Available online at [www.sciencedirect.com](http://www.sciencedirect.com/)

[Electronic Notes in Theoretical Computer Science 352 (2020) 79–103](https://doi.org/10.1016/j.entcs.2020.09.005)

[www.elsevier.com/locate/entcs](http://www.elsevier.com/locate/entcs)

Equational Logic and Categorical Semantics for Multi-Languages

Samuele Buroa and Roy Croleb and Isabella Mastroenia[1](#_bookmark0)*,*[2](#_bookmark0)*,*[3](#_bookmark0)

a*University of Verona, Italy and* b*University of Leicester, UK*

**Abstract**

Programming language interoperability is the capability of two programming languages to interact as parts of a single system. Each language may be optimized for specific tasks, and a programmer can take advantage of this. HTML, CSS, and JavaScript yield a form of interoperability, working in conjunction to render webpages. Some object oriented languages have interoperability via a virtual machine host (.NET CLI compliant languages in the Common Language Runtime, and JVM compliant languages in the Java Virtual Machine). A high-level language can interact with a lower level one (Apple’s Swift and Objective-C).

While there has been some research exploring the interoperability mechanisms (Section [1](#_bookmark1)) there is little development of theoretical foundations. This paper presents an approach to interoperability based around theories of equational logic, and categorical semantics.

We give ways in which two languages can be blended, and interoperability reasoned about using equations over the blended language. Formally, *multi-language equational logic* is defined within which one may deduce valid equations starting from a collection of axioms that postulate properties of the combined language. Thus we have the notion of a *multi-language theory* and much of the paper is devoted to exploring the properties of these theories. This is accomplished by way of *category theory*, giving us a very general and flexible semantics, and hence a nice collection of models. Classifying categories are constructed, and hence equational theories furnish each categorical model with an internal language; from this we can also establish soundness and completeness. A set-theoretic semantics follows as an instance, itself sound and complete. The categorical semantics is based on some pre-existing research, but we give a presentation that we feel is easier and simpler to work with, improves and mildly extends current research, and in particular is well suited to computer scientists. Throughout the paper we prove some interesting properties of the new semantic machinery. We provide a small running example throughout the paper to illustrate our ideas, and a more complex example in conclusion.

*Keywords:* categorical logic, equational logic, interoperability, multi-languages, order-sorted signatures and theories, programming languages, subsort polymorphism.

# Introduction

The theory of equational algebra has been a compelling topic since the early days of *universal algebra* [[33,](#_bookmark55)[2](#_bookmark23)]. Research on *equational logic*, addressing the problem of *reasoning by deduction* about term equality, has been prolific (see [[34,](#_bookmark56)[17](#_bookmark39)] for

1 Email: [samuele.buro@univr.it](mailto:samuele.buro@univr.it)

2 Email: [rlc3@le.ac.uk](mailto:rlc3@le.ac.uk)

3 Email: [isabella.mastroeni@univr.it](mailto:isabella.mastroeni@univr.it)

<https://doi.org/10.1016/j.entcs.2020.09.005>

1571-0661/© 2020 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

surveys). In particular, many *sound* and *complete* deduction systems have arisen. For instance, if *Sg* is a *one-sorted*, *many-sorted*, or *order-sorted* signature (sorts and function symbols), such systems appear in [[2](#_bookmark23)], [[10](#_bookmark32)], and [[13](#_bookmark33)], respectively. These developments have had a remarkable impact on *operational semantics* and *automatic theorem proving*. In particular, the pioneering works of [[8,](#_bookmark30)[18](#_bookmark40)] to operationalize equational deduction led to the theory of *term rewriting systems* [[15,](#_bookmark37)[35](#_bookmark57)] which has extensive applications.

*Multi-languages* are programming languages arising from the combination of already existing languages [[27,](#_bookmark49)[1,](#_bookmark24)[32,](#_bookmark54)[9,](#_bookmark31)[14,](#_bookmark34)[26,](#_bookmark48)[24,](#_bookmark46)[21](#_bookmark43)]. Intuitively, terms of multi- languages are obtained by performing cross-language substitutions. For instance, the multi-language designed in [[21](#_bookmark43)] allows programmers to replace ML expressions with Scheme expressions and vice versa. Potential benefits are code reuse and software interoperability. In order to provide a semantics, [[21](#_bookmark43)] introduces new con- structs to regulate the flow of values between the underlying languages, the so-called *boundary functions*.

But what are the formal properties and semantics of multi-languages? Indeed, how general can we be? And in any case, what is a good formal definition of a multi- language in the first place? Buro and Mastroeni [[4](#_bookmark26)] extended the approach of [[21](#_bookmark43)] to the broader class of *order-sorted algebras*, providing a systematic and more general way to define multi-languages, but they did not address equational reasoning. We do so here. In more detail, a multi-language is specified by combining two order-sorted signatures *Sg* 1 and *Sg* 2. But how exactly should the signatures be combined? And what is the formal equational theory of “term interoperability”? More precisely, since equations will not only be defined between two terms both of which are over just *one* of the *Sgi*, but also defined between terms containing symbols from both *Sg* 1 *and Sg* 2, what is a good deduction system for such multi-language equations? We tackle these questions by specifying *multi-language equational logic* which is shown to be sound and complete.

*Contributions in Detail:* Conceptually we lift the basic syntactic theories of *order-sorted equational logic* [[13](#_bookmark33)], and models of the theories, to the algebraic multi- language framework defined by [[4](#_bookmark26)]. The models in [[13](#_bookmark33)] are built from sets, but we adapt the categorical approach in [[20](#_bookmark42)]. *The main contribution is a deductive system for multi-languages with a* sound *and* complete *categorical semantics. We also prove some interesting semantic properties.* There is a running example application throughout the paper, and an outline of a more elaborate application in Section [3.3](#_bookmark19) where we combine an imperative language and a lambda calculus. Our account of order-sorted equational theories builds on and refines [[20](#_bookmark42)], with all our deductive systems presented with uniform and clear inductive rules. Further, we include explicit type information in equality judgements, and include axioms that may be conditional equations. We give a simplified categorical semantics along with categorical type-theory correspondence and classifying category, and also give an explicit connection to free set-algebra semantics.

*Remarks and Intuitions:* We work with *order-sorted signatures* [[13](#_bookmark33)]. As such, our languages enjoy *subsort* polymorphism; there is no provision for *parametric*

polymorphism. Such signatures may satisfy criteria known as monotonicity and regularity. Since the intuitions of these criteria are usually omitted from technical papers, we make some remarks. Language terms *t* are built inductively by applying function symbols *f* to existing terms (which begin as constants). Such symbols *f* can be polymorphically sorted. As such, if one input sort of *f* is a subsort of another input sort, we would like the term *t* to be subsort polymorphic with respect to the output sorts. *Monotonicity* formalises this requirement. One needs to place some control over such polymorphism: one way of doing so is to apply a requirement whereby every term *t* has a least sort. *Regularity* [[13](#_bookmark33)] is such a condition. To ensure every term *t* to has a least sort, we might naively achieve this by requiring each subsort-polymorphic function symbol *f* to have a least input and least output sort, hoping that *t* would do so inductively. We would need to give all constants a least sort to start the induction; and we could simply stipulate that each constant has just one sort. In initially building terms, if there were any function symbol *f* with a (polymorphic) input sort that is a strict subsort of the sort of a constant, such *f* could not be applied to that constant. We are led to refine our naive idea: Fix any *f* . Consider *only those constants that this f can be polymorphically applied to*. Now fix one such constant of sort s; this imposes a lower bound for the input sort of *f* , namely s. We then require that for all such polymorphic instances of *f* , there is one instance with a least input sort *s*, where of course s *≤ s*, and a least output sort. This requirement, formalised, is *Regularity*.

*Paper Structure:* In Section [2](#_bookmark2) we present a transparent rule based deduction system for order-sorted equational logic with conditional axioms, together with a categorical semantics which is proved sound and complete. In Section [3](#_bookmark7) we present a similar set of results for multi-languages. In Section [4](#_bookmark20) we give some further examples of multi-languages. In Section [3.3](#_bookmark19) we present an outline of an extended example in which a traditional lambda calculus and an imperative while language are blended as a single multi-language.

# Order-Sorted Equational Logic

We review order-sorted equational theories (see for example [[13,](#_bookmark33)[20](#_bookmark42)]). Here we give an improved presentation that is syntactically simpler than in loc cit, and further we extend theories to include conditional equational axioms. We also present a detailed but stylistically improved summary of the categorical models from [[20](#_bookmark42)], along with a simpler construction of the classifying category (up to equivalence). We then prove a result relating the classifying category to free order-sorted algebras.

* 1. *Order-Sorted Algebras*

A set *S* is usually regarded as a **set of sorts** or **set of ground types**. Often *S* is partially ordered by *≤*, and then *Sn* , *S × · · ·× S* (*n*-times cartesian product for *n ≥* 1) inherits the pointwise order, with typical instances written *w ≤ wj*. If *w ∈ Sn*, we usually make explicit its components by writing *w* , *s*1*,..., sn*, sometimes referring to a **sequence** of sorts.

We write (*As | s ∈ S*) for a **family indexed by** *S* where each *As* is some- times a set, but more generally an object in a category *C*. We sometimes refer to the family as an *S***-sorted set** (or, an *S***-sorted object**). Such indexed families are simply functors *A* in the presheaf category *Set S* (*CS*). As such, an *S***-sorted function** (*S***-sorted morphism**) *h* : *A → B* is simply a morphism (that is, natural transformation) in *Set S* (*CS*) where *S* is a set or poset.

In this paper all categories have finite products, and functors preserve them up to *isomorphism*. If *A*, *B*, and *A* (1 *≤ i ≤ n*) are objects in a category *C*, we write

*i*

*A × B* for the binary product of *A* and *B*, *A*1 *× · · ·× An* or Q*n Ai* for the finite

1

product of the *Ai*, and 1 for the terminal object. Mediating morphisms for binary product are written *⟨f, fj⟩*, and as usual *f × fj* , *⟨f ◦ π, fj ◦ πj⟩* (for suitable *f* and *fj* and the usual projections). We adopt the obvious extension of notation for finite products. If *A* is an *S*-sorted object and *w* , *s*1*,..., sn*, we denote by *Aw* the product *As*1 *×· · ·×Asn* . Likewise, if *f* is an *S*-sorted morphism, then the morphism *fw* is defined by *fs*1 *×···× fsn* . The coproduct object of *A* and *B* is written *A* + *B*. We write *l*1 *... ln* or *l*1*,..., ln* for a typical finite list, and we may abbreviate just to

***l***. In the special case of a list of sorts *s*1*,..., sn* we usually abbreviate to *w*. When we define order-sorted signatures *Sg* 1, *Sg* 2, *Sg* , *Sgj*, etc., we shall implicitly assume that their posets of sorts are denoted by (*S*1*, ≤*1), (*S*2*, ≤*2), (*S, ≤*), (*Sj, ≤j*), etc., respectively.

Key ingredients of *order-sorted equational theories* are the definitions of *signature* and *algebra*. The former defines the symbols from which the terms of a language are built, and the latter provides terms with a meaning. This meaning can be both set-theoretic and category-theoretic [[13,20](#_bookmark42)].

**Definition 2.1 (Order-Sorted Signature)** *An* ***order-sorted signature*** *Sg is speciﬁed by*

* + - *a poset* (*S, ≤*) *of* ***sorts****;*
    - *a collection of* ***function symbols*** *f* : *s*1*,..., sa → s each with* ***arity*** *a ≥* 1 *and*

(*w, s*) *∈ Sa × S the* ***rank*** *of f where w* , *s*1*,..., sa;*

* + - *a collection of* ***constants*** *k* : *s, each of a* unique ***rank*** *s (just a single sort); and*
    - *a* ***monotonicity requirement*** *that whenever f* : *w*1 *→ s and f* : *w*2 *→ r with*

*w*1 *≤ w*2*, then s ≤ r.*

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

A key property of such signatures *Sg* , related to polymorphism, is *regularity*. We will shortly show how to build a set of *terms* out of *Sg* , and regularity ensures that each term has a unique least sort [[13](#_bookmark33), Proposition 2.10]. (All signatures in this paper are assumed regular).

**Definition 2.2 (Regularity of an Order-Sorted Signature)** *An order-sorted signature Sg is* ***regular*** *if for each f* : *w → s and for each lower bound wl ≤ w the set {* (*wj, sj*) *| f* : *wj → sj ∧ wl ≤ wj }⊆ Sn × S has a minimum, called the* ***least rank of*** *f* ***with respect to*** *wl.*

*—*

Γ*,x* : *s,* Γ*j ▶ x* : *s*

(*∀*1 *≤ i ≤ a*) Γ *▶ ti* : *si*

*—* (*k* : *s*) Γ *▶ k* : *s*

Γ *▶ t* : *s*

(Fn) (*f* : *s*1*,..., sa → s*)

Γ *▶ f* (*t*1*,..., ta*): *s*

Γ *▶ t* : *r*

(*s ≤ r*)

Fig. 1. Proved terms generated by an order-sorted signature *Sg* .

**Raw terms** over a signature *Sg* are defined by *t* ::= *x | k | f* (*t*1*,..., ta*) with *x ∈ Var* (a countably infinite set of variables), *k* a constant, and *f* a function symbol with arity *a*. A **context** is a finite list of ordered pairs *x* : *s* formed by a variable *x* and a sort *s* in *Sg* . We usually define a context by writing Γ , [*x*1 : *s*1*,..., xn* : *sn*]. We denote context concatenation of Γ and Γ*j* by Γ*,* Γ*j*. We work with sorting judgements of the form Γ *▶ t* : *s*. Those that are generated by the sorting rules in Figure [1](#_bookmark3) are called **proved terms**. Note that a term *t* may have more than one sort *s* for which Γ *▶ t* : *s* is a proved term. However there is always a unique least sort.

**Lemma 2.3 (Terms Have A Least Sort)** *Suppose that* Γ *is a context and t a raw term for a given regular signature Th. If there is any sort s for which* Γ *▶ t* : *s is a proved term, then there is a least such sort, st.*

**Proof.** One uses rule induction. The proof is easy, though in the literature a key step is often omitted. By induction, for the rule Fn, one easily uses regularity to obtain a sort, say *s*ˇ, such that Γ *▶ f* (*t*1*,..., ta*): *s*ˇ. Now *s*ˇ is a candidate for the least sort of *f* (*t*1*,..., ta*). Most authors state that such a sort *s*ˇ is least. This is true, but proving it so requires a separate (though trivial) rule induction. *2*

We denote by *t*[*u/x*] the **substitution** of the raw term *u* for the variable *x* in *t*, and by *t*[***u****/****x***] the **simultaneous substitution** of raw terms ***u*** , *u*1*,..., un* for variables ***x*** , *x*1*,.* *, xn*.

**Definition 2.4 (Inclusion Structure and FPI-category)** *An* ***inclusion structure*** I *in a category C is speciﬁed by a subposet (subcategory)* I *of C such that*

* + - * *for any two objects A and B of C, the unique morphism A* > *B in* I*, if any, is a monic in C;*
      * *for any object A in C, the identity id A is in* I *(so* I *is a* luff *subcategory: it has the same objects as C);*
      * *if ι*1 : *A*1 > *B*1 *and ι*2 : *A*2 > *B*2 *are morphisms in* I*, then so to is ι*1 *×ι*2 : *A*1 *×*

*A*2 > *B*1 *× B*2*.*

*A pair* (*C,* I) *is called an* ***FPI-category****. The intuition is that products model lists of sorts, and inclusions model subsort polymorphism.*

Thus, an FPI-category can be used as the basis for a definition of an *algebra* for a signature, namely

**Definition 2.5 (Order-Sorted Algebra)** *Given an order-sorted signature Sg, an Sg* ***-algebra* A** *in an FPI-category* (*C,* I) *is speciﬁed by*

* + - *an object* J*s*)**A** *in C for each sort s and object* J*w*)**A** , J*s*1)**A** *×· · · ×* J*sn*)**A** *for each w* , *s*1*,..., sn ∈ Sn;*
    - *morphisms* J*f* : *w → s*)**A** : J*w*)**A** *→* J*s*)**A** *and* J*k*)**A** : 1 *→* J*s*)**A** *for each f* : *w → s*

*and k* : *s; and*

* + - *a morphism* J*s ≤ r*)**A** : J*s*)**A** > J*r*)**A** *in* I *for each s ≤ r in S, where we set*

J*s ≤ s*)**A** , *id* J*s*)

*such that if the function symbol f appears with more than one rank f* : *w*1 *→ s and f* : *w*2 *→ r in Sg with s*1*,..., sa* , *w*1 *≤ w*2 , *r*1*,..., ra, then the following diagram commutes:*

J*s*1)**A** *× ··· ×* J*sa*)**A**

J*f* : *w*1*→s*)**A**

J*s*)**A**

J*s*1*≤r*1)**A***×···×*J*sa≤ra*)**A**

J*r*1)**A** *× ··· ×* J*ra*)**A**

J*f* : *w*2*→r*)**A**

J*s≤r*)**A**

J*r*)**A**

From now on, we drop the algebra subscript and the ranks of function symbols in the semantic brackets whenever they are clear by context.

**Definition 2.6 (Order-Sorted Homomorphism)** *Let Sg be an order-sorted sig- nature and let* **A** *and* **B** *be Sg-algebras. An Sg* ***-homomorphism*** *h* : **A** *→* **B** *is an S-sorted morphism* (*hs* : J*s*)**A** *→* J*s*)**B** *| s ∈ S*) *such that given f* : *s*1*,..., sa → s, k* : *s, and s ≤ r in Sg the following diagrams commute:*

J*k*)**B**

J*s*1)**A** *× ··· ×* J*sa*)**A**

*hs*1 *×···×hsa *

J*s*1)**B** *× ··· ×* J*sa*)**B**

J*f* )**A**

J*f* )**B**

J*s*)**A**

J*s*)**A**

*hs*

J*s*)**B**

J*s≤r*)**A**

J*r*)**A**

1 J*k*)**A**

J*s*)**A**

*hs*

J*s*)**B**

*hs*

J*s*)**B**

J*s≤r*)**B**

*hr*

J*r*)**B**

*We deﬁne hw* , *hs*1 *× ··· × hsa provided that w* , *s*1*,..., ×sa.*

Given an order-sorted signature *Sg* , the class of all the order-sorted *Sg* -algebras and the class of all the order-sorted *Sg* -homomorphisms form a category denoted by *OSAlg* (*C,* I)*Sg* .

If Γ *▶ t* : *s* isa proved term ina *regular* signature *Sg* and Γ , [*x*1 : *s*1*,..., xn* : *sn*], any *Sg* -algebra **A** induces a (unique) morphism from JΓ)**A** , J*s*1)**A** *×· · · ×* J*sn*)**A** to J*s*)**A** in *C* according to the inductive definition that appears in Figure [2](#_bookmark4). We denote such an arrow by JΓ *▶ t* : *s*)**A** and we refer to it as the **semantics of** Γ *▶ t* : *s*.

Since terms can be assigned different types in one given context, we should consider whether the definition in Figure [2](#_bookmark4) is a sensible one. As such, we have the following lemma, where one sees that semantics of substitutions of terms is given

*—*

JΓ*,x* : *s,* Γ*j ▶ x* : *s*) , *π* : JΓ) *×* J*s*) *×* JΓ*j*) *→* J*s*)

*—* (*k* : *s*)

JΓ *▶ k* : *s*) , J*k*)*◦*!: JΓ) *→* 1 *→* J*s*)

(*∀*1 *≤ i ≤ a*) JΓ *▶ ti* : *si*) , *mi* : JΓ) *→* J*si*)

Q*a*

(*f* : *s*1*,..., sa → s*)

JΓ *▶ f* (*t*1*,..., ta*): *s*) , J*f* ) *◦ ⟨m*1*,..., ma⟩* : JΓ) *→* (

JΓ *▶ t* : *s*) = *m* : JΓ) *→* J*s*)

1 J*si*)) *→* J*s*)

(*s ≤ r*)

JΓ *▶ t* : *r*) = J*s ≤ r*) *◦ m* : JΓ) *→* J*s*) *→* J*r*)

Fig. 2. Categorical semantics for proved terms.

as usual by morphism composition:

**Lemma 2.7 (Well-Defined Semantics)** *Given a proved term* Γ *▶ t* : *s and an algebra* **A***:*

* + - * *The semantic morphism* JΓ *▶ t* : *s*) *is unique; that is, the assignment ξ '→* J*ξ*)

*is a total function.*

* + - * *The algebra induces a functor* (*S, ≤*) *→* I *between posetal categories, where s ≤ sj '→* J*s ≤ sj*) : J*s*) > J*sj*)*, and so as a consequence the semantics can be factored through the morphism* JΓ *▶ t* : *st*)*, that is to say,* JΓ *▶ t* : *s*) = J*st ≤*

*s*) *◦* JΓ *▶ t* : *st*)*.*

* + - * *Let* Γ , [*x*1 : *s*1*,..., xn* : *sn*]*. If we have proved terms* Γ *▶ t* : *s and* Γ*j ▶ ui* : *si where* 1 *≤ i ≤ n, then* Γ*j ▶ t*[***u****/****x***]: *s and* JΓ*j ▶ t*[***u****/****x***]: *s*) = JΓ *▶ t* : *s*) *◦ ⟨*JΓ*j ▶ u*1 : *s*1)*,...,* JΓ*j ▶ un* : *sn*)*⟩.*

*Equations* between proved terms are defined only for *coherent* signatures. To define coherence, first let *≡* be the symmetric and transitive closure of *≤*. The equivalence classes induced by *≡* on *S* are the **connected components** of (*S, ≤*); and (*S, ≤*) is **locally filtered**, if for every two sorts *sj* and *sjj* in the same *connected component* there is a sort *s* such that *sj, sjj ≤ s*. Then a signature *Sg* is said to be **coherent** if and only if it is regular and locally filtered. The intuition is that terms of sort *sj* and *sjj* respectively could potentially be judged equal if they have a “common super-sort” *s*.

**Definition 2.8 (Order-Sorted Equation and Satisfaction)** *Let Sg be a co- herent signature. An* ***equation (in-context)*** *in Sg is denoted by* Γ *▶ t* = *tj* : *s, where*

* + - * *there are sorts sj, sjj such that* Γ *▶ t* : *sj and* Γ *▶ tj* : *sjj are proved terms in Sg;*
      * *sj and sjj fall in the same connected component of* (*S, ≤*)*; and*
      * *s is a common supersort of sj and sjj (which exists by the coherence condition).*

*A* ***conditional equation (in-context)*** *in Sg is a list of m* +1 *equations-in-*

*context (where m ≥* 1*) suggestively denoted by* *m* Γ *▶ tα* = *tj* : *sα* =*⇒* Γ *▶ t* =

*α*=1

*α*

Let Γ , [*x*1 : *s′ ,..., xn* : *s′* ] be a context in (AxSub) and (AxCSub).

1

(AxSub)

*n*

Γ *▶ t* = *tj* : *s ∈ Ax* (*∀*1 *≤ i ≤ n*) Γ*j ▶ ui* : *sj* Γ*j ▶ t*[***u****/****x***]= *tj*[***u****/****x***]: *s*

*i*

*m* Γ *▶ tα* = *tj* : *sα* =*⇒* Γ *▶ t* = *tj* : *s ∈ Ax*

*α*

1

(*∀*1 *≤ i ≤ n*) Γ*j ▶ ui* : *sj*

*i*

(AxCSub)

*α*

(*∀*1 *≤ α ≤ m*) Γ*j ▶ tα*[***u****/****x***]= *tj* [***u****/****x***]: *sα*

Γ*j ▶ t*[***u****/****x***]= *tj*[***u****/****x***]: *s*

(Ref)

Γ *▶ t* : *s*

Γ *▶ t* = *t* : *s*

(Sym)

Γ *▶ t* = *tj* : *s*

Γ *▶ tj* = *t* : *s*

(Trans)

Γ *▶ t* = *tj* : *s* Γ *▶ tj* = *tjj* : *s*

Γ *▶ t* = *tjj* : *s*

Γ *▶ t* = *tj* : *s*

(Subsort) (*s ≤ r*) Γ *▶ t* = *tj* : *r*

Γ *▶ t* : *s* (*∀*1 *≤ i ≤ n*) Γ*j ▶ ui* = *uj* : *si*

*i*

(Cong) (Γ , [*x*1 : *s*1*,..., xn* : *sn*])

Γ*j ▶ t*[***u****/****x***]= *t*[***uS****/****x***]: *s*

Fig. 3. Theorems generated by an order-sorted theory *Th* , (*Sg, Ax* ).

*tj* : *s. We say that an algebra* **A *satisfies*** *an equation if* JΓ *▶ t* : *s*) = JΓ *▶ tj* : *s*)*. We say that an algebra* **A *satisfies*** *a conditional equation if for all morphisms u* : *U →* JΓ)*,* JΓ *▶ tα* : *sα*)*◦u* = JΓ *▶ tj* : *sα*)*◦u implies* JΓ *▶ t* : *s*)*◦u* = JΓ *▶ tj* : *s*)*◦u.*

We define an **order-sorted theory** *Th* , (*Sg, Ax* ) to be a pair consisting of a signature *Sg* and a **set of axioms** *Ax* . Each axiom is either an equation or a conditional equation. The **theorems** of the theory *Th* are those *equations* generated by the rules of *equational logic* in Figure [3.](#_bookmark6)

*α*

**Lemma 2.9 (Generalised Substitution)** *The following rule is admissible by a routine rule induction*

Γ *▶ t* = *tj* : *s* (*∀*1 *≤ i ≤ n*) Γ*j ▶ ui* : *si*

(Sub) (Γ , [*x*1 : *s*1*,..., xn* : *sn*])

Γ*j ▶ t*[***u****/****x***]= *tj*[***u****/****x***]: *s*

Let *Th* , (*Sg, Ax* ) be an order-sorted theory. If an *Sg* -algebra **A** satisfies all the axioms in *Ax* , we call **A** a **model** of *Th*. The **category of models** *OSMod* (*C,* I)*Th* is the full subcategory of *OSAlg* (*C,* I)*Sg* given by all the models of *Th* in (*C,* I).

**Lemma 2.10 (Satisfaction is Well-Defined)** *As a consequence of Lemma* [*2.7*](#_bookmark5)*, satisfaction is well-deﬁned up to subsort-polymorphic equality, as follows: Suppose that we have a theorem* Γ *▶ t* = *tj* : *s satisﬁed in a model* **A***. If* Γ *▶ t* = *tj* : *s*ˆ *is also a theorem, then it too is satisﬁed.*

**Proof.** The existence of least sorts s*t* and s*t′* means that *s* and *s*ˆ are connected, and so have a super-type *sj*. Thus each term has this type *sj*, and the result follows by using factorisation from Lemma [2.7](#_bookmark5) and the left-cancellation properties

of monomorphisms. *2*

The category *Pres×,*>((*C,* I)*,* (*D,* J)) is defined by having objects functors *F* : *C → D* such that finite products are preserved and *F* restricts to a functor *F|*I : I *→* J (that is monics are also preserved). Suppose that we have a model **A** in *OSMod* (*C,* I)*Th* . Then there is a model *F∗***A** in *OSMod* (*D,* J)*Th* that is, roughly speaking, defined by “taking the image of *Sg* in (*C,* I) induced by the model **A**, and *applying F* ”. Equally one may “apply *F* to homomorphisms of models” and this process (which is absolutely standard in categorical type theory/logic; see for example [[6,16](#_bookmark38)]) leads to a functor *Ap***A** : *Pres×,*>((*C,* I)*,* (*D,* J)) *→ OSMod* (*D,* J)*Th* . A **classifying category** *Cl* (*Th*) for a theory *Th* is an FPI-category such that there is an equivalence of categories

*Ap***G** : *Pres×,*>(*Cl* (*Th*)*,* (*D,* J)) *OSMod* (*D,* J)*Th*

and thus models of *Th* in (*D,* J) correspond to such structure preserving functors with source *Cl* (*Th*).

**Theorem 2.11 (Existence of Classifying Category)** *There is an FPI- category Cl* (*Th*) *constructed out of the syntax of Th, in which there is a* ***generic*** *model of Th with the property that equality of morphisms corresponds to derivability of term equations. At an abstract level this notion is standard in categorical type theory/logic; see for example [*[*6*](#_bookmark28)*,*[*16*](#_bookmark38)*]. We feel that our concrete construction is simpler than that found in [*[*20*](#_bookmark42)*], and regard this as a small contribution.* Such existence proofs are notoriously tricky to get completely correct, and there are notable errors in the literature. We use matching contexts and permutation invariance [[7,](#_bookmark29)[29](#_bookmark51)] to replace the usual substitutions that rename variables, and we think this makes our proofs simpler to state and prove (and hence less error prone)*.*

**Proof.** Let *Var* be a (countable) fixed set of variables *{V*1*, V*2*,.. .}*. We call the context Γ***s*** , [*V*1 : *s*1*,..., Vn* : *sn*]) the **primary** context for any sorts *s*1*,.* *, sn*. In

general, below, the metavariables *x*, *y*, *z* etc, possibly subscripted, *range over Var* . Thus Γ , [*x*1 : *s*1*,..., xn* : *sn*] is a typical context as before, and we say that any such context **matches *s*** , *s*1*,.* *, sn*.

First we define the objects **T**(Γ) of *FOSAlg* . Of course **T**(Γ) must be an *S*-sorted set, and the components are sets of equivalence classes

**T**(Γ)*s* , *Term*(Γ)*s/ ∼* where *Term*(Γ)*s* , *{ t |* Γ *▶ t* : *s* in *Th }*

where we define the equivalence relation *t ∼ tj* just in case we can derive Γ *▶ t* = *tj* : *s*

in *Th*, and write [ *t* ] for a typical equivalence class.

Let Γ and Γ*j* match *s*1*,..., sn* and *sj ,..., sj*

respectively. The morphisms

1 *m*

*h* : **T**(Γ) *→* **T**(Γ*j*) must be *S*-sorted functions (*hs* : **T**(Γ)*s →* **T**(Γ*j*)*s | s ∈ S*). These

are specified by lists *hs* , *{*Γ*j ▶ t*1*,..., tn}* where *ti ∈ Term*(Γ*j*)*si* and where

*hs*([ *t*ˆ] *∈* **T**(Γ)*s*) , [ *t*ˆ[*t*1*,..., tm/x*1*,..., xm*]] *∈* **T**(Γ*j*)*s*

It is easy to check this is well defined. Note that if

*h* , *{*Γ*j ▶ t*1*,..., tn}* : **T**(Γ) *→* **T**(Γ*j*) and *hj* , *{*Γ*jj ▶ tj ,..., tj }* : **T**(Γ*j*) *→* **T**(Γ*jj*)

1 *m*

then we have *h ◦ hj* defined by

*{*Γ*jj ▶ t*1[*tj ,..., tj /x*1*,..., xm*]*,..., tn*[*tj ,..., tj*

*/x*1*,..., xm*]*}*

1 *m* 1 *m*

It is tedious but routine to verify that this gives rise to a category, relying crucially on the substitution rules for equation derivation. *2*

**Theorem 2.12 (Soundness and Completeness)** *Let Th be an order-sorted theory.* Γ *▶ t* = *tj* : *s is a theorem of Th if and only if* Γ *▶ t* = *tj* : *s is satis- ﬁed by every model of Th.*

**Proof.** Soundness follows by rule induction for Figure [3](#_bookmark6). For completeness, suppose that Γ *▶ t* = *tj* : *s* is satisfied in any model. Then in particular it is satisfied in the generic model **G** in the classifying category *Cl* (*Th*). Thus we have JΓ *▶ t* : *s*)**G** = JΓ *▶ tj* : *s*)**G** and so we have (Γ *| t*)= (Γ *| tj*) which holds precisely when Γ *▶ t* = *tj* : *s* is a theorem. *2*

We conclude this section with a new result, although it is motivated by analogous theorems [[28](#_bookmark50)]. The proof also makes use of matching contexts and permutations of variables.

**Theorem 2.13 (Relationship to Free Algebras)** *There is an equivalence be- tween FPI-categories Cl* (*Th*) *and* (*FOSAlg op,* J) *where FOSAlg is the cate- gory of free order-sorted algebras over ﬁnite sets of variables, and order- sorted homomorphisms. Moreover the equivalence is given by an FPI-functor* Φ: *Cl* (*Th*)(*FOSAlg op,* J)*.*

**Proof.** With a view to showing that (*FOSAlg op,* J) is an FPI-category, we shall show that *FOSAlg* is has finite coproducts, and then define J. Given objects **T**(Γ) and **T**(Γ*j*) then the binary coproduct object is given by **T**(Δ , Γ***s****,* Γ***s0*** ) where the sorts match Γ and Γ*j* respectively. The coproduct insertions are given by *{*Δ *▶ v*1*,..., vn}* and *{*Δ *▶ vn*+1*,.* *, vn*+*m}*. Given morphisms

*{*Γ*jj ▶ t*1*,..., tn}* : **T**(Γ) *→* **T**(Γ*jj*) and *{*Γ*jj ▶ tj ,..., tj }* : **T**(Γ*j*) *→* **T**(Γ*jj*)

1 *m*

then the mediating morphism is *{*Γ*jj ▶ t*1*,..., tn, tj ,..., tj }*. Note that **T**(∅) is the

1 *m*

initial object.

Suppose that *si ≤ ri* for each 1 *≤ i ≤ n*. Then there is an epic morphism *i* ,

*{*Γ***s*** *▶ x*1*,..., xn}* : **T**(Γ***r***) *→* **T**(Γ***s***); it’s easy to verify that this is an epimorphism, and hence yields a monomorphism in *FOSAlgop*. The luff subcategory J has all of its morphisms the monomorphisms *iop* , *{*Γ***s*** *▶ x*1*,..., xn}* : **T**(Γ***s***) *→* **T**(Γ***r***). This is certainly an inclusion category.

Now we prove the equivalence. We define a functor Φ: *Cl* (*Th*) *→* (*FOSAlgop,* J) as follows. Given a morphism (Γ *| t*1) *...* (Γ *| tm*): ***s*** *→* ***r*** then Φ sends this to

*{*Γ***s*** *▶ πt*1*,..., πtm}* : **T**(Γ***r***) *→* **T**(Γ***s***)

where permutation *π* is specified by *π* : *xi '→ Vi*. We check this is well defined. Suppose that

(Γ *| t*1) *...* (Γ *| tm*)= (Γ*j | tj* ) *...* (Γ *| tj* )*.*

1 *m*

We need to check that

*{*Γ***s*** *▶ πt*1*,..., πtm}* , Φ((Γ *| t*1) *...* (Γ *| tm*))

= Φ((Γ*j | tj* ) *...* (Γ *| tj* )) , *{*Γ***s*** *▶ πjtj ,..., πjtj }*

1 *m* 1 *m*

By definition we have Γ *▶ tj* = *ρtj* : *rj* where *ρ* is specified by *ρ* : *xj '→ xi*. We can

*j* *i*

deduce, using substitution rules for equations, that *π*Γ *▶ πtj* = *π*(*ρtj* ): *rj* and this

*j*

is exactly Γ***s*** *▶ πtj* = *πjtj* : *rj* as required, since *πj* = *π ◦ ρ*. We feel that the use of permutations, while equivalent to the use of simultaneous variable renamings by substitution, improves readability and more importantly simplifies calculations by making use of judgements that are permutation invariant.

*j*

(Φ is essentially surjective):

For any object ***s*** in *Cl* (*Th*) we have Φ(***s***) = **T**(Γ***s***). But one easily shows that for any **T**(Γ) where Γ matches ***s***, we have **T**(Γ) *∼*= **T**(Γ***s***) where the inverse homomorphisms “swap variables” *xi* and *Vi*.

(Φ is faithful): Let

Φ((Γ *| t*1) *...* (Γ *| tm*)) = Φ((Γ*j | tj* ) *...* (Γ *| tj*

))*.*

1 *m*

We need to check that Γ *▶ tj* = *ρtj* : *rj*. By the assumption we have

*j*

*{*Γ***s*** *▶ πt*1*,..., πtm}* = *{*Γ***s*** *▶ πjtj ,..., πjtj }*

1 *m*

Hence Γ***s*** *▶ πtj* = *πtj* : *rj*. Therefore we can deduce that Γ *▶ tj* = (*π—*1 *◦ πj*)*tj* : *rj*;

*j j*

we are done since *π—*1 *◦ πj* = *ρ*.

(Φ is full): Let

*{*Γ***s*** *▶ t*1*,..., tm}* : Φ(***r***)= **T**(Γ***r***) *→* Φ(***s***)= **T**(Γ***s***)

Then (Γ***s*** *| t*1*,..., tm*) is the appropriate *Cl* (*Th*) morphism.

(Φ is an object of *Pres×,*>(*Cl* (*Th*)*,* (*FOSAlg op,* J))): We need to check that Φ*|*I : I *→* J where I is the inclusion category of *Cl* (*Th*). Let ***s*** *≤* ***r***. Note that Φ((Γ *| x*1) *...* (Γ *| xn*): ***s*** *→* ***r***) is the monomorphism *{*Γ***s*** *▶ x*1*,..., xn}* : Γ***r*** *→* Γ***s*** *2*

# Multi-Language Equational Logic

* 1. *Fundamentals of Multi-Languages*

*Throughout this section we often refer to a* **Running Example**, introduced below and subsequently extended, to illustrate how the theory works in a concrete setting

0 : nat zero constant

s: nat *→* nat successor function

+: nat*,* nat *→* nat addition operator

* + 1. *Sg* 1 operators.

(*∀c ∈* Char) *c* : chr character constant

next: chr *→* chr next character function

+: str*,* str *→* str string concatenation

* + 1. *Sg* 2 operators.

Fig. 4. Operators of order-sorted signatures *Sg* 1 and *Sg* 2.

(see Section [3.3](#_bookmark19) for the outline of a more complex example).

**Running Example.** Our example is defined using the following order-sorted sig- natures:

* The signature *Sg* 1 defines the symbols of a language for constructing simple mathematical expressions over natural numbers in Peano’s notation. Let the poset of sorts (*S*1*, ≤*1) of *Sg* 1 be a poset with a single sort nat denoting the type of natural numbers, and let the operators be those in Figure [4a.](#_bookmark8)
* Let *c ∈* Char , *{*a*,* b*,...,* z*}* be the metavariable ranging over a finite set Char of characters. The signature *Sg* 2 defines a language to build strings over Char. The set of sorts *S*2 of *Sg* 2 carries the sort str for strings and the sort chr for characters. The subsort relation *≤*2 is the reflexive relation on *S*2 plus chr *≤*2 str (*i.e.*, characters are one-symbol strings), and the operator symbols in *Sg* 2 appear in Figure [4b.](#_bookmark8)

We model *Sg* 1 and *Sg* 2 by the order-sorted algebras **A**1 and **A**2 (see Figure [5](#_bookmark9)) in (*Set, Incl* ), the FPI-category of sets with inclusion functions forming the inclusion structure. The symbol *cj* in the definition of Jnext)**A** denotes the character that follows *c* in Char (assuming the standard alphabetical order).

2

**Remark 3.1** The forthcoming definitions and results gradually define and illus- trate multi-languages, and give relationships between multi-languages and order- sorted languages. A multi-language signature [3.2](#_bookmark10) is specified as two order-sorted signatures (as in the **Running Example**) together with *an interoperability rela- tion* between the two signatures. This determines the terms of the multi-language. Note that the relation is *not* a universal property of the underlying signatures; and also note a multi-language signature *explicitly* provides users with the *original two* language specifications. We show in [3.3](#_bookmark11) that there are nice notions of categories of signatures, both order-sorted and multi-language. We shall also see that we can exhibit a functor that maps a multi-language signature to an order-sorted signa- ture (the *associated signature* [3.4](#_bookmark12)), blending the two original signatures into one. After defining multi-language algebras [3.5](#_bookmark13) and homomorphisms [3.6](#_bookmark14), Theorem [3.7](#_bookmark15) will provide, via associated signatures, a clear semantic relationship between multi

Sorts Jnat)**A**1 , N

Jchr)**A**2 , Char

Jstr)**A** , Char*∗*

2

Operators J0)**A**1 , *∗ '→* 0: 1 *→* N

Js)**A**1 , *n '→ n* + 1 : N *→* N

J+)**A** , (*n*1*, n*2) *'→ n*1 + *n*2 : N2 *→* N

1

(*∀c ∈* Char) J*c*)**A**2 , *∗ '→ c* : 1 *→* Char

Jnext)**A** , *c '→ cj* : Char *→* Char

2

J+)**A** , (*s*1*, s*2) *'→ s*1*s*2 : (Char*∗*)2 *→* Char*∗*

2

Subsorts Jchr *≤*2 str)**A** , *c '→ c* : Char *→* Char*∗*

2

Fig. 5. Categorical semantics of *Sg* 1 and *Sg* 2.

and order-sorted languages. In particular, it suggests how to give the definitions of (multi-language) terms, equations, and satisfaction using the associated signature. We can then reason equationally about *interoperability* of the two given languages. This takes us to Section [3.2](#_bookmark16) where we study equational reasoning in detail.

In order to define multi-language signatures we introduce some crucial notation. We denote by + the **disjoint union** of two sets: the insertion morphisms that form a coproduct in the category of sets are injective functions, thus they have left inverses (and one has a model of disjoint union). In the following, if *S*1 and *S*2 are two sets of sorts and *s ∈ Si* with *i* , 1*,* 2, we write

*si* for the element *ιi*(*s*) *∈ S*1 + *S*2 where *ιi*(*s*) , (*s, i*) *∈ S*1 *× {*1*}∪ S*2 *× {*2*}*

*Thus in relationships si* *sj we have s ∈ Si and sj ∈ Sj*. This is a very useful

notation but perhaps requires care on first reading. Moreover, if *w* , *s*1*,..., sn ∈*

*j*

*Sn*, then we write *wi* for (*s*1)*i,...,* (*sn*)*i*.

*i*

**Definition 3.2 (Multi-Language Signature)** *A* ***multi-language signature***

*SG* , (*Sg* 1*, Sg* 2*,* ) *is*

* + *a pair of order-sorted signatures Sg* 1 *and Sg* 2 *with posets of sorts* (*S*1*, ≤*1) *and*

(*S*2*, ≤*2)*, respectively; and*

* + *a* (binary) relation ***join*** *over S*1 + *S*2 *such that si* *sj*

*j*

*with i, j ∈ {*1*,* 2*} and*

*i ≤*= *j.*

The idea is that if *si* *sj* , and Γ *▶ t* : *s* is a proved term in *one* language, then *t* can be used in place of a term *tj* such that Γ*j ▶ tj* : *sj* in the *other* language: as in [[21](#_bookmark43)], “ML code can be used in place of Scheme code”. This is made precise in

*j*

due course.

**Definition 3.3** *OSSg is the* ***category of order-sorted signatures*** *with mor- phisms h* : *Sg* 1 *→ Sg* 2 *given by*

* *a monotone function h* : (*S*1*, ≤*1) *→* (*S*2*, ≤*2) *(where we will write h*(*w*) ,

*h*(*s*1)*,..., h*(*sn*) *for w* , *s*1*,..., sn ∈ Sn), and*

1

* *a mapping h from the operators in Sg* 1 *to those in Sg* 2 *that preserves rank: given k* : *s in Sg* 1*, then h*(*k*): *h*(*s*) *in Sg* 1*; and given f* : *w → s in Sg* 1*, then h*(*f* ): *h*(*w*) *→ h*(*s*) *in Sg* 2*.*

*Moreover, we denote by MLSg the* ***category of multi-language signatures***

*in which a morphism*

*H* , (*h*1*, h*2): (*Sg* 1*, Sg* 2*,* ) *→* (*Sgj , Sgj ,* *j*)

1 2

*is deﬁned by two morphisms h*1 : *Sg* 1 *→ Sgj*

1

*and h*2 : *Sg* 2 *→ Sgj*

*in OSSg such that*

*they preserve the join relation, namely si* *sj in* (*Sg* 1*, Sg* 2*,* ) *implies* (*hi*(*s*)*i* *j*

2

*j*

(*hj*(*sj*))*j in* (*Sgj , Sgj ,* *j*)*.*

1 2

**Definition 3.4 (Associated Signature)** *Let SG* , (*Sg* 1*, Sg* 2*,* ) *be a multi- language signature. The* ***associated signature*** *SG of SG is the order-sorted sig- nature deﬁned as follows:*

* *the poset of sorts is given by* (*S*1 + *S*2*, ≤*)*, where si ≤ rj if i* = *j and s ≤i r;*
* *if f* : *w → s is an operator in Sgi for some i* , 1*,* 2*, then fi* : *wi → si is a function symbol in SG;*
* *if k* : *s is a constant in Sgi for some i* , 1*,* 2*, then ki* : *si is a constant in SG; and*
* *a* ***conversion operator*** *‹→s ,s′* : *si → sj for each constraint si* *sj .*

*i j j j*

The **associated signature functor** (*−*): *MLSg → OSSg* maps each multi- language signature *SG* , (*Sg* 1*, Sg* 2*,* ) to its associated signature *SG* , and each multi-language signature morphism *H* : *SG → SGj* to the order-sorted signature morphism *H* : *SG → SGj* given by *H*(*si*) , (*hi*(*s*))*i* for each *s ∈ Si* (hence *si ∈ S*1 + *S*2) and *H*(*fi*) , (*hi*(*f* ))*i* for each *f ∈ Sgi* (hence *fi* in *SG* ).

**Running Example.** (*−*) embeds the multi-language signature *SG* into *OSSg* , pro- viding the order-sorted version *SG* of the multi-language. *SG* generates *Sgi*-terms (see Section [3.2](#_bookmark16)) as well as hybrid multi-language terms involving conversion op- erators such as [*c* : chr2*,n* : nat1] *▶* +1( *‹→*chr2*,*nat1 (*c*)*, n*): nat1. From now on, we use colours in the examples for disambiguating the left and the right inclusion in place of subscripts 1 and 2. Moreover, we use an infix notation whenever the operators lend themselves well to do so. That is, the previous term is represented by [*c* : chr*,n* : nat] *▶ ‹→*chr*,*nat (*c*) + *n* : nat.

Such a functor outlines an *embedding* of multi-language signatures into order- sorted signatures, enabling us to see a multi-language as an ordinary language.

Indeed, it is easy to see that (*−*) is both injective on objects and a faithful functor.

**Definition 3.5 (Multi-Language Algebra)** *Let SG* , (*Sg* 1*, Sg* 2*,* ) *be a multi- language signature. An SG* ***-algebra* A** *in an FPI-category* (*C,* I) *is given by*

* + *a pair of order-sorted algebras* **A**1 *and* **A**2 *in* (*C,* I) *over Sg* 1 *and Sg* 2*, respec- tively; and*

*j*

* + *a* ***boundary*** *morphism* J*si* *sj* )**A** : J*s*)**A**

*j*

*i*

*→* J*sj*)**A**

*in C for each constraint*

*si* *sj .*

*j*

An algebra sets out the meaning of a multi-language: The meaning of the un- derlying languages, and *how* terms of sort *s ∈ Si* can be interpreted as terms of sort *sj ∈ Sj*. Put differently, “boundary morphisms regulate the flow of values across **A**1 and **A**2” [[22](#_bookmark44)].

**Definition 3.6 (Multi-Language Homomorphism)** *Let SG* , (*Sg* 1*, Sg* 2*,* ) *be a multi-language signature, and let* **A** *and* **B** *be two SG-algebras. An SG* ***- homomorphism*** *h* : **A** *→* **B** *is given by a pair of order-sorted homomorphisms h*1 : **A**1 *→* **B**1 *and h*2 : **A**2 *→* **B**2 *such that they commute with boundary functions, namely, if si* *sj , then the following diagram commutes:*

*j*

*j*

*j*

J*s*)**A***i*

J*si* *s′* )**A**

J*sj*)**A**

(*hi*)*s*

J*s*)**B***i*

J*si* *s′* )**B**

(*hj* )*s′*

J*sj*)**B**

*j*

Given a multi-language signature *SG* , the class of all *SG* -algebras and *SG* - homomorphisms form a category denoted by *MLAlg* (*C,* I)*SG* . We have a simple connection between this category and *OSAlg* (*C,* I)*SG* , outlined in Theorem [3.7](#_bookmark15), after more of the Running Example.

*j*

**Running Example.** Suppose we are interested in a multi-language *SG* ,

(*Sg* 1*, Sg* 2*,* ) according to the following specifications:

* + terms denoting natural numbers can be used in place of characters according to the function chr: N *→* Char that maps the natural number *n* to the *n*-th character symbol modulo *|*Char*|*; and
  + terms denoting strings can be used in place of natural numbers according to the function len: Char*∗ →* N, namely the length of the string.

In order to get such a multi-language, we provide (1) the join relation on *S*1+*S*2

and (2) a boundary morphism J*si* *sj* )**A** : J*s*)**A**

*j*

*j*

*i*

*j*

*→* J*sj*)**A**

for each constraint *si* *sj*

introduced by :

* + nat1 chr2 and nat1 str2 with boundaries Jnat1 chr2)**A**(*n*) , Jnat1

str2)**A**(*n*) , chr(*n*); and

* + chr2 nat1 and str2 nat1 with boundaries Jchr2 nat1)**A**(*c*) , len(*c*)=1 and Jstr2 nat1)**A**(*s*) , len(*s*).

The next theorem yields a formal correspondence between multi-languages and order-sorted languages: We can make a multi-language signature *SG* into an order- sorted one by applying the functor (*−*), and thus blending the underlying languages. Nevertheless, we do not lose any semantical information if we consider the category of algebras over *SG* and *SG* .

**Theorem 3.7** *There is a natural isomorphism between the category of multi- language algebras over SG and the category of order-sorted algebras over the as- sociated signature SG*

*η* : *MLAlg* (*C,* I) *⇒ OSAlg* (*C,* I) *◦* (*−*) *inducing MLAlg* (*C,* I)*SG ∼*= *OSAlg* (*C,* I)*SG*

*where there are functors MLAlg* (*C,* I): *MLSg → Cat* op *and OSAlg* (*C,* I): *OSSg → Cat* op *that map signatures to their category of algebras in* (*C,* I)*.*

**Proof.** The functors are defined on objects by *MLAlg* (*C,* I)(*SG* ) , *MLAlg* (*C,* I)*SG* and *OSAlg* (*C,* I)(*Sg* ) , *OSAlg* (*C,* I)*Sg* . Now, let *SG* , (*Sg* 1*, Sg* 2*,* ) and *SGj* , (*Sgj , Sgj ,* *j*), and let *H* , (*h*1*, h*2): *SG → SGj* be a signature morphism between

1 2

them. We shall define a functor *H∗* : *MLAlg* (*C,* I)*SG′ → MLAlg* (*C,* I)*SG* and let *MLAlg* (*C,* I)(*H*) , *H∗*. Let **A** be an *SGj*-algebra in *MLAlg* (*C,* I)*SG′* . Then, the multi-language *SG* -algebra *H∗***A** is defined as follows:

* We define its order-sorted components (*H∗***A**)1 and (*H∗***A**)2. Let *i* , 1*,* 2:
* the interpretation of sorts is given by J*s*)(*H∗* **A**)*i* , J*hi*(*s*))**A***i* for each *s ∈ Si*;
* given the function symbol *f* : *w → s* in *Sgi*, we define J*f* : *w → s*)(*H∗* **A**)*i* ,

J*hi*(*f* ): *hi*(*w*) *→ hi*(*s*))**A***i* ;

* the constant symbol *k* : *s* in *Sgi* is interpreted by letting J*k*)(*H∗* **A**)*i* ,

J*hi*(*k*))**A***i* ; and

* J*s ≤i r*)(*H∗* **A**)*i*

*i*

*Sgi*.

, J*hi*(*s*) *≤j*

*hi*(*r*))**A***i*

for each subsort constraint *s ≤i r* in

The fact that (*H∗***A**)*i* is a proper order-sorted *Sgi*-algebra is ensured by the properties of the (multi-language) signature morphism *H*.

* Boundary morphisms are defined by J*si* *sj* )*H∗* **A** , J*hi*(*s*) *j hj*(*sj*))**A** for each

*j*

constraint *si* *sj*

*j*

in *SG* .

In order to define the action of *H∗* on homomorphisms, suppose that *g* : **A** *→* **B** is a multi-language *SGj*-homomorphism in *MLAlg* (*C,* I)*SG′* . Then, (*H∗g*)*i* : (*H∗***A**)*i →* (*H∗***B**)*i* is defined by the *Si*-sorted morphisms

* (*H∗g*)*i*

*j s*

, (*gi*)*hi*(*s*), which is well-defined since *gi* : **A***i →* **B***i* is an order-sorted

*Sgi*-homomorphism and J*hi*(*s*))**A***i* = J*s*)(*H∗* **A**)*i* .

The commutativity of the diagram in Definition [3.6](#_bookmark14) is given by a tedious but simple diagram chase.

J*hi*(*s*))**A***i*

J*hi*(*s*) *′hj* (*s′*))**A**

J*hj*(*sj*))**A**

J*si* *s′* )*H∗* **A**

*j*

*j*

*j*

J*s*)(*H∗* **A**)*i * J*s* )(*H∗* **A**)*j*

(*gi*)*hi*(*s*)

(*H∗g*)*i* 

J*s*)(*H∗* **B**)*i*

*s*

J*si* *s′* )*H∗* **B**

*j*

*s′*

(*H∗g*)*j*

J*sj*)(*H∗* **B**)

*j*

(*gj* )*hj* (*s′*)

J*hi*(*s*))**B***i*

J*hi*(*s*) *′hj* (*s′*))**B**

J*hj*(*sj*))**B**

We next define a functor *h∗* : *OSAlg* (*C,* I)*Sg*

*j*

2

*→ OSAlg* ((*C,* I))*Sg*1

and set

*OSAlg* (*C,* I)(*h*) , *h∗*, where *h* : *Sg* 1 *→ Sg* 2 is an order-sorted signature morphism.

This is similar to the definition of *H∗* above. First pick any (order-sorted) *Sg* 2- algebra **A**. We need to define the order-sorted *Sg* 1-algebra *h∗***A**. We define

* + objects J*s ∈ S*1)*h∗* **A** , J*h*(*s*))**A** in *C* and hence J*w*)*h∗* **A** , J*h*(*s*1))**A** *× · · · ×*

J*h*(*sn*))**A** for *w* , *s*1*,..., sn ∈ Sn*;

1

* + morphisms J*f* : *w → s ∈ Sg* 1)*h∗* **A** , J*h*(*f* ): *h*(*w*) *→ h*(*s*) *∈ Sg* 2)**A** : J*w*)*h∗* **A** *→*

J*s*)*h∗* **A** and morphisms J*k ∈ Sg* 1)*h∗* **A** , J*h*(*k*))**A** : 1 *→* J*s*)*h∗* **A**; and

* + a morphism J*s ≤*1 *r ∈ S*1)*h∗* **A** , J*h*(*s*) *≤*2 *h*(*r*))**A** : J*s*)*h∗* **A** > J*r*)*h∗***A** in I.

We omit the verification that semantics of operators commutes with the semantics of subsorting, although this is essentially immediate since **A** is an *Sg* 2-algebra. Now let *g* : **A** *→* **B** be an order-sorted *Sg* 2-homomorphism. We define the *Sg* 1-

homomorphism *h∗*(*g*): *h∗***A** *→ h∗***B** by setting the components to be (*h∗*(*g*))*s∈S* ,

1

*gh*(*s*) : J*h*(*s*))**A** *→* J*h*(*s*))**B**.

Now we define the natural transformation *η* by specifying the components *ηSG* : *MLAlg* (*C,* I)*SG → OSAlg* (*C,* I)*SG* . Pick any *SG* -homomorphism *h* : **A** *→* **B**. First we define the (order-sorted) *SG* -algebra *ηSG* **A** by setting

* + J*si ∈ S*1 + *S*2)*ηSG* **A** , J*s*)**A***i* in *C* and J*w*)*ηSG* **A** , J*s*1)**A***i ×· · ·×* J*sn*)**A***i* for each

*w* , *s*1*,..., sn ∈* (*S*1 + *S*2)*n*;

* + morphisms J*fi* : *wi → si*)*ηSG* **A** , J*f* : *w → s*)**A***i* and J*ki*)*ηSG* **A** , J*k*)**A***i* : 1 *→*

J*s*)**A***i* ; and

* + J*‹→s ,s′* : *si → sj* )*η*

**A** , J*si* *sj* )**A** : J*s*)**A** *→* J*sj*)**A**

*i j j SG*

*j i j*

* + J*si ≤ rj*)*ηSG* **A** , J*s ≤i r*)**A***i* : J*s*)**A***i* > J*r*)**A***i* in I for each *si ≤ rj* in *S*1 + *S*2

(don’t forget that *i* = *j*).

where the required commutation properties follow immediately since the **A***i* are *Sgi*-algebras. Second we define *SG* -homomorphism *ηSG* (*h*): *ηSG* **A** *→ ηSG* **B**. Since by the definition of *h* there are *Sgi*-homomorphisms *hi* : **A***i →* **B***i*, we can define *ηSG* (*h*)*si∈S*1+*S*2 , (*hi*)*s* : J*s*)**A***i →* J*s*)**B***i* , and *ηSG* (*h*)*si* inherits the required commut- ing properties from *h* (commuting with *‹→si,s′* ), and from the *hi* (commuting with

*j*

the *fi*).

One can see that *ηSG* is an isomorphism by reversing the construction and by noting there is a one-one correspondence between the sorts, operators, and subsort constraints in *SG* and those in *Sg* 1 and *Sg* 2. This is straightforward. More difficult is naturality of *η* which we show next.

Let *H* , (*h*1*, h*2): *SG* , (*Sg* 1*, Sg* 2*,* ) *→ SGj* , (*Sgj , Sgj ,* *j*). Then we need to

1 2

show that the diagram below commutes.

*MLAlg* (*C,* I)*SG*

*H∗*

*MLAlg* (*C,* I)*SG′*

*ηSG*

*ηSG′*

*OSAlg* (*C,* I)*SG*

*H*

*∗*

*OSAlg* (*C,* I)*SG′*

Pick any morphism *g* : **A** *→* **B** in *MLAlg* (*C,* I)*SG′* . First we need to show that

*ηSG* (*H*

*∗***A**)= *H*

*∗*

(*ηSG′* **A**). Let us check only that these *SG* -algebras provide equal

meaning to sorts *si ∈ S*1 + *S*2 where we have (*hi*(*s*))*i ∈ Sj* + *Sj* .

1 2

J*si*)*ηSG* (*H∗* **A**) = J*s ∈ Si*)(*H∗* **A**)*i* = J*hi*(*s*) *∈ Si*)**A***i*

*j*

= J(*hi*(*s*))*i ∈ Sj* + *Sj* )*η*

1 2 *SG′*

**A** = J*H*(*si*))*η*

*SG′*

**A** = J*si*)*H∗* (*η* **A**)

Now *g* furnishes us with *SGj*-homomorphisms *gi* : **A***i →* **B***i*, and we need to show that

*SG′*

*∗ ∗ j j*

*ηSG* (*H*

*g*) = *H* (*ηSG′ g*): J*hi*(*s*) *∈ Si*)**A***i →* J*hi*(*s*) *∈ Si*)**B***i*

is an equality of *SG* -homomorphisms. But this follows from the following calculation on components of *S*1 + *S*2-sorted morphisms

*∗*

*∗ ∗*

(*ηSG* (*H g*))*si* = ((*H g*)*i*)*s* = (*gi*)*hi*(*s*) = (*ηSG′ g*)(*hi*(*s*))*i* = (*ηSG′ g*)(*H*(*si*)) = (*H* (*ηSG′ g*))*si*

and the proof is completed. *2*

**Running Example.** The multi-language semantics of the term introduced in the previous example is given by the algebra *ηSG* **A** which leads to J[*c* : chr*,n* : nat] *▶*

*‹→*chr*,*nat (*c*) + *n* : nat)*ηSG* **A** = (*c, n*) *'→ n* + 1 : Char *×* N *→* N.

* 1. *Equational Reasoning in a Multi-Language Context*

In this section we define multi-language proved terms, and give them a semantics. Then we define multi-language equations and semantic satisfaction. From this we can define theories and models, and hence prove soundness and completeness.

Let *SG* , (*Sg* 1*, Sg* 2*,* ) be a multi-language signature. A **(multi-language) proved term** Γ *▶ t* : *si* is a proved term over the associated signature *SG* . It follows that if Γ *▶ t* : *s* is a proved term over *Sgi*, then Γ *▶ t* : *s* is a proved term in *SG* , where Γ *▶ t* : *s* , Γ *▶ t* : *s* and

* *s* , *si* for each *s ∈ Si*; and Γ , [*x*1 : *s*1*,..., xn* : *sn*] for each context Γ ,

[*x*1 : *s*1*,.* *, xn* : *sn*] over *Sgi*;

* *t* is recursively defined over the syntax of raw terms generated by *Sgi*: *x* , *x*; *k* , *ki*; and *f* (*t*1*,..., ta*) , *fi*(*t*1*,.* *tn*).

(*eq* 1*,*1) [*n* : nat] *▶* 0 + *n* = *n* : nat

(*eq* 1*,*2) [*n* : nat*,m* : nat] *▶* s(*n*) + *m* = s(*n* + *m*): nat

(*eq* 1*,*3) [*n* : nat*,m* : nat] *▶ n* + *m* = *m* + *n* : nat

1. *Th*1 axioms.

(*eq* 2*,*1) *▶* next(a) = b: chr

(*eq* 2*,...*) *▶ ...* = *...* : chr

(*eq* 2*,*26) *▶* next(z) = a: chr

1. *Th*2 axioms.

Fig. 6. Axioms of order-sorted theories *Th*1 and *Th*2.

Due to the injectivity of this construction, we shall refer to it as the *inclusion* of an order-sorted term into the multi-language, and we informally say that a multi- language “contains” the underlying languages. Furthermore, the definition of multi- language terms also includes *hybrid* terms that are not the result of the inclusion of an order-sorted term but which are constructed using the conversion operators in the associated signature.

Given a multi-language *SG* -algebra **A**, the categorical **semantics of a (multi- language) term** Γ *▶ t* : *si* is the order-sorted semantics of Γ *▶ t* : *si* induced by *ηSG* **A**, namely JΓ *▶ t* : *si*)**A** , JΓ *▶ t* : *si*)*ηSG* **A**. As expected, a multi-language preserves the semantics of the underlying terms:

**Proposition 3.8** *Let* **A** *be a multi-language SG-algebra over SG* , (*Sg* 1*, Sg* 2*,* )*. If* Γ *▶ t* : *s is a proved term over Sgi, then* JΓ *▶ t* : *s*)**A** = JΓ *▶ t* : *s*)*ηSG* **A** = JΓ *▶ t* : *s*)**A***i .*

Regularity and coherence for a multi-language signature *SG* , (*Sg* 1*, Sg* 2*,* ) are defined with respect to its associated signature. That is, *SG* is **regular** (resp., **coherent**) if *SG* is regular (resp., coherent). It is immediate that *SG* is regular (resp., coherent) if and only if *Sg* 1 and *Sg* 2 are regular (resp., coherent).

**Definition 3.9 (Multi-Language Equation and Satisfaction)** *Let SG be a coherent multi-language signature. A (conditional)* ***equation*** *for SG is an order- sorted (conditional) equation over SG. A multi-language algebra* **A *satisfies*** *any such (conditional) equation if the (conditional) equation is satisﬁed by ηSG* **A***.*

An immediate consequence of Proposition [3.8](#_bookmark18) is that every *Sgi*-equation satisfied by **A***i* is also satisfied by the multi-language algebra **A** (in its inclusion form provided by the mapping (*−*)).

A **multi-language theory** *TH* , (*SG, AX* ) is a pair of a multi-language sig- nature *SG* and a set of (conditional) multi-language equations *AX* over *SG* , namely the **axioms** of the theory. The **theorems** of *TH* are the equations Γ *▶ t* = *tj* : *si* derivable from (*SG, AX* ). A multi-language *SG* -algebra that satisfies all the axioms in *AX* is said a **model** of *TH* , and *MLMod* (*C,* I)*TH* denotes the full subcategory of models of *MLAlg* (*C,* I)*TH* . We now introduce the categories of theories in order to define the *associated theory* of a multi-language theory. From now on, when we write order-sorted theories *Th*1, *Th*2, *Th*, *Thj*, etc., we assume they are defined as *Th*1 , (*Sg* 1*, Ax* 1), *Th*2 , (*Sg* 2*, Ax* 2), *Th* , (*Sg, Ax* ), *Thj* , (*Sgj, Axj*), etc., respectively.

**Running Example.** Let *Th*1 , (*Sg* 1*, Ax* 1) and *Th*2 , (*Sg* 2*, Ax* 2) be the order- sorted theories over *Sg* 1 and *Sg* 2 axiomatized by the equations provided in Figure [6.](#_bookmark17)

We can generate from *Ax* 1 and *Ax* 2 a set *AX* of multi-language equations by apply- ing (*−*) to each equation. For instance, (*eq* 1*,*1) , [*n* : nat] *▶* 0 + *n* = *n* : nat becomes (*eq* 1*,*1) , [*n* : nat] *▶* 0 + *n* = *n* : nat. Note that a substantial change occurs when mapping an order-sorted equation to a multi-language one. Consider again (*eq* 1*,*1). A substitution in the order-sorted world can only plug *t*, where Γ *▶ t* : nat in *Sg* 1, into the variable *n* : nat. However, a multi-language substitution can substitute any *tj*, where Γ*j ▶ tj* : nat in *SG* , for *n* : nat in the lifted equation (*eq* 1*,*1)—including, crucially, the possibility that *tj* is a hybrid multi-language term.

The behaviour of boundary morphisms can be axiomatized by adding the fol- lowing equations to *AX* :

(*EQ* 1) *▶ ‹→*nat*,*chr (0)= a: chr

(*EQ* 2) *▶ ‹→*nat*,*str (0)= a: str

(*EQ* 3) [*c* : chr] *▶ ‹→*chr*,*nat (*c*)= s(0): nat

(*EQ* 4) [*c* : chr] *▶ ‹→*str*,*nat (*c*)= s(0): nat

(*EQ* 5) [*n* : nat] *▶ ‹→*nat*,*chr (s(*n*)) = next(*‹→*nat*,*chr (*n*)): chr

(*EQ* 6) [*n* : nat] *▶ ‹→*nat*,*str (s(*n*)) = next(*‹→*nat*,*str (*n*)): str

(*EQ* 7) [*s* : str*,v* : str] *▶ ‹→*str*,*nat (*s* + *v*) = *‹→*str*,*nat (*s*) + *‹→*str*,*nat (*v*): nat

**Definition 3.10** *Let OSTh be the* ***category of order-sorted theories*** *whose morphisms h* : *Th*1 *→ Th*2 *are signature morphisms h* : *Sg* 1 *→ Sg* 2 *in OSSg that preserve theorems, that is, if* Γ *▶ t* = *tj* : *si is a theorem of Th*1 *with* Γ , [*x*1 : *s*1*,..., xn* : *sn*]*, then* Γ*j ▶ h*(*t*) = *h*(*tj*): *h*(*si*) *is a theorem of Th*2*, where* Γ*j* , [*x*1 : *h*(*s*1)*,..., xn* : *h*(*sn*)] *and h*(*t*) *and h*(*tj*) *are inductively deﬁned over the syntax according to the action of h on function symbols and constants.*

*The* ***category of multi-language theories*** *is denoted by MLTh and a theory morphism H* : (*SG* 1*, AX* 1) *→* (*SG* 2*, AX* 2) *is a signature morphism H* : *SG* 1 *→ SG* 2 *in MLSg such that if* Γ *▶ t* = *tj* : *si is a theorem of* (*SG* 1*, AX* 1) *with* Γ , [*x*1 : *s*1*,..., xn* : *sn*]*, then* Γ*j ▶ H*(*t*) = *H*(*tj*): *H*(*si*) *is a theorem of* (*SG* 2*, AX* 2)*,*

*where* Γ*j* , [*x*1 : *H*(*s*1)*,..., xn* : *H*(*sn*)]*.*

Functors *MLAlg* (*C,* I) and *OSAlg* (*C,* I) can be easily extended to *MLMod* (*C,* I): *MLTh → Cat* op and *OSMod* (*C,* I): *OSTh → Cat* op, respectively, such that they associate to each signature its corresponding category of models. Then, (*−*): *MLTh → OSTh* is defined by *TH* , (*SG, AX* ) on objects and by *H* on morphisms *H* : *TH* 1 *→ TH* 2.

**Proposition 3.11** *MLMod* (*C,* I) *and OSMod* (*C,* I) *◦* (*−*) *are isomorphic functors. Let η be the natural isomorphism between them and TH a multi-language the- ory. Then, ηTH is the isomorphism between categories MLMod* (*C,* I)*TH and OSMod* (*C,* I)*TH .*

**Theorem 3.12 (Soundness and Completeness)** *Let TH be a multi-language theory.* Γ *▶ t* = *tj* : *s is a theorem of TH if and only if* Γ *▶ t* = *tj* : *s is satisﬁed by every model of TH .*

* 1. *An Extended Example*

The Running Example has the sole purpose of illustrating our theory in an elemen- tary way: we are very much aware of its limitations. Here we give a taste of a more realistic example. For space reasons, we can convey only the main ideas: full details are in an extended version of the paper [[3](#_bookmark25)].

We define a new multi-language by blending a simple functional core with a minimal imperative language. The former is of course suited to writing programs that are easier to reason about, whereas the latter provides a more straightfor- ward procedural and low-level approach to software development. We formalise the *simply-typed lambda calculus* and a simple *imperative language* as two equational theories, and we blend them together in order to provide the gist of an interoper- ability between the functional and imperative paradigms. More complex examples can be built along the lines of the one presented here.

We assume Imp and *λ*⭢ to be the signatures of a small imperative language and the simply-typed lambda-calculus, respectively. In the following, we use colours blue for denoting Imp code and red for *λ*⭢-terms. The interoperability we wish to provide should allow the use of *λ*⭢-terms as Imp-expressions and vice versa. For instance, we would like to write multi-language programs such as x = (*λ*y:int . y + y) 1, which encodes the assignment to the variable x of the value obtained by applying the *λ*⭢-function (*λ*y:int . y + y) to the Imp-numeral 1. Although minimal, its interpretation requires several applications of the boundary functions: First, we need to compute the result of the function application, which in turn needs the evaluation of 1 as a *λ*⭢-term. Then, the resulting term has to be converted back to an Imp-numeral in order to be assigned to x.

The multi-language signature *λ*⭢-Imp providing the desired interoperability is given by coupling the signatures Imp and *λ*⭢ with the join relation specified by e exp and exp e, where e is the sort of Imp-expressions and exp the sort of *λ*⭢-terms. The semantics of the generated multi-language programs is obtained by introducing a boundary function for each -constraint. For instance, given a standard denotational semantics for both the underlying languages, the boundary function Je exp) can provide each Imp-expression with a *λ*⭢-meaning in the follow- ing way: Let *e* be the semantics of such an Imp-expression. We can first transform a *λ*⭢-environment to an Imp-environment, run *e* on its conversion, and then move the resulting Imp-values to suitable *λ*⭢-values.

The equational axiomatization of such a boundary function can be specified by the following multi-language equations: (1) *‹→*e*,*exp (*i*) = *i* and (2) *‹→*e*,*exp (*x*) = *x*. The first equation allows *λ*⭢-integers to be converted to Imp-numerals of the same form. In more realistic examples, the conversion of values across languages should

take into consideration different machine representations (for instance, if the *λ*⭢ language does not admit an explicit representation of integers, we may convert the integer *i* to its corresponding Church-numeral). Equation (2) provides a match between Imp and *λ*⭢ variables with the same name. This enables a natural way for moving stored values across the two languages. For instance, the multi-language program x = (*λ*y:int . y + y) z acts in the same way of the previously described

one but applying the *λ*⭢-function to the value stored in the Imp-variable z.

On the other hand, the boundary function Jexp e) works in a dual manner for providing *λ*⭢-terms with an Imp-meaning. Given all these specifications, the equational logic provides the following chain of equalities:

x = (*λ*y:int . y + y) 1 = x = 1 + 1 = x = 2 = x = 2

# Further Multi-Language Constructions

Buro and Mastroeni [[4](#_bookmark26)] provides three different multi-language constructions based on boundary morphism properties (although in their work, morphisms are only set- theoretic functions). In Section [3](#_bookmark7), we studied a categorical equational logic for the simplest construction. Here we briefly discuss the other two, each a refinement of the first.

The first refinement of multi-language signatures is accomplished by allowing all

conversion operators *‹→**s ,s′* : *si → sj*

in the associated signature to be replaced by

*i j j*

subsort polymorphic operators *‹→* : *si → sj*

*j*

that *do not carry* any sort information.

One can check that any associated signature *SG* defined in this way remains an order-sorted signature if and only if the following additional constraint holds for *SG* :

*si* *sj* , *ri* *rj* , and *si ≤i ri* imply *sj ≤j rj*

(1)

*j j j j*

Multi-language algebras are then restricted by the following monotonicity require- ment:

*si* *sj* , *ri* *rj* , and *si ≤i ri* imply J*sj ≤j rj*)**A** *◦* J*si* *sj* )**A** = J*ri* *rj* )**A** *◦* J*sj ≤j rj*)**A**

*j j j j j*

*j*

(2)

In this new multi-language construction, we can prove the following version of The- orem [3.7](#_bookmark15):

**Theorem 4.1** *Assume (*[*1*](#_bookmark21)*) and (*[*2*](#_bookmark22)*) for multi-language signatures and algebras, respectively. There is a natural isomorphism η* : *MLAlg* (*C,* I) *⇒ OSAlg* (*C,* I) *◦* (*−*) *inducing MLAlg* (*C,* I)*SG ∼*= *OSAlg* (*C,* I)*SG , where there are functors MLAlg* (*C,* I): *MLSg → Cat* op *and OSAlg* (*C,* I): *OSSg → Cat* op *that map signatures to their category of algebras in* (*C,* I)*.*

**Proof.** The proof is almost identical to the proof of Theorem [3.7](#_bookmark15). That each

*ηSG* **A** is a proper order-sorted algebra boils down to the fact that each J*‹→* : *si →*

*sj* )*η*

*j*

*SG*

**A** commutes with the desired morphisms in I; but this commutativity follows

immediately from ([2](#_bookmark22)). *2*

The second refinement of multi-language signatures aims to achieve a multi- language construction which consists only of the union of the underlying languages, that is no conversion operator is added to the associated signature and single- language operators are not tagged. Such a construction is particularly useful when mode ling the extension of a language rather than the union of two already existing languages.

The notion of multi-language signature is refined by assuming that

* (*S*1 + *S*2*,* ) is a poset; and
* *f* : *w → s* in *Sgi* and *f* : *wj → sj* in *Sgj* with *wi* *wj* , then *si* *sj* .

*j j*

and the associated signature *SG* is defined as follows:

* the poset of sorts is given by (*S*1 + *S*2*, ≤*), where *si ≤ rj* if *i* = *j* and *s ≤i r* or

*i ≤*= *j* and *si* *rj*;

* if *f* : *w → s* is a function symbol in *Sgi*, then *f* : *wi → si* is a function symbol in *SG* , and similarly for constants.

Multi-language algebras now force boundary morphisms to act as subsort mor- phisms. This means that if the function symbol *f* appears with more than one rank *f* : *w → s* and *f* : *wj → r* in *Sgi* and *Sgj* , respectively, with (*s*1)*i,...,* (*sa*)*i* , *w* *wj* , (*r*1)*j,...,* (*ra*)*j*, then the following diagram commutes:

J*s*1)**A***i ×· · · ×* J*sa*)**A***i*

J*f* : *w*1*→s*)**A***i*

J*s*)**A***i*

J(*s*1)*i* (*r*1)*j* )**A***×···×*J(*sa*)*i* (*ra*)*j* )**A**

J*r*1)**A***j ×· · · ×* J*ra*)**A***j*

J*f* : *w*2*→r*)**A***j*

J*si* *rj* )**A**

J*r*)**A***j*

**Theorem 4.2** *Assume these new hypotheses for multi-language signatures and al- gebras, respectively. There is a natural isomorphism η* : *MLAlg* (*C,* I) *⇒ OSAlg* (*C,* I)*◦* (*−*) *inducing MLAlg* (*C,* I)*SG ∼*= *OSAlg* (*C,* I)*SG , where (as before) there are functors MLAlg* (*C,* I): *MLSg → Cat* op *and OSAlg* (*C,* I): *OSSg → Cat* op*.*

# Conclusions and Future Research

Equational logic is a simple fragment of first-order logic with several applications to computer science [[12,](#_bookmark35)[34,](#_bookmark56)[17](#_bookmark39)]. In this paper, we have addressed the problem of equational deduction in a multi-language context. We have lifted the order-sorted equational logic of [[13](#_bookmark33)] to the algebraic framework of multi-languages introduced by [[4](#_bookmark26)], and we have proved the soundness and the completeness of the resulting deduction system. The main benefit of the theoretical development in this paper is a solid mathematical foundation for reasoning about equalities in a multi-language context.

Among all the applications, one motivation for extending the theory of equa- tional logic to a multi-language context resides in the possibility of providing *oper- ational semantics* to multi-languages, in a similar way to [[11](#_bookmark36)]. In future work, we plan to investigate this in the context of *rewriting logic* [[31](#_bookmark53)], where axioms might be partitioned into a set *R* of *rewriting rules* and a set *E* of equations in order to perform *rewriting modulo E*.

We know that there is considerable practical interest in understanding how real languages and systems may be integrated to exploit advantages of each individual system. To make real progress, we believe that practical advances need to be made

in synchrony with theoretical developments, with each approach supporting and informing the other. To this end, we are pursuing practical developments of the work presented in this paper. As a side note, we have also begun to look at the implementation of our examples within Maude [[5](#_bookmark27)].

We deduce *unconditional* equations but allow *conditional* axioms. This approach has merit from the point of view of practical specifications, and reasoning about them. That said, one could be rather more expressive if one allows conditional equa- tions as primary judgements of a deduction system. In such a case, the semantics of judgements could be given in an internal manner by making use of categories with equalisers [[19](#_bookmark41)]. We are currently working on such a system, with a view to giving a sound and complete semantics, and the results will appear in a future paper. There are interesting questions concerning the appropriate category theory, and the answers will have connections to work such as [[25](#_bookmark47)]. And further, since equational theories give rise to *free algebra monads* [[30](#_bookmark52)], further studies should investigate the possibility of extending/generalizing the results in this paper to the notion of *monad* [[23](#_bookmark45)]. Here, however, our intention has been to provide an account that is very general (categorical) but not so abstract that applications become obfuscated.

Finally, we wish to note that the approach presented in this paper generalises to the combination of an arbitrary number *n* of languages by recursively combin- ing the (associated theory of the) first *Th*1*,..., Thn—*1 theories with *Thn*. Such a modularity property strengthens the framework both from a theoretical and practi- cal perspective, enabling the construction of complex theories on the basis of more elementary ones.

# References

1. Amal Ahmed and Matthias Blume. An Equivalence-Preserving CPS Translation via Multi-Language Semantics. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, volume 46, pages 431–444, New York, NY, USA, 2011. ACM.
2. Garrett Birkhoff. On the Structure of Abstract Algebras. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 31, pages 433–454, Cambrdige, UK, October 1935. Cambridge University Press.
3. Samuele Buro, Roy Crole, and Isabella Mastroeni. Equational Logic and Categorical Semantics for Multi-Languages. MFPS36 paper extended with an appendix, June 2020. [www.cs.le.ac.uk/people/](http://www.cs.le.ac.uk/people/rlc3/research/papers/mfps2020-paper-elcsml.pdf)

[rlc3/research/papers/mfps2020-paper-elcsml.pdf](http://www.cs.le.ac.uk/people/rlc3/research/papers/mfps2020-paper-elcsml.pdf)

1. Samuele Buro and Isabella Mastroeni. On the Multi-Language Construction. In Lu´ıs Caires, editor,

*Programming Languages and Systems (ETAPS)*, pages 293–321, Berlin/Heidelberg, DE, 2019. Springer.

1. Manuel Clavel, Francisco Dur´an, Steven Eker, Patrick Lincoln, Narciso Mart´ı-Oliet, Jos´e Meseguer, and Carolyn Talcott. *All About Maude-a High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic*. Springer-Verlag, 2007.
2. R. L. Crole. *Categories for Types*. Cambridge Mathematical Textbooks. Cambridge University Press, 1993. xvii+335 pages, ISBN 0521450926HB, 0521457017PB.
3. Roy L. Crole. *α*-Equivalence Equalities. *Theoretical Computer Science*, 433:1–19, May 2012.
4. Trevor Evans. On Multiplicative Systems Defined by Generators and Relations: I. Normal Form Theorems. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 47, pages 637–649. Cambridge University Press, 1951.
5. Michael Furr and Jeffrey S. Foster. Checking TYpe Safety of Foreign Function Calls. *SIGPLAN Not.*, 40(6):62–72, June 2005.
6. Joseph Goguen and Jos´e Meseguer. Completeness of Many-Sorted Equational Logic. *Houston Journal* *of Mathematics*, 11(3):307–334, 1985.
7. Joseph A. Goguen, Jean-Pierre Jouannaud, and Jos´e Meseguer. Operational Semantics for Order-Sorted Algebra. In *International Colloquium on Automata, Languages, and Programming*, pages 221–231. Springer, 1985.
8. Joseph A Goguen and KAi Lin. Specifying, Programming and Verifying with Equational Logic. In *We Will Show Them!(2)*, pages 1–38, 2005.
9. Joseph A. Goguen and Jos´e Meseguer. Order-Sorted algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. *Theoretical Computer Science*, 105(2):217–273, 1992.
10. Kathryn E. Gray. Safe Cross-Language Inheritance. In Jan Vitek, editor, *ECOOP 2008 – Object-* *Oriented Programming*, pages 52–75, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
11. Gerard Huet and Derek C. Oppen. Equations and Rewrite Rules: A Survey. Technical report, Stanford, CA, USA, 1980.
12. Peter T Johnstone. *Sketches of an Elephant: a Topos theory Compendium*. Oxford logic guides. Oxford Univ. Press, 2002.
13. Claude Kirchner and H´el`ene Kirchner. Equational Logic and Rewriting. In *Computational Logic*, pages 255–282, 2014.
14. Donald E Knuth and Peter B Bendix. Simple Word Problems in Universal Algebras. In *Automation of Reasoning*, pages 342–376. Springer, 1983.
15. M. Makkai and G.E. Reyes. *First Order Categorical Logic*. Lecture Notes in Mathematics. Springer Verlag, 1977.
16. Narciso Mart´ı-Oliet and Jos´e Meseguer. Inclusions and Subtypes I: First-order Case. *Journal of Logic and Computation*, 6(3):409–438, 1996.
17. Jacob Matthews and Robert Bruce Findler. Operational Semantics for Multi-Language Programs.

*ACM Transactions on Programming Languages and Systems*, 31(3):12:1–12:44, 2009.

1. Jacob Matthews and Robert Bruce Findler. Operational Semantics for Multi-Language Programs.

*ACM Transactions on Programming Languages and Systems*, 31(3):1–44, 2009.

1. Eugenio Moggi. Notions of Computation and Monads. *Information and computation*, 93(1):55–92, 1991.
2. Peter-Michael Osera, Vilhelm Sj¨oberg, and Steve Zdancewic. Dependent Interoperability. In Koen Claessen and Nikhil Swamy, editors, *Proceedings of the Sixth Workshop on Programming Languages Meets Program Verification*, pages 3–14, New York, NY, USA, 2012. ACM.
3. E. Palmgren and S.J. Vickers. Partial Horn logic and Cartesian Categories. *Annals of Pure and Applied Logic*, 145(3):314 – 353, 2007.
4. Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed. Funtal: Reasonably Mixing a Functional Language with Assembly. In *Proceedings of the 38th ACM SIGPLAN Conference on* *Programming Language Design and Implementation*, pages 495–509, New York, NY, USA, 2017. ACM.
5. James T. Perconti and Amal Ahmed. Verifying an Open Compiler Using Multi-Language Semantics. In *Proceedings of the 23rd European Symposium on Programming Languages and Systems*, pages 128–148, Berlin, DE, 2014. Springer.
6. A. M. Pitts. Categorical Logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, chapter 2, pages 39–128. Oxford University Press, 2000.
7. A. M. Pitts. Nominal Techniques. *ACM SIGLOG News*, 3(1):57–72, January 2016.
8. Gordon Plotkin. Some Varieties of Equational Logic. In *Algebra, Meaning, and Computation*, pages 150–156. Springer, 2006.
9. Traian Florin S¸erb˘anu¸t˘a, Grigore Ro¸su, and Jos´e Meseguer. A Rewriting Logic Approach to Operational Semantics. *Information and Computation*, 207(2):305–340, 2009.
10. Gang Tan and Greg Morrisett. Ilea: Inter-language Analysis across Java and C. *SIGPLAN Not.*, 42(10):39–56, October 2007.
11. Alfred Tarski. Equational Logic and Equational Theories of Algebras. In *Studies in Logic and the Foundations of Mathematics*, volume 50, pages 275–288. Elsevier, Amsterdam, NL, 1968.
12. Walter Taylor. Equational Logic. *Houston Journal of Mathematics*, 47(2), 1979.
13. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.