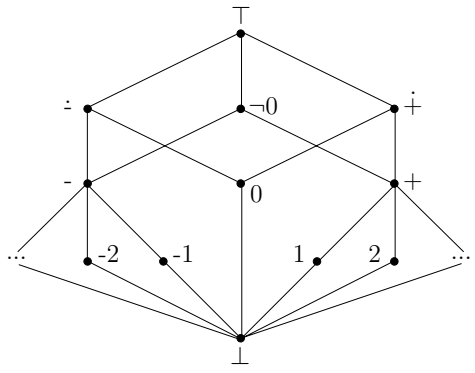
$$\forall S \subseteq \overline{\mathcal{A}} : \bigcap S \in \overline{\mathcal{A}} \ .$$

In particular $\bigcap \emptyset = \Sigma \in \overline{\mathcal{A}}$ so that any Moore family has a supremum.

Observe that if the abstract domain $\overline{\mathcal{A}}$ is assumed to be a Moore family of a complete lattice $\langle \wp(\Sigma), \subseteq, \emptyset, \Sigma, \cup, \cap \rangle$ then it is a complete lattice $\langle \overline{\mathcal{A}}, \subseteq, \cap\overline{\mathcal{A}}, \Sigma, \lambda S \bullet \cap\{P \in \overline{\mathcal{A}} \mid \cup S \subseteq P\}, \cap \rangle$.

[4] in that case, [21] uses an abstraction function.
[5] Note that the strategies of making a circumstantial or arbitrary choice also cover the case of absence of minimal approximations, see [21].
[6] in that case, [21] uses an abstraction/concretization Galois connection, which is the only case later considered in this paper.
[7] Moore families are called *topped intersection structures* or *closure systems* in [33].

Example 6. Let us consider the complete lattice of concrete properties as given in Fig. 5. It is a Moore family of the concrete properties $\langle \wp(\mathbb{Z}),\ \subseteq \rangle$ capturing signs and integer constant information. A further abstraction is



$$
\begin{aligned}
\bot &\triangleq \emptyset, && \text{infimum;}\\
z &\triangleq \{z\}, && \text{singleton, } z \in \mathbb{Z};\\
- &\triangleq \{z \in \mathbb{Z} \mid z < 0\}, && \text{negative;}\\
\dot{-} &\triangleq \{z \in \mathbb{Z} \mid z \leq 0\}, && \text{non-positive;}\\
\neg 0 &\triangleq \{z \in \mathbb{Z} \mid z \neq 0\}, && \text{non-zero;}\\
\dot{+} &\triangleq \{z \in \mathbb{Z} \mid z \geq 0\}, && \text{non-negative;}\\
+ &\triangleq \{z \in \mathbb{Z} \mid z > 0\}, && \text{positive;}\\
\top &\triangleq \mathbb{Z}, && \text{supremum.}
\end{aligned}
$$

Fig. 5. The complete lattice of constants and signs

captured by the Moore family of Fig. 6 where strict sign and integer constant information is ignored. □

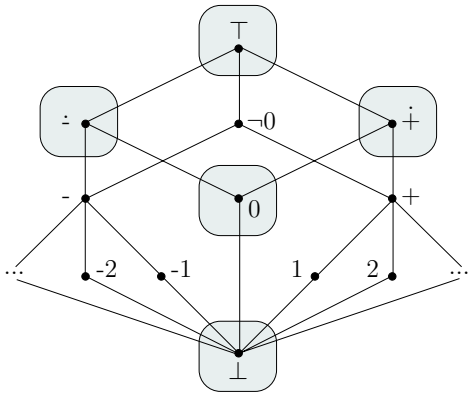


Fig. 6. Moore family-based abstraction

The set $\mathcal{M}(\wp(\wp(\Sigma)))$ of all abstractions i.e. of Moore families on the set $\wp(\Sigma)$ of concrete properties is the complete *lattice of abstractions*:

$$\langle \mathcal{M}(\wp(\wp(\Sigma))),\ \supseteq,\ \wp(\Sigma),\ \{\Sigma\},\ \lambda S \bullet \mathcal{M}(\cup S),\ \cap \rangle$$

where:

$$\mathcal{M}(\overline{\mathcal{A}}) = \{\bigcap S \mid S \subseteq \overline{\mathcal{A}}\}$$

is the $\subseteq$-least Moore family containing $\overline{\mathcal{A}}$ and the set image $f(X)$ of a set $X \subseteq D$ by a map $f \in D \mapsto E$ is $f(X) \stackrel{\triangle}{=} \{f(x) \mid x \in X\}$.

2.3 Closure Operator-Based Abstraction

The map $\rho_{\bar{\mathcal{A}}}$ mapping a concrete property $P \in \wp(\Sigma)$ to its best abstraction $\rho_{\bar{\mathcal{A}}}(P)$ in the Moore family $\overline{\mathcal{A}}$ is:

$$\rho_{\bar{\mathcal{A}}}(P) = \bigcap \{\overline{P} \in \overline{\mathcal{A}} \mid P \subseteq \overline{P}\} \ .$$

It is a *closure operator* [19, Sec. 5.2]:

extensive $(\forall P \in \wp(\Sigma) : P \subseteq \rho(P))$,

idempotent $(\forall P \in \wp(\Sigma) : \rho(\rho(P)) = \rho(P))$,

isotone/monotonic $(\forall P, P' \in \wp(\Sigma) : P \subseteq P' \Rightarrow \rho(P) \subseteq \rho(P'))$;

such that

$$P \in \bar{\mathcal{A}} \iff P = \rho_{\bar{\mathcal{A}}}(P)$$

hence

$$\overline{\mathcal{A}} = \rho_{\bar{\mathcal{A}}}(\wp(\Sigma)) \ .$$

$\rho_{\bar{\mathcal{A}}}$ is called the *closure operator induced by the abstraction* $\bar{\mathcal{A}}$.

Example 7. The closure operator induced by the abstraction of Fig. 6 in shown in Fig. 7. □

Closure operators are isomorphic to their fixpoints hence to the Moore families. Therefore, any closure operator ρ on the set of properties $\wp(\Sigma)$ induces an abstraction $\rho(\wp(\Sigma))$. For example:

$\lambda P \bullet P$ is the most precise abstraction (*identity*),

$\lambda P \bullet \Sigma$ is the most imprecise abstraction (*I don't know*).

The abstract domain $\bar{\mathcal{A}} = \rho(L)$ defined by a closure operator ρ on a complete lattice of concrete properties $\langle L, \sqsubseteq, \bot, \top, \sqcup, \sqcap \rangle$ is a complete lattice $\langle \rho(L), \sqsubseteq, \rho(\bot), \top, \lambda S \bullet \rho(\sqcup S), \sqcap \rangle$.

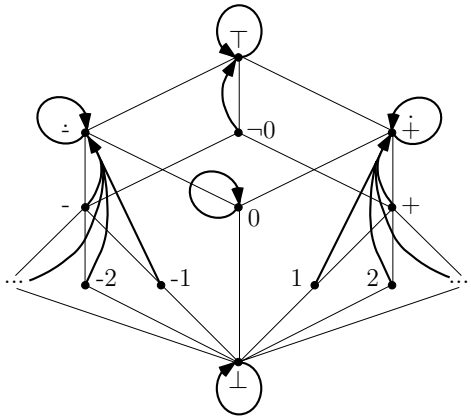


Fig. 7. Closure operator-based abstraction

The set $\mathbf{clo}(\wp(\Sigma) \mapsto \wp(\Sigma))$ of all abstractions, i.e. isomorphically, closure operators ρ on the set $\wp(\Sigma)$ of concrete properties is the complete *lattice of abstractions* for pointwise inclusion [54]:

$$\langle \mathbf{clo}(\wp(\Sigma) \mapsto \wp(\Sigma)),\ \dot{\subseteq},\ \lambda P \bullet P,\ \lambda P \bullet \Sigma,\ \lambda S \bullet \mathbf{ide}(\dot{\cup} S),\ \dot{\cap} \rangle$$

where:

— the glb $\dot{\cap}$ is the *reduced product*;
— $\mathbf{ide}(\rho) = \mathrm{lfp}_{\rho}^{\dot{\subseteq}}\, \lambda f \bullet f \circ f$ is the $\dot{\subseteq}$-least idempotent operator on $\wp(\Sigma)$ $\dot{\subseteq}$-greater than ρ.

2.4 Galois Connection-Based Abstraction

For closure operators ρ, we have:

$$\rho(P) \subseteq \rho(P') \iff P \subseteq \rho(P')$$

stating that $\rho(P')$ is an abstraction of a property P if and only if it $\subseteq$-approximates its best abstraction $\rho(P)$. This can be written:

$$\langle \wp(\Sigma),\ \subseteq \rangle \underset{\rho}{\overset{1}{\leftrightarrows}} \langle \rho(\wp(\Sigma)),\ \subseteq \rangle$$

where 1 is the identity and:

$$\langle \wp(\Sigma),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{D}},\ \sqsubseteq \rangle$$

means that $\langle \alpha,\ \gamma \rangle$ is a *Galois surjection*, that is:

— $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \ \Leftrightarrow \ P \subseteq \gamma(\overline{P})$;
— α is onto (equivalently $\alpha \circ \gamma = 1$ or γ is one-to-one).

Observe that reciprocally if $\langle \wp(\Sigma),\ \subseteq \rangle \underset{\rho}{\overset{1}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle \rho(\wp(\Sigma)),\ \subseteq \rangle$ holds that ρ is a closure operator so that this can be taken as an equivalent definition of closure operators.

We can define an *abstract domain* as an isomorphic representation $\overline{\mathcal{D}}$ of the set $\overline{\mathcal{A}} \subsetneq \wp(\Sigma) = \rho(\wp(\Sigma))$ of abstract properties (up to some order-isomorphism ι).

Example 8. Kildall's constant propagation lattice $\langle \{\bot, \top\} \cup \{i \mid i \in \mathbb{Z}\},\ \sqsubseteq \rangle$ [40] shown in Fig. 18 encodes the abstract domain $\langle \{\emptyset, \mathbb{Z}\} \cup \{\{i\} \mid i \in \mathbb{Z}\},\ \subseteq \rangle$. □

Then, with such an encoding, we have the Galois surjection [8] :

$$\langle \wp(\Sigma),\ \subseteq \rangle \underset{\iota \circ \rho}{\overset{\iota^{-1}}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle \overline{\mathcal{D}},\ \sqsubseteq \rangle$$

More generally, the correspondence between concrete and abstract properties can be established by an arbitrary Galois surjection [19, Sec. 5.3]:

$$\langle \wp(\Sigma),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\twoheadrightarrow}} \langle \overline{\mathcal{D}},\ \sqsubseteq \rangle$$

where this notation means (again) that:

— $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \ \Leftrightarrow \ P \subseteq \gamma(\overline{P})$;
— α is onto (equivalently $\alpha \circ \gamma = 1$ or γ is one-to-one).

Example 9. The abstraction of the concrete properties of Fig. 5 in the abstract domain $\{\bot, -1, 0, +1, \top\}$ is given in Fig. 8. The corresponding concretization is given in Fig. 9. □

Relaxing the condition that α is onto:

$$\langle \wp(\Sigma),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \overline{\mathcal{D}},\ \sqsubseteq \rangle$$

that is to say:

$$\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \ \Leftrightarrow \ P \subseteq \gamma(\overline{P});$$

[8] Also called Galois insertion since γ is injective.

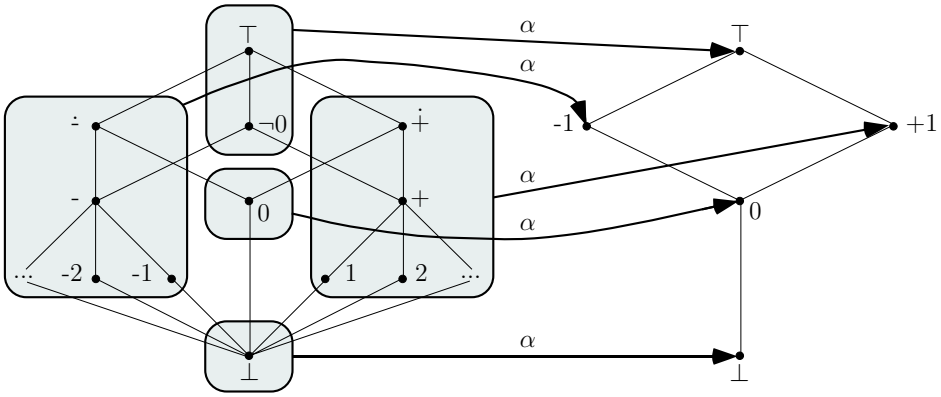


Fig. 8. Galois surjection-based abstraction (α)

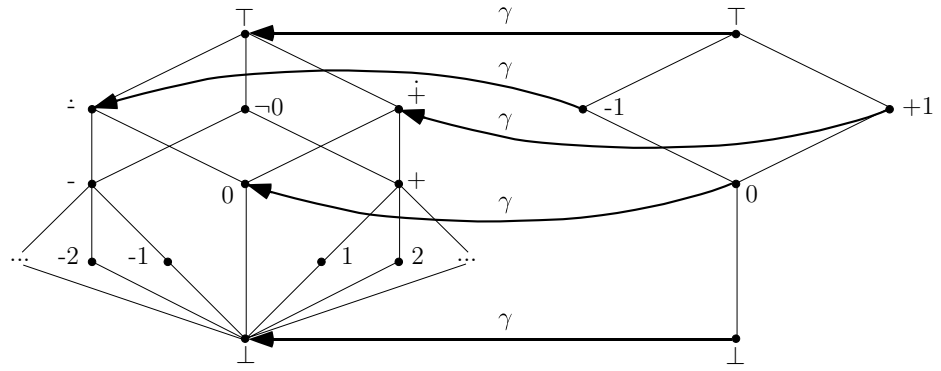


Fig. 9. Galois surjection-based concretization (γ)

so ρ is now $\gamma \circ \alpha$ [9] . When α is not onto, the same abstract property can have different representations.

Example 10. This is illustrated on Fig. 10 where "positive" can be represented as both "+1" and "1". The corresponding concretization is given in Fig. 11 where both "+1" and "1" mean "$\dot{+}$". □

Example 11. The classical idea of abstraction of an object $s \in \Sigma$ by a simpler one $h(s) \in \mathcal{M}$ where $h \in \Sigma \mapsto \mathcal{M}$ leads to the Galois connection $\langle \wp(\Sigma), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \wp(\mathcal{M}), \subseteq \rangle$ where:

$$\alpha(S) \triangleq \{h(s) \mid s \in S\},$$
$$\gamma(M) \triangleq \{s \in \Sigma \mid h(s) \in M\}.$$

It is a Galois surjection when h is onto.

[9] In absence of best approximation, one can use a semi-connection, requiring only $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \;\Rightarrow\; P \subseteq \gamma(\overline{P})$, see [21].

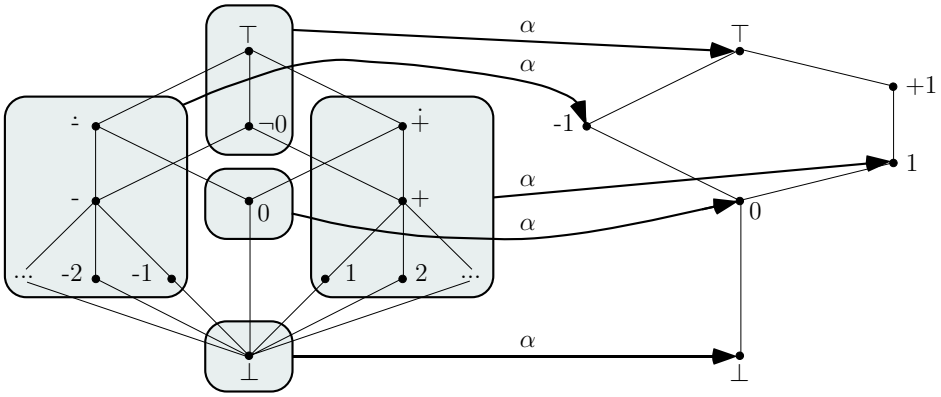


Fig. 10. Galois surjection-based abstraction (α)

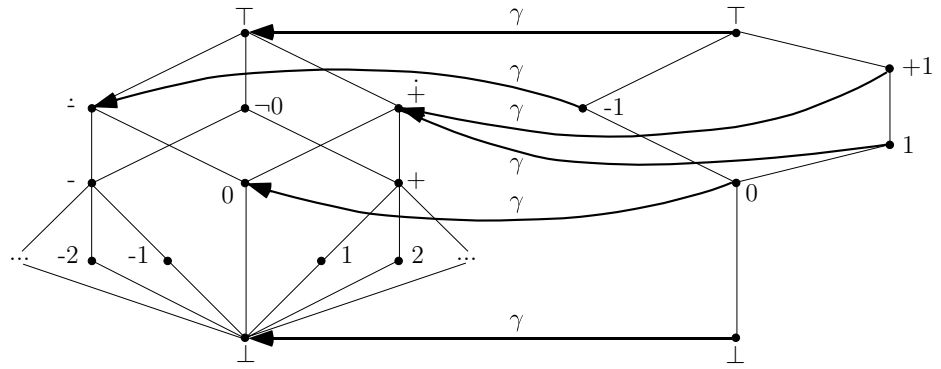


Fig. 11. Galois surjection-based concretization (γ)

An example of object-based surjective abstraction is the rule of signs where $h(0) = 0$, $h(i) = +1$ when $i > 0$ and $h(i) = -1$ when $h(i) = -1$ when $i < 0$ together with an isomorphic encoding such that e.g. $\emptyset$ is encoded as $\bot$, $\{-1, 0, +1\}$ as $\top$, $\{0, +1\}$ as $\dot{+}$, $\{+1\}$ as $+$, etc.

This object-based abstraction is the only type of abstraction considered in [8] and most other references in model checking (with the notable exception of polyhedral abstractions to verify hybrid systems [38]).

This object-based type of abstraction is a particular case of the more general property-based abstraction considered in abstract interpretation. For example the constant propagation, intervals, octagons, polyhedral, etc abstract domains are not object-based abstractions. Indeed, if for constant propagation, we want to have $\alpha(\{1, 2\} = \{h(1), h(2)\} = \{\top\}$ then necessarily $h(i) = \top$ when $i \in \mathbb{Z}$ and so $\alpha(\{1\} = \{h(1)\} = \{\top\}$ as opposed to the desired $\alpha(\{1\} = \{h(1)\} = \{1\}$ requiring $h(i) = i$ when $i \in \mathbb{Z}$. $\square$

Observe that the inverse dual of $\langle L, \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq\rangle$ is $\langle M, \sqsupseteq\rangle \underset{\gamma}{\overset{\alpha}{\leftrightarrows}} \langle L, \geq\rangle$.

Given posets $\langle L, \leq \rangle$ and $\langle M, \sqsubseteq \rangle$, the definition $\forall P \in \wp(\Sigma), \overline{P} \in \overline{\mathcal{D}} : \alpha(P) \sqsubseteq \overline{P} \Leftrightarrow P \subseteq \gamma(\overline{P})$ of Galois connections $\langle L, \leq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq \rangle$ is equivalent to the following four conditions:

— α is monotonic ($\forall x, x' \in L : x \leq x' \Rightarrow \alpha(x) \sqsubseteq \alpha(x')$, thus α preserves concrete implication $\leq$ in the abstract), as illustrated in Fig. 12;
— γ is monotonic ($\forall y, y' \in M : y \sqsubseteq y' \Rightarrow \gamma(y) \leq \gamma(y')$, thus γ preserves abstract implication $\sqsubseteq$ in the concrete), as illustrated in Fig. 13;
— $\gamma \circ \alpha$ is extensive ($\forall x \in L : x \leq \gamma(\alpha(x))$, so $\rho \triangleq \gamma \circ \alpha$ and the approximation is from above), as illustrated in Fig. 14;
— $\alpha \circ \gamma$ is reductive ($\forall y \in M : \alpha(\gamma(y)) \sqsubseteq y$, so concretization can loose no information), as illustrated in Fig. 15.

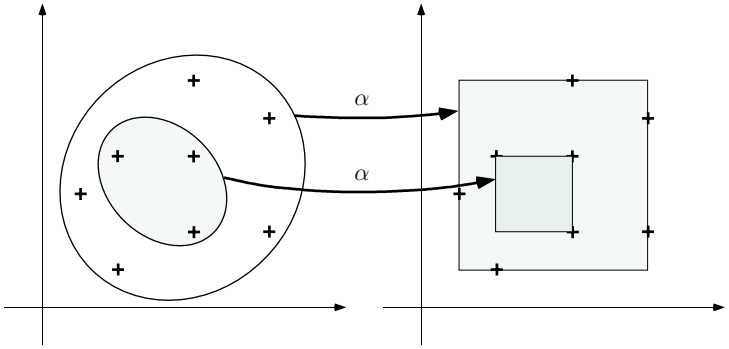


Fig. 12. The abstraction α is monotonic

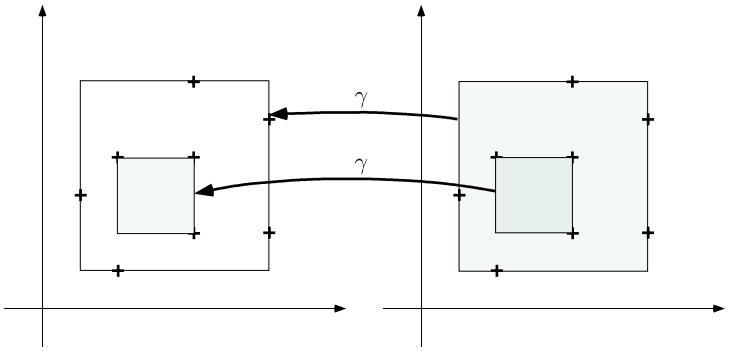


Fig. 13. The concretization γ is monotonic

The composition $\alpha \circ \gamma$ is the identity if and only if α is onto or equivalently γ is one-to-one.

The composition of Galois connections:

$$\langle L, \leq \rangle \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} \langle M, \sqsubseteq \rangle$$

and:

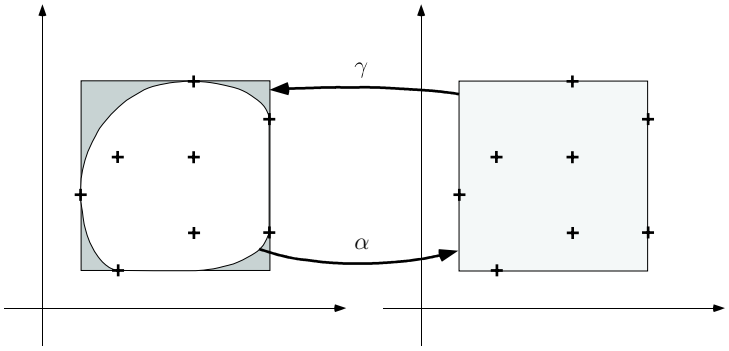


Fig. 14. The abstraction-concretization composition $\gamma \circ \alpha$ is extensive

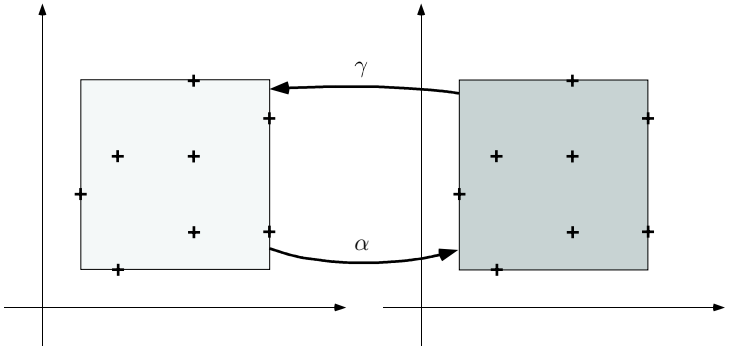


Fig. 15. The concretization-abstraction composition $\alpha \circ \gamma$ is reductive (here identity)

$$\langle M, \sqsubseteq \rangle \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} \langle N, \preceq \rangle$$

is a Galois connection:

$$\langle L, \leq \rangle \underset{\alpha_2 \circ \alpha_1}{\overset{\gamma_1 \circ \gamma_2}{\leftrightarrows}} \langle N, \preceq \rangle \ .$$

Example 12. The composition of the octagon and interval abstractions is illustrated in Fig. 16. □

2.5 Function Abstraction

Given a complete lattice $\langle L, \subseteq, \bot, \top, \cup, \cap \rangle$, and an abstraction ρ on L, the best abstraction of a monotone operator $f \in L \overset{\text{mon}}{\longmapsto} L$ on the complete lattice L is $\rho \circ f \in \rho(L) \overset{\text{mon}}{\longmapsto} \rho(L)$ [19, Sec. 7.2]. Indeed given any other $\overline{f} \in \rho(L) \overset{\text{mon}}{\longmapsto} \rho(L)$ and $\overline{x} \in \rho(L)$ the soundness requirement $\overline{f}(\overline{x}) \supseteq f(\overline{x})$ implies $f(\overline{x}) \subseteq \rho \circ \overline{f}(\overline{x})$ since ρ is idempotent whence $\rho \circ f(\overline{x}) \subseteq \rho \circ \overline{f}(\overline{x})$ proving that $\rho \circ f$ is more

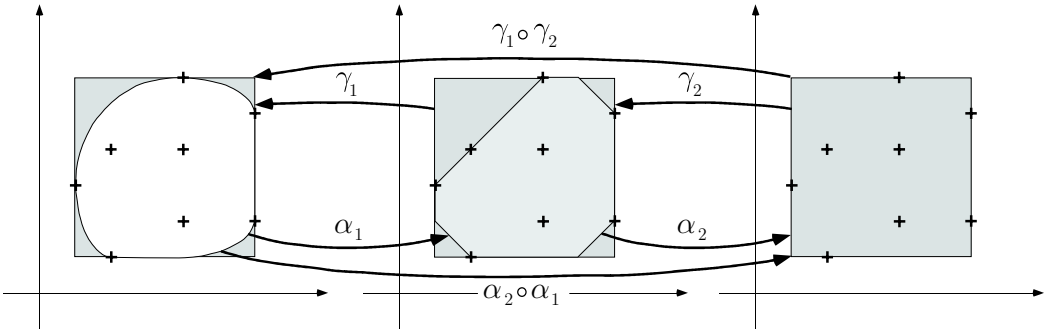


Fig. 16. The composition of octagon and interval abstractions

precise than any other sound abstraction $\overline{f} \in \rho(L) \stackrel{\text{mon}}{\longmapsto} \rho(L)$ of $f \in L \stackrel{\text{mon}}{\longmapsto} L$: $\rho \circ f \mathbin{\dot{\subseteq}} \overline{f}$ for the pointwise ordering $f \mathbin{\dot{\subseteq}} g$ if and only if $\forall x \in L : f(x) \subseteq g(x)$.

In terms of Galois connections, we have:

$$\langle P, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle Q, \sqsubseteq\rangle \quad \Rightarrow \quad \langle P \stackrel{\text{mon}}{\longmapsto} P, \mathbin{\dot{\subseteq}}\rangle \underset{\lambda F \bullet \alpha \circ F \circ \gamma}{\overset{\lambda F^\sharp \bullet \gamma \circ F^\sharp \circ \alpha}{\leftrightarrows}} \langle Q \stackrel{\text{mon}}{\longmapsto} Q, \mathbin{\dot{\sqsubseteq}}\rangle \ .$$

2.6 Fixpoint Abstraction

Given a complete lattice $\langle L, \subseteq, \bot, \top, \cup, \cap\rangle$ and a monotone operator $f \in L \stackrel{\text{mon}}{\longmapsto} L$ on the complete lattice L, its least fixpoint $\text{lfp}_a^{\subseteq} f$ greater than $a \in L$ is defined, if it exists, as:

$$\begin{aligned} &a \subseteq \text{lfp}_a^{\subseteq} f = f(\text{lfp}_a^{\subseteq} f);\\ &\forall x \in L : a \subseteq x = f(x) \Rightarrow \text{lfp}_a^{\subseteq} f \subseteq x. \end{aligned}$$

If $a \subseteq f(a)$ then $\text{lfp}_a^{\subseteq} f$ does exists [52] and is the limit of the ultimately stationary transfinite sequence $f^\eta, \eta \in \mathbb{O}$ defined by [18]:

$$\begin{aligned} f^0 &\triangleq a;\\ f^{\eta+1} &\triangleq f(f^\eta), \text{ for successor ordinals } \eta + 1;\\ f^\lambda &\triangleq \bigcup_{\eta<\lambda} f^\eta, \text{ for limit ordinals } \lambda. \end{aligned}$$

In particular the least fixpoint of f is $\text{lfp}^{\subseteq} f \triangleq \text{lfp}_\bot^{\subseteq} f$.

Given $f \in L \stackrel{\text{mon}}{\longmapsto} L$ on the complete lattice L and an abstraction ρ, we would like to approximate $\text{lfp}_a^{\subseteq} f$ in $\rho(L)$. The best abstraction $\rho(\text{lfp}_a^{\subseteq} f)$ is in general

not computable since neither $\mathrm{lfp}_a^{\subseteq} f$ nor ρ are. However, following [19, Sec. 7.1], a computable pointwise over approximation $\overline{f}$ is sufficient to check for an over approximation of $\mathrm{lfp}_a^{\subseteq} f$ in $\rho(L)$:

$$\forall y \in \rho(L) : \Big(\rho \circ f \mathrel{\dot{\subseteq}} \overline{f} \wedge \rho(a) \subseteq y \wedge \overline{f}(y) \subseteq y\Big) \Rightarrow \Big(\mathrm{lfp}_a^{\subseteq} f \subseteq y\Big) \ . \qquad (1)$$

Moreover the $\subseteq$-least such y is $\mathrm{lfp}^{\subseteq}_{\rho(a)} \rho \circ f$ (which does exist since $a \subseteq f(a)$ implies $\rho(a) \subseteq \rho \circ f(\rho(a))$) such that $\rho(y) = y$. This means that we can abstract fixpoints by fixpoints, as illustrated in Fig. 17.

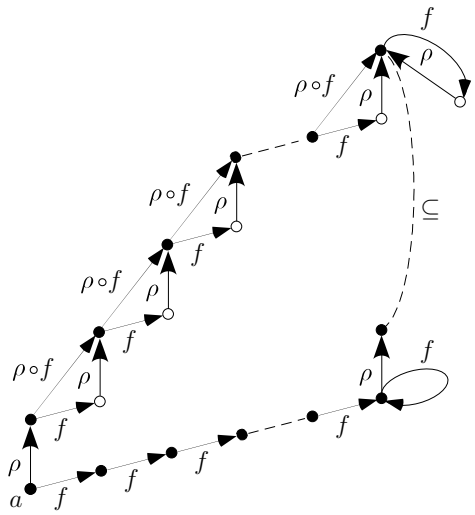


Fig. 17. Fixpoint approximate abstraction

3 Application to Ground Predicate Abstraction

Ground predicate abstraction was introduced by [37] as a verification method by abstract interpretation which can be implemented using a theorem prover and a model-checker [1, 2, 3, 9, 30, 31, 32, 34, 35, 36, 37, 39, 46, 48, 50, 53]. The general idea is that elements of the abstract domain are chosen as ground abstract predicates which interpretation is a set of states of a given program. Transfer functions are computed using a theorem prover or preferably a simplifier. If the abstract domain is finite it can be encoded in a boolean abstract domain so as to reuse existing model-checkers for fixpoint computations [10] .

[10] This encoding is not specific to ground predicate abstraction and is applicable for any *finite* abstract domain.

3.1 Syntax of the Programming Language

We consider a simple imperative programming language. The syntax, variables and labelling of expressions, boolean expressions and commands is defined by an attribute grammar [42].

3.1.1 Syntax of Expressions $E \in \mathbb{E}$

We assume that we are given a denumerable set of programming variables $\mathtt{X} \in \mathbb{X}$. The attribute $\mathrm{var}[\![E]\!]$ is the set of programming variables appearing in the expression $E \in \mathbb{E}$.

$$\begin{array}{lll} E ::= \mathtt{1} & & \mathrm{var}[\![E]\!] = \emptyset \\ \quad | \quad \mathtt{X} & & \mathrm{var}[\![E]\!] = \{\mathtt{X}\} \\ \quad | \quad ? & & \mathrm{var}[\![E]\!] = \emptyset \\ \quad | \quad E_1 - E_2 & & \mathrm{var}[\![E]\!] = \mathrm{var}[\![E_1]\!] \cup \mathrm{var}[\![E_2]\!] \end{array}$$

3.1.2 Syntax of Boolean Expressions $B \in \mathbb{B}$

The attribute $\mathrm{var}[\![B]\!]$ is the set of programming variables appearing in the boolean expression $B \in \mathbb{B}$.

$$\begin{array}{lll} B ::= E_1 < E_2 & & \mathrm{var}[\![B]\!] = \mathrm{var}[\![E_1]\!] \cup \mathrm{var}[\![E_2]\!] \\ \quad | \quad \neg B_1 & & \mathrm{var}[\![B]\!] = \mathrm{var}[\![B_1]\!] \\ \quad | \quad B_1 \wedge B_2 & & \mathrm{var}[\![B]\!] = \mathrm{var}[\![B_1]\!] \cup \mathrm{var}[\![B_2]\!] \end{array}$$

3.1.3 Syntax of Commands $C \in \mathbb{C}$

Each command $C \in \mathbb{C}$ has labels $\mathrm{lab}[\![C]\!]$ to denote execution points. These labels include an entry label $\mathrm{at}[\![C]\!]$, an exit label $\mathrm{after}[\![C]\!]$ and possibly labels of sub-commands in $\mathrm{in}[\![C]\!]$. In the following attribute grammar defining commands $C \in \mathbb{C}$, we globally assume that:

$$\begin{gathered} \mathrm{lab}[\![C]\!] = \{\mathrm{at}[\![C]\!]\} \cup \mathrm{in}[\![C]\!] \cup \{\mathrm{after}[\![C]\!]\}, \\ \mathrm{at}[\![C]\!] \neq \mathrm{after}[\![C]\!], \\ \{\mathrm{at}[\![C]\!], \mathrm{after}[\![C]\!]\} \cap \mathrm{in}[\![C]\!] = \emptyset \ . \end{gathered}$$

The commands C use and modify global variables in $\mathrm{var}[\![C]\!]$.

$$
\begin{array}{lll}
C ::= \texttt{skip} & & \mathrm{var}[\![C]\!] = \emptyset \\
 & & \mathrm{in}[\![C]\!] = \emptyset \\
\mid \quad \texttt{X} := E & & \mathrm{var}[\![C]\!] = \{\texttt{X}\} \cup \mathrm{var}[\![E]\!] \\
 & & \mathrm{in}[\![C]\!] = \emptyset \\
\mid \quad C_1 \,;\, C_2 & & \mathrm{var}[\![C]\!] = \mathrm{var}[\![C_1]\!] \cup \mathrm{var}[\![C_2]\!] \\
 & & \mathrm{lab}[\![C]\!] = \mathrm{lab}[\![C_1]\!] \cup \mathrm{lab}[\![C_2]\!] \\
 & & \mathrm{lab}[\![C_1]\!] \cap \mathrm{lab}[\![C_2]\!] = \{\mathrm{after}[\![C_1]\!]\} = \{\mathrm{at}[\![C_2]\!]\} \\
 & & \mathrm{at}[\![C]\!] = \mathrm{at}[\![C_1]\!] \\
 & & \mathrm{after}[\![C]\!] = \mathrm{after}[\![C_2]\!] \\
\mid \quad \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 & & \mathrm{var}[\![C]\!] = \mathrm{var}[\![B]\!] \cup \mathrm{var}[\![C_1]\!] \cup \mathrm{var}[\![C_2]\!] \\
 & & \mathrm{in}[\![C]\!] = \mathrm{lab}[\![C_1]\!] \cup \mathrm{lab}[\![C_2]\!] \\
 & & \mathrm{lab}[\![C_1]\!] \cap \mathrm{lab}[\![C_2]\!] = \emptyset \\
\mid \quad \texttt{while}\ B\ \texttt{do}\ C_1 & & \mathrm{var}[\![C]\!] = \mathrm{var}[\![B]\!] \cup \mathrm{var}[\![C_1]\!] \\
 & & \mathrm{in}[\![C]\!] = \mathrm{lab}[\![C_1]\!]
\end{array}
$$

3.2 Concrete Reachability Semantics of the Programming Language

We let $\mathcal{D}$ be the domain of value of the programming variables $\texttt{X}$.

3.2.1 Semantics of Expressions $E \in \mathbb{E}$

The semantics $\mathcal{E}[\![E]\!] \in \mathcal{M} \mapsto \wp(\mathcal{D})$ of expressions $E \in \mathbb{E}$ is defined on all memory states $\mathcal{M} \triangleq \mathcal{V} \mapsto \mathcal{D}$ where $\mathcal{V}$ is any set of programming variables including all such variables appearing in E, that is $\mathrm{var}[\![E]\!] \subseteq \mathcal{V}$. Because of the random choice ?, for all memory states $m \in \mathcal{M}$, $\mathcal{E}[\![E]\!]m$ has to be chosen as a set of possible values in $\wp(\mathcal{D})$. In denotational semantics style [51], we define:

$$\mathcal{E}[\![E]\!]m \triangleq \text{match } E \text{ with}$$

$$
\begin{array}{rl}
| & \mathtt{1} \to \{1\} \\
| & ? \to \mathcal{D} \\
| & \mathtt{X} \to \{m(\mathtt{X})\} \\
| & E_1 - E_2 \to \{v_1 - v_2 \mid v_1 \in \mathcal{E}[\![E_1]\!]m \wedge v_2 \in \mathcal{E}[\![E_2]\!]m\}
\end{array}
$$

Defining the judgement

$$m \vdash E \Rightarrow v \triangleq v \in \mathcal{E}[\![E]\!]m$$

this can be written in structured operational semantics style [49] as:

$$
\begin{array}{cc}
m \vdash \mathtt{1} \Rightarrow 1 & m \vdash ? \Rightarrow v, \quad v \in \mathcal{D} \\
m \vdash \mathtt{X} \Rightarrow m(\mathtt{X}) & \dfrac{m \vdash E_1 \Rightarrow v_1 \qquad m \vdash E_2 \Rightarrow v_2}{m \vdash E_1 - E_2 \Rightarrow v_1 - v_2}
\end{array}
$$

3.2.2 Semantics of Boolean Expressions $B \in \mathbb{B}$

The semantics $\mathcal{B}[\![B]\!] \in \wp(\mathcal{M})$ and $\overline{\mathcal{B}}[\![B]\!] \in \wp(\mathcal{M})$ of boolean expressions $B \in \mathbb{B}$ is defined for memory states in $\mathcal{M} \triangleq \mathcal{V} \mapsto \mathcal{D}$ where $\mathcal{V}$ is any set of programming variables including all such variables appearing in B, that is $\mathrm{var}[\![B]\!] \subseteq \mathcal{V}$. $\mathcal{B}[\![B]\!]$ defines the set of memory states in which B may be true while $\overline{\mathcal{B}}[\![B]\!]$ defines the set of memory states in which B may be false. We have $\mathcal{B}[\![B]\!] \cup \overline{\mathcal{B}}[\![B]\!] = \mathcal{M}$ but may be $\mathcal{B}[\![B]\!] \cap \overline{\mathcal{B}}[\![B]\!] \neq \emptyset$ because of non-determinism as in $? < \mathtt{1}$. We define:

$$
\begin{array}{rl}
\mathcal{B}[\![E_1 < E_2]\!] & \triangleq \{m \in \mathcal{M} \mid \exists x_1, x_2 \in \mathcal{D} : x_1 \in \mathcal{E}[\![E_1]\!]m \wedge x_2 \in \mathcal{E}[\![E_2]\!]m \wedge x_1 < x_2\} \\
\mathcal{B}[\![\neg B]\!] & \triangleq \overline{\mathcal{B}}[\![B]\!] \\
\mathcal{B}[\![B_1 \wedge B_2]\!] & \triangleq \mathcal{B}[\![B_1]\!] \cap \mathcal{B}[\![B_2]\!] \\
\overline{\mathcal{B}}[\![E_1 < E_2]\!] & \triangleq \{m \in \mathcal{M} \mid \exists x_1, x_2 \in \mathcal{D} : x_1 \in \mathcal{E}[\![E_1]\!]m \wedge x_2 \in \mathcal{E}[\![E_2]\!]m \wedge x_1 \geq x_2\} \\
\overline{\mathcal{B}}[\![\neg B_1]\!] & \triangleq \mathcal{B}[\![B]\!] \\
\overline{\mathcal{B}}[\![B_1 \wedge B_2]\!] & \triangleq \overline{\mathcal{B}}[\![B_1]\!] \cup \overline{\mathcal{B}}[\![B_2]\!]
\end{array}
$$

By defining the judgements:

$$
\begin{array}{c}
m \Vdash B \triangleq m \in \mathcal{B}[\![B]\!] \\
m \nVdash B \triangleq m \in \overline{\mathcal{B}}[\![B]\!]
\end{array}
$$

the definition of $\mathcal{B}$ and $\overline{\mathcal{B}}$ can also be rephrased as:

$$\frac{m \vdash E_1 \Rightarrow v_1 \quad m \vdash E_2 \Rightarrow v_2 \quad v_1 < v_2}{m \Vdash E_1 < E_2} \qquad \frac{m \vdash E_1 \Rightarrow v_1 \quad m \vdash E_2 \Rightarrow v_2 \quad v_1 \geq v_2}{m \nVdash E_1 < E_2}$$

$$\frac{m \nVdash B}{m \Vdash \neg B} \qquad \frac{m \Vdash B}{m \nVdash \neg B}$$

$$\frac{m \Vdash B_1 \quad m \Vdash B_2}{m \Vdash B_1 \wedge B_2} \qquad \frac{m \nVdash B_1}{m \nVdash B_1 \wedge B_2} \quad \frac{m \nVdash B_2}{m \nVdash B_1 \wedge B_2}$$

3.2.3 Reachability Semantics of Commands $C \in \mathbb{C}$

Given a domain $\mathcal{D}$ of value of the programming variables `X`, the reachability semantics $\mathcal{R}[\![C]\!]$ of a command $E \in \mathbb{E}$ is defined on any set of states $\Sigma = \langle \mathcal{L}, \mathcal{M} \rangle$ such that $\mathcal{L}$ is a set of labels/program points/control states and $\mathcal{M} \triangleq \mathcal{V} \mapsto \mathcal{D}$ is a set of memory states on variables $\mathcal{V}$ chosen such that $\mathrm{lab}[\![C]\!] \subseteq \mathcal{L}$ and $\mathrm{var}[\![C]\!] \subseteq \mathcal{V}$. The reachability semantics for the command C is then $\langle \Sigma, \mathcal{R}[\![C]\!] \rangle$. For a program C, we can choose $\Sigma = \langle \mathrm{lab}[\![C]\!], \mathrm{var}[\![C]\!] \mapsto \mathcal{D} \rangle$. We have:

$$\mathcal{R}[\![C]\!] \in \wp(\Sigma) \mapsto \wp(\Sigma) \tag{2}$$

$\mathcal{R}[\![C]\!]P \triangleq$ match C with

$$\mid \ \texttt{skip} \ \rightarrow \ P \cup \{\langle \mathrm{after}[\![C]\!], m\rangle \mid \langle \mathrm{at}[\![C]\!], m\rangle \in P\} \tag{3}$$

$$\mid \ \texttt{X} := E \ \rightarrow \ P \cup \{\langle \mathrm{after}[\![C]\!], m[\texttt{X} := v]\rangle \mid \langle \mathrm{at}[\![C]\!], m\rangle \in P \wedge v \in \mathcal{E}[\![E]\!]m\} \tag{4}$$

$$\mid \ C_1 \,;\, C_2 \ \rightarrow \ \text{let } P_1 = \mathcal{R}[\![C_1]\!]P \text{ in} \tag{5}$$

$$P_1 \cup \mathcal{R}[\![C_2]\!]\{\langle \mathrm{at}[\![C_2]\!], m\rangle \mid \langle \mathrm{after}[\![C_1]\!], m\rangle \in P_1\}$$

$$\mid \ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2 \quad \rightarrow \tag{6}$$

$$\begin{array}{l} \text{let} \quad P_1 = \mathcal{R}[\![C_1]\!]\{\langle \mathrm{at}[\![C_1]\!], m\rangle \mid \langle \mathrm{at}[\![C]\!], m\rangle \in P \wedge m \in \mathcal{B}[\![B]\!]\} \\ \text{and } P_2 = \mathcal{R}[\![C_2]\!]\{\langle \mathrm{at}[\![C_2]\!], m\rangle \mid \langle \mathrm{at}[\![C]\!], m\rangle \in P \wedge m \in \overline{\mathcal{B}}[\![B]\!]\} \\ \text{and } P_e = \{\langle \mathrm{after}[\![C]\!], m\rangle \mid \langle \mathrm{after}[\![C_1]\!], m\rangle \in P_1 \vee \langle \mathrm{after}[\![C_2]\!], m\rangle \in P_2\} \text{ in} \\ \quad P \cup P_1 \cup P_2 \cup P_e \end{array}$$

$$\mid \ \texttt{while}\ B\ \texttt{do}\ C_1 \ \rightarrow \tag{7}$$

$$\begin{array}{l}\text{let } P_1 = \mathrm{lfp}^{\subseteq}\, \lambda X \bullet \mathcal{R}[\![C_1]\!](\{\langle \mathrm{at}[\![C_1]\!],\, m\rangle \mid (\langle \mathrm{at}[\![C]\!],\, m\rangle \in P \vee \\ \qquad\qquad\qquad\qquad \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in X) \wedge m \in \mathcal{B}[\![B]\!]\}) \quad \text{in} \\ P \cup P_1 \cup \{\langle \mathrm{after}[\![C]\!],\, m\rangle \mid (\langle \mathrm{at}[\![C]\!],\, m\rangle \in P \vee \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in P_1) \wedge m \in \overline{\mathcal{B}}[\![B]\!]\}\end{array}$$

Let us define:

$$m \in P \downarrow \ell \triangleq \langle \ell,\, m\rangle \in P$$
$$P \vdash C \stackrel{*}{\Rightarrow} \ell, m \triangleq \langle \ell,\, m\rangle \in \mathcal{R}[\![C]\!]P$$

so as to define the reachability semantics in rule-based form:

$$\frac{m \in P \downarrow \ell}{P \vdash C \stackrel{*}{\Rightarrow} \ell, m}$$

$$\frac{m \in P \downarrow \mathrm{at}[\![\texttt{skip}]\!]}{P \vdash \texttt{skip} \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{skip}]\!], m}$$

$$\frac{m \in P \downarrow \mathrm{at}[\![\texttt{X} := \texttt{E}]\!] \qquad m \vdash E \Rightarrow v}{P \vdash \texttt{X} := \texttt{E} \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{X} := \texttt{E}]\!], m[\texttt{X} := v]}$$

$$\frac{P \vdash C_1 \stackrel{*}{\Rightarrow} \ell, m}{P \vdash C_1\,;C_2 \stackrel{*}{\Rightarrow} \ell, m}$$

$$\frac{P \vdash C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![C_1]\!], m \qquad \{\langle \mathrm{at}[\![C_2]\!],\, m\rangle\} \vdash C_2 \stackrel{*}{\Rightarrow} \ell, m'}{P \vdash C_1\,;C_2 \stackrel{*}{\Rightarrow} \ell, m'}$$

$$\frac{\begin{array}{c} m \in P \downarrow \mathrm{at}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] \qquad m \Vdash B \\ \{\langle \mathrm{at}[\![C_1]\!],\, m\rangle\} \vdash C_1 \stackrel{*}{\Rightarrow} \ell, m' \end{array}}{P \vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \stackrel{*}{\Rightarrow} \ell, m'}$$

$$\frac{\begin{array}{c} m \in P \downarrow \mathrm{at}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] \qquad m \Vdash B \\ \{\langle \mathrm{at}[\![C_1]\!],\, m\rangle\} \vdash C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![C_1]\!], m' \end{array}}{P \vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!], m'}$$

$$\frac{\begin{array}{c} m \in P \downarrow \mathrm{at}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] \qquad m \nVdash B \\ \{\langle \mathrm{at}[\![C_2]\!],\, m\rangle\} \vdash C_2 \stackrel{*}{\Rightarrow} \ell, m' \end{array}}{P \vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \stackrel{*}{\Rightarrow} \ell, m'}$$

$$\frac{\begin{array}{c} m \in P \downarrow \mathrm{at}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!] \qquad m \nVdash B \\ \{\langle \mathrm{at}[\![C_2]\!],\, m\rangle\} \vdash C_2 \stackrel{*}{\Rightarrow} \mathrm{after}[\![C_2]\!], m' \end{array}}{P \vdash \texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2 \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{if } B \texttt{ then } C_1 \texttt{ else } C_2]\!], m'}$$

$$\frac{m \in P \downarrow \mathrm{at}[\![\texttt{while}\ B\ \texttt{do}\ C_1]\!] \qquad m \Vdash B}{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{at}[\![C_1]\!], m}$$

$$\frac{m \in P \downarrow \mathrm{at}[\![\texttt{while}\ B\ \texttt{do}\ C_1]\!] \qquad m \nVdash B}{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{while}\ B\ \texttt{do}\ C_1]\!], m}$$

$$\frac{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![C_1]\!], m \qquad m \Vdash B}{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{at}[\![C_1]\!], m} \tag{8}$$

$$\frac{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![C_1]\!], m \qquad m \nVdash B}{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \mathrm{after}[\![\texttt{while}\ B\ \texttt{do}\ C_1]\!], m} \tag{9}$$

$$\frac{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \ell, m \qquad \{\langle \ell,\ m\rangle\} \vdash C_1 \stackrel{*}{\Rightarrow} \ell', m'}{P \vdash \texttt{while}\ B\ \texttt{do}\ C_1 \stackrel{*}{\Rightarrow} \ell', m'} \tag{10}$$

The rules are by structural induction on the command syntax but (10), (9) and (8), which are self-referential, and therefore should be interpreted as fixpoints [23].

3.2.4 Ground Abstract Predicates

Predicate abstraction is defined by a set $\mathfrak{P}$ of syntactic predicates which, for simplicity, we choose to be boolean expressions $\mathfrak{P} \subseteq \mathbb{B}$. We define $\mathrm{var}[\![\mathfrak{P}]\!] \triangleq \bigcup_{\mathfrak{p}\in\mathfrak{P}} \mathrm{var}[\![\mathfrak{p}]\!]$.

Example 13. For the sign analysis [19] of a command $C \in \mathbb{C}$, one would choose

$$\mathfrak{P} = \{\mathrm{tt}, \mathrm{ff}\} \cup \bigcup_{\mathtt{X}\in\mathrm{var}[\![C]\!]} \{\mathtt{X} \leq 0, \mathtt{X} < 0, \mathtt{X} = 0, \mathtt{X} \neq 0, \mathtt{X} > 0, \mathtt{X} \geq 0\}$$

□

The set $\mathfrak{P}$ may be infinite, as shown by the following example.

Example 14. For Kildall's constant propagation [40] for a command $C \in \mathbb{C}$, one would represent the lattice of Fig. 18 by

$$\mathfrak{P} = \{\mathrm{tt}, \mathrm{ff}\} \cup \bigcup_{\mathtt{X}\in\mathrm{var}[\![C]\!]} \bigcup_{v\in\mathcal{D}} \{\mathtt{X} = v\}$$

□

Predicate abstraction uses a prover to prove theorems $t \in \mathbb{T}$ with interpretation

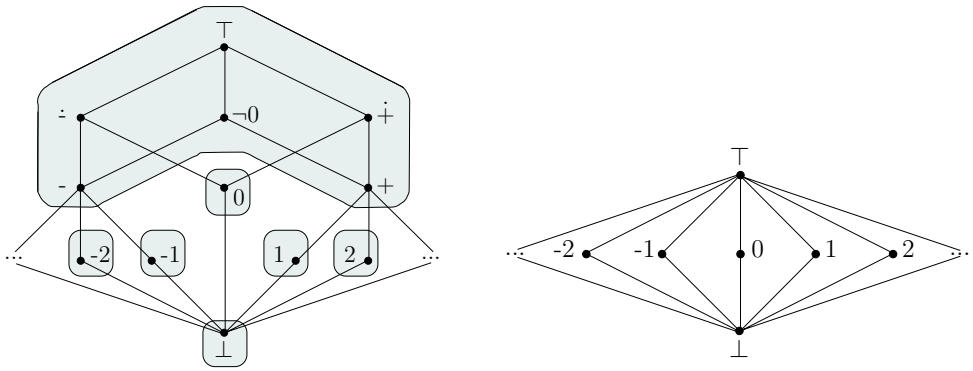


Fig. 18. Kildall's constant propagation abstraction

$\mathcal{I} \in \mathbb{T} \mapsto \wp(\mathcal{M})$ assigning a semantics/interpretation/meaning $\mathcal{I}[\![t]\!]$ to all syntactic predicates $t \in \mathbb{T}$. The syntax is as follows:

$$
\begin{array}{lll}
t ::= \mathtt{p} \quad (\text{where } \mathtt{p} \in \mathfrak{P}) & & \mathrm{var}[\![t]\!] = \mathrm{var}[\![\mathtt{p}]\!] \\
\quad | \quad \mathtt{tt} & & \mathrm{var}[\![t]\!] = \emptyset \\
\quad | \quad \mathtt{ff} & & \mathrm{var}[\![t]\!] = \emptyset \\
\quad | \quad \mathtt{X} \in E & & \mathrm{var}[\![t]\!] = \{\mathtt{X}\} \cup \mathrm{var}[\![E]\!] \\
\quad | \quad \neg t_1 & & \mathrm{var}[\![t]\!] = \mathrm{var}[\![t_1]\!] \\
\quad | \quad t_1 \Rightarrow t_2 & & \mathrm{var}[\![t]\!] = \mathrm{var}[\![t_1]\!] \cup \mathrm{var}[\![t_2]\!] \\
\quad | \quad \bigwedge\limits_{i \in \Delta} t_i & & \mathrm{var}[\![t]\!] = \bigcup\limits_{i \in \Delta} \mathrm{var}[\![t_i]\!] \\
\quad | \quad \forall \mathtt{X} : t_1 & & \mathrm{var}[\![t]\!] = \mathrm{var}[\![t_1]\!] \setminus \{\mathtt{X}\}
\end{array}
$$

with semantics $\mathcal{I}[\![t]\!] \in \wp(\mathcal{M})$:

$$
\begin{aligned}
\mathcal{I}[\![\mathtt{p}]\!] &\triangleq \mathcal{B}[\![\mathtt{p}]\!], \\
\mathcal{I}[\![\mathtt{tt}]\!] &\triangleq \mathcal{M}, \\
\mathcal{I}[\![\mathtt{ff}]\!] &\triangleq \emptyset, \\
\mathcal{I}[\![\mathtt{X} \in E]\!] &\triangleq \{m \in \mathcal{M} \mid m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m\}, \\
\mathcal{I}[\![\neg t]\!] &\triangleq \{m \in \mathcal{M} \mid m \notin \mathcal{I}[\![t]\!]\}, \\
\mathcal{I}[\![\bigwedge_{i \in \Delta} t_i]\!] &\triangleq \bigcap_{i \in \Delta} \mathcal{I}[\![t_i]\!], \\
\mathcal{I}[\![t_1 \Rightarrow t_2]\!] &\triangleq \{m \in \mathcal{M} \mid m \notin \mathcal{I}[\![t_1]\!] \vee m \in \mathcal{I}[\![t_2]\!]\}, \\
\mathcal{I}[\![\forall \mathtt{X} : t]\!] &\triangleq \{m \in \mathcal{M} \mid \forall v \in \mathcal{D} : m[\mathtt{X} := v] \in \mathcal{I}[\![t]\!]\} \ .
\end{aligned}
$$

Variable substitution $t[\mathtt{X}/\mathtt{X}']$ is defined as usual with renaming of conflicting

dummy variables such that:

$$\begin{aligned}
\mathtt{X} \notin \mathrm{var}[\![t]\!] &\Rightarrow t[\mathtt{X}/\mathtt{X}'] = t, \\
\mathcal{E}[\![E[\mathtt{X}/\mathtt{X}']]\!]m &= \mathcal{E}[\![E]\!]m[\mathtt{X} := m(\mathtt{X}')], \qquad (11) \\
\mathtt{X} \neq \mathtt{Y} \wedge \mathtt{Y} \notin \mathrm{var}[\![t]\!] &\Rightarrow \mathcal{I}[\![\forall \mathtt{X} : t]\!] = \mathcal{I}[\![\forall \mathtt{Y} : (t[\mathtt{X}/\mathtt{Y}])]\!], \\
\mathtt{Z} \notin \mathrm{var}[\![t]\!] \cup \{\mathtt{X}, \mathtt{Y}\} &\Rightarrow \mathcal{I}[\![(\forall \mathtt{X} : t)[\mathtt{Y}/\mathtt{X}]]\!] = \mathcal{I}[\![\forall \mathtt{Z} : (t[\mathtt{X}/\mathtt{Z}][\mathtt{Y}/\mathtt{X})]]\!], \\
\mathcal{I}[\![t[\mathtt{X}/\mathtt{X}']]\!] &= \{m \mid m[\mathtt{X} := m(\mathtt{X}')] \in \mathcal{I}[\![t]\!]\} \ . \qquad (12)
\end{aligned}$$

The prover is assumed to be sound in that:

$$\forall t \in \mathbb{T} : \mathrm{prover}[\![t]\!] \ \Rightarrow \ (\mathcal{I}[\![t]\!] = \mathcal{M}) \ . \qquad (13)$$

(The inverse is not valid since provers are incomplete.)

3.2.5 Ground Predicate Abstraction

Given a set of states in $\wp(\Sigma)$ where $\Sigma = \langle \mathcal{L}, \mathcal{M} \rangle$, we can use an isomorphic representation associating sets of memory states to labels/program points/control states thank to the following correspondence:

$$\langle \wp(\mathcal{L} \times \mathcal{M}), \subseteq \rangle \underset{\alpha_\downarrow}{\overset{\gamma_\downarrow}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathcal{M}), \dot{\subseteq} \rangle \qquad (14)$$

where

$$\begin{aligned}
\alpha_\downarrow(P) &\triangleq \lambda \ell \bullet \{m \mid \langle \ell, m \rangle \in P\} \qquad (15) \\
\gamma_\downarrow(Q) &\triangleq \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge m \in Q_\ell\}
\end{aligned}$$

and $\dot{\subseteq}$ is the pointwise ordering $Q \dot{\subseteq} Q'$ if and only if $\forall \ell \in \mathcal{L} : Q_\ell \subseteq Q'_\ell$.

A memory state property $Q \in \wp(\mathcal{M})$ is approximated by the subset of predicates p of $\mathfrak{P}$ which holds when Q holds (formally $Q \subseteq \mathcal{B}[\![p]\!]$). This defines a Galois connection:

$$\langle \wp(\mathcal{M}), \subseteq \rangle \underset{\alpha_\mathfrak{P}}{\overset{\gamma_\mathfrak{P}}{\leftrightarrows}} \langle \wp(\mathfrak{P}), \supseteq \rangle \qquad (16)$$

where:

$$\alpha_{\mathfrak{P}}(Q) \triangleq \{\mathfrak{p} \in \mathfrak{P} \mid Q \subseteq \mathcal{B}[\![\mathfrak{p}]\!]\} \tag{17}$$
$$\gamma_{\mathfrak{P}}(P) \triangleq \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in P\} \tag{18}$$
$$= \mathcal{I}[\![\bigwedge P]\!] \tag{19}$$

Observe that in general $\gamma_{\mathfrak{P}}$ is not one-to-one (e.g. $\gamma_{\mathfrak{P}}(\{\mathtt{X} = 1, \mathtt{X} \geq 1\}) = \gamma_{\mathfrak{P}}(\{\mathtt{X} = 1\})$ so $\alpha_{\mathfrak{P}}$ is not onto[11] .

By pointwise extension, we have:

$$\langle \mathcal{L} \mapsto \wp(\mathcal{M}),\ \dot{\subseteq} \rangle \underset{\dot{\alpha}_{\mathfrak{P}}}{\overset{\dot{\gamma}_{\mathfrak{P}}}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathfrak{P}),\ \dot{\supseteq} \rangle$$

where:

$$\dot{\alpha}_{\mathfrak{P}}(Q) \triangleq \lambda \ell \cdot \alpha_{\mathfrak{P}}(Q_\ell) \tag{20}$$
$$\dot{\gamma}_{\mathfrak{P}}(P) \triangleq \lambda \ell \cdot \gamma_{\mathfrak{P}}(P_\ell) \tag{21}$$
$$P \dot{\supseteq} P' \triangleq \forall \ell \in \mathcal{L} : P_\ell \supseteq P'_\ell$$

By composition, we get:

$$\langle \wp(\mathcal{L} \times \mathcal{M}),\ \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathfrak{P}),\ \dot{\supseteq} \rangle \tag{22}$$

where:

$$\alpha(P) \triangleq \dot{\alpha}_{\mathfrak{P}} \circ \alpha_{\downarrow}(P) \tag{23}$$
$$= \lambda \ell \cdot \{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell,\ m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}$$
$$\gamma(Q) \triangleq \gamma_{\downarrow} \circ \dot{\gamma}_{\mathfrak{P}} \tag{24}$$
$$= \{\langle \ell,\ m \rangle \mid \ell \in \mathcal{L} \wedge m \in \gamma_{\mathfrak{P}}(Q_\ell)\} \tag{25}$$
$$= \{\langle \ell,\ m \rangle \mid \ell \in \mathcal{L} \wedge \forall \mathfrak{p} \in Q_\ell : m \in \mathcal{I}[\![\mathfrak{p}]\!]\}$$

3.2.6 On Boolean Ground Predicate Abstraction

If $\mathfrak{P}$ is assumed to be finite then characteristic functions of subsets of $\mathfrak{P}$ can be encoded as boolean vectors thus later allowing for the reuse of model-checker

[11] In a Galois connection $\langle L,\ \leq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M,\ \sqsubseteq \rangle$, α is onto if and only if γ is one-to-one if and only if $\gamma \circ \alpha$ is the identity so, by duality, γ is onto if and only if α is one-to-one if and only if $\alpha \circ \gamma$ is the identity.

to solve fixpoint equations. However, this encoding of sets is very expensive to encode small subsets of large subsets and cannot be used to encode infinite sets.

Counter Example 15. For Kildall's constant propagation [40] as already considered in Ex. 14, the encoding of subsets of $\mathfrak{P}$ by bit vectors is impossible when the domain of values $\mathcal{D}$ is infinite or very large. With this encoding (otherwise useful to reuse boolean model-checkers), $\mathfrak{P}$ must be restricted to the finite small set of those constants which are useful during the analysis of the command C. These useful constants have to be discovered by some other means (e.g. user assistance or abstraction refinement), which are much less automatic and/or efficient than Kildall's constant propagation algorithm. □

We let $\mathfrak{B} = \{\mathrm{tt}, \mathrm{ff}\}$ be the set of booleans with $\mathrm{ff} \Rightarrow \mathrm{ff} \Rightarrow \mathrm{tt} \Rightarrow \mathrm{tt}$. Assuming $\mathfrak{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$, we have:

$$\langle \wp(\mathfrak{P}),\ \supseteq \rangle \underset{\alpha_b}{\overset{\gamma_b}{\leftrightarrows}} \langle \prod_{i=1}^{k} \mathfrak{B},\ \Leftarrow \rangle \tag{26}$$

where:

$$\begin{aligned} \alpha_b(P) &\triangleq \prod_{i=1}^{k} (\mathfrak{p}_i \in P) \\ \gamma_b(Q) &\triangleq \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge Q_i\} \\ Q \dot{\Leftarrow} Q' &\triangleq \forall i : 1 \leq i \leq k : Q_i \Leftarrow Q'_i \end{aligned}$$

By pointwise extension, we have:

$$\langle \mathcal{L} \mapsto \wp(\mathfrak{P}),\ \dot{\supseteq} \rangle \underset{\dot{\alpha}_b}{\overset{\dot{\gamma}_b}{\leftrightarrows}} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathfrak{B},\ \dot{\Leftarrow} \rangle$$

where:

$$\begin{aligned} \dot{\alpha}_b(P) &\triangleq \lambda \ell \bullet\ \alpha_b(P_\ell) \\ \dot{\gamma}_b(Q) &\triangleq \lambda \ell \bullet\ \gamma_b(Q_\ell) \\ Q \dot{\Leftarrow} Q' &\triangleq \forall \ell \in \mathcal{L} : Q_\ell \Leftarrow Q'_\ell \end{aligned}$$

By composition, we get the variant:

$$\langle \wp(\mathcal{L} \times \mathcal{M}),\ \subseteq \rangle \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathfrak{B},\ \dot{\Leftarrow} \rangle \tag{27}$$

where:

$$\begin{aligned}
\alpha'(P) &\triangleq \dot{\alpha}_b \circ \alpha(P) \\
&= \lambda \ell \cdot \prod_{i=1}^{k} (\{m \mid \langle \ell, m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}_i]\!]) \\
\gamma'(Q) &\triangleq \gamma \circ \dot{\gamma}_b(Q) \\
&= \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge m \in \mathcal{I}[\![\bigwedge_{i=1}^{k} (Q_\ell)_i \Rightarrow \mathfrak{p}_i]\!]\}
\end{aligned}$$

Because of the limitation to a finite set $\mathfrak{P}$ of abstract predicates introduced by the boolean abstraction (27), we prefer to use the set abstraction (22) for which efficient encodings of finite subsets do exist [7, 10] and the corresponding fixpoint computation engine can be efficiently implemented [7].

3.2.7 Abstract Reachability Semantics of Commands $C \in \mathbb{C}$

Given a set of abstract predicates $\mathfrak{P}$, the abstract reachability semantics of command $C \in \mathbb{C}$ is

$$\begin{aligned}
\overline{\mathcal{R}}[\![C]\!] &\in \wp(\mathfrak{P}) \mapsto (\mathrm{lab}[\![C]\!] \mapsto \wp(\mathfrak{P})) \\
\overline{\mathcal{R}}[\![C]\!]\overline{P} &= \alpha(\mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})))
\end{aligned}$$

that is the abstraction of the reachable states of C from its entry point $\mathrm{at}[\![C]\!]$ in initial memory states $m \in \gamma_{\mathfrak{P}}(\overline{P})$.

Because of undecidability, whence theorem prover incompleteness, we look for a $\dot{\supseteq}$-over approximation $\overline{\mathcal{R}}[\![C]\!]\overline{P}$ such that:

$$\alpha(\mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))) \mathrel{\dot{\supseteq}} \overline{\mathcal{R}}[\![C]\!]\overline{P} \tag{28}$$

$$\Leftrightarrow \quad \mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \subseteq \gamma(\overline{\mathcal{R}}[\![C]\!]\overline{P}) \tag{29}$$

We proceed by structural induction on the syntax of commands $C \in \mathbb{C}$.

— For the null command , we have

$$\begin{aligned}
& \overline{\mathcal{R}}[\![\texttt{skip}]\!]\overline{P} \\
= \;& \alpha(\mathcal{R}[\![\texttt{skip}]\!](\{\mathrm{at}[\![\texttt{skip}]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))) && \wr \text{def. } \overline{\mathcal{R}} \wr \\
= \;& \text{let } P = \{\mathrm{at}[\![\texttt{skip}]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}) \text{ in} \\
& \quad \alpha(P \cup \{\langle \mathrm{after}[\![C]\!], m \rangle \mid \langle \mathrm{at}[\![C]\!], m \rangle \in P\}) && \wr \text{def. } \mathcal{R} \wr
\end{aligned}$$

$=$ let $P = \{\mathrm{at}[\![\texttt{skip}]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})$ in
$\quad \alpha(P) \dot{\cap} \alpha(\{\langle \mathrm{after}[\![C]\!],\, m\rangle \mid \langle \mathrm{at}[\![C]\!],\, m\rangle \in P\})$ ⟅by (22), so that α is a complete join morphism⟆
$=$ $\alpha(\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \alpha(\{\mathrm{after}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))$ ⟅by def. P⟆

Let us go on with the term $\alpha(P)$ where $P = \{\mathrm{loc}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})$ with $\mathrm{loc} \in \{\mathrm{at}, \mathrm{after}\}$ (we define $(\!|\mathrm{tt} \mathbin{?} t : f|\!) = t$, $(\!|\mathrm{ff} \mathbin{?} t : f|\!) = f$ and $(\!|b_1 \mathbin{?} t_1 \parallel b_2 \mathbin{?} t_2 : f|\!) = (\!|b_1 \mathbin{?} t_1 : (\!|b_2 \mathbin{?} t_2 : f|\!)|\!)$).

$\alpha(\{\mathrm{loc}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))$
$=$ $\lambda\ell\bullet\{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell,\, m\rangle \in \{\mathrm{loc}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}$ ⟅def. α⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \gamma_{\mathfrak{P}}(\overline{P}) \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} : \{\mathfrak{p} \in \mathfrak{P} \mid \emptyset \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}|\!)$ ⟅def. conditional⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in \overline{P}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} : \mathfrak{P}|\!)$ ⟅def. $\gamma_{\mathfrak{P}}$ and $\subseteq$⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \bigcap\{\mathcal{I}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in \overline{P}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} : \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \mathcal{I}[\![\bigwedge \overline{P}] \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} : \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \mathcal{I}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\} : \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆
$\dot{\supseteq}$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\} : \mathfrak{P}|\!)$ ⟅prover soundness (13)⟆
$\dot{\supseteq}$ $\lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \overline{P} : \mathfrak{P}|\!)$ (30)
⟅which is less precise but avoids a call to the prover⟆

We have shown that for $\mathrm{loc} \in \{\mathrm{at}, \mathrm{after}\}$:

$$\alpha(\{\mathrm{loc}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \;\dot{\supseteq}\; \lambda\ell\bullet(\!|\ell = \mathrm{loc}[\![C]\!] \mathbin{?} \overline{P} : \mathfrak{P}|\!) \tag{32}$$

It follows that:

$\overline{\mathcal{R}}[\![\texttt{skip}]\!]\overline{P}$
$=$ $\alpha(\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \alpha(\{\mathrm{after}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))$ ⟅by def. P⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{at}[\![C]\!] \mathbin{?} \overline{P} : \mathfrak{P}|\!) \dot{\cap} \lambda\ell\bullet(\!|\ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{P} : \mathfrak{P}|\!)$ ⟅by (32)⟆
$=$ $\lambda\ell\bullet(\!|\ell = \mathrm{at}[\![C]\!] \mathbin{?} \overline{P} \parallel \ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{P} : \mathfrak{P}|\!)$ ⟅by $\mathrm{at}[\![C]\!] \neq \mathrm{after}[\![C]\!]$ and def. $\dot{\cap}$⟆

— For the assignment $\mathtt{X} := E$, we have

$\overline{\mathcal{R}}[\![\mathtt{X} := E]\!]\overline{P}$
$=$ $\alpha(\mathcal{R}[\![\mathtt{X} := E]\!](\{\mathrm{at}[\![\mathtt{X} := E]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})))$ ⟅def. $\overline{\mathcal{R}}$⟆

$= \quad \text{let } P = \{\text{at}[\![\mathtt{X} := E]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}) \text{ in}$
$\quad\quad \alpha(P \cup \{\langle \text{after}[\![C]\!],\, m[\mathtt{X} := v]\rangle \mid \langle \text{at}[\![C]\!],\, m\rangle \in P \wedge v \in \mathcal{E}[\![E]\!]m\})$ ⟅def. $\mathcal{R}$⟆
$= \quad \text{let } P = \{\text{at}[\![\mathtt{X} := E]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}) \text{ in}$
$\quad\quad \alpha(P) \dot{\cap} \alpha(\{\langle \text{after}[\![C]\!],\, m[\mathtt{X} := v]\rangle \mid \langle \text{at}[\![C]\!],\, m\rangle \in P \wedge v \in \mathcal{E}[\![E]\!]m\})$
$\quad\quad$ ⟅(22), so that α is a complete join morphism⟆

The first term has already been evaluated in (32). Given $P = \{\text{at}[\![\mathtt{X} := E]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})$, the second term is

$\quad\quad \alpha(\{\langle \text{after}[\![\mathtt{X} := E]\!],\, m[\mathtt{X} := v]\rangle \mid \langle \text{at}[\![\mathtt{X} := E]\!],\, m\rangle \in P \wedge v \in \mathcal{E}[\![E]\!]m\})$
$= \quad \alpha(\{\langle \text{after}[\![\mathtt{X} := E]\!],\, m[\mathtt{X} := v]\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \wedge v \in \mathcal{E}[\![E]\!]m\})$ ⟅def. P⟆
$= \quad \lambda\ell\bullet\{\mathfrak{p} \in \mathfrak{P} \mid \{m' \mid \langle \ell,\, m'\rangle \in \{\langle \text{after}[\![\mathtt{X} := E]\!],\, m[\mathtt{X} := v]\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \wedge v \in \mathcal{E}[\![E]\!]m\}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}$ ⟅def. α⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \wedge v \in \mathcal{E}[\![E]\!]m\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. conditional and $\subseteq$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid m \in \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in \overline{P}\} \wedge v \in \mathcal{E}[\![E]\!]m\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\gamma_{\mathfrak{P}}$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid m \in \bigcap\{\mathcal{I}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in \overline{P}\} \wedge v \in \mathcal{E}[\![E]\!]m\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid m \in \mathcal{I}[\![\bigwedge \overline{P}]\!] \wedge v \in \mathcal{E}[\![E]\!]m\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v'][\mathtt{X} := v] \mid m[\mathtt{X} := v'] \in \mathcal{I}[\![\bigwedge \overline{P}]\!] \wedge v \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v']\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by letting $v' = m(\mathtt{X})$ so that $m = m[\mathtt{X} := v']$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid \exists v' \in \mathcal{D} : m[\mathtt{X} := v'] \in \mathcal{I}[\![\bigwedge \overline{P}]\!] \wedge v \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v']\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅since $m[\mathtt{X} := v'][\mathtt{X} := v] = m[\mathtt{X} := v]$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m[\mathtt{X} := v] \mid m[\mathtt{X} := m(\mathtt{X}')] \in \mathcal{I}[\![\bigwedge \overline{P}]\!] \wedge v \in \mathcal{E}[\![E]\!]m[\mathtt{X} := m(\mathtt{X}')]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by letting $v' = m(\mathtt{X}')$ where $\mathtt{X}'$ is a fresh variable such that $\mathtt{X}' \notin \{\mathtt{X}\} \cup \text{var}[\![E]\!] \cup \text{var}[\![\mathfrak{P}]\!]$ so that $\mathtt{X}' \notin \text{var}[\![\overline{P}]\!]$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m' \mid m'[\mathtt{X} := m'(\mathtt{X}')] \in \mathcal{I}[\![\bigwedge \overline{P}]\!] \wedge m'(\mathtt{X}) \in \mathcal{E}[\![E]\!]m'[\mathtt{X} := m'(\mathtt{X}')]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by letting $m' = m[\mathtt{X} := v]$ so that $v = m'(\mathtt{X})$, $m(\mathtt{X}') = m'(\mathtt{X}')$ and $m[\mathtt{X} := m(\mathtt{X}')] = m'[\mathtt{X} := m'(\mathtt{X}')]$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X} := m(\mathtt{X}')] \in \mathcal{I}[\![(\bigwedge \overline{P})]\!] \wedge m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by letting $m = m'$ and $v = m'(\mathtt{X}')$⟆
$= \quad \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v][\mathtt{X} := m(\mathtt{X}')] \in \mathcal{I}[\![(\bigwedge \overline{P})]\!] \wedge m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅since $\mathtt{X}' \notin \text{var}[\![\overline{P}]\!]$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v][\mathtt{X} := m[\mathtt{X} := v](\mathtt{X}')] \in \mathcal{I}[\![(\bigwedge \overline{P})]\!] \wedge m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅since $\mathtt{X} \neq \mathtt{X}'$ so $m(\mathtt{X}') = m[\mathtt{X} := v](\mathtt{X}')$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \{m' \mid m'[\mathtt{X} := m'(\mathtt{X}')] \in \mathcal{I}[\![(\bigwedge \overline{P})]\!]\} \wedge m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by def. $\in$ while letting $m' = m[\mathtt{X} := v]$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \wedge m(\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := v]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by (12)⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \wedge m[\mathtt{X}' := v](\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X} := m[\mathtt{X}' := v](\mathtt{X}')]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $m[\mathtt{X} := v]$ and $\mathtt{X} \neq \mathtt{X}'$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \wedge m[\mathtt{X}' := v](\mathtt{X}) \in \mathcal{E}[\![E]\!]m[\mathtt{X}' := v][\mathtt{X} := m[\mathtt{X}' := v](\mathtt{X}')]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅since $\mathtt{X}' \notin \text{var}[\![E]\!]$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \wedge m[\mathtt{X}' := v] \in \{m' \in \mathcal{M} \mid m'(\mathtt{X}) \in \mathcal{E}[\![E]\!]m'[\mathtt{X} := m'(\mathtt{X}')]\}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\in$ while letting $m' = m[\mathtt{X}' := v]$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \cap \{m' \in \mathcal{M} \mid m'(\mathtt{X}) \in \mathcal{E}[\![E]\!]m'[\mathtt{X} := m'(\mathtt{X}')]\}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\cap$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \cap \{m' \in \mathcal{M} \mid m'(\mathtt{X}) \in \mathcal{E}[\![E[\mathtt{X}/\mathtt{X}']]\!]m'\}\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅by (11)⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![(\bigwedge \overline{P})[\mathtt{X}/\mathtt{X}']]\!] \cap \mathcal{I}[\![\mathtt{X} \in E[\mathtt{X}/\mathtt{X}']]\!]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![\bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']]\!]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. substitution⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \{m \in \mathcal{M} \mid \exists v \in \mathcal{D} : m[\mathtt{X}' := v] \in \mathcal{I}[\![\bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']]\!]\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \mathcal{I}[\![\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']]\!] \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆

$= \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \mathcal{I}[\![(\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']) \Rightarrow \mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆

$\dot{\supseteq}\ \lambda\ell\bullet(\!|\ell = \text{after}[\![\mathtt{X} := E]\!] \mathbin{?} \{\mathfrak{p} \in \mathfrak{P} \mid \text{prover}[\![(\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']) \Rightarrow \mathfrak{p}]\!]\} \mathbin{:} \mathfrak{P}|\!)$ ⟅def. $\mathcal{I}$⟆

Grouping both cases together, we have

$$\overline{\mathcal{R}}[\![\mathtt{X} := E]\!]\overline{P}$$

$= \lambda\ell\bullet(\!|\ \ell = \mathrm{at}[\![\mathtt{X} := E]\!] \ ?\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\} \ ;\ \mathfrak{P}|\!) \mathbin{\dot\cap} \lambda\ell\bullet(\!|\ \ell = \mathrm{after}[\![\mathtt{X} := E]\!] \ ?\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![(\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']) \Rightarrow \mathfrak{p}]\!]\} \ ;\ \mathfrak{P}|\!)$

$= \lambda\ell\bullet (\!|\ \ell = \mathrm{at}[\![C]\!] \ ?\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\}$
$\quad [\!]\ \ell = \mathrm{after}[\![\mathtt{X} := E]\!] \ ?\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![(\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}']) \Rightarrow \mathfrak{p}]\!]\}$
$\quad ;\ \mathfrak{P}|\!)$
〈by def. $\dot\cap$, conditional and $\mathrm{at}[\![C]\!] \neq \mathrm{after}[\![C]\!]$〉

— For the sequential composition $C_1 \,;\, C_2$, we have

$\overline{\mathcal{R}}[\![C_1 \,;\, C_2]\!]\overline{P}$
$= \alpha(\mathcal{R}[\![C_1 \,;\, C_2]\!](\{\mathrm{at}[\![C_1 \,;\, C_2]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})))$ 〈def. $\overline{\mathcal{R}}$〉
$=$ let $P_1 = \mathcal{R}[\![C_1]\!](\{\mathrm{at}[\![C_1 \,;\, C_2]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))$ in
$\quad \alpha(P_1 \cup \mathcal{R}[\![C_2]\!]\{\langle \mathrm{at}[\![C_2]\!],\, m\rangle \mid \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in P_1\})$ 〈def. $\mathcal{R}$〉
$=$ let $P_1 = \mathcal{R}[\![C_1]\!](\{\mathrm{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}))$ in
$\quad \alpha(P_1) \mathbin{\dot\cap} \alpha(\mathcal{R}[\![C_2]\!]\{\langle \mathrm{at}[\![C_2]\!],\, m\rangle \mid \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in P_1\})$ 〈(22), so that α is a complete join morphism and $\mathrm{at}[\![C_1 \,;\, C_2]\!] = \mathrm{at}[\![C_1]\!]$〉
$\dot\supseteq$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{P})$ in
$\quad \overline{P}_1 \mathbin{\dot\cap} \alpha(\mathcal{R}[\![C_2]\!]\{\langle \mathrm{at}[\![C_2]\!],\, m\rangle \mid \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in \gamma(\overline{P}_1)\})$ 〈by induction hypothesis (28) and (22), so that $\gamma \circ \alpha$ is extensive and α is monotone〉
$=$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{P})$ in $\overline{P}_1 \mathbin{\dot\cap} \alpha(\mathcal{R}[\![C_2]\!](\{\mathrm{at}[\![C_2]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}_1(\mathrm{after}[\![C_1]\!]))))$ 〈by def. (25) of γ〉
$\dot\supseteq$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{P})$ in $\overline{P}_1 \mathbin{\dot\cap} \overline{\mathcal{R}}[\![C_2]\!](\overline{P}_1(\mathrm{after}[\![C_1]\!]))$ 〈by induction hypothesis (28)〉

— For tests B, we have:

$\gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!]$
$= \mathcal{I}[\![\bigwedge \overline{P}]\!] \cap \mathcal{B}[\![B]\!]$ 〈by def. (19) of $\gamma_{\mathfrak{P}}$〉
$\subseteq \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid (\mathcal{I}[\![\bigwedge \overline{P}]\!] \cap \mathcal{B}[\![B]\!]) \subseteq \mathcal{B}[\![\mathfrak{p}]\!]\}$ 〈by def. upper bounds〉
$= \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid (\mathcal{I}[\![\bigwedge \overline{P}]\!] \cap \mathcal{I}[\![B]\!]) \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}$ 〈by def. $\mathcal{I}$〉
$= \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathcal{I}[\![\bigwedge \overline{P} \wedge B]\!] \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}$ 〈by def. $\mathcal{I}$〉
$= \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathcal{I}[\![(\bigwedge \overline{P} \wedge B) \Rightarrow \mathfrak{p}]\!] = \mathcal{M}\}$ 〈by def. $\mathcal{I}$〉
$\subseteq \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathrm{prover}[\![(\bigwedge \overline{P} \wedge B) \Rightarrow \mathfrak{p}]\!]\}$ 〈by prover soundness (13)〉
$\subseteq \gamma_{\mathfrak{P}}(\{\mathfrak{p} \mid \mathrm{prover}[\![(\bigwedge \overline{P} \wedge B) \Rightarrow \mathfrak{p}]\!]\})$ 〈def. (18) of $\gamma_{\mathfrak{P}}$〉

Therefore we define:

$$\overline{\mathcal{R}}[\![B]\!]\overline{P} \triangleq \{\mathfrak{p} \mid \mathrm{prover}[\![(\bigwedge P \wedge B) \Rightarrow \mathfrak{p}]\!]\}$$

such that:

$$\gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P}) \supseteq \gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!] \qquad (33)$$
$$\Leftrightarrow \overline{\mathcal{R}}[\![B]\!]\overline{P} \subseteq \alpha_{\mathfrak{P}}(\gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!])$$

— For the conditional $C = \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2$, we have

$\overline{\mathcal{R}}[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!]\overline{P}$

$=$ $\alpha(\mathcal{R}[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2]\!](\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})))$ ⟅def. $\overline{\mathcal{R}}$⟆

$=$ let $P_1 = \mathcal{R}[\![C_1]\!]\{\langle \text{at}[\![C_1]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!]\}$
and $P_2 = \mathcal{R}[\![C_2]\!]\{\langle \text{at}[\![C_2]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \cap \overline{\mathcal{B}}[\![B]\!]\}$
and $P_e = \{\langle \text{after}[\![C]\!],\ m\rangle \mid \langle \text{after}[\![C_1]\!],\ m\rangle \in P_1 \vee \langle \text{after}[\![C_2]\!],\ m\rangle \in P_2\}$ in
$\alpha((\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \cup P_1 \cup P_2 \cup P_e)$ ⟅def. $\mathcal{R}$⟆

$=$ let $P_1 = \mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!])$
and $P_2 = \mathcal{R}[\![C_2]\!](\{\text{at}[\![C_2]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P}) \cap \overline{\mathcal{B}}[\![B]\!])$
and $\overline{P}_e = \alpha(\{\langle \text{after}[\![C]\!],\ m\rangle \mid \langle \text{after}[\![C_1]\!],\ m\rangle \in P_1 \vee \langle \text{after}[\![C_2]\!],\ m\rangle \in P_2\})$ in
$\alpha(\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \alpha(P_1) \dot{\cap} \alpha(P_2) \dot{\cap} \overline{P}_e$ ⟅(22), so that α is a complete join morphism⟆

$\dot{\supseteq}$ let $P_1 = \mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P}))$
and $P_2 = \mathcal{R}[\![C_2]\!](\{\text{at}[\![C_2]\!]\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![\neg B]\!]\overline{P}))$
and $\overline{P}^t_e = \alpha(\{\langle \text{after}[\![C]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}_1(\text{after}[\![C_1]\!]))\})$
and $\overline{P}^f_e = \alpha(\{\langle \text{after}[\![C]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}_2(\text{after}[\![C_2]\!]))\})$ in
$\alpha(\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \alpha(P_1) \dot{\cap} \alpha(P_2) \dot{\cap} \overline{P}^t_e \dot{\cap} \overline{P}^f_e$ ⟅by $\overline{\mathcal{B}}[\![B]\!] = \mathcal{B}[\![\neg B]\!]$, (33), monotonicity of $\mathcal{R}[\![C]\!]$ and (22), so that α is a complete join morphism hence monotonic⟆

$\dot{\supseteq}$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P})$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!](\overline{\mathcal{R}}[\![\neg B]\!]\overline{P})$
and $\overline{P}^t_e = \dot{\alpha}_{\mathfrak{P}} \circ \alpha_{\downarrow}(\{\langle \text{after}[\![C]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}_1(\text{after}[\![C_1]\!]))\})$
and $\overline{P}^f_e = \dot{\alpha}_{\mathfrak{P}} \circ \alpha_{\downarrow}(\{\langle \text{after}[\![C]\!],\ m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}_2(\text{after}[\![C_2]\!]))\})$ in
$\alpha(\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \overline{P}_1 \dot{\cap} \overline{P}_2 \dot{\cap} \overline{P}^t_e \dot{\cap} \overline{P}^f_e$ ⟅by induction hypothesis (28), (22), and monotony of α and $\dot{\cap}$ and def. (23) of α⟆

$=$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P})$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!](\overline{\mathcal{R}}[\![\neg B]\!]\overline{P})$
and $\overline{P}^t_e = \lambda\ell\bullet\ \alpha_{\mathfrak{P}}(\{m \mid \ell = \text{after}[\![C]\!] \wedge m \in \gamma_{\mathfrak{P}}(\overline{P}_1(\text{after}[\![C_1]\!]))\})$
and $\overline{P}^f_e = \lambda\ell\bullet\ \alpha_{\mathfrak{P}}(\{m \mid \ell = \text{after}[\![C]\!] \wedge m \in \gamma_{\mathfrak{P}}(\overline{P}_2(\text{after}[\![C_2]\!]))\})$ in
$\alpha(\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \overline{P}_1 \dot{\cap} \overline{P}_2 \dot{\cap} \overline{P}^t_e \dot{\cap} \overline{P}^f_e$ ⟅by def. (20) of $\dot{\alpha}_{\mathfrak{P}}$ and (15) of $\alpha_{\downarrow}$⟆

$\dot{\supseteq}$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P})$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!](\overline{\mathcal{R}}[\![\neg B]\!]\overline{P})$
and $\overline{P}^t_e = \lambda\ell\bullet (\!|\ell = \text{after}[\![C]\!]\ ?\ \overline{P}_1(\text{after}[\![C_1]\!])\ :\ \mathfrak{P}|\!)$
and $\overline{P}^f_e = \lambda\ell\bullet (\!|\ell = \text{after}[\![C]\!]\ ?\ \overline{P}_2(\text{after}[\![C_2]\!])\ :\ \mathfrak{P}|\!)$ in
$\alpha(\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \dot{\cap} \overline{P}_1 \dot{\cap} \overline{P}_2 \dot{\cap} \overline{P}^t_e \dot{\cap} \overline{P}^f_e$ ⟅by def. (17) of $\alpha_{\mathfrak{P}}$, (16) so that $\alpha_{\mathfrak{P}} \circ \gamma_{\mathfrak{P}}$ is reductive⟆

$\dot\supseteq$ let $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P})$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!](\overline{\mathcal{R}}[\![\neg B]\!]\overline{P})$
and $\overline{P}e = \lambda\ell\bullet(\!|\ell = \text{after}[\![C]\!] \mathbin{?} \overline{P}_1(\text{after}[\![C_1]\!]) \cap \overline{P}_2(\text{after}[\![C_2]\!]) \mathbin{:} \mathfrak{P}|\!)$ in
$\lambda\ell\bullet(\!|\ell = \text{at}[\![C]\!] \mathbin{?} \overline{P} \mathbin{:} \mathfrak{P}|\!) \dot\cap \overline{P}_1 \dot\cap \overline{P}_2 \dot\cap \overline{P}_e$ ⟅by (30)⟆

— For the iteration $C =$ `while` B `do` C_1, given $\overline{P} \in \mathcal{L} \mapsto \wp(\mathfrak{P})$ and $X \subseteq \Sigma = \mathcal{L} \times \mathcal{M}$, we define:

$$F(X) \triangleq \mathcal{R}[\![C_1]\!](\{\langle \text{at}[\![C_1]\!],\, m\rangle \mid (\langle \text{at}[\![C]\!],\, m\rangle \in (\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \vee \langle \text{after}[\![C_1]\!],\, m\rangle \in X) \wedge m \in \mathcal{B}[\![B]\!]\})$$

We can now design the abstraction $\overline{F}$ of F such that $\alpha \circ F \mathrel{\dot\supseteq} \overline{F} \circ \alpha$:

$\alpha(F(X))$
$= \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \{m \mid (\langle \text{at}[\![C]\!],\, m\rangle \in (\{\text{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \vee \langle \text{after}[\![C_1]\!],\, m\rangle \in X) \wedge m \in \mathcal{B}[\![B]\!]\}))$ ⟅by def. F⟆
$\dot\supseteq \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \{m \mid (m \in \gamma_{\mathfrak{P}}(\overline{P}) \vee \langle \text{after}[\![C_1]\!],\, m\rangle \in \gamma \circ \alpha(X)) \wedge m \in \mathcal{B}[\![B]\!]\}))$ ⟅by (22) so that $\gamma \circ \alpha$ is extensive and monotony of $\mathcal{R}[\![C_1]\!]$ and α⟆
$\dot\supseteq \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}} \circ \alpha_{\mathfrak{P}}(\{m \mid (m \in \gamma_{\mathfrak{P}}(\overline{P}) \vee \langle \text{after}[\![C_1]\!],\, m\rangle \in \gamma \circ \alpha(X)) \wedge m \in \mathcal{B}[\![B]\!]\}))$ ⟅by (16) so that $\gamma_{\mathfrak{P}} \circ \alpha_{\mathfrak{P}}$ is extensive, monotony of $\mathcal{R}[\![C_1]\!]$ and (22) so that α is monotonic⟆
$\dot\supseteq \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\alpha_{\mathfrak{P}}(\gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!]) \cap \alpha_{\mathfrak{P}}(\{m \mid \langle \text{after}[\![C_1]\!],\, m\rangle \in \gamma \circ \alpha(X) \wedge m \in \mathcal{B}[\![B]\!]\}))))$ ⟅by (16) so that $\alpha_{\mathfrak{P}}$ is a complete join morphism⟆
$\dot\supseteq \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\alpha_{\mathfrak{P}}(\gamma_{\mathfrak{P}}(\overline{P}) \cap \mathcal{B}[\![B]\!]) \cap \alpha_{\mathfrak{P}}(\gamma_{\mathfrak{P}}(\alpha(X)_{\text{after}[\![C_1]\!]}) \cap \mathcal{B}[\![B]\!]))))$ ⟅by def. (25) of γ⟆
$\dot\supseteq \alpha(\mathcal{R}[\![C_1]\!](\{\text{at}[\![C_1]\!]\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P} \cap \overline{\mathcal{R}}[\![B]\!](\alpha(X)_{\text{after}[\![C_1]\!]}))))$ ⟅by (33) and monotony ⟆
$\dot\supseteq \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P} \cap \overline{\mathcal{R}}[\![B]\!](\alpha(X)_{\text{after}[\![C_1]\!]}))$ ⟅by induction hypothesis (28)⟆

We have $\alpha(\emptyset) = \lambda\ell\bullet\mathfrak{P}$ so that it follows that:

$$\alpha(\text{lfp}^{\subseteq}\, \lambda X \bullet \mathcal{R}[\![C_1]\!](\{\langle \text{at}[\![C_1]\!],\, m\rangle \mid (\langle \text{at}[\![C]\!],\, m\rangle \in P \vee \langle \text{after}[\![C_1]\!],\, m\rangle \in X) \wedge m \in \mathcal{B}[\![B]\!]\})) \tag{34}$$
$$\dot\supseteq\ \text{lfp}^{\dot\supseteq}_{\lambda\ell\bullet\mathfrak{P}}\, \lambda X \bullet \overline{\mathcal{R}}[\![C_1]\!](\overline{\mathcal{R}}[\![B]\!]\overline{P} \cap \overline{\mathcal{R}}[\![B]\!](X_{\text{after}[\![C_1]\!]})) \triangleq \overline{P}_1 \tag{35}$$

We will need to evaluate evaluate:

$\alpha(\{\langle \ell_1,\, m\rangle \mid \langle \ell_2,\, m\rangle \in \gamma(\overline{P}) \wedge m \in \mathcal{B}[\![B]\!]\})$
$= \alpha(\{\langle \ell_1,\, m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}_{\ell_2}) \wedge m \in \mathcal{B}[\![B]\!]\})$ ⟅by def. (25) of γ⟆
$= \alpha(\{\ell_1\} \times (\gamma_{\mathfrak{P}}(\overline{P}_{\ell_2}) \cap \mathcal{B}[\![B]\!]))$ ⟅by def. $\cap$ and $\times$⟆

$\dot\supseteq$ $\alpha(\{\ell_1\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P}_{\ell_2}))$ ⟅by (33) and (22), so that α monotonic⟆

$=$ $\dot\alpha_{\mathfrak{P}} \circ \alpha_{\downarrow}(\{\ell_1\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P}_{\ell_2}))$ ⟅by def. (23) of α⟆

$=$ $\lambda\ell\bullet\, \alpha_{\mathfrak{P}}(\{m \mid \langle \ell,\, m\rangle \in \{\ell_1\} \times \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![B]\!]\overline{P}_{\ell_2})\}$ ⟅by def. (20) of $\dot\alpha_{\mathfrak{P}}$ and (15) of $\alpha_{\downarrow}$⟆

$\dot\supseteq$ $\lambda\ell\bullet(\!|\ell = \ell_1 \mathbin{?} \overline{\mathcal{R}}[\![B]\!]\overline{P}_{\ell_2} \mathbin{:} \mathfrak{P}|\!)$ ⟅by (16) so that $\alpha_{\mathfrak{P}} \circ \gamma_{\mathfrak{P}}$ is reductive⟆ (36)

and therefore for $C =$ `while` B `do` C_1:

$\alpha(\mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})))$

$=$ $\alpha((\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \cup P_1 \cup \{\langle \mathrm{after}[\![C]\!],\, m\rangle \mid (m \in \gamma_{\mathfrak{P}}(\overline{P}) \vee \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in P_1) \wedge m \in \overline{\mathcal{B}}[\![B]\!]\})$ ⟅by (7)⟆

$\dot\supseteq$ $\alpha(\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(\overline{P})) \mathbin{\dot\cap} \overline{P}_1 \mathbin{\dot\cap} \alpha(\{\langle \mathrm{after}[\![C]\!],\, m\rangle \mid m \in \gamma_{\mathfrak{P}}(\overline{P}) \wedge m \in \overline{\mathcal{B}}[\![B]\!]\}) \mathbin{\dot\cap} \alpha(\{\langle \mathrm{after}[\![C]\!],\, m\rangle \mid \langle \mathrm{after}[\![C_1]\!],\, m\rangle \in P_1 \wedge m \in \overline{\mathcal{B}}[\![B]\!]\})$ ⟅by (35) and (22), so that α is a complete join morphism⟆

$\dot\supseteq$ $\lambda\ell\bullet(\!|\ell = \mathrm{at}[\![C]\!] \mathbin{?} \overline{P} \mathbin{:} \mathfrak{P}|\!) \mathbin{\dot\cap} \overline{P}_1 \mathbin{\dot\cap} \lambda\ell\bullet(\!|\ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{\mathcal{R}}[\![\neg B]\!]\overline{P} \mathbin{:} \mathfrak{P}|\!) \mathbin{\dot\cap} \lambda\ell\bullet(\!|\ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{\mathcal{R}}[\![\neg B]\!]\overline{P}_{\mathrm{after}[\![C_1]\!]} \mathbin{:} \mathfrak{P}|\!)$ ⟅by (32), $\overline{\mathcal{B}}[\![B]\!] = \mathcal{B}[\![\neg B]\!]$ and (36)⟆

$=$ $\overline{P}_1 \mathbin{\dot\cap} \lambda\ell\bullet(\!|\ell = \mathrm{at}[\![C]\!] \mathbin{?} \overline{P} \parallel \ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{\mathcal{R}}[\![\neg B]\!]\overline{P} \cap \overline{\mathcal{R}}[\![\neg B]\!]\overline{P}_{\mathrm{after}[\![C_1]\!]} \mathbin{:} \mathfrak{P}|\!)$ ⟅by def $\dot\cap$ and conditional⟆

— In conclusion of these calculi, we have proved that:

$\overline{\mathcal{R}}[\![C]\!]\overline{P}\ell = (\!|\ell \notin \mathrm{lab}[\![C]\!] \mathbin{?} \mathfrak{P} \mathbin{:} \mathrm{match}\ C,\ \ell\ \mathrm{with}$

| $\_, \mathrm{at}[\![C]\!] \ \to\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\}$ (38)

| `skip`$, \mathrm{after}[\![C]\!] \ \to\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![\bigwedge \overline{P} \Rightarrow \mathfrak{p}]\!]\}$

| $\mathtt{X} := E, \mathrm{after}[\![C]\!] \ \to\ \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![(\exists \mathtt{X}' : \bigwedge \overline{P}[\mathtt{X}/\mathtt{X}'] \wedge \mathtt{X} \in E[\mathtt{X}/\mathtt{X}'])) \Rightarrow \mathfrak{p}]\!]\}$

| $C_1\,;C_2, \_ \ \to\ \mathrm{let}\ \overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!]\overline{P}$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!]\overline{P}_1(\mathrm{after}[\![C_1]\!])$ in
$\overline{P}_1(\ell) \cap \overline{P}_2(\ell)$

| `if` B `then` C_1 `else` $C_2, \_ \ \to$ (39)
let $\overline{P}_t = \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![(\bigwedge \overline{P} \wedge B) \Rightarrow \mathfrak{p}]\!]\}$ and $\overline{P}_1 = \overline{\mathcal{R}}[\![C_1]\!]\overline{P}_t$
and $\overline{P}_f = \{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![(\bigwedge \overline{P} \wedge \neg B) \Rightarrow \mathfrak{p}]\!]\}$ and $\overline{P}_2 = \overline{\mathcal{R}}[\![C_2]\!]\overline{P}_f$
in $\overline{P}_1(\ell) \cap \overline{P}_2(\ell) \cap (\!|\ell = \mathrm{after}[\![C]\!] \mathbin{?} \overline{P}_1(\mathrm{after}[\![C_1]\!]) \cap \overline{P}_2(\mathrm{after}[\![C_2]\!]) \mathbin{:} \mathfrak{P}|\!)$

| `while` B `do` $C_1, \_ \ \to$ (40)

$$\begin{array}{l}\text{let } \overline{P}_1 = \mathrm{gfp}^{\dot{\subseteq}}_{\lambda\ell\cdot\mathfrak{P}}\ \lambda X\cdot\lambda\ell'\cdot(\!|\ell' = \mathrm{at}[\![C_1]\!]\ ?\ \{\mathfrak{p}\in\mathfrak{P}\mid \mathrm{prover}[\![((\bigwedge\overline{P}\vee\\ \qquad\qquad X(\mathrm{after}(C_1)))\wedge B)\Rightarrow\mathfrak{p}]\!]\}\ :\ \mathfrak{P}|\!)\,\dot{\cap}\,\overline{\mathcal{R}}[\![C_1]\!]X(\mathrm{at}[\![C_1]\!])\ell'\\ \text{in } \overline{P}_1(\ell)\cap(\!|\ell = \mathrm{after}[\![C]\!]?\ \{\mathfrak{p}\in\mathfrak{P}\mid \mathrm{prover}[\![((\bigwedge\overline{P}\vee\\ \qquad\qquad \overline{P}_1(\mathrm{after}[\![C_1]\!])\wedge\neg B)\Rightarrow\mathfrak{p}]\!]\}\ :\ \mathfrak{P}|\!)\end{array}\tag{41}$$

$$|\!]\tag{42}$$

Observe that if $\langle\wp(\mathfrak{P}),\ \supseteq\rangle$ is Notherian then this abstract semantics can be computed by induction on the program structure with a finite iteration for fixpoints. If $\langle\wp(\mathfrak{P}),\ \supseteq\rangle$ is finite then the equivalent abstraction (27) would provide an isomorphic boolean encoding allowing for the reuse of boolean fixpoint computation algorithms as found in model-checkers, but this is not mandatory since the iteration is trivial and the induction over the program structure would be lost (or would have to be rediscovered by the fixpoint engine).

3.2.8 Reduced Set of Ground Abstract Predicates

Observe that because of the normalization $\{\mathfrak{p}\in\mathfrak{P}\mid \mathrm{prover}[\![\bigwedge\overline{P}\Rightarrow\mathfrak{p}]\!]\}$ of $\overline{P}\in\wp(\mathfrak{P})$, the abstract domain $\langle\wp(\mathfrak{P},\ \supseteq\rangle$ can be reduced [20, Prop. 10] to

$$\langle\{\{\mathfrak{p}\in\mathfrak{P}\mid \mathrm{prover}[\![\bigwedge\overline{P}\Rightarrow\mathfrak{p}]\!]\}\mid\overline{P}\in\wp(\mathfrak{P}\},\ \supseteq\rangle\ .\tag{43}$$

The abstract domain $\langle\wp(\mathfrak{P}),\supseteq\rangle$ may not be Notherian while the above reduced abstract domain (43) is Notherian. In this case, the iterative computation of the abstract fixpoint for loops does terminate in the reduced abstract domain (43). Otherwise the iterative fixpoint computation may not terminate whence may require to be over approximated by a widening [16].

Example 16. In Kildall's constant propagation [40] for a command C, the set of abstract predicates is $\mathfrak{P} = \{\mathtt{X} = v\mid \mathtt{X}\in\mathrm{var}[\![C]\!]\wedge v\in\mathcal{D}\}$. Its reduction (43) yields the abstract predicates $\emptyset$ ("I don't know"), $\mathfrak{P}$ ("false") and the $\{\mathtt{X}_1 = v_1,\ldots,\mathtt{X}_n = v_n\}$ (where $i\neq j$ implies $\mathtt{X}_i\neq\mathtt{X}_j$). This reduced set of abstract predicates is still infinite but Notherian. It is isomorphic to the smash product of Kildall's lattice $\langle\{\bot,\top\}\cup\{i\mid i\in\mathbb{Z}\},\sqsubseteq\rangle$ shown in Fig. 18 for each variable $\mathtt{X}\in\mathrm{var}[\![C]\!]$ of command C. The fixpoint (42) can be computed iteratively (as in [40] which corresponds to a particular chaotic iteration strategy [17, 12]). □

3.2.9 Local Ground Abstract Predicates

Instead of choosing the set $\mathfrak{P}$ of abstract predicates globally, it can be chosen locally, by choosing a particular set of abstract predicates $\mathfrak{P}_\ell$ attached to each

commend label $\ell \in \mathrm{lab}[\![C]\!]$. Then terms of the form $\{\mathfrak{p} \in \mathfrak{P} \mid \mathrm{prover}[\![P \Rightarrow \mathfrak{p}]\!]$ attached to program point ℓ in the definition of the abstract predicate transformer (38)–(42) are to be simply replaced by $\{\mathfrak{p} \in \mathfrak{P}_\ell \mid \mathrm{prover}[\![P \Rightarrow \mathfrak{p}]\!]$.

3.2.10 Safety Verification of Commands $C \in \mathbb{C}$

The verification that a command satisfies a safety specification $S \in \mathrm{lab}[\![C]\!] \mapsto \wp(\mathfrak{P})$ consists in checking for each point $\ell \in \mathrm{lab}[\![C]\!]$ that:

$$\mathrm{prover}[\![(\bigwedge \overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))\ell) \Rightarrow \bigwedge S(\ell)]\!] \ .$$

This is sound since:

$$\begin{array}{ll}
& \forall \ell \in \mathrm{lab}[\![C]\!] : \mathrm{prover}[\![(\bigwedge \overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))\ell) \Rightarrow \bigwedge S(\ell)]\!] \\
\Rightarrow & \forall \ell \in \mathrm{lab}[\![C]\!] : \mathcal{I}[\![(\bigwedge \overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))\ell) \Rightarrow \bigwedge S(\ell)]\!] = \mathcal{M} & \text{⟅by soundness (13) of the prover⟆} \\
\Rightarrow & \forall \ell \in \mathrm{lab}[\![C]\!] : \mathcal{I}[\![(\bigwedge \overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))\ell)]\!] \subseteq \mathcal{I}[\![\bigwedge S(\ell)]\!] & \text{⟅by def. of } \mathcal{I}\text{⟆} \\
\Rightarrow & \forall \ell \in \mathrm{lab}[\![C]\!] : \gamma_{\mathfrak{P}}(\overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))\ell) \subseteq \gamma_{\mathfrak{P}}(S(\ell)) & \text{⟅by (21)⟆} \\
\Rightarrow & \dot{\gamma}_{\mathfrak{P}}(\overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))) \mathbin{\dot{\subseteq}} \dot{\gamma}_{\mathfrak{P}}(S) & \text{⟅by (19)⟆} \\
\Rightarrow & \gamma_{\downarrow} \circ \dot{\gamma}_{\mathfrak{P}}(\overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))) \mathbin{\dot{\subseteq}} \gamma_{\downarrow} \circ \dot{\gamma}_{\mathfrak{P}}(S) & \text{⟅by (14) so that } \gamma_{\downarrow} \text{ is monotone⟆} \\
\Rightarrow & \gamma(\overline{\mathcal{R}}[\![C]\!](S(\mathrm{at}[\![C]\!]))) \mathbin{\dot{\subseteq}} \gamma(S) & \text{⟅by def. (24) of } \gamma\text{⟆} \\
\Rightarrow & \mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(S(\mathrm{at}[\![C]\!]))) \subseteq \gamma(S) & \text{⟅by (29)⟆} \\
\Rightarrow & \forall \ell \in \mathrm{lab}[\![C]\!] : \mathcal{R}[\![C]\!](\{\mathrm{at}[\![C]\!]\} \times \gamma_{\mathfrak{P}}(S(\mathrm{at}[\![C]\!])))\ell \subseteq \gamma_{\mathfrak{P}}(S(\ell)) & \text{⟅def. } \gamma\text{⟆}
\end{array}$$

so that, informally, $S(\ell)$ holds whenever program point ℓ is reached during any execution of command C starting at point $\mathrm{at}[\![C]\!]$ with an initial memory state satisfying $S(\mathrm{at}[\![C]\!])$.

4 Application to Predicate Abstraction Completion

In this section we show that the refinement process which is used in predicate abstraction [5] is an instance of the fixpoint completion of Sec. **??** and simply consists in computing the concrete reachability semantics of Sec. 3.2, whence the need for a widening and the remark that a boolean predicate abstraction analysis at each iteration step is uselessly laborious.

5 Application to Generic Predicate Abstraction

The inconvenience of ground predicate abstractions is that the ground predicates directly refer to the program states and control by explicitly naming program constants, variables and may be control points. Consequently, the abstract domain, being program specific, has to be redesigned for each new or modified program. This design can be partially automatize by the refinement techniques of Sec. 4, including convergence acceleration by widening, but this alternation of analyzes and refinements would be costly for precise analysis of large programs. An alternative is to provide program independent predicates by designing *generic abstract domains*.

This section introduces more presumptive ideas and is presented informally through a sorting example.

5.1 Generic Abstract Domains

A generic abstract domains is parameterized so that a particular abstract domain instantiation for a given program is obtained by binding the parameters to the constants, variables, control points, etc. of this specific program.

Example 17. For a simple example, Kildall's generic abstract domain for constant propagation $D(C, V)$ is:

$$D(C, V) = \prod_{\ell \in C} \prod_{\mathtt{X} \in V(\ell)} L \ .$$

where L is Kildall's complete lattice of Ex. 8. Given a command C, it is instantiated to $D(\mathrm{lab}[\![C]\!], \mathrm{var}[\![C]\!])$ where $\mathrm{lab}[\![C]\!]$ is the set of labels of command C and $\mathrm{var}[\![C]\!](\ell)$ is the set of program variables $\mathtt{X}$ which are visible at this program point ℓ of command C. □

5.2 Generic Comparison Abstract Domain

We let $\mathcal{D}_r(X)$ be a generic relational integer abstract domain parameterized by a set X of program and auxiliary variables (such as octagons [44, 45] or polyhedra [29]). This abstract domain is assumed to have abstract operations on $r, r_1, r_2 \in \mathcal{D}_r(X)$ such as the projection or variable elimination $\exists x \in X : r$, disjunction $r_1 \vee r_2$, conjunction $r_1 \wedge r_2$, abstract predicate transformers for assignments and tests, etc.

Then we define the generic comparison abstract domain:

$$\mathcal{D}_{\mathrm{lt}}(X) = \{\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle \mid \mathtt{t} \in X \wedge a, b, c, d \notin X \wedge r \in \mathcal{D}_r(X \cup \{a, b, c, d\})\}\ .$$

The meaning $\gamma(\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle)$ of an abstract predicate $\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle$ is informally that all elements of $\mathtt{t}$ between indices a and b are less than any element of $\mathtt{t}$ between indices c and d and moreover r holds:

$$\gamma(\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle) = \exists a, b, c, d : \mathtt{t}.\ell \le a \le b \le \mathtt{t}.h \wedge \mathtt{t}.\ell \le c \le d \le \mathtt{t}.h \wedge \forall i \in [a, b] : \forall j \in [c, d] : \mathtt{t}[i] \le \mathtt{t}[j] \wedge r$$

where $\mathtt{t}.\ell$ is the lower bound and $\mathtt{t}.h$ is the upper bound of the indices i of the array $\mathtt{t}$ with elements $\mathtt{t}[i]$.

More formally, there should be a declaration $\mathtt{t}$: $\mathtt{array}[\ell, h]$ $\mathtt{of\ int}$ so that $\gamma(\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle)$ defines a set of environments ρ mapping program and auxiliary variables $\mathtt{X}$ to their value $\rho(\mathtt{X})$ for which the above concrete predicate holds:

$$\begin{aligned}\gamma(\langle \mathrm{lt}(t, a, b, c, d),\ r\rangle) = \{\rho \mid\ & \exists a, b, c, d : \rho(\mathtt{t}).\ell \le a \le b \le \rho(\mathtt{t}).h \\ & \wedge\ \rho(\mathtt{t}).\ell \le c \le d \le \rho(\mathtt{t}).h \\ & \wedge\ \forall i \in [a, b] : \forall j \in [c, d] : \rho(\mathtt{t})[i] \le \rho(\mathtt{t})[j] \\ & \wedge\ \rho \in \gamma(r)\}\end{aligned}$$

where the domain of the ρ is $X \cup \{a, b, c, d\}$ and $\gamma(r)$ is the concretization of the abstract predicate $r \in \mathcal{D}_r(X \cup \{a, b, c, d\})$ specifying the possible values of the variables in X and the auxiliary variables a, b, c, d.

5.3 Abstract Logical Operations of the Generic Comparison Abstract Domain

Then the abstract domain must be equipped with abstract operations such as the implication $\Rightarrow$, conjunction $\wedge$, disjunction $\vee$, etc. We simply provided a few examples.

5.3.1 Abstract Implication

We have $\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle \Rightarrow r$. If $r \Rightarrow r'$ and $a \le b \le c \le d$ and $e \le f \le g \le h$ then;

$$\langle \mathrm{lt}(\mathtt{t}, a, d, e, h),\ r\rangle \Rightarrow \langle \mathrm{lt}(\mathtt{t}, b, c, f, g),\ r'\rangle \tag{44}$$

as shown in Fig. 19.

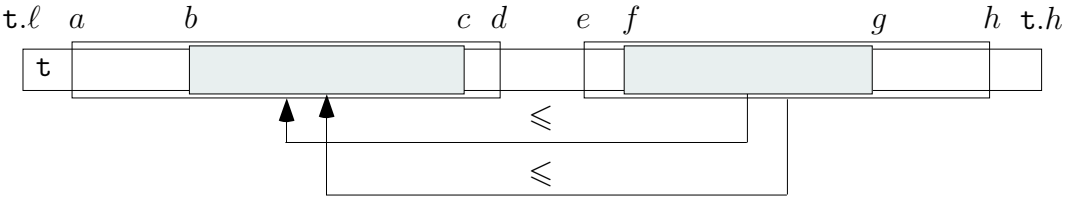


Fig. 19. Implication

5.3.2 Abstract Conjunction

If $\mathtt{t}, i, j, k, \ell \notin \mathrm{var}[\![r]\!]$, then:

$$r \wedge \langle \mathrm{lt}(\mathtt{t}, a, c, f, h),\ r'\rangle = \langle \mathrm{lt}(\mathtt{t}, a, c, f, h),\ r \wedge r'\rangle \tag{45}$$

If $a \leq b \leq c \leq d$ and $e \leq f \leq g \leq h$ then we have:

$$\langle \mathrm{lt}(\mathtt{t}, a, c, f, h),\ r\rangle \wedge \langle \mathrm{lt}(\mathtt{t}, b, d, e, g),\ r'\rangle, = \langle \mathrm{lt}(\mathtt{t}, b, c, f, g),\ \exists a, d, e, h : r \wedge r'\rangle$$

as shown in Fig. 20. The same way, as shown in Fig. 21, we have:

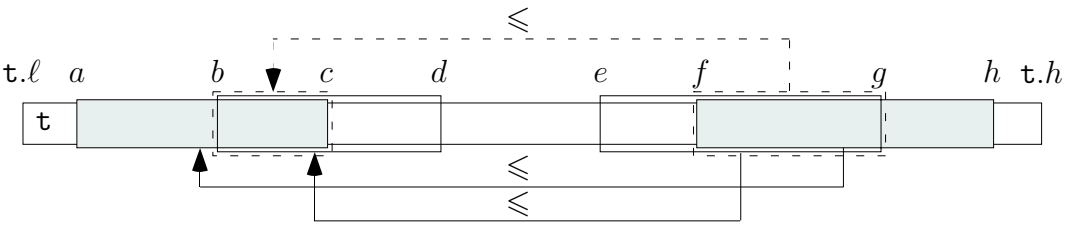


Fig. 20. Conjunction

$$\langle \mathrm{lt}(\mathtt{t}, a, b, c, e),\ r\rangle \wedge \langle \mathrm{lt}(\mathtt{t}, d, f, g, h),\ r'\rangle = \langle \mathrm{lt}(\mathtt{t}, a, b, g, h),\ \exists c, e, d, f : r \wedge r'\rangle \tag{46}$$

when $(r \wedge r') \Rightarrow (c \leq d \leq e \leq f)$.

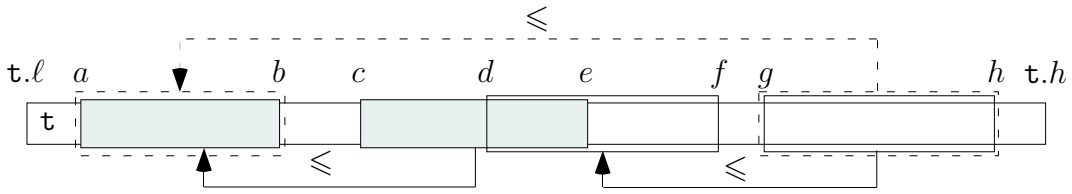


Fig. 21. Conjunction

5.3.3 Abstract Disjunction

We have:

$$\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle \vee \langle \mathrm{lt}(\mathtt{t}, e, f, g, h),\ r'\rangle = \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (\exists a, b, c, d : i = a \quad (47)$$
$$\wedge\, j = b \wedge k = c \wedge \ell = d \wedge r) \vee (\exists e, f, g, h : i = e \wedge j = f \wedge k = g \wedge \ell = h \wedge r')\rangle$$

In case one of the terms does not refer to the array ($\mathtt{t} \notin \mathrm{var}[\![r]\!]$), a criterion must be used to force the introduction of an identically true array term $\mathrm{lt}(\mathtt{t}, i, i, i, i)$. For example if the auxiliary variables d, f, g, h in r' depend upon one selectively chosen variable $\mathtt{I}$, then we have:

$$r \vee \langle \mathrm{lt}(\mathtt{t}, d, f, g, h),\ r'\rangle = \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} \wedge r) \vee \quad (48)$$
$$(\exists d, f, g, h : i = d \wedge j = f \wedge k = g \wedge \ell = h \wedge r')\rangle$$

This case appears typically in loops, which can also be handled by unrolling, see Sec. 5.5.

5.4 Abstract Predicate Transformers for the Generic Comparison Abstract Domain

Then the abstract domain must be equipped with abstract predicate transformers for tests, assignments, etc. We consider forward strongest postconditions (although weakest preconditions, which avoid an existential quantifier in assignments, may sometimes be simpler [47]).

We depart from traditional predicate abstraction which uses a simplifier (or a theorem prover) to formally evaluate the abstract predicate transformer $\alpha \circ F \circ \gamma$ approximating the concrete predicate transformer F. The alternative proposed below is traditional in static program analysis and directly provides an over-approximation of the best abstract predicate transformer $\alpha \circ F \circ \gamma$ in the form of an algorithm (which correctness must be established formally). The

simplifier/prover/pattern-matcher is used only to reduce the post-condition in the normal form (44) which is required for the abstract predicates.

5.4.1 Abstract Strongest Postconditions for Tests

$$
\begin{array}{l}
\{\ P_1\ \} \\
\textbf{if}\ (\mathtt{t}[\mathtt{I}]\ >\ \mathtt{t}[\mathtt{I}+1])\ \textbf{then} \\
\quad \{\ P_1 \wedge \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{I} \wedge j = \mathtt{I}+1 \wedge k = \mathtt{I} \wedge \ell = \mathtt{I} \rangle\ \} \qquad (49) \\
\quad \ldots \\
\quad \{\ P_2\ \} \\
\textbf{else} \\
\quad \{\ P_1 \wedge \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{I} \wedge j = k = \ell = \mathtt{I}+1 \rangle\ \} \qquad (50) \\
\quad \ldots \\
\quad \{\ P_3\ \} \\
\textbf{fi} \\
\{\ P_2 \vee P_3\ \} \qquad (51)
\end{array}
$$

5.4.2 Abstract Strongest Postconditions for Assignments

For assignment, assuming $\mathtt{t} \notin \mathrm{var}[\![r]\!]$ and $r \Rightarrow (i = \mathtt{I} \wedge j = \mathtt{I}+1 \wedge k = \mathtt{I} \wedge \ell = \mathtt{I})$, we have:

$$
\begin{array}{l}
\{\ \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ r \rangle\} \\
\mathtt{t}[\mathtt{I}]\ :=:\ \mathtt{t}[\mathtt{I}+1] \qquad (52) \\
\{\ \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ \exists i, j, k, \ell : r \wedge m = \mathtt{I} \wedge n = p = q = \mathtt{I}+1 \rangle\ \}\ .
\end{array}
$$

The same way if $\mathtt{t} \notin \mathrm{var}[\![r]\!]$ and $r \Rightarrow (\mathtt{I} \in [i, j] \wedge \mathtt{J} \in [i, j]) \vee (\mathtt{J} \in [k, \ell] \wedge \mathtt{I} \in [k, \ell])$ then:

$$
\begin{array}{l}
\{\ \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ r \rangle\} \\
\mathtt{t}[\mathtt{I}]\ :=:\ \mathtt{t}[\mathtt{J}] \qquad (53) \\
\{\ \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ r \rangle\}
\end{array}
$$

since the swap of the array elements does not interfere with the assertions.

Finally the abstract domain must be equipped with a widening (and optionally a narrowing to improve precision) to speed up the convergence of iterative fixpoint computations [16]. We choose to define the widening $\triangledown$ as:

$$\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ r\rangle \triangledown \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ r'\rangle \tag{54}$$

$$= \text{let } \langle \mathrm{lt}(\mathtt{t}, r, s, t, u),\ r''\rangle = \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ r\rangle \vee \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ r'\rangle \text{ in}$$

$$\langle \mathrm{lt}(\mathtt{t}, r, s, t, u),\ r \triangledown r''\rangle\ . \tag{55}$$

Typically, when handling loops, one encounters widenings of the form $r \triangledown \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ r'\rangle$ where r corresponds to the loop entry condition while the term $\mathrm{lt}(\mathtt{t}, m, n, p, q)$ appears during the analysis of the loop body. There are several ways to handle this situation:

(1) Incorporate the term $\mathrm{lt}(\mathtt{t}, i, j, k, \ell)$ in the form of a tautology, as already described in (48) for the abstract disjunction;
(2) Use disjunctive completion (see Sec. **??**) to preserve the disjunction within the loop (which may ultimately lead to infinite disjunctions) or better allow only abstract predicates of the more restricted form $r \vee \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ r'\rangle$ (which definitively avoids the previous potential explosion);
(3) Use *semantically loop unrolling* (as in [7, Sec. 6.5]) so that the loop:

$$\text{while } B \text{ do } C \text{ od}$$

is handled in the abstract semantics as if written in the form:

$$\text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ od fi}$$

which is equivalent in the concrete semantics. More generally, if several abstract terms of different kinds are considered (like $\mathrm{lt}(\mathtt{t}, i, j, k, \ell)$ and $\mathrm{s}(\mathtt{t}, m, n)$ in the forthcoming Sec. 5.10), a further semantic unrolling can be performed each time a term of a new kind does appear, while all terms of the same king are merged by the widening.

5.6 Refined Generic Comparison Abstract Domains

The generic comparison abstract domain $\mathcal{D}_{\mathrm{lt}}(X)$ of Sec. 5.2 may be imprecise since it allows only for one term $\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ r\rangle$. First we could consider several arrays, with one such term per array. Second, we could consider the conjunction of such terms for a given array, which is more precise but may potentially lead to infinite conjunctions within loops (e.g. for which termination

cannot be established). So we will consider this alternative within tests only, then applying the above abstract domain operators term by term [12] .

The same way we could the disjunctive completion (see Sec. **??**) of this domain, that is terms of the form $\bigvee_i \bigwedge_j \langle \mathrm{lt}(\mathtt{t}, a_{ij}, b_{ij}, c_{ij}, d_{ij}),\ r_{ij} \rangle$. This would introduce an exponential complexity factor, which we prefer to avoid. If necessary, we will use *local trace partitioning* [7, Sec. 6.6] instead.

5.7 Generic Comparison Static Program Analysis

Let us consider the following program (where $a \leq b$) which is similar to the inner loop of bubble sort [41]:

```
1 : var t : array [a, b] of int;
2 : I := a;
3 : while (I < b) do
4 :     if (t[I] > t[I + 1]) then
5 :         t[I] :=: t[I + 1]
6 :     fi;
7 :     I := I + 1
8 : od
```

We let P_p^i be the value of the local predicate attached to the program point $p = 1, ..., 8$ at the i^{th} iteration. Initially, $P_1^0 = (\mathtt{a} \leq \mathtt{b})$ while $P_p^0 = \text{false}$ for $p = 2, ..., 8$. We choose the octagonal abstract domain [44, 45] as the generic relational integer abstract domain $\mathcal{D}_r(X)$ parameterized by the set X of program variables `I`, `J`,... and auxiliary variables i, j, etc. The fixpoint iterates are as follows:

$$
\begin{aligned}
P_1^1 &= (\mathtt{a} \leq \mathtt{b}) && \text{⟅initialization to } P_1^0 \text{⟆} \\
P_2^1 &= (\mathtt{I} = \mathtt{a} \leq \mathtt{b}) && \text{⟅assignment } (\mathtt{I} := \mathtt{a}) \text{⟆} \\
P_3^1 &= (\mathtt{I} = \mathtt{a} < \mathtt{b}) && \text{⟅loop condition } \mathtt{I} < \mathtt{b} \text{⟆}
\end{aligned}
$$

[12] For short we avoid to resort to semantical loop unrolling which is better adapted to automatization but would yield to lengthy handmade calculations in this section. This technique will be illustrated anyway in the forthcoming Sec. 5.10.

$P_4^1 = \langle \mathrm{lt}(\mathtt{t}, i, j, k, l),\ i = k = \ell = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge j = \mathtt{I} + 1 \rangle$ ⟅by (49) for test condition $(\mathtt{t}[\mathtt{I}] > \mathtt{t}[\mathtt{I}+1])$⟆

$P_5^1 = \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ \exists i, j, k, \ell : i = k = \ell = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge j = \mathtt{I} + 1 \wedge m = \mathtt{I} \wedge n = \mathtt{I} + 1 \wedge p = \mathtt{I} + 1 \wedge q = \mathtt{I} + 1 \rangle$
⟅by assignment (52) which, by octagonal projection, simplifies into:⟆

$= \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ m = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge n = p = q = \mathtt{I} + 1 \rangle$

$P_6^1 = (P_3^1 \wedge \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge j = k = \ell = \mathtt{I} + 1 \rangle) \vee P_5^1$
⟅by (50) for test condition $(\mathtt{t}[\mathtt{I}] > \mathtt{t}[\mathtt{I}+1])$ and join (51)⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge j = k = \ell = \mathtt{I} + 1 \rangle) \vee (\langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ m = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge n = p = q = \mathtt{I} + 1 \rangle)$
⟅by def. P_3^1 and (45) as well as by def. of P_5^1⟆

$= \langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ (\exists i, j, k, \ell : a = i \wedge b = j \wedge c = k \wedge d = \ell \wedge i = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge j = k = \ell = \mathtt{I} + 1) \vee (\exists m, n, p, q : a = m \wedge b = n \wedge c = p \wedge d = q \wedge m = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge n = p = q = \mathtt{I} + 1) \rangle$ ⟅by def. (47) of the abstract union $\vee$⟆

$= \langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ (a = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge b = c = d = \mathtt{I} + 1) \vee (a = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge b = c = d = \mathtt{I} + 1) \rangle$ ⟅by octagonal projection⟆

$= \langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ a = \mathtt{I} = \mathtt{a} < \mathtt{b} \wedge b = c = d = \mathtt{I} + 1 \rangle$ ⟅by octagonal disjunction⟆

$P_7^1 = \langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ a = \mathtt{I} - 1 = \mathtt{a} < \mathtt{b} \wedge b = c = d = \mathtt{I} \rangle$ ⟅by invertible assignment $\mathtt{I} := \mathtt{I} + 1$⟆

$= \langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ \mathtt{I} = a + 1 = \mathtt{a} + 1 \leq \mathtt{b} \wedge b = c = d = \mathtt{I} \rangle$ ⟅octagonal simplification⟆

$P_3^2 = (P_2^1 \vee P_7^1) \wedge (\mathtt{I} < \mathtt{b})$ ⟅loop condition $\mathtt{I} < \mathtt{b}$ and absence of widening on first iterate⟆

$= ((\mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee (\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ \mathtt{I} = a + 1 = \mathtt{a} + 1 \leq \mathtt{b} \wedge b = c = d = \mathtt{I} \rangle)) \wedge (\mathtt{I} < \mathtt{b})$ ⟅def. P_2^1 and P_7^1⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee (\exists a, b, c, d : i = a \wedge b = j \wedge c = k \wedge d = \ell \wedge \mathtt{I} = a + 1 = \mathtt{a} + 1 \leq \mathtt{b} \wedge b = c = d = \mathtt{I}) \rangle) \wedge (\mathtt{I} < \mathtt{b})$
⟅def. (48) of abstract disjunction, the octagonal predicate (56) depending only on $\mathtt{I}, \mathtt{a}$ and $\mathtt{b}$ which leads to the selection of $\mathtt{I}$, the only of these variables which is modified within the loop body⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee (\mathtt{I} = i + 1 = \mathtt{a} + 1 \leq \mathtt{b} \wedge j = k = \ell = \mathtt{I}) \rangle) \wedge (\mathtt{I} < \mathtt{b})$ ⟅by octagonal projection⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} = \mathtt{a} < \mathtt{b}) \vee (\mathtt{I} = i + 1 = \mathtt{a} + 1 < \mathtt{b} \wedge j = k = \ell = \mathtt{I}) \rangle)$ ⟅by octagonal conjunction⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} \leq \mathtt{a} + 1 \leq \mathtt{b} \rangle$ ⟅by octagonal disjunction⟆

$P_3^3 = P_3^2 \triangledown \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} \leq \mathtt{a} + 2 \leq \mathtt{b} \rangle$ ⟅in absence of stabilization of the iterates, by a similar computation at the next iteration⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle$ ⟅by def. (55) of the widening $\triangledown$⟆

$P_4^3 = P_3^3 \wedge \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ m = p = q = \mathtt{I} \wedge n = \mathtt{I} + 1 \rangle$ ⟅by (49) for test condition $(\mathtt{t}[\mathtt{I}] > \mathtt{t}[\mathtt{I} + 1])$⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle \wedge \langle \mathrm{lt}(\mathtt{t}, m, n, p, q),\ m = p = q = \mathtt{I} \wedge n = \mathtt{I} + 1 \rangle$ ⟅by def. P_4^3, the conjunction being left symbolic since it cannot be simplified, see Sec. 5.6⟆

$P_5^3 = \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell), i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle \wedge \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell), \exists m, n, p, q : m = p = q = \mathtt{I} \wedge n = \mathtt{I} + 1 \wedge i = \mathtt{I} \wedge j = k = \ell = \mathtt{I} + 1 \rangle$ ⟅by (53) and (52) where $\mathtt{t} \notin \mathrm{var}[\![d]\!]$ and $d \Rightarrow m = p = q = \mathtt{I} \wedge n = \mathtt{I} + 1$⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle \wedge \langle \mathrm{lt}(\mathtt{t}, i', j', k', \ell'),\ i' = \mathtt{I} \wedge j' = k' = \ell' = \mathtt{I} + 1 \rangle$ ⟅by octagonal projection⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} + 1 \leq \mathtt{b} \rangle$ ⟅by def. (46), of conjunction and octagonal projection⟆

$P_6^3 = (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle \wedge \langle \mathrm{lt}(\mathtt{t}, i', j', k', \ell'),\ i' = \mathtt{I} \wedge j' = k' = \ell' = \mathtt{I} + 1 \rangle) \vee \langle \mathrm{lt}(\mathtt{t}, i'', j'', k'', \ell''),\ i'' = \mathtt{a} \leq j'' = k'' = \ell'' = \mathtt{I} + 1 \leq \mathtt{b} \rangle$ ⟅by $P_6^3 = (P_3^3 \wedge (\mathtt{t}[\mathtt{I}] \leq \mathtt{t}[\mathtt{I} + 1])) \vee P_5^3$ and (50)⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} + 1 \leq \mathtt{b} \rangle \vee \langle \mathrm{lt}(\mathtt{t}, i'', j'', k'', \ell''),\ i'' = \mathtt{a} \leq j'' = k'' = \ell'' = \mathtt{I} + 1 \leq \mathtt{b} \rangle$ ⟅by def. (46), of conjunction and octagonal projection⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} + 1 \leq \mathtt{b} \rangle$ ⟅by $P \vee P = P$⟆

$P_7^3 = \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} \leq \mathtt{b} \rangle$ ⟅by assignment $\mathtt{I} := \mathtt{I} + 1$⟆

Now the iterates have stabilized since:

$(P_2^3 \vee P_7^3) \wedge (\mathtt{I} < \mathtt{b})$

$= (P_2^1 \vee P_7^3) \wedge (\mathtt{I} < \mathtt{b})$ ⟅since $P_2^3 = P_2^1$ is stable⟆

$= ((\mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} \leq \mathtt{b} \rangle) \wedge (\mathtt{I} < \mathtt{b})$ ⟅def. P_2^1 and P_7^3⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee (\exists a, b, c, d : i = a \wedge b = j \wedge c = k \wedge d = \ell \wedge \mathtt{I} = a + 1 = \mathtt{a} + 1 \leq \mathtt{b} \wedge b = c = d = \mathtt{I}) \rangle) \wedge (\mathtt{I} < \mathtt{b})$ ⟅def. (48) of abstract disjunction with selection of $\mathtt{I}$ as in (56)⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ (i = j = k = \ell = \mathtt{I} = \mathtt{a} \leq \mathtt{b}) \vee (j = k = \ell = \mathtt{I} = i + 1 = \mathtt{a} + 1 \leq \mathtt{b}) \rangle) \wedge (\mathtt{I} < \mathtt{b})$ ⟅by octagonal projection⟆

$= (\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} \leq \mathtt{b} \wedge \mathtt{a} \leq \mathtt{b} \rangle) \wedge (\mathtt{I} < \mathtt{b})$ ⟅by octagonal disjunction⟆

$= \langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \leq j = k = \ell = \mathtt{I} < \mathtt{b} \rangle$ ⟅by abstract conjunction (45)⟆

$\Rightarrow P_3^3$ ⟅by def. (44) of abstract implication⟆

It remains to compute the loop exit invariant:

$(P_2^3 \vee P_7^3) \wedge (\mathtt{I} \geq \mathtt{b})$

$=$ $(\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \le j = k = \ell = \mathtt{I} \le \mathtt{b} \wedge \mathtt{a} \le \mathtt{b} \rangle) \wedge (\mathtt{I} \ge \mathtt{b})$ ⟅by octagonal disjunction⟆

$=$ $\langle \mathrm{lt}(\mathtt{t}, i, j, k, \ell),\ i = \mathtt{a} \le j = k = \ell = \mathtt{I} = \mathtt{b} \rangle$ ⟅by abstract conjunction (45)⟆

The static analysis has therefore discovered the following invariants:

```
        var t : array [a, b] of int;
1 :     {a ≤ b}
        I := a;
2 :     {I = a ≤ b}
        while (I < b) do
3 :         {lt(t, a, I, I, I) ∧ I < b}
            if (t[I] > t[I + 1]) then
4 :             {lt(t, a, I, I, I) ∧ I < b ∧ lt(t, I, I + 1, I, I)}
                t[I] :=: t[I + 1]
5 :             {lt(t, a, I + 1, I + 1, I + 1) ∧ I + 1 ≤ b}
            fi;
6 :         {lt(t, a, I + 1, I + 1, I + 1) ∧ I + 1 ≤ b}
            I := I + 1
7 :         {lt(t, a, I, I, I) ∧ I ≤ b}
        od
8 :     {lt(t, a, I, I, I) ∧ I = b}
```

5.8 Generic Sorting Abstract Domain

Then we define the generic sorting abstract domain:

$$\mathcal{D}_s(X) = \{\langle \mathrm{s}(\mathtt{t}, a, b),\ r\rangle \mid \mathtt{t} \in X \wedge a, b \notin X \wedge r \in \mathcal{D}_r(X \cup \{a, b\})\}\ .$$

The meaning $\gamma(\langle \mathrm{s}(\mathtt{t}, a, b),\ r\rangle)$ of an abstract predicate $\langle \mathrm{s}(\mathtt{t}, a, b),\ r\rangle$ is, informally that the elements of $\mathtt{t}$ between indices a and b are sorted:

$$\gamma(\langle \mathrm{s}(\mathtt{t}, a, b),\ r\rangle) = \exists a, b : \mathtt{t}.\ell \le a \le b \le \mathtt{t}.h \wedge \forall i, j \in [a, b] : (i \le j) \Rightarrow (\mathtt{t}[i] \le \mathtt{t}[j]) \wedge r\ .$$

5.9 Generic Comparison and Sorting Abstract Domain

The analysis of sorting algorithms involves the reduced product [19] of the generic comparison abstract domain of Sec. 5.2 and sorting abstract domain of Sec. 5.8, that is triples of the form:

$$\langle \mathrm{lt}(\mathtt{t}, a, b, c, d),\ \mathrm{s}(\mathtt{t}, e, f),\ r \rangle\ .$$

The reduction involves interactions between terms such as, e.g.:

$$\mathrm{lt}(\mathtt{t}, a, b-1, b-1, b-1) \wedge \mathrm{lt}(\mathtt{t}, a, b, b, b) \tag{58}$$
$$\Rightarrow \mathrm{s}(\mathtt{t}, b-1, b) \wedge \mathrm{lt}(\mathtt{t}, a, b-1, b-1, b)$$
$$\mathrm{s}(\mathtt{t}, b+1, c) \wedge \mathrm{lt}(\mathtt{t}, a, b+1, b+1, c) \wedge \mathrm{lt}(\mathtt{t}, a, b, b, b) \tag{59}$$
$$\Rightarrow \mathrm{s}(\mathtt{t}, b, c) \wedge \mathrm{lt}(\mathtt{t}, a, b, b, c)$$
$$\mathrm{lt}(\mathtt{t}, a, a+1, a+1, b) \wedge \mathrm{s}(\mathtt{t}, a+1, b) \Rightarrow \mathrm{s}(\mathtt{t}, a, b) \tag{60}$$
$$\tag{61}$$

The reduction [19] also involves the refinement of abstract predicate transformers (see a.o. [11, 43]) which would be performed automatically e.g. if the abstract predicate transformers are obtained by automatic simplification of the formula $\alpha \circ F \circ \gamma$ (where F is the concrete semantics) by the simplifier of a theorem prover.

5.10 Generic Comparison and Sorting Static Program Analysis

Let us consider the bubble sort [41]:

```
                var t : array [a, b] of int;
1 :
                J := b;
2 :
                while (a < J) do
3 :
                    I := a;
4 :
                    while (I < J) do
5 :
                        if (t[I] > t[I + 1]) then
6 :
                            t[I] :=: t[I + 1]
7 :
                        fi;
8 :
                        I := I + 1
9 :
                    od;
10 :
                    J := J − 1
11 :
                od
12 :
```

The fixpoint approximation is as follows ($P_p^{i,k}$ denotes the local assertion attached to program point p at the i^{th} iteration and k^{th} loop unrolling, $P_p^i = P_p^{i,0}$ where $k = 0$ means that the decision to semantically unroll the loop is not yet taken):

$P_1^0 =$ $(\mathtt{a} \leq \mathtt{b})$ ⟅initialization⟆

$P_i^0 =$ false, $i = 2, \ldots, 8$

$P_1^1 = P_1^0$

$= (\mathtt{a} \leq \mathtt{b})$ ⟅def. P_1^0⟆

$P_2^1 = (\mathtt{a} \leq \mathtt{b} = \mathtt{J})$ ⟅assignment $\mathtt{J} := \mathtt{b}$⟆

$P_3^{1,0} = (\mathtt{a} < \mathtt{b} = \mathtt{J})$ ⟅test $(\mathtt{a} < \mathtt{J})$⟆

...

$P_{10}^{1,0} = \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{I}, \mathtt{I}, \mathtt{I}) \wedge \mathtt{a} < \mathtt{b} = \mathtt{I} = \mathtt{J}$ [13] ⟅as in Sec. 5.7 since the inner loop does not modify a, b or I⟆

$\Rightarrow \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{J}, \mathtt{J}, \mathtt{b}) \wedge \mathtt{a} < \mathtt{b} = \mathtt{J}$ ⟅by elimination (octagonal projection) of program variable I which is no longer live at program point 10⟆

$P_{11}^{1,0} = \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{J}+1, \mathtt{J}+1, \mathtt{b}) \wedge \mathtt{a} < \mathtt{b} \wedge \mathtt{J} = \mathtt{b} - 1$ ⟅postcondition for assignment $\mathtt{J} := \mathtt{J} - 1$⟆

$P_3^{1,1} = \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{J}+1, \mathtt{J}+1, \mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b} - 1$ ⟅by semantical loop unrolling (since a new symbolic "lt" term has appeared, see Sec. 5.5,) and test $(\mathtt{a} < \mathtt{J})$⟆

$\ldots$

$P_{10}^{1,1} = \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{1} \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{I},\mathtt{I},\mathtt{I}) \wedge \mathtt{I} = \mathtt{J}$

$\wr$as in Sec. 5.7 since the inner loop does not modify $\mathtt{a}$, $\mathtt{b}$ or $\mathtt{I}$ and the swap $\mathtt{t}[\mathtt{I}] := \mathtt{t}[\mathtt{I}+\mathtt{1}]$ does not interfere with $\mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1})$ according to $\mathtt{a} \leq \mathtt{I} < \mathtt{I}+\mathtt{1} \leq \mathtt{J} < \mathtt{J}+\mathtt{1}$ so $\mathtt{I}, \mathtt{I}+\mathtt{1} \in [\mathtt{a}, \mathtt{J}+\mathtt{1}]$ and (53)$\int$ (62)

$\Rightarrow \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{J}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{1}$ $\wr$by elimination of $\mathtt{I}$ is dead at program point 10$\int$

$\Rightarrow \mathrm{s}(\mathtt{t},\mathtt{J},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{1}$ $\wr$by reduction (58)$\int$

$P_{11}^{1,1} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} \leq \mathtt{J} = \mathtt{b}-\mathtt{2}$ $\wr$by assignment $\mathtt{J} := \mathtt{J}-\mathtt{1}$$\int$

$P_{3}^{1,2} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2}$ $\wr$by semantical loop unrolling (since a new symbolic "s" term has appeared, see Sec. 5.5,) and test $(\mathtt{a} < \mathtt{J})$$\int$

$\ldots$

$P_{10}^{1,2} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2} \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{I},\mathtt{I},\mathtt{I}) \wedge \mathtt{I} = \mathtt{J}$ $\wr$by Sec. 5.7 and non interference, see (62)$\int$

$\Rightarrow \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2} \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{J})$ $\wr$since $\mathtt{I}$ is dead$\int$

$\Rightarrow \mathrm{s}(\mathtt{t},\mathtt{J},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2}$ $\wr$by reduction (59)$\int$

$P_{11}^{1,2} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} \leq \mathtt{J} = \mathtt{b}-\mathtt{3}$ $\wr$by assignment $\mathtt{J} := \mathtt{J}-\mathtt{1}$$\int$

$P_{3}^{2,2} = (P_{3}^{1,2} \triangledown (P_{11}^{1,2} \wedge (\mathtt{a} < \mathtt{J}))) \wedge (\mathtt{a} < \mathtt{J})$ $\wr$loop unrolling stops in absence of new abstract term and widening speeds-up convergence$\int$

$= ((\mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2}) \triangledown (\mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} \leq \mathtt{J} = \mathtt{b}-\mathtt{3} \wedge (\mathtt{a} < \mathtt{J}))) \wedge (\mathtt{a} < \mathtt{J})$ $\wr$def. $P_{3}^{1,2}$ and $P_{11}^{1,2}$$\int$

$= \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge ((\mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{2}) \triangledown (\mathtt{a} < \mathtt{J} = \mathtt{b}-\mathtt{3})) \wedge (\mathtt{a} < \mathtt{J})$ $\wr$by def. widening$\int$

$= \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} \leq \mathtt{b}-\mathtt{2}$ $\wr$by def. octagonal widening and conjunction$\int$

$\ldots$

$P_{10}^{2,2} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} \leq \mathtt{b}-\mathtt{2} \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{I},\mathtt{I},\mathtt{I}) \wedge \mathtt{I} = \mathtt{J}$ $\wr$by Sec. 5.7 and non interference, see (62)$\int$

$= \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} \leq \mathtt{b}-\mathtt{2} \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{J})$ $\wr$by elimination of the dead variable $\mathtt{I}$$\int$

$\Rightarrow \mathrm{s}(\mathtt{t},\mathtt{J},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J},\mathtt{J},\mathtt{b}) \wedge \mathtt{a} < \mathtt{J} \leq \mathtt{b}-\mathtt{2}$ $\wr$by reduction (59)$\int$

$P_{11}^{2,2} = \mathrm{s}(\mathtt{t},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathrm{lt}(\mathtt{t},\mathtt{a},\mathtt{J}+\mathtt{1},\mathtt{J}+\mathtt{1},\mathtt{b}) \wedge \mathtt{a} \leq \mathtt{J} \leq \mathtt{b}-\mathtt{3}$ $\wr$by assignment $\mathtt{J} := \mathtt{J}-\mathtt{1}$$\int$

[13] Notice that this notation is a shorthand for the more explicit notation $\exists i,j,k,\ell : \mathrm{lt}(\mathtt{t},i,j,k,\ell) \wedge i = \mathtt{a} \wedge j = \mathtt{I} \wedge k = \mathtt{I} \wedge \ell = \mathtt{I}) \wedge \mathtt{a} < \mathtt{b} \wedge \mathtt{b} = \mathtt{J} \wedge \mathtt{I} = \mathtt{J}$ as used

Now $(P_{11}^{2,2} \wedge \mathtt{a} < \mathtt{J}) \Rightarrow P_3^{1,2}$ so that the loop iterates stabilize to a post-fixpoint. On loop exit, we must collect all cases following from semantic unrolling:

$$
\begin{array}{lll}
P_{12}^2 = & (P_2^1 \wedge \mathtt{a} \geq \mathtt{J}) & \wr\text{no entry in the loop}\wr \\
& \vee\, (P_{11}^{1,0} \wedge \mathtt{a} \geq \mathtt{J}) & \wr\text{loop exit after one iteration}\wr \\
& \vee\, (P_{11}^{1,1} \wedge \mathtt{a} \geq \mathtt{J}) & \wr\text{loop exit after two iterations}\wr \\
& \vee\, (P_{11}^{2,2} \wedge \mathtt{a} \geq \mathtt{J}) & \wr\text{loop exit after three iterations or more}\wr \\
= & (\mathtt{a} = \mathtt{J} = \mathtt{b}) \vee (\mathrm{s}(\mathtt{t}, \mathtt{J}+1, \mathtt{b}) \wedge \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{J}+1, \mathtt{J}+1, \mathtt{b}) \wedge \mathtt{a} = \mathtt{J} \leq \mathtt{b}-1) & \\
& \wr\text{def. abstract disjunction}\wr & \\
= & (\mathtt{a} = \mathtt{J} = \mathtt{b}) \vee (\mathrm{s}(\mathtt{t}, \mathtt{a}{+}1, \mathtt{b}) \wedge \mathrm{lt}(\mathtt{t}, \mathtt{a}, \mathtt{a}{+}1, \mathtt{a}{+}1, \mathtt{b}) \wedge \mathtt{a} < \mathtt{b}) & \wr\text{elimination} \\
& \text{of dead variable } \mathtt{J}\wr & \\
= & (\mathtt{a} = \mathtt{b}) \vee (\mathrm{s}(\mathtt{t}, \mathtt{a}, \mathtt{b}) \wedge \mathtt{a} < \mathtt{b}) & \wr\text{by reduction (60)}\wr \\
= & \mathrm{s}(\mathtt{t}, \mathtt{a}, \mathtt{b}) \wedge \mathtt{a} \leq \mathtt{b} & \wr\text{by definition of abstract disjunction similar to} \\
& (48)\wr &
\end{array}
$$

The sorting proof would proceed in the same way by proving that the final array is a permutation of the original one.

Observe that *generic predicate abstraction* is defined for a programming language as opposed to *ground predicate abstraction* which is specific to a program, a usual distinction between abstract interpretation based static program analysis (a generic abstraction for a set of programs) and abstract model checking (an abstract model for a given program). Notice that the so-called *polymorphic predicate abstraction* of [4] is an instance of symbolic relational separate procedural analysis [26, Sec. 7] for *ground* predicate abstraction. The generalization to generic predicate abstraction is immediate since it only depends on the way concrete predicate transformers are defined (see [26, Sec. 7]).

6 Conclusion

In safety proofs by ground predicate abstraction, one has to provide (or compute by refinement) the ground atomic components of the inductive invariant which is to be discovered for the proof. Then the routine work of assembling the atomic components into a valid inductive invariant is mechanized which simplifies the proof. If the set of atomic components is finite then a boolean encoding allows for the reuse of model-checkers for fixpoint computation. Otherwise a specific fixpoint engine has to be used designed, which is mandatary even in cases as simple as constant propagation if the constants are to be dis-

in Sec. 5.7, so that, in particular, we freely replace i, j, k and ℓ in $\mathrm{lt}(\mathtt{t}, i, j, k, \ell)$ by equivalent expressions.

covered automatically and not explicitly provided in the list of ground atomic predicates.

Generic predicate abstraction provides a further abstraction step in that hints for the proof are provided in the form of parameterized atomic predicates (which will be instantiated automatically to program specific ground predicates) and reduction rules (which are hints for inductive reasoning on these generic predicates). This genericity immediately leads to infinite abstract domains (which was not the case for finitary ground predicate abstraction) which means that fixpoint iterations need more sophisticated inferences, which we can provide in the simple form of widenings. Moreover the presentation in the form of structured abstract domains, which can be systematically composed, reduces the need to appeal to theorem provers by reduction of the widening to well-studied and powerful basic relational abstract domains which can be viewed as undecidable theories with finitary extrapolation through widenings. The hope is then than generic predicate abstractions can handle families of algorithms and data structures of wider scope than a single program while avoiding the costly refinement process.

Acknowledgements I thank the participants to the meeting on predicate abstraction at New York University on Thursday Jan. 30$^{\text{th}}$ 2003 (Radhia Cousot, Dennis Dams, Kedar Namjoshi, Amir Pnueli, Lenore Zuck), in particular Amir Pnueli who proposed the bubble sort as a challenge which is handled in Sec. 5.

References

[1] R. Alur, T. Dang, and F. Ivancic. Reachability analysis of hybrid systems via predicate abstraction. In C. Tomlin and M.R. Greenstreet, editors, *Proceedings of the Fifth International Workshop on Hybrid Systems: Computation and Control, HSCC '2002*, pages 35–48, Stanford, California, United States, Lecture Notes in Computer Science 2289, 25–27 March 2002. Springer-Verlag, Berlin, Germany.

[2] R. Alur, T. Dang, and F. Ivancic. Reachability analysis of hybrid systems using counter-example guided predicate abstraction. In H. Garavel and J. Hatcliff, editors, *Proceedings of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '2003*, pages ???–???, Varsaw, Poland, United States, Lecture Notes in Computer Science ????, 7–11 April 2003. Springer-Verlag, Berlin, Germany.

[3] T. Ball, R. Majumdar, T.D. Millstein, and S.K. Rajamani. Automatic predicate abstraction of C programs. In *Proceedings of the ACM*

SIGPLAN 2001 Conference on Programming Language Design and Implementation (PLDI). ACM SIGPLAN Notices 36(5), pages 203–213. ACM Press, New York, New York, United States, June 2001.

[4] T. Ball, T. Millstein, and S.K. Rajamani. Polymorphic predicate abstraction. Technical report MSR-TR-2001-10, Microsoft Reasearch, Redmond, Washington, United States, 17 June 2002. 21 p.

[5] T. Ball, A. Podelski, and S.K. Rajamani. Relative completeness of abstraction refinement for software model checking. In J.-P. Katoen and P. Stevens, editors, *Proceedings of the Eight International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '2002*, Grenoble, France, Lecture Notes in Computer Science 2280, pages 158–172. Springer-Verlag, Berlin, Germany, 8–12 April 2002.

[6] G. Birkhoff. *Lattice Theory*, volume 25 of *Colloquium publications*. American Mathematical Society, Providence, Rhode Island, United States, third edition, 1973.

[7] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software, invited chapter. In T. Mogensen, D.A. Schmidt, and I.H. Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, Lecture Notes in Computer Science 2566, pages 85–108. Springer-Verlag, Berlin, Germany, 2002.

[8] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. In *Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 343–354, Albuquerque, New Mexico, 1992. ACM Press, New York, New York, United States.

[9] M. Colón and T.E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In A.J. Hu and M.Y. Vardi, editors, *Proceedings of the Tenth International Conference on Computer Aided Verification, CAV '98*, Vancouver, British Columbia, Canada,Lecture Notes in Computer Science 1427, pages 293–304. Springer-Verlag, Berlin, Germany, 28 June – 2 July 1998.

[10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. InterÉditions, Paris, France, 1990.

[11] A. Cortesi, B. Le Charlier, and P. van Hentenryck. Combinations of abstract domains for logic programming: open product and generic pattern construction. *Science of Computer Programming*, 38(1–3):27–71, 2000.

[12] 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.

[13] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Electronic Notes in Theoretical Computer Science*, 6, 1997.

http://www.elsevier.nl/locate/entcs/volume6.html, 25 pages.

[14] P. Cousot. Types as abstract interpretations, invited paper. In *Conference Record of the Twentyfourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 316–331, Paris, France, January 1997. ACM Press, New York, New York, United States.

[15] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science*, 277(1—2):47–103, 2002.

[16] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, New York, United States.

[17] P. Cousot and R. Cousot. Automatic synthesis of optimal invariant assertions: mathematical foundations. In *ACM Symposium on Artificial Intelligence & Programming Languages*, Rochester, New York, ACM SIGPLAN Notices 12(8):1–12, 1977.

[18] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

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

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

[21] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, August 1992.

[22] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the Ninthteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 83–94, Albuquerque, New Mexico, 1992. ACM Press, New York, New York, United States.

[23] P. Cousot and R. Cousot. Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form, invited paper. In P. Wolper, editor, *Proceedings of the Seventh International Conference on Computer Aided Verification, CAV '95*, Liège, Belgium, Lecture Notes in Computer Science 939, pages 293–308. Springer-Verlag, Berlin, Germany, 3–5 July 1995.

[24] P. Cousot and R. Cousot. Formal language, grammar and set-constraint-based program analysis by abstract interpretation. In *Proceedings of*

the Seventh ACM Conference on Functional Programming Languages and Computer Architecture, pages 170–181, La Jolla, California, 25–28 June 1995. ACM Press, New York, New York, United States.

[25] P. Cousot and R. Cousot. Temporal abstract interpretation. In *Conference Record of the Twentyseventh Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 12–25, Boston, Massachusetts, January 2000. ACM Press, New York, New York, United States.

[26] P. Cousot and R. Cousot. Modular static program analysis, invited paper. In R.N. Horspool, editor, *Proceedings of the Eleventh International Conference on Compiler Construction, CC '2002*, pages 159–178, Grenoble, France, April 6—14 2002. Lecture Notes in Computer Science 2304, Springer-Verlag, Berlin, Germany.

[27] P. Cousot and R. Cousot. Systematic design of program transformation frameworks. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 178–190, Portland, Oregon, January 2002. ACM Press, New York, New York, United States.

[28] P. Cousot and R. Cousot. Parsing as abstract interpretation of grammar semantics. *Theoretical Computer Science*, 290(1):531–544, January 2003.

[29] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, New York, United States.

[30] S. Das and D.L. Dill. Successive approximation of abstract transition relations. In *Proceedings of the Sixteenth Annual IEEE Symposium on Logic in Computer Science, LICS '88*, pages 51–60, Boston, Massachusetts, United States, June 1988. IEEE Computer Society Press, Los Alamitos, California, United States.

[31] S. Das and D.L. Dill. Counter-example based predicate discovery in predicate abstraction. In M. Aagaard and J.W. O'Leary, editors, *Proceedings of the Fourth International Conference on Formal Methods in Computer-Aided Design, FMCAD 2002*, Portland,, Oregon, United States, Lecture Notes in Computer Science 1633, pages 19–32. Springer-Verlag, Berlin, Germany, November 2002.

[32] S. Das, D.L. Dill, and S. Park. Experience with predicate abstraction. In N. Halbwachs and D. Peled, editors, *Proceedings of the Eleventh International Conference on Computer Aided Verification, CAV '99*, Trento, Italy, Lecture Notes in Computer Science 1633, pages 160–171. Springer-Verlag, Berlin, Germany, 6–10 July 1999.

[33] B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order, Second Edition*. Cambridge University Press, Cambridge, United Kindom, 2002.

[34] C. Flanagan and S. Qadeer. Predicate abstraction for software verifica-

tion. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 191–202, Portland, Oregon, January 2002. ACM Press, New York, New York, United States.

[35] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings of the Twelfth International Conference on Concurrency Theory, CONCUR '96*, number 2154 in Lecture Notes in Computer Science, pages 426–440. Springer-Verlag, Berlin, Germany, Aalborg, Denmark, 2025 August 2001.

[36] P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In E. Brinksma and K.G. Larsen, editors, *Proceedings of the Fourteenth International Conference on Computer Aided Verification, CAV '2002*, Copenhagen, Denmark, Lecture Notes in Computer Science 2404, pages 137–150. Springer-Verlag, Berlin, Germany, 27–31 July 2002.

[37] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proceedings of the Ninth International Conference on Computer Aided Verification, CAV '97*, Haifa, Israel, Lecture Notes in Computer Science 1254, pages 72–83. Springer-Verlag, Berlin, Germany, 22–25 July 1997.

[38] N. Halbwachs, J.-É. Proy, and P. Raymond. Verification of linear hybrid systems by means of convex approximations. In B. Le Charlier, editor, *Proceedings of the First International Symposium on Static Analysis, SAS '94*, Namur, Belgium, 20–22 september 1994, Lecture Notes in Computer Science 864, pages 223–237. Springer-Verlag, Berlin, Germany, 1994.

[39] I. Hwang, H. Balakrishnan, R. Ghosh, and C. Tomlin. Reachability analysis of delta-notch lateral inhibition using predicate abstraction. In S. Sahni, V.K. Prasanna, and U. Shukla, editors, *Proceedings of the Ninth International Conference on High Performance Computing, HiPC 2002*, pages 715–724, Bangalore, India, Lecture Notes in Computer Science 2552, 18–21 December 2002. Springer-Verlag, Berlin, Germany.

[40] G. Kildall. A unified approach to global program optimization. In *Conference Record of the First Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 194–206, Boston, Massachusetts, October 1973. ACMpress.

[41] D.E. Knuth. Sorting and searching. In *The Art of Computer Programming*, volume 3. Addison-Wesley Pub. Co., Reading, Massachusetts, United States, 1973.

[42] D.E. Knuth. The genesis of attribute grammars. In P. Deransart and M. Jourdan, editors, *Proceedings of the International Conference on Attribute Grammars and their Applications, WAGA '90*, Paris, France, Lecture Notes in Computer Science 461, pages 1–12. Springer-Verlag, Berlin, Germany, 19–21 september 1990.

[43] S. Lerner, D. Grove, and C. Chambers. Composing dataflow analy-

ses and transformations. In *Conference Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 270–282, Portland, Oregon, January 2002. ACM Press, New York, New York, United States.

[44] A. Miné. A new numerical abstract domain based on difference-bound matrices. In 0. Danvy and A. Filinski, editors, *Proceedings of the Second Symposium PADO '2001, Programs as Data Objects*, Århus, Denmark, 21–23 May 2001, Lecture Notes in Computer Science 2053, pages 155–172. Springer-Verlag, Berlin, Germany, 2001. `http://www.di.ens.fr/~mine/publi/article-mine-padoII.pdf`.

[45] A. Miné. A few graph-based relational numerical abstract domains. In M. Hermenegildo and G. Puebla, editors, *SAS'02*, volume 2477 of *Lecture Notes in Computer Science*, pages 117–132. Springer-Verlag, Berlin, Germany, 2002. `http://www.di.ens.fr/~mine/publi/article-mine-sas02.pdf`.

[46] M.O. Möller, H. Rueß, and M. Sorea. Predicate abstraction for dense real-time systems. *Electronic Notes in Theoretical Computer Science*, 65(6), 2002. http://www.elsevier.com/locate/entcs/volume65.html.

[47] J.H. Morris and B. Wegbreit. Subgoal induction. *Communications of the Association for Computing Machinary*, 20(4):209–222, April 1977.

[48] A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proceedings of the ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation, PEPM '93*, Copenhagen, Denmark, 14–16 June 1993, pages 80–87. ACM Press, New York, New York, United States, 1993.

[49] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, september 1981.

[50] H. Saïdi and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *Proceedings of the Eleventh International Conference on Computer Aided Verification, CAV '99*, Trento, Italy, Lecture Notes in Computer Science 1633, pages 443–454. Springer-Verlag, Berlin, Germany, 6–10 July 1999.

[51] D.S. Scott and C. Strachey. Towards a mathematical semantics for computer languages. In *Proceedings of the Symposium on Computers and Automata*, volume 21 of *Microwave Research Institute Symposia Series*, 1971.

[52] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–310, 1955.

[53] W. Visser, S.J. Park, and J. Penix. Using predicate abstraction to reduce object oriented programs for model checking. In *Proceedings of the Third ACM SIGSOFTWorkshop on Formal Methods in Software Practice, FMSP 2000*, pages 2–12, Portland, Oregon, United States, 2000. ACM Press, New York, New York, United States.

[54] M. Ward. The closure operators of a lattice. *Annals of Mathematics*, 43:191–196, 1942.

A Proofs

Proof of (1). We assume that f and $\overline{f}$ are respective monotone operators on complete lattice $\langle L, \leq, \wedge\rangle$ and $\langle M, \sqsubseteq, \sqcap\rangle$ such that $\langle L, \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq\rangle$. We assume that $\alpha \circ f \circ \gamma \mathbin{\dot{\sqsubseteq}} \overline{f}$, $\alpha(a) \sqsubseteq y$ and $\overline{f}(y) \sqsubseteq y$ and prove that $\mathrm{lfp}_a^{\leq} f \leq \gamma(y)$, the least such y being $\mathrm{lfp}^{\sqsubseteq}_{\alpha(a)} \alpha \circ f \circ \gamma$ so that the theorem is proved with $\alpha = \rho$ and $gamma = 1$.

$\gamma \circ \alpha$ is extensive and γ monotone so $a \leq \gamma \circ \alpha(a) \leq \gamma(y)$, $f(\gamma(y)) \leq \gamma \circ \alpha \circ f(\gamma(y)) \leq \gamma \circ \overline{f}(y) \leq \gamma(y)$ and $\mathrm{lfp}_a^{\leq} f$ is $\mathrm{lfp}^{\leq} f$ on the complete lattice $\langle \{x \in L \mid a \leq x\}, \leq, \wedge\rangle$ so by Tarski's fixpoint theorem [52], $\mathrm{lfp}_a^{\leq} f = \bigwedge\{x \in L \mid a \leq x \wedge f(x) \leq x\} \leq \gamma(y)$ by def. of glbs.

We let $z = \mathrm{lfp}^{\sqsubseteq}_{\alpha(a)} \alpha \circ f \circ \gamma$ so that $\alpha \circ f \circ \gamma \mathbin{\dot{\sqsubseteq}} \alpha \circ f \circ \gamma$, $\alpha(a) \sqsubseteq z$ and $\alpha \circ f \circ \gamma = y$ whence $\alpha \circ f \circ \gamma \sqsubseteq y$ by reflexivity. We have $\alpha(a) \sqsubseteq y$ and $\alpha \circ f \circ \gamma(y) \sqsubseteq \overline{f}(y) \sqsubseteq y$ proving, by Tarski's fixpoint theorem [52], that $z \sqsubseteq \bigsqcap\{x \in L \mid \alpha(a) \sqsubseteq x \wedge \alpha \circ f \circ \gamma(x) \sqsubseteq x\} \sqsubseteq y$. ■

Proof of (14).

$$
\begin{array}{lll}
 & \alpha_{\downarrow}(P) \mathbin{\dot{\subseteq}} Q & \\
\Leftrightarrow & \forall \ell \in \mathcal{L} : \alpha_{\downarrow}(P)\ell \subseteq Q_\ell & \text{⟅by def. } \dot{\subseteq}\text{⟆} \\
\Leftrightarrow & \forall \ell \in \mathcal{L} : \{m \mid \langle \ell, m\rangle \in P\} \subseteq Q_\ell & \text{⟅by def. } \alpha_{\downarrow}\text{⟆} \\
\Leftrightarrow & \forall \ell \in \mathcal{L} : \forall m \in \mathcal{M} : \langle \ell, m\rangle \in P \Rightarrow m \in Q_\ell & \text{⟅by def. } \subseteq\text{⟆} \\
\Leftrightarrow & P \subseteq \{\langle \ell, m\rangle \mid \ell \in \mathcal{L} \wedge m \in Q_\ell\} & \text{⟅by def. } \subseteq\text{⟆} \\
\Leftrightarrow & P \subseteq \gamma_{\downarrow}(Q) & \text{⟅by def. } \gamma_{\downarrow}\text{⟆}
\end{array}
$$

Moreover for all $Q \in \mathcal{L} \mapsto \wp(\mathcal{M})$:

$$
\begin{array}{lll}
 & \alpha_{\downarrow}(\gamma_{\downarrow}(Q)) & \\
= & \lambda\ell\bullet\{m \mid \langle \ell, m\rangle \in \gamma_{\downarrow}(Q)\} & \text{⟅by def. } \alpha_{\downarrow}\text{⟆} \\
= & \lambda\ell\bullet\{m \mid \langle \ell, m\rangle \in \{\langle \ell, m\rangle \mid \ell \in \mathcal{L} \wedge m \in Q_\ell\}\} & \text{⟅by def. } \gamma_{\downarrow}\text{⟆} \\
= & \lambda\ell\bullet\{m \mid m \in Q_\ell\} & \text{⟅by def. } \in \text{ and } Q \in \mathcal{L} \mapsto \wp(\mathcal{M}) \text{ so } \ell \in \mathcal{L}\text{⟆} \\
= & Q & \text{⟅since } Q \in \mathcal{L} \mapsto \wp(\mathcal{M})\text{⟆}
\end{array}
$$

and for all $P \in \wp(\langle \mathcal{L}, \mathcal{M}\rangle)$:

$$\gamma_{\downarrow}(\alpha_{\downarrow}(P))$$

$$
\begin{array}{lll}
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge m \in \alpha_{\downarrow}(P)_{\ell}\} & \wr\text{by def. } \gamma_{\downarrow}\int \\
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge m \in \{m \mid \langle \ell,\, m\rangle \in P\}\} & \wr\text{by def. } \alpha_{\downarrow}\int \\
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge \langle \ell,\, m\rangle \in P\} & \wr\text{by def. } \in\int \\
= & P & \wr\text{since } P \in \wp(\langle \mathcal{L},\, \mathcal{M}\rangle) \text{ so } \ell \in \mathcal{L}\ .\int
\end{array}
$$

■

Proof of (16).

$$
\begin{array}{ll}
& \alpha_{\mathfrak{P}}(Q) \supseteq P \\
\Leftrightarrow & P \subseteq \alpha_{\mathfrak{P}}(Q) \qquad \wr\text{by inversion}\int \\
\Leftrightarrow & P \subseteq \{\mathfrak{p} \in \mathfrak{P} \mid Q \subseteq \mathcal{B}[\![\mathfrak{p}]\!]\} \qquad \wr\text{by def. } \alpha_{\mathfrak{P}}\int \\
\Leftrightarrow & \forall p \in P : Q \subseteq \mathcal{B}[\![p]\!] \qquad \wr\text{by def. } \subseteq \text{ and } P \in \wp(\mathfrak{P})\int \\
\Leftrightarrow & Q \subseteq \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in P\} \qquad \wr\text{by def. } \cap\int \\
\Leftrightarrow & Q \subseteq \gamma_{\mathfrak{P}}(P) \qquad \wr\text{by def. } \gamma_{\mathfrak{P}}\int
\end{array}
$$

■

Proof of (22). We get $\langle \wp(\mathcal{L}\times\mathcal{M}), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftleftarrows\!\!\rightrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathfrak{P}), \dot{\supseteq}\rangle$ by composition of $\langle \wp(\mathcal{L} \times \mathcal{M}),\ \subseteq\rangle \underset{\alpha_{\downarrow}}{\overset{\gamma_{\downarrow}}{\leftleftarrows\!\!\rightrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathcal{M}),\ \dot{\subseteq}\rangle$ and $\langle \mathcal{L} \mapsto \wp(\mathcal{M}),\ \dot{\subseteq}\rangle \underset{\dot{\alpha}_{\mathfrak{P}}}{\overset{\dot{\gamma}_{\mathfrak{P}}}{\leftleftarrows\!\!\rightrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathfrak{P}),\ \dot{\supseteq}\rangle$.

For all $P \in \mathcal{L} \mapsto \wp(\mathfrak{P})$, we have

$$
\begin{array}{lll}
& \alpha(P) & \\
= & \dot{\alpha}_{\mathfrak{P}} \circ \alpha_{\downarrow}(P) & \wr\text{by def. } \alpha\int \\
= & \dot{\alpha}_b(\dot{\alpha}_{\mathfrak{P}}(\alpha_{\downarrow}(P))) & \wr\text{by def. } \circ\int \\
= & \dot{\alpha}_b(\dot{\alpha}_{\mathfrak{P}}(\lambda\ell\bullet\{m \mid \langle \ell,\, m\rangle \in P\})) & \wr\text{by def. } \alpha_{\downarrow}\int \\
= & \lambda\ell\bullet\, \alpha_{\mathfrak{P}}(\{m \mid \langle \ell,\, m\rangle \in P\})) & \wr\text{by def. } \dot{\alpha}_{\mathfrak{P}}\int \\
= & \lambda\ell\bullet\, \alpha_{\mathfrak{P}}(\{m \mid \langle \ell,\, m\rangle \in P\})) & \wr\text{by def. } \dot{\alpha}_b\int \\
= & \lambda\ell\bullet\{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell,\, m\rangle \in P\} \subseteq \mathcal{B}[\![\mathfrak{p}]\!]\} & \wr\text{by def. } \alpha_{\mathfrak{P}}\int \\
= & \lambda\ell\bullet\{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell,\, m\rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\} & \wr\text{by def. } \mathcal{I}\int
\end{array}
$$

The same way for all $Q \subseteq \mathfrak{P}$, we have:

$$
\begin{array}{lll}
& \gamma(Q) & \\
= & \gamma_{\downarrow} \circ \dot{\gamma}_{\mathfrak{P}}(Q) & \text{⟅by def. } \gamma\text{⟆} \\
= & \gamma_{\downarrow}(\lambda \ell \bullet \gamma_{\mathfrak{P}}(Q_\ell))) & \text{⟅by def. } \dot{\gamma}_{\mathfrak{P}}\text{⟆} \\
= & \gamma_{\downarrow}(\lambda \ell \bullet \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in Q_\ell\}) & \text{⟅by def. } \gamma_{\mathfrak{P}}\text{⟆} \\
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge m \in \bigcap\{\mathcal{B}[\![\mathfrak{p}]\!] \mid \mathfrak{p} \in Q_\ell\}\} & \text{⟅by def. } \gamma_{\downarrow}\text{⟆} \\
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge \forall \mathfrak{p} \in Q_\ell : m \in \mathcal{B}[\![\mathfrak{p}]\!]\} & \text{⟅by def. } \cap\text{⟆} \\
= & \{\langle \ell,\, m\rangle \mid \ell \in \mathcal{L} \wedge \forall \mathfrak{p} \in Q_\ell : m \in \mathcal{I}[\![\mathfrak{p}]\!]\} & \text{⟅by def. } \mathcal{I}\text{⟆}
\end{array}
$$

■

Proof of (26).

$$
\begin{array}{lll}
& \alpha_b(P) \mathrel{\dot{\Leftarrow}} Q & \\
\Leftrightarrow & \prod_{i=1}^{k}(\mathfrak{p}_i \in P) \mathrel{\dot{\Leftarrow}} Q & \text{⟅by def. } \alpha_b\text{⟆} \\
\Leftrightarrow & \forall i : 1 \leq i \leq k : (\mathfrak{p}_i \in P) \Leftarrow Q_i & \text{⟅by def. } \dot{\Leftarrow}\text{⟆} \\
\Leftrightarrow & \forall i : 1 \leq i \leq k : Q_i \Rightarrow (\mathfrak{p}_i \in P) & \text{⟅by inversion of } \Leftarrow\text{⟆} \\
\Leftrightarrow & \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge Q_i\} \subseteq P & \text{⟅by def. } \subseteq \text{ and } P \subseteq \mathfrak{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}\text{⟆} \\
\Leftrightarrow & P \supseteq \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge Q_i\} & \text{⟅by inversion of } \subseteq\text{⟆} \\
\Leftrightarrow & P \supseteq \gamma_b(Q) & \text{⟅by def. } \gamma_b\text{⟆}
\end{array}
$$

Moreover:

$$
\begin{array}{lll}
& \alpha_b(\gamma_b(Q)) & \\
= & \prod_{i=1}^{k}(\mathfrak{p}_i \in \gamma_b(Q)) & \text{⟅by def. } \alpha_b\text{⟆} \\
= & \prod_{i=1}^{k}(\mathfrak{p}_i \in \{\mathfrak{p}_j \mid 1 \leq j \leq k \wedge Q_j\}) & \text{⟅by def. } \gamma_b\text{⟆} \\
= & \prod_{i=1}^{k} Q_i & \text{⟅by def } \in \text{ and } 1 \leq i \leq k\text{⟆} \\
= & Q & \text{⟅since } Q \in \textstyle\prod_{i=1}^{k} \mathbb{B}\text{⟆}
\end{array}
$$

and:

$$
\begin{array}{lll}
& \gamma_b(\alpha_b(P)) & \\
= & \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge \alpha_b(P)_i\} & \text{⟅by def. } \gamma_b\text{⟆} \\
= & \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge \mathfrak{p}_i \in P\} & \text{⟅by def. } \alpha_b\text{⟆}
\end{array}
$$

$= P$ ⦇since $P \subseteq \mathfrak{P} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_k\}$⦈

■

Proof of (27). We get $\langle \wp(\mathcal{L} \times \mathcal{M}), \subseteq \rangle \underset{\alpha'}{\overset{\gamma'}{\leftrightarrows}} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathfrak{B}, \dot{\Leftarrow} \rangle$ by composition of $\langle \wp(\mathcal{L} \times \mathcal{M}), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathcal{L} \mapsto \wp(\mathfrak{P}), \dot{\supseteq} \rangle$ and $\langle \mathcal{L} \mapsto \wp(\mathfrak{P}), \dot{\supseteq} \rangle \underset{\dot{\alpha}_b}{\overset{\dot{\gamma}_b}{\leftrightarrows}} \langle \mathcal{L} \mapsto \prod_{i=1}^{k} \mathbb{B}, \dot{\Leftarrow} \rangle$.

For all $P \in \mathcal{L} \mapsto \wp(\mathfrak{P})$, we have

$$
\begin{array}{ll}
& \alpha'(P) \\
= & \dot{\alpha}_b \circ \alpha(P) & \text{⦇by def. } \alpha'\text{⦈} \\
= & \dot{\alpha}_b(\lambda \ell \bullet \{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell, m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}) & \text{⦇by def. } \alpha\text{⦈} \\
= & \lambda \ell \bullet \alpha_b(\{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell, m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}) & \text{⦇by def. } \dot{\alpha}_b\text{⦈} \\
= & \lambda \ell \bullet \prod_{i=1}^{k} (\mathfrak{p}_i \in \{\mathfrak{p} \in \mathfrak{P} \mid \{m \mid \langle \ell, m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}]\!]\}) & \text{⦇by def. } \alpha_b\text{⦈} \\
= & \lambda \ell \bullet \prod_{i=1}^{k} (\{m \mid \langle \ell, m \rangle \in P\} \subseteq \mathcal{I}[\![\mathfrak{p}_i]\!]) & \text{⦇since } \mathfrak{p}_i \in \mathfrak{P}\text{⦈}
\end{array}
$$

The same way for all $Q \in \mathcal{L} \mapsto \prod_{i=1}^{k} \mathbb{B}$, we have:

$$
\begin{array}{ll}
& \gamma'(Q) \\
= & \gamma \circ \dot{\gamma}_b(Q) & \text{⦇by def. } \gamma'\text{⦈} \\
= & \gamma \circ \dot{\gamma}_b(Q) & \text{⦇by def. } \gamma\text{⦈} \\
= & \gamma(\dot{\gamma}_b(Q)) & \text{⦇by def. } \circ\text{⦈} \\
= & \gamma(\lambda \ell \bullet \gamma_b(Q_\ell)) & \text{⦇by def. } \dot{\gamma}_b\text{⦈} \\
= & \gamma(\lambda \ell \bullet \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge (Q_\ell)_i\}) & \text{⦇by def. } \gamma_b\text{⦈} \\
= & \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge \forall \mathfrak{p} \in \{\mathfrak{p}_i \mid 1 \leq i \leq k \wedge (Q_\ell)_i\} : m \in \mathcal{I}[\![\mathfrak{p}]\!]\} & \text{⦇by def. } \gamma\text{⦈} \\
= & \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge \bigwedge_{i=1}^{n} (Q_\ell)_i \Rightarrow m \in \mathcal{I}[\![\mathfrak{p}_i]\!]\} & \text{⦇by def. } \in\text{⦈} \\
= & \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge m \in \bigcap_{i=1}^{n} (\!|Q_\ell)_i \mathbin{?} \mathcal{I}[\![\mathfrak{p}_i]\!] \mathbin{:} \mathcal{M}|\!)\} & \text{⦇by def. } \cap \text{ and conditional}\text{⦈} \\
= & \{\langle \ell, m \rangle \mid \ell \in \mathcal{L} \wedge m \in \mathcal{I}[\![\bigwedge_{i=1}^{k} (Q_\ell)_i \Rightarrow \mathfrak{p}_i]\!]\} & \text{⦇by def. } \mathcal{I}\text{⦈}
\end{array}
$$

■