On Multi-Language Abstraction*

Towards a Static Analysis of Multi-Language Programs

Abstract. Modern software development rarely takes place within a single programming language. Often, programmers appeal to *cross-language interoperability*. Examples are exploitation of novel features of one language within another, and cross-language code reuse. Previous works developed a theory of so-called *multi-languages*, which arise by *combining existing languages*, defining a precise notion of (algebraic) multi-language semantics. As regards static analysis, the heterogeneity of the multi-language context opens up new and unexplored scenarios. In this paper, we provide a general theory for the *combination of abstract interpretations* of existing languages, regardless of their inherent nature, in order to gain an abstract semantics of *multi-language programs*. As a part of this general theory, we show that formal properties of interest of multi-language abstractions (*e.g.*, *soundness* and *completeness*) boil down to the features of the interoperability mechanism that binds the underlying languages together. We extend many of the standard concepts of abstract interpretation to the framework of multi-languages.

Keywords: multi-languages · abstract interpretation · interoperability · algebraic semantics.

1 Introduction

There is currently a myriad of programming languages, many of which have extensive library support. With programs becoming larger and increasingly complex, *interoperability mechanisms* streamline program development by enabling the interplay between pieces of code written in different languages. Examples are *embedded interpreters* [37], consisting of a runtime engine implemented in the host language (such as Jython [26] that lets Java interoperate with Python), or the *foreign function interface* system that allows one language to call routines written in another (*e.g.*, the Java Native Interface [29] enables Java code to call C++ functions). On one hand these mechanisms are essential tools, but on the other hand they hamper our understanding of the resulting programs.

The *multi-language* framework of [7] provides a theoretical model to formalise cross-language interoperability from an abstract standpoint. Multi-languages arise from the combination of already existing languages [9,36,1,16,23,35,31]. Intuitively, terms of multi-languages are obtained by performing cross-language substitutions (*e.g.*, the multi-language designed in [31] allows programmers to replace ML expressions with Scheme expressions and vice versa) and the semantics is determined by new constructs able to regulate the flow of values between the underlying languages, the so-called *boundary functions* [31].

* Supported by organization x.

The Problem. Despite the wide range of frameworks for interoperability, there is a lack of techniques for combining *static analyses* of different languages. Static analysis consists of a range of well-established and widely used techniques for automatically extracting dynamic (*i.e.*, runtime) behaviours statically (i.e., without executing the code). When it comes to multi-languages, two new challenges need to be tackled. Firstly, single-language analysers are not conceived for inspecting external code, and secondly the combination of analyses is not straightforward, since the interoperability mechanism that blends the underlying languages adds a new semantic layer. For instance, consider the following Java code snippet analysed with SonarQube Scanner [8]:[1]

```
String hello = null;
String helloWorld = hello.concat(" World!"); // NullPointerException: hello is null
```

The analyser raises a warning of a null pointer exception at the second line. Instead, if we run the analyser on the next semantically equivalent but multi-language code the runtime exception goes unnoticed.

```
String hello = (String) js.eval("null"); // Evaluate "null" in JS and convert it back
String helloWorld = hello.concat(" World!"); // NullPointerException: hello is null
```

The method `eval` evaluates the JavaScript code `null` via the Nashorn engine (a JavaScript interpreter developed by Oracle and included in Java 8) and returns the equivalent Java value `null`. This trivial example underlines how easy it is to deceive an analysis when writing multi-language programs.

Of course, nothing prevents us from redesigning the abstract semantics of the multi-language from scratch and to implement the corresponding analyser. However, besides the obvious time-consuming task, we will end up without any theoretical properties of the abstraction (*e.g.*, soundness or completeness). In fact, what we would like to achieve is a framework that takes advantage, as far as possible, of the already existing abstractions of the underlying languages and at the same time provides theoretical results.

A General Solution. Abstract interpretation [12] has allowed a disparate collection of (practical) methods and algorithms proposed along the years for static analysis to evolve into a mature discipline, founded on a robust theoretical framework. This provides a good environment for designing static analysis methods within a language, semantics, and approximation independent way [13]. It has broad scope and wide applicability. Our aim is to retain such broad scope, but to work with multi-language programs: Instead of fixing two programming languages and combining their respective analyses, we model abstract interpetation itself, within the algebraic framework of multi-language semantics [7]. Such an approach allows us to lay down the first steps of a general technique for designing static analyses of multi-language programs, in a way that (1) is independent of both underlying languages and analyses and (2) preserves the design and properties of the single-language abstract semantics.

[1] A commercial static code analyser for Java (version `3.2.0.1227` for Linux 64 bit).

Contributions and Paper Structure. Our main contribution is a general technique for abstracting multi-language semantics given the interoperation of the underlying languages and of their abstract semantics. We exploit abstract interpretation theory [12] for retaining independency from the underlying analyses, and the algebraic framework of multi-languages [7] for generality of the blended languages. In Sect. 2, we provide background definitions for multi-languages. In Sect. 3, we give a general, algebraic, and fixpoint construction of the *collecting semantics*, namely the reference semantics for defining and proving the correctness of approximated properties. In Sect. 4, we instantiate the abstract interpretation-based semantics approximation in the algebraic framework, in order to fill the gap betweeen the algebraic approach to program semantics and static analysis. Finally, in Sect. 5, we combine all these concepts, obtaining an algebraic framework for modeling *abstract interpretation of multi-language programs*. The proofs are given in Appx. A. We assume familiarity with abstract interpretation theory.

2 The Multi-Language Framework

We summarise the framework of multi-languages [7] based on the theory of order-sorted algebras [20]. Intuitively, multi-languages result from the combination of two order-sorted specifications defining syntax and semantics of the underlying languages. Order-sorted algebras provide a simple and yet powerful framework for modelling formal systems as algebraic structure, and have been widely used for specifying and prototyping programming languages (see [19] for survey).

Sorted Sets and Functions. Let S be a **set of sorts**. An S**-sorted set** is a family of sets $A \triangleq (\, A_s \mid s \in S \,)$. Given two S-sorted sets A and B, an S**-sorted function** $h\colon A \to B$ is a family $(\, h_s\colon A_s \to B_s \mid s \in S \,)$ of set-theoretic functions. If $f\colon A \to B$ and $g\colon B \to C$ are two S-sorted functions, clearly their composition $g \circ f \triangleq (\, (g \circ f)_s \triangleq g_s \circ f_s \mid s \in S \,)$ is an S-sorted function from A to C. We extend set-theoretic operators and predicates componentwise. For instance, if A and B are two S-sorted sets, $A \subseteq B$ if $A_s \subseteq B_s$ for each $s \in S$, and we define the cartesian product $A \times B$ by taking each component $(A \times B)_s \triangleq A_s \times B_s$. If A is an S-sorted set and $w \triangleq s_1 \dots s_n \in S^*$, we denote by A_w the cartesian product $A_{s_1} \times \cdots \times A_{s_n}$ (when $w \triangleq \varepsilon$, then $A_w \triangleq \{\bullet\}$ is the one-point domain). Likewise, if f is an S-sorted function and $a_i \in A_{s_i}$ for $i \triangleq 1, \dots, n$, then the function $f_w\colon A_w \to B_w$ is defined by $f_w(a_1, \dots, a_n) \triangleq (f_{s_1}(a_1), \dots, f_{s_n}(a_n))$. Moreover, if S is partially ordered by $\leq$, then S^* inherits the pointwise order. Finally, we introduce the product operator $\times$ used in Sect. 3. Let A be a family of sets indexed by I. The product operator $\times\colon \prod_{i \in I} \wp(A_i) \to \wp\big(\prod_{i \in I} A_i\big)$ defines the mapping $(X_1, \dots, X_n) \mapsto X_1 \times \cdots \times X_n$.

2.1 Order-Sorted Algebras

A *signature* defines the symbols of the language (that is, the syntax), and an *algebra* provides them with a meaning.

$$\textsf{(Const)}\ \frac{-}{k\colon s}\ (k\colon s) \qquad \textsf{(Fun)}\ \frac{(\forall 1 \le i \le n)\ \ t_i\colon s_i}{f(t_1,\ldots,t_n)\colon s}\ (f\colon s_1,\ldots,s_n \to s) \qquad \textsf{(Sub)}\ \frac{t\colon s}{t\colon r}\ (s \le r)$$

Fig. 1. Well-formed terms generated by an order-sorted signature Sg.

Definition 1 (Order-Sorted Signature). *An* ***order-sorted signature*** Sg *is specified by*

- *a poset* $\langle S, \le \rangle$ *of* ***sorts****;*
- *a set of* ***function symbols*** $f\colon s_1,\ldots,s_n \to s$ *each with* ***arity*** $n \ge 1$ *and* $(w, s) \in S^* \times S$ *the* ***rank*** *of* f *where* $w \triangleq s_1 \ldots s_n$*;*
- *a set of* ***constants*** $k\colon s$*, each of a* unique ***rank*** s *(just a single sort); and*
- *a* ***monotonicity requirement*** *that whenever* $f\colon w_1 \to s$ *and* $f\colon w_2 \to r$ *with* $w_1 \le w_2$*, then* $s \le r$*.*

By an ***operator*** *we mean either a function symbol or a constant.*

Well-formed (ground) **terms** generated by a signature Sg are defined by the inference rules depicted in Fig. 1. A judgement of the form $t\colon s$ means that t is a well-formed term of sort s built out of Sg. Note that a term t may have more than one sort, and we call t **polymorphic**; see [20] for details on polymorphism in order-sorted algebras and [7] for the role of polymorphism in multi-languages.

Definition 2 (Order-Sorted Algebra). *Given an order-sorted signature* Sg*, an* Sg***-algebra*** $\mathscr{C}$ *is specified by*

- *a* ***carrier set*** $[\![s]\!]_{\mathscr{C}}$ *for each sort* s *and a set* $[\![w]\!]_{\mathscr{C}} \triangleq [\![s_1]\!]_{\mathscr{C}} \times \cdots \times [\![s_n]\!]_{\mathscr{C}}$ *for each* $w \triangleq s_1 \ldots s_n \in S^*$*;*
- *a function* $[\![f\colon w \to s]\!]_{\mathscr{C}}\colon [\![w]\!]_{\mathscr{C}} \to [\![s]\!]_{\mathscr{C}}$ *for each* $f\colon w \to s$ *and an element* $[\![k]\!]_{\mathscr{C}} \in [\![s]\!]_{\mathscr{C}}$ *for each constant* $k\colon s$

such that if $s \le r$ *then* $[\![s]\!]_{\mathscr{C}} \subseteq [\![r]\!]_{\mathscr{C}}$*, and if the function symbol* f *appears with more than one rank* $f\colon w_1 \to s$ *and* $f\colon w_2 \to r$ *in* Sg *with* $w_1 \le w_2$*, then* $[\![f\colon w_1 \to s]\!]_{\mathscr{C}}(x) = [\![f\colon w_2 \to r]\!]_{\mathscr{C}}(x)$ *for each* $x \in [\![w_1]\!]_{\mathscr{C}}$*.*

We often refer to the **carrier set** of an Sg-algebra $\mathscr{C}$, meaning the S-sorted family of carrier sets $C \triangleq (\, C_s \triangleq [\![s]\!]_{\mathscr{C}} \mid s \in S \,)$, where S is the set of sorts in Sg.

Example 1. If Sg is the signature of an imperative language, then the *loop* and *skip* operators are likely to be sorted as $\mathit{loop}\colon \mathit{exp}, \mathit{com} \to \mathit{com}$ and $\mathit{skip}\colon \mathit{com}$. Their denotational semantics may be defined by an algebra $\mathscr{D}$, where $[\![\mathit{skip}]\!]_{\mathscr{D}} \triangleq \rho \mapsto \rho$ and $[\![\mathit{loop}]\!]_{\mathscr{D}}(e, c) \triangleq \operatorname{lfp} F_{e,c}$ (assume ρ to be an environment and F the usual continuous operator [41]); see Appx. C for further examples of different styles of algebraic semantics of an imperative language.

The *term algebra* $\mathscr{T}_{\mathsf{Sg}}$ has carrier sets each consisting of terms of a given sort:

Definition 3 (Order-Sorted Term Algebra). *The* ***term*** Sg***-algebra*** $\mathscr{T}_{\mathsf{Sg}}$ *is defined by taking* $[\![s]\!]_{\mathscr{T}_{\mathsf{Sg}}} \triangleq \{\, t \mid t\colon s \text{ in } \mathsf{Sg} \,\}$ *and by syntactically interpreting each operator,* i.e., $[\![k]\!]_{\mathscr{T}_{\mathsf{Sg}}} \triangleq k$ *for each* $k\colon s$ *and* $[\![f\colon w \to s]\!]_{\mathscr{T}_{\mathsf{Sg}}}(t_1, \ldots, t_n) \triangleq f(t_1, \ldots, t_n)$ *for each* $f\colon w \to s$*, where* $w \triangleq s_1 \ldots s_n$ *and* $t_i \in [\![s_i]\!]_{\mathscr{T}_{\mathsf{Sg}}}$*.*

Homomorpisms between algebras are sorted functions between carrier sets that preserve the meaning of the operators:

Definition 4 (Order-Sorted Homomorpism). *An* ***order-sorted*** Sg***-homomorphism*** $h\colon \mathscr{C} \to \mathscr{D}$ *between two* Sg*-algebras* $\mathscr{C}$ *and* $\mathscr{D}$ *is an* S*-sorted function* $h\colon C \to D$ *between their carrier sets satisfying the following:*

- $h_s([\![k]\!]_{\mathscr{C}}) = [\![k]\!]_{\mathscr{D}}$ *for each* $k\colon s$ *and* $h_s \circ [\![f\colon w \to s]\!]_{\mathscr{C}} = [\![f\colon w \to s]\!]_{\mathscr{D}} \circ h_w$ *for each* $f\colon w \to s$ *(recall the sorted function notation); and*
- *if* $s \leq r$*, then* $h_s(x) = h_r(x)$ *for each* $x \in [\![s]\!]_{\mathscr{C}}$*.*

When the source of a homomorphism $h\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}$ is the term algebra, it provides terms of Sg with a meaning in the carrier set C of $\mathscr{C}$. The conditions for h in the previous definition amount to require the semantics to be *compositional*. The class of Sg-algebras and Sg-homomorphisms form a category denoted by $\mathbf{Alg}(\mathsf{Sg})$. If the signature Sg is *regular*,[2] there is a unique homomorphism $h\colon \mathscr{T}_{\mathsf{Sg}} \to C$ for each Sg-algebra $\mathscr{C}$. This fact is summed up by the following

Theorem 1 ([20]). *If* Sg *is regular, the term algebra* $\mathscr{T}_{\mathsf{Sg}}$ *is* initial *in* $\mathbf{Alg}(\mathsf{Sg})$*, that is, for every algebra* $\mathscr{C}$ *there is a unique homomorphism* $h\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}$*.*

We refer to any such $h\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}$ as a **semantic function**. In the following, we write $[\![t]\!]_{\mathscr{C}} \triangleq h_{ls(t)}(t)$ where $ls(t)$ is the least sort of t, which exists by regularity [20, Prop. 2.10].

2.2 Multi-Languages and Their Algebras

We next provide an analogue of the previous section for multi-languages. A *multi-language* signature, defining a *multi-language*, is specified by two order-sorted signatures together with an *interoperability relation* on sorts:

Definition 5 (Multi-Language Signature). *A* ***multi-language signature*** $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$ *is specified by*

- *a pair of order-sorted signatures* Sg_1 *and* Sg_2 *with posets of sorts* $\langle S_1, \leq_1 \rangle$ *and* $\langle S_2, \leq_2 \rangle$*, respectively; and*
- *an* ***interoperability (binary) relation*** $\ltimes$ *over* $S_1 \cup S_2$ *such that* $s \ltimes s'$ *implies* $s \in S_i$ *and* $s' \in S_j$ *with* $i, j \in \{1, 2\}$ *and* $i \neq j$*.*

We suppose that S_1 *and* S_2 *are two disjoint sets (this hypothesis is non-restrictive: We can always construct a disjoint union).*

[2] We do not discuss regularity here (see [20] for the formal definition).

$$\textsf{(CONST)}\ \frac{-}{k_i\colon s}\ (k\colon s \text{ in } \mathsf{Sg}_i) \qquad \textsf{(FUN)}\ \frac{(\forall 1 \le j \le n)\ \ t_j\colon s_j}{f_i(t_1,\ldots,t_n)\colon s}\ (f\colon s_1,\ldots,s_n \to s \text{ in } \mathsf{Sg}_i)$$

$$\textsf{(SUB)}\ \frac{t\colon s}{t\colon r}\ (s \le_i r \text{ in } \mathsf{Sg}_i) \qquad \textsf{(CONV)}\ \frac{t\colon s}{\hookrightarrow_{s,s'}(t)\colon s'}\ (s \ltimes s' \text{ in } \mathsf{SG})$$

Fig. 2. Multi-language terms generated by $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$.

We shall see that an interoperability constraint $s \ltimes s'$ enables the use of Sg_i-terms of sort s in place of Sg_j-terms of sort s' (as in [31], "ML code can be used in place of Scheme code"). The **multi-language terms** are inductively defined by the rules in Fig. 2. Note that single-language operators, k or f, of Sg_i are tagged as k_i and f_i in multi-language terms, precisely for avoiding the introduction of unintended polymorphism (Sg_1 and Sg_2 may share operators with the same name). And also note the role of **conversion operators** $\hookrightarrow_{s,s'}$ that move terms t of sort s to terms $\hookrightarrow_{s,s'}(t)$ of sort s', for each $s \ltimes s'$. Henceforth, when we write an order-sorted signature Sg_i, we understand its poset of sorts to be denoted by $\langle S_i, \le_i \rangle$, and for each $s \ltimes s'$ we tacitly assume that s in Sg_i and s' in Sg_j with $i, j \in \{1, 2\}$ and $i \neq j$.

A multi-language SG-algebra is a pair of order-sorted algebras together with a family of *boundary functions*.

Definition 6 (Multi-Language Algebra). *Let* $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$ *be a multi-language signature. An* SG-***algebra*** $\mathscr{C}$ *is given by*

- *a pair of order-sorted algebras* $\mathscr{C}_1$ *and* $\mathscr{C}_2$ *on* Sg_1 *and* Sg_2*, respectively; and*
- *a* ***boundary function*** $[\![s \ltimes s']\!]_{\mathscr{C}}\colon [\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j}$ *for each constraint* $s \ltimes s'$*.*

Boundary functions are understood as the semantics of conversion operators, that is they specify *how* the underlying languages interoperate.

The multi-language term algebra is defined in a similar way to the order-sorted one in order to carry multi-language terms.

Definition 7 (Multi-Language Term Algebra). *Let* $\mathscr{T}_{\mathsf{SG}}$ *denote the multi-language term* SG*-algebra. The underlying* Sg_i*-algebras* $(\mathscr{T}_{\mathsf{SG}})_i$ *are defined by taking* $[\![s]\!]_{(\mathscr{T}_{\mathsf{SG}})_i} \triangleq \{\, t \mid t\colon s \text{ in } \mathsf{SG} \,\}$ *for each sort* s *in* Sg_i *and by defining* $[\![k]\!]_{(\mathscr{T}_{\mathsf{SG}})_i} \triangleq k_i$ *for each* $k\colon s$ *in* Sg_i *and* $[\![f\colon w \to s]\!]_{(\mathscr{T}_{\mathsf{SG}})_i}(t_1,\ldots,t_n) \triangleq f_i(t_1,\ldots,t_n)$ *for each* $f\colon s_1,\ldots,s_n \to s$ *in* Sg_i*, where* $t_j \in [\![s_j]\!]_{(\mathscr{T}_{\mathsf{SG}})_i}$*. Boundary functions of* $\mathscr{T}_{\mathsf{SG}}$ *are defined as* $[\![s \ltimes s']\!]_{\mathscr{T}_{\mathsf{SG}}}(t) \triangleq \hookrightarrow_{s,s'}(t)$*, for each* $s \ltimes s'$ *and* $t \in [\![s]\!]_{(\mathscr{T}_{\mathsf{SG}})_i}$*.*

Definition 8 (Multi-Language Homomorphism). *Let* $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$ *be a multi-language signature, and let* $\mathscr{C}$ *and* $\mathscr{D}$ *be two* SG*-algebras. An* SG-***homomorphism*** $h\colon \mathscr{C} \to \mathscr{D}$ *is given by a pair of order-sorted homomorphisms* $h_1\colon \mathscr{C}_1 \to \mathscr{D}_1$ *and* $h_2\colon \mathscr{C}_2 \to \mathscr{D}_2$ *such that they commute with boundary functions, namely, if* $s \ltimes s'$*, then* $(h_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{C}} = [\![s \ltimes s']\!]_{\mathscr{D}} \circ (h_i)_s$*.*

Given a multi-language algebra $\mathscr{C}$, we can define an S-sorted set C by setting $C_{s_i} \triangleq [\![s]\!]_{\mathscr{C}_i}$; and given any homomorphism $h\colon \mathscr{C} \to \mathscr{D}$, there is an S-sorted homomorphism $\overline{h}\colon C \to D$ given by $\overline{h}_{s_i} \triangleq (h_i)_s\colon C_{s_i} \to D_{s_i}$. Note that we will usually write h for $\overline{h}$, thus identifying the two concepts, and regard $h\colon \mathscr{C} \to \mathscr{D}$ and $h\colon C \to D$ as inter-changeable throughout the paper.

SG-algebras and SG-homomorphisms form a category denoted by $\mathbf{Alg}(\mathsf{SG})$. [7] provides the multi-language version of Thm. 1, so that for every multi-language algebra $\mathscr{C}$, there is a unique SG-homomorphism $h\colon \mathscr{T}_{\mathsf{SG}} \to \mathscr{C}$ providing multi-language terms with a meaning, and we use the same notation $[\![t]\!]_{\mathscr{C}}$ for denoting its semantics.

3 Algebraic Perspective on Collecting Semantics

We now give a general construction of *collecting semantics*. First we set up notation. We let Sg be a regular order-sorted signature and $\mathscr{C}$ an Sg-algebra. Thm. 1 guarantees the existence of a homomorphism $[\![-]\!]_{\mathscr{C}}\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}$ providing Sg-terms $\mathtt{P}$, called **programs**, with a meaning $[\![\mathtt{P}]\!]_{\mathscr{C}}$.

Remark 1. Signatures are of course completely general. Sg might specify an enriched lambda-calculus with $[\![-]\!]_{\mathscr{C}}$ its denotational semantics, as in [21, Sect. 3.2], or Sg might specify the syntax of an imperative language with $[\![-]\!]_{\mathscr{C}}$ its small-step operational semantics (as in Appx. C.1).

A property of a set is any subset. By *semantic* properties of programs, we mean properties of the (components of) the carrier set C of an algebra $\mathscr{C}$. In this section, we provide a systematic construction of an algebra $\mathscr{C}^*$ able to compute the **strongest property of programs**, that is $[\![\mathtt{P}]\!]_{\mathscr{C}^*} = \{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$ for each program $\mathtt{P}$ (Prop. 1). The induced semantic function $[\![-]\!]_{\mathscr{C}^*}$ is usually called the **(standard) collecting semantics** [14]. Henceforth, we distinguish the semantics $[\![-]\!]_{\mathscr{C}}$ from the collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ by referring to the former as the **standard semantics**, and we shall link them algebraically in the category of algebras $\mathbf{Alg}(\mathsf{Sg})$ via Prop. 2. We conclude this section by providing a general fixpoint calculation of $[\![-]\!]_{\mathscr{C}^*}$ whenever $[\![-]\!]_{\mathscr{C}}$ is a fixpoint semantics (Thm. 3).

Definition 9 (Collecting Semantics). *Let $\mathscr{C}$ be an Sg-algebra. The **collecting semantics** $\mathscr{C}^*$ over $\mathscr{C}$ is defined as follows:*

- *the carrier sets are $[\![s]\!]_{\mathscr{C}^*} \triangleq \wp[\![s]\!]_{\mathscr{C}}$ for each sort s; and*
- *the semantics of the operators is $[\![k]\!]_{\mathscr{C}^*} \triangleq \{[\![k]\!]_{\mathscr{C}}\}$ for each constant $k\colon s$, and $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \triangleq \wp[\![f\colon w \to s]\!]_{\mathscr{C}} \circ \times$ (see Sect. 2 for the definition of $\times$) for each function symbol $f\colon w \to s$, where for any function $\theta\colon A \to B$ the function $\wp(\theta)\colon \wp(A) \to \wp(B)$ computes the image of θ.*

The homomorphism $[\![-]\!]_{\mathscr{C}^*}\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}^*$ induced by $\mathscr{C}^*$ maps programs to their most precise semantic property, justifying the name of collecting semantics:

Proposition 1. $[\![-]\!]_{\mathscr{C}^*} \colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}^*$ *computes the strongest program property for each program* $\mathtt{P}$ *generated from* Sg*, that is* $[\![\mathtt{P}]\!]_{\mathscr{C}^*} = \{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$.

Remark 2. Other forms of collecting semantics found in literature are abstractions of the collecting semantics provided here. For instance, [2] defines a collecting semantics for functional programs interpreted on $D_\bot \to D_\bot$ by taking $f \colon D_\bot \to D_\bot$ to $\wp(f)$ on the lifted domain $\wp(D_\bot) \to_{\mathsf{cjm}} \wp(D_\bot)$ of *complete-join morphisms* (cjm). Such a collecting semantics computes all the possible results of a functional program with respect to a set of input values, and it can be obtained as an abstraction of the standard collecting semantics:

$$\alpha\big(S \in \wp(D_\bot \to D_\bot)\big) \triangleq X \in \wp(D_\bot) \mapsto \{\, f(x) \in D_\bot \mid x \in X \wedge f \in S \,\}$$
$$\gamma\big(F \in \wp(D_\bot) \to_{\mathsf{cjm}} \wp(D_\bot)\big) \triangleq \{\, x \in D_\bot \mapsto f(x) \in D_\bot \mid f(x) \in F(\{x\}) \,\}^{3}$$

The *(forward) reachability semantics* is widely used for invariance analyses or, in general, for discovering state properties of programs [39,6,27]. For each program $\mathtt{P}$, it collects the set of states that are reachable by running $\mathtt{P}$ from a set of initial states. It can be shown that it is an abstraction of the standard collecting semantics over, for instance, a *trace semantics* (such a construction is formalised in Appx. C.3).

Standard and collecting semantics are related by the property established in Prop. 1. Moreover, the singleton function $\{-\} \colon C \to \wp(C)$ that maps standard semantics $[\![\mathtt{P}]\!]_{\mathscr{C}}$ of programs to their strongest property $\{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$ is an Sg-homomorphism $\{-\} \colon \mathscr{C} \to \mathscr{C}^*$. From the abstract interpretation perspective, it means that $\{-\}$ acts as a *complete abstraction*, hence with no loss of precision [17].

Proposition 2. *The singleton function* $\{-\} \colon C \to \wp(C)$ *defined by* $c \mapsto \{c\}$ *is an* Sg*-homomorphism* $\{-\} \colon \mathscr{C} \to \mathscr{C}^*$*, and therefore* $\{-\} \circ [\![-]\!]_{\mathscr{C}} = [\![-]\!]_{\mathscr{C}^*}$.

Remark 3. Readers familiar with category theory may notice that $\{-\}$ is the unit of the *powerset monad* on $\mathbf{Alg}(\mathsf{Sg})$. However, we do not pursue this here.

3.1 Fixpoint Calculation of Collecting Semantics

Preliminary Notions. We call $[\![-]\!]_{\mathscr{C}}$ a **fixpoint semantics** if $[\![\mathtt{P}]\!]_{\mathscr{C}} \triangleq \mathrm{lfp}^{\preccurlyeq}_{\bot} F$ for some *semantic transformer* $F \colon C \to C$ (depending on Sg) on a *semantic domain* $\langle C, \preccurlyeq, \bot, \curlyvee \rangle$. More precisely, we follow [11] and we assume the following:

- The **semantic domain** $\langle C, \preccurlyeq, \bot, \curlyvee \rangle$ is a poset $\langle C, \preccurlyeq \rangle$ with a smallest element $\bot$ and a *partially* defined least upper bound (lub) operator $\curlyvee$.
- The **semantic transformer** $F \colon C \to C$ is a monotone function such that its *transfinite iterates* $F^0 \triangleq \bot$, $F^{\delta+1} \triangleq F(F^\delta)$ for successor ordinals $\delta + 1$, and $F^\lambda \triangleq \curlyvee_{\delta<\lambda} F^\delta$ for limit ordinals λ are well-defined.

[3] One can check that $\alpha(S)$ is a cjm and (α, γ) form a Galois connection.

Under these assumptions, the fixpoint semantics is exactly $[\![\mathsf{P}]\!]_{\mathscr{C}} \triangleq \mathrm{lfp}^{\preccurlyeq}_{\bot} F = F^{\epsilon}$, where ϵ is the least ordinal such that $F(F^{\epsilon}) = F^{\epsilon}$.

The goal of this section is to provide a fixpoint calculation of the collecting semantics. We assume the standard semantics $[\![\mathsf{P}]\!]_{\mathscr{C}} \triangleq \mathrm{lfp}^{\preccurlyeq}_{\bot} F$ to be a fixpoint semantics over a generic semantic domain $\langle C, \preccurlyeq, \bot, \curlyvee \rangle$, and we define a new *computational order* $\preccurlyeq^*$ on $\wp(C)$ that makes the collecting semantics $[\![\mathsf{P}]\!]_{\mathscr{C}^*}$ the least fixpoint of $F^* \triangleq \wp(F)$ (Thm. 3).

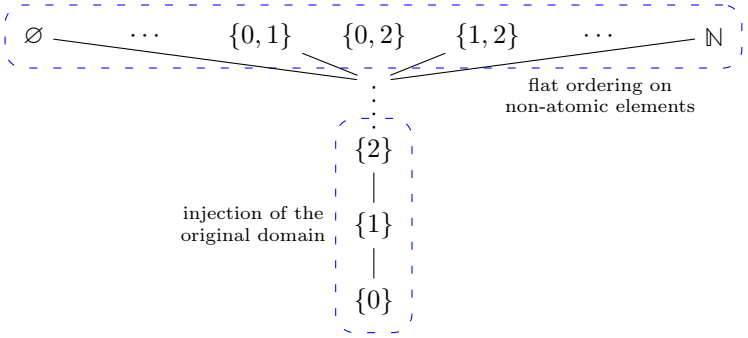
Remark 4. The problem of finding the right partial order $\preccurlyeq^*$ on $\wp(C)$ for achieving such a fixpoint definition of the collecting semantics has been recently addressed in [14, Sect. 7.2], by considering a preorder on $\wp(C)$ that is partial *only* along the iterates.[4] Here, we show that it can be extended to a fully-fledged partial order $\preccurlyeq^*$ over the whole set $\wp(C)$.

Definition 10 (Collecting Semantics Domain). *Let $\langle C, \preccurlyeq, \bot, \curlyvee \rangle$ be a (non-trivial) semantic domain. The* ***collecting semantics domain*** *$\langle \wp(C), \preccurlyeq^*, \bot^*, \curlyvee^* \rangle$ with respect to $\langle C, \preccurlyeq, \bot, \curlyvee \rangle$ is defined as follows:*

– *Let $X, Y \in \wp(C)$. Then,*

$$X \preccurlyeq^* Y \text{ iff } \begin{cases} X = \{x\} \text{ and } Y = \{y\} \text{ and } x \preccurlyeq y \text{ for some } x, y \in C & (1) \\ X = Y & (2) \\ X = \{x\} \text{ and } Y \neq \{y\} \text{ for some } x \in C \text{ and for all } y \in C & (3) \end{cases}$$

Example 2. Let $C \triangleq \mathbb{N}$ and $\preccurlyeq \triangleq \leq$. Then, $\langle \wp(\mathbb{N}), \leq^* \rangle$ is



The smallest element of $\langle \wp(C), \preccurlyeq^* \rangle$ is $\bot^* = \{\bot\}$ and the following lemma characterises the lub operator $\curlyvee^*$ on chains of atoms (*i.e.*, singletons):

Lemma 1. *Let $\mathcal{D} \subseteq \wp(C)$ be a non-empty set of atoms of the form $\{x\}$ for some $x \in C$. Then, $\curlyvee^* \mathcal{D}$ exists if and only if $\curlyvee \cup \mathcal{D}$ exists, and when either one exists $\curlyvee^* \mathcal{D} = \{\curlyvee \cup \mathcal{D}\}$.*

Let us recall that $F^* \triangleq \wp(F) \triangleq X \in \wp(C) \mapsto \{\, F(x) \in C \mid x \in X \,\}$. F^* is trivially monotone, and we shall now prove that its transfinite iterates exist:

[4] A similar approach has been previously taken in [30] and in the thesis of Pasqua [34].

Proposition 3. *For each ordinal δ, $(F^*)^\delta = \{F^\delta\}$.*

Proof. By transfinite induction:

- Let $\delta \triangleq 0$. Then, $(F^*)^0 = \bot^* = \{\bot\} = \{F^0\}$.
- Let $\delta + 1$ be a successor ordinal. Then, $(F^*)^{\delta+1} = F^*((F^*)^\delta) \overset{\text{IH}}{=} F^*(\{F^\delta\}) = \{F(F^\delta)\} = \{F^{\delta+1}\}$.
- Let λ be a limit ordinal. Then, $(F^*)^\lambda = \Upsilon^*_{\delta<\lambda}(F^*)^\delta \overset{\text{IH}}{=} \Upsilon^*_{\delta<\lambda}\{F^\delta\}$. Since $(\,\{F^\delta\} \mid \delta < \lambda\,)$ is a set of atoms, by Lemma 1 we conclude

$$(F^*)^\lambda = \bigUpsilon^*_{\delta<\lambda}\{F^\delta\} = \left\{\bigUpsilon \bigcup_{\delta<\lambda}\{F^\delta\}\right\} = \left\{\bigUpsilon_{\delta<\lambda} F^\delta\right\} = \{F^\lambda\}$$

By Prop. 3, F^* is a proper semantic transformer over the previously defined collecting semantics domain. The fixpoint definition of the collecting semantics now follows from the application of the Kleenian fixpoint transfer theorem in its most general formulation (that we now recall).

Theorem 2 (Kleenian Fixpoint Transfer Theorem [11]). *Let $(D, \leq, \bot, \vee)$ and $(D^\natural, \leq^\natural, \bot^\natural, \vee^\natural)$ be two semantic domains and $F\colon D \to D$ and $F^\natural\colon D^\natural \to D^\natural$ two semantic transformer over them. Let $\alpha\colon D \to D^\natural$ be a function such that (i) $\alpha(\bot) = \bot^\natural$, (ii) $F^\natural \circ \alpha = \alpha \circ F$, and (iii) α preserves the lub of the iterates, that is $\alpha(\vee_{\delta<\lambda} F^\delta) = \vee^\natural_{\delta<\lambda}\alpha(F^\delta)$ for each limit ordinal λ. Then, $\alpha(\mathrm{lfp}^{\leq}_{\bot} F) = \mathrm{lfp}^{\leq^\natural}_{\bot^\natural} F^\natural$.*

The singleton function $\{-\}\colon C \to \wp(C)$ introduced in Prop. 2 satisfies the hypotheses for α in Thm. 2 for domains $\langle C, \preccurlyeq, \bot, \Upsilon\rangle$ and $\langle \wp(C), \preccurlyeq^*, \bot^*, \Upsilon^*\rangle$ and transformers F and F^*, respectively. Therefore, given the above,

Theorem 3 (Fixpoint Collecting Semantics). *The function $\{-\}\colon C \to \wp(C)$ satisfies the hypotheses (i), (ii), and (iii) of Thm. 2, hence*

$$[\![\mathsf{P}]\!]_{\mathscr{C}^*} \overset{\text{Prop. 1}}{=} \{[\![\mathsf{P}]\!]_{\mathscr{C}}\} \overset{\text{Hypo.}}{=} \{\mathrm{lfp}^{\preccurlyeq}_{\bot} F\} \overset{\text{Thm. 2}}{=} \mathrm{lfp}^{\preccurlyeq^*}_{\bot^*} F^*$$

4 Basic Notions of Algebraic Abstract Semantics

We proceed to characterise *abstract interpretations* of the standard collecting semantics in the algebraic setting. There are several frameworks in which to design sound approximations of program semantics [13]. Here, we study both the ideal case in which a *Galois connection* (gc) ties the concrete and the abstract domain, and the more general scenario characterised by the absence of a best approximation function. Although it is not the most general abstract interpretation framework to work with, it meets the usual setting in which static analyses are designed [38].

We still denote by $\mathscr{C}$ the Sg-algebra inducing the standard semantics of the language. We recall that C is the carrier set of $\mathscr{C}$ and $\wp(C)$ the carrier set of the collecting semantics $\mathscr{C}^*$. An Sg-algebra $\mathscr{A}$ is said to be **abstract** with respect to $\mathscr{C}$ if (1) its carrier set A is a poset $\langle A, \sqsubseteq\rangle$ and (2) it is equipped with a monotone **concretisation function** $\gamma\colon \langle A, \sqsubseteq\rangle \to \langle \wp(C), \subseteq\rangle$ that maps abstract elements to concrete properties. We refer to the carrier set A as an **abstract domain** and we call the induced semantic function $[\![-]\!]_{\mathscr{A}}$ the **abstract semantics**.

Definition 11 (Soundness). *Let $\mathscr{A}$ be an abstract* Sg*-algebra (with carrier set A) and $\gamma\colon A \to \wp(C)$ a monotone concretisation function. The algebra $\mathscr{A}$ is* ***sound*** *if its operators soundly approximate the concrete ones,* i.e.*,*

- $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ *for each* $f\colon w \to s$*; and*
- $[\![k]\!]_{\mathscr{C}^*} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ *for each constant* $k\colon s$*.*

A straightforward consequence of this definition is that sound algebras induce sound abstract semantic functions (but see the proof in Appx. A for subtleties related to polymorphism):

Proposition 4. *If $\mathscr{A}$ is sound, then $[\![\mathsf{P}]\!]_{\mathscr{C}^*} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ for each program* P*.*

If $\langle \wp(C), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$ is a gc between the concrete and the abstract domain, we can define the *most precise (abstract)* Sg*-algebra* $\mathscr{A}^\diamond$ out of the *best correct approximation* provided by (α, γ).

Definition 12 (Most Precise Algebra). *Let $\langle \wp(C), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$ be a gc between the concrete and the abstract domain. The* ***most precise algebra*** *$\mathscr{A}^\diamond$ approximating $\mathscr{C}$ with respect to (α, γ) is defined as follows:*

- *carrier sets are $[\![s]\!]_{\mathscr{A}^\diamond} \triangleq A_s$ (recall the notation for sorted sets); and*
- *the semantics of the operators is $[\![k]\!]_{\mathscr{A}^\diamond} \triangleq \alpha[\![k]\!]_{\mathscr{C}^*}$ for each constant $k\colon s$, $[\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \triangleq \alpha \circ [\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma$ for each function symbol $f\colon w \to s$.*

The abstract semantics $[\![-]\!]_{\mathscr{A}^\diamond}$ induced by $\mathscr{A}^\diamond$ enjoys the soundness property (the reader may want to check that $\mathscr{A}^\diamond$ is a proper Sg-algebra), and it is the most precise among all the sound algebras, in the following sense:

Proposition 5. *$\mathscr{A}^\diamond$ soundly approximates the concrete semantics. Moreover, $\mathscr{A}^\diamond$ is the most precise abstraction with respect to (α, γ), that is, for any other sound algebra $\mathscr{A}$, $\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ and $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ for each operator in* Sg*. Therefore, by Prop. 4, $\gamma[\![\mathsf{P}]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ for each program* P*.*

In general, abstraction and concretisation functions α and γ are not homomorphisms between $\mathscr{A}$ and $\mathscr{C}^*$. However, if they are homomorphisms, then $\mathscr{A}$ is *backward* and *forward complete*, respectively (Props. 6 and 7).

Definition 13 ((Backward) Completeness). *Let $\mathscr{A}$ be an* Sg*-algebra and $\langle \wp(C), \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$ a gc. The left adjoint α encodes concrete properties in the abstract domain. The algebra $\mathscr{A}$ is* ***complete*** *with respect to*

- *a function symbol $f\colon w \to s$ if $\alpha \circ [\![f\colon w \to s]\!]_{\mathscr{C}^*} = [\![f\colon w \to s]\!]_{\mathscr{A}} \circ \alpha$; and*
- *a constant $k\colon s$ if $\alpha[\![k]\!]_{\mathscr{C}^*} = [\![k]\!]_{\mathscr{A}}$.*

Proposition 6. *Let $\alpha\colon \mathscr{C}^* \to \mathscr{A}$ be an* Sg*-homomorphism. Then, $\mathscr{A}$ is complete with respect to each operator in* Sg*, and therefore $\alpha[\![\mathsf{P}]\!]_{\mathscr{C}^*} = [\![\mathsf{P}]\!]_{\mathscr{A}}$ for each program* P*.*

Note that, in general, the existence of a best abstract approximation is not guaranteed (*e.g.*, for convex polyhedra [15] or the final state automata domain [3]). In such cases, a dual notion of completeness (*forward completeness* [18]) with respect to the concretisation function is investigated.

Definition 14 (Forward Completeness). *Let $\mathscr{A}$ be an* Sg*-algebra and* $\gamma\colon A \to \wp(C)$ *a monotone concretisation function. The algebra $\mathscr{A}$ is* ***forward complete*** *with respect to*

- *a function symbol* $f\colon w \to s$ *if* $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma = \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$*; and*
- *a constant* $k\colon s$ *if* $[\![k]\!]_{\mathscr{C}^*} = \gamma[\![k]\!]_{\mathscr{A}}$*.*

Proposition 7. *Let* $\gamma\colon \mathscr{A} \to \mathscr{C}^*$ *be an* Sg*-homomorphism. Then, $\mathscr{A}$ is forward complete with respect to each operator in* Sg*, and therefore* $[\![\mathtt{P}]\!]_{\mathscr{C}^*} = \gamma[\![\mathtt{P}]\!]_{\mathscr{A}}$ *for each program* `P`*.*

5 The Multi-Language Abstraction

Our aim in this section is to define *abstractions of the multi-language semantics* by relying on the *abstractions of the single-languages*. The whole section is accompanied by a *running example* inspired by a common scenario in the interoperability field: The *language binding*, an Application Program Interface (API) that allows one language to call library functions implemented in another language. Major examples include `openGL` library, which is interoperable with Java through the Java OpenGL (JOGL) wrapper library or from Python via PyOpenGL, and GNU Octave language that have interoperability with Ruby and Python (*e.g.*, see `octave-ruby` and `oct2py` libraries). Our running example mimics such an interoperability mechanism.

In Sect. 5.1 we set up our example. We present the core of an imperative language Imp, and by recalling the multi-language construction of Def. 5, we apply the construction so that Imp interoperates with a very simple mathematical language Num (in the spirit of Octave). In Sect. 5.2 we define the combination of abstract interpretations of different languages, a key contribution of our work. By example we apply our theory to two different sign abstract semantics (one for each language), and we show how to derive the sign semantics for the multi-language NImp obtained by blending Imp and Num.

5.1 The Multi-Language Construction: A Running Example

Throughout this section, we let Sg_1 and Sg_2 be two order-sorted signatures defining the syntax of two languages, and let $i \triangleq 1, 2$. We denote by $\mathscr{C}_i$ the order-sorted Sg_i-algebra inducing the semantics $[\![-]\!]_{\mathscr{C}_i}$ of the language.

Running Example 1. Let Imp be (the syntax of) the imperative programming language in Fig. 3. Variables $x \in \mathbb{X}$ and values $i \in \mathbb{Z}_\bot$ (where $\mathbb{Z}_\bot \triangleq \mathbb{Z} \cup \{\bot\}$) occur in the language as terminal symbols, and for each production defining

(i)	$\langle exp_1 \rangle ::= i$	integers ($i \in \mathbb{Z}_\perp$)
(x)	$\langle exp_1 \rangle ::= x$	variables ($x \in \mathbb{X}$)
$(bop_\odot)$	$\langle exp_1 \rangle ::= \langle exp_1 \rangle \odot \langle exp_1 \rangle$	binary operations
$(skip)$	$\langle com_1 \rangle ::=$ `skip`	do-nothing
$(assign_x)$	$\langle com_1 \rangle ::=$ `x =` $\langle exp_1 \rangle$	assignment
$(cond)$	$\langle com_1 \rangle ::=$ `if` $\langle exp_1 \rangle$ `then` $\langle com_1 \rangle$ `else` $\langle com_1 \rangle$	conditional
$(loop)$	$\langle com_1 \rangle ::=$ `while` $\langle exp_1 \rangle$ `do` $\langle com_1 \rangle$	loop statement
(seq)	$\langle com_1 \rangle ::= \langle com_1 \rangle$ `;` $\langle com_1 \rangle$	composition

Fig. 3. Syntax of the imperative language Imp.

(q)	$\langle exp_2 \rangle ::= q$	rationals ($q \in \mathbb{Q}_\perp$)
(x)	$\langle exp_2 \rangle ::= x$	variables ($x \in \mathbb{X}$)
(f_n)	$\langle exp_2 \rangle ::= f_n$ `(` $\langle exp_2 \rangle_1$ `,` $\ldots$ `,` $\langle exp_2 \rangle_n$ `)`	n-ary operations
$(?)$	$\langle exp_2 \rangle ::= \langle exp_2 \rangle$ `?` $\langle exp_2 \rangle$ `:` $\langle exp_2 \rangle$	ternary operator
(let_x)	$\langle com_2 \rangle ::=$ `x =` $\langle com_2 \rangle$	assignment
$(block)$	$\langle com_2 \rangle ::=$ `{` $\langle com_2 \rangle_1$ `;` $\ldots$ `;` $\langle com_2 \rangle_n$ `}`	statements block

Fig. 4. Syntax of the mathematical language Num.

the syntax of Imp (on the right), we introduce a corresponding algebraic operator (on the left), or a family of operators when they are parametric on a subscript. The rank of each algebraic operator can be inferred by the non-terminals appearing in the production rules; for instance, the operator *cond* is sorted as $cond\colon exp_1, com_1, com_1 \to com_1$, where com_1 and exp_1 denote the sort of commands and expressions of Imp. We assume a denotational semantics $[\![-]\!]_{\mathscr{D}_1}$ provided by an Imp-algebra $\mathscr{D}_1$ (see, for instance, [41] or Appendix B). As usual, we let $\mathbb{E}\mathbb{n}\mathbb{v}_1 \triangleq \mathbb{X} \to \mathbb{Z}_\perp$ be the set of environments of Imp, and we set $\mathbb{E}\mathbb{n}\mathbb{v}_1^\perp \triangleq \mathbb{E}\mathbb{n}\mathbb{v}_1 \cup \{\perp\}$. The carrier sets of commands and expressions are $[\![com_1]\!]_{\mathscr{D}_1} \triangleq \mathbb{E}\mathbb{n}\mathbb{v}_1^\perp \to \mathbb{E}\mathbb{n}\mathbb{v}_1^\perp$ and $[\![exp_1]\!]_{\mathscr{D}_1} \triangleq \mathbb{E}\mathbb{n}\mathbb{v}_1 \to \mathbb{Z}_\perp$. Moreover, we assume that Imp provides users with very basic operators, *i.e.*, $\odot \in \{\texttt{+}, \texttt{-}, \texttt{<}, \texttt{>}, \texttt{==}, \texttt{!=}\}$.

We let Num be a mathematical language with more advanced numerical functions, such as modulo and bitwise operators, rational numbers, trigonometric functions, *etc.* Its syntax is depicted in Fig. 4. We consider variables $x \in \mathbb{X}$ and values $q \in \mathbb{Q}_\perp \triangleq \mathbb{Q} \cup \{\perp\}$ terminal symbols. We denote by f_n mathematical functions of the language with arity n, such as the modulo binary operator `%`, the unary `sin` function, etc. Here too we assume a denotational semantics $[\![-]\!]_{\mathscr{D}_2}$ induced by the Num-algebra $\mathscr{D}_2$, where $[\![exp_2]\!]_{\mathscr{D}_2} \triangleq \mathbb{E}\mathbb{n}\mathbb{v}_2 \to \mathbb{Q}_\perp$ and $[\![com_2]\!]_{\mathscr{D}_2} \triangleq \mathbb{E}\mathbb{n}\mathbb{v}_2 \to \mathbb{E}\mathbb{n}\mathbb{v}_2$ with $\mathbb{E}\mathbb{n}\mathbb{v}_2 \triangleq \mathbb{X} \to \mathbb{Q}_\perp$.

The reader may want to check Appx. B for the thorough formalisation of the algebraic semantics provided by $\mathscr{D}_1$ and $\mathscr{D}_2$.

Recall (Definition 5) that the signature of a multi-language, which we shall refer to as SG, is specified by blending the order-sorted signatures Sg_1 and Sg_2 of

```
res = 1;
while n > 0 do
  if n & 1 then // true iff n is odd
    res = res * a;
  a = a * a;
  n = n >> 1 // division by 2
```

The multi-language program on the right implements the exponentiation by squaring [10] for efficiently computing the powers of an integer number: It stores in `res` the `n`-th power of `a`. The binary operator `&` is the bitwise `and` operator and `>>` is the right shift operation.

Fig. 5. Multi-language exponentiation by squaring algorithm.

the single-languages through an *interoperability relation* $\ltimes$ on sorts. In particular, an interoperability constraint $s \ltimes s'$ implies that Sg_i-terms of sort s can be used in place of Sg_j-terms of sort s' (with $i, j \in \{1, 2\}$ and $i \neq j$). This determines the terms of the multi-language $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$.

Running Example 2. For instance, Num provides users with more advanced binary operators and values than those of Imp. However, the purpose of Num is limited to define handy mathematical functions (indeed, it is not even Turing-complete). We can take advantage of such mathematical expressiveness without sacrificing computational power by allowing the use of Num-expressions (that is, terms with sort exp_2) into Imp-programs, in place of Imp-expression of sort exp_1. Therefore, the interoperability relation shall simply specify $exp_2 \ltimes exp_1$, and we let NImp to be formally defined as $\mathsf{NImp} \triangleq (\mathsf{Imp}, \mathsf{Num}, \ltimes)$. As a result, programmers may write programs such as the one in Fig. 5, where terms in `magenta` are Num-expressions used in place of Imp-expressions (*i.e.*, we use colours rather than applying the conversion operator $\hookrightarrow_{exp_2, exp_1}$ for clarity reasons).

The multi-language SG-semantics $\mathscr{C}$ is then obtained by pairing the single-language algebras $\mathscr{C}_1$ and $\mathscr{C}_2$ with boundary functions $[\![s \ltimes s']\!]_{\mathscr{C}} : [\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j}$ that specify how terms of sort s in Sg_i can be interpreted as terms of sort s' in Sg_j. In other words, boundary functions regulate the flow of values between the underlying languages [31]. For instance, they can act as a bridge between different type representations in the underlying languages (*e.g.*, to enable the interoperability of Java and JavaScript in Nashorn [33] or the interoperability of Java and Kotlin [25]), or deal between different machine-integers implementation (*e.g.*, the mapping between Java primitive types and C types in JNI [32]), etc.

Running Example 3. We denote by $\mathscr{D}$ the multi-language NImp-algebra defined by coupling $\mathscr{D}_1$ and $\mathscr{D}_2$ with the boundary function $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ defined below (recall Def. 6). Note that values of Num-expressions range over the set of rational numbers $\mathbb{Q}$, whereas Imp only handles integer values in $\mathbb{Z}$. A natural choice for the boundary function $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ of NImp that converts Num-expressions to Imp-expressions is to *truncate* the value of Num-expression to their nearest integer value. Since expressions only have values with respect to an environment, we shall specify a conversion from $\mathbb{E}\mathrm{nv}_2 \to \mathbb{Q}_\perp$ to $\mathbb{E}\mathrm{nv}_1 \to \mathbb{Z}_\perp$ (we use ρ_1 and ρ_2

as metavariables for $\mathbb{Env}_1$ and $\mathbb{Env}_2$, respectively):

$$[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}(e \in \mathbb{Env}_2 \to \mathbb{Q}_\bot) \triangleq \rho_1 \in \mathbb{Env}_1 \mapsto \psi(e(\phi(\rho_1))) \in \mathbb{Z}_\bot$$

where $\phi\colon \mathbb{Env}_1 \to \mathbb{Env}_2$ is the inclusion function and $\psi\colon \mathbb{Q}_\bot \to \mathbb{Z}_\bot$ truncates rational values, *i.e.*, $\psi(q) \triangleq \text{truncate}(q)$ if $q \neq \bot$ and $\psi(\bot) \triangleq \bot$. For instance, the semantics of the Num-expression $[\![\texttt{n / 2}]\!]_{\mathscr{D}_2} = \rho_2 \mapsto \rho_2(\mathsf{n}) \,/\, 2$ is mapped by $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ to the function $\rho_1 \mapsto \rho_1(\mathsf{n}) /_{\mathbb{Z}} 2$, where $/_{\mathbb{Z}}$ is the integer division.[5]

5.2 Combining Abstractions of Different Languages

The first step towards an abstract interpretation theory for multi-languages is to find a suitable notion of *multi-language collecting semantics*.

Definition 15 (Multi-Language Collecting Semantics). *Let $\mathscr{C}$ be a multi-language* SG*-algebra over the multi-language signature* $\mathsf{SG} \triangleq (\mathsf{Sg}_1, \mathsf{Sg}_2, \ltimes)$*. The* ***multi-language collecting semantics*** *$\mathscr{C}^*$ over $\mathscr{C}$ is specified by:*

- *the collecting* Sg_i*-semantics $\mathscr{C}_i^*$ over $\mathscr{C}_i$, for $i = 1, 2$; and*
- *boundary functions $[\![s \ltimes s']\!]_{\mathscr{C}^*} \triangleq \wp[\![s \ltimes s']\!]_{\mathscr{C}}$ for each $s \ltimes s'$.*

We can then lift Prop. 1 and 2 to the multi-language world in order to show that $\mathscr{C}^*$ has the desired properties:

Proposition 8. *Let $\mathscr{C}$ be a multi-language* SG*-algebra. The collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$ computes the strongest program property for each* multi-language *program* P *generated by* SG*, that is $[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \{[\![\mathsf{P}]\!]_{\mathscr{C}}\}$. Moreover, the singleton function $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$ is a multi-language* SG*-homomorphism.*

We are now interested in whether we can obtain a fixpoint definition of the multi-language collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$. At a minimum, a fixpoint definition of the two underlying language semantics is needed, since every single-language program is also a multi-language one. However, the semantics of the multi-language does not only depend on these specifications (that is, it is not a *universal property* of the underlying semantics) but is determined up to a family of boundary functions defining the interoperability of Sg_1 and Sg_2.

Theorem 4. *Let $\mathscr{C}$ be a multi-language* SG*-algebra whose boundary functions admit a constructive definition, that is $[\![s \ltimes s']\!]_{\mathscr{C}} \triangleq \operatorname{lfp} F_{s \ltimes s'}$ for each $s \ltimes s'$ in* SG *and for some $F_{s \ltimes s'}\colon ([\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j}) \to ([\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j})$. Then, the multi-language collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$ admits a fixpoint computation if and only if $\mathscr{C}_1$ and $\mathscr{C}_2$ does.*

Proof (Sketch). Each operator in SG admits a fixpoint definition.

The second step is to combine the already existing abstractions of the underlying languages. Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the Sg_i-algebras providing the abstract semantics of Sg_i, with $i \triangleq 1, 2$, and $\gamma_i\colon A_i \to \wp(D_i)$ their concretisation functions, respectively. We can blend $\mathscr{A}_1$ and $\mathscr{A}_2$ into the multi-language SG-algebra $\mathscr{A}$ by defining an abstract semantics of the conversion operators $[\![s \ltimes s']\!]_{\mathscr{A}}$, for each $s \ltimes s'$. We call such an $\mathscr{A}$ an **abstract** multi-language algebra.

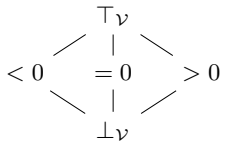


Fig. 6. Sign abstract domain.

$$
\begin{array}{ll}
\widetilde{\gamma}_1(\top_{\mathcal{V}}) \triangleq \mathbb{Z} & \widetilde{\gamma}_2(\top_{\mathcal{V}}) \triangleq \mathbb{Q} \\
\widetilde{\gamma}_1(< 0) \triangleq \{\, v \in \mathbb{Z} \mid v < 0 \,\} & \widetilde{\gamma}_2(< 0) \triangleq \{\, v \in \mathbb{Q} \mid v < 0 \,\} \\
\widetilde{\gamma}_1(> 0) \triangleq \{\, v \in \mathbb{Z} \mid v > 0 \,\} & \widetilde{\gamma}_2(> 0) \triangleq \{\, v \in \mathbb{Q} \mid v > 0 \,\} \\
\widetilde{\gamma}_1(= 0) \triangleq \{0\} & \widetilde{\gamma}_2(= 0) \triangleq \{0\} \\
\widetilde{\gamma}_1(\bot_{\mathcal{V}}) \triangleq \{\bot\} & \widetilde{\gamma}_2(\bot_{\mathcal{V}}) \triangleq \{\bot\}
\end{array}
$$

Fig. 7. Concretisation functions $\widetilde{\gamma}_i \colon A_{\mathcal{V}} \to \wp(\mathbb{V}_i)$.

Running Example 4. Let $\langle A_{\mathcal{V}}, \sqsubseteq_{\mathcal{V}}, \sqcup_{\mathcal{V}}, \sqcap_{\mathcal{V}}, \bot_{\mathcal{V}}, \top_{\mathcal{V}} \rangle$ be the standard sign domain (Fig. 6) and $\widetilde{\gamma}_i \colon A_{\mathcal{V}} \to \wp(\mathbb{V}_i)$ (where $\mathbb{V}_1 \triangleq \mathbb{Z}_\bot$ and $\mathbb{V}_2 \triangleq \mathbb{Q}_\bot$) the corresponding concretisation function (Fig. 7). We let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the abstract algebras defining a sign analysis for languages Imp and Num, respectively (that is, the computation induced by $\mathscr{A}_i$ is carried out using abstract values in $A_{\mathcal{V}}$ instead of concrete ones in $\mathbb{V}_i$). The abstract semantics $\mathscr{A}_i$ is the standard one (see, for instance, [38]), and it is reported in Appx. B for completeness.

The concretisation of abstract environments $\mathbb{E}\mathrm{nv}^\sharp \triangleq \mathbb{X} \to A_{\mathcal{V}}$ is defined by $\mathring{\gamma}_i(\rho^\sharp \in \mathbb{E}\mathrm{nv}^\sharp) \triangleq \{\, \rho_i \in \mathbb{E}\mathrm{nv}_i \mid \forall x \in \mathbb{X} \,.\, \rho_i(x) \in \widetilde{\gamma}_i(\rho^\sharp(x)) \,\}$. The abstract interpretation of an expression E in Imp or Num is a function $e^\sharp \in [\![exp_i]\!]_{\mathscr{A}_i} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to A_{\mathcal{V}}$ that takes abstract environments to abstract values. Similarly, the abstract interpretation $c^\sharp \in [\![com_i]\!]_{\mathscr{A}_i} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to \mathbb{E}\mathrm{nv}^\sharp$ of a command C is a transformation of abstract environments. The concretisations of $e^\sharp$ and $c^\sharp$ are therefore (sorted) functions $(\gamma_i)_{exp_i} \colon [\![exp_i]\!]_{\mathscr{A}_i} \to [\![exp_i]\!]_{\mathscr{D}_i^*}$ and $(\gamma_i)_{com_i} \colon [\![com_i]\!]_{\mathscr{A}_i} \to [\![com_i]\!]_{\mathscr{D}_i^*}$ from the carrier sets of $\mathscr{A}_i$ to those of the collecting semantics $\mathscr{D}_i^*$:

$$
\begin{aligned}
(\gamma_i)_{exp_i}(e^\sharp) &\triangleq \{\, e_i \in [\![exp_i]\!]_{\mathscr{D}_i} \mid \forall \rho^\sharp \in \mathbb{E}\mathrm{nv}^\sharp \,.\, \forall \rho_i \in \mathring{\gamma}_i(\rho^\sharp) \,.\, e_i(\rho_i) \in \widetilde{\gamma}_i(e^\sharp(\rho^\sharp)) \,\} \\
(\gamma_i)_{com_i}(c^\sharp) &\triangleq \{\, c_i \in [\![com_i]\!]_{\mathscr{D}_i} \mid \forall \rho^\sharp \in \mathbb{E}\mathrm{nv}^\sharp \,.\, \forall \rho_i \in \mathring{\gamma}_i(\rho^\sharp) \,.\, c_i(\rho_i) \in \mathring{\gamma}_i(c^\sharp(\rho^\sharp)) \,\}
\end{aligned}
$$

The following theorems aim to show that all the properties of interest of the resulting multi-language abstraction rely entirely on the abstract semantics of the boundary functions. We recall that when we write $s \ltimes s'$ we implicitly assume that s is a sort of Sg_i and s' one of Sg_j for $i, j \in \{1, 2\}$ and $i \neq j$.

Theorem 5 (Soudness). *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be sound Sg_i-algebras with concretisation functions $\gamma_i \colon A_i \to \wp(C_i)$, for $i \triangleq 1, 2$. If $[\![s \ltimes s']\!]_{\mathscr{C}^*} \circ \gamma_i \subseteq \gamma_j \circ [\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$, then the multi-language abstract semantics $\mathscr{A}$ is sound, that is $[\![\mathsf{P}]\!]_{\mathscr{C}^*} \subseteq \gamma [\![\mathsf{P}]\!]_{\mathscr{A}}$ for each multi-language program P generated by SG.*

The derived abstraction $\mathscr{A}$ of the multi-language semantics preserves the completeness of single-language operators.

Theorem 6 (Completeness). *Let $\mathscr{A}$ be the multi-language abstract semantics. If the order-sorted Sg_i-algebra $\mathscr{A}_i$ is forward (resp., backward) complete with respect to $k \colon s$ and $f \colon w \to s$ in Sg_i, then $\mathscr{A}$ is forward (resp., backward) complete with respect $k_i \colon s$ and $f_i \colon w \to s$ in SG, respectively.*

[5] We ignore the case where $\rho_2(\mathsf{n}) = \bot$, as it is clearly trivial.

Step	0	1	2
res	(> 0)	(> 0)	(> 0)
a	(> 0)	(> 0)	(> 0)
n	(> 0)	$\top_{\mathcal{V}}$	$\top_{\mathcal{V}}$

The abstract interpretation of the program in Fig. 5 guarantees the result of a^n, with $a, n > 0$, to be positive. The imprecision on the final abstract value of the variable n is due to poor choice of the abstract domain.

Fig. 8. The abstract iterates of the loop in Fig. 5.

Then, complete boundary functions do not alter the completeness of complete programs as a corollary:

Corollary 1. *Let $\mathscr{A}$ be the multi-language abstract semantics and $[\![s \ltimes s']\!]_{\mathscr{A}}$ a forward (resp., backward) complete boundary functions. For each multi-language program* P *sorted by s, if* P *is forward (resp., backward) complete, then so too is the program $\hookrightarrow_{s \ltimes s'}(\mathsf{P})$.*

Equivalent multi-language versions of Props. 6 and 7 also apply. The proof boils down to the fact that multi-language homomorphisms are pair of order-sorted homomorphisms that also commute with boundary functions.

Running Example 5. The multi-language abstract algebra $\mathscr{A}$ is obtained by combining $\mathscr{A}_1$ and $\mathscr{A}_2$ with an abstraction of the boundary function $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ defined in Ex. 3. Since the underlying algebras share the same abstract domain for expressions, that is $[\![exp_1]\!]_{\mathscr{A}_1} = [\![exp_2]\!]_{\mathscr{A}_2}$, there is no need to convert abstract values between two identical domains, therefore we set $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{A}} = id$.

Let us show the computation of the abstract semantics of the multi-language program in Fig. 5, starting from the set of initial states in which both a and n are greater than 0. The abstract precondition before entering the loop is given by the abstract environment $\{\mathsf{a} \mapsto (> 0), \mathsf{n} \mapsto (> 0), \mathsf{res} \mapsto (> 0)\}$, where res is positive due to the assignment on line 1. The abstract iterates of the loop converge in three steps, as shown in Fig. 8. Since the example trivially satisfies the hypotheses of Thm. 5, the result is sound.

6 Related Works

Cross-language interoperability is a popular field of research which has been driven more by practical needs than by theoretical questions. Several works focus on the implementation of runtime mechanisms for interoperability. Non-exhaustive examples are [22], defining a type system for the Microsoft Intermediate Language (IL) employed by the .NET to interoperate underlying languages (*e.g.*, C#, Visual Basic, VBScript, *etc.*). [24] designs a virtual machine able to interoperate with dynamically typed programming languages (Ruby and JavaScript) with a statically typed one (C). [4] describes a multi-language runtime mechanism obtained by blending single-language interpreters of Python

and Prolog. More examples can be found in [31]. On the other hand, various works [37,5,23,35] focus on specific theoretical problems arising from language interoperability, such as typing issues and value exchanging techniques. To the best of our knowledge, the first paper addressing the problem of formal reasoning in a multi-language context has been [31]. The authors introduced the notion of *boundary functions* as language constructs that move values between the underlying languages (ML and Scheme), ensuring their interoperability. Buro and Mastroeni [7] generalises such an approach in a language independent way, extending the construction to the broader class of order-sorted algebras. Finally, a few works concentrated on analysis-related aspects in a multi-language scenario. In the Java Native Interface context, [40] proposes a specification language which extends the Java Virtual Machine Language with primitives that approximate C code that cannot be compiled into Java. [28] introduces Pungi, a system that transforms Python/C interface code to affine programs with the aim of correctly handling Python's heap when it interoperates with C++.

7 Discussion and Concluding Remarks

The lack of static analysis techniques for verifying multi-language programs is a major issue so long as modern software relies on heterogenous code. Current state of the art sees relatively few works [28,40] that solve context-specific tasks in the cross-language interoperability field. However, none of these works address the problem from a general, theoretical point of view.

In this paper, we applied abstract interpretation theory to the algebraic framework of multi-languages, providing a general technique for defining the abstract semantics of the combined language. The taken approach has the advantage of being independent both from the underlying languages and analyses, and, at the same time, guarantees theoretical properties of interest, *e.g.*, soundness and completeness of the abstraction. Moreover, we have shown that such properties rely crucially on the definition of the boundary functions, thus providing guidelines for defining their abstract semantics.

Further research should consider the *asymmetrical* lifting of a single-language analysis to a multi-language. In the previous section, we assumed the existence of two algebras, $\mathscr{A}_1$ and $\mathscr{A}_2$, providing the abstract semantics of the underlying languages. Then, the abstract semantics of boundary functions defines the flow of abstract values during the abstract computations. Even though our framework is general enough to allow such algebras to be different (*e.g.*, $\mathscr{A}_1$ may define a sign semantics whereas $\mathscr{A}_2$ provides an interval analysis), we do not discuss the case in which there exists only one anlaysis. It may be fruitful to investigate this asymmetrical situation, for instance in the case where one of the underlying languages cannot alter the values flowing from the other (see the *lump embedding* construction of [31]).

In addition, future studies must focus on practical aspects of implementation. The proof of Thm. 2 in [7] provides a recursive definition of homomorphisms out

of the multi-language term algebra, *i.e.*, semantic functions, that suggests there is a straightforward implementation of the multi-language abstract interpreter.

References

1. Ahmed, A., Blume, M.: An equivalence-preserving CPS translation via multi-language semantics. In: Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming. vol. 46, pp. 431–444. ACM, New York, NY, USA (2011)
2. Amato, G., Meo, M.C., Scozzari, F.: On collecting semantics for program analysis. Theoretical Computer Science (2020)
3. Arceri, V., Mastroeni, I.: Static program analysis for string manipulation languages. Electronic Proceedings in Theoretical Computer Science **299**, 19–33 (2019)
4. Barrett, E., Bolz, C.F., Tratt, L.: Approaches to interpreter composition. Computer Languages, Systems & Structures **44**, 199–217 (2015)
5. Benton, N.: Embedded interpreters. Journal of functional programming **15**(4), 503–542 (2005)
6. Bjørner, N., Gurfinkel, A.: Property directed polyhedral abstraction. In: International Workshop on Verification, Model Checking, and Abstract Interpretation. pp. 263–281. Springer (2015)
7. Buro, S., Mastroeni, I.: On the multi-language construction. In: European Symposium on Programming. pp. 293–321. Springer (2019)
8. Campbell, G., Papapetrou, P.P.: SonarQube in action. Manning Publications Co. (2013)
9. Chisnall, D.: The challenge of cross-language interoperability. Commun. ACM **56**(12), 50–56 (Dec 2013)
10. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of elliptic and hyperelliptic curve cryptography. CRC press (2005)
11. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theoretical Computer Science **277**(1-2), 47–103 (2002)
12. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 238–252 (1977)
13. Cousot, P., Cousot, R.: Abstract interpretation frameworks. Journal of logic and computation **2**(4), 511–547 (1992)
14. Cousot, P., Giacobazzi, R., Ranzato, F.: A^2i: abstract^2 interpretation. Proceedings of the ACM on Programming Languages **3**(POPL), 1–31 (2019)
15. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 84–96 (1978)
16. Furr, M., Foster, J.S.: Checking type safety of foreign function calls. SIGPLAN Not. **40**(6), 62–72 (Jun 2005)
17. Giacobazzi, R., Ranzato, F.: Completeness in abstract interpretation: A domain perspective. In: International Conference on Algebraic Methodology and Software Technology. pp. 231–245. Springer (1997)
18. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. Journal of the ACM (JACM) **47**(2), 361–416 (2000)

19. Goguen, J.A., Diaconescu, R.: An oxford survey of order sorted algebra. Mathematical Structures in Computer Science **4**(3), 363–392 (1994)
20. Goguen, J.A., Meseguer, J.: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theoretical Computer Science **105**(2), 217–273 (1992)
21. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. Journal of the ACM (JACM) **24**(1), 68–95 (1977)
22. Gordon, A.D., Syme, D.: Typing a multi-language intermediate code. In: Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001. pp. 248–260. ACM, New York, NY, USA (2001)
23. Gray, K.E.: Safe cross-language inheritance. In: Vitek, J. (ed.) ECOOP 2008 – Object-Oriented Programming. pp. 52–75. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
24. Grimmer, M., Schatz, R., Seaton, C., Würthinger, T., Luján, M.: Cross-language interoperability in a multi-language runtime. ACM Trans. Program. Lang. Syst. **40**(2), 8:1–8:43 (May 2018)
25. JetBrains: Calling java code from kotlin. https://kotlinlang.org/docs/reference/java-interop.html
26. Juneau, J., Baker, J., Wierzbicki, F., Soto, L., Ng, V.: The definitive guide to Jython: Python for the Java platform. Apress, Berkely, CA, USA, 1st edn. (2010)
27. Kochems, J., Ong, C.: Improved functional flow and reachability analyses using indexed linear tree grammars. In: 22nd International Conference on Rewriting Techniques and Applications (RTA'11). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2011)
28. Li, S., Tan, G.: Finding reference-counting errors in python/c programs with affine analysis. In: European Conference on Object-Oriented Programming. pp. 80–104. Springer (2014)
29. Liang, S.: Java Native Interface: Programmer's guide and reference. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1999)
30. Mastroeni, I., Pasqua, M.: Hyperhierarchy of semantics-a formal framework for hyperproperties verification. In: International Static Analysis Symposium. pp. 232–252. Springer (2017)
31. Matthews, J., Findler, R.B.: Operational semantics for multi-language programs. ACM Transactions on Programming Languages and Systems **31**(3), 12:1–12:44 (2009)
32. Oracle: Jni types and data structures. https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/types.html
33. Oracle: Nashorn user's guide. https://docs.oracle.com/en/java/javase/14/nashorn/introduction.html
34. Pasqua, M.: Hyper Static Analysis of Programs – An Abstract Interpretation-Based Framework for Hyperproperties Verification. Ph.D. thesis, University of Verona (2019)
35. Patterson, D., Perconti, J., Dimoulas, C., Ahmed, A.: Funtal: Reasonably mixing a functional language with assembly. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 495–509. ACM, New York, NY, USA (2017)
36. Perconti, J.T., Ahmed, A.: Verifying an open compiler using multi-language semantics. In: Proceedings of the 23rd European Symposium on Programming Languages and Systems. pp. 128–148. Springer, Berlin, DE (2014)

37. Ramsey, N.: ML module mania: A type-safe, separately compiled, extensible interpreter. Electron. Notes Theor. Comput. Sci. **148**(2), 181–209 (Mar 2006)
38. Rival, X., Yi, K.: Introduction to static analysis (2019)
39. Spoto, F., Jensen, T.: Class analyses as abstract interpretations of trace semantics. ACM Transactions on Programming Languages and Systems (TOPLAS) **25**(5), 578–630 (2003)
40. Tan, G., Morrisett, G.: Ilea: Inter-language analysis across Java and C. SIGPLAN Not. **42**(10), 39–56 (Oct 2007)
41. Tennent, R.D.: The denotational semantics of programming languages. Communications of the ACM **19**(8), 437–453 (1976)

A Proofs

Proposition 9. *$\mathscr{C}^*$ is a proper order-sorted Sg-algebra in the category $\mathbf{Alg}(\mathsf{Sg})$.*

Proof. Firstly, note that $[\![s]\!]_{\mathscr{C}^*} \subseteq [\![r]\!]_{\mathscr{C}^*}$ whenever $s \leq r$. Now, let $f\colon w_1 \to s$ and $f\colon w_2 \to r$ with $w_1 \leq w_2$ be subsort polymorphic in Sg. Therefore, $s \leq r$ by monotonicity condition 1. We need to prove that[6]

$$[\![s \leq r]\!]_{\mathscr{C}^*} \circ [\![f\colon w_1 \to s]\!]_{\mathscr{C}^*} = [\![f\colon w_2 \to r]\!]_{\mathscr{C}^*} \circ [\![w_1 \leq w_2]\!]_{\mathscr{C}^*}$$

By Def. 9, functoriality of $\wp$, and $[\![s \leq r]\!]_{\mathscr{C}^*} = \wp[\![s \leq r]\!]_{\mathscr{C}}$, it is equivalent to prove

$$\wp([\![s \leq r]\!]_{\mathscr{C}} \circ [\![f\colon w_1 \to s]\!]_{\mathscr{C}}) \circ \times = \wp([\![f\colon w_2 \to r]\!]_{\mathscr{C}} \circ [\![w_1 \leq w_2]\!]_{\mathscr{C}}) \circ \times$$

that holds since $\mathscr{C}$ is an Sg-algebra.

Proposition 1. *$[\![-]\!]_{\mathscr{C}^*}\colon \mathscr{T}_{\mathsf{Sg}} \to \mathscr{C}^*$ computes the strongest program property for each program $\mathtt{P}$ generated from Sg, that is $[\![\mathtt{P}]\!]_{\mathscr{C}^*} = \{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$.*

Proof (Sketch). By structural induction on $\mathtt{P}$.

Proposition 2. *The singleton function $\{-\}\colon C \to \wp(C)$ defined by $c \mapsto \{c\}$ is an Sg-homomorphism $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$, and therefore $\{-\} \circ [\![-]\!]_{\mathscr{C}} = [\![-]\!]_{\mathscr{C}^*}$.*

Proof. Let $s \leq r$ in Sg and $x \in [\![s]\!]_{\mathscr{C}}$. Then, $\{-\}_s(x) = \{x\} = \{-\}_r(x)$. Now, let $f\colon w \to s$ be a function symbol in Sg and let $x \in [\![w]\!]_{\mathscr{C}}$. Then,

$$\begin{aligned}(\{-\}_s \circ [\![f\colon w \to s]\!]_{\mathscr{C}})(x) &= \{[\![f\colon w \to s]\!]_{\mathscr{C}}(x)\} \\ &= (\wp[\![f\colon w \to s]\!]_{\mathscr{C}} \circ \times)(\{x\}) \\ &= ([\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \{-\}_w)(x)\end{aligned}$$

Finally, let $k\colon s$ be a constant in Sg. Then, $\{-\}_s \circ [\![k]\!]_{\mathscr{C}} = \{[\![k]\!]_{\mathscr{C}}\} = [\![k]\!]_{\mathscr{C}^*}$.

Lemma 1. *Let $\mathcal{D} \subseteq \wp(C)$ be a non-empty set of atoms of the form $\{x\}$ for some $x \in C$. Then, $\Upsilon^*\mathcal{D}$ exists if and only if $\Upsilon \cup \mathcal{D}$ exists, and when either one exists $\Upsilon^*\mathcal{D} = \{\Upsilon \cup \mathcal{D}\}$.*

Proof. ($\Longleftarrow$) Suppose $\hat{c} \triangleq \Upsilon \cup \mathcal{D}$ is exists. Then, $x \preccurlyeq \hat{c}$ for each $\{x\} \in \mathcal{D}$, and therefore $\{x\} \preccurlyeq^* \{\hat{c}\}$ by (1). Now, let $Y \in \wp(C)$ be such that $\{x\} \preccurlyeq^* Y$ for each $\{x\} \in \mathcal{D}$. If $Y = \{y\}$, we conclude $\hat{c} \preccurlyeq y$ and therefore $\{\hat{c}\} \preccurlyeq^* \{y\}$ by (1). Otherwise, if $Y \neq \{y\}$, then by (3) every atom is smaller than Y, including $\{\hat{c}\}$. Hence, $\Upsilon^*\mathcal{D} = \{\hat{c}\}$. ($\Longrightarrow$) Suppose $\Upsilon^*\mathcal{D}$ is exists, and let $X, Y \in \wp(C)$ be two distinct non-atomic sets (which exist since C is assumed not to be $\{\bot\}$). Note that by (3) they both are greater than every atom $\{x\} \in \mathcal{D}$. Since they are non-comparable according to $\preccurlyeq^*$, then $\Upsilon^*\mathcal{D}$ must be atomic for some $\hat{c} \in C$, and therefore $x \preccurlyeq \hat{c}$ by (1). Now, let $y \in C$ be such that $x \preccurlyeq y$ for every $\{x\} \in \mathcal{D}$. Then, $\{x\} \preccurlyeq^* \{y\}$ and therefore $\{\hat{c}\} \preccurlyeq^* \{y\}$ by (1). Hence, we conclude that $\hat{c} \preccurlyeq y$ and $\Upsilon \cup \mathcal{D} = \hat{c}$.

[6] $[\![s \leq r]\!]_{\mathscr{C}}$ denotes the inclusion function from $[\![s]\!]_{\mathscr{C}}$ to $[\![r]\!]_{\mathscr{C}}$, for any Sg-algebra $\mathscr{C}$.

Theorem 3 (Fixpoint Collecting Semantics). *The function* $\{-\}\colon C \to \wp(C)$ *satisfies the hypotheses (i), (ii), and (iii) of Thm. 2, hence*

$$[\![\mathsf{P}]\!]_{\mathscr{C}^*} \overset{\text{Prop. 1}}{=} \{[\![\mathsf{P}]\!]_{\mathscr{C}}\} \overset{\text{Hypo.}}{=} \{\mathrm{lfp}^{\preccurlyeq}_{\bot} F\} \overset{\text{Thm. 2}}{=} \mathrm{lfp}^{\preccurlyeq^*}_{\bot^*} F^*$$

Proof. (i) follows from $\bot^* = \{\bot\}$, (iii) is a consequence of Proposition 3, and (ii) holds by $(F^* \circ \{-\})(c) = \{F(c)\} = (\{-\} \circ F)(c)$ for each $c \in C$.

Proposition 4. *If* $\mathscr{A}$ *is sound, then* $[\![\mathsf{P}]\!]_{\mathscr{C}^*} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ *for each program* P.

Proof. By structural induction on P:

- Let $\mathsf{P} \triangleq k$ for some $k\colon s$ in Sg. Then, it follows directly by Def. 11.
- Let $\mathsf{P} \triangleq f(\mathsf{P}_1, \ldots, \mathsf{P}_n)$. By regularity, there are $w \triangleq s_1, \ldots, s_n$ and s for which $f\colon w \to s$ and such that (w, s) is the small rank with respect to $w_0 \triangleq ls(\mathsf{P}_1), \ldots, ls(\mathsf{P}_n)$. Since P is well-formed, it follows that $\mathsf{P}_i\colon s_i$ for each $1 \leq i \leq n$ and $\mathsf{P}\colon s$. Then,

$$\begin{aligned}\gamma[\![f(\mathsf{P}_1, \ldots, \mathsf{P}_n)]\!]_{\mathscr{A}} &= (\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}})([\![\mathsf{P}_1]\!]_{\mathscr{A}}, \ldots, [\![\mathsf{P}_n]\!]_{\mathscr{A}}) \\ &\supseteq ([\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma)([\![\mathsf{P}_1]\!]_{\mathscr{A}}, \ldots, [\![\mathsf{P}_n]\!]_{\mathscr{A}})\end{aligned}$$

  and since $[\![f\colon w \to s]\!]_{\mathscr{C}^*}$ is monotone, then by induction hypothesis

$$\begin{aligned}&\supseteq [\![f\colon w \to s]\!]_{\mathscr{C}^*}([\![\mathsf{P}_1]\!]_{\mathscr{C}^*}, \ldots, [\![\mathsf{P}_n]\!]_{\mathscr{C}^*}) \\ &= [\![f(\mathsf{P}_1, \ldots, \mathsf{P}_n)]\!]_{\mathscr{C}^*}\end{aligned}$$

Proposition 5. $\mathscr{A}^\diamond$ *soundly approximates the concrete semantics. Moreover,* $\mathscr{A}^\diamond$ *is the most precise abstraction with respect to* (α, γ)*, that is, for any other sound algebra* $\mathscr{A}$*,* $\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ *and* $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ *for each operator in* Sg*. Therefore, by Prop. 4,* $\gamma[\![\mathsf{P}]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ *for each program* P.

Proof. We first prove that $\mathscr{A}^\diamond$ is sound. First condition of Def. 11 follows by extensivity of $\gamma \circ \alpha$ and monotonicity of $[\![f\colon w \to s]\!]_{\mathscr{C}^*}$, and the second one follows by (α, γ) being a Galois connection. Proving that $\mathscr{A}^\diamond$ is the most precise abstraction with respect to (α, γ) simply means that (1) $\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ and (2) $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ for each operator in Sg.

(1) By the soundness of $\mathscr{A}$, $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$. Then, since (α, γ) is a Galois connection, $\alpha \circ [\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma \sqsubseteq [\![f\colon w \to s]\!]_{\mathscr{A}}$, that is $[\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \sqsubseteq [\![f\colon w \to s]\!]_{\mathscr{A}}$. The thesis follows by the monotonicity of γ.
(2) (The proof of $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ is similar.)

Note that $\gamma[\![\mathsf{P}]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ follows by a simple structural induction on P.

Proposition 6. *Let* $\alpha\colon \mathscr{C}^* \to \mathscr{A}$ *be an* Sg*-homomorphism. Then,* $\mathscr{A}$ *is complete with respect to each operator in* Sg*, and therefore* $\alpha[\![\mathsf{P}]\!]_{\mathscr{C}^*} = [\![\mathsf{P}]\!]_{\mathscr{A}}$ *for each program* P.

Proposition 7. *Let $\gamma\colon \mathscr{A} \to \mathscr{C}^*$ be an* Sg*-homomorphism. Then, $\mathscr{A}$ is forward complete with respect to each operator in* Sg*, and therefore $[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ for each program* P*.*

Proof. Both proofs follow by a simple structural induction on P, in the same style of Prop. 4.

Proposition 8. *Let $\mathscr{C}$ be a multi-language* SG*-algebra. The collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$ computes the strongest program property for each* multi-language *program* P *generated by* SG*, that is $[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \{[\![\mathsf{P}]\!]_{\mathscr{C}}\}$. Moreover, the singleton function $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$ is a multi-language* SG*-homomorphism.*

Proof. We first prove that $[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \{[\![\mathsf{P}]\!]_{\mathscr{C}}\}$ by structural induction on P:

– if $\mathsf{P} \triangleq k_i$ for some constant symbol $k\colon s$ in Sg_i, then

$$[\![k_i]\!]_{\mathscr{C}^*} = [\![k]\!]_{\mathscr{C}_i^*} \overset{\text{Prop. 1}}{=} \{[\![k]\!]_{\mathscr{C}_i}\} = \{[\![k_i]\!]_{\mathscr{C}}\}$$

– if $\mathsf{P} \triangleq f_i(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ for some function symbol $f\colon w \to s$ in Sg_i with arity n, then

$$\begin{aligned}
[\![f_i(\mathsf{P}_1, \ldots, \mathsf{P}_n)]\!]_{\mathscr{C}^*} &= [\![f_i]\!]_{\mathscr{C}^*}([\![\mathsf{P}_1]\!]_{\mathscr{C}^*}, \ldots, [\![\mathsf{P}_n]\!]_{\mathscr{C}^*}) \\
&= [\![f_i]\!]_{\mathscr{C}^*}(\{[\![\mathsf{P}_1]\!]_{\mathscr{C}}\}, \ldots, \{[\![\mathsf{P}_n]\!]_{\mathscr{C}}\}) && \triangleright \text{ by induction hypothesis} \\
&= [\![f]\!]_{\mathscr{C}_i^*}(\{[\![\mathsf{P}_1]\!]_{\mathscr{C}}\}, \ldots, \{[\![\mathsf{P}_n]\!]_{\mathscr{C}}\}) \\
&= \{[\![f]\!]_{\mathscr{C}_i}([\![\mathsf{P}_1]\!]_{\mathscr{C}}, \ldots, [\![\mathsf{P}_n]\!]_{\mathscr{C}})\} \\
&= \{[\![f_i(\mathsf{P}_1, \ldots, \mathsf{P}_n)]\!]_{\mathscr{C}}\}
\end{aligned}$$

– if $\mathsf{P} \triangleq \hookrightarrow_{s,s'}(\mathsf{P'})$ for some $s \ltimes s'$ in SG, then

$$\begin{aligned}
[\![\hookrightarrow_{s,s'}(\mathsf{P'})]\!]_{\mathscr{C}^*} &= [\![\hookrightarrow_{s,s'}]\!]_{\mathscr{C}^*}([\![\mathsf{P'}]\!]_{\mathscr{C}^*}) \\
&= [\![\hookrightarrow_{s,s'}]\!]_{\mathscr{C}^*}(\{[\![\mathsf{P'}]\!]_{\mathscr{C}}\}) && \triangleright \text{ by induction hypothesis} \\
&= \{[\![\hookrightarrow_{s,s'}]\!]_{\mathscr{C}}([\![\mathsf{P'}]\!]_{\mathscr{C}})\} \\
&= \{[\![\hookrightarrow_{s,s'}(\mathsf{P'})]\!]_{\mathscr{C}}\}
\end{aligned}$$

Now, the homomorphic nature of $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$ follows by Prop. 2 and by observing that $\{-\}$ trivially commutes with boundary functions.

Theorem 5 (Soudness). *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be sound Sg_i-algebras with concretisation functions $\gamma_i\colon A_i \to \wp(C_i)$, for $i \triangleq 1, 2$. If $[\![s \ltimes s']\!]_{\mathscr{C}^*} \circ \gamma_i \subseteq \gamma_j \circ [\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$, then the multi-language abstract semantics $\mathscr{A}$ is sound, that is $[\![\mathsf{P}]\!]_{\mathscr{C}^*} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ for each multi-language program P generated by* SG*.*

Proof. We first prove that for each constant $k_i\colon s$ and function symbol $f_i\colon w \to s$ in SG we have $[\![k_i]\!]_{\mathscr{C}^*} \subseteq \gamma_s[\![k]\!]_{\mathscr{A}}$ and $[\![f_i\colon w \to s]\!]_{\mathscr{C}^*} \subseteq \gamma_s[\![f_i\colon w \to s]\!]_{\mathscr{A}}$. Then, $[\![\mathsf{P}]\!]_{\mathscr{C}^*} \subseteq \gamma[\![\mathsf{P}]\!]_{\mathscr{A}}$ for each multi-language program P generated by SG follows by a simple structural induction, in the same style of the proof of Prop. 4. Let $\gamma_i\colon \wp(C_i) \to A_i$ be the concretisation function of $\mathscr{A}_i$,

$$\llbracket exp_1 \rrbracket_{\mathscr{D}_1} \triangleq \mathbb{E}\mathrm{nv}_1 \to \mathbb{Z}_\bot \qquad \llbracket exp_1 \rrbracket_{\mathscr{A}_1} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to A_{\mathcal{V}}$$

$$\begin{array}{rl} \llbracket i \rrbracket_{\mathscr{D}_1} \triangleq & \rho_1 \mapsto i \\ \llbracket x \rrbracket_{\mathscr{D}_1} \triangleq & \rho_1 \mapsto \rho_1(x) \\ \llbracket bop_\odot \rrbracket_{\mathscr{D}_1}(e_1, e_2) \triangleq & \rho_1 \mapsto e_1(\rho_1) \odot e_2(\rho_1) \end{array} \qquad \begin{array}{rl} \llbracket i \rrbracket_{\mathscr{A}_1} \triangleq & \rho^\sharp \mapsto \widetilde{\alpha}_1(\{i\}) \\ \llbracket x \rrbracket_{\mathscr{A}_1} \triangleq & \rho^\sharp \mapsto \rho^\sharp(x) \\ \llbracket bop_\odot \rrbracket_{\mathscr{A}_1}(e_1^\sharp, e_2^\sharp) \triangleq & \rho^\sharp \mapsto e_1^\sharp(\rho^\sharp) \odot^\sharp e_2^\sharp(\rho^\sharp) \end{array}$$

Fig. 9. Denotational and sign semantics of Imp expressions.

– $\llbracket k_i \rrbracket_{\mathscr{C}^*} = \llbracket k \rrbracket_{\mathscr{C}_i^*} \subseteq (\gamma_i)_s \llbracket k \rrbracket_{\mathscr{A}_i} = \gamma_s \llbracket k_i \rrbracket_{\mathscr{A}}$; and
–

$$\begin{aligned} \llbracket f_i \colon w \to s \rrbracket_{\mathscr{C}^*} \circ \gamma_w &= \llbracket f \colon w \to s \rrbracket_{\mathscr{C}_i^*} \circ (\gamma_i)_w \\ &\subseteq (\gamma_i)_s \circ \llbracket f \colon w \to s \rrbracket_{\mathscr{A}_i} \\ &= \gamma_s \circ \llbracket f_i \colon w \to s \rrbracket_{\mathscr{A}} \end{aligned}$$

Theorem 6 (Completeness). *Let $\mathscr{A}$ be the multi-language abstract semantics. If the order-sorted Sg_i-algebra $\mathscr{A}_i$ is forward (resp., backward) complete with respect to $k \colon s$ and $f \colon w \to s$ in Sg_i, then $\mathscr{A}$ is forward (resp., backward) complete with respect $k_i \colon s$ and $f_i \colon w \to s$ in SG, respectively.*

Proof. We only prove the forward completeness of the function symbol $f_i \colon w \to s$ in SG, the other cases are similar.

$$\begin{aligned} \llbracket f_i \colon w \to s \rrbracket_{\mathscr{C}^*} \circ \gamma_w &= \llbracket f \colon w \to s \rrbracket_{\mathscr{C}_i^*} \circ (\gamma_i)_w \\ &= (\gamma_i)_s \circ \llbracket f \colon w \to s \rrbracket_{\mathscr{A}_i} \\ &= \gamma_s \circ \llbracket f_i \colon w \to s \rrbracket_{\mathscr{A}} \end{aligned}$$

where $\gamma_i \colon \wp(C_i) \to A_i$ is the concretisation function of the abstract algebra $\mathscr{A}_i$.

B Concrete and Abstract Semantics of Imp and Num

We provide a thorough formalisation of the algebraic semantics mentioned in Sect. 5. In particular, we define the denotational semantics $\mathscr{D}_i$ and the sign semantics $\mathscr{A}_i$ for both languages Imp and Num.

Concrete and Abstract Semantics of Expressions Denotational and sign semantics of expressions are defined in Figs. 9 and 10. The carrier sets on which they are defined are $\llbracket exp_i \rrbracket_{\mathscr{D}_i} \triangleq \mathbb{E}\mathrm{nv}_i \to \mathbb{V}_i$ and $\llbracket exp_i \rrbracket_{\mathscr{A}_i} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to A_{\mathcal{V}}$, respectively. Note that there is an obvious abstraction function $\widetilde{\alpha}_i \colon \wp(\mathbb{V}_i) \to A_{\mathcal{V}}$ left adjoint to $\widetilde{\gamma}_i$ (Fig. 7) providing $\langle \wp(\mathbb{V}_i), \subseteq \rangle \underset{\widetilde{\alpha}_i}{\overset{\widetilde{\gamma}_i}{\leftrightarrows}} \langle A_{\mathcal{V}}, \sqsubseteq_{\mathcal{V}} \rangle$; and also note that we abuse notation and assume that $\odot$ and f_n are both syntactical symbols and functions over values, that is $\odot \colon \mathbb{Z}_\bot^2 \to \mathbb{Z}_\bot$ and $f_n \colon \mathbb{Q}_\bot^n \to \mathbb{Q}_\bot$. We denote by $\odot^\sharp \colon A_{\mathcal{V}}^2 \to A_{\mathcal{V}}$ and $f_n^\sharp \colon A_{\mathcal{V}}^n \to A_{\mathcal{V}}$ the sign semantics of $\odot$ and f_n, respectively.

$$[\![exp_2]\!]_{\mathscr{D}_2} \triangleq \mathbb{E}_{\mathbb{m}\mathbb{v}_2} \to \mathbb{Q}_\bot \qquad [\![exp_2]\!]_{\mathscr{A}_2} \triangleq \mathbb{E}_{\mathbb{m}\mathbb{v}^\sharp} \to A_{\mathcal{V}}$$

$$[\![q]\!]_{\mathscr{D}_2} \triangleq \rho_2 \mapsto q \qquad [\![q]\!]_{\mathscr{A}_2} \triangleq \rho^\sharp \mapsto \widetilde{\alpha}_2(\{q\})$$

$$[\![x]\!]_{\mathscr{D}_2} \triangleq \rho_2 \mapsto \rho_2(x) \qquad [\![x]\!]_{\mathscr{A}_2} \triangleq \rho^\sharp \mapsto \rho^\sharp(x)$$

$$[\![f_n]\!]_{\mathscr{D}_2}(e_1, e_2) \triangleq \rho_2 \mapsto f_n(e_1(\rho_2), e_2(\rho_2)) \qquad [\![f_n]\!]_{\mathscr{A}_2}(e_1^\sharp, e_2^\sharp) \triangleq \rho^\sharp \mapsto f_n^\sharp(e_1^\sharp(\rho^\sharp), e_2^\sharp(\rho^\sharp))$$

$$[\![?]\!]_{\mathscr{D}_2}(e_1, e_2, e_3) \triangleq \rho_2 \mapsto \begin{cases} \bot & e_1(\rho_2) = \bot \\ e_2(\rho_2) & e_1(\rho_2) \neq 0 \\ e_3(\rho_2) & e_1(\rho_2) = 0 \end{cases}$$

$$[\![?]\!]_{\mathscr{A}_2}(e_1^\sharp, e_2^\sharp, e_3^\sharp) \triangleq \rho^\sharp \mapsto \begin{cases} \bot_{\mathcal{V}} & e_1^\sharp(\rho^\sharp) = \bot_{\mathcal{V}} \\ e_2^\sharp(\rho^\sharp) & e_1^\sharp(\rho^\sharp) \in \{> 0, < 0\} \\ e_3^\sharp(\rho^\sharp) & e_1^\sharp(\rho^\sharp) = (= 0) \end{cases}$$

Fig. 10. Denotational and sign semantics of Num expressions.

$$[\![com_1]\!]_{\mathscr{D}_1} \triangleq \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot} \xrightarrow{\bot} \mathbb{E}_{\mathbb{m}\mathbb{v}_i^\bot}$$

$$[\![skip]\!]_{\mathscr{D}_1} \triangleq \rho \mapsto \rho$$

$$[\![assign_x]\!]_{\mathscr{D}_1}(e) \triangleq \rho \overset{\bot}{\longmapsto} \rho[x \leftarrow e(\rho)] \qquad [\![seq]\!]_{\mathscr{D}_1}(c_1, c_2) \triangleq \rho \mapsto (c_2 \circ c_1)(\rho)$$

$$[\![cond]\!]_{\mathscr{D}_1}(e, c_1, c_2) \triangleq \rho \overset{\bot}{\longmapsto} \begin{cases} c_1(\rho) & e(\rho) \neq 0 \\ c_2(\rho) & e(\rho) = 0 \end{cases} \qquad [\![loop]\!]_{\mathscr{D}_1}(e, c) \triangleq \mathrm{lfp}^{\overset{\circ}{\sqsubseteq}}_{\overset{\circ}{\bot}} F_{e,c}$$

$$F_{e,c}(f) = \rho \overset{\bot}{\longmapsto} \begin{cases} \rho & e(\rho) = 0 \\ f(c(\rho)) & e(\rho) \neq 0 \end{cases}$$

Fig. 11. Denotational semantics of Imp commands.

Concrete Semantics of Commands Denotational semantics of Imp-commands is depicted in Fig. 11. We omit the semantics of Num-commands since they are a subset of those of Imp. The interpretation domain of commands is the set of $\bot$-preserving functions $[\![com_1]\!]_{\mathscr{D}_1} \triangleq \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot} \xrightarrow{\bot} \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$. We use the notation $\overset{\bot}{\longmapsto}$ to impose the strictness condition when defining anonymous functions. Consider the flat ordering $\langle \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}, \sqsubseteq \rangle$ on environments where $\bot \sqsubseteq \rho$ for each $\rho \in \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$ and every pair $\rho, \rho' \in \mathbb{E}_{\mathbb{m}\mathbb{v}_1}$ of non-bottom environments cannot be ordered. We can then make $\mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot} \xrightarrow{\bot} \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$ into a pointed dcpo by defining the information ordering $f \overset{\circ}{\sqsubseteq} g$ if $f(\rho) \sqsubseteq g(\rho)$ for every $\rho \in \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$. The smallest element is $\overset{\circ}{\bot} = \rho \mapsto \bot$ and the lub operator of a directed set $D \subseteq \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot} \xrightarrow{\bot} \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$ is $\overset{\circ}{\sqcup} D = \rho \mapsto \sqcup\{ f(\rho) \mid f \in D \}$ where $\sqcup$ is the partially defined lub operator on $\mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$ (which is well-defined since D is directed).[7] Finally, note that every function $f \in \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot} \xrightarrow{\bot} \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}$ is trivially continuous, and so too is $F_{e,c}$ (and therefore it has a least fixpoint).

[7] Indeed, $\langle \mathbb{E}_{\mathbb{m}\mathbb{v}_1^\bot}, \sqsubseteq, \bot, \sqcup \rangle$ is a pointed dcpo.

$$[\![com_1]\!]_{\mathscr{A}_1} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to \mathbb{E}\mathrm{nv}^\sharp$$

$$[\![skip]\!]_{\mathscr{A}_1} \triangleq \rho^\sharp \mapsto \rho^\sharp$$

$$[\![assign_x]\!]_{\mathscr{A}_1}(e^\sharp) \triangleq \rho^\sharp \mapsto \rho^\sharp[x \leftarrow e^\sharp(\rho^\sharp)] \quad [\![seq]\!]_{\mathscr{A}_1}(c_1^\sharp, c_2^\sharp) \triangleq \rho^\sharp \mapsto (c_2^\sharp \circ c_1^\sharp)(\rho^\sharp)$$

$$[\![cond]\!]_{\mathscr{A}_1}(e^\sharp, c_1^\sharp, c_2^\sharp) \triangleq \rho^\sharp \mapsto \begin{cases} c_1^\sharp(\rho^\sharp) \mathbin{\mathring{\sqcup}_\mathcal{V}} c_2^\sharp(\rho^\sharp) & e^\sharp(\rho^\sharp) = \top_\mathcal{V} \\ c_2^\sharp(\rho^\sharp) & e^\sharp(\rho^\sharp) = (=0) \\ c_1^\sharp(\rho^\sharp) & e^\sharp(\rho^\sharp) \in \{(<0), (>0)\} \\ \mathring{\bot}_\mathcal{V} & e^\sharp(\rho^\sharp) = \bot_\mathcal{V} \end{cases}$$

$$[\![loop]\!]_{\mathscr{A}_1}(e^\sharp, c^\sharp) \triangleq \mathrm{lfp}^{\dot{\sqsubseteq}}_{\dot{\bot}} F^\sharp_{e^\sharp, c^\sharp}$$

$$F^\sharp_{e^\sharp, c^\sharp}(f^\sharp) = \rho^\sharp \mapsto \begin{cases} \rho^\sharp \mathbin{\mathring{\sqcup}_\mathcal{V}} f^\sharp(c^\sharp(\rho^\sharp)) & e^\sharp(\rho^\sharp) = \top_\mathcal{V} \\ \rho^\sharp & e^\sharp(\rho^\sharp) = (=0) \\ f^\sharp(c^\sharp(\rho^\sharp)) & e^\sharp(\rho^\sharp) \in \{(<0), (>0)\} \\ \mathring{\bot}_\mathcal{V} & e^\sharp(\rho^\sharp) = \bot_\mathcal{V} \end{cases}$$

Fig. 12. Sign semantics of Imp commands.

Sign Semantics of Commands For the same reasons in the previous paragraph, we just provide the sign semantics of Imp-commands. It is defined over the carrier set $[\![com_1]\!]_{\mathscr{A}_1} \triangleq \mathbb{E}\mathrm{nv}^\sharp \to \mathbb{E}\mathrm{nv}^\sharp$. The poset $\langle \mathbb{E}\mathrm{nv}^\sharp, \dot{\sqsubseteq}_\mathcal{V} \rangle$, where $\rho_0^\sharp \mathbin{\dot{\sqsubseteq}_\mathcal{V}} \rho_1^\sharp$ if $\rho_0^\sharp(x) \sqsubseteq \rho_1^\sharp(x)$ for each $x \in \mathbb{X}$, is trivially a complete lattice $\langle \mathbb{E}\mathrm{nv}^\sharp, \dot{\sqsubseteq}_\mathcal{V}, \mathring{\sqcup}_\mathcal{V}, \mathring{\sqcap}_\mathcal{V}, \mathring{\bot}_\mathcal{V}, \mathring{\top}_\mathcal{V} \rangle$. We can then lift such a posetal structure defined on $\mathbb{E}\mathrm{nv}^\sharp$ to the function space $\mathbb{E}\mathrm{nv}^\sharp \to \mathbb{E}\mathrm{nv}^\sharp$, thus making it a complete lattice $\langle \mathbb{E}\mathrm{nv}^\sharp \to \mathbb{E}\mathrm{nv}^\sharp, \widetilde{\sqsubseteq}_\mathcal{V}, \widetilde{\sqcup}_\mathcal{V}, \widetilde{\sqcap}_\mathcal{V}, \widetilde{\bot}_\mathcal{V}, \widetilde{\top}_\mathcal{V} \rangle$. In particular, the lub operator is $\widetilde{\sqcup}_\mathcal{V} S = \rho^\sharp \mapsto \mathring{\sqcup}_\mathcal{V} \{\, f^\sharp(\rho^\sharp) \mid f^\sharp \in S \,\}$, and the smallest and greatest element are $\widetilde{\bot}_\mathcal{V} = \rho^\sharp \mapsto \mathring{\bot}_\mathcal{V}$ and $\widetilde{\top}_\mathcal{V} = \rho^\sharp \mapsto \mathring{\top}_\mathcal{V}$, respectively. One can check that $F^\sharp_{e^\sharp, c^\sharp}(f^\sharp)$ is continuous, so that the *loop* semantics is well-defined.

C A Simple Imperative Language

We illustrate a simple imperative language Imp on which we define various kinds of semantics. Let $\mathbb{X}$ be a set of variables and $\mathbb{V}$ a set of scalar values with metavariables x and v, respectively. Variables and values occur in the language as terminal symbols, and for each production defining the syntax of the language (on the right), we introduce a corresponding algebraic operator (on the left), or a family

of operators when they are parametric on a subscript:

(v)	$\langle exp \rangle ::= v$	scalar values
(x)	$\langle exp \rangle ::= x$	variables
$(bop_{\odot})$	$\langle exp \rangle ::= \langle exp \rangle \odot \langle exp \rangle$	binary operations
$(skip)$	$\langle com \rangle ::= \texttt{skip}$	do-nothing
$(assign_x)$	$\langle com \rangle ::= \texttt{x = } \langle exp \rangle$	assignment
$(cond)$	$\langle com \rangle ::= \texttt{if } \langle exp \rangle \texttt{ then } \langle com \rangle \texttt{ else } \langle com \rangle$	conditional
$(loop)$	$\langle com \rangle ::= \texttt{while } \langle exp \rangle \texttt{ do } \langle com \rangle$	loop statement
(seq)	$\langle com \rangle ::= \langle com \rangle ; \langle com \rangle$	composition

where $\odot$ is a binary operator such as `+`, `-`, `*`, etc. We abuse notation and assume that $\odot$ denotes both a syntactical symbol of the language and a mathematical function $\odot \colon \mathbb{V}^2 \to \mathbb{V}$ over values. The rank of each algebraic operator can be inferred by the non-terminals appearing in the production rules; for instance, the operator *cond* is sorted as

$$cond \colon exp, com, com \to com$$

In the examples in the following sections, we often use the correspondence between algebraic and context-free terms. For instance, we may write the algebraic term $cond(bop_{>}(\texttt{x}, \texttt{0}), skip, assign_{\texttt{x}}(bop_{-}(\texttt{0}, \texttt{x})))$ in the less cumbersome context-free form `if x > 0 then skip else x = 0 - x`.

C.1 A Small-Step Operational Semantics

We define a small-step operational semantics $\mathscr{S}$ describing the program execution steps. The presentation provided here is purely algebraic, and therefore less intuitive than the traditional rule-based style. However, the algebraic framework allows to express many more kinds of semantics in the same formalism, thus favouring their comparison.

Expressions We treat expressions `E` as "atomic" terms that are fully evaluable into a scalar value in a single-step. Let $\mathbb{S}_{exp} \triangleq \{ \langle \texttt{E}, \rho \rangle \mid \texttt{E} \in [\![exp]\!]_{\mathscr{T}_{\mathsf{Imp}}} \wedge \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \}$ be the set of configurations where `E` is an expression and ρ an environment in $\mathbb{E}\mathbb{n}\mathbb{v} \triangleq \mathbb{X} \to \mathbb{V}$. The small-step semantics of expressions is given in Fig. 13. Intuitively, starting from an expression `E`, we build a set of pairs in $\wp(\mathbb{S}_{exp} \times \mathbb{V})$ representing the one-step evaluation of `E` in each environment ρ. More precisely, $\langle \texttt{E}, \rho \rangle \rightarrowtriangle v \in [\![\texttt{E}]\!]_{\mathscr{S}}$ simply means that `E` is evaluated into v in ρ. We write $[\![\texttt{E}]\!]^{\rho}_{\mathscr{S}}$ for denoting such v (unique by construction).

Remark 5. Note that from the small-step semantics $[\![\texttt{E}]\!]_{\mathscr{S}}$ of an expression `E`, we are able to recover the term `E`. Indeed, $[\![\texttt{E}]\!]_{\mathscr{S}} \neq \varnothing$ and if $\langle \texttt{E}_1, \rho_1 \rangle \rightarrowtriangle v_1$ and $\langle \texttt{E}_2, \rho_2 \rangle \rightarrowtriangle v_2$ are transitions (that is, pairs) in $[\![\texttt{E}]\!]_{\mathscr{S}}$, then $\texttt{E}_1 = \texttt{E} = \texttt{E}_2$ (this can be shown by a simple structural induction on `E`).

$$[\![exp]\!]_{\mathscr{S}} \triangleq \wp(\mathbb{S}_{exp} \times \mathbb{V})$$

$$\begin{aligned}
[\![v]\!]_{\mathscr{S}} &\triangleq \{\, \langle v, \rho\rangle \rightarrowtriangle v \mid \rho \in \mathbb{E}\mathrm{nv} \,\} \\
[\![x]\!]_{\mathscr{S}} &\triangleq \{\, \langle x, \rho\rangle \rightarrowtriangle \rho(x) \mid \rho \in \mathbb{E}\mathrm{nv} \,\} \\
[\![bop_{\odot}]\!]_{\mathscr{S}}([\![\mathtt{E}_1]\!]_{\mathscr{S}}, [\![\mathtt{E}_2]\!]_{\mathscr{S}}) &\triangleq \{\, \langle bop_{\odot}(\mathtt{E}_1, \mathtt{E}_2), \rho\rangle \rightarrowtriangle [\![\mathtt{E}_1]\!]^{\rho}_{\mathscr{S}} \odot [\![\mathtt{E}_2]\!]^{\rho}_{\mathscr{S}} \mid \rho \in \mathbb{E}\mathrm{nv} \,\}
\end{aligned}$$

Fig. 13. Small-step operational semantics of Imp expressions.

$$[\![com]\!]_{\mathscr{S}} \triangleq \wp(\mathbb{S}^2_{com})$$

$$\begin{aligned}
[\![skip]\!]_{\mathscr{S}} &\triangleq \{\, \langle skip, \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \,\} \\
[\![assign_x]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}) &\triangleq \{\, \langle assign_x(\mathtt{E}), \rho\rangle \rightarrowtriangle \langle \bot, \rho[x \leftarrowtail [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}}]\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \,\} \\
[\![cond]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}, [\![\mathtt{C}_1]\!]_{\mathscr{S}}, [\![\mathtt{C}_2]\!]_{\mathscr{S}}) &\triangleq \{\, \langle cond(\mathtt{E}, \mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_1, \rho\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} \neq \mathtt{0} \,\} \\
&\quad \cup \{\, \langle cond(\mathtt{E}, \mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_2, \rho\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} = \mathtt{0} \,\} \\
[\![loop]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}, [\![\mathtt{C}]\!]_{\mathscr{S}}) &\triangleq \{\, \langle loop(\mathtt{E}, \mathtt{C}), \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} = \mathtt{0} \,\} \\
&\quad \cup \{\, \langle loop(\mathtt{E}, \mathtt{C}), \rho\rangle \rightarrowtriangle \langle seq(\mathtt{C}, loop(\mathtt{E}, \mathtt{C})), \rho\rangle \\
&\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} \neq \mathtt{0} \,\} \\
[\![seq]\!]_{\mathscr{S}}([\![\mathtt{C}_1]\!]_{\mathscr{S}}, [\![\mathtt{C}_2]\!]_{\mathscr{S}}) &\triangleq \{\, \langle seq(\mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_2, \rho'\rangle \\
&\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{C}_1]\!]^{\rho}_{\mathscr{S}} = \langle \bot, \rho'\rangle \,\} \\
&\quad \cup \{\, \langle seq(\mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle seq(\mathtt{C}'_1, \mathtt{C}_2), \rho'\rangle \\
&\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge [\![\mathtt{C}_1]\!]^{\rho}_{\mathscr{S}} = \langle \mathtt{C}'_1, \rho'\rangle \,\}
\end{aligned}$$

Fig. 14. Small-step operational semantics of Imp commands.

Remark 6. There are some missing cases in the definition of the interpretation functions for the operators in Fig. 13. For instance, we have defined $[\![bop_{\odot}]\!]_{\mathscr{S}}$ on arguments $[\![\mathtt{E}_1]\!]_{\mathscr{S}}$ and $[\![\mathtt{E}_2]\!]_{\mathscr{S}}$. However, there are semantic elements in $\wp(\mathbb{S}_{exp} \times \mathbb{V})$ that are not the image of any expressions $\mathtt{E}$ (*e.g.*, the empty set $\varnothing$). We shall leave implicit that $[\![bop_{\odot}]\!]_{\mathscr{S}}(e_1, e_2) \triangleq \varnothing$ whenever there are no $\mathtt{E}_1$ or $\mathtt{E}_2$ such that $e_1 = [\![\mathtt{E}_1]\!]_{\mathscr{S}}$ and $e_2 = [\![\mathtt{E}_2]\!]_{\mathscr{S}}$. (This remark and Rem. 5 shall also apply to the next definitions.)

Commands Let $\mathbb{S}_{com} \triangleq \{\, \langle \mathtt{C}, \rho\rangle \mid \mathtt{C} \in [\![com]\!]_{\mathscr{T}_{\mathsf{Imp}}} \cup \{\bot\} \wedge \rho \in \mathbb{E}\mathrm{nv} \,\}$ where $\mathtt{C}$ is a command (or $\bot$, denoting the end of a computation) and ρ an environment. For each command operator of Imp we define its semantics by specifying exactly the pairs of configurations which are related by the action of such an operator (Fig. 14). We write $[\![\mathtt{C}]\!]^{\rho}_{\mathscr{S}}$ for the unique $\langle \mathtt{C}', \rho'\rangle$ such that $\langle \mathtt{C}, \rho\rangle \rightarrowtriangle \langle \mathtt{C}', \rho'\rangle \in [\![\mathtt{C}]\!]_{\mathscr{S}}$.

Example 3. We show a small example of the application of the newly defined semantics $[\![-]\!]_{\mathscr{S}}$. We adopt the more intuitive notation provided by the context-free grammar for denoting terms, and we avoid the use of subscripts $_{\mathscr{S}}$. Suppose we want to compute the small-step semantics of the conditional statement

`if x > 0 then skip else x = 0 - x`. Then,

$$[\![\texttt{if x > 0 then skip else x = 0 - x}]\!] = [\![\mathit{cond}]\!]([\![\texttt{x > 0}]\!], [\![\texttt{skip}]\!], [\![\texttt{x = 0 - x}]\!])$$

where the semantics of the condition is

$$\begin{aligned}[\![\texttt{x > 0}]\!] = [\![\mathit{bop}_{>}]\!]([\![\texttt{x}]\!], [\![\texttt{0}]\!]) = &\{\, \langle \texttt{x > 0}, \rho\rangle \rightarrowtriangle \texttt{1} \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{1} \,\} \\ &\cup \{\, \langle \texttt{x > 0}, \rho\rangle \rightarrowtriangle \texttt{0} \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{0} \,\}\end{aligned}$$

and therefore,

$$\begin{aligned}&[\![\mathit{cond}]\!]\,([\![\texttt{x > 0}]\!], [\![\texttt{skip}]\!], [\![\texttt{x = 0 - x}]\!]) = \\ &\quad = \{\, \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{skip}, \rho\rangle \\ &\qquad \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{1} \,\} \\ &\quad \cup \{\, \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{x = 0 - x}, \rho\rangle \\ &\qquad \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{0} \,\}\end{aligned}$$

Note that the same result would have been achieved with a traditional rule-based style for specifying small-step semantics.

C.2 Fixpoint Definition of Prefix Trace Semantics

Prefix trace semantics associates each program `P` with the set of all finite traces obtained by iterating an arbitrarily large number of times the small-step semantics $\mathscr{S}$ from $\langle \texttt{P}, \rho\rangle$, for each environment ρ.

Let $\mathbb{S}^*_{com} \triangleq \bigcup_{n\in\mathbb{N}} \mathbb{S}^n_{com}$ be the set of finite sequences of command configurations (that is, *finite traces*). A trace $\tau \in \mathbb{S}^n_{com}$ is denoted by $\langle \texttt{C}_1, \rho_1\rangle \rightarrowtriangle \cdots \rightarrowtriangle \langle \texttt{C}_n, \rho_n\rangle$. The prefix trace semantics $\mathscr{P}$ is defined by keeping the one-step evaluation semantics for expressions `E` (*i.e.*, $[\![\texttt{E}]\!]_{\mathscr{P}} \triangleq [\![\texttt{E}]\!]_{\mathscr{S}}$), and by defining the following fixpoint semantics for command operators $f\colon w \to s$ on the domain $\langle [\![\mathit{com}]\!]_{\mathscr{P}} \triangleq \wp(\mathbb{S}^*_{com}), \subseteq, \varnothing, \cup\rangle$:[8]

$$[\![f\colon w \to s]\!]_{\mathscr{P}}([\![\texttt{P}_1]\!]_{\mathscr{P}}, \ldots, [\![\texttt{P}_n]\!]_{\mathscr{P}}) \triangleq \mathrm{lfp}^{\subseteq}_{\varnothing} F_{f(\texttt{P}_1,\ldots,\texttt{P}_n)}$$

where $F_{f(\texttt{P}_1,\ldots,\texttt{P}_n)}\colon \wp(\mathbb{S}^*_{com}) \to \wp(\mathbb{S}^*_{com})$ is defined as

$$\begin{aligned}X \mapsto\ &\{\varepsilon\} \cup \{\, \langle f(\texttt{P}_1, \ldots, \texttt{P}_n), \rho\rangle \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \,\} \\ &\cup \{\, \tau \rightarrowtriangle \langle \texttt{C}, \rho\rangle \rightarrowtriangle \langle \texttt{C}', \rho'\rangle \in \mathbb{S}^*_{com} \mid \tau \rightarrowtriangle \langle \texttt{C}, \rho\rangle \in X \wedge \langle \texttt{C}', \rho'\rangle = [\![\texttt{C}]\!]^{\rho}_{\mathscr{S}} \,\}\end{aligned}$$

The constructive computation of $\mathrm{lfp}^{\subseteq}_{\varnothing} F_{f(\texttt{P}_1,\ldots,\texttt{P}_n)}$ is guaranteed by the Kleene's theorem ($F_{f(\texttt{P}_1,\ldots,\texttt{P}_n)}$ is continuous on the pointed dcpo $\langle \wp(\mathbb{S}^*_{com}), \subseteq, \varnothing, \cup\rangle$).

Example 4. We restate Ex. 3 for the prefix trace semantics $\mathscr{P}$ applied to the same term $\texttt{P} \triangleq$ `if x > 0 then skip else x = 0 - x`:

$$[\![\texttt{if x > 0 then skip else x = 0 - x}]\!] = \mathrm{lfp}^{\subseteq}_{\varnothing} F_{\texttt{P}}$$

[8] The trace semantics of the constant *skip* is trivially defined by $[\![\mathit{skip}]\!]_{\mathscr{P}} \triangleq \{\varepsilon\} \cup \{\, \langle \mathit{skip}, \rho\rangle \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \,\} \cup \{\, \langle \mathit{skip}, \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \mid \rho \in \mathbb{E}\mathbb{n}\mathbb{v} \,\}$.

where the iterates of $F_{\mathtt{P}}$ are

$$
\begin{aligned}
F_{\mathtt{P}}^0 &= \varnothing \\
F_{\mathtt{P}}^1 &= \{\varepsilon\} \cup \{\langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle\} \\
F_{\mathtt{P}}^2 &= \{\ \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{skip}, \rho\rangle \\
&\qquad\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{1}\ \} \\
&\quad \cup \{\ \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{x = 0 - x}, \rho\rangle \\
&\qquad\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{0}\ \} \\
&\quad \cup F_{\mathtt{P}}^1 \\
F_{\mathtt{P}}^3 &= \{\ \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{skip}, \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \\
&\qquad\qquad \mid \rho \in \mathbb{E}\mathrm{nv} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{1}\ \} \\
&\quad \cup \{\ \langle \texttt{if x > 0 then skip else x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \texttt{x = 0 - x}, \rho\rangle \rightarrowtriangle \\
&\qquad\qquad \rightarrowtriangle \langle \bot, \rho[\texttt{x} \leftarrowtail \texttt{-x}]\rangle \mid \rho \in \mathbb{E}\mathrm{nv} \wedge (\rho(\texttt{x}) > \texttt{0}) = \texttt{0}\ \} \\
&\quad \cup F_{\mathtt{P}}^1 \\
F_{\mathtt{P}}^{\delta>3} &= F_{\mathtt{P}}^3
\end{aligned}
$$

and therefore $[\![\mathtt{P}]\!]_{\mathscr{P}}$ is the union of the iterates.

C.3 Reachability Semantics as Abstraction of Trace Semantics

Reachability semantics aims at computing the set of states that a program P may reach during its execution. Such a set can be parametric on program points (that is, location) or it can be the union of all the environments reached in any point. We show that both of these versions can be obtained by abstracting the collecting semantics $\mathscr{P}^*$ over traces provided in the previous section.

Reachability on Program Points Let $\mathscr{R}$ be the reachability semantics that collects states per program point. Its carrier set of sort *com* is defined as $[\![com]\!]_{\mathscr{R}} \triangleq \wp(\mathbb{S}_{com})$, thus a command is interpreted as a set of configurations (where program code denotes locations). We show that $\mathscr{R}$ can be obtained by abstracting the collecting semantics $\mathscr{P}^*$ by establishing a Galois connection between their carrier sets:

$$\langle \wp(\wp(\mathbb{S}^*_{com}))\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \wp(\mathbb{S}_{com})$$

The abstraction function α maps a semantic property $\mathcal{X} \subseteq \wp(\mathbb{S}^*_{com})$ (*i.e.*, a set of sets of finite traces) to the set of states that appears in those traces:

$$\alpha(\mathcal{X}) \triangleq \{\ \langle \mathtt{C}, \rho\rangle \in \mathbb{S}_{com} \mid \exists \tau \in \cup\mathcal{X}\ :\ \exists \langle \mathtt{C}, \rho\rangle \in \tau\ \}$$

Conversely, the concretization function γ maps each set of states C to the set containing only those traces whose configurations are in C:

$$\gamma(C) \triangleq \{\ X \in \wp(\mathbb{S}^*_{com}) \mid \forall \tau \in X\ .\ \langle \mathtt{C}, \rho\rangle \in \tau \implies \langle \mathtt{C}, \rho\rangle \in C\ \}$$

Now, the definition of $\mathscr{R}$ follows by the existence of a best correct approximation, as shown in Sect. 4.

Reachability without Program Points The reachability semantics $\mathscr{R}_{\cup}$ forgets about program locations and simply collects the environments reached during the exectuion of a program. The carrier set of commands is defined as $[\![com]\!]_{\mathscr{R}_{\cup}} \triangleq \wp(\mathbb{E}\mathrm{nv})$. $\mathscr{R}_{\cup}$ can be obtained by abstracting the collecting semantics $\mathscr{P}^*$ over traces:

$$\alpha\big(\mathcal{X} \in \wp(\wp(\mathbb{S}^*_{com}))\big) \triangleq \{\, \rho \in \mathbb{E}\mathrm{nv} \mid \exists \tau \in \cup\mathcal{X} \,:\, \exists \langle \mathtt{c}, \rho\rangle \in \tau \,\}$$
$$\gamma\big(R \in \wp(\mathbb{E}\mathrm{nv})\big) \triangleq \{\, X \in \wp(\mathbb{S}^*_{com}) \mid \forall \tau \in X \,.\, \langle \mathtt{c}, \rho\rangle \in \tau \implies \rho \in R \,\}$$