On the Multi-Language Construction

Samuele Buro and Isabella Mastroeni

University of Verona, Department of Computer Science
Strada le Grazie 15, 37134 Verona, Italy
{samuele.buro,isabella.mastroeni}@univr.it

Abstract Modern software is no more developed in a single programming language. Instead, programmers tend to exploit *cross-language interoperability mechanisms* to combine code stemming from different languages, and thus yielding fully-fledged *multi-language programs*. Whilst this approach enables developers to benefit from the strengths of each single-language, on the other hand it complicates the semantics of such programs. Indeed, the resulting multi-language does not meet any of the semantics of the combined languages. In this paper, we broaden the *boundary functions*-based approach à la Matthews and Findler to propose an algebraic framework that provides a constructive mathematical notion of *multi-language* able to determine its *semantics*. The aim of this work is to overcome the lack of a formal method (resp., model) to design (resp., represent) a multi-language, regardless of the inherent nature of the underlying languages. We show that our construction ensures the uniqueness of the *semantic function* (i.e., the multi-language semantics induced by the combined languages) by proving the *initiality* of the term model (i.e., the abstract syntax of the multi-language) in its category.

Keywords: multi-language design · program semantics · interoperability.

1 Introduction

Two elementary arguments lie at the heart of the *multi-language paradigm*: the large availability of existing programming languages, along with a very high number of already written libraries, and software that, in general, needs to *interoperate*. Although there is consensus in claiming that there is no best programming language regardless of the context [4,8], it is equally true that many of them are conceived and designed in order to excel for specific tasks. Such examples are R for statistical and graphical computation, Perl for data wrangling, Assembly and C for low-level memory management, etc. *"Interoperability between languages has been a problem since the second programming language was invented"* [8], so it is hardly surprising that developers have focused on the design of *cross-language interoperability mechanisms*, enabling programmers to combine code written in different languages. In this sense, we speak of *multi-languages*.

The field of cross-language interoperability has been driven more by practical concerns than by theoretical questions. The current scenario sees several engines and frameworks [47,28,44,13,29] (among others) to mix programming languages but only [30] discusses the semantic issues related to the multi-language design from a theoretical perspective. Moreover, the existing interoperability mechanisms differ considerably not only from the viewpoint of the combined languages, but also in terms of the approach used to provide the interoperation. For instance, Nashorn [47] is a JavaScript interpreter written in Java to allow embedding JavaScript in Java applications. Such engineering design works in a similar fashion of *embedded interpreters* [41,40].[1] On the contrary, Java Native Interface (JNI) framework [29] enables the interoperation of Java with native code written in C, C++, or Assembly through external procedure calls between languages, mirroring the widespread mechanism of *foreign function interfaces (FFI)* [14], whereas theoretical papers follow the more elegant approach of *boundary functions* (or, for short, *boundaries*) in the style of Matthews and Findler's multi-language semantics [30]. Simply put, boundaries act as a gate between single-languages. When a value needs to flow on the other language, they perform a conversion so that it complies to the other language specifications.

The major issue concerning this new paradigm is that multi-language programs do not obey any of the semantics of the combined languages. As a consequence, any method of formal reasoning (such as static program analysis or verification) is neutralized by the absence of a semantics specification. In this paper, we propose an algebraic framework based on the mechanism of boundary functions [30] that unambiguously yields the syntax and the semantics of the multi-language regardless the combined languages.

The Lack of a Multi-Language Framework. The notion of *multi-language* is employed naively in several works in literature [37,2,49,14,21,36,35,30] to indicate the embedding of two programming languages into a new one, with its own syntax and semantics.

The most recurring way to design a multi-language is to exploit a mechanism (like embedded interpreters, FFI, or boundary functions) able to regulate both control flow and value conversion between the underlying languages [30], thus adequate to provide *cross-language interoperability* [8]. The full construction is usually carried out manually by language designers, which define the multi-language by reusing the formal specifications of the single-languages [36,37,2,30] and by applying the selected mechanism for achieving the interoperation. Inevitably, therefore, all these resulting multi-languages notably differ one from another.

These different ways to achieve a cross-language interoperation are all attributable to the lack of a formal description of multi-language that does not provide neither a method for language designers to conceive new multi-languages nor any guarantee on the correctness of such constructions.

The Proposed Framework: Roadmap and Contributions. Matthews and Findler [30] propose *boundary functions* as a way to regulate the flow of values be-

[1] Other popular engines that obey the embedded interpreters paradigm are Jython [28], JScript [44], and Rhino [13].

tween languages. They show their approach on different variants of the same multi-language obtained by mixing ML [33] and Scheme [9], representing two "syntactically sugared" versions of the simply-typed and untyped lambda calculi, respectively.

Rather than showing the embedding of two fixed languages, we extend their approach to the much broader class of *order-sorted algebras* [19] with the aim of providing a framework that works regardless of the inherent nature of the combined languages. There are a number of reasons to choose order-sorted algebras as the underlying framework for generalizing the multi-language construction. From the first formulation of *initial algebra semantics* [17], the algebraic approach to program semantics [16] has become a cornerstone in the theory of programming languages [27]. Order-sorted algebras provide a mathematical tool for representing formal systems as algebraic structures through a systematic use of the notion of *sort* and *subsort* to model different forms of polymorphism [19,18], a key aspect when dealing with multi-languages sharing operators among the single-languages. They were initially proposed to ensure a rigorous model-theoretic semantics for error handling, multiple inheritance, retracts, selectors for multiple constructors, polymorphism, and overloading. In the years, several uses [25,52,3,11,38,39,24,6] and different variants [51,45,43,38] have been proposed for order-sorted algebras, making them a solid starting point for the development of a new framework. In particular, results on *rewriting logic* [32] extend easily to the order-sorted case [31], thus facilitating a future extension of this paper towards the *operational semantics* world. Improvements of the order-sorted algebra framework have also been proposed to model languages together with their type systems [10] and to extend order-sorted specification with high-order functions [38] (see [48] and [18] for detailed surveys).

In this paper, we propose three different multi-language constructions according to the semantic properties of boundary functions. The first one models a general notion of multi-language that do not require any constraints on boundaries (Sect. 3). We argue that when such generality is superfluous, we can achieve a neater approach where boundary functions do not need to be annotated with sorts. Indeed, we show that when the cross-language conversion of a term does not depend on the sort at which the term is considered (i.e., when boundaries are *subsort polymorphic*) the framework is powerful enough to apply the correct conversion (Sect. 4.1). This last construction is an improvement of the original notion of boundaries in [30]. From a practical point of view, it allows programmers to avoid to explicitly deal with sorts when writing code, a non-trivial task that could introduce type cast bugs in real world languages. Finally, we provide a very specific notion of multi-language where no extra operator is added to the syntax (Sect. 4.2). This approach is particularly useful to extend a language in a modular fashion and ensuring the backward compatibility with "old" programs. For each one of these variants we prove an *initiality theorem*, which in turn ensures the uniqueness of the multi-language semantics and thereby legitimating the proposed framework. Moreover, we show that the framework guarantees a fundamental closure property on the construction: The resulting multi-language admits an order-sorted representation, i.e., it falls within the same formal model of the combined languages. Finally, we model the multi-language designed in [30] in order to show an instantiation of the framework (Sect. 6).

2 Background

All the algebraic background of the paper is firstly stated in [17,15,19]. We briefly introduce here the main definitions and results, and we illustrate them on a simple running example.

Given a *set of sorts* S, an *S-sorted set* A is a family of sets indexed by S, i.e., $A = \{ A_s \mid s \in S \}$. Similarly, an *$S$-sorted function* $f\colon A \to B$ is a family of functions $f = \{ f_s\colon A_s \to B_s \mid s \in S \}$. We stick to the convention of using s and w as metavariables for sorts in S and S^*, respectively, and we use the $\mathbb{blackboard}$ $\mathbb{bold}$ typeface to indicate a specific sort in S. In addition, if A is an S-sorted set and $w = s_1 \dots s_n \in S^+$, we denote by A_w the cartesian product $A_{s_1} \times \cdots \times A_{s_n}$. Likewise, if f is an S-sorted function and $a_i \in A_{s_i}$ for $i = 1, \dots, n$, then the function $f_w\colon A_w \to B_w$ is such that $f_w(a_1, \dots, a_n) = (f_{s_1}(a_1), \dots, f_{s_n}(a_n))$. Given $P \subseteq S$, the restriction of an S-sorted function f to P is denoted by $f\big|_P$ and it is the P-sorted function $f\big|_P = \{ f_s \mid s \in P \}$. Finally, if $g\colon A \to B$ is a function, we still use the symbol g to denote the *direct image map of g* (also called the *additive lift* of g), i.e., the function $g\colon \wp(A) \to \wp(B)$ such that $g(X) = \{ g(a) \in B \mid a \in X \}$. Analogously, if $\leq$ is a binary relation on a set A (with elements $a \in A$), we use the same relation symbol to denote its *pointwise extension*, i.e., we write $a_1 \dots a_n \leq a'_1 \dots a'_n$ for $a_1 \leq a'_1, \dots, a_n \leq a'_n$.

The basic notions underpinning the order-sorted algebra framework are the definitions of *signature*, that models symbols forming terms of the language, and *algebra*, that provides an algebraic meaning to symbols.

Definition 1 (Order-Sorted Signature). *An* order-sorted signature *is a triple $\langle S, \leq, \Sigma \rangle$, where S is a set of sorts, $\leq$ is a binary relation on S, and Σ is an $S^* \times S$-sorted set $\Sigma = \{ \Sigma_{w,s} \mid w \in S^* \wedge s \in S \}$, satisfying the following conditions:*

(1os) *$\langle S, \leq \rangle$ is a poset; and*
(2os) *$\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \leq w_2$ imply $s_1 \leq s_2$.*

If $\sigma \in \Sigma_{w,s}$ (or, $\sigma\colon w \to s$ and $\sigma\colon s$ when $w = \varepsilon$, as shorthands), we call σ an *operator* (*symbol*) or *function symbol*, w the *arity*, s the *sort*, and (w, s) the *rank* of σ; if $w = \varepsilon$, we say that σ is a *constant* (*symbol*). We name $\leq$ the *subsort relation* and Σ a *signature* when $\langle S, \leq \rangle$ is clear from the context. We abuse notation and write $\sigma \in \Sigma$ when $\sigma \in \bigcup_{w,s} \Sigma_{w,s}$.

Definition 2 (Order-Sorted Algebra). *An* order-sorted $\langle S, \leq, \Sigma \rangle$-algebra $\mathcal{A}$ *over an order-sorted signature $\langle S, \leq, \Sigma \rangle$ is an S-sorted set A of* interpretation domains *(or,* carrier sets *or* semantic domains*) $A = \{ A_s \mid s \in S \}$, together with* *interpretation functions* $[\![\sigma]\!]^{w,s}_{\mathcal{A}} \colon A_w \to A_s$ *(or, if* $w = \varepsilon$, $[\![\sigma]\!]^{\varepsilon,s}_{\mathcal{A}} \in A_s$*)*[2] *for each* $\sigma \in \Sigma_{w,s}$, *such that:*

(1oa) $s \leq s'$ *implies* $A_s \subseteq A_{s'}$*; and*
(2oa) $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ *and* $w_1 \leq w_2$ *imply that* $[\![\sigma]\!]^{w_1,s_1}_{\mathcal{A}}(a) = [\![\sigma]\!]^{w_2,s_2}_{\mathcal{A}}(a)$ *for each* $a \in A_{w_1}$.

An important property of signatures, related to polymorphism, is *regularity.* Its relevance lies in the possibility of linking each term to a unique least sort (see Proposition 2.10 in [19]).

Definition 3 (Regularity of an Order-Sorted Signature). *An order-sorted signature* $\langle S, \leq, \Sigma \rangle$ *is* regular *if for each* $\sigma \in \Sigma_{\tilde{w},\tilde{s}}$ *and for each* lower bound $w_0 \leq \tilde{w}$ *the set* $\{\, (w, s) \mid \sigma \in \Sigma_{w,s} \land w_0 \leq w \,\}$ *has minimum. This minimum is called least rank of* σ *with respect to* w_0.

The freely generated algebra $\mathcal{T}_\Sigma$ over a given signature $\mathfrak{S} = \langle S, \leq, \Sigma \rangle$ provides the notion of *term* with respect to $\mathfrak{S}$.

Definition 4 (Order-Sorted Term Algebra). *Let* $\langle S, \leq, \Sigma \rangle$ *be an order-sorted signature. The* order-sorted term $\langle S, \leq, \Sigma \rangle$-algebra $\mathcal{T}_\Sigma$ *is an order-sorted algebra such that:*

– *The* S*-sorted set* $T_\Sigma = \{\, T_{\Sigma,s} \mid s \in S \,\}$ *is inductively defined as the least family satisfying:*
  (1ot) $\Sigma_{\varepsilon,s} \subseteq T_{\Sigma,s}$;
  (2ot) $s \leq s'$ *implies* $T_{\Sigma,s} \subseteq T_{\Sigma,s'}$*; and*
  (3ot) $\sigma \in \Sigma_{w,s}$, $w = s_1 \dots s_n \in S^+$, *and* $t_i \in T_{\Sigma,s_i}$ *for* $i = 1, \dots, n$ *imply* $\sigma(t_1 \dots t_n) \in T_{\Sigma,s}$.
– *For each* $\sigma \in \Sigma_{w,s}$ *the interpretation function* $[\![\sigma]\!]^{w,s}_{\mathcal{T}_\Sigma} \colon T_{\Sigma,w} \to T_{\Sigma,s}$ *is defined as*
  (4ot) $[\![\sigma]\!]^{\varepsilon,s}_{\mathcal{T}_\Sigma} = \sigma$ *if* $\sigma \in \Sigma_{\varepsilon,s}$*; and*
  (5ot) $[\![\sigma]\!]^{w,s}_{\mathcal{T}_\Sigma}(t_1, \dots, t_n) = \sigma(t_1 \dots t_n)$ *if* $\sigma \in \Sigma_{w,s}$, $w = s_1 \dots s_n \in S^+$, *and* $t_i \in T_{\Sigma,s_i}$ *for* $i = 1, \dots, n$.

Homomorphisms between algebras capture the *compositionality* nature of semantics: The meaning of a term is determined by the meanings of its constituents. They are defined as order-sorted functions that preserve the interpretation of operators.

Definition 5 (Order-Sorted Homomorphism). *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be* $\langle S, \leq, \Sigma \rangle$*-algebras. An* order-sorted $\langle S, \leq, \Sigma \rangle$-homomorphism *from* $\mathcal{A}$ *to* $\mathcal{B}$*, denoted by* $h \colon \mathcal{A} \to \mathcal{B}$*, is an* S*-sorted function* $h \colon A \to B = \{\, h_s \colon A_s \to B_s \mid s \in S \,\}$ *such that:*

(1oh) $h_s([\![\sigma]\!]^{w,s}_{\mathcal{A}}(a)) = [\![\sigma]\!]^{w,s}_{\mathcal{B}}(h_w(a))$ *for each* $\sigma \in \Sigma_{w,s}$ *and* $a \in A_w$*; and*
(2oh) $s \leq s'$ *implies* $h_s(a) = h_{s'}(a)$ *for each* $a \in A_s$.

[2] To be pedantic, we should introduce the *one-point domain* $A_\varepsilon = \{\, \bullet \,\}$ and then define $[\![\sigma]\!]^{\varepsilon,s}_{\mathcal{A}}(\bullet) \in A_s$.

$e \; ::= \; n \;\mid\; e \texttt{ + } e \quad \text{where } n \in \mathbb{N}$

(a) The BNF grammar of L_1.

$s \; ::= \; \texttt{-} \;\mid\; a \;\mid\; s \texttt{ + } s \quad \text{where } a \in \mathbb{A}$

(b) The BNF grammar of L_2.

Figure 1: The BNF grammars of the running example languages.

$$\begin{cases} [\![n]\!] = n \\ [\![e \texttt{ + } e']\!] = [\![e]\!] + [\![e']\!] \end{cases}$$

(a) The formal semantics of L_1.

$$\begin{cases} [\![\texttt{-}]\!] = \varepsilon \\ [\![a]\!] = a \\ [\![s \texttt{ + -}]\!] = [\![\texttt{- + } s]\!] = [\![s]\!] \\ [\![s \texttt{ + - + } s']\!] = [\![s \texttt{ + } s']\!] \\ [\![a_0 \texttt{ + } \dots \texttt{ + } a_n]\!] = a_0 \dots a_n \quad n > 0 \end{cases}$$

(b) The formal semantics of L_2.

Figure 2: The two formal semantics of the running example languages.

The class of all the order-sorted $\langle S, \leq, \Sigma \rangle$-algebras and the class of all order-sorted $\langle S, \leq, \Sigma \rangle$-homomorphisms form a category denote by $\mathbf{OSAlg}(S, \leq, \Sigma)$. Furthermore, the homomorphism definition determines the property of the term algebra $\mathcal{T}_\Sigma$ of being an *initial object* in its category whenever the signature is *regular*. Since *initiality* is preserved by isomorphisms, it allows to identify $\mathcal{T}_\Sigma$ with the *abstract syntax* of the language. If $\mathcal{T}_\Sigma$ is initial, the homomorphism leaving $\mathcal{T}_\Sigma$ and going to an algebra $\mathcal{A}$ is called the *semantic function* (with respect to $\mathcal{A}$).

Example. Let L_1 and L_2 be two formal languages (see Fig. 1). The former is a language to construct simple mathematical expressions: $n \in \mathbb{N}$ is the metavariable for natural numbers, while e inductively generates all the possible additions (Fig. 1a). The latter is a language to build strings over a finite alphabet of symbols $\mathbb{A} = \{ \texttt{a}, \texttt{b}, \dots, \texttt{z} \}$: $a \in \mathbb{A}$ is the metavariable for atoms (or, characters), whereas s concatenates them into strings (Fig. 1b). A term in L_1 and L_2 denotes an element in the sets $\mathbb{N}$ and $\mathbb{A}^*$, accordingly to equations in Fig. 2a and 2b, respectively.

The syntax of the language L_1 can be modeled by an order-sorted signature $\mathfrak{S}_1 = \langle S_1, \leq_1, \Sigma_1 \rangle$ defined as follows: $S_1 = \{ \mathbb{e}, \mathbb{n} \}$, a set with sorts $\mathbb{e}$ (stands for *expressions*) and $\mathbb{n}$ (stands for *natural numbers*); $\leq_1$ is the reflexive relation on S_1 plus $\mathbb{n} \leq_1 \mathbb{e}$ (natural numbers are expressions); and the operators in Σ_1 are $0, 1, 2, \dots : \mathbb{n}$ and $\texttt{+} \colon \mathbb{e}\,\mathbb{e} \to \mathbb{e}$. Similarly, the signature $\mathfrak{S}_2 = \langle S_2, \leq_2, \Sigma_2 \rangle$ models the syntax of the language L_2: the set $S_2 = \{ \mathbb{s}, \mathbb{a} \}$ carries the sort for *strings* $\mathbb{s}$ and the sort for *atomic symbols* (or, characters) $\mathbb{a}$; the subsort relation $\leq_2$ is the reflexive relation on S_2 plus $\mathbb{a} \leq_2 \mathbb{s}$ (characters are one-symbol strings); and the operator symbols in Σ_2 are $\texttt{a}, \dots, \texttt{z} \colon \mathbb{a}$, $\texttt{-} \colon \mathbb{s}$, and $\texttt{+} \colon \mathbb{s}\,\mathbb{s} \to \mathbb{s}$. Semantics of L_1 and L_2 can be embodied by algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ over the signatures $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively. We set the interpretation domains of $\mathcal{A}_1$ to $A^1_{\mathbb{n}} = A^1_{\mathbb{e}} = \mathbb{N}$ and those of $\mathcal{A}_2$ to $A^2_{\mathbb{a}} = \mathbb{A} \subseteq \mathbb{A}^* = A^2_{\mathbb{s}}$. Moreover, we define the interpretation functions as follows (the juxtaposition of two or more strings denotes their concatenation, and we use $\hat{a}$ as metavariable ranging over $\mathbb{A}^*$):

$$\begin{cases} [\![n]\!]^{\varepsilon,\mathbb{n}}_{\mathcal{A}_1} = n \\ [\![\texttt{+}]\!]^{\mathbb{e}\,\mathbb{e},\mathbb{e}}_{\mathcal{A}_1}(n_1, n_2) = n_1 + n_2 \end{cases} \qquad \begin{cases} [\![\texttt{-}]\!]^{\varepsilon,\mathbb{s}}_{\mathcal{A}_2} = \varepsilon \\ [\![a]\!]^{\varepsilon,\mathbb{a}}_{\mathcal{A}_2} = a \\ [\![\texttt{+}]\!]^{\mathbb{s}\,\mathbb{s},\mathbb{s}}_{\mathcal{A}_2}(\hat{a}_1, \hat{a}_2) = \hat{a}_1\hat{a}_2 \end{cases}$$

Since $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are regular, then $\mathcal{A}_1$ and $\mathcal{A}_2$ induce the semantic functions $h_1 \colon \mathcal{T}_{\Sigma_1} \to \mathcal{A}_1$ and $h_2 \colon \mathcal{T}_{\Sigma_2} \to \mathcal{A}_2$, providing semantics to the languages.

3 Combining Order-Sorted Theories

The first step towards a multi-language specification is the choice of which terms of one language can be employed in the others [36,30,35]. For instance, a multi-language requirement could demand to use ML expressions in place of Scheme expressions and, possibly, but not necessarily, vice versa (such a multi-language is designed in [30]). A *multi-language signature* is an amenable formalism to specify the compatibility relation between syntactic categories across two languages.

Definition 6 (Multi-Language Signature). *A* multi-language signature *is a triple $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$, where $\mathfrak{S}_1 = \langle S_1, \leq_1, \Sigma_1 \rangle$ and $\mathfrak{S}_2 = \langle S_2, \leq_2, \Sigma_2 \rangle$ are order-sorted signatures, and $\leq$ is a binary relation on $S = S_1 \cup S_2$, such that satisfies the following condition:*

(1s) $s, s' \in S_i$ implies $s \leq s'$ if and only if $s \leq_i s'$, for $i = 1, 2$.

To make the notation lighter, we introduce the following binary relations on S: $s \ltimes s'$ if $s \leq s'$ but neither $s \leq_1 s'$ nor $s \leq_2 s'$, and $s \preccurlyeq s'$ if $s \leq s'$ but not $s \ltimes s'$.

In the following, we always assume that the sets of sorts S_1 and S_2 of the order-sorted signatures $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are disjoint.[3] Condition (1s) requires the *multi-language subsort relation* $\leq$ to *preserve* the original subsort relations $\leq_1$ and $\leq_2$ (i.e., $\leq \cap\, S_i \times S_i = \leq_i$). The *join relation* $\ltimes$ provides a compatibility relation between sorts[4] in $\mathfrak{S}_1$ and $\mathfrak{S}_2$. More precisely, $S_i \ni s \ltimes s' \in S_j$ suggests that we want to use terms in $T_{\Sigma_i,s}$ in place of terms in $T_{\Sigma_j,s'}$, whereas the *intra-language subsort relation* $\preccurlyeq$ shifts the standard notion of subsort from the order-sorted to the multi-language world. In a nutshell, the relation $\leq\, = \,\preccurlyeq \cup \ltimes$ can only join (through $\ltimes$) the underlying languages without introducing distortions (indeed, $\preccurlyeq\, = \,\leq_1 \cup \leq_2$).

The role of an algebra is to provide an interpretation domain for each sort, as well as the meaning of every operator symbol in a given signature. When moving towards the multi-language context, the join relation $\ltimes$ may add subsort

[3] This hypothesis is non-restrictive: We can always perform a renaming of the sorts.

[4] Sorts may be understood as syntactic categories, in the sense of formal grammars. Given a context-free grammar G, it is possible to define a many-sorted signature Σ_G where non-terminals become sorts and such that each term t in the term algebra $\mathcal{T}_{\Sigma_G}$ is isomorphic to the parse tree of t with respect to G (see [15] for details).

constraints between sorts belonging to different signatures. Consequently, if $s \ltimes s'$, a multi-language algebra has to specify how values of sort s may be interpreted as values of sort s'. These specifications are called *boundary functions* [30] and provide an algebraic meaning to the subsort constraints added by $\ltimes$. Henceforth, we define $S = S_1 \cup S_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, and, given $(w, s) \in S_i^* \times S_i$, we denote by $\Sigma^i_{w,s}$ the (w, s)-sorted component in Σ_i.

Definition 7 (Multi-Language Algebra). *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be a multi-language signature. A* multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$ *is an S-sorted set A of* interpretation domains *(or,* carrier sets *or* semantic domains*) $A = \{ A_s \mid s \in S \}$, together with* interpretation functions $[\![\sigma]\!]^{w,s}_{\mathcal{A}} \colon A_w \to A_s$ *for each $\sigma \in \Sigma_{w,s}$, and with a $\ltimes$-sorted set α of* boundary functions $\alpha = \{ \alpha_{s,s'} \colon A_s \to A_{s'} \mid s \ltimes s' \}$, *such that the following constraint holds:*

(1a) the projected algebra $\mathcal{A}_i$, *where $i = 1, 2$, specified by the carrier set $A_i = \{ A^i_s = A_s \mid s \in S_i \}$ and interpretation functions $[\![\sigma]\!]^{w,s}_{\mathcal{A}_i} = [\![\sigma]\!]^{w,s}_{\mathcal{A}}$ for each $\sigma \in \Sigma^i_{w,s}$, must be an order-sorted $\mathfrak{S}_i$-algebra.*

If $\mathcal{M}$ is an algebra, we adopt the convention of denoting by M (standard math font) its carrier set and by μ (Greek math font) its boundary functions whenever possible. Condition (1a) is the semantic counterpart of condition (1s): It requires the multi-language to carry (i.e., preserve) the underlying languages order-sorted algebras, whereas the the boundary functions model how values can flow between languages.

Given two multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebras $\mathcal{A}$ and $\mathcal{B}$ we can define morphisms between them that preserve the sorted structure of the underlying projected algebras.

Definition 8 (Multi-Language Homomorphism). *Let $\mathcal{A}$ and $\mathcal{B}$ be multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebras with sets of boundary functions α and β, respectively. A* multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-homomorphism $h \colon \mathcal{A} \to \mathcal{B}$ *is an S-sorted function $h \colon A \to B$ such that:*

(1h) the restriction $h\big|_{S_i}$ is an order-sorted $\mathfrak{S}_i$-homomorphism $h\big|_{S_i} \colon \mathcal{A}_i \to \mathcal{B}_i$, for $i = 1, 2$; and
(2h) $s \ltimes s'$ implies $h_{s'} \circ \alpha_{s,s'} = \beta_{s,s'} \circ h_s$.

Conditions (1h) and (2h) are easily intelligible when the domain algebra is the abstract syntax of the language [15]: Simply put, both conditions require the semantics of a term to be a function of the meaning of its subterms, in the sense of [15,46]. In particular, the second condition demands that boundary functions act as operators.[5]

The identity homomorphism on a multi-language algebra $\mathcal{A}$ is denoted by $\mathrm{id}_{\mathcal{A}}$ and it is the set-theoretic identity on the carrier set A of the algebra $\mathcal{A}$. The composition of two homomorphisms $f \colon \mathcal{A} \to \mathcal{B}$ and $g \colon \mathcal{B} \to \mathcal{C}$ is defined as

[5] This is essential in order to generalize the concept of syntactical boundary functions of [30] to semantic-only functions in Sect. 4.2.

the sorted function composition $g \circ f \colon A \to C$, thus $\mathrm{id}_{\mathcal{A}} \circ f = f = f \circ \mathrm{id}_{\mathcal{B}}$ and associativity follows easily by the definition of $\circ$.

Proposition 1. *Multi-language homomorphisms are closed under composition.*

Hence, as in the many-sorted and order-sorted case [15,19], we have immediately the category of all the multi-language algebras over a multi-language signature:

Theorem 1. *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be a multi-language signature. The class of all $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebras and the class of all $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-homomorphisms form a category denoted by $\mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$.*

3.1 The Initial Term Model

In this section, we introduce the concepts of *(multi-language) term* and *(multi-language) semantics* in order to show how a multi-language algebra yields a unique interpretation for any *regular* (see Def. 11) multi-language specification.

Multi-language terms should comprise all of the underlying languages terms, plus those obtained by the merging of the two languages according to the join relation $\ltimes$. In particular, we aim for a construction where subterms of sort s' may have been replaced by terms of sort s, whenever $s \ltimes s'$ (we recall that s and s' are two syntactic categories of different languages due to Def. 6). Nonetheless, we must be careful not to add ambiguities during this process: A term t may belong to both $\mathfrak{S}_1$ and $\mathfrak{S}_2$ term algebras but with different meanings $[\![t]\!]_{\mathcal{A}_1}$ and $[\![t]\!]_{\mathcal{A}_2}$ (assuming that $\mathcal{A}_1$ and $\mathcal{A}_2$ are algebras over $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively). When t is included in the multi-language, we lose the information to determine which one of the two interpretations choose, thus making the (multi-language) semantics of t ambiguous. The same problem arises whenever an operator σ belongs to both languages with different interpretation functions. The simplest solution to avoid such issues is to add syntactical notations to make explicit the context of the language in which we are operating.

Definition 9 (Associated Signature). *The* associated signature *to the multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ is the ordered triple $\langle S, \preccurlyeq, \Pi \rangle$, where $S = S_1 \cup S_2$, $\preccurlyeq\, = \,\leq_1 \cup \leq_2$, and*

$$\begin{aligned} \Pi = &\{ \sigma_1 \colon w \to s \mid \sigma \colon w \to s \in \Sigma_1 \} \\ &\cup \{ \sigma_2 \colon w \to s \mid \sigma \colon w \to s \in \Sigma_2 \} \\ &\cup \{ \hookrightarrow_{s,s'} \colon s \to s' \mid s \ltimes s' \} \end{aligned}$$

It is trivial to prove that an associated signature is indeed an order-sorted signature, thus admitting a term algebra $\mathcal{T}_\Pi$. All the symbols forming terms in $\mathcal{T}_\Pi$ carry the source language information as a subscript, and all the new operators $\hookrightarrow_{s,s'}$ specify when a term of sort s is used in place of a term of sort s'. Although $\mathcal{T}_\Pi$ seems a suitable definition for multi-language terms, it is not a multi-language algebra according to Def. 7. However, we can exploit the construction of $\mathcal{T}_\Pi$ in order to provide a fully-fledged multi-language algebra able to generate multi-language terms.

Definition 10 (Multi-Language Term Algebra). *The* multi-language term algebra $\mathcal{T}$ *over a multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *with boundary functions* τ *is defined as follows:*

(1t) $s \in S$ *implies* $T_s = T_{\Pi,s}$*;*
(2t) $\sigma \in \Sigma^i_{w,s}$ *implies* $[\![\sigma]\!]^{w,s}_{\mathcal{T}} = [\![\sigma_i]\!]^{w,s}_{\mathcal{T}_\Pi}$ *for* $i = 1, 2$*; and*
(3t) $s \ltimes s'$ *implies* $\tau_{s,s'} = [\![\hookrightarrow_{s,s'}]\!]^{s,s'}_{\mathcal{T}_\Pi}$*.*

Proving that $\mathcal{T}$ satisfies Def. 7 is easy and omitted. $\mathcal{T}$ and $\mathcal{T}_\Pi$ share the same carrier sets (condition (1t)), and each single-language operator $\sigma \in \Sigma^i_{w,s}$ is interpreted as its annotated version σ_i in $\mathcal{T}_\Pi$ (condition (2t)). Furthermore, the multi-language operators $\hookrightarrow_{s,s'}$ no longer belong to the signature (they do not belong neither to $\mathfrak{S}_1$ nor to $\mathfrak{S}_2$) but their semantics is inherited by the boundary functions τ (condition (3t)), while their syntactic values are still in the carrier sets of the algebra (this construction is highly technical and very similar to the freely generated $\Sigma(X)$-algebra over a set of variables X, see [15]).

Note that this is exactly the formalization of the ad hoc multi-language specifications in [37,2,36,30]: [37,2,36] exploit distinct colors to disambiguate the source language of the operators, whereas [30] use different font styles for different languages. Moreover, boundary functions in [30] conceptually match the introduced operators $\hookrightarrow_{s,s'}$.

The last step in order to finalize the framework is to provide semantics for each term in $\mathcal{T}$. As with the order-sorted case, we need a notion of *regularity* for proving the initiality of the term algebra in its category, which in turn ensures a single eligible *(initial algebra) semantics.*

Definition 11 (Regularity). *A multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is* regular *if its associated signature* $\langle S, \preccurlyeq, \Pi \rangle$ *is regular.*

Proposition 2. *The associated signature* $\langle S, \preccurlyeq, \Pi \rangle$ *of a multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is regular if and only if* $\mathfrak{S}_1$ *and* $\mathfrak{S}_2$ *are regular.*

The last proposition enables to avoid checking the multi-language regularity whenever the regularity of the order-sorted signatures is known.

Theorem 2 (Initiality of $\mathcal{T}$). *The multi-language term algebra* $\mathcal{T}$ *over a regular multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is initial in the category* $\mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$*.*

Initiality of $\mathcal{T}$ is essential to assign a unique mathematical meaning to each term, as in the order-sorted case: Given a multi-language algebra $\mathcal{A}$, there is only one way of interpreting each term $t \in \mathcal{T}$ in $\mathcal{A}$ (satisfying the homomorphism conditions).

Definition 12 ((Multi-Language) Semantics). *Let* $\mathcal{A}$ *be a multi-language algebra over a regular multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$*. The* (multi-language) semantics *of a (multi-language) term* $t \in \mathcal{T}$ *induced by* $\mathcal{A}$ *is defined as*

$$[\![t]\!]_{\mathcal{A}} = h_{\mathrm{ls}(t)}(t)$$

The last equation is well-defined since h is the unique multi-language homomorphism $h\colon \mathcal{T} \to \mathcal{A}$ and for each $t \in \mathcal{T}$ there exists a least sort $\mathrm{ls}(t) \in S$ such that $t \in T_{\Pi,\mathrm{ls}(t)}$ (see Prop. 2.10 in [19]).

Example. Suppose we are interested in a multi-language over the signatures $\mathfrak{S}_1$ and $\mathfrak{S}_2$ specified in the example given in the background section such that satisfies the following properties:

- Terms denoting natural numbers can be used in place of characters $a \in \mathbb{A}$ according to the function $\mathrm{chr}\colon \mathbb{N} \to \mathbb{A}$ that maps the natural number n to the character symbol $a^{(n \mod |\mathbb{A}|)}$ (we are assuming a total lexicographical order $a^{(0)}, a^{(1)}, \ldots, a^{(|A|-1)}$ on $\mathbb{A}$);
- Terms denoting strings can be used in place of natural numbers $n \in \mathbb{N}$ according to the function $\mathrm{ord}\colon \mathbb{A} \to \mathbb{N}$, which is the inverse of chr restricted the initial segment on natural numbers $\mathbb{N}_{<|A|}$.

In order to achieve such a multi-language specification, we can simply provide a join relation $\ltimes$ on S and a boundary function $\alpha_{s,s'}$ for each extra-language subsort relation $s \ltimes s'$ introduced by $\ltimes$. We define the join relation and the boundary functions as follows:

$$\begin{array}{lllll}
\mathbb{e} \ltimes \mathbb{a} & \wedge & \mathbb{n} \ltimes \mathbb{a} & \longrightarrow & \alpha_{\mathbb{e},\mathbb{a}}(n) = \alpha_{\mathbb{n},\mathbb{a}}(n) = \mathrm{chr}(n) \\
\mathbb{s} \ltimes \mathbb{n} & \wedge & \mathbb{a} \ltimes \mathbb{n} & \longrightarrow & \begin{cases} \alpha_{\mathbb{a},\mathbb{n}}(a) = \mathrm{ord}(a) \\ \alpha_{\mathbb{s},\mathbb{n}}(a_0 \ldots a_n) = \sum_{k=0}^{n} \alpha_{\mathbb{a},\mathbb{n}}(a_k) \cdot 10^k \end{cases}
\end{array}$$

The multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$ can now be obtained by joining the projected algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ with the set of boundary functions α. The term algebra $\mathcal{T}$ over $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ provides all the multi-language terms, and Thm. 2 ensures a unique denotation of each $t \in \mathcal{T}$ in $\mathcal{A}$. For instance, the term

$$t = \hookrightarrow_{\mathbb{s},\mathbb{n}}(\underbrace{\mathtt{+}_2(\mathtt{f}_2, \overbrace{\mathtt{+}_2(\mathtt{o}_2, \underbrace{\hookrightarrow_{\mathbb{e},\mathbb{a}}(\overbrace{\mathtt{+}_1(10_1, 5_1)}^{t_4})}_{t_3})}^{t_2})}_{t_1}) \tag{1}$$

is syntactically equivalent to the following but with a less pedantic notation, where language subscripts are replaced by colors (red for one, and blue for two) and prefix notation is replaced by infix notation

$$\hookrightarrow_{\mathbb{s},\mathbb{n}}(\mathtt{f}\ \mathtt{+}\ \mathtt{o}\ \mathtt{+}\ \hookrightarrow_{\mathbb{e},\mathbb{a}}(10\ \mathtt{+}\ 5))$$

and it denotes the natural numbers 765:

$$\begin{aligned}
[\![t_4]\!]_{\mathcal{A}} &= h_{\mathrm{ls}(t_4)}(t_4) = h_{\mathbb{e}}(t_4) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}([\![10]\!]_{\mathcal{A}}, [\![5]\!]_{\mathcal{A}}) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(10, 5) = 15 \\
[\![t_3]\!]_{\mathcal{A}} &= h_{\mathrm{ls}(t_3)}(t_3) = h_{\mathbb{a}}(t_3) = [\![\hookrightarrow_{\mathbb{e},\mathbb{a}}]\!]_{\mathcal{A}}^{\mathbb{e},\mathbb{a}}([\![t_4]\!]_{\mathcal{A}}) = [\![\hookrightarrow_{\mathbb{e},\mathbb{a}}]\!]_{\mathcal{A}}^{\mathbb{e},\mathbb{a}}(15) = \mathtt{o} \\
[\![t_2]\!]_{\mathcal{A}} &= h_{\mathrm{ls}(t_2)}(t_2) = h_{\mathbb{s}}(t_2) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}([\![\mathtt{o}]\!]_{\mathcal{A}}, [\![t_3]\!]_{\mathcal{A}}) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}(\mathtt{o}, \mathtt{o}) = \mathtt{oo} \\
[\![t_1]\!]_{\mathcal{A}} &= h_{\mathrm{ls}(t_1)}(t_1) = h_{\mathbb{s}}(t_1) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}([\![\mathtt{f}]\!]_{\mathcal{A}}, [\![t_2]\!]_{\mathcal{A}}) = [\![\mathtt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}(\mathtt{f}, \mathtt{oo}) = \mathtt{foo} \\
[\![t]\!]_{\mathcal{A}} &= h_{\mathrm{ls}(t)}(t) = h_{\mathbb{n}}(t) = [\![\hookrightarrow_{\mathbb{s},\mathbb{n}}]\!]_{\mathcal{A}}^{\mathbb{s},\mathbb{n}}([\![t_1]\!]_{\mathcal{A}}) = [\![\hookrightarrow_{\mathbb{s},\mathbb{n}}]\!]_{\mathcal{A}}^{\mathbb{s},\mathbb{n}}(\mathtt{foo}) = 765
\end{aligned}$$

(see the proof of Prop. 2.10 in [19] to check how to compute the least sort of a term).

4 Refining the Construction

The construction in Sect. 3 does not set any constraint on boundary functions, thus giving a great deal of flexibility to language designers. For instance, they can provide boundary functions that act differently with respect to the intra-language subsort relation $\preccurlyeq$: According to the previous example, it would have been possible to define $\alpha_{\mathbb{n},\mathbb{a}} \neq \alpha_{\mathbb{e},\mathbb{a}}$ to employ different value conversion specifications for terms in $T_{\mathbb{n}}$, based on whether they are used as natural numbers ($\mathbb{n}$) or as expressions ($\mathbb{e}$). However, when this amount of flexibility is not needed, we can refine the previous construction by reducing the amount of syntax introduced by the associated signature. In this section we examine

- the case where boundary functions satisfy the monotonicity conditions of order-sorted algebra operators (Sect. 4.1); and
- the case where boundary functions commutes with the semantics of operator symbols (Sect. 4.2).

In both cases, we prove that the introduced refinements do not affect the initiality of the term algebra, thereby providing unambiguous semantics to the multi-language.

4.1 Subsort Polymorphic Boundary Functions

In Sect. 3, the join relation constraints $s \ltimes s'$ are turned in syntactical operators $\hookrightarrow_{s,s'}$ in the associated signature $\langle S, \preccurlyeq, \Pi \rangle$. We now show how to handle all the syntactical overhead introduced by $\ltimes$ with a single polymorphic operator $\hookrightarrow$ whenever the boundary functions satisfy the monotonicity conditions of the order-sorted algebras [19]. Such conditions require a subsort relation $s_1 \leq s_2$ between the sorts of a polymorphic operator $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$, assuming that $w_1 \leq w_2$. In our case, $\sigma = \hookrightarrow$, and thus we extend Def. 6 with the following ad hoc constraint (2s*):

Definition 6* (SP Multi-Language Signature). *A* subsort polymorphic (SP) multi-language signature *is a multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *such that*

(2s)* $s_1 \ltimes s_1'$*,* $s_2 \ltimes s_2'$*, and* $s_1 \preccurlyeq s_2$ *imply* $s_1' \preccurlyeq s_2'$*.*

Furthermore, order-sorted algebras demand consistency of the interpretation functions of a subsort polymorphic operator on the smaller domain, which results in the following condition (2a*) on boundary functions (that extends Def. 7):

Definition 7* (SP Multi-Language Algebra). *Let* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *be a SP multi-language signature. A* subsort polymorphic (SP) multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra *is a multi-language* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$*-algebra* $\mathcal{A}$ *such that*

(2a)* $s_1 \ltimes s_1'$*,* $s_2 \ltimes s_2'$*, and* $s_1 \preccurlyeq s_2$ *imply that* $\alpha_{s_1,s_1'}(a) = \alpha_{s_2,s_2'}(a)$ *for each* $a \in A_{s_1}$*.*

The notion of homomorphism in this new context does not change (an homomorphism between two SP algebras is still an S-sorted function decomposable in two order-sorted homomorphisms that commutes with boundaries), whereas the associated signature to an SP multi-language signature merely differs from Def. 9 for having a unique polymorphic operator $\hookrightarrow$ instead of a family of parametrized symbols $\{ \hookrightarrow_{s,s'} : s \to s' \mid s \ltimes s' \}$.

Definition 9* (SP Associated Signature). *The* subsort polymorphic (SP) associated signature *to the SP multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is the ordered triple* $\langle S, \preccurlyeq, \Pi \rangle$*, where* $S = S_1 \cup S_2$*,* $\preccurlyeq\, = \,\leq_1 \cup \leq_2$*, and*

$$\begin{aligned} \Pi = &\{ \sigma_1 : w \to s \mid \sigma : w \to s \in \Sigma_1 \} \\ \cup &\{ \sigma_2 : w \to s \mid \sigma : w \to s \in \Sigma_2 \} \\ \cup &\{ \hookrightarrow : s \to s' \mid s \ltimes s' \} \end{aligned}$$

Since the associated signature is the basis for the term algebra, we need to modify the condition (3t) in Def. 9:

Definition 10* (SP Multi-Language Term Algebra). *The* subsort polymorphic (SP) multi-language term algebra $\mathcal{T}$ *over a SP multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *with boundary functions* τ *is defined as follows:*

(1t) $s \in S$ *implies* $T_s = T_{\Pi,s}$*;*
(2t) $\sigma \in \Sigma^i_{w,s}$ *implies* $[\![\sigma]\!]^{w,s}_{\mathcal{T}} = [\![\sigma_i]\!]^{w,s}_{\mathcal{T}_\Pi}$ *for* $i = 1, 2$*; and*
(3t)* $s \ltimes s'$ *implies* $\tau_{s,s'} = [\![\hookrightarrow]\!]^{s,s'}_{\mathcal{T}_\Pi}$.

Signature regularity is still defined as in Def. 11 and Prop. 2 still holds for the extended version developed in this section. As a result, the SP multi-language term $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{T}$ is still initial in the category $\mathbf{Alg}^*(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ of SP multi-language algebras over the SP multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$.

Theorem 3. *Let* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *be a SP multi-language signature. The class of all SP* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$*-algebras and the class of all* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$*-homomorphisms form a category denoted by* $\mathbf{Alg}^*(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$*.*

Theorem 4 (Initiality of $\mathcal{T}$). *The SP multi-language term algebra* $\mathcal{T}$ *over a regular SP multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is initial in the category* $\mathbf{Alg}^*(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$*.*

The semantics of a term t induced by a SP multi-language algebra $\mathcal{A}$ is defined in the same way of Def. 12, thanks to the initiality result: $[\![t]\!]_{\mathcal{A}} = h_{\mathrm{ls}(t)}(t)$. The main advantage of dealing with SP multi-language terms is that the framework is able to determine the correct interpretation function of the operator $\hookrightarrow$, making the subscript notation developed in the previous section superfluous. This also means that programmers are exempted from explicitly annotating multi-language programs with sorts, a non-trivial task in the general case that could introduce type cast bugs.

Example. The boundary functions of the previous example are subsort polymorphic: $\alpha_{\mathbb{a},\mathbb{n}}(a) = \mathrm{ord}(a) = \alpha_{\mathbb{s},\mathbb{n}}(a)$ for each character $a \in \mathbb{A}$, and $\alpha_{\mathbb{n},\mathbb{a}} = \alpha_{\mathbb{e},\mathbb{a}}$ by definition. Thus, the equivalent of the term t (see Eq. 1) in the SP term algebra is

$$\dot{t} = \hookrightarrow(\texttt{+}_2(\texttt{f}_2, \texttt{+}_2(\texttt{o}_2, \hookrightarrow(\texttt{+}_1(10_1, 5_1))))) \tag{2}$$

or, according to the previous notation,

$$\hookrightarrow(\texttt{f + o + }\hookrightarrow(\texttt{10 + 5}))$$

and denoting the same natural number 765.

4.2 Semantic-Only Boundary Functions

In the previous section, we have shown how to handle the flow of values across different languages with a single polymorphic operator. Now, we present a new multi-language construction where neither extra operators are added to the associated signature, nor single-language operators have to be annotated with subscripts indicating their original language. Thus, the resulting multi-language syntax comprises only symbols in $\Sigma_1 \cup \Sigma_2$. Such a construction is achieved by:

- Imposing commutativity conditions on algebras, making homomorphisms transparently inherit the semantics of boundary functions. The framework is therefore able to apply the correct value conversion function whenever is necessary, without the need for an explicit syntactical operator $\hookrightarrow$.
- Requiring a new form of *cross-language polymorphism* able to cope with shared operators among languages. The initiality of term algebras is preserved by modifying the notion of signature in a way that every operator admits a least sort.

The variant of the framework presented in this section is particularly useful when designing the extension of a language in a modular fashion. For instance, if the signature $\mathfrak{S}_1$ models the syntax of a simple functional language (for an example, see [15, p. 77]) without an explicit encoding for string values, and $\mathfrak{S}_2$ is a language for manipulating strings (similar to the language L_2 of the running example of this paper), we can exploit the construction presented below in order to embed $\mathfrak{S}_2$ into $\mathfrak{S}_1$.

Signature. The main issue that can arise at this stage of multi-language signature is the presence of shared operators in Σ_1 and Σ_2. Contrary to the previous cases where such ambiguity is solved by adding subscripts in the associated signature, the trade off here is requiring ad hoc or subsort polymorphism across signatures.

Definition 6* (SO Multi-Language Signature). *A* semantic-only (SO) multi-language signature *is a multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *such that*

(2s*) $\langle S, \leq \rangle$ *is a poset; and*

(3s*) $\sigma \in \Sigma^i_{w_1,s_1} \cap \Sigma^j_{w_2,s_2}$ *and* $w_1 \ltimes w_2$ *imply* $s_1 \ltimes s_2$ *with* $i, j = 1, 2$ *and* $i \neq j$.

Condition (2s*) forces the subsort relation to be directed, avoiding symmetricity of syntactic categories (this is typical when modeling language extensions), while condition (3s*) shifts the monotonicity condition of order-sorted signature to syntactically equal operators in $\Sigma_1 \cap \Sigma_2$.

The associated signature is defined without adding extra symbols in the signature, i.e., $\Pi = \Sigma_1 \cup \Sigma_2$, and deliberately confounding the relations $\ltimes$ and $\preccurlyeq$ in $\leq$:

Definition 9* (SO Associated Signature). *The* SO associated signature *to the SO multi-language signature* $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ *is the ordered triple* $\langle S, \leq, \Pi \rangle$*, where* $S = S_1 \cup S_2$*,* $\leq \; = \; \preccurlyeq \cup \ltimes$*, and* $\Pi = \Sigma_1 \cup \Sigma_2$*.*

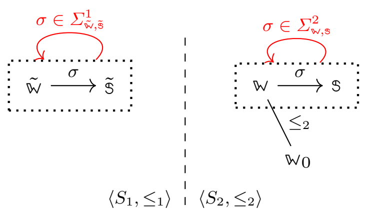
The embedding of $\ltimes$ in $\leq$ (i.e., $\ltimes \subseteq \leq$) in the associated signature enables the order-sorted term algebra construction to automatically build multi-language terms, without the need for an explicit operator $\hookrightarrow$ that acts as a bridge between syntactic categories. It is easy to see that the term algebra over the associated signature is precisely the symbols-free version of multi-language described at the beginning.

Unfortunately, multi-language regularity does not follow anymore from single-languages regularity and vice versa (see Figs. 3 and 4)[6]. More formally, Prop. 2 does not hold in this new context:

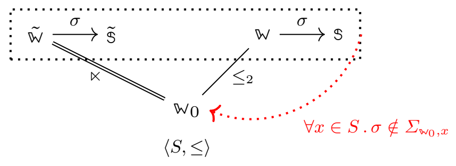
- Suppose $S_1 = \{\, \tilde{\mathtt{w}}, \tilde{\mathtt{s}} \,\}$, $S_2 = \{\, \mathtt{w}_0, \mathtt{w}, \mathtt{s} \,\}$, $\leq_1$ and $\leq_2$ to be the reflexive relations on S_1 and S_2, respectively, plus $\mathtt{w}_0 \leq_2 \mathtt{w}$, and $\sigma \in \Sigma^1_{\tilde{\mathtt{w}},\tilde{\mathtt{s}}} \cap \Sigma^2_{\mathtt{w},\mathtt{s}}$. If the join relation $\ltimes$ is defined as $\mathtt{w}_0 \ltimes \tilde{\mathtt{w}}$ and $\mathtt{s} \ltimes \tilde{\mathtt{s}}$, the resulting associated signature is no longer regular, although $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are regular (Fig. 3a). In Fig. 3b, it is easy to see that $\sigma \in \Sigma_{\tilde{\mathtt{w}},\tilde{s}}$ and $\mathtt{w}_0 \leq \mathtt{w}$, but the set $\{\, (\mathtt{w}, \mathtt{s}) \mid \sigma \in \Sigma_{\mathtt{w},\mathtt{s}} \wedge \mathtt{w}_0 \leq \mathtt{w} \,\} = \{\, (\tilde{\mathtt{w}}, \tilde{\mathtt{s}}), (\mathtt{w}, \mathtt{s}) \,\}$ does not have a least element w.r.t. $\mathtt{w}_0$.
- On the other hand, let $S_1 = \{\, \tilde{\mathtt{w}}, \mathtt{w}_0, \mathtt{w}_1, \tilde{\mathtt{s}} \,\}$, $S_2 = \{\, \mathtt{w}_2, \mathtt{s}_2 \,\}$, $\leq_1$ and $\leq_2$ be the reflexive relations on S_1 and S_2, respectively, plus $\mathtt{w}_0 \leq_1 \tilde{\mathtt{w}}$ and $\mathtt{w}_0 \leq_1 \mathtt{w}_1$, and $\sigma \in \Sigma^1_{\tilde{\mathtt{w}},\tilde{\mathtt{s}}} \cap \Sigma^1_{\mathtt{w}_1,\tilde{\mathtt{s}}} \cap \Sigma^2_{\mathtt{w}_2,\mathtt{s}_2}$. If the join relation $\ltimes$ is defined as $\mathtt{w}_2 \ltimes \tilde{\mathtt{w}}$, $\mathtt{w}_2 \ltimes \mathtt{w}_1$, $\mathtt{w}_0 \ltimes \mathtt{w}_2$, and $\mathtt{s}_2 \ltimes \tilde{\mathtt{s}}$, the resulting associated signature is regular (Fig. 4a), although $\mathfrak{S}_1$ is not: given $\sigma \in \Sigma_{\tilde{\mathtt{w}},\tilde{\mathtt{s}}}$ and $\mathtt{w}_0 \leq \tilde{\mathtt{w}}$, the set $\{\, (\mathtt{w}, \mathtt{s}) \mid \sigma \in \Sigma_{\mathtt{w},\mathtt{s}} \wedge \mathtt{w}_0 \leq \mathtt{w} \,\} = \{\, (\tilde{\mathtt{w}}, \tilde{\mathtt{s}}), (\mathtt{w}_1, \tilde{\mathtt{s}}), (\mathtt{w}_2, \mathtt{s}_2) \,\}$ has least element $(\mathtt{w}_2, \mathtt{s}_2)$ w.r.t. $\mathtt{w}_0$ (Fig. 4b).

A positive result can be obtained by recalling that regularity is easier to check when $\langle S, \leq \rangle$ satisfies the descending chain condition (DCC):

[6] An (horizontal) arrow from an arity symbol w to a sort s labelled with an operator symbol σ is an alternative shorthand for $\sigma \colon w \to s$. A (vertical) single line between two sorts s below s' labelled with a binary relation $\leq$ means that $s \leq s'$ (if the binary relation is the join relation $\ltimes$ the line is doubled). A dotted rectangle around operators is a graphical representation of the set of ranks (w, s) that must have a minimum element (red arrows) in order for the signature to be regular.

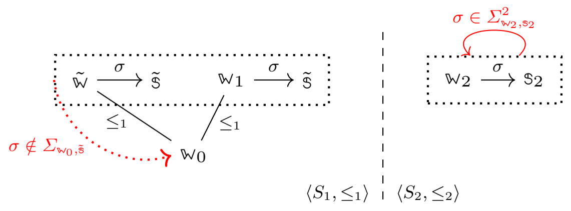

$\sigma \in \Sigma^1_{\tilde{w},\tilde{s}}$
$\tilde{w} \xrightarrow{\sigma} \tilde{s}$
$\sigma \in \Sigma^2_{w,s}$
$w \xrightarrow{\sigma} s$
$\leq_2$
w_0
$\langle S_1, \leq_1 \rangle$
$\langle S_2, \leq_2 \rangle$

(a) The Hasse-like diagrams of regular signatures $\mathfrak{S}_1$ (left) and $\mathfrak{S}_2$ (right).

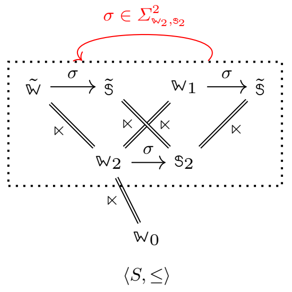

$\tilde{w} \xrightarrow{\sigma} \tilde{s}$
$w \xrightarrow{\sigma} s$
$\ltimes$
$\leq_2$
w_0
$\forall x \in S \,.\, \sigma \notin \Sigma_{w_0,x}$
$\langle S, \leq \rangle$

(b) The Hasse-like diagram of the non-regular multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$.

Figure 3: A non-regular multi-language signature comprising two regular order-sorted signatures.


$\tilde{w} \xrightarrow{\sigma} \tilde{s}$
$w_1 \xrightarrow{\sigma} \tilde{s}$
$\leq_1$
$\leq_1$
$\sigma \notin \Sigma_{w_0,\tilde{s}}$
w_0
$\sigma \in \Sigma^2_{w_2,s_2}$
$w_2 \xrightarrow{\sigma} s_2$
$\langle S_1, \leq_1 \rangle$
$\langle S_2, \leq_2 \rangle$

(a) The Hasse-like diagrams of signatures $\mathfrak{S}_1$ (non-regular, left) and $\mathfrak{S}_2$ (regular, right).


$\sigma \in \Sigma^2_{w_2,s_2}$
$\tilde{w} \xrightarrow{\sigma} \tilde{s}$
$w_1 \xrightarrow{\sigma} \tilde{s}$
$\ltimes$
$w_2 \xrightarrow{\sigma} s_2$
w_0
$\langle S, \leq \rangle$

(b) The Hasse-like diagram of the regular multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$.

Figure 4: A regular multi-language signature comprising a non-regular order-sorted signature.

Lemma 1 (Regularity over DCC poset [19]). *An order-sorted signature Σ over a DCC poset $\langle S, \leq \rangle$ is regular if and only if whenever $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and there is some $w_0 \leq w_1, w_2$, then there is some $w \leq w_1, w_2$ such that $\sigma \in \Sigma_{w,s}$ and $w_0 \leq w$.*

At this point, we can relate the DCC of the poset $\langle S, \leq \rangle$ in the associated signature of $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ to the DCC of $\langle S_1, \leq_1 \rangle$ and $\langle S_2, \leq_2 \rangle$:

Proposition 3. *Let $\langle S, \leq, \Sigma \rangle$ be the associated signature of $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$. Then, $\langle S, \leq \rangle$ is DCC if and only if $\langle S_1, \leq_1 \rangle$ and $\langle S_2, \leq_2 \rangle$ are DCC.*

As a result, whenever we know that $\langle S_1, \leq_1 \rangle$ and $\langle S_2, \leq_2 \rangle$ are DCC, we can check the regularity of $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ by employing the Lemma 1 without checking whether $\langle S, \leq \rangle$ is DCC.

Algebra. In this multi-language construction, the boundary functions behaviour is no more bounded to syntactical operators as in the previous sections, but it is inherited by homomorphisms. A necessary condition to accomplish this aim is the commutativity of interpretation functions with boundary functions:

Definition 7* (SO Multi-Language Algebra). *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be an SO multi-language signature. A* semantic-only (SO) multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra *is an SP multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$ such that*

(3a*) $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \ltimes w_2$ imply that $\alpha_{s_1,s_2}(\llbracket \sigma \rrbracket_{\mathcal{A}}^{w_1,s_1}(a)) = \llbracket \sigma \rrbracket_{\mathcal{A}}^{w_2,s_2}($ $\alpha_{w_1,w_2}(a))$ for each $a \in A_{w_1}$.

Note that $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and $w_1 \ltimes w_2$ imply $s_1 \ltimes s_2$ by condition (3s*). The notion of homomorphism remains unchanged from Def. 8 (to understand how the homomorphisms inherit the boundary functions behaviour, see the proof of Thm. 6).

The term algebra is defined similarly to Def. 10, except for boundary functions:

Definition 10* (SO Multi-Language Term Algebra). *The* semantic-only (SO) multi-language term algebra *$\mathcal{T}$ over an SO multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ with boundary functions τ is defined as follows:*

(1t*) $s \in S$ implies $T_s = T_{\Pi,s}$;
(2t*) $\sigma \in \Sigma_{w,s}$ implies $\llbracket \sigma \rrbracket_{\mathcal{T}}^{w,s} = \llbracket \sigma \rrbracket_{\mathcal{T}_\Pi}^{w,s}$; and
(3t*) $s \ltimes s'$ implies $\tau_{s,s'} = \mathrm{id}_{T_s}$.

Since the subsort relation $\leq$ includes the join relation $\ltimes$, $s \ltimes s'$ implies $T_{\Pi,s} = T_s \subseteq T_{s'} = T_{\Pi,s'}$. Thus, the boundary function $\tau_{s,s'}$ can be defined as the identity on the smaller domain (note that it trivially satisfies the commutativity condition (3a*)).

Proposition 4. *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be an SO multi-language signature. Then, the SO multi-language term $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra is a proper SO multi-language algebra.*

Theorem 5. *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be a SO multi-language signature. The class of all SO $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebras and the class of all $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-homomorphisms form a category denoted by $\mathbf{Alg}^\star(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$.*

We can now prove the initiality of $\mathcal{T}$ in its category.

Theorem 6 (Initiality of $\mathcal{T}$). *Let $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$ be a regular multi-language signature. Then, the term algebra $\mathcal{T}$ is an initial object in the category $\mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$.*

Thanks to the initiality of the term algebra, the definition of term semantics is the same of Def. 12.

Example. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two order-sorted algebras over the signatures $\mathfrak{S}_1$ and $\mathfrak{S}_2$, respectively, as formalized in the example in Sect. 3. Suppose we are interested in a new multi-language $\mathcal{A}$ over $\mathfrak{S}_1$ and $\mathfrak{S}_2$ such that any string expressions t of sort $\mathbb{s}$ in $\mathfrak{S}_2$ can denote the natural number length($[\![t]\!]_{\mathcal{A}_2}$) when embedded in $\mathfrak{S}_1$ terms. For instance, we require that $[\![10\ \texttt{+}\ 5]\!]_{\mathcal{A}} = [\![10\ \texttt{+}\ 5]\!]_{\mathcal{A}_1} = 15$ and $[\![\texttt{f + o}]\!]_{\mathcal{A}} = [\![\texttt{f + o}]\!]_{\mathcal{A}_2} = \texttt{fo}$, but $[\![(\texttt{f + o})\ \texttt{+}\ (10\ \texttt{+}\ 5)]\!]_{\mathcal{A}} = [\![\texttt{fo}\ \texttt{+}\ 15]\!]_{L} = 17$ (parentheses in the last term have only been used to disambiguate the parsing result).

Since the requirements demand to use string expressions in place of natural numbers, the join relation $\ltimes$ shall define $\mathbb{s} \ltimes \mathbb{n}$ and ensure transitivity, hence $\mathbb{s} \ltimes \mathbb{e}$, $\mathbb{c} \ltimes \mathbb{n}$, and $\mathbb{c} \ltimes \mathbb{e}$.

The signatures $\mathfrak{S}_1$ and $\mathfrak{S}_2$ are trivially regular. However, by merging $\mathfrak{S}_1$ and $\mathfrak{S}_2$, we are causing subsort polymorphism on the symbol $\texttt{+}$, which is used as sum operator in $\mathcal{A}_1$ and as concatenation operator in $\mathcal{A}_2$, and therefore we have to check the regularity: Let $w_1 = \mathbb{e}\,\mathbb{e}$, $w_2 = \mathbb{s}\,\mathbb{s}$, $s_1 = \mathbb{e}$, and $s_2 = \mathbb{s}$. Given $\texttt{+} \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ and the lower bound $w_0 = \mathbb{c}\,\mathbb{c} \leq w_1, w_2$, then there exists $w = \mathbb{s}\,\mathbb{s}$ such that $w \leq w_1, w_2$ and $\texttt{+} \in \Sigma_{w,s}$, where $s = \mathbb{s} \leq s_1, s_2$ (we have employed Lemma 1 thanks to Prop. 3). Analogously, when $w_0 = w_1, w_2$ the relative least rank is $(\mathbb{s}\,\mathbb{s}, \mathbb{s})$.

The multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$ is now defined by joining the projected algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ and by defining boundary functions $a_{s,s'}$ for each $s \ltimes s'$ such that convert strings in naturals (their length) when strings are used in place of naturals:

$$a_{\mathbb{c},\mathbb{n}}(a) = a_{\mathbb{c},\mathbb{e}}(a) = 1 \qquad\qquad a_{\mathbb{s},\mathbb{n}}(\hat{a}) = a_{\mathbb{s},\mathbb{e}}(\hat{a}) = \mathrm{length}(\hat{a})$$

The above definition of boundary functions satisfy both conditions (2a*) and (3a*).

The initiality theorem yields the semantic homomorphism from $\mathcal{T}$ to $\mathcal{A}$. For instance, suppose we want to compute the semantics of the term

$$t = \texttt{+}(\underbrace{\texttt{+}(\texttt{f}, \texttt{o})}_{t_1}, \overbrace{\texttt{+}(10, 5)}^{t_2})$$

The least sorts of t, t_1, and t_2 are $\mathbb{e}$, $\mathbb{s}$, and $\mathbb{e}$, respectively. The operator $\texttt{+}$ belongs to both $\Sigma_{\mathbb{e}\,\mathbb{e},\mathbb{e}}$ and $\Sigma_{\mathbb{s}\,\mathbb{s},\mathbb{s}}$, and its least rank w.r.t. the lower bound $\mathrm{ls}(t_1)\,\mathrm{ls}(t_2) = \mathbb{s}\,\mathbb{e}$ is $(\mathbb{e}\,\mathbb{e}, \mathbb{e})$. By Def. 12 we have

$$[\![t]\!]_{\mathcal{A}} = h_{\mathbb{e}}(t) = [\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(h_{\mathbb{e}}(t_1), h_{\mathbb{e}}(t_2))$$

At this point, since $\mathrm{ls}(t_1) = \mathbb{s}$ and $\mathrm{ls}(\texttt{f}) = \mathrm{ls}(\texttt{o}) = \mathbb{c}$, then the least rank of the root symbol $\texttt{+}$ of t_1 w.r.t. the lower bound $\mathrm{ls}(\texttt{f})\,\mathrm{ls}(\texttt{o}) = \mathbb{c}\,\mathbb{c}$ is $(\mathbb{s}\,\mathbb{s}, \mathbb{s})$, thus

$$h_{\mathbb{e}}(t_1) = a_{\mathbb{s},\mathbb{e}}(h_{\mathbb{s}}(t_1)) = a_{\mathbb{s},\mathbb{e}}([\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}(h_{\mathbb{s}}(\texttt{f}), h_{\mathbb{s}}(\texttt{o}))) = a_{\mathbb{s},\mathbb{e}}([\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{s}\,\mathbb{s},\mathbb{s}}(\texttt{f}, \texttt{o})) = a_{\mathbb{s},\mathbb{e}}(\texttt{fo}) = 2$$

Similarly, $\mathrm{ls}(t_2) = \mathbb{e}$ and $\mathrm{ls}(10) = \mathrm{ls}(5) = \mathbb{n}$. Then, the least rank of the root symbol $\texttt{+}$ of t_2 w.r.t. the lower bound $(\mathbb{n}, \mathbb{n})$ is $(\mathbb{e}\,\mathbb{e}, \mathbb{e})$ and therefore we have

$$h_{\mathbb{e}}(t_2) = [\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(h_{\mathbb{n}}(10), h_{\mathbb{n}}(5)) = [\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(10, 5) = 15$$

Finally,

$$[\![t]\!]_{\mathcal{A}} = h_{\mathbb{e}}(t) = [\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(h_{\mathbb{e}}(t_1), h_{\mathbb{e}}(t_2)) = [\![\texttt{+}]\!]_{\mathcal{A}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(2, 15) = 17$$

as desired.

We can observe that without any syntactical operator the framework is still able to apply the correct boundary functions to move values across languages.

5 Reduction to Order-Sorted Algbera

The constructions in the previous sections beg the question whether a multi-language algebra admits an equivalent order-sorted representation. Conceptually, it would mean that being a multi-language is essentially a matter of perspective: By forgetting how the multi-language has been constructed, what is left is simply an ordinary language. Mathematically speaking, it requires us to exhibit a *reduction functor* F from the multi-language category to an order-sorted one, such that there is an isomorphism ϕ between the carrier sets of the multi-language term $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{T}$ and $F(\mathcal{T})$, and such that $\llbracket t \rrbracket_{\mathcal{A}} = \llbracket \phi(t) \rrbracket_{F(\mathcal{A})}$ for each $t \in \mathcal{T}$ and for each multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$.

In the following, we denote the reduction functor by F, F^*, and $F^\star$ accordingly whether its domain is the category $\mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$, $\mathbf{Alg}^*(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$, and $\mathbf{Alg}^\star(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$, respectively.

In the case of $\mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ and $\mathbf{Alg}^*(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ categories, the construction of F and F^* is very simple, and we illustrate it only for the plain multi-language algebras of Sect. 3: Let $\mathcal{A}$ be a multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra. Then, we define the order-sorted $\langle S, \preccurlyeq, \Pi \rangle$-algebra $\mathcal{A}_\Pi$ (called the *associated order-sorted algebra* of $\mathcal{A}$) by setting

(1π) $A_{\Pi,s} = A_s$ for each $s \in S$;
(2π) $\llbracket \sigma_i \rrbracket^{w,s}_{\mathcal{A}_\Pi} = \llbracket \sigma \rrbracket^{w,s}_{\mathcal{A}}$ for each $\sigma \in \Sigma^i_{w,s}$ and $i = 1, 2$; and
(3π) $\llbracket \hookrightarrow_{s,s'} \rrbracket^{s,s'}_{\mathcal{A}_\Pi} = \alpha_{s,s'}$ for each $s \ltimes s'$.

If $\mathcal{A}$ and $\mathcal{B}$ are multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebras, and h is a multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-homomorphism from $\mathcal{A}$ to $\mathcal{B}$, the functor F maps $\mathcal{A}$ and $\mathcal{B}$ to their associated order-sorted algebras $\mathcal{A}_\Pi$ and $\mathcal{B}_\Pi$ and the homomorphism h to itself. Since $A_\Pi = A$, the isomorphism ϕ is the identity function.

Theorem 7. *$F\colon \mathbf{Alg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq) \to \mathbf{OSAlg}(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ is a functor for every multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$. Moreover, $\llbracket t \rrbracket_{\mathcal{A}} = \llbracket t \rrbracket_{F(\mathcal{A})}$ for each $t \in \mathcal{T}$ and for each multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$.*

If $\mathcal{A}$ is an SP multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra, the construction of the reduction functor F^* is similar to the definition of F. The only difference is the equation in the condition (3π) that turns into

(3π^*) $\llbracket \hookrightarrow \rrbracket^{s,s'}_{\mathcal{A}_\Pi} = \alpha_{s,s'}$ for each $s \ltimes s'$.

Finally, the definition of $F^\star$ starting from the category $\mathbf{Alg}^\star(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ of SO multi-language algebras is slightly different. We define $F^\star$ as a map from the multi-language category $\mathbf{Alg}^\star(\mathfrak{S}_1, \mathfrak{S}_2, \leq)$ to the order-sorted category $\mathbf{OSAlg}(S, \preccurlyeq, \Sigma)$. We denote the reduction of a multi-language algebra $\mathcal{A}$ and a homomorphism $h\colon \mathcal{A} \to \mathcal{B}$ as $F(\mathcal{A}) = \mathcal{A}_\downarrow$ and $F(h) = h_\downarrow\colon \mathcal{A}_\downarrow \to \mathcal{B}_\downarrow$. The order-sorted algebra $\mathcal{A}_\downarrow$ has the same carrier sets of the multi-language algebra $\mathcal{A}$, i.e., $A_\downarrow = A$, and interpretation functions $\llbracket \sigma \rrbracket^{w,s}_{\mathcal{A}_\downarrow} = \llbracket \sigma \rrbracket^{w,s}_{\mathcal{A}}$. Furthermore, we define $h_\downarrow = h$. Intuitively, the algebra $\mathcal{A}_\downarrow$ is formally defined simply by forgetting about the boundary functions, while the homomorphism $h_\downarrow\colon \mathcal{A}_\downarrow \to \mathcal{B}_\downarrow$ inherits their semantics from h. Again, the isomorphism ϕ is the identity.

Theorem 8. *$F^\star\colon \mathbf{Alg}^\star(\mathfrak{S}_1, \mathfrak{S}_2, \leq) \to \mathbf{OSAlg}(S, \preccurlyeq, \Sigma)$ is a functor for every SO multi-language signature $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$. Moreover, $[\![t]\!]_{\mathcal{A}} = [\![t]\!]_{F^\star(\mathcal{A})}$ for each $t \in \mathcal{T}$ and for each SO multi-language $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra $\mathcal{A}$.*

Unfortunately, even though $\mathcal{T}$ is an initial algebra in its category, $F^\star(\mathcal{T}) = \mathcal{T}_\downarrow$ is not: Given two multi-language algebras $\mathcal{A}$ and $\mathcal{A}'$ that differ only in the boundary functions (we denote by α and α' the families of boundary functions of $\mathcal{A}$ and $\mathcal{A}'$, respectively) they both get mapped by $F^\star$ to the same order-sorted algebra $\mathcal{A}_\downarrow$. Thus, if $h\colon \mathcal{T} \to \mathcal{A}$ and $h'\colon \mathcal{T} \to \mathcal{A}'$ are the unique homomorphisms going from $\mathcal{T}$ to $\mathcal{A}$ and $\mathcal{A}'$, the functor F maps them to two different order-sorted homomorphisms $h_\downarrow\colon \mathcal{T}_\downarrow \to \mathcal{A}_\downarrow$ and $h'_\downarrow\colon \mathcal{T}_\downarrow \to \mathcal{A}_\downarrow$ both leaving $\mathcal{T}_\downarrow$ and going to $\mathcal{A}_\downarrow$, hence losing the uniqueness property. However, this does not pose a problem once fixed a family of boundary functions:

Theorem 9. *Let $\mathcal{T}$ be the multi-language term $\langle \mathfrak{S}_1, \mathfrak{S}_2, \leq \rangle$-algebra and $\mathcal{A}$ be an order-sorted $\langle S, \preccurlyeq, \Sigma \rangle$-algebra. Given a family of boundary functions $\alpha = \{ \alpha_{s,s'} \mid s \ltimes s' \}$ such that satisfies condition (3a*), there exists a unique order-sorted $\langle S, \preccurlyeq, \Sigma \rangle$-homomorphism $h^\alpha\colon \mathcal{T}_\downarrow \to \mathcal{A}$ commuting with α, i.e., if $s \ltimes s'$, then $h^\alpha_{s'}(t) = \alpha_{s,s'}(h^\alpha_s(t))$ for each $t \in T_s$.*

The reduction theorems presented in this section have a strong consequence: all the already known results for the order-sorted algebras can be lifted to the multi-language world.

6 An Example of Multi-Language Construction

The first theoretical paper addressing the problem of multi-language construction is [30]. The authors study the so-called *natural embedding* (a more realistic improvement of the *lump embedding* [30,7,40,34]), in which Scheme terms can be converted to equivalent ML terms, and vice versa.[7] The novelty in their approach is how they succeed to define boundaries in order to translate values from Scheme to ML. Indeed, the latter does not admit an equivalent representation for each Scheme function. Their solution is to *"represent a Scheme procedure in ML at type $\tau_1 \to \tau_2$ by a new procedure that takes an argument of type τ_1, converts it to a Scheme equivalent, runs the original Scheme procedure on that value, and then converts the result back to ML at type τ_2"*.

Our goal here is not to discuss a fully explained presentation of ML and Scheme languages in the form of order-sorted algebras, but rather to show how we can model the natural embedding construction in our framework. Doing so, we provide a sketchy formalization of Scheme and ML syntax and semantics, and we redirect the reader to [30] for all the languages details.

To provide the semantics of Scheme, we follow the same approach of Goguen

[7] To be specific, the authors combine *"an extended model of the untyped call-by-value lambda calculus, which is used as a stand-in for Scheme, and an extended model of the simply-typed lambda calculus, which is used as a stand-in for ML"*.

et al. [15] where the denotational semantics of the *simple applicative language* (SAL) introduced by Reynolds [42] is given by means of an algebra, exploiting the initiality theorem. Such a language is a "syntactically sugared" version of the untyped lambda calculus with the fixpoint operator, which in turn is very similar to Scheme.

Let $X = \{\, \mathtt{x}_1, \mathtt{x}_2, \dots \}$ be a set of variables and $\mathbb{N}^\diamond$ be the naturals lattice with $\top$ and $\bot$ adjoined. From [46], there exists a complete lattice V such that satisfies the isomorphism $\phi \colon V \cong \mathbb{N}^\diamond + V \mathrel{\diamond\!\!\rightarrow} V$, where $+$ is the disjoint union with minimum and maximum elements identified, and $V \mathrel{\diamond\!\!\rightarrow} V$ is the complete lattice of Scott-continuous functions from V to V. Given $\xi \in \{\, \mathbb{N}^\diamond, V \mathrel{\diamond\!\!\rightarrow} V \,\}$, we define the injections $j_\xi \colon \xi \to \mathbb{N}^\diamond + V \mathrel{\diamond\!\!\rightarrow} V$ and $i_\xi = \phi^{-1} \circ j_\xi$, and the projection $\pi_\xi \colon V \to \xi$ such that $\pi_\xi(v) = (\!|\, \phi(v) \in \xi \mathrel{?} \phi(v) \mathrel{:} \bot \,|\!)$. The set of all Scheme environments is the lattice of all total functions $\mathrm{P} = X \to V$ with componentwise ordering $\rho \sqsubseteq \rho'$ if and only if $\rho(x) \sqsubseteq \rho'(x)$ in V for all $x \in X$. Furthermore, we define auxiliary functions (see [15] for a more detailed explanation) in order to provide the semantics of the language (in the following, $x \in X$ and $n \in \mathbb{N}^\diamond$):

- $\mathit{get}_x \colon \mathrm{P} \to V$, $\mathit{get}_x(\rho) = \rho(x)$ (evaluation function);
- $\mathit{val}_n \colon \mathrm{P} \to V$, $\mathit{val}_n(\rho) = n$ (n-constant function);
- $\mathit{put}_x \colon \mathrm{P} \times V \to \mathrm{P}$, $\mathit{put}_x(\rho, v) = \rho[v/x]$, where $\rho[v/x](x') = (\!|\, x = x' \mathrel{?} v \mathrel{:} \rho(x') \,|\!)$ (environment updating);
- $\mathit{app} \colon V^2 \to V$, $\mathit{app}(v_1, v_2) = (\pi_{V \diamond\!\rightarrow V}(v_1))(v_2)$ (function application);
- $\mathit{nat?} \colon V \to V$, $\mathit{nat?}(v) = (\!|\, v \in \mathbb{N}^\diamond \mathrel{?} \mathit{val}_0 \mathrel{:} \mathit{val}_1 \,|\!)$ (natural predicate);
- $\mathit{proc?} \colon V \to V$, $\mathit{proc?}(v) = (\!|\, v \in V \mathrel{\diamond\!\!\rightarrow} V \mathrel{?} \mathit{val}_0 \mathrel{:} \mathit{val}_1 \,|\!)$ (function predicate);
- given $\hat{e}_i \colon \mathrm{P} \to V$ for $1 \le i \le k$, then $\langle\!| \hat{e}_1, \dots, \hat{e}_k |\!\rangle \colon \mathrm{P} \to V^k$ is defined by $\langle\!| \hat{e}_1, \dots, \hat{e}_k |\!\rangle(\rho) = (\hat{e}_1(\rho), \dots, \hat{e}_k(\rho))$ (target-tupling); and
- given D, D' and D'', then $\mathit{abs} \colon ((D \times D') \mathrel{\diamond\!\!\rightarrow} D'') \to (D \mathrel{\diamond\!\!\rightarrow} (D' \mathrel{\diamond\!\!\rightarrow} D''))$ is defined by $((\mathit{abs}(f))(x))(y) = f(x, y)$ (abstraction); and
- $\mathit{choice} \colon V^3 \to V$ (conditional function), $\mathit{add} \colon V^2 \to V$ (addition), and $\mathit{sub} \colon V^2 \to V$ (subtraction)

$$\mathit{choice}(v_1, v_2, v_3) = \begin{cases} \top & \text{if } v_1 = \top \\ v_2 & \text{if } v_1 = 0 \\ v_3 & \text{if } v_1 \neq 0 \\ \bot & \text{otherwise} \end{cases} \qquad \mathit{add}(v_1, v_2) = \begin{cases} \top & \text{if } v_1, v_2 = \top \\ v_1 + v_2 & \text{if } v_1, v_2 \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases}$$

The definition of *sub* is analogous to the function *add*, with the only difference that, in the second case, $\mathit{sub}(v_1, v_2) = v_1 -_{\mathbb{N}} v_2$, where $v_1 -_{\mathbb{N}} v_2 = \max\{\, v_1 - v_2, 0 \,\}$ for each $v_1, v_2 \in \mathbb{N}$.

The semantics of the language is obtained by defining an algebra $\mathcal{H}$ over a signature $\mathfrak{H}$,[8] then the initiality yields the unique homomorphism from the term algebra. A Scheme term denotes a continuous function in the semantic domain

[8] We do not define $\mathfrak{H}$ explicitly since it can be inferred by the algebra equations below.

$H_{\mathbb{e}} = \mathrm{P} \diamond\!\!\rightarrow V$. The interpretation functions of the operators are defined by the following equations:

$$\begin{array}{ll}
[\![x]\!]_{\mathcal{H}}^{\varepsilon,\mathbb{e}} = get_x & [\![\lambda x]\!]_{\mathcal{H}}^{\mathbb{e},\mathbb{e}}(\hat{e}) = i_{V\diamond\!\!\rightarrow V} \circ abs_{\mathrm{P},V,V}(\hat{e} \circ put_x) \\
[\![\blacksquare]\!]_{\mathcal{H}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(\hat{e}_1,\hat{e}_2) = app \circ \langle\!|\hat{e}_1,\hat{e}_2|\!\rangle & [\![\texttt{proc?}]\!]_{\mathcal{H}}^{\mathbb{e},\mathbb{e}}(\hat{e}) = proc? \circ \hat{e} \\
[\![\overline{n}]\!]_{\mathcal{H}}^{\varepsilon,\mathbb{e}} = val_n & [\![\texttt{if0}]\!]_{\mathcal{H}}^{\mathbb{e}\,\mathbb{e}\,\mathbb{e},\mathbb{e}}(\hat{e}_1,\hat{e}_2,\hat{e}_3) = choice \circ \langle\!|\hat{e}_1,\hat{e}_2,\hat{e}_3|\!\rangle \\
[\![\texttt{+}]\!]_{\mathcal{H}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(\hat{e}_1,\hat{e}_2) = add \circ \langle\!|\hat{e}_1,\hat{e}_2|\!\rangle & [\![\texttt{nat?}]\!]_{\mathcal{H}}^{\mathbb{e},\mathbb{e}}(\hat{e}) = nat? \circ \hat{e} \\
[\![\texttt{-}]\!]_{\mathcal{H}}^{\mathbb{e}\,\mathbb{e},\mathbb{e}}(\hat{e}_1,\hat{e}_2) = sub \circ \langle\!|\hat{e}_1,\hat{e}_2|\!\rangle &
\end{array}$$

For the sake of simplicity, we made a minor change to the language presented in [30]. They have an extra operator `wrong` to print an error message in case of an illegal operation, due to the lack of a type system. For instance, the sum of two functions produces the error `wrong "non-number"`. To avoid to add cases almost everywhere in the definition of the interpretation functions, we let ill-typed terms to denote the value $\perp$ without an explicit encoding of the error message. Furthermore, we denote by $\blacksquare$ the function application.

The ML-like language defined in [30] is an extended version of the simply-typed lambda calculus. As before, we provide its semantics by defining an algebra $\mathcal{M}$ over an order-sorted signature $\mathfrak{M} = \langle S_2, \leq_2, \Sigma_2\rangle$.

Let I (should read 'iota') be a set of *base types* and K a I-sorted set of *base values* $K = \{ K_\iota \mid \iota \in \mathrm{I} \}$. We inductively define the set of *simple types* T: If ι is a base type, then it is a simple type; If τ, τ' are simple types, then $(\tau) \rightarrow (\tau')$ is a simple type (henceforth we omit the parentheses). We abuse notation and extend K to the T-sorted set of *simple values* $K = \{ K_\tau \mid \tau \in \mathrm{T} \}$ where $K_{\tau\rightarrow\tau'} = K_\tau \rightarrow K_{\tau'}$.

The set of all ML enviornments is defined as the set of all total functions $\Delta = Y \rightarrow K$, where $Y = \{ \mathtt{y}_1, \mathtt{y}_2, \dots \}$ is a set of variables disjoint from X (this assumption comes from [30]) and $K = \bigcup_{\tau\in\mathrm{T}} K_\tau$. We instantiate $\mathrm{I} = \{ \mathbb{n} \}$ and $K_{\mathbb{n}} = \mathbb{N}$. The poset $\langle S_2, \leq_2\rangle$ carries all the simple types (i.e., $\mathrm{T} \subseteq S_2$) and the sort $\mathbb{t}$; $\leq_2$ is the reflexive relation on S_2 plus $\tau \leq_2 \mathbb{t}$ for each $\tau \in \mathrm{T}$. An ML term of type τ denotes a total function in $M_\tau = \Delta \rightarrow K_\tau$, and we define $M_{\mathbb{t}} = \Delta \rightarrow K$. Due to the Turing-incompleteness of such a language, we do not need all the mathematical machinery of [15,46] to formalize its semantics.

$$\begin{array}{ll}
[\![y]\!]_{\mathcal{M}}^{\varepsilon,\mathbb{t}} = \delta \mapsto \delta(y) & [\![\lambda y^\tau]\!]_{\mathcal{M}}^{\tau',\tau\rightarrow\tau'}(\hat{t}) = \delta \mapsto k_\tau \mapsto \hat{t}(\delta[k_\tau/y]) \\
[\![\overline{n}]\!]_{\mathcal{M}}^{\varepsilon,\mathbb{n}} = \delta \mapsto n & [\![\blacksquare]\!]_{\mathcal{M}}^{\tau\rightarrow\tau'\,\tau,\tau'}(\hat{t}_1,\hat{t}_2) = \delta \mapsto (\hat{t}_1(\delta))(\hat{t}_2(\delta)) \\
[\![\texttt{+}]\!]_{\mathcal{M}}^{\mathbb{n}\,\mathbb{n},\mathbb{n}}(\hat{n}_1,\hat{n}_2) = \delta \mapsto \hat{n}_1(\delta) + \hat{n}_2(\delta) & [\![\texttt{-}]\!]_{\mathcal{M}}^{\mathbb{n}\,\mathbb{n},\mathbb{n}}(\hat{n}_1,\hat{n}_2) = \delta \mapsto \hat{n}_1(\delta) -_{\mathbb{N}} \hat{n}_2(\delta) \\
[\![\texttt{if0}]\!]_{\mathcal{M}}^{\mathbb{n}\,\tau\,\tau,\tau}(\hat{n},\hat{t}_1,\hat{t}_2) = \delta \mapsto & \\
\qquad (\!|\, \hat{n}(\delta) = 0 \;?\; \hat{t}_1(\delta) \;:\; \hat{t}_2(\delta) \,|\!) &
\end{array}$$

Until now, we have just formalized the single-languages. The multi-language $\mathcal{A}$ that combines Scheme and ML is obtained by requiring $\mathbb{e} \ltimes \tau$ and $\tau \ltimes \mathbb{e}$ in order to use ML terms in place of Scheme terms and vice versa. However, in the simplest version of the natural embedding, *"the system has stuck states, since a boundary might receive a value of an inappropriate shape"* [30]. They restore the type-soundness by first employing dynamic checks, and then by decoupling error-handling from the value conversion through the use of higher-order contracts [12]. We limit ourselves here to describe the first version; the subsequent refinements can be embodied by further complicating the semantics of the boundary functions (we do not have forced any constraints on them).

Since we need a value representing the notion of *stuck state* in ML, we have to extend the algebra $\mathcal{M}$. This is particularly easy by exploiting the underlying framework: We make $\mathcal{M}^{\perp}$ into an order-sorted $\mathfrak{M}$-algebra by defining $M^{\perp}_{\tau} = \Delta^{\perp} \to K^{\perp}_{\tau}$, where $\Delta^{\perp} = Y \to K^{\perp}$, $K^{\perp} = \bigcup_{\tau \in \mathrm{T}} K^{\perp}_{\tau}$, and $K^{\perp}_{\tau} = K_{\tau} \cup \{ \perp \}$, and the T-sorted injection ϕ from M_{τ} to $M^{\perp}_{\tau}$ such that $\varphi(\hat{t}) = \hat{t}$. Now, $\mathcal{M}^{\perp}$ becomes an algebra by letting φ to be an order-sorted $\mathfrak{M}$-homomorphism (this in turn forces $\llbracket - \rrbracket^{w,s}_{\mathcal{M}^{\perp}} = \llbracket - \rrbracket^{w,s}_{\mathcal{M}}$) and letting the interpretation functions to denote the value $\perp$ in the remaining non-yet defined cases (namely, they compute the value $\perp$ whenever one of their arguments is $\perp$).

The boundary function $\alpha_{\mathbb{e},\tau}(\hat{e})$ moves the Scheme value $\hat{e} \colon \mathrm{P} \diamond\!\!\to V$ in M_{τ}:

$$\alpha_{\mathbb{e},\tau}(\hat{e}) = \begin{cases} \alpha^{\mathbb{N}^{\diamond}}_{\mathbb{e},\tau}(\hat{e}) & \text{if } \hat{e} = \mathit{val}_n \text{ for some } n \in \mathbb{N}^{\diamond} \\ \alpha^{V \diamond\!\!\to V}_{\mathbb{e},\tau}(\hat{e}) & \text{otherwise} \end{cases}$$

where $\alpha^{\mathbb{N}^{\diamond}}_{\mathbb{e},\tau}(\mathit{val}_n) = \llparenthesis\, \tau = \mathbb{n} \wedge n \in \mathbb{N} \mathrel{?} \delta \mapsto n \mathrel{:} \perp \rrparenthesis$ and

$$\alpha^{V \diamond\!\!\to V}_{\mathbb{e},\tau}(\hat{e}) = \begin{cases} \delta \mapsto k'_{\tau} \mapsto \llbracket \lambda y^{\tau'} \rrbracket^{\tau'', \tau' \to \tau''}_{\mathcal{M}^{\perp}} (\alpha_{\mathbb{e},\tau''}(\hat{e}' \circ \mathit{put}_x(\perp, \alpha_{\tau',\mathbb{e}}(k_{\tau'})))) \\ \qquad \text{if } \tau = \tau' \to \tau'' \text{ and } \hat{e} = i_{V \diamond\!\!\to V} \circ \mathit{abs}_{\mathrm{P},V,V}(\hat{e}' \circ \mathit{put}_x) \\ \qquad\quad \text{for some } x \in X \text{ and } \hat{e}' \in V \diamond\!\!\to V \\ \perp \\ \qquad\quad \text{otherwise} \end{cases}$$

Vice versa, $\alpha_{\tau,\mathbb{e}}(\hat{t})$ moves values from ML to Scheme. Its definition is analogous to the previous case: $\alpha_{\mathbb{n},\mathbb{e}}(\hat{n}) = \mathit{val}_n$ where $\hat{n} = \delta \mapsto n$, and

$$\alpha_{\tau \to \tau',\mathbb{e}} = \rho \mapsto v \mapsto \llbracket \lambda x \rrbracket^{\mathbb{e},\mathbb{e}}_{\mathcal{H}} (\alpha_{\tau',\mathbb{e}}(\hat{t}(\perp[\alpha_{\mathbb{e},\tau}(v)/y])))$$

These definitions adhere the conversion approach of the natural embedding in [30]: If $\hat{e}$ is the value denoted by a natural number in Scheme, then it is converted — aside from cases deriving from ill-typed terms — by $\alpha^{\mathbb{N}^{\diamond}}_{\mathbb{e},\mathbb{n}}$ to the corresponding constant function denoting the same natural value in ML. Otherwise, if $\hat{e}$ is the value denoted by a Scheme function, then it is mapped by $\alpha^{V \diamond\!\!\to V}_{\mathbb{e},\tau \to \tau'}$ to the ML function with variable x at type $\tau \to \tau'$ such that converts its argument of type τ to the Scheme equivalent by its conversion through $\alpha_{\tau,\mathbb{e}}$ to x. Then it runs the original procedure $\hat{e}$ on it and convert back the result by $\alpha_{\mathbb{e},\tau'}$.

Since the given boundary functions are subsort polymorphic, we can improve the construction and handle all the value conversions with a single polymorphic operator as explained in Sect. 4.1.

7 Concluding Remarks

In this paper, we have addressed the problem of providing a formal semantics to the combination of programming languages, the so-called *multi-languages*. We have introduced a new algebraic framework for modeling this new paradigm, and we have constructively shown how to attain a multi-language specification by only stipulate (1) how the syntactic categories of the single-languages have to be combined and (2) how the values may flow from one language to the other. We have proved the suitability of the framework to unambiguously yield the algebraic semantics of each multi-language term, while simultaneously preserving the single-languages semantics. We have also proved that combining languages is a close operation, i.e., that every multi-language admits an equivalent order-sorted representation. In particular, we have focused our study on the semantic properties of boundary functions in order to provide three different notions of multi-language designed to suit both general and specific cases.

To the best of our knowledge, this is the first attempt to provide a formal semantics of a multi-language independently from the combined languages.

Related Works. Cross-language interoperability is a well-researched area both from theoretical and practical points of view. The most related work to our approach is undoubtedly [30], which provides operational semantics to a combined language obtained by embedding a Scheme-like language into an ML-like language. Such an outcome is achieved by introducing *boundaries*, syntactic constructs that model the flow of values from one language to the other. Ours *boundary functions* draw heavily from their work. Nonetheless, we shift them to a semantic level, in order to several variants of multi-language constructions.

[40,7,21,53,36] take a similar line and combine typed and untyped languages (Lua and ML [40], Java and PLT Scheme [21], or Assembly and a typed functional language [36]), focusing on typing issues and values exchanging techniques. Instead of focusing on a particular problem, we adopt a rather general framework to model languages. This choice abstracts away many low-level details, allowing us to reason on semantic concerns in more general terms, without having to fix any particular pair of languages.

A lot of work has been done on multi-language runtime mechanisms: [20] provides a type system for a fragment of Microsoft Intermediate Language (IL) used by the .NET framework, that allows programmers to write components in several languages (C#, Visual Basic, VBScript, . . .) which are then translated to IL. [22] proposes a virtual machine that can execute the composition of dynamically typed programming languages (Ruby and JavaScript) and statically typed one (C). [5,4] describes a multi-language runtime mechanism achieved by combining single-language interpreters of (different versions of) Python and Prolog.

Future Works. From our perspective, the research presented in this paper opens up on three directions. Firstly, future works should aim to provide an operational semantics to the formalization of multi-languages. Rewriting logic seems the most reasonable approach to unifying the denotational world, presented in this paper, to the operational one [31]. This line of research is particularly useful in order to move towards an implementation of an automatic tool able to combine languages such that the resulting multi-language guarantees the results proved in the paper.

Secondly, future research applies to use the multi-language model in order to study the problem of analyzing multi-language programs. In particular, we aim at investigating how it is possible to obtain analyses of multi-language programs by merging already existing analyses of the single combined languages.

Finally, further studies should investigate the problem of compiling multi-languages. Current compilers are closed tools, non-parametric on language constructs (for instance, we cannot compile a single `if-then-else` term of a standard language like C or Java unless it is plugged into a valid program). Several works on typing [20,1,26], compiling [2,37], and running [50,23] multi-language programs already exist, but without providing a formal notion of multi-language. It would be beneficial to study how their approaches can be applied to the formal framework developed in this paper.

References

1. Abadi, M., Cardelli, L., Pierce, B.C., Plotkin, G.D.: Dynamic typing in a statically typed language. ACM Trans. Program. Lang. Syst. **13**(2), 237–268 (1991)
2. Ahmed, A., Blume, M.: An equivalence-preserving CPS translation via multi-language semantics. SIGPLAN Not. **46**(9), 431–444 (Sep 2011)
3. Alencar, A.J., Goguen, J.A.: Object-oriented specification case studies. In: Lano, K., Haughton, H. (eds.) Object-oriented specification case studies, chap. Specification in OOZE with Examples, pp. 158–183. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK (1994)
4. Barrett, E., Bolz, C.F., Tratt, L.: Unipycation: A case study in cross-language tracing. In: Proceedings of the 7th ACM workshop on Virtual machines and intermediate languages. pp. 31–40. ACM, ACM, New York, NY, USA (2013)
5. Barrett, E., Bolz, C.F., Tratt, L.: Approaches to interpreter composition. Computer Languages, Systems & Structures **44**, 199–217 (2015)
6. Beierle, C., Meyer, G.: Run-time type computations in the warren abstract machine. J. Log. Program. **18**(2), 123–148 (1994)
7. Benton, N.: Embedded interpreters. Journal of functional programming **15**(4), 503–542 (2005)
8. Chisnall, D.: The challenge of cross-language interoperability. Commun. ACM **56**(12), 50–56 (Dec 2013)
9. Dybvig, R.K.: The Scheme Programming Language, 4th Edition. The MIT Press, 4th edn. (2009)
10. Erwig, M.: Specifying type systems with multi-level order-sorted algebra. In: Algebraic Methodology and Software Technology (AMAST'93), pp. 177–184. Springer (1994)
11. Erwig, M., Güting, R.H.: Explicit graphs in a functional model for spatial databases. IEEE Trans. Knowl. Data Eng. **6**(5), 787–804 (1994)
12. Findler, R.B., Felleisen, M.: Contracts for higher-order functions. In: Proceedings of the Seventh ACM SIGPLAN International Conference on Functional Programming. pp. 48–59. ICFP '02, ACM, New York, NY, USA (2002)

13. Flanagan, D.: JavaScript: The definitive guide. O'Reilly Media, Inc. (2006)
14. Furr, M., Foster, J.S.: Checking type safety of foreign function calls. SIGPLAN Not. **40**(6), 62–72 (Jun 2005)
15. Goguen, J.A., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Initial algebra semantics and continuous algebras. J. ACM **24**(1), 68–95 (Jan 1977)
16. Goguen, J.: Tossing algebraic flowers down the great divide (1999)
17. Goguen, J.A.: Semantics of computation. In: Proceedings of the Proceedings of the First International Symposium on Category Theory Applied to Computation and Control. pp. 151–163. Springer-Verlag, London, UK, UK (1975)
18. Goguen, J.A., Diaconescu, R.: An oxford survey of order sorted algebra. Mathematical Structures in Computer Science **4**(3), 363–392 (1994)
19. Goguen, J.A., Meseguer, J.: Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Theor. Comput. Sci. **105**(2), 217–273 (Nov 1992)
20. 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)
21. 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)
22. 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)
23. Grimmer, M., Seaton, C., Schatz, R., Würthinger, T., Mössenböck, H.: High-performance cross-language interoperability in a multi-language runtime. In: Proceedings of the 11th Symposium on Dynamic Languages, DLS 2015, part of SPLASH 2015, Pittsburgh, PA, USA, October 25-30, 2015. pp. 78–90. ACM, New York, NY, USA (2015)
24. Haxthausen, A.E.: Order-sorted algebraic specifications with higher-order functions. Theor. Comput. Sci. **183**(2), 157–185 (1997)
25. Hearn, A.C., Schrüfer, E.: A computer algebra system based on order-sorted algebra. Journal of Symbolic Computation **19**(1), 65 – 77 (1995)
26. Henglein, F., Rehof, J.: Safe polymorphic type inference for scheme: Translating scheme to ML. In: Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995. pp. 192–203. ACM, New York, NY, USA (1995)
27. Johann, P., Ghani, N.: Initial algebra semantics is enough! In: Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications. pp. 207–222. TLCA'07, Springer-Verlag, Berlin, Heidelberg (2007)
28. 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)
29. Liang, S.: Java Native Interface: Programmer's guide and reference. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edn. (1999)
30. Matthews, J., Findler, R.B.: Operational semantics for multi-language programs. SIGPLAN Not. **42**(1), 3–10 (Jan 2007)
31. Meseguer, J., Rosu, G.: The rewriting logic semantics project. Electr. Notes Theor. Comput. Sci. **156**(1), 27–56 (2006)
32. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science **96**(1), 73 – 155 (1992)

33. Milner, R., Tofte, M., Macqueen, D.: The Definition of Standard ML. MIT Press, Cambridge, MA, USA (1997)
34. Ohori, A., Kato, K.: Semantics for communication primitives in a polymorphic language. In: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 99–112. POPL '93, ACM, New York, NY, USA (1993)
35. Osera, P.M., Sjöberg, V., Zdancewic, S.: Dependent interoperability. In: Proceedings of the Sixth Workshop on Programming Languages Meets Program Verification. pp. 3–14. PLPV '12, ACM, New York, NY, USA (2012)
36. Patterson, D., Perconti, J., Dimoulas, C., Ahmed, A.: FunTAL: Reasonably mixing a functional language with Assembly. SIGPLAN Not. **52**(6), 495–509 (Jun 2017)
37. 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 - Volume 8410. pp. 128–148. Springer-Verlag New York, Inc., New York, NY, USA (2014)
38. Poigné, A.: Parametrization for order-sorted algebraic specification. J. Comput. Syst. Sci. **40**(2), 229–268 (1990)
39. Qian, Z.: Another look at parameterization for order-sorted algebraic specifications. J. Comput. Syst. Sci. **49**(3), 620–666 (1994)
40. Ramsey, N.: Embedding an interpreted language using higher-order functions and types. In: Proceedings of the 2003 Workshop on Interpreters, Virtual Machines and Emulators. pp. 6–14. IVME '03, ACM, New York, NY, USA (2003)
41. Ramsey, N.: ML module mania: A type-safe, separately compiled, extensible interpreter. Electron. Notes Theor. Comput. Sci. **148**(2), 181–209 (Mar 2006)
42. Reynolds, J.C.: Definitional interpreters for higher-order programming languages. In: Proceedings of the ACM Annual Conference - Volume 2. pp. 717–740. ACM '72, ACM, New York, NY, USA (1972)
43. Robinson, E.: Variations on algebra: Monadicity and generalisations of equational therories. Formal Aspects of Computing **13**(3), 308–326 (Jul 2002)
44. Rogers, J.: Microsoft JScript.Net programming. Sams, Indianapolis, IN, USA (2001)
45. Schmidt-Schauß, M.: Computational Aspects of an Order-Sorted Logic with Term Declarations, Lecture Notes in Computer Science, vol. 395. Springer-Verlag, Berling, Germany (1989)
46. Scott, D.S., Strachey, C.: Toward a mathematical semantics for computer languages, vol. 1. Oxford University Computing Laboratory, Programming Research Group (1971)
47. Sharan, K.: Scripting in Java: Integrating with Groovy and JavaScript. Apress, Berkely, CA, USA, 1st edn. (2014)
48. Stell, J.G.: A framework for order-sorted algebra. In: Algebraic Methodology and Software Technology, 9th International Conference, AMAST 2002, Saint-Gilles-les-Bains, Reunion Island, France, September 9-13, 2002, Proceedings. pp. 396–410. Springer-Verlag, Berlin, Germany (2002)
49. Tan, G., Morrisett, G.: Ilea: Inter-language analysis across Java and C. SIGPLAN Not. **42**(10), 39–56 (Oct 2007)
50. Trifonov, V., Shao, Z.: Safe and principled language interoperation. In: Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, 22-28 March, 1999, Proceedings. pp. 128–146. Springer-Verlag, Berlin, Germany (1999)

51. Waldmann, U.: Semantics of order-sorted specifications. Theoretical Computer Science **94**(1), 1–35 (1992)
52. Wieringa, R.: A formalization of objects using equational dynamic logic. In: DOOD. pp. 431–452. Springer-Verlag, Berlin, Germany (1991)
53. Wrigstad, T., Nardelli, F.Z., Lebresne, S., Östlund, J., Vitek, J.: Integrating typed and untyped code in a scripting language. In: Hermenegildo, M.V., Palsberg, J. (eds.) Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010. pp. 377–388. ACM (2010)