UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI MATEMATICA
"TULLIO LEVI-CIVITA"

Corso di Laurea Magistrale in Matematica

Collecting Operational Abstract Interpreters

Relatore:
Prof. Francesco Ranzato

Laureando:
Dott. Enrico Gallana
Matricola: 1206960

11 Dicembre 2020
Anno Accademico 2020/2021

Contents

1 Introduction

The theory of abstract interpretation, introduced by Cousot and Cousot in 1977, is a general theory of the approximation of formal program semantics.
It is a useful tool to prove the accuracy of static analysis and permits to express mathematically the link between the output of practical, approximate analysis and the original uncomputable program semantics.
Given a programming language, abstract interpretation consists of giving several semantics linked by a relation of abstraction; a semantics is intended to be the mathematical characterization of a program's possible behavior.

There are several approaches to semantics each one focused on different properties of a given program. For instance, operational semantics focuses on *how* to execute a program, and in particular, structural operational semantics deals with *how* the single step of the computation takes place. On the other hand, the denotational approach is merely interested in the *effect* of the program's computation, i.e., to find a relationship between input and output data passing through mathematical structures. It is clear that, since these two approaches are different, the final results must be coherent with each other, and this induces to speculate they may be considered equivalent in a suitable sense.

Thanks to its streamlined notions and proofs, the denotational approach to abstract interpretation has already been studied many times, and many properties have been developed and well formalized. Instead, the operational correspondent versions have rarely been strictly formalized and proved, even if intuitively accepted as true. The main purpose of this work is to fill this gap: to study in detail the operational approach to abstract interpretation and to formalize in this particular setting some of the well-known denotational properties, providing mathematical proofs.

In practice, using a standard WHILE programming language, we define several operational interpreters, i.e., theoretical models that rule the correspondent transition relations between computational states of a given program. We start with the concrete operational interpreter, which models the computation of a program on precise input data through transitions between computational states, which are two-entries tuples: on the first entry, it records the remaining instructions to be executed, and on the second, the current store properties of the variables.

We then move towards a 'collecting' view: the single input data is replaced with a set of values, and we analyze the propagation of information to the different computational states. In this perspective, we define the parallel collecting interpreter and the sequential collecting interpreter, which differ in the modeling of the conditional construct if-then-else; in case of termination, the two modes turn out to be equivalent (Theorem 4.3).
The transitions between computational states occur through the so-called transfer functions: functions that modify the numerical properties stored in the variables following the execution of the commands. For instance consider the command $x := x + 1;$. We write

$$\langle x := x + 1;\ ,\ x/5\rangle \to \langle \epsilon,\ x/6\rangle,$$

where ϵ denotes the empty string of commands, and $\{x/6\} = [\![x := x + 1;]\!]\{x/5\}$ is the image of $\{x/5\}$ through the transfer function for assignments.
If we replace the singleton $\{x/5\}$ with $\mathcal{P} = \{x/0, x/5, x/6\}$,

$$\langle \epsilon, x := x + 1; , \mathcal{P} \rangle \rightarrow_{\mathrm{pc}} \langle \epsilon, \epsilon, \{x/1, x/6, x/7\} \rangle,$$

where $\rightarrow_{\mathrm{pc}}$ denotes the transition relation of parallel collecting interpreter and
$\{x/1, x/6, x/7\} = [\![x := x + 1;]\!]\ \mathcal{P}$. The first entry of the tuples is used for Stack: it records variable properties in particular situations, e.g. loops.

The main objects we deal with are the so-called invariants: sets whose elements are all the possible taken values by a certain variable at a precise program point. Invariants include even the most improbable values that, in the worst case, a variable might take.

A crucial part of this work is the computation analysis on a generic abstract domain, a lattice with particular correspondence properties with the set of variables values. *Soundness* denotes over-approximation rather than under-approximation.
We define the abstract correspondent of the parallel collecting interpreter and observe that the abstract transfer functions are the abstraction of the concrete ones. Hence, their definition determines the accuracy of the variable properties approximation: in Theorems 5.8 and 5.15 we establish the relationship between concrete and abstract invariants at each program point.

In the end, we compare the operational approach to computation modeling with the standard denotational one of static analysis. Although, operational collecting interpreters are not convergent machines, since they follow step-by-step the program's computation. The static analysis denotational approach, on the other hand, forces, through tools like widening, the identification of fixed points and hence of invariants.
This issue is overcome with the definition of a further operational interpreter, which opens a non-deterministic branch whenever it exhausts the instructions contained in a loop body. In this way, it is possible to determine invariants at each program point even with the operational approach. Not only: in the last chapter, we show the equivalence between the standard denotational static analysis approach and the operational non-deterministic collecting approach we have carried out.

In most of the proofs, we use a specific proof technique called *structural induction*. We believe it is necessary to explain how it works here. All references will be adequately clarified below.
Structural induction exploits the compositional definitions of language syntax, i.e., the distinction between *basis elements* and *composite elements*.
In practice, it consists of proving the thesis firstly for basis elements and at a second time for composite elements, assuming it true for their immediate constituents (*inductive hypothesis*).
We define WHILE-language syntax in continuation-style; that is why proofs may usually consist of two nested structural induction algorithms: the outermost running on Stm (Statements) and the innermost on Cmd (Commands). Stm has basis element ϵ, and $cS,\ c \in \mathrm{Cmd}$ as composite one. Hence, the

inductive hypothesis consists of assuming the thesis true for $S \in \mathrm{Stm}$, and prove it for $cS,\ c \in \mathrm{Cmd}$: here it starts structural induction on Cmd. It has the skip-command and the assignments as basis elements, and the if-then-else and while-do constructs as composite ones.

The rest of the thesis is organized as follows:

- Section 2 contains necessary mathematical backgrounds; it is introduced and explained the notion of *soundness*, and the abstract interpretation elements we are working with;
- Section 3 contains the definition of WHILE-language, the concrete interpreter, the transition relation, and transfer functions of semantics;
- Section 4 contains definitions and properties of the collecting interpreters; it is also introduced the notion of invariant in two versions, which are proved to be equivalent;
- Section 5 contains the definition of the abstract interpreter and the main theorems as the *Soundness theorem*, the *Completeness theorem*, and the non-deterministic version of *Soundness theorem*;
- Section 6 contains the proof of the equivalence between operational and denotational approaches.

2 Background

We now proceed with a review of the mathematical foundations of abstract interpretation. We introduce notations, definitions, key theorems we are using in the rest of the thesis.

2.1 Order theory

The main structure we require is a partially ordered set, where:

Definition 2.1. A *partial order* '$\leq$' on a set X is a binary relation $\leq \subseteq X \times X$ that is:

- reflexive: $\forall x \in X \,.\, x \leq x$;
- anti-symmetric: $\forall x, y \in X \,.\, x \leq y \ \wedge \ y \leq x \ \Rightarrow \ x = y$;
- transitive: $\forall x, y, z \in X \,.\, x \leq y \ \wedge \ y \leq z \ \Rightarrow \ x \leq z$.

We denote with $(X, \leq)$ the set X equipped with the partial order $\leq$ and we call this pair a *poset*.
A partial order is *total* when for every $x, y \in X$ either $x \leq y$ or $y \leq x$ holds.

In Abstract Interpretation partial orders are used to model different concepts: the idea of *approximation*, *soundness* and *iterations*.

Let now introduce the notions of *lower and upper bounds.*
Let X be a poset and consider two elements $x, y \in X$. An *upper bound* of x and y is an element $z \in X$ such that $x \leq z \wedge y \leq z$. It is the *least upper bound (lub)* if for any other upper bound $w \in X$ it holds $z \leq w$, i.e. z is the smallest element greater than both x and y.
Likewise, it is possible to define the notions of *lower bound* and of *greater lower bound (glb)*.
We denote rispectively with $\top$ (called *top*) and $\bot$ (called *bottom*) the greatest and least element of the poset, if they exist.

Some useful definitions:

- A *chain* in a poset $(X, \leq)$ is a subset C of X that is totally ordered, i.e.

$$\forall x, y \in C \,.\, x \leq y \vee y \leq x.$$

- A *complete partial order (CPO)* is a poset in which every chian admits the lub.
  Notice that $\emptyset$ is supposed to be a chain, and by convention we set $\vee\emptyset = \top$.
- A *lattice* $(X, \leq, \vee, \wedge)$ is a poset such that $\forall x, y \in X \,.\, x \vee y$ and $x \wedge y$ exist.
- A *complete lattice* $(X, \leq, \vee, \wedge, \top, \bot)$ is a poset such that:
  - $\forall A \subseteq X. \bigvee A$ exists;

- $\forall A \subseteq X \,.\, \bigwedge A$ exists;
- X has a greatest element $\top$;
- X has a least element $\bot$.

Notice that a complete lattice is both a lattice and a poset.

2.2 Fixpoints

Definition 2.2. Let $(X, \leq)$ a poset and $f : X \to X$ a function. $x \in X$ is a fixed point for f if $f(x) = x$.
We define the set

$$fp(f) \triangleq \{x \in X \mid f(x) = x\}.$$

If such a set has a *minimum*, it is called *least fixed point (lfp)* of f.
Dually, it is possible to define the notions of *greatest fixed point (gfp).*

Obviously, fixpoints don't neccessarily exist. To ensure their existence we need some extra hypothesis on the function f.

Definition 2.3. A function between two posets $f : (X_1, \leq_1) \to (X_2, \leq_2)$ is *monotonic* if

$$\forall x, y \in X_1 \,.\, x \leq_1 y \Rightarrow f(x) \leq_2 f(y).$$

A function between two CPOs $f : (X_1, \leq_1, \vee_1) \to (X_2, \leq_2, \vee_2)$ is *continuos* if for every chain $C \subseteq X_1$ then $f(C) = \{f(c) \mid c \in C\} \subseteq X_2$ is also a chain, and the limits coincide, namely:

$$f(\vee_1 C) = \vee_2 f(C).$$

Observe that continuity implies monotonicity: assume $x, y \in X_1$ such that $x \leq_1 y$. The set $\{x, y\}$ is a chain, hence, by continuity of f. $f(\{x, y\}) = \{f(x), f(y)\}$ is a chain. Moreover

$$f(y) = f(\vee_1\{x, y\}) = \vee_2 f(\{x, y\}) = \vee_2\{f(x), f(y)\}.$$

Hence $f(x) \leq_2 f(y)$.

Let us recall a very significant theorem; the proof is omitted.

Theorem 2.4. *(Knaster-Tarski) Let $f : X \to X$ be a continuos function on a CPO $(X, \leq, \vee, \top)$ with least element $\bot$.*
Then $lfp\, f$ does exist. Moreover

$$lfp\, f \;= \bigvee_{i \geq 0} f^i(\bot).$$

2.3 Abstract Domains

Partially ordered sets are used to model an amount of information. Thus, we assume $(C, \leq)$ and $(A, \sqsubseteq)$ be two posets. We refer to them respectively as concrete and abstract domain.
The minimum connection between these two is a *concretization* function, usually denoted as γ.

Definition 2.5. A *concretization* function $\gamma : A \to C$ is a monotonic poset function assigning a concrete meaning, in C, to each abstract element of A.

It is now possible to formalize what a sound abstraction means.
Given $c \in C$, $a \in A$ is a *sound abstraction* of c if and only if $c \leq \gamma(a)$. It is moreover an *exact abstraction* if $c = \gamma(a)$.

There are more complex structures specifically made to design sound and accurate analyses. Firstly, we assume the existence of a monotonic *abstraction function* $\alpha : C \to A$. The resulting structure is called *Galois connection*:

Definition 2.6. Given two posets $(C, \leq)$ and $(A, \sqsubseteq)$, the pair $(\alpha : C \to A\ ,\ \gamma : A \to C)$ is a *Galois connection* if

$$\forall a \in A,\ c \in C,\ c \leq \gamma(a) \iff \alpha(c) \sqsubseteq a$$

which is denoted as (C, A, γ, α).
Moreover, α and γ are said to be *adjoint functions*.

The characterizing property of the Galois connection provides a very strong connection between concrete and abstract domains. Here is an equivalent characterization:

Theorem 2.7. *(C, A, γ, α) is a Galois connection if and only if the function pair (α, γ) satisfies the following properties:*

- *γ is monotonic;*
- *α is monotonic;*
- *$\gamma \circ \alpha$ is extensive, i.e. $\forall c \in C\ .\ c \leq \gamma \circ \alpha(c)$;*
- *$\alpha \circ \gamma$ is reductive, i.e. $\forall a \in A\ .\ \alpha \circ \gamma(a) \sqsubseteq a$.*

Extensivity of $\gamma \circ \alpha$ denotes a loss of accuracy by giving the concrete representation of the abstraction of a concrete element. The loss is effective when a concrete element has no exact abstract representation.

If we require $\alpha \circ \gamma$ to be the identity, we obtain a *Galois insertion (GI).*

Definition 2.8. A Galois connection (C, A, γ, α) is a *Galois insertion* if any of the following, equivalent properties hold:

- α is surjective: $\forall a \in A\ \exists c \in C\ .\ \alpha(c) = a$;

- γ is injective: $\forall a,\ a' \in A\ \ \gamma(a) = \gamma(a') \Rightarrow a = a'$;
- $\alpha \circ \gamma = id$.

The first and second properties state that no elements of A are superfluous. Third property allows us to view the abstract domain A as isomorphic to a subset of the concrete domain C.
Notice that it is always possible to derive a Galois insertion from a Galois connection, simply restricting A to $\alpha(C)$.

Let us now state how to obtain the abstract approximation of a poset function $f : C \to C$.

Definition 2.9. With the usual notations, let $\gamma : A \to C$, $f : C \to C$ and $g : A \to A$:

- g is a *sound abstraction* of f if $\ \forall a \in A\ .\ f(\gamma(a)) \leq \gamma(g(a))$;
- g is an *exact abstraction* of f if $f \circ \gamma = \gamma \circ g$.

Notice that an exact abstraction is always sound.
If we have a Galois connection, we can talk about best abstraction of functions:

Definition 2.10. Given a Galois connection (C, A, γ, α) and a function $f : C \to C$, the *best correct approximation (bca)* of f is given by $\alpha \circ f \circ \gamma$.
In particular, every sound abstraction of f is always greater than $\alpha \circ f \circ \gamma$.

Next, we want to describe one of the most popular abstract domains: the Interval domain.

2.3.1 Interval Abstract Domain

The interval domain is based on interval arithmetic and adapted to static analysis by Cousot and Cousot. It is simple and inexpensive, and yet it can express and infer valuable properties for program verification.
For these reasons, we are using it in all further examples.

The interval domain abstracts the set of possible values of a variable as an interval. The abstract values are either non-empty intervals with finite or infinite bounds, or $\bot_{\text{Int}}$:

$$\text{Int} = \{[a, b] \mid a \in \mathbb{Z} \cup \{-\infty\},\ b \in \mathbb{Z} \cup \{+\infty\},\ a \leq b\} \cup \{\bot_{\text{Int}}\}$$

Obviously, the greatest element $\top_{\text{Int}}$ is given by $[-\infty, +\infty]$.

We have a Galois insertion:

- the partial order is defined as follows.

$$[a, b] \sqsubseteq_{\text{Int}} [c, d] \iff (a \geq c) \wedge (b \leq d)$$
$$[a, b] \vee_{\text{Int}} [c, d] = [min(a, c),\ max(b, d)]$$
$$[a, b] \wedge_{\text{Int}} [c, d] = \begin{cases} [max(a, c),\ min(b, d)] & \text{if } max(a, c) \leq min(b, d) \\ \bot_{\text{Int}} & \text{otherwise} \end{cases}$$

- Concretization function $\gamma : \mathrm{Int} \to \wp(\mathbb{Z})$ acts as follows:

$$\gamma(\bot_{\mathrm{Int}}) = \emptyset$$
$$\gamma([a,b]) = \{x \in \mathbb{Z} \mid a \leq x \leq b\}$$

- Abstraction function $\alpha : \wp(\mathbb{Z}) \to \mathrm{Int}$:

$$\alpha(X) = \begin{cases} \bot_{\mathrm{Int}} & \text{if } X = \emptyset \\ [min\ X,\ max\ X] & \text{otherwise} \end{cases}$$

- The condition of Definition 2.6 is satisfied: fix $X \in \wp(\mathbb{Z})$ and $[a,b] \in \mathrm{Int}$.
  Observe
$$X \subseteq \gamma([a,b]) \Leftrightarrow X \subseteq \{x \mid a \leq x \leq b\} \Leftrightarrow \alpha(X) \sqsubseteq [a,b]$$

- α is surjective: fix $[a,b] \in \mathrm{Int}$ and consider $X = \{a,b\} \in \wp(\mathbb{Z})$. Then $\alpha(X) = [a,b]$.

3 Language

We define a standard WHILE-language whose continuation-style syntax and small-step operational semantics are inspired by [7] and precisely defined in [5].

$$
\begin{array}{l}
\mathrm{Int} \ni v \qquad \mathrm{Vars} \ni x \\
\mathrm{Aexp} \ni E ::= v \mid x \mid E_1 \,\mathrm{op}\, E_2 \\
\mathrm{Bexp} \ni B ::= \mathbf{tt} \mid \mathbf{ff} \mid E_1 \leq E_2 \mid \neg B \mid B_1 \wedge B_2 \\
\mathrm{Cmd} \ni c ::= \mathbf{skip};\ \mid x := E;\ \mid \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ B\ \mathbf{do}\ S \\
\mathrm{Stm} \ni S ::= \epsilon \mid cS
\end{array}
$$

where ϵ stands for the empty string. Thus, any statement $S \in \mathrm{Stm}$ is a possibly empty sequence of commands c^n, with $n \geq 0$. An empty body in some if-then-else or while command will be semantically equivalent to have a no-op **skip** body. We follow [5] in making an abuse in program syntax by assuming that if $S_1, S_2 \in \mathrm{Stm}$ then $S_1 S_2 \in \mathrm{Stm}$, where $S_1 S_2$ denotes a simple string concatenation of S_1 and S_2.

Let Stm^* denotes the set whose elements are finite, eventually empty, ordered sequences of statements where the empty sequence is denoted by $[\,]$. Moreover let $\circ : \mathrm{Stm}^* \times \mathrm{Stm}^* \to \mathrm{Stm}^*$ be the lists concatenation.
The continuation function $\mathcal{C} : \mathrm{Stm} \to \mathrm{Stm}^*$, is recursively defined by the following clauses:

$$
\begin{array}{rcl}
\mathcal{C}(\epsilon) & \triangleq & [\,\epsilon\,] \\
\mathcal{C}(\mathbf{skip}; K) & \triangleq & [\mathbf{skip}; K] \circ \mathcal{C}(K) \\
\mathcal{C}(x := E; K) & \triangleq & [x := E; K] \circ \mathcal{C}(K) \\
\mathcal{C}((\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K) & \triangleq & [(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K] \circ (\mathcal{C}(S_1) \star K) \circ (\mathcal{C}(S_2) \star K) \circ \mathcal{C}(K) \\
\mathcal{C}((\mathbf{while}\ B\ \mathbf{do}\ S)K) & \triangleq & [(\mathbf{while}\ B\ \mathbf{do}\ S)K\,] \circ (\mathcal{C}(S) \star ((\mathbf{while}\ B\ \mathbf{do}\ S)K)) \circ \mathcal{C}(K)
\end{array}
$$

where the binary operation $\star : \mathrm{Stm}^* \times \mathrm{Stm} \to \mathrm{Stm}^*$ is defined by

$$
\begin{array}{ll}
[\,] \star K = [\,] & \\
[\,\epsilon\,] \star K = [\,] & \\
[\,A_1, ..., A_n\,] \star K = [\,A_1; K\,, ...,\ A_n; K\,] & \forall A_1, ..., A_n, K \in \mathrm{Stm}\,.
\end{array}
$$

Pay attention that whenever there is a branching whose boolean guard is $B \in \mathrm{Bexp}$, the function $\mathcal{C}$ analyzes the true-branch first.
Thus, $\mathcal{C}(S)$ contains all the possible continuations for the program S and the order of appearance provides a total order relation on $\mathcal{C}(S)$.

Let us point out that $\mathcal{C}(S)$ also includes unreachable continuations, and that we can have continuations with multiplicity greater than 1, for example

$\mathcal{C}((\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;) \textbf{skip};) =$
$[\,(\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;) \textbf{skip};\,] \circ\ (\,(\,[\,x := 1; y := 1;\,] \circ\ [\,y := 1;\,] \circ\ [\,\epsilon\,]) \star (\textbf{skip};))\circ$
$(\,(\,[\,y := 1;\,] \circ\ [\epsilon]) \star (\textbf{skip};))\ \circ (\,[\,\textbf{skip};\,] \circ\ [\,\epsilon\,]) =$
$[\,(\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;) \textbf{skip};] \circ\ [\,x := 1; y := 1; \textbf{skip};\ ,\ y := 1; \textbf{skip};\,]\circ$
$[\,y := 1; \textbf{skip};] \circ\ [\,\textbf{skip};\ ,\ \epsilon\,] =$
$[\,(\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;) \textbf{skip};\ ,\ x := 1; y := 1; \textbf{skip};\ ,$
$y := 1; \textbf{skip};\ ,\ y := 1; \textbf{skip};\ ,\ \textbf{skip};\ ,\ \epsilon\,].$

$\mathcal{C}(S)$ provides an alternative representation for the program points of S. In fact, an injective labelling function $\ell : \mathcal{C}(S) \to \mathrm{Label}$ for all the possible continuations of a program S allows us to define a control flow graph (CFG) for S, where the set of nodes is $\{\ell(K) \mid K \in \mathcal{C}(S)\}$, and transitions are labelled with skip, assignments and Boolean tests as follows:

$$\ell(\textbf{skip}; K) \xrightarrow{\textbf{skip}} \ell(K)$$
$$\ell(x := E; K) \xrightarrow{x:=E} \ell(K)$$
$$\ell((\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K) \xrightarrow{B} \ell(S_1 K)$$
$$\ell((\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K) \xrightarrow{\neg B} \ell(S_2 K)$$
$$\ell((\textbf{while } B \textbf{ do } S)K) \xrightarrow{B} \ell(S(\textbf{while } B \textbf{ do } S)K)$$
$$\ell((\textbf{while } B \textbf{ do } S)K) \xrightarrow{\neg B} \ell(K)$$

Moreover, it is possible to provide the set $\{\ell(K) \mid K \in \mathcal{C}(S)\}$ with a total order relation, which follows from the order of appearance in $\mathcal{C}(S)$.

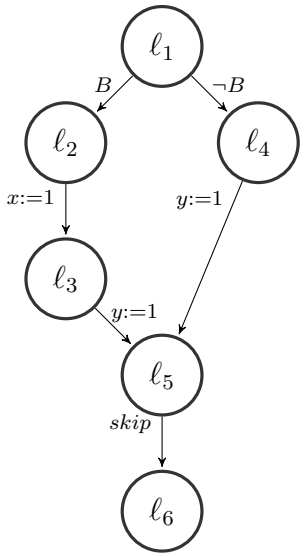
As a first example, consider the program $P \equiv (\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } (y := 1;))\textbf{skip};$. We have previously seen that

$$\mathcal{C}(P) = [\,(\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;)\textbf{skip};\ ,\ x := 1; y := 1; \textbf{skip};\ ,$$
$$y := 1; \textbf{skip};\ ,\ y := 1; \textbf{skip};\ ,\ \textbf{skip};\ ,\ \epsilon\,].$$

The set of Labels is given by $\{\ell_1, \ell_2, \ell_3, \ell_4, \ell_5, \ell_6\}$ where

$$\ell_1 = \ell((\textbf{if } B \textbf{ then } (x := 1; y := 1;) \textbf{ else } y := 1;)\textbf{skip};)$$
$$\ell_2 = \ell(x := 1; y := 1; \textbf{skip};)$$
$$\ell_3 = \ell(y := 1; \textbf{skip};)$$
$$\ell_4 = \ell(y := 1; \textbf{skip};)$$
$$\ell_5 = \ell(\textbf{skip};)$$
$$\ell_6 = \ell(\epsilon)$$

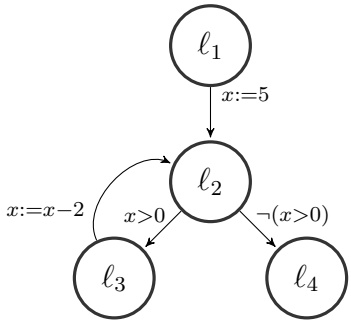
and the resulting CFG is



As a second example, consider the program $Q \equiv\ x := 5;$ **while** $x > 0$ **do** $x := x - 2;$.

$$
\begin{array}{l}
\mathcal{C}(Q) = [x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ ([\textbf{ while } x > 0 \textbf{ do } x := x - 2;] \circ \\
((\,[x := x - 2;] \circ [\,\epsilon\,]\,) \star (\textbf{while } x > 0 \textbf{ do } x := x - 2;)\,) \circ [\,\epsilon\,]\,) = \\
[x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ \\
[\textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ [\,x := 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ [\,\epsilon\,] = \\
[\,x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\ , \\
x := x - 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \epsilon\,].
\end{array}
$$

It is represented by the following CFG:



where:

$$
\begin{array}{ll}
\ell_1 \triangleq \ell(x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;) & \ell_2 \triangleq \ell(\textbf{while } x > 0 \textbf{ do } x := x - 2;) \\
\ell_3 \triangleq \ell(x := x - 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;) & \ell_4 \triangleq \ell(\epsilon)
\end{array}
$$

3.0.1 Remarks

Let us point out that if $K \in \mathcal{C}(S)$ then $\ell(S) \leq \ell(K)$, while it is generally not true that $\mathcal{C}(K) \subseteq \mathcal{C}(S)$. For example, consider the program Q above and $K \equiv x := x - 2;$ **while** $x > 0$ **do** $x := x - 2;$.
Let us compute $\mathcal{C}(K)$.

$$\begin{aligned}&\mathcal{C}(K) = [x := x - 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ [\textbf{while } x > 0 \textbf{ do } x := x - 2;] \circ \\ &([x := x - 2;\ ,\ \epsilon] \star (\textbf{while } x > 0 \textbf{ do } x := x - 2;\ )) \circ [\epsilon] = \\ &[x := x - 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;\ , \textbf{while } x > 0 \textbf{ do } x := x - 2;\ , x := x - 2; \\ &\textbf{while } x > 0 \textbf{ do } x := x - 2;\ , \epsilon].\end{aligned}$$

Notice that $K \in \mathcal{C}(Q)$ but $\mathcal{C}(K) \not\subseteq \mathcal{C}(Q)$, since Stm^* is the set whose elements are finite ordered sequences of statements, and the inclusion '$\subseteq$' has to consider the order.
This fact is actually counterintuitive, but the requirement to operate in Stm^* is due to the fact that it may present the case in which a continuation has molteplicity greater than one (as in the program P of the example above).
Although, this case does no longer exist if the else-branch of the conditional construct is removed, as in [3, Section 10].

Reasoning in the same way, let us consider the set of commands $\widetilde{\mathrm{Cmd}}$ whose syntax is obtained from Cmd replacing the command **if** B **then** S_1 **else** $S_2 \in \mathrm{Cmd}$ with

$$\textbf{if } B \textbf{ then } S \in \widetilde{\mathrm{Cmd}}$$

i.e., removing the symmetric conditional branching. The set $\widetilde{\mathrm{Stm}}$ of possibly empty sequences of elements of $\widetilde{\mathrm{Cmd}}$ gives rise to a Turing complete language and it is easily possible to define a translating function $\phi : \mathrm{Stm} \to \widetilde{\mathrm{Stm}}$. The command **if** B **then** S_1 **else** S_2 is mapped to the construct:

$$\begin{aligned}&B_{aux} = \textbf{tt};\\ &\textbf{if } B \textbf{ then}\{\\ &\qquad S_1\\ &\qquad B_{aux} = \textbf{ff};\}\\ &\textbf{if } B_{aux} \textbf{ then } S_2\end{aligned}$$

where the new variable B_{aux} is necessary to avoid the possible side-effect issue of S_1 on B.

We want to highlight that, in such a setting, it is possible to define the continuation function $\widetilde{\mathcal{C}} : \widetilde{\mathrm{Stm}} \to \wp(\widetilde{\mathrm{Stm}})$, without dealing with the set of ordered lists $\widetilde{\mathrm{Stm}}^*$, recursively defined by the

clauses:

$$\begin{aligned}
\widetilde{\mathcal{C}}(\epsilon) &\triangleq \{\epsilon\} \\
\widetilde{\mathcal{C}}(\textbf{skip}; K) &\triangleq \{\textbf{skip}; K\} \cup \widetilde{\mathcal{C}}(K) \\
\widetilde{\mathcal{C}}(x := E; K) &\triangleq \{x := E; K\} \cup \widetilde{\mathcal{C}}(K) \\
\widetilde{\mathcal{C}}((\textbf{if } B \textbf{ then } S)K) &\triangleq \{(\textbf{if } B \textbf{ then } S)K\} \cup \widetilde{\mathcal{C}}(S\ K) \cup \widetilde{\mathcal{C}}(K) \\
\widetilde{\mathcal{C}}((\textbf{while } B \textbf{ do } S)K) &\triangleq \{(\textbf{while } B \textbf{ do } S)K\} \cup \widetilde{\mathcal{C}}(S(\textbf{while } B \textbf{ do } S)K) \cup \widetilde{\mathcal{C}}(K).
\end{aligned}$$

Observe that if $S \in \widetilde{\mathrm{Stm}}$ and $K \in \widetilde{\mathcal{C}}(S)$ then $\widetilde{\mathcal{C}}(K) \subseteq \widetilde{\mathcal{C}}(S)$ where the relation '$\subseteq$' is the standard set inclusion.
As an example, consider the program $Q \equiv\ x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;$. Then

$$\begin{aligned}
&\widetilde{\mathcal{C}}(Q) = \{x := 5;\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\} \cup \{\textbf{while } x > 0 \textbf{ do } x := x - 2;\} \cup \\
&\{x := x - 2;\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\} \cup \widetilde{\mathcal{C}}(\textbf{while } x > 0 \textbf{ do } x := x - 2;) \cup \{\epsilon\} = \\
&\{x := 5; \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\ , \\
&x := x - 2; \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \epsilon\}
\end{aligned}$$

where the last equality is derived since $\widetilde{\mathcal{C}}(\textbf{while } x > 0 \textbf{ do } x := x - 2;)$ is already contained in $\{\textbf{while } x > 0 \textbf{ do } x := x - 2;\} \cup \{x := x - 2;\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\} \cup \{\epsilon\}$.

Now take $K \equiv x := x - 2;\ \textbf{while } x > 0 \textbf{ do } x := x - 2;$. Then

$$\widetilde{\mathcal{C}}(K) = \{x := x - 2;\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \textbf{while } x > 0 \textbf{ do } x := x - 2;\ ,\ \epsilon\}$$

and it holds $\widetilde{\mathcal{C}}(K) \subseteq \widetilde{\mathcal{C}}(S)$.

3.1 Small Step Operational Semantics

3.1.1 Expressions

Variables are assumed to have integer type and, therefore, they store integer values ranging in $\mathbb{Z}$. A store $\rho \in \mathrm{Store} \triangleq \mathrm{Vars} \hookrightarrow \mathbb{Z}$ is a partial map from variables to integer values, where $|\rho| \triangleq |\operatorname{dom}(\rho)|$ denotes its size of definition.

The semantics of arithmetic and Boolean expressions is defined by the following denotational partial semantic functions:

$$\mathbf{E}[\![E]\!] : \mathrm{Store} \hookrightarrow \mathbb{Z} \qquad \mathbf{B}[\![B]\!] : \mathrm{Store} \hookrightarrow \{\textit{false}, \textit{true}\}$$

where undefinedness models an evaluation error, e.g. we have that $\mathbf{E}[\![x/y]\!]\{x/1, y/0\} = \mathrm{undef}$ and $\mathbf{B}[\![x/(y-1) < x]\!]\{x/1, y/1\} = \mathrm{undef}$.
We also use the following notation: $[\![B]\!] \triangleq \{\rho \in \mathrm{Store} \mid \mathbf{B}[\![B]\!]\rho = \textit{true}\}$.

We denote by $[\![x := E]\!]$ and $[\![B]\!]$ the so-called transfer functions for assignments and Boolean tests, which are defined in their collecting versions on sets of stores as usual:

$$[\![x := E]\!], [\![B]\!] : \wp(\mathrm{Store}) \to \wp(\mathrm{Store})$$
$$[\![x := E]\!]X \triangleq \{\rho[x \mapsto v] \mid \rho \in X, v = \mathbf{E}[\![E]\!]\rho\}$$
$$[\![B]\!]X \triangleq \{\rho \in X \mid \mathbf{B}[\![B]\!]\rho = \mathit{true}\} = [\![B]\!] \cap X$$

Hence, $[\![B]\!]$ is defined as the selector of stores making the expression B true.

3.1.2 Program Semantics

A program state for a continuation-based operational semantics is a pair consisting of the program left to evaluate and the current store, namely $\mathrm{State} \triangleq \mathrm{Stm} \times \mathrm{Store}$. The small-step transition relation $\to \subseteq \mathrm{State} \times \mathrm{State}$ is given in standard continuation-style by the following rules, where $K \in \mathrm{Stm}$:

$$\frac{}{\langle \textbf{skip}; K, \rho\rangle \to \langle K, \rho\rangle} \qquad \frac{\mathbf{E}[\![E]\!]\rho \in \mathrm{Store}}{\langle x := E; K, \rho\rangle \to \langle K, \rho[x \mapsto \mathbf{E}[\![E]\!]\rho]\rangle}$$

$$\frac{\mathbf{B}[\![B]\!]\rho = \mathit{true}}{\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \rho\rangle \to \langle S_1 K, \rho\rangle} \qquad \frac{\mathbf{B}[\![B]\!]\rho = \mathit{false}}{\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \rho\rangle \to \langle S_2 K, \rho\rangle}$$

$$\frac{\mathbf{B}[\![B]\!]\rho = \mathit{false}}{\langle (\textbf{while } B \textbf{ do } S)K, \rho\rangle \to \langle K, \rho\rangle} \qquad \frac{\mathbf{B}[\![B]\!]\rho = \mathit{true}}{\langle (\textbf{while } B \textbf{ do } S)K, \rho\rangle \to \langle S(\textbf{while } B \textbf{ do } S)K, \rho\rangle}$$

Of course, the operational semantics $\to$ is deterministic.

3.2 Traces and Invariants

Let $\mathrm{State}^\infty \triangleq \mathrm{State}^+ \cup \mathrm{State}^\omega$ be the set of nonempty infinitary (i.e., finite and infinite) sequences of states.

A partial trace is any nonempty possibly infinite sequence (indexed by $\mathbb{N}$) of program states which are related by the transition relation $\to$. Hence, the set Trace^∞ of (finite and infinite, partial and maximal) traces is defined as follows:

$$\mathrm{Trace}^\infty \triangleq \{\tau \in \mathrm{State}^\infty \mid \forall i \in [1, |\tau|).\, \tau_{i-1} \to \tau_i\}.$$

The trace semantics of a program S is in turn defined as follows:

$$\mathrm{Trace}[S] \triangleq \{\tau \in \mathrm{Trace}^\infty \mid \exists \rho \in \mathrm{Store}\,.\, \tau_0 = \langle S, \rho\rangle\}$$

and the traces for S with initial store $\rho \in \mathrm{Store}$ are given by:

$$\mathrm{Trace}[S, \rho] \triangleq \{\tau \in \mathrm{Trace}[S] \mid \tau_0 = \langle S, \rho\rangle\}$$

while if $In \in \wp(\mathrm{Store})$ then

$$\mathrm{Trace}[S, In] \triangleq \bigcup_{\rho \in In} \mathrm{Trace}[S, \rho]$$

Given a trace $\tau \in \mathrm{Trace}[S]$, we define the function $\mathrm{Inv}[\tau] : \mathcal{C}(S) \to \wp(\mathrm{Store})$, which, for any continuation $K \in \mathcal{C}(S)$, returns the strongest invariant property for K along the trace τ:

$$\mathrm{Inv}[\tau](K) \triangleq \{\rho \in \mathrm{Store} \mid \exists i \in \mathbb{N}.\ \tau_i = \langle K, \rho \rangle\}.$$

In particular, if K does not occur in τ then $\mathrm{Inv}[\tau](K) = \varnothing$. Also, the function
$\mathrm{Inv}[S, In] : \mathcal{C}(S) \to \wp(\mathrm{Store})$, for any continuation $K \in \mathcal{C}(S)$, returns the strongest invariant properties for K along any trace in $\mathrm{Trace}[S, In]$, namely:

$$\mathrm{Inv}[S, In](K) \triangleq \bigcup_{\tau \in \mathrm{Trace}[S, In]} \mathrm{Inv}[\tau](K).$$

Hence, a continuation $K \in \mathcal{C}(S)$ is unreachable from In if and only if $\mathrm{Inv}[S, In](K) = \varnothing$.

The execution of a program S from an initial store ρ correctly terminates when $\langle S, \rho \rangle \to^* \langle \epsilon, \rho' \rangle$, and, in this case, the store ρ' will be the result of the evaluation. Pairs $\langle \epsilon, \rho \rangle \in \mathrm{State}$ are therefore called final states (notice that $\langle \epsilon, \rho \rangle \not\to$).
On the other hand, $\langle S, \rho \rangle \in \mathrm{State}$ is called an error state when $S \not\equiv \epsilon$ and $\langle S, \rho \rangle \not\to$. For instance, if $\mathbf{E}[\![E]\!]\rho$ is not defined then $\langle x := E; K, \rho \rangle$ is an error state. Analogously, if $\mathbf{B}[\![B]\!]\rho$ is not defined then $\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \rho \rangle$ and $\langle (\textbf{while } B \textbf{ do } S)K, \rho \rangle$ are error states.

The evaluation of S from ρ terminates with an error when $\langle S, \rho \rangle \to^* \langle S', \rho' \rangle$ and $\langle S', \rho' \rangle$ is an error state. On the other hand, the evaluation of S from ρ does not terminate when $\mathrm{Trace}[S, \rho]$ contains an infinite trace. Let us observe that $\langle \textbf{while tt do } \epsilon, \rho \rangle \to \langle \textbf{while tt do } \epsilon, \rho \rangle$, thus leading to nontermination.

3.3 Safety Verification Problem and Hoare Triples

The reachable states collecting semantics of a program $S \in \mathrm{Stm}$ is a function

$$\mathrm{Post}^*[S] : \wp(\mathrm{Store}) \to \wp(\mathrm{State}),$$

which provides the set of states $\mathrm{Post}^*[S]\mathcal{P}$ which are reachable by S from a given set $\mathcal{P} \in \wp(\mathrm{Store})$ of initial stores, i.e.,

$$\mathrm{Post}^*[S]\mathcal{P} \triangleq \{\langle S', \rho' \rangle \in \mathrm{State} \mid \exists \rho \in \mathcal{P}.\ \langle S, \rho \rangle \to^* \langle S', \rho' \rangle\}$$

The safety verification problem for $S \in \mathrm{Stm}$ and $In \in \wp(\mathrm{Store})$ consists in effectively computing a machine representable program invariant $I : \mathcal{C}(S) \to \wp(\mathrm{State})$ such that, for any $K \in \mathcal{C}(S)$, $\mathrm{Inv}[S, In](K) \subseteq I(K)$ holds, or, equivalently, $\langle K, \rho \rangle \in \mathrm{Post}^*[S]In \Rightarrow \rho \in I(K)$.

If $S \in \mathrm{Stm}$ and $\mathcal{P} \in \wp(\mathrm{Store})$, the strongest postcondition holding after executing S in a state belonging to $\mathcal{P}$ is

$$\mathrm{sp}(S, \mathcal{P}) \triangleq \{\rho' \in \mathrm{Store} \mid \langle \epsilon, \rho' \rangle \in \mathrm{Post}^*[S]\mathcal{P}\}$$

Also, if $\mathcal{Q} \in \wp(\mathrm{Store})$ then $\{\mathcal{P}\}S\{\mathcal{Q}\}$ is a valid Hoare triple if

$$\mathrm{sp}(S, \mathcal{P}) \subseteq \mathcal{Q}$$

4 Small Step Collecting Interpreter

The small step parallel collecting interpreter relies on an evaluation stack Stack whose records r are defined as follows, where $\mathcal{P} \in \wp(\mathrm{Store})$ is any store property:

$$\mathrm{Stack} \ni r ::= [S]_{\mathbf{t}} \mid [S]_{\mathbf{e}} \mid [(\textbf{while } B \textbf{ do } S)K, \mathcal{P}]_{\mathbf{w}}$$

The informal meaning of Stack goes as follows:

– if $[K]_{\mathbf{t}}$ or $[K]_{\mathbf{e}}$ happens to be at the top of the stack then the interpreter is currently evaluating in parallel the 'then' and 'else' branches of some **if** B **then** S_1 **else** S_2 command, whose continuation is K. Once the parallel evaluation of both branches successful terminates with $\mathcal{P}_{\textbf{then}}$ and $\mathcal{P}_{\textbf{else}}$, the interpreter goes on with the evaluation of K from the store property $\mathcal{P}_{\textbf{then}} \cup \mathcal{P}_{\textbf{else}}$.

– if $[(\textbf{while } B \textbf{ do } S)K, \mathcal{P}]_{\mathbf{w}}$ is at the top of the stack then the interpreter is currently evaluating the while-command **while** B **do** S whose continuation is K and whose current loop invariant property is $\mathcal{P}$.

If $\Sigma \in \mathrm{Stack}^*$ is an evaluation stack then $\mathrm{Cont}(\Sigma) \in \mathrm{Stm}$ denotes the corresponding whole continuation:

$$\mathrm{Cont}(\Sigma) \triangleq \begin{cases} \epsilon & \text{if } \Sigma \equiv \epsilon \\ K\,\mathrm{Cont}(\Sigma') & \text{if } \Sigma \equiv [K]_{\mathbf{t}} \cdot \Sigma' \text{ or } \Sigma \equiv [K]_{\mathbf{e}} \cdot \Sigma' \\ (\textbf{while } B \textbf{ do } S)K\,\mathrm{Cont}(\Sigma') & \text{if } \Sigma \equiv [(\textbf{while } B \textbf{ do } S)K, \mathcal{P}]_{\mathbf{w}} \cdot \Sigma' \end{cases}$$

A sequential state in $\mathrm{SState}_{\to_{\mathrm{pc}}}$ of the small step parallel collecting interpreter is therefore defined as a triple whose first component is an evaluation stack in Stack^*:

$$\langle \Sigma, S, \mathcal{P} \rangle \in \mathrm{SState}_{\to_{\mathrm{pc}}} \triangleq \mathrm{Stack}^* \times \mathrm{Stm} \times \wp(\mathrm{Store})$$

Moreover, due to the evaluation of if-then-else commands, the interpreter may be in a parallel state $C_1 \parallel C_2$, so that the possibly parallel states C of the parallel interpreter are defined as follows:

$$\mathrm{State}_{\to_{\mathrm{pc}}} \ni C ::= \langle \Sigma, S, \mathcal{P} \rangle \;\mid\; C_1 \parallel C_2$$

The collecting transition relation $\to_{\mathrm{pc}}$ is nondeterministic due to the parallel evaluation of the two branches of an if-then-else, meaning that the following standard rules for parallelism are used:

$$\frac{C_1 \to_{\mathrm{pc}} C_1'}{C_1 \parallel C_2 \to_{\mathrm{pc}} C_1' \parallel C_2} \qquad \frac{C_2 \to_{\mathrm{pc}} C_2'}{C_1 \parallel C_2 \to_{\mathrm{pc}} C_1 \parallel C_2'}$$

The transition relation $\to_{\mathrm{pc}} \subseteq \mathrm{State}_{\to_{\mathrm{pc}}} \times \mathrm{State}_{\to_{\mathrm{pc}}}$ of the small step parallel collecting interpreter is defined by the rules in Figure 1 . Obviously, the nondeterministic choices in a parallel evaluation of $C_1 \parallel C_2$ does not affect the termination of the collecting interpreter and, in case of termination, the output values. However, in case of nontermination, the nondeterministic choices between the two branches of a if-then-else command may also be unfair.

It turns out that this small step collecting interpreter, when it terminates, outputs the strongest postcondition.

Theorem 4.1. *For any program* $S \in \mathrm{Stm}$*:*

(1) $\langle S, \rho\rangle \to^* \langle \epsilon, \rho'\rangle$ *iff* $\langle \epsilon, S, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \{\rho'\}\rangle$.

(2) $\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle \Rightarrow \mathrm{sp}(S, \mathcal{P}) = \mathcal{Q}$.

Of course, the converse of Theorem 4.1 (2) does not hold, meaning that the collecting interpreter is not complete for strongest postconditions. Obviously, this is due to nontermination. However, observe that the simplest nonterminating program $S \equiv$ **while** tt **do skip**; does not yield a counterexample, since for any $\mathcal{P}$ we have that $\mathrm{sp}(S, \mathcal{P}) = \varnothing$ and

$$\langle \epsilon, S, \mathcal{P}\rangle \to_{\mathrm{pc}} \langle [S, \mathcal{P}]_{\mathbf{w}}, \mathbf{skip}; , \mathcal{P}\rangle \to_{\mathrm{pc}} \langle [S, \mathcal{P}]_{\mathbf{w}}, \epsilon, \mathcal{P}\rangle \to_{\mathrm{pc}} \langle \epsilon, \epsilon, [\![\neg \,\mathrm{tt}\,]\!]\mathcal{P} = \varnothing\rangle$$

A counterexample to the converse of Theorem 4.1 (2) is the following.

Example 4.2. Consider the following program $S \equiv$ **while** $x \neq 0$ **do** $x := x - 2$; and $\mathcal{P} = \{x/2, x/3\} \in \wp(\mathrm{Store})$. We have that $\mathrm{sp}(S, \mathcal{P}) = \{x/0\}$ while the evaluation of $\langle \epsilon, S, \mathcal{P}\rangle$ does not terminate, since the collecting interpreter generates the following infinite trace:

$$\begin{aligned}\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle [S, \{x/2, x/3\}]_{\mathbf{w}}, \epsilon, \{x/0, x/1\}\rangle \to^*_{\mathrm{pc}}& \\ \langle [S, \{x/0, x/1, x/2, x/3\}]_{\mathbf{w}}, \epsilon, \{x/\text{-}1, x/0, x/1\}\rangle \to^*_{\mathrm{pc}}& \\ \langle [S, \{x/\text{-}1, x/0, x/1, x/2, x/3\}]_{\mathbf{w}}, \epsilon, \{x/\text{-}3, x/\text{-}1, x/0, x/1\}\rangle \to^*_{\mathrm{pc}} \cdots&\end{aligned}$$

so that $\langle \epsilon, S, \mathcal{P}\rangle \not\to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathrm{sp}(S, \mathcal{P})\rangle$.

□

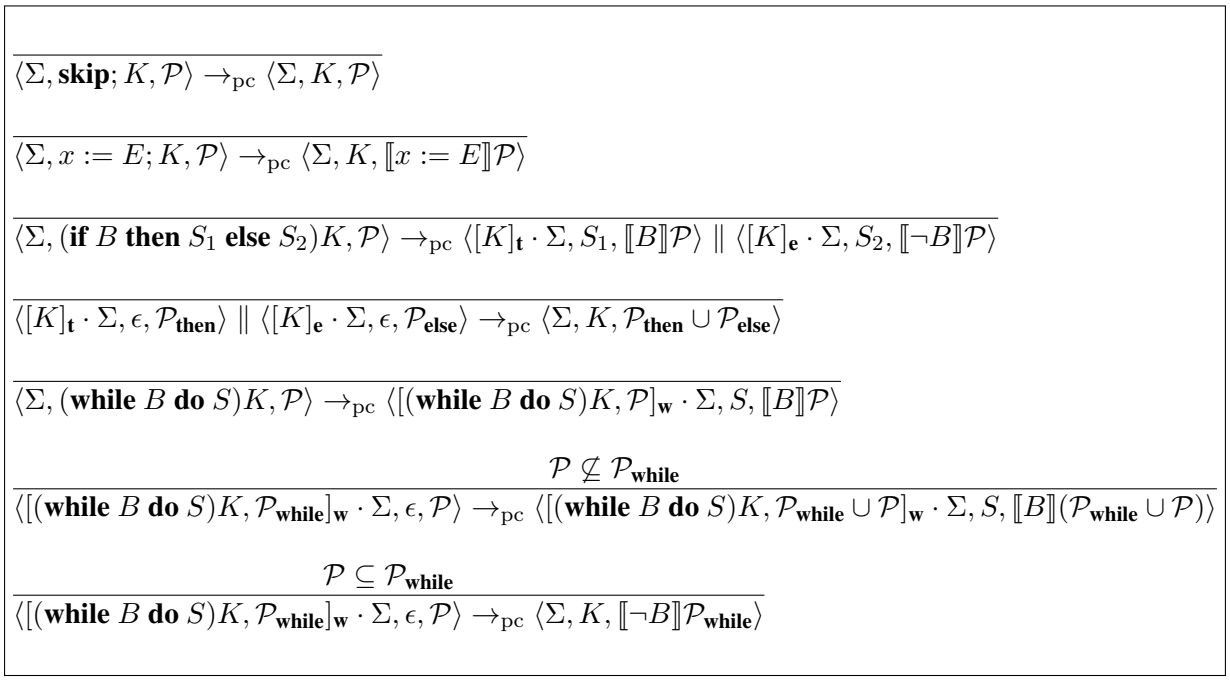


Figure 1: The (forward and parallel) small step collecting interpreter.

4.1 The sequential collecting interpreter

The interleaved parallel evaluation of branches of conditional commands can be easily replaced by a sequential evaluation strategy, where, e.g., the then-branch is evaluated first. Here, the evaluation stack is defined by:

$$\mathrm{Stack}_{\mathrm{sc}} \ni r ::= [(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K, \mathcal{P}]_{\mathbf{t}} \mid [(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K, \mathcal{P}]_{\mathbf{e}} \mid [(\mathbf{while}\ B\ \mathbf{do}\ S)K, \mathcal{P}]_{\mathbf{w}}$$

and the meaning of the records $[\cdot]_{\mathbf{t}}$ and $[\cdot]_{\mathbf{e}}$ is the following:

- when $[(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K, \mathcal{P}]_{\mathbf{t}}$ happens to be at the top of the interpreter stack, the evaluation is currently inside the then-branch of the $\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2$ command, whose overall continuation is K, while the component $\mathcal{P}$ records the initial store property for the successive evaluation of the body S_2 of the else-branch.
- when $[(\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2)K, \mathcal{P}]_{\mathbf{e}}$ happens to be at the top of the stack, we have that the evaluation is currently inside the else-branch of the $\mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2$ command, whose overall continuation is K, while the component $\mathcal{P}$ records the final store property of the previous evaluation of the body S_1 of the then-branch.

Also, states of this sequential interpreter are simply given by:

$$\langle \Sigma, S, \mathcal{P}\rangle \in \mathrm{State}_{\to_{\mathrm{sc}}} \triangleq \mathrm{Stack}_{\mathrm{sc}}^* \times \mathrm{Stm} \times \wp(\mathrm{Store})$$

$$\frac{}{\langle \Sigma, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{P}\rangle \to_{\mathrm{sc}} \langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, [\![\neg B]\!]\mathcal{P}]_{\mathbf{t}} \cdot \Sigma, S_1, [\![B]\!]\mathcal{P}\rangle}$$

$$\frac{}{\langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{P}_{\textbf{else}}]_{\mathbf{t}} \cdot \Sigma, \epsilon, \mathcal{P}\rangle \to_{\mathrm{sc}} \langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{P}]_{\mathbf{e}} \cdot \Sigma, S_2, \mathcal{P}_{\textbf{else}}\rangle}$$

$$\frac{}{\langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{P}_{\textbf{then}}]_{\mathbf{e}} \cdot \Sigma, \epsilon, \mathcal{P}\rangle \to_{\mathrm{sc}} \langle \Sigma, K, \mathcal{P}_{\textbf{then}} \cup \mathcal{P}\rangle}$$

Figure 2: The sequential small step collecting interpreter.

The transition relation $\to_{\mathrm{sc}} \subseteq \mathrm{State}_{\to_{\mathrm{sc}}} \times \mathrm{State}_{\to_{\mathrm{sc}}}$ is defined by the rules in Figure 2 for if-then-else commands which replace the corresponding rules in Figure 1, while the remaining rules are unchanged.

The parallel and sequential collecting interpreters are equivalent for terminating evaluations.

To show this, we need some preliminary notations. If $C \in \mathrm{State}_{\to_{\mathrm{pc}}}$, we write $\langle \Sigma, S, \mathcal{P}\rangle \in C$ to mean that $\langle \Sigma, S, \mathcal{P}\rangle$ occurs in the state C. If $\langle \Sigma, S, \mathcal{P}\rangle$ and $\langle \Sigma', S', \mathcal{P}'\rangle$ are sequential states, then we abuse notation by writing $\langle \Sigma, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$ to mean that there exists some $C' \in \mathrm{State}_{\to_{\mathrm{pc}}}$ such that $\langle \Sigma', S', \mathcal{P}'\rangle \in C'$ and $\langle \Sigma, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} C'$.

Theorem 4.3.

(1) $\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle$ *if and only if* $\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle$

(2) If the (parallel or sequential) evaluation of $\langle \epsilon, S, \mathcal{P}\rangle$ *terminates then* $\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{sc}} \langle \Sigma_{\mathrm{s}}, S', \mathcal{P}'\rangle$ *iff* $\langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma_{\mathrm{p}}, S', \mathcal{P}'\rangle$.

As a consequence, we obtain that

$$\langle S, \rho\rangle \to^* \langle \epsilon, \rho'\rangle \;\Leftrightarrow\; \langle \epsilon, S, \{\rho\}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \{\rho'\}\rangle \qquad \langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle \;\Rightarrow\; \mathrm{sp}(S, \mathcal{P}) = \mathcal{Q}$$

Instead, in case of nontermination sequential and parallel collecting interpreters obviously may behave differently, as in following example.

Example 4.4. Consider the program

$$\begin{array}{l} x := 5; \\ \textbf{while } x > 1 \textbf{ do} \\ \quad \textbf{if } x \bmod 2 = 0 \textbf{ then} \\ \quad\quad \textbf{while tt do } x := x + 1; \\ \quad \textbf{else } x := x - 1; \end{array}$$

The parallel interpreter acts as follows:

$\langle \epsilon, x := 5; K, \wp(\mathrm{Store})\rangle \to_{\mathrm{pc}} \langle \epsilon, \textbf{while } x > 1 \textbf{ do } T, \{x/5\}\rangle \to_{\mathrm{pc}}$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5\}]_w, \textbf{if } x \textbf{ mod } 2 = 0 \textbf{ then } S_1 \textbf{ else } S_2, \{x/5\}\rangle \to_{\mathrm{pc}}$
$\langle [W, \{x/5\}]_w, \textbf{while tt do } x := x + 1; , \emptyset\rangle \parallel \langle [W, \{x/5\}]_w, x := x - 1; , \{x/5\}\rangle \to^*_{\mathrm{pc}}$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, \textbf{if } x \textbf{ mod } 2 = 0 \textbf{ then } S_1 \textbf{ else } S_2, \{x/5,\ x/4\}\rangle \to_{\mathrm{pc}}$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, \textbf{while tt do } x := x + 1; , \{x/4\}\rangle \parallel$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, x := x - 1; , \{x/5\}\rangle \to_{\mathrm{pc}}$
$\langle [\textbf{while tt do } x := x + 1; , \{x/4\}]_w \cdot [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, x := x + 1; , \{x/4\}\rangle \parallel$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, \epsilon, \{x/4\}\rangle \to^*_{\mathrm{pc}} ...$

and the first branch never ends because the loop invariant does not get to a fixed point:

$$\{x/5,\ x/6,\ , ..., x/n + 1\} \not\subseteq \{x/4, x/5, ..., x/n\}.$$

Although, we can write that

$$\langle \epsilon, S, \{x/5\}\rangle \to^*_{\mathrm{pc}} \langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, \epsilon, \{x/4\}\rangle.$$

This is not true for the sequential interpreter: at the second iteration of the external loop, it gets stuck in the true-branch of the if-then-else and never reaches the else-branch, meaning

$\langle \epsilon, x := 5; K, \wp(\mathrm{Store})\rangle \to^*_{\mathrm{sc}}$
$\langle [\textbf{while } x > 1 \textbf{ do } T, \{x/5,\ x/4\}]_w, \textbf{if } x \textbf{ mod } 2 = 0 \textbf{ then } S_1 \textbf{ else } S_2, \{x/5,\ x/4\}\rangle \to_{\mathrm{sc}}$
$\langle [\textbf{if } x \textbf{ mod } 2 = 0 \textbf{ then } S_1 \textbf{ else } S_2, \{x/5\}]_t \cdot [W, \{x/5,\ x/4\}]_w, \textbf{while tt do } x := x + 1, \{x/4\}\rangle \to_{\mathrm{sc}}$
$\langle [\textbf{while tt do } x := x + 1, \{x/4\}]_w \cdot [\textbf{if } , \{x/5\}]_t \cdot [W, \{x/5,\ x/4\}]_w, x := x + 1, \{x/4\}\rangle \to_{\mathrm{sc}}$
$\langle [\textbf{while tt do } x := x + 1, \{x/4\}]_w \cdot [\textbf{if } , \{x/5\}]_t \cdot [W, \{x/5,\ x/4\}]_w, \epsilon, \{x/5\}\rangle \to^*_{\mathrm{sc}}$
$\langle [\textbf{while tt do } x := x + 1, \{x/4, x/5\}]_w \cdot [\textbf{if } , \{x/5\}]_t \cdot [W, \{x/5,\ x/4\}]_w, \epsilon, \{x/5, x/6\}\rangle \to^*_{\mathrm{sc}} ...$

4.2 Proof

Before staring with proving the theorems of the previous section, it may be useful to recall two classical lemmas of small step semantics of programs. The proofs are omitted.

Lemma 4.5. *(Decomposition Lemma) If* $\langle S_1 S_2, \rho\rangle \to^{(n)} \langle \epsilon, \rho'\rangle$ *then there exist a* $\rho'' \in \mathrm{Store}$ *and natural numbers* k_1, k_2 *such that*

$$\langle S_1 S_2, \rho\rangle \to^{(k_1)} \langle S_2, \rho''\rangle \quad \textit{and} \quad \langle S_2, \rho''\rangle \to^{(k_2)} \langle \epsilon, \rho'\rangle.$$

Moreover $k_1 + k_2 = n$.

Lemma 4.6. *(Termination lemma) If* $\langle(\textbf{while } B \textbf{ do } T)K, \rho\rangle \to^* \langle K, \rho'\rangle$ *then* $\mathbf{B}[\![B]\!]\rho = \mathit{false}$.

Let us introduce an order relation on Stm: we write $S \subseteq T \;\Leftrightarrow\; S \in \mathcal{C}(T)$. This relation can be extended to $\mathrm{SState}_{\to_{\mathrm{pc}}}$:

$$\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \subseteq \langle \Sigma_2, S_2, \mathcal{P}_2\rangle \text{ if and only if } \begin{cases} S_1\,\mathrm{Cont}(\Sigma_1) \subseteq S_2\,\mathrm{Cont}(\Sigma_2) \\ \mathcal{P}_1 \subseteq \mathcal{P}_2 \end{cases}$$

Moreover we say $\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \equiv \langle \Sigma_2, S_2, \mathcal{P}_2\rangle$ if and only if both $\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \subseteq \langle \Sigma_2, S_2, \mathcal{P}_2\rangle$ and $\langle \Sigma_2, S_2, \mathcal{P}_2\rangle \subseteq \langle \Sigma_1, S_1, \mathcal{P}_1\rangle$ hold.

The following technical lemma contextualizes such a relation.

Lemma 4.7. *Suppose* $\langle \Sigma_1, S_1, \mathcal{P}_1\rangle,\ \langle \Sigma_2, S_2, \mathcal{P}_2\rangle \in \mathrm{SState}_{\to_{\mathrm{pc}}}$ *with* $\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \equiv \langle \Sigma_2, S_2, \mathcal{P}_2\rangle$. *Let* $S' \in \mathrm{Stm}$ *and* $\Sigma' \in \mathrm{Stack}^*$. *If* $S'\,\mathrm{Cont}(\Sigma') \subseteq \mathrm{Cont}(\Sigma_1)$ *and* $S'\,\mathrm{Cont}(\Sigma') \subseteq \mathrm{Cont}(\Sigma_2)$ *then*

$$\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \quad\Longleftrightarrow\quad \langle \Sigma_2, S_2, \mathcal{P}_2\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle.$$

Proof. It is enough to show

$$\langle \Sigma, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle \epsilon, S\,\mathrm{Cont}(\Sigma), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$

with $S'\,\mathrm{Cont}(\Sigma') \subseteq \mathrm{Cont}(\Sigma)$.
In fact, by hypothesis

$$\begin{cases} S_1\,\mathrm{Cont}(\Sigma_1) = S_2\,\mathrm{Cont}(\Sigma_2) \\ \mathcal{P}_1 = \mathcal{P}_2 \end{cases}$$

hence

$$\begin{aligned} &\langle \Sigma_1, S_1, \mathcal{P}_1\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle \epsilon, S_1\,\mathrm{Cont}(\Sigma_1), \mathcal{P}_1\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow \\ &\langle \epsilon, S_2\,\mathrm{Cont}(\Sigma_2), \mathcal{P}_2\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle \Sigma_2, S_2, \mathcal{P}_2\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle. \end{aligned}$$

By structural induction on $\Sigma \in \mathrm{Stack}^*$.

- $\Sigma \equiv \epsilon$; trivial because $\mathrm{Cont}(\epsilon) = \epsilon$.
- $\Sigma \equiv [K] \cdot \Sigma''$; then $\mathrm{Cont}(\Sigma) = K\,\mathrm{Cont}(\Sigma'')$. Observe that by inductive hypothesis it holds

  $$\langle \epsilon, SK\,\mathrm{Cont}(\Sigma''), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle \Sigma'', SK, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle.$$

  Let us argue by structural induction on $S \in \mathrm{Stm}$.
  If $S \equiv \epsilon$ then

  $$\begin{aligned} &\langle \Sigma, \epsilon, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle [K] \cdot \Sigma'', \epsilon, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow \\ &\langle \Sigma'', K, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction}}{\Leftrightarrow} \langle \epsilon, K\,\mathrm{Cont}(\Sigma''), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle. \end{aligned}$$

If $S \equiv cK'$ then consider $c \in \mathrm{Cmd}$.
$c \equiv x := E;$. Then

$$\langle \Sigma, x := E; K', \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow\; \langle [K]\cdot\Sigma'', x := E; K', \mathcal{P}\rangle \to_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [K]\cdot\Sigma'', K', [\![x := E]\!]\mathcal{P}\rangle \to_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction on K'}}{\Leftrightarrow} \langle \Sigma'', K'K, [\![x := E]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$
$$\overset{\text{induction}}{\Leftrightarrow} \langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), [\![x := E]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle.$$

On the other hand

$$\langle \epsilon, x := E; K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), [\![x := E]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$

hence the 'if and only if' holds.

The case $c \equiv$ **skip**; is analoguos.

If $c \equiv$ **if** B **then** S_1 **else** S_2 then observe that

$$\langle \Sigma, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K', \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [K]\cdot\Sigma'', (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K', \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [K']_t\cdot[K]\cdot\Sigma'', S_1, [\![B]\!]\mathcal{P}\rangle \parallel \langle [K']_e\cdot[K]\cdot\Sigma'', S_2, [\![\neg B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{S'\,\mathrm{Cont}(\Sigma')\subseteq\mathrm{Cont}(\Sigma)}{\Leftrightarrow}$$
$$\langle [K]\cdot\Sigma'', K', \mathcal{P}_t \cup \mathcal{P}_e\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction}}{\Leftrightarrow}$$
$$\langle \Sigma'', K'K, \mathcal{P}_t \cup \mathcal{P}_e\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction}}{\Leftrightarrow}$$
$$\langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}_t \cup \mathcal{P}_e\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle.$$

On the other hand

$$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [K']_t\cdot[K]\cdot\Sigma'', S_1, [\![B]\!]\mathcal{P}\rangle \parallel \langle [K']_e\cdot[K]\cdot\Sigma'', S_2, [\![\neg B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S'.\mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}_t \cup \mathcal{P}_e\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$

hence the double implication holds.
If $c \equiv$ **while** B **do** T then observe that

$$\langle [K]\cdot\Sigma'', (\textbf{while } B \textbf{ do } T)K', \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [(\textbf{while } B \textbf{ do } T)K', \mathcal{P}]_w\cdot[K]\cdot\Sigma'', T, [\![B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{S'\,\mathrm{Cont}(\Sigma')\subseteq\mathrm{Cont}(\Sigma)}{\Leftrightarrow}$$
$$\langle [K]\cdot\Sigma'', K', \mathcal{P}_w\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction}}{\Leftrightarrow} \langle \Sigma'', K'K, \mathcal{P}_w\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \overset{\text{induction}}{\Leftrightarrow}$$
$$\langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}_w\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$

and on the other hand

$$\langle \epsilon, (\textbf{while } B \textbf{ do } T)K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \;\Leftrightarrow$$
$$\langle [(\textbf{while } B \textbf{ do } T)K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}]_w, T, [\![B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle \Leftrightarrow$$
$$\langle \epsilon, K'K\,\mathrm{Cont}(\Sigma''), \mathcal{P}_w\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S', \mathcal{P}'\rangle$$

and the 'if and only if' holds.

- The case $\Sigma \equiv [(\textbf{while } B \textbf{ do } S)K, \mathcal{P}]_w \cdot \Sigma'$ follows in the same way.

□

The proof of Theorem 4.1 exploits the following preliminary result:

Lemma 4.8. *For any program* $S \in \mathrm{Stm}$*:*

(1) $\langle cK, \rho\rangle \to^* \langle K, \rho'\rangle$ *if and only if* $\forall\, \Sigma \in \mathrm{Stack}^*\ .\ \langle \Sigma, cK, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, K, \{\rho'\}\rangle$.

(2) If $\langle S, \rho\rangle \to^* \langle S', \rho'\rangle$ *then there exists* $\mathcal{Q} \in \wp(\mathrm{Store})$, $\Sigma \in \mathrm{Stack}^*$, $S'' \in \mathrm{Stm}$ *such that* $\rho' \in \mathcal{Q}$, $S''\,\mathrm{Cont}(\Sigma) \equiv S'$ *and* $\langle \epsilon, S, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S'', \mathcal{Q}\rangle$.

(3) If $\langle \epsilon, S, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle$ *then for any* $\rho' \in \mathcal{Q}$ *,* $\langle S, \rho\rangle \to^* \langle S'\,\mathrm{Cont}(\Sigma), \rho'\rangle$.

Proof. (1). By structural induction on $c \in \mathrm{Cmd}$.

- $c \equiv \textbf{skip}$;. We have $\langle \textbf{skip}; K, \rho\rangle \to \langle K, \rho\rangle$ and $\langle \Sigma, \textbf{skip}; K, \{\rho\}\rangle \to_{\mathrm{pc}} \langle \Sigma, K, \{\rho\}\rangle$.

- $c \equiv x := E;$. Assume $\rho \in \mathrm{dom}(\mathbf{E}[\![E]\!])$.

$$\langle x := E; K, \rho\rangle \to \langle K, \rho[x \mapsto \mathbf{E}[\![E]\!]\rho]\rangle \text{ and}$$
$$\langle \Sigma, x := E; K, \{\rho\}\rangle \to_{\mathrm{pc}} \langle \Sigma, K, [\![x := E]\!]\{\rho\}\rangle \equiv \langle \Sigma, K, \{\rho[x \mapsto \mathbf{E}[\![E]\!]\rho]\}\rangle.$$

- $c \equiv \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2$. Assume $\mathbf{B}[\![B]\!]\rho = \textbf{tt}$.

$$\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \rho\rangle \to \langle S_1K, \rho\rangle \to^* \langle K, \rho'\rangle.$$

Then

$$\langle \Sigma, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \{\rho\}\rangle \to_{\mathrm{pc}}$$
$$\langle [K]_t \cdot \Sigma, S_1, [\![B]\!]\{\rho\}\rangle \parallel \langle [K]_e \cdot \Sigma, S_2, [\![\neg B]\!]\{\rho\}\rangle \equiv \langle \Sigma, S_1K, \{\rho\}\rangle \parallel \langle \Sigma, S_2K, \emptyset\rangle \to^*_{\mathrm{pc}}$$
$$\langle \Sigma, K, \{\rho'\}\rangle$$

by inductive hypothesis.

Conversely, we have

$\langle \Sigma, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \{\rho\}\rangle \to_{\mathrm{pc}} \langle [K]_t{\cdot}\Sigma, S_1, [\![B]\!]\{\rho\}\rangle \parallel \langle [K]_e{\cdot}\Sigma, S_2, \emptyset\rangle \to^*_{\mathrm{pc}} \langle \Sigma, K, \{\rho'\}\rangle.$

Hence

$$\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \rho\rangle \to \langle S_1 K, \rho\rangle \to^* \langle K, \rho'\rangle$$

by inductive hypothesis.

The case $\mathbf{B}[\![B]\!]\rho = \textbf{ff}$ is analoguos.

- $c \equiv \textbf{while } B \textbf{ do } S$. Assume $\langle (\textbf{while } B \textbf{ do } S)K, \rho\rangle \to^* \langle K, \rho'\rangle$. By Termination Lemma 4.6 $\mathbf{B}[\![B]\!]\rho' = \textbf{ff}$. We can moreover assume $\{\rho, \rho^1, ..., \rho^n\}$ are $n+1$ distinct elements of Store such that

$$\mathbf{B}[\![B]\!]\rho^i = \textbf{tt},\ \langle S, \rho^i\rangle \to^* \langle \epsilon, \rho^{i+1}\rangle \text{ for } i = 0, ..., n-1 \text{ and } \langle S, \rho^n\rangle \to^* \langle \epsilon, \rho'\rangle.$$

Hence

$$\begin{aligned}
&\langle \Sigma, (\textbf{while } B \textbf{ do } S)K, \{\rho\}\rangle \to_{\mathrm{pc}} \langle [(\textbf{while } B \textbf{ do } S)K, \{\rho\}]_w \cdot \Sigma, S, [\![B]\!]\{\rho\}\rangle \to^*_{\mathrm{pc}}\\
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho\}]_w \cdot \Sigma, \epsilon, \{\rho^1\}\rangle \to^*_{\mathrm{pc}}\\
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho, \rho^1\}]_w \cdot \Sigma, S, [\![B]\!]\{\rho, \rho^1\}\rangle \to^*_{\mathrm{pc}}\\
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho, \rho^1\}]_w \cdot \Sigma, \epsilon, \{\rho^1, \rho^2\}\rangle \to^*_{\mathrm{pc}} ... \to^*_{\mathrm{pc}}\\
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho, \rho^1, ..., \rho'\}]_w \cdot \Sigma, \epsilon, \{\rho^1, \rho^2, ..., \rho'\}\rangle \to_{\mathrm{pc}}\\
&\langle \Sigma, K, [\![\neg B]\!]\{\rho, \rho^1, ..., \rho'\}\rangle \equiv \langle \Sigma, K, \{\rho'\}\rangle
\end{aligned}$$

where all the transitions

$$\begin{aligned}
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho, \rho^1, ..., \rho^i\}]_w \cdot \Sigma, S, [\![B]\!]\{\rho, ..., \rho^i\}\rangle \to^*_{\mathrm{pc}}\\
&\langle [(\textbf{while } B \textbf{ do } S)K, \{\rho, \rho^1, ..., \rho^i\}]_w \cdot \Sigma, \epsilon, \{\rho^1, ..., \rho^{i+1}\}\rangle
\end{aligned}$$

are infered by structural induction.

Conversely assume

$$\langle \Sigma, (\textbf{while } B \textbf{ do } S)K, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, K, \{\rho'\}\rangle.$$

We can deduce that $\{\rho'\} = [\![\neg B]\!]\mathcal{P}_w$ where $\mathcal{P}_w = \{\rho, \rho^1, ..., \rho^n, \rho'\} \subseteq \mathrm{Store}$, $\mathbf{B}[\![B]\!]\rho^i = \textbf{tt}$ for all $i = 0, 1, ..., n$ and $\langle S, \rho^i\rangle \to^* \langle \epsilon, \rho^{i+1}\rangle$.
Hence

$$\begin{aligned}
&\langle (\textbf{while } B \textbf{ do } S)K, \rho\rangle \to \langle S(\textbf{while } B \textbf{ do } S)K, \rho\rangle \to^* \langle (\textbf{while } B \textbf{ do } S)K, \rho^1\rangle \to^*\\
&\langle (\textbf{while } B \textbf{ do } S)K, \rho'\rangle \to \langle K, \rho'\rangle
\end{aligned}$$

where all the transitions

$$\langle S(\textbf{while } B \textbf{ do } S)K, \rho^i\rangle \to^* \langle (\textbf{while } B \textbf{ do } S)K, \rho^{i+1}\rangle$$

are infered by structural induction.

(2). By structural induction on $S \in \mathrm{Stm}$.

$S \equiv \epsilon$.

We have $\langle \epsilon, \rho \rangle \not\to$ because this is a terminal state, hence the thesis is vacuosly true.

Assume it is true for $S \in \mathrm{Stm}$ and prove for cS,by structural induction on $c \in \mathrm{Cmd}$.

- $c \equiv$ **skip**.
  We have $\langle \textbf{skip}; S, \rho \rangle \to \langle S, \rho \rangle \to^* \langle S', \rho' \rangle$.
  Now
  $$\langle \epsilon, \textbf{skip}; S, \{\rho\} \rangle \to_{\mathrm{pc}} \langle \epsilon, S, \{\rho\} \rangle.$$
  Moreover the inductive hypothesis implies the existence of a $\mathcal{Q} \in \wp(\mathrm{Store})$ such that
  $$\langle \epsilon, S, \{\rho\} \rangle \to^*_{\mathrm{pc}} \langle \Sigma, S'', \mathcal{Q} \rangle.$$
  The conclusion follows composing the transitions.

- $c \equiv x := E$.
  $\langle x := E; S, \rho \rangle \to \langle S, \rho[x \mapsto \mathbf{E}[\![E]\!]\rho] \rangle \to^* \langle S', \rho' \rangle$.
  Thus
  $$\langle \epsilon, x := E; S, \{\rho\} \rangle \to_{\mathrm{pc}} \langle \epsilon, S, \{\rho[x \mapsto \mathbf{E}[\![E]\!]\rho]\} \rangle \to^*_{\mathrm{pc}} \langle \Sigma, S'', \mathcal{Q} \rangle$$
  where the existence of such a $\mathcal{Q} \in \wp(\mathrm{Store})$ and the fact that $S'' \,\mathrm{Cont}(\Sigma) \equiv S'$ are ensured by inductive hypothesis, as in the previous case.

- $c \equiv$ **if** B **then** S_1 **else** S_2.
  Assume $\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \rho \rangle \to^* \langle S', \rho' \rangle$.
  If $S' \in \mathcal{C}(S)$ then
  $$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \{\rho\} \rangle \to^*_{\mathrm{pc}} \langle \epsilon, S, \{\rho''\} \rangle \to^*_{\mathrm{pc}} \langle \Sigma, S'', \mathcal{Q} \rangle$$
  by inductive hypothesis, for a suitable $\rho'' \in \mathrm{Store}$.
  If $S' \in \mathcal{C}(S_1) \star S$ and $\mathbf{B}[\![B]\!]\rho =$ **ff** then the thesis is vacuosly true because $\mathcal{Q} = \emptyset$.
  If $S' \in \mathcal{C}(S_1) \star S$ and $\mathbf{B}[\![B]\!]\rho =$ **tt** then $\langle S_1 S, \rho \rangle \to^* \langle S', \rho' \rangle$ hence, by inductive hypothesis
  $$\langle \epsilon, S_1 S, \{\rho\} \rangle \to^*_{\mathrm{pc}} \langle \Sigma, S'', \mathcal{Q} \rangle \text{ with } S'' \,\mathrm{Cont}(\Sigma) \equiv S'.$$
  Now observe that
  $$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \{\rho\} \rangle \to^*_{\mathrm{pc}} \langle [S], S_1, \{\rho\} \rangle \parallel \langle [S], S_2, \emptyset \rangle \equiv \langle \epsilon, S_1 S, \{\rho\} \rangle$$

- $c \equiv$ **while** B **do** T.
  Assume $\langle (\textbf{while } B \textbf{ do } T)S, \rho \rangle \to^* \langle S', \rho' \rangle$.
  If $S' \in \mathcal{C}(S)$ then conclude by induction as in the previous case.

Let $S' \in \mathcal{C}((\textbf{while } B \textbf{ do } T)S)$. Recall that

$$\mathcal{C}((\textbf{while } B \textbf{ do } T)S) = \ [(\textbf{while } B \textbf{ do } T)S] \circ (\mathcal{C}(T) \star ((\textbf{while } B \textbf{ do } T)S)) \circ \mathcal{C}(S).$$

The only relevant case is $S' \in \mathcal{C}(T) \star ((\textbf{while } B \textbf{ do } T)S)$.
We have $\langle T(\textbf{while } B \textbf{ do } T)S, \rho\rangle \to^* \langle S', \rho'\rangle$.
Following the definition of the function $\mathcal{C}$ it turns out that

$$S' \equiv P(\textbf{while } B \textbf{ do } T)S, \text{ with } P \in \mathcal{C}(T).$$

By Decomposition Lemma 4.5 $\langle T, \rho\rangle \to^* \langle P, \rho'\rangle$ and by inductive hypothesis

$$\langle \epsilon, T, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma', S'', \mathcal{Q}'\rangle \text{ where } \rho' \in \mathcal{Q}', \ S'' \,\mathrm{Cont}(\Sigma') \equiv P.$$

Hence

$\langle \epsilon, (\textbf{while } B \textbf{ do } T)S, \{\rho\}\rangle \to_{\mathrm{pc}}$
$\langle [(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w, T, \{\rho\}\rangle \equiv \langle \epsilon, T(\textbf{while } B \textbf{ do } T)S, \{\rho\}\rangle \to^*_{\mathrm{pc}}$
$\langle \Sigma' \cdot [(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w, S'', \mathcal{Q}'\rangle$ and
$S'' \,\mathrm{Cont}(\Sigma' \cdot [(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w) \equiv P \,\mathrm{Cont}([(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w) \equiv$
$P(\textbf{while } B \textbf{ do } T)S \equiv S'$.

(3). By structural induction on $S \in \mathrm{Stm}$.
If $S \equiv \epsilon$ the thesis is vacuosly true.
Assume it true for S and prove it for cS, by structural induction on $c \in \mathrm{Cmd}$.

- $c \equiv \textbf{skip}$; , $c \equiv x := E$; are easily obtained by induction.
- $c \equiv \textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2$.
  Assume $\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle$.
  If $S' \in \mathcal{C}(S)$ then conclude by induction.

  If $S' \in \mathcal{C}(S_1)$ and $[\![B]\!]\{\rho\} = \emptyset$ then the thesis is vacuosly true.
  If $S' \in \mathcal{C}(S_1)$ and $[\![B]\!]\{\rho\} = \{\rho\}$ then

  $\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \{\rho\}\rangle \to_{\mathrm{pc}}$
  $\langle [S]_t, S_1, [\![B]\!]\{\rho\}\rangle \parallel \langle [S]_e, S_2, [\![\neg B]\!]\{\rho\}\rangle \equiv \langle [S]_t, S_1, \{\rho\}\rangle \parallel \langle [S]_e, S_2, \emptyset\rangle \equiv \langle [S]_t, S_1, \{\rho\}\rangle$ and
  $\langle [S]_t, S_1, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \{\rho'\}\rangle$.

  Hence by structural induction

  $$\langle (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \rho\rangle \to^* \langle S_1, \rho\rangle \to^* \langle S', \rho'\rangle.$$

- $c \equiv$ **while** B **do** T.
  $S' \operatorname{Cont}(\Sigma) \in \mathcal{C}((\textbf{while } B \textbf{ do } T)S)$.
  As before, the only relevant case is $S' \operatorname{Cont}(\Sigma) \in \mathcal{C}(T) \star ((\textbf{while } B \textbf{ do } T)S)$.
  By definition of $\mathcal{C}$ it turns out that $S' \operatorname{Cont}(\Sigma) = P(\textbf{while } B \textbf{ do } T)S$ with $P \in \mathcal{C}(T)$.
  Hence

$$\langle \epsilon, (\textbf{while } B \textbf{ do } T)S, \{\rho\}\rangle \to_{\mathrm{pc}} \langle [(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w, T, [\![B]\!]\{\rho\}\rangle \to^*_{\mathrm{pc}}$$
$$\langle [(\textbf{while } B \textbf{ do } T)S, \{\rho\}]_w, P, \{\rho'\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle.$$

  Now, by inductive hypothesis

$$\langle T(\textbf{while } B \textbf{ do } T)S, \rho\rangle \to^* \langle P(\textbf{while } B \textbf{ do } T)S, \rho'\rangle \equiv \langle S' \operatorname{Cont}(\Sigma), \rho'\rangle$$

  for all $\rho' \in \mathcal{Q}$.

□

Proof. (Theorem4.1) (1) Act by structural induction on $S \in \mathrm{Stm}$.
If $S \equiv \epsilon$, then both $\langle \epsilon, \rho\rangle \not\to$ and $\langle \epsilon, \epsilon, \{\rho\}\rangle \not\to$. Hence the 'if and only if' holds vacuosly.
Now

$$\langle cS, \rho\rangle \to^* \langle S, \rho''\rangle \to^* \langle \epsilon, \rho'\rangle \iff \langle \epsilon, cS, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, S, \{\rho''\}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \{\rho'\}\rangle$$

thanks to the Lemma 4.8 (1) and inductive hypothesis.
(2) Recall

$$sp(S, \mathcal{P}) = \{\rho' \mid \langle \epsilon, \rho'\rangle \in \mathrm{Post}^*[S]\mathcal{P}\}$$

where $\mathrm{Post}^*[S]\mathcal{P} = \{\langle S', \rho'\rangle \mid \exists \rho \in \mathcal{P}. \langle S, \rho\rangle \to^* \langle S', \rho'\rangle\}$.
($\supseteq$) Let $\rho' \in \mathcal{Q}$. By Lemma 4.8 (3)

$$\langle S, \rho\rangle \to^* \langle \epsilon, \rho'\rangle \quad \exists \rho \in \mathcal{P}.$$

Hence $\rho' \in sp(S, \mathcal{P})$.
($\subseteq$) Let $\rho' \in sp(S, \mathcal{P})$.
Then $\langle \epsilon, \rho'\rangle \in \mathrm{Post}^*[S]\mathcal{P}$, hence there exists $\rho \in \mathcal{P}$ such that $\langle S, \rho\rangle \to^* \langle \epsilon, \rho'\rangle$ and applying Lemma 4.8 (2) we obtain $\rho' \in \mathcal{Q}$. □

Proof. (Theorem4.3) (1) By induction on $S \in \mathrm{Stm}$.
If $S \equiv \epsilon$ the thesis is vacuosly true.
Assume $S \equiv cS$ with $c \in \mathrm{Cmd}$ and act by induction on $c \in \mathrm{Cmd}$.

- $c \equiv$ **skip**$; , x := E; ,$ **while** B **do** T, the thesis is true because sequential and parallel interpreters follow the same rules.

- $c \equiv$ **if** B **then** S_1 **else** S_2.
  Assume that $\langle S_1, [\![B]\!]\mathcal{P}\rangle \to^* \langle \epsilon, \mathcal{P}'_{true}\rangle$ and $\langle S_2, [\![\neg B]\!]\mathcal{P}\rangle \to^* \langle \epsilon, \mathcal{P}'_{else}\rangle$ because otherwise there is nothing to prove. We have

  $\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to_{\mathrm{sc}} \langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, [\![\neg B]\!]\mathcal{P}]_t, S_1, [\![B]\!]\mathcal{P}\rangle \to^*_{\mathrm{sc}}$
  $\langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}_{else}]_t, \epsilon, \mathcal{P}'_{then}\rangle \to_{\mathrm{sc}} \langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}'_{then}]_e, S_2, \mathcal{P}_{else}\rangle \to^*_{\mathrm{sc}}$
  $\langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}'_{then}]_e, \epsilon, \mathcal{P}'_{else}\rangle \to_{\mathrm{sc}} \langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle$.

  Hence

  $$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle \Leftrightarrow \langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle.$$

  Moreover

  $$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to_{\mathrm{pc}} \langle [S]_t, S_1, [\![B]\!]\mathcal{P}\rangle \parallel \langle [S]_e, S_2, [\![\neg B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle [S]_t, \epsilon, \mathcal{P}'_{then}\rangle \parallel \langle [S]_e, \epsilon, \mathcal{P}'_{else}\rangle \to_{\mathrm{pc}} \langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle.$$

  Hence

  $$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle \;\Leftrightarrow\; \langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle$$

  So the equivalence is proved because, by inductive hypothesis, it holds:

  $$\langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle \to^*_{\mathrm{sc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle \;\Leftrightarrow\; \langle \epsilon, S, \mathcal{P}'_{then} \cup \mathcal{P}'_{else}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle$$

(2) By induction on $S \in \mathrm{Stm}$.
Trivial if $S \equiv \epsilon$.
Let us prove it for $cS \in \mathrm{Stm}$ by induction on $c \in \mathrm{Cmd}$.
As in the previous case, the only case to analyze is the $((\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S)$ case because in the other cases sequential and parallel interpreters follow the same rules.
$S' \in \mathcal{C}((\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S)$. Recall that

$$\mathcal{C}((\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S) = [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S] \circ (\mathcal{C}(S_1) \star S) \circ (\mathcal{C}(S_2) \star S) \circ \mathcal{C}(S).$$

Asssume $S' \,\mathrm{Cont}(\Sigma_s) \in \mathcal{C}(S_1) \star S$. Hence, by definition, $S' \equiv T$ where $T \in \mathcal{C}(S_1)$.
We have

$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to_{\mathrm{sc}} \langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, [\![\neg B]\!]\mathcal{P}]_t, S_1, [\![B]\!]\mathcal{P}\rangle \to^*_{\mathrm{sc}}$
$\langle [(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, [\![\neg B]\!]\mathcal{P}]_t, S', \mathcal{P}'\rangle$

hence

$$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)S, \mathcal{P}\rangle \to_{\mathrm{pc}} \langle [S]_t, S_1, [\![B]\!]\mathcal{P}\rangle \parallel \langle [S]_e, S_2, [\![\neg B]\!]\mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle [S]_t, S', \mathcal{P}'\rangle \parallel \langle ...\rangle.$$

On the other hand, assuming that $S' \,\mathrm{Cont}(\Sigma_p) \in \mathcal{C}(S_1) \star S$ implies in the same way that $S' \in \mathcal{C}(S_1)$ and the other implication turns out in the same way. □

4.3 More on Invariants

The function $\mathrm{Inv_c}[S, \mathcal{P}] : \mathcal{C}(S) \to \wp(\mathrm{Store})$, for any continuation $K \in \mathcal{C}(S)$ and store property $\mathcal{P} \in \wp(\mathrm{Store})$, returns the strongest invariant property for the continuation K of the program S when evaluated by the small step collecting interpreter:

$$\mathrm{Inv_c}[S, \mathcal{P}](K) = \bigcup \{\mathcal{Q} \in \wp(\mathrm{Store}) \mid \langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle,\ K \equiv S'\,\mathrm{Cont}(\Sigma)\}$$

Corollary 4.9. *For any $S \in \mathrm{Stm}$, $\mathcal{P} \in \wp(\mathrm{Store})$ and $K \in \mathcal{C}(S)$, it holds:*

$$\mathrm{Inv}[S, \mathcal{P}](K) = \mathrm{Inv_c}[S, \mathcal{P}](K)$$

In particular, $\mathrm{sp}(S, \mathcal{P}) = \mathrm{Inv_c}[S, \mathcal{P}](\epsilon)$.

Proof. Recall

$$\mathrm{Inv}[\mathcal{S}, \mathcal{P}](K) = \bigcup_{\tau \in \mathrm{Trace}[S,\mathcal{P}]} \{\rho \mid \tau_i = \langle K, \rho\rangle\ \exists i \in \mathbb{N}\};$$

$$\mathrm{Inv_c}[\mathcal{S}, \mathcal{P}](K) = \bigcup \{\mathcal{Q} \mid \langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K\}.$$

($\subseteq$) Let $\mathcal{Q} \subseteq \bigcup_{\tau \in \mathrm{Trace}[S,\mathcal{P}]} \{\rho \mid \tau_i = \langle K, \rho\rangle \exists i \in \mathbb{N}\}$.
For every $\rho \in \mathcal{Q}$ there exist a trace $\tau \in \mathrm{Trace}[S, \mathcal{P}]$ and an index $i \in \mathbb{N}$ such that $\tau_i = \langle K, \rho\rangle$;
this means that $\langle S, \bar\rho\rangle \to^* \langle K, \rho\rangle$ for a certain $\bar\rho \in \mathcal{P}$.
By the Lemma 4.8 (2) there exists $\mathcal{Q}'_\rho \in \wp(\mathrm{Store})$ such that

$$\langle \epsilon, S, \{\bar\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}'_\rho\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K.$$

In particular

$$\mathcal{Q} \subseteq \bigcup_{\rho \in \mathcal{Q}} \mathcal{Q}'_\rho \text{ and } \mathcal{Q}'_\rho \subseteq \mathrm{Inv_c}[S, \mathcal{P}](K)\ \forall \rho \in \mathcal{Q}.$$

Hence $\mathcal{Q} \subseteq \mathrm{Inv_c}[S, \mathcal{P}](K)$.

($\supseteq$) Let $\mathcal{Q} \in \{\mathcal{Q} \mid \langle \epsilon, S, \mathcal{P}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K\}$.
In particular for every $\rho \in \mathcal{P}$ there exists a $\mathcal{Q}_\rho \subseteq \mathcal{Q}$ such that $\langle \epsilon, S, \{\rho\}\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}_\rho\rangle$ and hence, by Lemma 4.8 (3),

$$\langle S, \rho\rangle \to^* \langle S'\,\mathrm{Cont}(\Sigma), \rho'\rangle \equiv \langle K, \rho'\rangle \quad \forall \rho' \in \mathcal{Q}_\rho.$$

This implies that $\rho' \in \mathrm{Inv}[S, \mathcal{P}](K)\ \forall \rho' \in \mathcal{Q}_\rho$ and hence $\mathcal{Q}_\rho \subseteq \mathrm{Inv}[S, \mathcal{P}](K)\ \forall \rho \in \mathcal{P}$.
Now

$$\mathcal{Q} = \bigcup_{\rho \in \mathcal{P}} \mathcal{Q}_\rho \ \subseteq\ \mathrm{Inv}[S, \mathcal{P}](K).$$

For the second part it is enough to observe that

$$\mathrm{Inv}[S, \mathcal{P}](\epsilon) = \bigcup_{\tau \in \mathrm{Trace}[S,\mathcal{P}]} \{\rho \mid \tau_i = \langle \epsilon, \rho\rangle\ \exists i \in \mathbb{N}\} = \mathrm{sp}[S, \mathcal{P}].$$

□

We conclude this section with a definition that will be useful later:

Definition 4.10. Consider a program $S \equiv$ **while** B **do** $T \in \mathrm{Stm}$ and let $\mathcal{Q} \in \wp(\mathrm{Store})$. Then the sequence $(\mathcal{Q}_i)_{i\in\mathbb{N}}$ of the concrete loop invariants computed at the entry point of the loop, after i iterations, is defined recursively by the clauses:

$$\begin{cases} \mathcal{Q}_0 = \mathcal{Q} \\ \mathcal{Q}_n = \mathcal{Q}_{n-1} \cup [\![T]\!][\![B]\!]\mathcal{Q}_{n-1} \end{cases}$$

5 Small Step Abstract Interpreter

We show that a small step abstract interpreter is simply an abstract interpretation of the collecting interpreter. We consider a join-semilattice $(A, \leq, \vee)$ such that $(\wp(\mathrm{Store}), A, \gamma, \alpha)$ is a Galois connection, where α and γ denote the corresponding abstraction and concretization maps. In turn, abstract states are defined by $\mathrm{State}_A \triangleq \mathrm{Stm} \times A$.

Let us now define the best correct approximations of the transfer functions:

$$\begin{array}{rr} [\![x := E]\!]^A : A \to A & [\![x := E]\!]^A a \triangleq \alpha([\![x := E]\!]\gamma(a)) \\ [\![B]\!]^A : A \to A & [\![B]\!]^A a \triangleq \alpha([\![B]\!]\gamma(a)) \end{array}$$

Actually, a sound abstract interpreter can be defined for any sound approximation of the transfer functions, and their correctness can be stated just by relying on the concretization map γ as follows:

$$\begin{array}{rlr} [\![x := E]\!]^\sharp : A \to A & \text{such that} & [\![x := E]\!]\gamma(a) \subseteq \gamma([\![x := E]\!]^\sharp a) \\ [\![B]\!]^\sharp : A \to A & \text{such that} & [\![B]\!]\gamma(a) \subseteq \gamma([\![B]\!]^\sharp a) \end{array}$$

Example 5.1. *Interval Abstract Domain Transfers Functions*

Let us give some examples of how to compute Interval abstract transfer functions, as the *best correct approximations* of the collecting ones.
Let $[a, b] \in \mathrm{Int}$ and $k \in \mathbb{Z}$.

- Consider the expression $E \equiv x + k$. Then

$$[\![x := x + k]\!] : \wp(\mathrm{Store}) \to \wp(\mathrm{Store}),$$

$$[\![x := x + k]\!]X = \{\rho[x \mapsto v] \mid \rho \in X, v = \mathbf{E}[\![x := x + k]\!]\rho\} = \{\rho[x \mapsto x + k] \mid \rho \in X\}$$

Hence

$$[\![x := x + k]\!]^{\mathrm{Int}} : \mathrm{Int} \to \mathrm{Int},$$

$$\begin{array}{l} [\![x := x + k]\!]^{\mathrm{Int}}([a, b]) \overset{\text{bca}}{=} \alpha \circ [\![x := x + k]\!] \circ \gamma[a, b] = \\ \alpha([\![x := x + k]\!]\{\rho \in \mathrm{Store} \mid a \leq \rho(x) \leq b\}) = \alpha(\{\rho[x \mapsto x + k] \mid a \leq \rho(x) \leq b\}) = \\ \alpha(\{\rho \in \mathrm{Store} \mid a + k \leq \rho(x) \leq b + k\}) = [a + k, b + k] \end{array}$$

- In the same way, consider the boolean expression $B \equiv x > k$. Recall

$$[\![x > k]\!] : \wp(\mathrm{Store}) \to \wp(\mathrm{Store}),$$

$$[\![x > k]\!]X = \{\rho \in X \mid \mathbf{B}[\![x > k]\!]\rho = \mathit{true}\} = \{\rho \in X \mid \rho(x) > k\}$$

Hence

$$[\![x > k]\!]^{\mathrm{Int}} : \mathrm{Int} \to \mathrm{Int},$$

$$[\![x > k]\!]^{\mathrm{Int}}([a,b]) \stackrel{\text{bca}}{=} \alpha \circ [\![x > k]\!] \circ \gamma[a,b] =$$
$$\alpha([\![x > k]\!]\{\rho \in \mathrm{Store} \mid a \leq \rho(x) \leq b\}) = \alpha(\{\rho \in X \mid a \leq \rho(x) \leq b \wedge \rho(x) > k\})$$

Now, if $a \leq k < b$ then

$$[\![x > k]\!]^{\mathrm{Int}}([a,b]) = \alpha(\{\rho \in X \mid k < \rho(x) \leq b\}) = [k+1, b]$$

If $k < a$ or $k \geq b$ then $\{\rho \in X \mid a \leq \rho(x) \leq b \wedge \rho(x) > k\} = \emptyset$, hence

$$[\![x > k]\!]^{\mathrm{Int}}([a,b]) = \alpha(\emptyset) = \bot_{\mathrm{Int}}.$$

5.1 Parallel Abstract Interpreter

The forward parallel abstract interpreter over the abstract domain A relies on an abstract evaluation stack Stack_A whose records r are defined as follows:

$$\mathrm{Stack}_A \ni r ::= [S]_{\mathbf{t}} \mid [S]_{\mathbf{e}} \mid [(\textbf{while } B \textbf{ do } S)K, a]_{\mathbf{w}}$$

The informal meaning of Stack_A is similar to the stack for the parallel collecting interpreter: if $[K]_{\mathbf{t}}$ or $[K]_{\mathbf{e}}$ are at the top of the stack then the interpreter is currently evaluating in parallel the two branches of an if-then-else command whose continuation is K, while if $[(\textbf{while } B \textbf{ do } S)K, a]_{\mathbf{w}}$ is at the top of the stack then the interpreter is evaluating the while-command **while** B **do** S whose continuation is K and whose current abstract loop invariant is a.

A sequential abstract state in $\mathrm{State}_{\to A}$ of the small step parallel collecting interpreter is defined as a triple

$$\langle \Sigma, S, a \rangle \in \mathrm{Stack}_A^* \times \mathrm{Stm} \times \mathrm{State}_A$$

while the possibly parallel abstract states are defined by

$$\mathrm{State}_{\to_{\mathrm{p},A}} \ni C ::= \langle \Sigma, S, a \rangle \;\mid\; C_1 \parallel C_2$$

The forward parallel abstract interpreter is in Figure 3. Here, the transition relation $\to_{\mathrm{p},A} \subseteq \mathrm{State}_{\to_{\mathrm{p},A}} \times \mathrm{State}_{\to_{\mathrm{p},A}}$ is nondeterministic due to the parallel evaluation of the two branches of an if-then-else, meaning that the execution of 'then' and 'else' statements can be freely interleaved by the abstract interpreter, while at the end, the interpreter computes the lub of the output abstract values of the two branches. This abstract interpreter exploits the best correct approximations $[\![x := E]\!]^A$ and $[\![B]\!]^A$, although it could also use any other correct approximation.

The forward parallel abstract interpreter may also consider a widening operator ∇ on the abstract domain A. This can be obtained by using the following rule which encodes a widening point at the entry point of a while-loop.

$$\frac{a \not\leq a_{\textbf{while}}}{\langle [(\textbf{while } B \textbf{ do } S)K, a_{\textbf{while}}]_{\mathbf{w}} \cdot \Sigma, \epsilon, a \rangle \to_{\mathrm{p},A} \langle [(\textbf{while } B \textbf{ do } S)K, a_{\textbf{while}} \nabla (a_{\textbf{while}} \vee a)]_{\mathbf{w}} \cdot \Sigma, S, [\![B]\!]^A (a_{\textbf{while}} \nabla (a_{\textbf{while}} \vee a)) \rangle}$$

$$\frac{}{\langle \Sigma, \textbf{skip}; K, a\rangle \to_{\mathrm{p},A} \langle \Sigma, K, a\rangle}$$

$$\frac{}{\langle \Sigma, x := E; K, a\rangle \to_{\mathrm{p},A} \langle \Sigma, K, [\![x := E]\!]^A a\rangle}$$

$$\frac{}{\langle \Sigma, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, a\rangle \to_{\mathrm{p},A} \langle [K]_{\mathbf{t}} \cdot \Sigma, S_1, [\![B]\!]^A a\rangle \parallel \langle [K]_{\mathbf{e}} \cdot \Sigma, S_2, [\![\neg B]\!]^A a\rangle}$$

$$\frac{}{\langle [K]_{\mathbf{t}} \cdot \Sigma, \epsilon, a_{\textbf{then}}\rangle \parallel \langle [K]_{\mathbf{e}} \cdot \Sigma, \epsilon, a_{\textbf{else}}\rangle \to_{\mathrm{p},A} \langle \Sigma, K, a_{\textbf{then}} \vee a_{\textbf{else}}\rangle}$$

$$\frac{}{\langle \Sigma, (\textbf{while } B \textbf{ do } S)K, a\rangle \to_{\mathrm{p},A} \langle [(\textbf{while } B \textbf{ do } S)K, a]_{\mathbf{w}} \cdot \Sigma, S, [\![B]\!]^A a\rangle}$$

$$\frac{a \not\leq a_{\textbf{while}}}{\langle [(\textbf{while } B \textbf{ do } S)K, a_{\textbf{while}}]_{\mathbf{w}} \cdot \Sigma, \epsilon, a\rangle \to_{\mathrm{p},A} \langle [(\textbf{while } B \textbf{ do } S)K, a_{\textbf{while}} \vee a]_{\mathbf{w}} \cdot \Sigma, S, [\![B]\!]^A (a_{\textbf{while}} \vee a)\rangle}$$

$$\frac{a \leq a_{\textbf{while}}}{\langle [(\textbf{while } B \textbf{ do } S)K, a_{\textbf{while}}]_{\mathbf{w}} \cdot \Sigma, \epsilon, a\rangle \to_{\mathrm{p},A} \langle \Sigma, K, [\![\neg B]\!]^A a_{\textbf{while}}\rangle}$$

Figure 3: The forward parallel abstract interpreter.

5.2 Examples

We give now several examples to show that the termination of concrete or collecting interpreters and the termination of the abstract one are not related with each others.

Example 5.2. Consider the following program P:

$$\begin{array}{l}
\textcircled{1}\ x := 5;\\
\textcircled{2}\ \textbf{while } x > 1 \textbf{ do}\\
\quad \textcircled{3}\ \textbf{if } (x \bmod 2 = 0) \textbf{ then } \textcircled{4}\ x := x + 3;\\
\qquad \textbf{else } \textcircled{5}\ x := x - 2;\\
\textcircled{6}
\end{array}$$

Here the concrete evaluation terminates, i.e., for any $N \in \mathbb{Z}$, $\langle P, \{x/N\}\rangle \to^* \langle \epsilon, \{x/1\}\rangle$. Furthermore, the forward collecting interpreter also terminates, that is, $\langle \epsilon, P, \mathrm{Store}\rangle \to^*_{\mathrm{pc}} \langle \epsilon, \epsilon, \{x/1\}\rangle$. Also, the strongest invariant properties at the program points of P are as follows:

$$\begin{aligned}\mathrm{Inv}[P, \mathrm{Store}] = \{&\textcircled{1} \mapsto \mathrm{Store}, \textcircled{2} \mapsto \{x/1, x/3, x/5\}, \textcircled{3} \mapsto \{x/3, x/5\},\\ &\textcircled{4} \mapsto \varnothing, \textcircled{5} \mapsto \{x/3, x/5\}, \textcircled{6} \mapsto \{x/1\}\}\end{aligned}$$

Consider the abstract domain Int of Example 2.3.1. It turns out that the abstract evaluation of $\langle \epsilon, P, a\rangle$, where $a \in \mathrm{Int} \setminus \bot_{\mathrm{Int}}$, for $\rightarrow_{\mathrm{p,Int}}$ does not converge. Of course, this non-termination depends on the fact that the assignment $x := x + 3;$ is dead-code, but the abstract domain of intervals is not able to detect this. Hence, if S denotes the body of the while-loop of P then we have the following infinite trace:

$$\begin{aligned}\langle \epsilon, P, a\rangle \rightarrow_{\mathrm{p,Int}} & \langle \epsilon, \textbf{while } x > 1 \textbf{ do } S, [5,5]\rangle \rightarrow_{\mathrm{p,Int}} \langle [\textbf{while } x > 1 \textbf{ do } S, [5,5]]_w, S, [5,5]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, [5,5]]_w, \epsilon, [3,3]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, [3,5]]_w, \epsilon, [1,7]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, [1,7]]_w, \epsilon, [1,9]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, [1,9]]_w, \epsilon, [1,11]\rangle \rightarrow^*_{\mathrm{p,Int}} \quad \cdots\end{aligned}$$

Let us point out that

$$\langle \epsilon, \textbf{while } x > 1 \textbf{ do } S, [5,5]\rangle \rightarrow^*_{\mathrm{p,Int}} \langle [\textbf{while } x > 1 \textbf{ do } S, [3,5]]_w, \epsilon, [1,7]\rangle$$

because:

$$\begin{aligned}&[\![x \bmod 2 = 0]\!]^{\mathrm{Int}}[3,5] = \alpha \circ [\![x \bmod 2 = 0]\!] \circ \gamma([3,5]) = [4,4], \quad [\![x := x + 3]\!]^{\mathrm{Int}}[4,4] = [7,7], \\ &[\![\neg(x \bmod 2 = 0)]\!]^{\mathrm{Int}}[3,5] = [3,5], \quad [\![x := x - 2]\!]^{\mathrm{Int}}[3,5] = [1,3],\end{aligned}$$

and $[1,3] \vee [7,7] = [1,7]$. Hence, the abstract evaluation of $\langle \epsilon, P, a\rangle$ for Int does not terminate.
Of course, if the abstract interpreter makes use of the standard widening operator for intervals then we obtain the following finite trace:

$$\begin{aligned}\langle \epsilon, P, a\rangle \rightarrow_{\mathrm{p,Int}} & \langle \epsilon, \textbf{while } x > 1 \textbf{ do } S, [5,5]\rangle \rightarrow_{\mathrm{p,Int}} \langle [\textbf{while } x > 1 \textbf{ do } S, [5,5]]_w, S, [5,5]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, [5,5]]_w, \epsilon, [3,3]\rangle \rightarrow_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, (-\infty,5]]_w, S, [2,5]\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, (-\infty,5]]_w, \epsilon, [1,7]\rangle \rightarrow_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, (-\infty,+\infty)]_w, S, [2,+\infty)\rangle \rightarrow^*_{\mathrm{p,Int}} \\ & \langle [\textbf{while } x > 1 \textbf{ do } S, (-\infty,+\infty)]_w, \epsilon, [1,+\infty)\rangle \rightarrow_{\mathrm{p,Int}} \\ & \langle \epsilon, \epsilon, (-\infty,1]\rangle\end{aligned}$$

so that the abstract evaluation of $\langle \epsilon, P, a\rangle$ for Int does terminate. □

Example 5.3. Consider the following program P:

$$
\begin{array}{l}
\quad x := 5; \\
\textcircled{p}\ \textbf{while}\ x > 1\ \textbf{do} \\
\quad\quad \textbf{if}\ (x \bmod 2 = 0)\ \textbf{then}\ \{ \\
\quad\quad\quad x := x + 1; \\
\quad\quad \textcircled{q}\ \textbf{while}\ x > 1\ \textbf{do} \\
\quad\quad\quad\quad \textbf{if}\ (x \bmod 2 = 0)\ \textbf{then}\ x := x + 3; \\
\quad\quad\quad\quad \textbf{else}\ x := x - 2; \\
\quad\quad \} \\
\quad\quad \textbf{else}\ x := x - 2;
\end{array}
$$

This is an alteration of the program in Example 5.2 where the dead-code includes a while-loop. Here again, the abstract evaluation of P of the interval abstraction Int does not terminate: at the second iteration for the outermost while-loop W the abstract loop invariant is $[3,5]$, so that $[\![x \bmod 2 = 0]\!]^{\mathrm{Int}}[3,5] = [4,4]$ and, in turn, $[\![x := x+1]\!]^{\mathrm{Int}}[4,4] = [5,5]$, so that, by Example 5.2, the abstract evaluation of the innermost while-loop from the initial abstract state $[5,5]$ does not terminate. Moreover, the abstract evaluation of P reaches the program point $\textcircled{p}$ twice with abstract states: $\langle \epsilon, W, [5,5] \rangle$ and $\langle [W,[5,5]]_w, \epsilon, [3,3] \rangle$. On the other hand, $\mathrm{Inv}[P, \{\langle x/N \rangle\}]_{\textcircled{p}} = \{x/1, x/3, x/5\}$. Furthermore, the evaluation of $\langle \epsilon, P, \mathrm{Store} \rangle$ by the collecting interpreter terminates:

$$\langle \epsilon, P, \mathrm{Store} \rangle \to \langle \epsilon, \epsilon, \{x/1\} \rangle$$

The strongest invariant property at program point $\textcircled{p}$ for the collecting interpreter is $\{x/1, x/3, x/5\}$. Hence, the lub of the abstract values at program point $\textcircled{p}$ along the infinite trace starting from $\langle \epsilon, P, a \rangle$, with $a \neq \bot_{\mathrm{Int}}$, provides the interval $[3,5]$, which is not sound for this program point $\textcircled{p}$, w.r.t. both the concrete and collecting interpreters. Thus, a possibly nonterminating abstract interpreter is inherently unsound. □

Example 5.4. Consider the following program P:

$$
\begin{array}{l}
\textcircled{1}\ x := 1; \\
\textcircled{2}\ \textbf{while}\ x < 2\ \textbf{do} \\
\quad \textcircled{3}\ \textbf{if}\ (x > 0)\ \textbf{then}\ \textcircled{4}\ x := 0; \\
\quad\quad \textbf{else}\ \textcircled{5}\ x := 1; \\
\textcircled{6}
\end{array}
$$

Here, the concrete evaluation of P does not terminate and the strongest invariant properties are as follows:

$$
\begin{array}{l}
\mathrm{Inv}[P, \mathrm{Store}] = \{\textcircled{1} \mapsto \mathrm{Store}, \textcircled{2} \mapsto \{x/0, x/1\}, \\
\qquad\qquad \textcircled{3} \mapsto \{x/0, x/1\}, \textcircled{4} \mapsto \{x/1\}, \textcircled{5} \mapsto \{x/0\}, \textcircled{6} \mapsto \varnothing\}
\end{array}
$$

Also, the evaluation of P by the collecting interpreter terminates. In fact, if W denotes the while-loop of P then we have that:

$$\langle \epsilon, P, \mathrm{Store}\rangle \to_{\mathrm{pc}} \langle \epsilon, W, \{x/1\}\rangle \to^*_{\mathrm{pc}} \langle [W, \{\langle x/1\rangle\}]_w, \epsilon, \{x/0\}\rangle \to^*_{\mathrm{pc}} \\ \langle [W, \{x/0, x/1\}]_w, \epsilon, \{x/0, x/1\}\rangle \to_{\mathrm{pc}} \langle \epsilon, \epsilon, \varnothing\rangle$$

On the other hand, the abstract evaluation of P for Int in this case terminates:

$$\langle \epsilon, P, \top\rangle \to_{\mathrm{Int}} \langle \epsilon, W, [1,1]\rangle \to^*_{\mathrm{Int}} \langle [W, [1,1]]_w, \epsilon, [0,0]\rangle \to^*_{\mathrm{Int}} \langle [W, [0,1]]_w, \epsilon, [0,1]\rangle \to_{\mathrm{Int}} \langle \epsilon, \epsilon, [0,1]\rangle$$

We finally observe that $\mathrm{Trace}[P, \mathrm{Store}]$ is complete for Int. □

Example 5.5. Consider the following program P:

① $x := 1$;
② **while** $x < 2$ **do**
  ③ **if** $(x > 0)$ **then** ④ $x := -1$;
    else ⑤ $x := 1$;
⑥

Here, the concrete evaluation of P does not terminate, while the abstract evaluation of P for Int does terminate. We observe that in this case $\mathrm{Trace}[P, \mathrm{Store}]$ is not complete for Int:

$$\alpha(\llbracket \neg(x > 0)\rrbracket\{-1, 1\}) = [-1, -1] < [-1, 0] = \alpha(\llbracket \neg(x > 0)\rrbracket\gamma(\alpha(\{-1, 1\})))$$

□

Of course, it may also happen that the collecting interpreter does not terminate while the abstract interpreter terminates. It is enough to consider the program P in Example 4.2, whose evaluation by the collecting interpreter does not terminate, while, for any finite abstract domain (e.g., a sign abstraction) the abstract evaluation of P terminates.

5.3 Abstract Traces

The set Trace^∞_A of (finite and infinite, partial and maximal) abstract traces of the abstract interpreter on the abstract domain A is defined as follows:

$$\mathrm{Trace}^\infty_A \triangleq \{\tau \in \mathrm{State}^\infty_{\to_{\mathrm{p},A}} \mid \forall i \in [1, |\tau|).\ \tau_{i-1} \to_{\mathrm{p},A} \tau_i\}.$$

The abstract trace semantics of a program S is in turn defined as follows:

$$\mathrm{Trace}_A[S] \triangleq \{\tau \in \mathrm{Trace}^\infty_A \mid \exists a \in A.\ \tau_0 = \langle \epsilon, S, a\rangle\}$$
$$\mathrm{Trace}_A[S, a] \triangleq \{\tau \in \mathrm{Trace}_A[S] \mid \tau_0 = \langle \epsilon, S, a\rangle\}$$

The abstract execution of S from a terminates if $\mathrm{Trace}_A[S,a]$ has elements of finite lenght.

The function $\mathrm{Inv}^A[S,a] : \mathcal{C}(S) \to A$, for any continuation $K \in \mathcal{C}(S)$, returns the (strongest) abstract invariant for K:

$$\mathrm{Inv}_A[S,a](K) \triangleq \bigvee\{a' \in A \mid \langle \epsilon, S, a\rangle \to^*_{\mathrm{p},A} \langle \Sigma', S', a'\rangle, K \equiv S'\,\mathrm{Cont}(\Sigma')\}$$

Let us define the abstract equivalent of Definition 4.10:

Definition 5.6. Consider a program $S \equiv$ **while** B **do** $T \in \mathrm{Stm}$ and $a \in A$. Then the sequence $(a_i)_{i\in\mathbb{N}}$ of abstract loop invariants computed starting from a, at the entry point of the loop, after i iterations, is defined recursively by the following clauses:

$$\begin{cases} a_0 = a \\ a_n = a_{n-1} \vee [\![T]\!]^A [\![B]\!]^A a_{n-1} \end{cases}$$

5.4 Soundness and Completeness

It turns out that the forward parallel abstract interpreter provides a sound static analyzer in case of termination, meaning that, if the abstract interpreter on a program S terminates, then, for each continuation $K \in \mathcal{C}(S)$, it computes an abstract invariant for K which is a safe approximation of the strongest concrete invariant property for K. Instead, if the abstract interpreter on S does not terminate then the abstract invariants may be unsound.

Example 5.7 (Unsoundness of nonterminating abstract interpreters). Consider the program P in Example 5.3, where we observed that $\mathrm{Inv}[P,\{x/N\}]_{\textcircled{p}} = \{x/1, x/3, x/5\}$ while the abstract evaluation for Int of $\langle \epsilon, P, \{x/[N,N]\}\rangle$ does not terminate and is such that $\mathrm{Inv}^{\mathrm{Int}}[P,\{x/[N,N]\}]_{\textcircled{p}} = \{x/[3,5]\}$, so that it is unsound:

$$\mathrm{Inv}[P,\{x/N\}] \not\subseteq \gamma(\mathrm{Inv}^{\mathrm{Int}}[P,\{x/[N,N]\}])$$

□

If $\mathrm{Trace}_A[S,a]$ elements have finite lenght then $\mathrm{Inv}^A[S,a] : \mathcal{C}(S) \to A$, for any continuation $K \in \mathcal{C}(S)$, returns the (strongest) abstract invariant for K.

Theorem 5.8 (Soundness). *If* $\mathrm{Trace}_A[S,a]$ *elements have finite lenght then for any* $K \in \mathcal{C}(S)$,

$$\alpha(\mathrm{Inv}[S,\gamma(a)](K)) \leq_A \mathrm{Inv}^A[S,a](K).$$

Proof. Recall that

$$\mathrm{Inv}[S,\gamma(a)] : \mathcal{C}(S) \to \wp(\mathrm{Store}), \quad \mathrm{Inv}^A[S,a] : \mathcal{C}(S) \to A$$

where

$$\mathrm{Inv}[S,\gamma(a)](K) = \bigcup\{\mathcal{Q} \mid \langle \epsilon, S, \gamma(a)\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K\}$$

and

$$\mathrm{Inv}^A[S,a](K) = \bigvee\{a' \mid \langle \epsilon, S, a\rangle \to^*_{\mathrm{p},A} \langle \Sigma, S', a'\rangle,\ S' \,\mathrm{Cont}(\Sigma) \equiv K\}.$$

For the sake of clarity, these sets will be referred to respectively as $\{\mathcal{Q} \mid \gamma(a),\ K\}$ and $\{a' \mid a,\ K\}$.

Since $(\wp(\mathrm{Store}), \gamma, A, \alpha)$ is a Galois connection,

$$\alpha(\mathrm{Inv}[S,\gamma(a)](K)) = \alpha(\bigcup\{\mathcal{Q} \mid \gamma(a),\ K\}) = \bigvee\{\alpha(\mathcal{Q}) \mid \gamma(a),\ K\}.$$

The strategy of the proof is to show that, for any $K \in \mathcal{C}(S)$,

$$\forall \mathcal{Q} \in \{\mathcal{Q} \mid \gamma(a), K\}\ \exists a' \in \{a' \mid a, K\} \text{ such that } \ \alpha(\mathcal{Q}) \leq_A a'.$$

In such a case, we say that the program S is sound.
This is enough to ensure the truth of the thesis; in fact, since the abstract traces are finite, $\{a' \mid a, K\} = \{a_1, ..., a_n\}$ for some $a_1, ..., a_n \in A$. Hence

$$\forall \mathcal{Q} \in \{\mathcal{Q} \mid \gamma(a), K\} \text{ it holds } \alpha(\mathcal{Q}) \leq_A \bigvee_{1\leq i\leq n} a_i$$

$$\text{Hence} \quad \bigvee\{\alpha(\mathcal{Q}) \mid \gamma(a), K\} \leq_A \bigvee_{1\leq i\leq n} a_i$$

$$\text{which is} \quad \alpha(\mathrm{Inv}[S,\gamma(a)](K)) \leq_A \mathrm{Inv}^A[S,a](K).$$

First of all, assume that $\mathcal{C}(S)$ has no elements starting for 'while'.
Let $K \in \mathcal{C}(S)$ and fix $\mathcal{Q} \in \{\mathcal{Q} \mid \gamma(a),\ K\}$.
Here the rules used to obtain $\mathcal{Q}$ are simply determined by the syntax of S and do not depend on the store properties.
Hence, if $\langle \epsilon, S, \gamma(a)\rangle \to^{(n)}_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle, \quad S' \,\mathrm{Cont}(\Sigma) \equiv K$, we can act by induction on n:

- the case $n = 0$ follows immediately since $\alpha\gamma(a) = a$;
- assume now the $n - 1^{th}$ configuration is $\langle \Sigma', S'', \mathcal{Q}'\rangle$. It is lead to the n^{th} configuration through one of the rules of the interpreter. By induction, there exists $a'' \in A$ such that $\alpha(\mathcal{Q}') \leq_A a''$.

  Hence:
  if the n^{th} rule is the **skip**-one there is nothing to prove;
  if it is an assignment, it is enough to observe that

  $$\alpha(\mathcal{Q}) = \alpha([\![x := E]\!]\mathcal{Q}') \underset{\text{G.C.}}{\leq_A} \alpha([\![x := E]\!]\gamma\alpha(\mathcal{Q}')) \leq_A \alpha([\![x := E]\!]\gamma(a'')) \underset{\text{bca}}{=} [\![x := E]\!]^A(a'');$$

  if it is the opening **if** B **then** S_1 **else** S_2-rule, observe that

  $$\alpha(\mathcal{Q}) = \alpha([\![B]\!]\mathcal{Q}') \leq_A \alpha([\![B]\!]\gamma\alpha(\mathcal{Q}')) \leq_A \alpha([\![B]\!]\gamma(a'')) = [\![B]\!]^A(a'');$$

  if it is the contracting **if** B **then** S_1 **else** S_2-rule, then

  $$\alpha(\mathcal{Q}) = \alpha(\mathcal{Q}_t \cup \mathcal{Q}_e) = \alpha(\mathcal{Q}_t) \vee \alpha(\mathcal{Q}_e) \underset{\text{induction}}{\leq_A} a''_t \vee a''_e.$$

Hence, it is always possible to find $a' \in \{a' \mid a,\ K\}$ such that $\alpha(\mathcal{Q}) \leq_A a'$.

Next, assume $S \equiv\ S_1(\ \textbf{while}\ B\ \textbf{do}\ T)\ S_2$ with $S_1,\ T,\ S_2$ sound subprograms.
Let

$$\begin{aligned}\ell_1 &= \ell(S);\\ \ell_p &= \ell((\textbf{while}\ B\ \textbf{do}\ T)\ S_2);\\ \ell_q &= \ell(S_2).\end{aligned}$$

Fix $K \in \mathcal{C}(S)$ and let us distinguish three different cases, according to the position of K in the CFG of S, i.e., according to the index of $\ell(K)$. Observe that $\ell_1 \leq \ell(K)$.

(1) Assume $\ell(K) \lneq \ell_p$. Here, there is nothing to prove since S_1 is assumed to be sound.

(2) Now, assume $\ell_p \leq \ell(K) \lneq \ell_q$.
Fix $\mathcal{Q} \in \{\mathcal{Q} \mid \gamma(a), K\}$. Assume that $\mathcal{Q}$ is reached at the n^{th} iteration of the loop.
Using the notations of Definition 4.10 it turns out that: if $\ell_p = \ell(K)$ then $\mathcal{Q} = \mathcal{Q}_n$; if $\ell_p \lneq \ell(K)$ then $\mathcal{Q} = [\![T']\!][\![B]\!]\mathcal{Q}_n$, with $T' \subseteq T$ and hence sound.
Now, observe that the application of the first case at $\ell_p = \ell((\textbf{while}\ B\ \textbf{do}\ T)\ S_2)$, provides the existence of $a_0 \in \{a' \mid a, (\textbf{while}\ B\ \textbf{do}\ T)\ S_2\}$, such that $\alpha(\mathcal{Q}_0) \leq_A a_0$.
Recall that the abstract traces are finite, hence there exists $\bar{a} \in \{a' \mid a, (\textbf{while}\ B\ \textbf{do}\ T)\ S_2\}$ which is the strongest abstract loop invariant, i.e.

$$[\![T]\!]^A[\![B]\!]^A\bar{a} \leq_A \bar{a}.$$

Obviously, $a_0 \leq_A \bar{a}$ and by transitivity $\alpha(\mathcal{Q}_0) \leq \bar{a}$. Apply the following

Lemma 5.9. *With the notation above*

$$\alpha(\mathcal{Q}_0) \leq_A \bar{a}\ \Rightarrow\ \alpha(\mathcal{Q}_n) \leq_A \bar{a}.$$

Proof. By induction on n.
Recall that, by Definition 4.10, $\mathcal{Q}_n = \mathcal{Q}_{n-1} \cup [\![T]\!][\![B]\!]\mathcal{Q}_{n-1}$.
If $n = 0$ there is nothing to prove;
if $n = 1$: $\mathcal{Q}_1 = \mathcal{Q}_0 \cup [\![T]\!][\![B]\!]\mathcal{Q}_0$. Observe that

$$\begin{aligned}\alpha([\![T]\!][\![B]\!]\mathcal{Q}_0) \leq_A \alpha([\![T]\!][\![B]\!]\gamma\alpha(\mathcal{Q}_0)) &\leq_A \alpha([\![T]\!][\![B]\!]\gamma(\bar{a}))\\ &\leq_A \alpha([\![T]\!]\gamma\alpha([\![B]\!]\gamma(\bar{a}))) = [\![T]\!]^A[\![B]\!]^A\bar{a} \leq_A \bar{a}\end{aligned}$$

and hence

$$\alpha(\mathcal{Q}_1) = \alpha(\mathcal{Q}_0) \vee \alpha([\![T]\!][\![B]\!]\mathcal{Q}_0) \leq_A \bar{a}.$$

Assume $\alpha(\mathcal{Q}_{n-1}) \leq_A \bar{a}$. Then, applying the same argument as before

$$\alpha(\mathcal{Q}_n) = \alpha(\mathcal{Q}_{n-1}) \vee \alpha((\llbracket T \rrbracket \llbracket B \rrbracket \mathcal{Q}_{n-1}) \leq_A \bar{a} \vee \llbracket T \rrbracket^A \llbracket B \rrbracket^A \bar{a} = \bar{a}$$

□

We have proved that, at the entry point of the loop, soundness holds at every iteration.
What is left to do is leading soundness at K.

Recall $\mathcal{Q} = \llbracket T' \rrbracket \llbracket B \rrbracket \mathcal{Q}_n$, with T' sound.
Observe that

$$\mathcal{Q} \in \{\mathcal{Q} \,|\, \langle \Sigma, T, \mathcal{Q}_n \rangle \rightarrow^*_{\text{pc}} \langle \Sigma', S', \mathcal{Q} \rangle,\ S' \,\text{Cont}(\Sigma') \equiv K\} \quad \text{and} \quad \alpha(\mathcal{Q}_n) \leq_A \bar{a}.$$

The conclusion easily follows since T is sound.

(3) Assume now $\ell_q \leq \ell(K)$, hence $K \in \mathcal{C}(S_2)$ (K is fixed below the while-loop).
Fix $\mathcal{Q} \in \{\mathcal{Q} \,|\, \gamma(a),\ K\}$.
If the concrete traces are finite, there exists $\mathcal{Q}_w \in \wp(\text{Store})$ such that $\llbracket T \rrbracket \llbracket B \rrbracket \mathcal{Q}_w \subseteq \mathcal{Q}_w$, and hence the loop output store property is $\llbracket \neg B \rrbracket \mathcal{Q}_w$.
Let $\bar{a}$ be the strongest abstract loop invariant as before (it does exist since the abstract traces are finite). It follows, from the second case, that $\alpha(\mathcal{Q}_w) \leq_A \bar{a}$. Hence

$$\alpha(\llbracket \neg B \rrbracket \mathcal{Q}_w) \leq_A \alpha(\llbracket \neg B \rrbracket \gamma\alpha(\mathcal{Q}_w)) \leq_A \alpha(\llbracket \neg B \rrbracket \gamma(\bar{a})) \underset{\text{b.c.a}}{=} \llbracket \neg B \rrbracket^A \bar{a}$$

and now the conclusion follows since S_2 is sound.
If there is a concrete infinite trace (i.e. the collecting interpreter cannot exit from the while-loop), then $\mathcal{Q} = \emptyset$ and

$$\alpha(\emptyset) = \bot_A \leq_A a,\ \forall a \in A.$$

Now, the theorem is proved for $S \equiv S_1(\textbf{while}\ B\ \textbf{do}\ T)S_2$ with $S_1,\ S_2,\ T$ sound. Let us show how every program can fit into this scheme.
A generic program may consist of one or more of the following cases: two (or more) nested while-loops, two (or more) disjoint subsequent while-loops, at least a while-loop nested in an if-then-else command.

(i) Assume $\mathcal{C}(S)$ has two elements starting with 'while', namely **while** B **do** $T \subseteq S$ and **while** C **do** $R \subseteq T$. The key observation is that we can assume $T \equiv S_1(\textbf{while}\ C\ \textbf{do}\ R)S_2$ with $S_1,\ S_2$ sound.
Looking at $\mathcal{C}(R)$, if it has an element starting for 'while', observe that it can be decomposed as $R \equiv S'_1(\textbf{while}\ C'\ \textbf{do}\ R')S'_2$ with $S'_1,\ S'_2$ sound. Repeat this stuff until the innermost loop body is reached: it has no more while-loops, hence it is sound.
For the sake of clarity, we assume that $\mathcal{C}(R)$ has no elements starting for 'while'.

– Let K be fixed inside the internal loop body.
If the internal loop does not terminate, it reduces to the second case of the proof.
If the internal loop terminates, we show that soundness holds at K, after n iterations of the external one, for every n. This provides soundness at K, either if the external loop terminates or not.
The key observation is that, at the i^{th} iteration ($i \geq 0$) of the external loop, the second case of the proof provides soundness at K for the store properties related to that precise iteration. In other words, it is possible to define

$$\mathrm{Inv}^{(i)}[S, \gamma(a)](K) = \bigcup \{ \mathcal{Q} \mid \langle \Sigma, T, [\![B]\!] \mathcal{Q}_w^{(i)} \rangle \rightarrow_{\mathrm{pc}}^{*} \langle \Sigma', S', \mathcal{Q} \rangle,\ S' \, \mathrm{Cont}(\Sigma') \equiv K \}$$

where $\mathcal{Q}_w^{(i)}$ is the store property computed at the entry point of the external loop, at the i^{th} iteration. Moreover, we can assume, by the second case, that there exists $a^{(i)} \in \{a' \mid a, K\}$ such that $\alpha(Inv^{(i)}[S, \gamma(a)](K)) \leq_A a^{(i)}$.
At this point, the concrete invariant, computed after n iterations of the external loop at K, is

$$\bigcup_{i \leq n} \mathrm{Inv}^{(i)}[S, \gamma(a)](K).$$

We want to show, by induction on $n \geq 1$, that

$$\alpha(\bigcup_{i \leq n} \mathrm{Inv}^{(i)}[S, \gamma(a)](K)) \leq_A \mathrm{Inv}^{A}[S, a](K).$$

The case $n = 1$ is easy since, from the previous observation, we know

$$\alpha(\mathrm{Inv}^{(0)}[S, \gamma(a)](K)) \leq_A a^{(0)}$$

for $a_0 \in \{a' \mid a, K\}$. Obviously $a_0 \leq_A \mathrm{Inv}^{A}[S, a](K)$.
By induction, assume that

$$\alpha(\bigcup_{i \leq n} \mathrm{Inv}^{(i)}[S, \gamma(a)](K)) \leq_A \mathrm{Inv}^{A}[S, a](K).$$

Now, from the previous observation, there exists $a_n \in \{a' \mid a, K\}$ such that

$$\alpha(\mathrm{Inv}^{(n)}[S, \gamma(a)](K)) \leq_A a_n.$$

Hence

$$\begin{aligned}
\alpha(\bigcup_{i \leq n+1} \mathrm{Inv}^{(i)}[S, \gamma(a)](K)) &= \alpha(\bigcup_{i \leq n} \mathrm{Inv}^{(i)}[S, \gamma(a)](K) \ \cup \mathrm{Inv}^{(n)}[S, \gamma(a)](K)) \\
&= \alpha(\bigcup_{i \leq n} \mathrm{Inv}^{(i)}[S, \gamma(a)](K)) \ \vee \alpha(\mathrm{Inv}^{(n)}[S, \gamma(a)](K)) \\
&\underset{\text{induction}}{\leq_A} \mathrm{Inv}^{A}[S, a](K) \vee a_n = \mathrm{Inv}^{A}[S, a](K)
\end{aligned}$$

where the last equality is due to the fact that $a_n \leq_A \mathrm{Inv}^A[S, a](K)$.

– Let now K be fixed inside the external loop body, but below the internal one.
Here, the thrid case provides soundness at K, at each iteration of the external loop (either if the internal one terminates or not). Following the same induction argument of the previous case, soundness is provided after n iterations of the external loop, for every $n \in \mathbb{N}$.

The same is true if K is fixed inside the external loop body and above the internal one. In this case, the first case of the proof provides soundness at K, at each iteration of the external loop. As before, the induction argument provides soundness after n iterations of the external loop, for every $n \in \mathbb{N}$.

(ii) Assume now to have two disjoint subsequent while-loops, i.e. **while** B **do** $T \subseteq S$ and **while** C **do** $R \subseteq S$, both T and R sound.
K could be inside the second loop body or below it. In both cases, if the first loop does not terminate, the third case of the proof provides soundness at K.
If it terminates, fix a continuation K' below the first loop and above the second one.
The third case provides soundness at K'. Moreover, the second case applied starting from K' (if K is inside the second loop body), or the third case starting from K' (if K is below the second loop body) provides soundness at K.
Notice that, if the disjoint subsequent while-loops are more than two and K is fixed below or inside the n^{th} one, it is enough to repeat this argument a suitable number of times. In fact, the above discussion has provided soundness below the first two loops. By an easy induction argument, soundness is provided below or inside the n^{th} one, precisely at K.

(iii) Assume to have a while-loop nested in an if-then-else command, i.e.
if B **then** S_1 **else** $S_2 \subseteq S$ and **while** C **do** $R \in \mathcal{C}(S_1)$. In particular, $S_1 \equiv S_1'(\textbf{while}\ C\ \textbf{do}\ R)S_2'$.

If K is fixed in the true branch above, inside, below the while-loop body, there is nothing to prove because these are, respectively, the first, second, third cases of the proof.

If K is fixed in the other branch, notice that $\ell(\textbf{while}\ C\ \textbf{do}\ R) \lneq \ell(K)$. But, since the interpreter works in parallel, the information flow arriving at K does not pass through **while** C **do** R. Hence, soundness at K is provided by the suitable case of the proof, regardless of what happens in the other branch.
If the while-loops are more than one, they can be both in the same branch or not, nested or subsequent. In all these cases, soundness at K is provided combining some of the previous discussions.

Resuming, given a generic program S and $K \in \mathcal{C}(S)$, these cases (individually or combined) are enough to provide soundness at K.
These scheme will be reused for the following theorem.

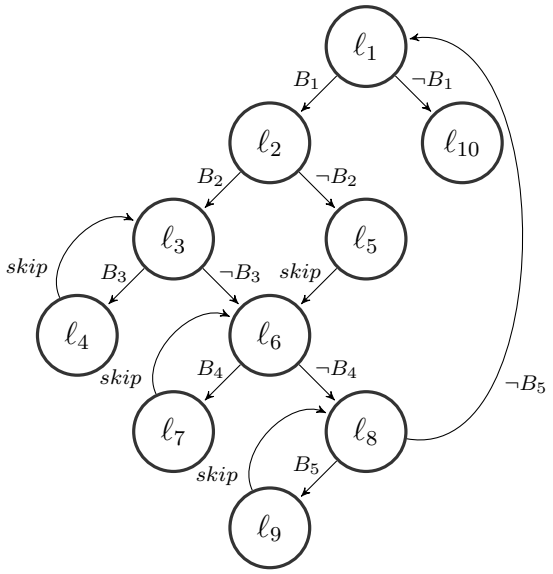
□

Let us exhibit an example of how to treat a technically difficult program, combining the cases of the proof.

Example 5.10. Consider the following program P

while B_1 **do**
  if B_2 **then**
    while B_3 **do skip**;
  else skip;
  while B_4 **do skip**;
  while B_5 **do skip**;

and its CFG (we omit the computation of $\mathcal{C}(P)$ for the sake of clarity)



Assume to be at the i^{th} iteration of the external loop. Let $\mathcal{Q}^{(i)}$ be the store property at ℓ_1, and assume that there exists $a^{(i)} \in A$ such that $\alpha(\mathcal{Q}^{(i)}) \leq_A a^{(i)}$.
Look at the loop body S: it consists of a while-loop nested in a conditional command in ℓ_3, and two disjoint subsequent while-loops in ℓ_6 and ℓ_8.
Notice that the store property reaching ℓ_2 is $[\![B_1]\!]\mathcal{Q}^{(i)}$, and $\alpha([\![B_1]\!]\mathcal{Q}^{(i)}) \leq_A [\![B_1]\!]^A a^{(i)}$. The case (iii) of the proof provides soundness at $\ell_3,\ \ell_4,\ \ell_5$.
Now, if the loop at ℓ_3 does not terminate, soundness trivially holds at the subsequent nodes. If it terminates, what arrives at ℓ_6 is $\mathcal{Q}_6^{(i)} = [\![\neg B_3]\!]\mathcal{Q}_{w_3} \cup [\![\neg B_2]\!][\![B_1]\!]\mathcal{Q}^{(i)}$, where $\mathcal{Q}_{w_3}$ is the strongest loop invariant and there exist a_{w_3} and a_2 such that

$$\begin{cases} \alpha(\mathcal{Q}_{w_3}) \leq_A a_{w_3} \\ \alpha([\![\neg B_2]\!][\![B_1]\!]\mathcal{Q}^{(i)})) \leq_A a_2 \end{cases} \qquad \text{hence} \quad \alpha(\mathcal{Q}_6^{(i)}) \leq_A a_{w_3} \vee a_2.$$

The application of case (ii) of the proof, provides soundness at $\ell_6,\ \ell_7,\ \ell_8,\ \ell_9$. We have obtained that S is sound. Soundness at ℓ_1 is consequently provided after n iterations of the external loop, for every n.
Now, let $K \in \mathcal{C}(S)$ be fixed at ℓ_4, i.e., in one of the nested loops. Notice that the external loop-body S can be written as

$$S_1(\textbf{while } B_3 \textbf{ do skip};)S_2$$

with $S_1,\ S_2$ sound. Soundness at K is, hence, provided by case (i). □

Actually, this result can be refined requiring a stronger hypothesis.

Definition 5.11. Given an abstraction A, a command $c \in \mathrm{Cmd}$ is *complete* for a set of stores $X \in \wp(\mathrm{Store})$ when:

- $c \equiv$ **skip**; is always complete for X
- $c \equiv x := E$; is complete for X when $\alpha([\![x := E]\!]X) = [\![x := E]\!]^A\alpha(X)$
- $c \equiv$ **if** B **then** S_1 **else** S_2 is complete for X when $\alpha([\![B]\!]X) = [\![B]\!]^A\alpha(X)$ and $\alpha([\![\neg B]\!]X) = [\![\neg B]\!]^A\alpha(X)$
- $c \equiv$ **while** B **do** S is complete for X when $\alpha([\![B]\!]X) = [\![B]\!]^A\alpha(X)$ and $\alpha([\![\neg B]\!]X) = [\![\neg B]\!]^A\alpha(X)$

Definition 5.12. A trace $\tau \in \mathrm{Trace}[S]$ is *complete* for A when for any $cK \in \mathcal{C}(S)$, c is complete for $\mathrm{Inv}[\tau](cK)$. $\mathrm{Trace}[S, In]$ is called complete for A when any trace $\tau \in \mathrm{Trace}[S, In]$ is complete for A.

Theorem 5.13 (Completeness). *Assume* $\mathrm{Trace}[S, In]$ *has finite elements, is complete for* A*, and* $\mathrm{Trace}_A[S, \alpha(In)]$ *has finite elements. Then, for any* $K \in \mathcal{C}(S)$*,*

$$\alpha(\mathrm{Inv}[S, In](K)) = \mathrm{Inv}^A[S, \alpha(In)](K)$$

Proof. First of all, notice that the thesis is equivalent to showing that, for every $K \in \mathcal{C}(S)$:

$$\bigvee\{a' \mid \langle \epsilon, S, \alpha(In)\rangle \to^*_{\mathrm{p},A} \langle \Sigma, S', a'\rangle,\ S' \,\mathrm{Cont}(\Sigma) \equiv K\} = \\ = \bigvee\{\alpha(\mathcal{Q}) \mid \langle \epsilon, S, In\rangle \to^*_{\mathrm{pc}} \langle \Sigma, S', \mathcal{Q}\rangle,\ S' \,\mathrm{Cont}(\Sigma) \equiv K\}.$$

For the sake of clarity we refer to these sets, respectively, as $\{a' \mid \alpha(In), K\}$ and $\{\alpha(\mathcal{Q}) \mid In, K\}$.

Let us begin with '$\geq_A$'.
It follows directly from GC properties, since

$$\alpha(\mathrm{Inv}[S, In](K)) \leq_A \alpha(\mathrm{Inv}[S, \gamma\alpha(In)](K)) \leq_A \mathrm{Inv}^A[S, \alpha(In)](K)$$

by Theorem 5.8.

For the other inclusion, it is enough to show that, for every $K \in \mathcal{C}(S)$, $\{a' \mid \alpha(In), K\} \subseteq \{\alpha(\mathcal{Q}) \mid In, K\}$. In this case we say that S is complete.
As in the previous theorem, consider the case in which $\mathcal{C}(S)$ has no elements starting for 'while'. Fix $K \in \mathcal{C}(S)$ and $a' \in \{a' \mid \alpha(In), K\}$. Notice that the rules leading to a' from $\alpha(In)$, depend only on the syntax of S and not on the store properties.
Hence, assuming that $\langle \epsilon, S, \alpha(In)\rangle \to^{(n)}_{\mathrm{p},A} \langle \Sigma, S', a'\rangle, \quad S' \operatorname{Cont}(\Sigma) \equiv K$ with $n \geq 0$, we can act by induction on $n \geq 0$.

- The case $n = 0$ is trivial since $\alpha(In) = \alpha(In)$;
- Assume the $n-1^{th}$ configuration is $\langle \Sigma', S'', a''\rangle$. By induction, there exists $\mathcal{Q}' \in \wp(\mathrm{Store})$ such that $a'' = \alpha(\mathcal{Q}')$.
  Moreover, the n^{th} configuration is reached through one of the abstract interpreter rules.
  If it is the **skip**-rule then
  $$a' = a'' = \alpha(\mathcal{Q}');$$
  if it is the assignment rule, then
  $$a' = [\![x := E]\!]^A(a'') = [\![x := E]\!]^A(\alpha(\mathcal{Q}')) \underset{\text{completeness}}{=} \alpha([\![x := E]\!]\mathcal{Q}');$$
  if it is the opening **if** B **then** S_1 **else** S_2-rule, then
  $$a' = [\![B]\!]^A(a'') \underset{\text{completeness}}{=} \alpha([\![B]\!]\mathcal{Q}');$$
  if it is the contracting **if** B **then** S_1 **else** S_2-rule, then
  $$a' = a''_t \vee a''_e \underset{\text{induction}}{=} \alpha(\mathcal{Q}'_t) \vee \alpha(\mathcal{Q}'_e) = \alpha(\mathcal{Q}_t \cup \mathcal{Q}_e).$$

Now let us consider $S \equiv S_1(\textbf{while } B \textbf{ do } T)S_2$ with $S_1,\ T$ and S_2 complete. The general case can be reached following the scheme in the second part of Theorem 5.8 proof.
Let

$$\begin{aligned}
&\ell_1 = \ell(S);\\
&\ell_p = \ell((\textbf{while } B \textbf{ do } T)\ S_2);\\
&\ell_q = \ell(S_2).
\end{aligned}$$

(1) Assume $\ell(K) \lneq \ell_p$, i.e. K is fixed inside S_1. There is nothing to prove, because S_1 is assumed to be complete.

(2) Assume $\ell_n \leq \ell(K) \lneq \ell_m$, i.e. K is a continuation taken inside the loop body T.
Fix $a' \in \{a' \mid \alpha(In), K\}$. Assume a' is reached at the n^{th} iteration of the loop. Let $\mathcal{Q}_w$ and a_w

be the concrete and abstract store properties, before entering in the loop. By the previous case, it holds $\alpha(\mathcal{Q}_w) = a_w$.

Now recall the notations of Definition 5.6: $\begin{cases} a_0 = a_w \\ a_n = a_{n-1} \vee [\![T]\!]^A [\![B]\!]^A a_{n-1} \end{cases}$

We need the following

Lemma 5.14. *With the notation above, for all $n \in \mathbb{N}$, there exists $\mathcal{Q} \in \wp(\mathrm{Store})$ such that*

$$a_n = \alpha(\mathcal{Q}_n).$$

Proof. By induction on $n \geq 0$.
For $n = 0$ it is enough to set $\mathcal{Q}_0 = \mathcal{Q}_w$.
Assume $a_{n-1} = \alpha(\mathcal{Q}_{n-1})$, for some $\mathcal{Q}_{n-1} \in \wp(\mathrm{Store})$. Now

$$a_n = a_{n-1} \vee [\![T]\!]^A [\![B]\!]^A a_{n-1} \underset{\text{induction}}{=} \alpha(\mathcal{Q}_{n-1}) \vee [\![T]\!]^A [\![B]\!]^A \alpha(\mathcal{Q}_{n-1}) \underset{\text{completeness}}{=}$$
$$\alpha(\mathcal{Q}_{n-1}) \vee \alpha([\![T]\!][\![B]\!]\mathcal{Q}_{n-1}) = \alpha(\mathcal{Q}_{n-1} \cup [\![T]\!][\![B]\!]\mathcal{Q}_{n-1}).$$

To complete the proof set $\mathcal{Q}_n = \mathcal{Q}_{n-1} \cup [\![T]\!][\![B]\!]\mathcal{Q}_{n-1}$. □

Now, if $\ell(K) = \ell_p$, $a' = \alpha(\mathcal{Q}_n)$ and the conclusion follows from the lemma.
If $\ell_p \lneq \ell(K)$ then $a' = [\![T']\!]^A [\![B]\!]^A a_n$ with $T' \subseteq T$. Since $a_n = \alpha(\mathcal{Q}_n)$, the thesis follows from the fact that T is complete.

(3) Assume $\ell_q \leq \ell(K)$, i.e. K is fixed inside S_2.
Fix $a' \in \{a' \mid \alpha(In), K\}$.
Focus on the while-loop: with the notation above, since the abstract traces are finite, there exists $n \in \mathbb{N}$ such that $[\![T]\!]^A [\![B]\!]^A a_n \leq_A a_n$, i.e. a_n is the strongest abstract loop invariant.
By the previous case, there exists $\mathcal{Q}_w \in \wp(\mathrm{Store})$ such that $\alpha(\mathcal{Q}_w) = a_n$.
Observe that

$$a' = [\![S']\!][\![\neg B]\!]a_n$$

with $S' \subseteq S_2$, and hence complete.
We have two cases:

- If $[\![T]\!][\![B]\!]\mathcal{Q}_w \subseteq \mathcal{Q}_w$ the thesis follows from the first case;
- If $[\![T]\!][\![B]\!]\mathcal{Q}_w \not\subseteq \mathcal{Q}_w$ the collecting interpreter goes through T again. Since the concrete traces are supposed to be finite, there exists $\tilde{\mathcal{Q}} \in \wp(\mathrm{Store})$ such that
  $[\![T]\!][\![B]\!]\tilde{\mathcal{Q}} \subseteq \tilde{\mathcal{Q}}$.
  We claim to show that $\alpha(\tilde{\mathcal{Q}}) = \alpha(\mathcal{Q}_w)$.

  Assume $\tilde{\mathcal{Q}}$ is reached after n extra iterations of the loop, and proceed by induction on n.
  Obviously, it holds that $\mathcal{Q}_w \subseteq \tilde{\mathcal{Q}}$. By monotonicity $\alpha(\mathcal{Q}_w) \leq_A \alpha(\tilde{\mathcal{Q}})$, hence it is enough

to show the other inequality.
If $n = 1$, we have $\tilde{\mathcal{Q}} = \mathcal{Q}_w \cup [\![T]\!][\![B]\!]\mathcal{Q}_w$. Hence

$$\alpha(\tilde{\mathcal{Q}}) = \alpha(\mathcal{Q}_w \cup [\![T]\!][\![B]\!]\mathcal{Q}_w) =$$
$$\alpha(\mathcal{Q}_w) \vee \alpha([\![T]\!][\![B]\!]\mathcal{Q}_w) \underset{\text{completeness}}{=} a_n \vee [\![T]\!]^A[\![B]\!]^A a_n \leq_A a_n.$$

Let $n \geq 2$ and assume $\tilde{\mathcal{Q}}_{n-1}$ is the loop invariant after $n-1$ extra iterations. By induction it holds that $\alpha(\tilde{\mathcal{Q}}_{n-1}) \leq_A \alpha(\mathcal{Q}_w)$.
Observe:

$$\alpha(\tilde{\mathcal{Q}}) = \alpha(\tilde{\mathcal{Q}}_{n-1} \cup [\![T]\!][\![B]\!]\tilde{\mathcal{Q}}_{n-1}) = \alpha(\tilde{\mathcal{Q}}_{n-1}) \vee \alpha([\![T]\!][\![B]\!]\tilde{\mathcal{Q}}_{n-1}) \underset{\text{completeness}}{=}$$
$$\alpha(\tilde{\mathcal{Q}}_{n-1}) \vee [\![T]\!]^A[\![B]\!]^A\alpha(\tilde{\mathcal{Q}}_{n-1}) \underset{\text{induction}}{\leq_A} \alpha(\mathcal{Q}_w) \vee [\![T]\!]^A[\![B]\!]^A(\alpha(\mathcal{Q}_w)) =$$
$$a_n \vee [\![T]\!]^A[\![B]\!]^A a_n = a_n.$$

Hence, $\alpha(\tilde{\mathcal{Q}}) = \alpha(\mathcal{Q}_w) = a_n$ and the thesis follows since S_2 is complete.

□

5.5 Safety Verification

Let $S \in \mathrm{Stm}$ and $In \in \wp(\mathrm{Store})$. Let $\mathrm{inv} : \mathcal{C}(S) \to \wp(\mathrm{Store})$ be a program point property for S. We write

$$[S, In] \models \mathrm{inv} \quad \text{when} \quad \forall K \in \mathcal{C}(S).\ \mathrm{Inv}[S, In](K) \subseteq \mathrm{inv}(K).$$

As usual, let $(\wp(\mathrm{Store}), A, \gamma, \alpha)$ be a Galois connection. inv is called A-representable when for any $K \in \mathcal{C}(S)$, $\gamma(\alpha(\mathrm{inv}(K))) = \mathrm{inv}(K)$. We write

$$[S, In] \models^A \mathrm{inv} \quad \text{when} \quad \tau_A[S, \alpha(In)] \text{ is finite and}$$
$$\forall K \in \mathcal{C}(S).\ \gamma(\mathrm{Inv}^A[S, \alpha(In)](K)) \subseteq \mathrm{inv}(K).$$

It turns out that abstract safety verification is sound, that is:

$$[S, In] \models^A \mathrm{inv} \ \Rightarrow\ [S, In] \models \mathrm{inv}$$

In fact, by Theorem 5.8, $\alpha(\mathrm{Inv}[S, In](K)) \leq_A \mathrm{Inv}^A[S, \alpha(In)](K)$ and this implies $\mathrm{Inv}[S, In](K) \subseteq \gamma(\mathrm{Inv}^A[S, \alpha(In)](K)) \subseteq \mathrm{inv}(K)$.

If concrete traces are complete and inv is A-representable, abstract safety verification is sound and complete, namely:

Theorem 5.15 (**Sound and Complete Safety Verification**). *If* $\mathrm{Trace}[S, In]$ *is complete for* A, inv *is* A*-representable and* $\tau_A[S, \alpha(In)]$ *is finite then*

$$[S, In] \models \mathrm{inv} \;\Leftrightarrow\; [S, In] \models^A \mathrm{inv}\,.$$

Proof. By Theorem 5.13, $\mathrm{Inv}^A[S, \alpha(In)](K) = \alpha(\mathrm{Inv}[S, In](K)) \leq_A \alpha(\mathrm{inv}(K))$ so that we obtain $\gamma(\mathrm{Inv}^A[S, \alpha(In)](K)) \subseteq \gamma(\alpha(\mathrm{inv}(K))) = \mathrm{inv}(K)$.

□

5.6 Nondeterministic Interpreter

The nondeterministic transition relation $\rightarrow^{\mathrm{nd}} \subseteq \mathrm{State}_{\rightarrow_{\mathrm{pc}}} \times \mathrm{State}_{\rightarrow_{\mathrm{pc}}}$ of the parallel interpreter adds the following rule to the rules in Figure 1:

$$\frac{\mathcal{P} \not\subseteq \mathcal{P}_{\textbf{while}}}{\langle [(\textbf{while}\ B\ \textbf{do}\ S)K, \mathcal{P}_{\textbf{while}}]_w \cdot \Sigma, \epsilon, \mathcal{P}\rangle \rightarrow^{\mathrm{nd}} \langle \Sigma, K, [\![\neg B]\!](\mathcal{P}_{\textbf{while}} \cup \mathcal{P})\rangle}$$

Thus, each time the evaluation of the body of a while-loop (**while** B **do** $S)K$ terminates with a concrete invariant $\mathcal{P}$ which is not a fixpoint (namely, if $\mathcal{P}_{\textbf{while}}$ is the current abstract loop invariant then $\mathcal{P} \not\subseteq \mathcal{P}_{\textbf{while}}$), we have a non-deterministic branching: on the one hand, the computation of the least concrete loop invariant may go on, and, on the other hand, the computation may proceed with the loop-continuation K by assuming the current loop invariant $\mathcal{P}_{\textbf{while}} \cup \mathcal{P}$, so that K is evaluated from the abstract exit condition $[\![\neg B]\!](\mathcal{P}_{\textbf{while}} \cup \mathcal{P})$.
In a sense, this gives rise to a form of dovetailing which allows a breadth-first search in a tree of concrete traces, which may potentially contain a path of infinite length. This corresponds to make the abstract interpreter 'fair', because all the concrete traces of this non-deterministic interpreter become fair in the sense that if the interpreter enters in a state infinitely often, it also takes every possible transition from that state.

This rule appears to be related to the hypercollecting semantics of while-loops in [1, Section 4], since this returns a set $\mathcal{R}$ of sets of stores where any set $R \in \mathcal{R}$ contains those stores that exit the loop in less than k iterations, for some $k \in \mathbb{N}$.

The function $\mathrm{Inv}^{\mathrm{nd}}[S, In] : \mathcal{C}(S) \to \wp(\mathrm{Store})$, for any continuation $K \in \mathcal{C}(S)$, returns the (strongest) concrete invariant at K for the non-deterministic collecting interpreter:

$$\mathrm{Inv}^{\mathrm{nd}}[S, In](K) = \bigcup\{\mathcal{Q}' \in \wp(\mathrm{Store}) \mid \langle \epsilon, S, \mathcal{Q}\rangle \rightarrow^{\mathrm{nd}*} \langle \Sigma, S', \mathcal{Q}'\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K\}.$$

In the same way, it is possible to define the non-deterministic abstract transition relation $\rightarrow^{nd}_A \subseteq \mathrm{State}_{\rightarrow_A} \times \mathrm{State}_{\rightarrow_A}$, obtained by adding the following rule to Figure 3:

$$\frac{a \not\leq_A a_{\textbf{while}}}{\langle [(\textbf{while}\ B\ \textbf{do}\ S)K, a_{\textbf{while}}]_w \cdot \Sigma, \epsilon, a\rangle \rightarrow^{\mathrm{nd}}_A \langle \Sigma, K, [\![\neg B]\!]^A(a_{\textbf{while}} \vee a)\rangle}$$

Also, for every continuation $K \in \mathcal{C}(S)$, it is possible to define the function $\mathrm{Inv}^{nd}_A[S,a] : \mathcal{C}(S) \to A$, which returns the strongest abstract invariant at K for the non-deterministic abstract interpreter:

$$\mathrm{Inv}^{nd}_A[S,a] = \bigvee\{a' \in A \,|\, \langle \epsilon, S, a\rangle \to^{nd*}_A \langle \Sigma, S', a'\rangle,\ S'\,\mathrm{Cont}(\Sigma) \equiv K\}.$$

Fairness implies soundness, that is the following result.

Theorem 5.16 (**Soundness**). *Let* $S \in \mathrm{Stm}$ *and* $In \in \wp(\mathrm{Store})$. *For any* $K \in \mathcal{C}(S)$,

$$\alpha(\mathrm{Inv}^{\mathrm{nd}}[S, In](K)) \leq_A \mathrm{Inv}^{\mathrm{nd}}_A[S, \alpha(In)](K).$$

Proof. The strategy of the proof is the same of Theorem 5.8. We show the thesis in the cases $\mathcal{C}(S)$ without elements starting for 'while' and $S \equiv S_1(\textbf{while}\ B\ \textbf{do}\ T)S_2$ with $S_1,\ S_2,\ T$ sound. The general case is provided through the scheme of the second part of Theorem 5.8 proof.
For the sake of clarity, let us use the following notations:

$$\mathrm{Inv}^{nd}[S, In](K) = \bigcup\{\mathcal{Q} \,|\, In, K\}^{nd} \qquad \mathrm{Inv}^{nd}_A[S, \alpha(In)](K) = \bigvee\{a' \,|\, \alpha(In), K\}^{nd}.$$

First of all, observe that the non-deterministic interpreter acts as the parallel one, except for the additional contraction while-rule.
By this fact, the cases in which $\mathcal{C}(S)$ has no elements starting for 'while', $S \equiv S_1(\textbf{while}\ B\ \textbf{do}\ T)S_2$ with $S_1,\ T,\ S_2$ sound, and K fixed inside S_1 or inside the loop body T, follows directly from Theorem 5.8.

The only relevant case is $K \in \mathcal{C}(S_2)$, i.e. K fixed inside S_2.
In this case, fix $\mathcal{Q} \in \{\mathcal{Q} \,|\, In, K\}^{\mathrm{nd}}$. We want to show that there exists $a' \in \{a \,|\, \alpha(In), K\}^{nd}$ such that $\alpha(\mathcal{Q}) \leq_A a'$.
This is enough to prove the thesis at K.

Notice that the continuation K is reached either if the while-loop does terminate or not, since, at every iteration, there is a store property which exits from the loop; in other words, using notations of Definition 4.10, at the i^{th} iteration the continuation S_2 is reached by $[\![\neg B]\!]\mathcal{Q}_i$, and consequently K is reached by $[\![S']\!][\![\neg B]\!]\mathcal{Q}_i$, with $S' \subseteq S_2$.
Moreover, by Theorem 5.8, there exists $a_i \in \{a' \,|\, \alpha(In), (\textbf{while}\ B\ \textbf{do}\ T)S_2\}$ such that $\alpha(\mathcal{Q}_i) \leq_A a_i$.

Assume now that $\mathcal{Q} = [\![S']\!][\![\neg B]\!]\mathcal{Q}_n$, i.e. $\mathcal{Q}$ is reached by the store property exiting at the n^{th} iteration.
By the previous observation, there exists $a_n \in \{a' \,|\, \alpha(In), (\textbf{while}\ B\ \textbf{do}\ T)S_2\}$ such that $\alpha(\mathcal{Q}_n) \leq_A a_n$. Hence

$$\alpha([\![\neg B]\!]\mathcal{Q}_n) \underset{\text{G.C.}}{\leq_A} \alpha([\![\neg B]\!]\gamma\alpha(\mathcal{Q}_n)) \underset{\text{b.c.a}}{=}$$
$$[\![\neg B]\!]^A\alpha(\mathcal{Q}_n) \leq_A [\![\neg B]\!]^A a_n.$$

Moreover, $S' \subseteq S_2$ and S_2 is sound. Hence

$$\alpha(\mathcal{Q}) = \alpha([\![S']\!][\![\neg B]\!]\mathcal{Q}_n) \leq_A [\![S']\!]^A[\![\neg B]\!]^A a_n$$

and the theorem is proved.

□

Example 5.17. Consider the program P in Example 5.2, where the abstract domain A is given by the interval abstraction. If we consider the non-deterministic collecting and abstract interpreter then we have that

$$\begin{aligned}\mathrm{Inv}^{\mathrm{nd}}[P,\{N\}] = \{① \mapsto \{N\}, ② \mapsto \{x/1, x/3, x/5\}, ③ \mapsto \{x/3, x/5\},\\ ④ \mapsto \emptyset, ⑤ \mapsto \{x/3, x/5\}, ⑥ \mapsto \{x/1\}\}\end{aligned}$$

$$\begin{aligned}
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]① = [N,N]\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]② = [1,+\infty] = \bigvee\{[5,5],[3,5],[1,7],[1,9],[1,11],...\}\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]③ = [2,+\infty] = \bigvee\{[5,5],[3,5],[2,7],[2,9],[2,11],...\}\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]④ = [2,+\infty] = \bigvee\{[4,4],[2,6],[2,8],[2,10],...\}\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]⑤ = [3,+\infty] = \bigvee\{[5,5],[3,5],[3,7],[3,9],...\}\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]⑥ = [1,1]
\end{aligned}$$

□

Example 5.18. Consider the program P in Example 5.3, where the abstract invariant at program point ⓟ for the interval abstraction was not sound. If we consider the non-deterministic abstract interpreter, we have that

$$\begin{aligned}
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]ⓠ = \bigvee(\{[5,5],[3,5]\} \cup \{[1,7+2n] \mid n \in \mathbb{N}\}) = [1,+\infty]\\
&\mathrm{Inv}^{\mathrm{nd}}_A[P,\{x/[N,N]\}]ⓟ = \bigvee\{[5,5],[3,5],[1,5]\} = [1,5]
\end{aligned}$$

so that the soundness at program point ⓟ has been restored. □

6 An equivalence result

In previous chapters, we studied the relation between collecting and abstract parallel interpreters. We saw that soundness might not hold if infinite abstract traces are there, and solved this issue by defining the non-deterministic interpreters.

In Theorem 5.16 we proved that soundness holds for non-deterministic interpreters, and this lets us hope for having equivalence with denotational collecting and abstract interpreters.
In fact, this is what we are showing. Previously, let us formally define operational and denotational semantic functions.

Definition 6.1. The operational non-deterministic semantic function is defined inductively:

$$\mathcal{S}^c_o : \mathrm{Stm} \to (\wp(\mathrm{Store}) \to \wp(\mathrm{Store}))$$

where

$$\mathcal{S}^c_o[\![S]\!]\mathcal{Q} = \begin{cases} \mathcal{Q} & \text{if } S = \epsilon; \\ \mathcal{S}^c_o[\![K]\!]I & \text{if } S = cK,\ c \in \mathrm{Cmd},\ I = \mathrm{Inv}^{nd}[cK, \mathcal{Q}](K) \end{cases}$$

Moreover,

Definition 6.2. The denotational semantic function is defined inductively

$$\mathcal{S}^c_{ds} : \mathrm{Stm} \to (\wp(\mathrm{Store}) \to \wp(\mathrm{Store})), \quad S \mapsto [\![S]\!],$$

where

$$\begin{aligned}
&[\![\epsilon]\!]\mathcal{Q} = \mathcal{Q} \\
&[\![\textbf{skip}; K]\!]\mathcal{Q} = [\![K]\!]\mathcal{Q} \\
&[\![x := E; K]\!]\mathcal{Q} = [\![K]\!]\{\rho[x \mapsto v] \mid \rho \in \mathcal{Q},\ v = \mathbb{E}[\![E]\!]\} \\
&[\![(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K]\!]\mathcal{Q} = [\![K]\!]([\![S_1]\!][\![B]\!]\mathcal{Q} \cup [\![S_2]\!][\![B]\!]\mathcal{Q}) \\
&[\![(\textbf{while } B \textbf{ do } S)K]\!] = [\![K]\!]([\![\neg B]\!]\, lfp(\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T).
\end{aligned}$$

Theorem 6.3. *Let* $S \in \mathrm{Stm}$ *. For all* $\mathcal{Q} \in \wp(\mathrm{Store})$ *it holds*

$$\mathcal{S}^c_o[\![S]\!]\mathcal{Q} = \mathcal{S}^c_{ds}[\![S]\!]\mathcal{Q}.$$

Proof. By induction on $S \in \mathrm{Stm}$.

- $S \equiv \epsilon$. Then $\mathcal{S}^c_o[\![\epsilon]\!]\mathcal{Q} = \mathcal{Q} = [\![\epsilon]\!]\mathcal{Q} = \mathcal{S}^c_{ds}[\![\epsilon]\!]$;
- $S \equiv cK$, $c \in \mathrm{Cmd}$. Here $\mathcal{S}^c_o[\![S]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!]I$ where $I = \mathrm{Inv}[cK, \mathcal{Q}](K)$.

– $c \equiv$ **skip**; . We have $\mathrm{Inv}[\textbf{skip}; K, \mathcal{Q}](K) = \mathcal{Q}$ because $\langle \epsilon, \textbf{skip}; K, \mathcal{Q}\rangle \rightarrow_{nd} \langle \epsilon, K, \mathcal{Q}\rangle$.
Hence $\mathcal{S}^c_o[\![\textbf{skip}; K]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!]\mathcal{Q}$.
On the other hand, $\mathcal{S}^c_{ds}[\![\textbf{skip}; K]\!]\mathcal{Q} = [\![\textbf{skip}; K]\!]\mathcal{Q} = [\![K]\!]\mathcal{Q}$.
By structural induction $\mathcal{S}^c_o[\![K]\!]\mathcal{Q} = \mathcal{S}^c_{ds}[\![K]\!]\mathcal{Q}$, and hence

$$\mathcal{S}^c_o[\![\textbf{skip}; K]\!]\mathcal{Q} = \mathcal{S}^c_{ds}[\![\textbf{skip}; K]\!]\mathcal{Q}.$$

– $c \equiv x := E;$. Here $\mathrm{Inv}[x := E; K, \mathcal{Q}](K) = \{\rho[x \mapsto v] \,|\, \rho \in \mathcal{Q},\, v = \mathbf{E}[\![E]\!]\rho\}$.
Hence

$$\mathcal{S}^c_o[\![x := E; K]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!]\{\rho[x \mapsto v \,|\, \rho \in \mathcal{Q},\, v = \mathbf{E}[\![E]\!]\rho\}.$$

Moreover $\mathcal{S}^c_{ds}[\![x := E; K]\!]\mathcal{Q} = \mathcal{S}^c_{ds}[\![K]\!]\{\rho[x \mapsto v \,|\, \rho \in \mathcal{Q},\, v = \mathbf{E}[\![E]\!]\rho\}$, and the conclusion follows by structural induction.

– $c \equiv$ **if** B **then** S_1 **else** S_2. In this case it may be difficult to compute directly $\mathrm{Inv}[(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{Q}](K)$.
Although, observe that

$$\begin{aligned}&\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, \mathcal{Q}\rangle \rightarrow_{nd} \langle [K]_t, S_1, [\![B]\!]\mathcal{Q}\rangle || \langle [K]_e, S_2, [\![\neg B]\!]\mathcal{Q}\rangle \\ &\rightarrow^*_{nd} \langle [K]_t, \epsilon, I_1\rangle || \langle [K]_e, \epsilon, I_2\rangle \rightarrow_{nd} \langle \epsilon, K, I_1 \cup I_2\rangle\end{aligned}$$

where

$$I_1 = \mathcal{S}^c_o[\![S_1]\!]([\![B]\!]\mathcal{Q}) \quad \text{and} \quad I_2 = \mathcal{S}^c_o[\![S_2]\!]([\![\neg B]\!]\mathcal{Q}).$$

Hence

$$\mathcal{S}^c_o[\![(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!](I_1 \cup I_2).$$

On the other hand

$$[\![(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K]\!]\mathcal{Q} = [\![K]\!]([\![S_1]\!][\![B]\!]\mathcal{Q} \cup [\![S_2]\!][\![\neg B]\!]\mathcal{Q}).$$

Observe now that, by induction, $\mathcal{S}^c_o[\![S_1]\!]([\![B]\!]\mathcal{Q}) = \mathcal{S}^c_{ds}[\![S_1]\!]([\![B]\!]\mathcal{Q})$; i.e.

$$\begin{aligned}&I_1 = [\![S_1]\!][\![B]\!]\mathcal{Q}, \quad \text{and in the same way} \\ &I_2 = [\![S_2]\!][\![\neg B]\!]\mathcal{Q}.\end{aligned}$$

Now, the conclusion follows by structural induction, since

$$\mathcal{S}^c_o[\![K]\!](I_1 \cup I_2) = [\![K]\!](I_1 \cup I_2) = \mathcal{S}^c_{ds}[\![(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K]\!].$$

– $c \equiv$ **while** B **do** S. Recall the sequence of elements in $\wp(\mathrm{Store})$ of Definition 4.10:
$\begin{cases}\mathcal{Q}_0 = \mathcal{Q} \\ \mathcal{Q}_i = \mathcal{Q}_{i-1} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1}.\end{cases}$
The following lemma holds:

Lemma 6.4. *With the notation above, for all* $i \geq 1$

$$\mathcal{Q}_i = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1}.$$

Proof. By induction on i:
$i = 1$. It is trivial, since $\mathcal{Q}_1 = \mathcal{Q}_0 \cup [\![S]\!][\![B]\!]\mathcal{Q}_0$ by definition.
Assume $\mathcal{Q}_i = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1}$. Now:

$$\mathcal{Q}_{i+1} = \mathcal{Q}_i \cup [\![S]\!][\![B]\!]\mathcal{Q}_i \underset{\text{inductive hp}}{=} \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1} \cup [\![S]\!][\![B]\!]\mathcal{Q}_i$$

Observe that $[\![S]\!][\![B]\!]\mathcal{Q}_i = [\![S]\!][\![B]\!](\mathcal{Q}_{i-1} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1})$ and $\mathcal{Q}_{i-1} \subseteq \mathcal{Q}_{i-1} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1}$.
Hence, by monotonicity of transfer functions

$$[\![S]\!][\![B]\!]\mathcal{Q}_{i-1} \subseteq [\![S]\!][\![B]\!]\mathcal{Q}_i$$

which implies

$$\mathcal{Q}_{i+1} = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1} \cup [\![S]\!][\![B]\!]\mathcal{Q}_i = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_i.$$

□

We have to distinguish two cases.
Assume there exists $n \in \mathbb{N}$ such that $[\![S]\!][\![B]\!]\mathcal{Q}_n \subseteq \mathcal{Q}_n$ (i.e. the loop terminates), and assume $\mathcal{Q}_n$ is the smallest element of $\wp(\mathrm{Store})$ containing $\mathcal{Q}$, with this property. In particular $[\![S]\!][\![B]\!]\mathcal{Q}_j \not\subseteq \mathcal{Q}_j,\ \forall j \lneq n$.
Our claim is to show that

$$I = \mathrm{Inv}[(\textbf{while } B \textbf{ do } S)K, \mathcal{Q}](K) = [\![\neg B]\!]\mathcal{Q}_n.$$

In fact:

$$\begin{aligned}
&\mathrm{Inv}[(\textbf{while } B \textbf{ do } S)K, \mathcal{Q}](K) = \\
&\bigcup \{\mathcal{P} \mid \langle \epsilon, (\textbf{while } B \textbf{ do } S)K, \mathcal{Q}\rangle \to^*_{nd} \langle \Sigma, S', \mathcal{P}\rangle,\ S' \,\mathrm{Cont}(\Sigma) \equiv K\} = \\
&\bigcup \{[\![\neg B]\!]\mathcal{Q}_j \mid 1 \leq j \leq n\}.
\end{aligned}$$

Now

$$\begin{aligned}
&\mathcal{Q}_j \subseteq \mathcal{Q}_n \,\forall\, 1 \leq j \leq n \Rightarrow [\![\neg B]\!]\mathcal{Q}_j \subseteq [\![\neg B]\!]\mathcal{Q}_n \,\forall\, 1 \leq j \leq n \\
&\text{hence } \bigcup \{[\![\neg B]\!]\mathcal{Q}_j \mid 1 \leq j \leq n\} \subseteq [\![\neg B]\!]\mathcal{Q}_n\,; \\
&\text{moreover } [\![\neg B]\!]\mathcal{Q}_n \subseteq \bigcup \{[\![\neg B]\!]\mathcal{Q}_j \mid 1 \leq j \leq n\}.
\end{aligned}$$

Hence the equality holds. This shows that

$$\mathcal{S}^c_o[\![(\textbf{while } B \textbf{ do } S)K]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!][\![\neg B]\!]\mathcal{Q}_n \underset{\text{induction}}{=} \mathcal{S}^c_{ds}[\![K]\!]([\![\neg B]\!]\mathcal{Q}_n).$$

Recall that $\mathcal{S}^c_{ds}[\![(\textbf{while } B \textbf{ do } S)K]\!]\mathcal{Q} = [\![K]\!]([\![\neg B]\!]\ lfp(\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T))$.
Hence it is enough to show that

$$\mathcal{Q}_n = lfp(\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T).$$

By Lemma 6.4 it holds $\mathcal{Q}_n = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{n-1}$. Moreover

$$\mathcal{Q}_n = \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{n-1} \subseteq \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_n \subseteq \mathcal{Q}_n$$

which shows that $\mathcal{Q}_n$ is a fixed point for $\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T$.
It is obviously the least one, since, if there exists $X \subsetneq \mathcal{Q}_n$ such that $\mathcal{Q} \cup [\![S]\!][\![B]\!]X = X$, then $[\![S]\!][\![B]\!]X \subseteq X$: against the assumption on $\mathcal{Q}_n$.

Assume now that $\forall n \in \mathbb{N}\,.\, [\![S]\!][\![B]\!]\mathcal{Q}_n \not\subseteq \mathcal{Q}_n$ (i.e. the loop does not terminate). As in the previous case

$$\mathrm{Inv}[(\textbf{while } B \textbf{ do } S)K, \mathcal{Q}](K) = \bigcup_{j\geq 1} [\![\neg B]\!]\mathcal{Q}_j = [\![\neg B]\!](\bigcup_{j\geq 1} \mathcal{Q}_j).$$

Hence

$$\mathcal{S}^c_o[\![(\textbf{while } B \textbf{ do } S)K]\!]\mathcal{Q} = \mathcal{S}^c_o[\![K]\!]([\![\neg B]\!]\bigcup_{j\geq 1} \mathcal{Q}_j) \underset{\text{induction}}{=} \mathcal{S}^c_{ds}[\![K]\!]([\![\neg B]\!]\bigcup_{j\geq 1} \mathcal{Q}_j).$$

Again $\mathcal{S}^c_{ds}[\![(\textbf{while } B \textbf{ do } S)K]\!]\mathcal{Q} = [\![K]\!]([\![\neg B]\!]lfp(\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T))$.
Hence, it is enough to show that

$$\bigcup_{j\geq 1} \mathcal{Q}_j = lfp\,(\lambda T.\mathcal{Q} \cup [\![S]\!][\![B]\!]T).$$

Let $F : \wp(\mathrm{Store}) \to \wp(Store), \quad X \mapsto \mathcal{Q} \cup [\![S]\!][\![B]\!]X$.
It is well known that, by Knaster-Tarski theorem,

$$lfp\,F = \bigcup_{i\geq 0} F^i(\emptyset).$$

Lemma 6.5. *With the notation above*

$$\mathcal{Q}_i = F^{i+1}(\emptyset).$$

Proof. By induction on i.
The case $i = 0$ is true by definition, since

$$\mathcal{Q}_0 = \mathcal{Q} \qquad F(\emptyset) = \mathcal{Q} \cup \emptyset = \mathcal{Q}.$$

Assume $\mathcal{Q}_{i-1} = F^i(\emptyset)$. Then

$$\mathcal{Q}_i \underset{\text{lemma 6.4}}{=} \mathcal{Q} \cup [\![S]\!][\![B]\!]\mathcal{Q}_{i-1} =$$
$$\mathcal{Q} \cup [\![S]\!][\![B]\!]F^i(\emptyset) = F(F^i(\emptyset)) = F^{i+1}(\emptyset).$$

□

Hence

$$\bigcup_{j\geq 1} \mathcal{Q}_j = \bigcup_{j\geq 0} \mathcal{Q}_j = \bigcup_{i\geq 0} F^i(\emptyset) = lfp\, F.$$

□

Next, we want to show that the same equivalence holds between non deterministic and denotational abstract interpreters.
We begin with the definitions of abstract operational semantic function and abstract denotational semantic function.

Definition 6.6. The abstract operational non-deterministic sematic function is defined inductively:

$$\mathcal{S}_o^\sharp : \mathrm{Stm} \to (A \to A)$$

where

$$\mathcal{S}_o^\sharp[\![S]\!]a = \begin{cases} a & \text{if } S = \epsilon \\ \mathcal{S}_o^\sharp[\![K]\!]a' & \text{if } a' = \mathrm{Inv}_A^{nd}[cK, a](K) \end{cases}$$

Definition 6.7. The abstract denotational semantic is defined inductively:

$$\mathcal{S}_{ds}^\sharp : \mathrm{Stm} \to (A \to A), \quad S \mapsto [\![S]\!]^\sharp$$

where

$$[\![\epsilon]\!]^\sharp a = a;$$
$$[\![\textbf{skip}; K]\!]^\sharp a = [\![K]\!]^\sharp a;$$
$$[\![x := E; K]\!]^\sharp a = [\![K]\!]^\sharp([\![x := E]\!]^\sharp a);$$
$$[\![(\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K]\!]^\sharp a = [\![K]\!]^\sharp([\![S_1]\!]^\sharp[\![B]\!]^\sharp a \vee [\![S_2]\!]^\sharp[\![B]\!]^\sharp a);$$
$$[\![(\textbf{while } B \textbf{ do } S)K]\!]^\sharp = [\![K]\!]^\sharp([\![\neg B]\!]^\sharp lfp(\lambda c. \vee [\![S]\!]^\sharp[\![B]\!]^\sharp c).$$

As for concrete interpreters, the following theorem holds:

Theorem 6.8. *Let* $S \in \mathrm{Stm}$. *For all* $a \in A$ *it holds:*

$$\mathcal{S}^\sharp_o[\![S]\!]a = \mathcal{S}^\sharp_{ds}[\![S]\!]a.$$

Proof. The proof is simply obtained from the proof of Theorem 6.3 replacing collecting elements with abstract ones.
The reasoning is exactly the same, since the definitions of the interpreters (concrete and abstract) are strictly related. □

Now, the equivalence has been established for concrete and abstract interpreters. Our next claim is to relate the invariants soundness studied in Theorems 5.8, 5.16 and the semantic functions above.

In turn, look at the scales: on the one side there is the notion of invariant $\mathrm{Inv}[S, In](K)$, and in the other, the semantic function $\mathcal{S}^c_o[\![S]\!]$.
It turns out that, for $S \in \mathrm{Stm},\ In \in \wp(\mathrm{Store})$,

$$\mathcal{S}^c_o[\![S]\!]In = \mathrm{Inv}^{nd}[S, In](\epsilon).$$

In fact, we can act by structural induction on $S \in \mathrm{Stm}$:
If $S \equiv \epsilon$ then $\mathcal{S}^c_o[\![\epsilon]\!]In = In$ and

$$\mathrm{Inv}^{nd}[\epsilon, In](\epsilon) = \bigcup\{\mathcal{Q} \mid \langle \epsilon, \epsilon, In\rangle \rightarrow^{nd^*}_{\mathrm{pc}} \langle \epsilon, \epsilon, \mathcal{Q}\rangle\} = In.$$

If $S \equiv cK$, $c \in \mathrm{Cmd}$ then

$$\mathcal{S}^c_o[\![cK]\!]In \underset{\text{def}}{=} \mathcal{S}^c_o[\![K]\!]\,\mathrm{Inv}^{nd}[cK, In](K) \underset{\text{induction}}{=} \mathrm{Inv}^{nd}[K, \mathrm{Inv}^{nd}[cK, In](K)](\epsilon).$$

This has to coincide with $\mathrm{Inv}^{nd}[cK, In](\epsilon)$. For the sake of clarity let $\mathrm{Inv}^{nd}[cK, In](K) = I$. Let us prove the double inclusion.
We need a preliminary

Lemma 6.9. *The set* $\{\mathcal{Q} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \Sigma, S', \mathcal{Q}\rangle,\ \ S'\,\mathrm{Cont}(\Sigma) \equiv K\}$ *is a filtering set, i.e., it is equal to* $\{\mathcal{Q}^0, \mathcal{Q}^1, ..., \mathcal{Q}^n, ...\}$, *with* $\mathcal{Q}^i \subseteq \mathcal{Q}^{i+1}$ *for every* $i \geq 0$.

Proof. By structural induction on $c \in \mathrm{Cmd}$.

- $c \equiv$ **skip;**, $c \equiv x := E$;. Then $\{\mathcal{Q} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \Sigma, S', \mathcal{Q}\rangle,\ \ S'\,\mathrm{Cont}(\Sigma) \equiv K\}$ is a singleton, and it is trivially a filtering set.

- $c \equiv$ **if** B **then** S_1 **else** S_2. Then

$$\langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, In\rangle \rightarrow^{nd} \langle [K]_t, S_1, [\![B]\!]In\rangle \parallel \langle [K]_e, S_2, [\![\neg B]\!]In\rangle$$
$$\rightarrow^{nd^*} \langle [K]_t, \epsilon, \mathcal{Q}_t\rangle \parallel \langle [K]_e, \epsilon, \mathcal{Q}_e\rangle \rightarrow^{nd} \langle \epsilon, K, \mathcal{Q}_t \cup \mathcal{Q}_e\rangle$$

  By induction, the sets

$$\{\mathcal{Q}_t \mid \langle \epsilon, S_1K, [\![B]\!]In\rangle \equiv \langle [K]_t, S_1, [\![B]\!]In\rangle \rightarrow^{nd^*} \langle [K]_t, \epsilon, \mathcal{Q}_t\rangle \equiv \langle \epsilon, K, \mathcal{Q}_t\rangle\}$$
$$\{\mathcal{Q}_e \mid \langle \epsilon, S_2K, [\![\neg B]\!]In\rangle \equiv \langle [K]_e, S_2, [\![\neg B]\!]In\rangle \rightarrow^{nd^*} \langle [K]_e, \epsilon, \mathcal{Q}_e\rangle \equiv \langle \epsilon, K, \mathcal{Q}_e\rangle\}$$

  are filtering sets, and hence

$$\{\mathcal{Q}_t \cup \mathcal{Q}_e \mid \langle \epsilon, (\textbf{if } B \textbf{ then } S_1 \textbf{ else } S_2)K, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, \mathcal{Q}_t \cup \mathcal{Q}_e\rangle\}$$

  is a filtering set too.

- $c \equiv$ **while** B **do** T. Let $\begin{cases} \mathcal{P}_0 = In \\ \mathcal{P}_n = \mathcal{P}_{n-1} \cup [\![T]\!][\![B]\!]\mathcal{P}_{n-1} \end{cases}$

  Then

$$\langle \epsilon, (\textbf{while } B \textbf{ do } T)K, In\rangle \rightarrow^{nd^*} \langle [(\textbf{while } B \textbf{ do } T)K, In]_w, T, [\![B]\!]In\rangle$$
$$\rightarrow^{nd^*} \langle [(\textbf{while } B \textbf{ do } T)K, In]_w, \epsilon, [\![T]\!][\![B]\!]In\rangle \rightarrow^{nd} \begin{cases} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_1]_w, T, [\![B]\!]\mathcal{P}_1\rangle \\ \langle \epsilon, K, [\![\neg B]\!]\mathcal{P}_1\rangle \end{cases}$$

  The upper branch goes on with

$$\rightarrow^{nd^*} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_1]_w, \epsilon, [\![T]\!][\![B]\!]\mathcal{P}_1\rangle \rightarrow^{nd} \begin{cases} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_2]_w, T, [\![B]\!]\mathcal{P}_2\rangle \\ \langle \epsilon, K, [\![\neg B]\!]\mathcal{P}_2\rangle \end{cases}$$

  Following the upper branch again $\rightarrow^{nd^*} \quad ... \quad \rightarrow^{nd^*}$

$$\rightarrow^{nd^*} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_n]_w, \epsilon, [\![T]\!][\![B]\!]\mathcal{P}_n\rangle \rightarrow^{nd}$$
$$\begin{cases} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_{n+1}]_w, T, [\![B]\!]\mathcal{P}_{n+1}\rangle \\ \langle \epsilon, K, [\![\neg B]\!]\mathcal{P}_n\rangle \end{cases} \rightarrow^{nd^*} ...$$

  At each iteration, the set

$$\{[\![T]\!][\![B]\!]\mathcal{P}_i \mid \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_i]_w, T, [\![B]\!]\mathcal{P}_i\rangle \rightarrow^{nd^*} \langle [(\textbf{while } B \textbf{ do } T)K, \mathcal{P}_i]_w, \epsilon, [\![T]\!][\![B]\!]\mathcal{P}_i\rangle\}$$

  is filtering by inductive hypothesis, and hence, by definition of $\mathcal{P}_i$, the set

$$\{[\![\neg B]\!]\mathcal{P}_i \mid \langle \epsilon, (\textbf{while } B \textbf{ do } T)K, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, [\![\neg B]\!]\mathcal{P}_i\rangle\}$$

is a filtering set too.
Notice that, if there exists $n \in \mathbb{N}$ such that $[\![T]\!][\![B]\!]\mathcal{P}_n \subseteq \mathcal{P}_n$, i.e. the loop terminates, then

$$\{[\![\neg B]\!]\mathcal{P}_i \mid \langle \epsilon, (\textbf{while } B \textbf{ do } T)K, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, [\![\neg B]\!]\mathcal{P}_i\rangle\} = \bigcup_{1 \leq i \leq n} [\![\neg B]\!]\mathcal{P}_i$$

otherwise, it is a numerable infinite union.

□

Let us come back to the proof of

$$\mathrm{Inv}^{nd}[K, \mathrm{Inv}^{nd}[cK, In](K)](\epsilon) = \mathrm{Inv}^{nd}[cK, In](\epsilon).$$

($\subseteq$) Let $\mathcal{Q} \in \{\mathcal{Q} \mid \langle \epsilon, K, I\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle\}$. Then, $\langle \epsilon, K, I\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle$.
Since

$$I = \mathrm{Inv}^{nd}[cK, In](K) = \bigcup\{\mathcal{P} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, \mathcal{P}\rangle\}$$

and, by the previous Lemma, this set is filtering, there exists $\mathcal{P} \in \{\mathcal{P} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, \mathcal{P}\rangle\}$ such that

$$\langle \epsilon, K, \mathcal{P}\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle.$$

Hence

$$\langle \epsilon, S, In\rangle \rightarrow^{nd^*} \langle \Sigma, S', \mathcal{P}\rangle \equiv \langle \epsilon, K, \mathcal{P}\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle,$$

i.e., $\mathcal{Q} \in \{\mathcal{Q} \mid \langle \epsilon, S, In\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle\}$. This, in particular, implies

$$\mathrm{Inv}^{nd}[K, \mathrm{Inv}^{nd}[cK, In](K)](\epsilon) \subseteq \mathrm{Inv}^{nd}[S, In](\epsilon).$$

($\supseteq$) Let $\mathcal{Q} \in \{\mathcal{Q} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle\}$. Then

$$\langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, \mathcal{P}\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle,$$

for some $\mathcal{P} \in \wp(\mathrm{Store})$. In particular,

$$\begin{aligned} \mathcal{P} \in \{\mathcal{P} \mid \langle \epsilon, cK, In\rangle \rightarrow^{nd^*} \langle \epsilon, K, \mathcal{P}\rangle\} &\implies \mathcal{P} \subseteq \mathrm{Inv}^{nd}[cK, In](K). \\ \text{and} \quad \mathcal{Q} \in \{\mathcal{Q} \mid \langle \epsilon, K, \mathcal{P}\rangle \rightarrow^{nd^*} \langle \epsilon, \epsilon, \mathcal{Q}\rangle\} &\implies \mathcal{Q} \subseteq \mathrm{Inv}^{nd}[K, \mathcal{P}](\epsilon). \end{aligned}$$

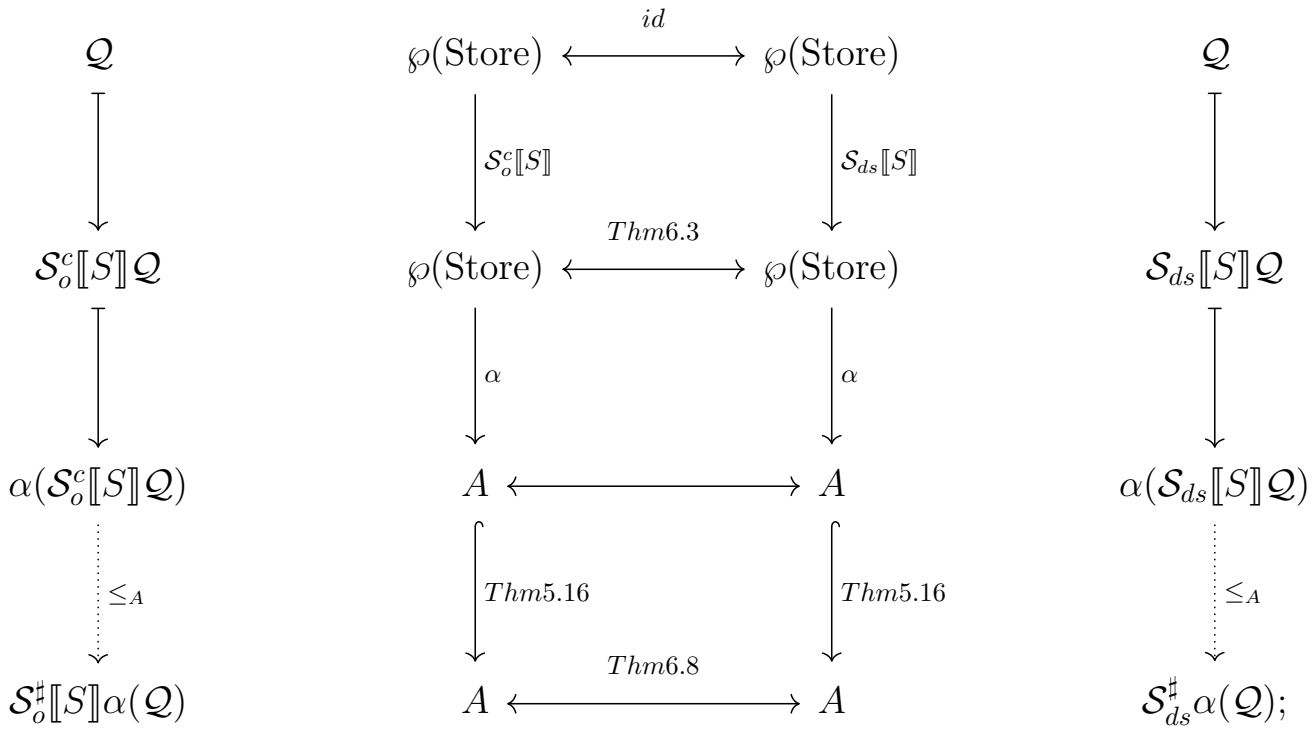
Hence, $\mathrm{Inv}^{nd}[K, \mathcal{P}](\epsilon) \subseteq \mathrm{Inv}^{nd}[K, \mathrm{Inv}^{nd}[cK, In](K)](\epsilon)$
and $\mathcal{Q} \subseteq \mathrm{Inv}^{nd}[K, \mathrm{Inv}^{nd}[cK, In](K)](\epsilon)$. Then, the equality is proved.

Obviously, the abstract version holds too, i.e., for $S \in \mathrm{Stm}, \ \ a \in A, , \mathcal{S}^{\sharp}_o[\![S]\!]a = \mathrm{Inv}^{nd}_A[S, a](\epsilon)$.

In particular, we can apply Theorem 5.16 deriving that

$$\alpha(\mathcal{S}^c_o[\![S]\!]\mathcal{Q}) = \alpha(\mathrm{Inv}^{nd}[S, \mathcal{Q}](\epsilon)) \leq_A \mathrm{Inv}^{nd}_A[S, \alpha(\mathcal{Q})](\epsilon) = \mathcal{S}^{\sharp}_o[\![S]\!]\alpha(\mathcal{Q})$$

In conclusion, given $S \in \mathrm{Stm}$, we can summarize all the previous paragraphs within the following diagram.



We conclude with an example, in which every step of the diagram is computed.
We consider the computation on two initial store properties, to show that the $\leq_A$ cannot be refined.

Example 6.10. Let us consider $A = \mathrm{Int}$ and $S \equiv$ **while** $x > 0$ **do** $x := x - 2;$.
Let $\mathcal{Q} \in \wp(\mathrm{Store})$, $\mathcal{Q} = \{x/0,\ x/3,\ x/4\}$;
Then $\alpha(\mathcal{Q}) = [0,\ 4]$. Let us compute $\mathcal{S}^c_o[\![S]\!]\mathcal{Q}$.

$\langle \epsilon, \textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{Q}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{Q}]_w, x := x - 2, \{x/3,\ x/4\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{Q}]_w, \epsilon, \{x/1,\ x/2\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \{x/0,\ x/1,\ x/2,\ x/3,\ x/4\}]_w, x := x - 2;, \{x/1,\ x/2,\ x/3,\ x/4\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \{x/0,\ x/1,\ x/2,\ x/3,\ x/4\}]_w, \epsilon, \{x/-1,\ x/0,\ x/1,\ x/2\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \{x/-1,\ x/0,\ x/1,\ x/2,\ x/3,\ x/4\}]_w,$

$x := x - 2;, \{x/1,\ x/2,\ x/3,\ x/4\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \{x/-1,\ x/0,\ x/1,\ x/2,\ x/3,\ x/4\}]_w, \epsilon, \{x/-1,\ x/0,\ x/1,\ x/2\}\rangle \to^{\mathrm{nd}}$

$\langle \epsilon, \epsilon, \{x/-1,\ x/0\}\rangle$

Notice that, since the while-loop terminates, the non-deterministic interpreter acts exactly as the parallel one.

We have obtained $\mathcal{S}^c_o[\![S]\!]\mathcal{Q} = \{x/-1,\ x/0\}$ and $\alpha(\ \mathcal{S}^c_o[\![S]\!]\mathcal{Q}\ ) = [-1,\ 0]$.
Now compute $[\![S]\!]^\sharp\alpha(\mathcal{Q})$.

$\langle \epsilon, \textbf{while } x > 0 \textbf{ do } x := x - 2;, [0,\ 4]\rangle \to^{\mathrm{nd}}_{\mathrm{A}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, [0, 4]]_w, x := x - 2;, [1, 4]\rangle \to^{\mathrm{nd}}_{\mathrm{A}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \ [0, 4]]_w, \epsilon, [-1, 2]\rangle \to^{\mathrm{nd}}_{\mathrm{A}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, [-1, 4]]_w, x := x - 2;, [1, 4]\rangle \to^{\mathrm{nd}}_{\mathrm{A}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, [-1, 4]]_w, \epsilon, [-1, 2]\rangle \to^{\mathrm{nd}}_{\mathrm{A}}$

$\langle \epsilon, \epsilon, [-1, 0]\rangle$.

It turns out that $[\![S]\!]^\sharp\alpha(\mathcal{Q}) = [-1, 0]$, and it coincides with $= \alpha(\mathcal{S}^c_o[\![S]\!]\mathcal{Q})$.

Now consider $\mathcal{P} = \{x/0,\ x/2,\ x/4\}$, and repeat the computation.

$\langle \epsilon, \textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{P}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{P}]_w, x := x - 2;, \{x/2, x/4\}\rangle \to^{\mathrm{nd}}$

$\langle [\textbf{while } x > 0 \textbf{ do } x := x - 2;, \mathcal{P}]_w, \epsilon, \{x/0,\ x/2\}\rangle \to^{\mathrm{nd}}$

$\langle \epsilon, \epsilon, \{x/0\}\rangle$,

hence $\mathcal{S}^c_o[\![S]\!]\mathcal{P} = \{x/0\}$.
Notice that $\alpha(\mathcal{P}) = \alpha(\mathcal{Q}) = [0, 4]$, hence $[\![S]\!]^\sharp\alpha(\mathcal{P}) = [-1, 0]$, and

$$\alpha(\mathcal{S}^c_o[\![S]\!]\mathcal{P}) = \alpha(\{x/0\}) = [0,\ 0] \lneq_A [-1,\ 0] = [\![S]\!]^\sharp\alpha(\mathcal{P}).$$

7 Conclusions and further work

This thesis firstly aims to show off a complete analysis of the operational approach to abstract interpretation. Starting from the definition of a standard continuation-style WHILE language and its operational semantics, we moved to the definition of the small step collecting interpreter and proofs of the strict correlation between the concrete and collecting interpreter.

After that, we considered a generic abstract domain A. Assuming to have a Galois connection, we defined the abstract operational interpreter and the abstract transfer functions as the best correct approximations of the concrete ones. We proved the *Soundness theorem*, which represents maybe the most meaningful result of this work. It provides a gauge to compare collecting and abstract invariants at every continuation of a given program. This point has certainly represented the toughest step of the thesis: the result had to be set on 'every continuation'; although, we had no tools to analyze the number of continuations of a generic program. Hence, we split up the problem: it turned out that the main issue was to analyze a particular case of program. The general case should have been an intuitive and obvious consequence of this particular case. Although this obviousness was as well tough to formalize and it is still hard to be confident to have considered the whole totality of programs. But after all, we are pretty sure we can accept this fact.
On the other hand, the second part of the *Soundness theorem* proof provides a logic scheme which has been used in several following proofs to make them accordant to that one.
Between them, we want to remark the *Soundness theorem for non-deterministic interpreter* proof. This interpreter is defined as adding a while-contraction rule to the parallel one, to guarantee a soundness result even with infinite abstract traces.
The parallel interpreter gets stuck in an infinite loop while the non-deterministic can keep on the computation: the added rule makes it 'fair' in the sense that, if it enters in a state infinitely often, it also takes every transition from that state through non-deterministic branches.

Such a behavior reminds us of the denotational interpreter: it computes invariants through fixed points and does not matter if there are non-terminating loops.
In fact, in the final chapter, we proved the equivalence between non-deterministic collecting interpreters and denotational ones, both in concrete and abstract versions.

We see several interesting avenues for further work on this topic. For instance, we may quote the one that actually was the originary goal of this thesis work: proving that the abstract operational interpreter is not a Universal Turing Machine, i.e., find a program whose interpretation on a coded input is different from the coding of its output on the same input data.

References

[1] M. Assaf, D.A. Naumann, J. Signoles, É. Totel, and F. Tronel. Hypercollecting semantics and its application to static analysis of information flow. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*. ACM, New York, NY, USA, pp. 874-887, 2017. DOI: https://doi.org/10.1145/3009837.3009889

[2] T. Ball. Formalizing Counterexample-Driven Refinement with Weakest Preconditions. In Broy et al. editors, *Engineering Theories of Software Intensive Systems: Proceedings of the NATO Advanced Study Institute on Engineering Theories of Software Intensive Systems*, Marktoberdorf, Germany, Springer, pp. 121-139, 2005.

[3] S. Dissegna, F. Logozzo, F. Ranzato. An Abstract Interpretation-based Model of Tracing Just-in-Time Compilation. In *ACM Transactions on Programming Languages and Systems*. January 2016, Article No.: 7. https://doi.org/10.1145/2853131

[4] M. Gordon and H. Collavizza. Forward with Hoare. In A.W. Roscoe et al. editors, *Reflections on the Work of C.A.R. Hoare*, Springer London, pp. 101-121, 2010.

[5] S. Guo and J. Palsberg. The essence of compiling with traces. In *Proceedings of the 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2011)*. ACM, New York, NY, USA, 563–574, 2011.

[6] N.D. Jones. *Computability and Complexity from a Programming Perspective*. The MIT Press, 1997.

[7] A.J. Kfoury, R.N. Moll and M.A. Arbib. *A Programming Approach to Computability*. Springer-Verlag, 1982.

[8] A. Miné. Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation. In *Foundations and Trends in Programming Languages*, Vol. 4, No. 3-4 (2017), pag. 120-372.

[9] H.R. Nielson and F. Nielson. Semantics with Applications: An Appetizer. *Undergraduate Topics in Computer Science.* Springer-Verlag London Limited, 2007.