University of Verona
Department of Computer Science

Doctoral Thesis

On Multi-Language Semantics

Semantic Models, Equational Logic, and Abstract Interpretation of Multi-Language Code

Samuele Buro

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy

Verona, 19th September 2021

On Multi-Language Semantics

On Multi-Language Semantics

Semantic Models, Equational Logic, and Abstract Interpretation of Multi-Language Code

---

Samuele Buro

University of Verona
Department of Computer Science
Strada le Grazie 15
37134 Verona, Italy

On Multi-Language Semantics: Semantic Models, Equational Logic, and Abstract Interpretation of Multi-Language Code, Samuele Buro, September 2021

Doctoral Advisor: Isabella Mastroeni (University of Verona)

Doctoral Thesis Referees: Roberto Bruni (University of Pisa), Peter Müller (ETH Zürich)

Doctoral Advisory Committee: Maria Paola Bonacina (University of Verona), Isabella Mastroeni (University of Verona), Massimo Merro (University of Verona)

I, Samuele Buro, declare that this thesis titled, "On Multi-Language Semantics: Semantic Models, Equational Logic, and Abstract Interpretation of Multi-Language Code" and the work presented in it are my own. I confirm that: • This work was done wholly or mainly while in candidature for a research degree at this University. • Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated. • Where I have consulted the published work of others, this is always clearly attributed. • Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work. • I have acknowledged all main sources of help. • Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Verona, 19th September 2021

To my late father, to whom I promised to complete this thesis before he left this world.

To my mother and sisters.

To Margherita.

Acknowledgments

This thesis is the result of three years of research activity carried out during my PhD at the University of Verona, with a period of about three months spent as a visiting student at the University of Leicester.

I would like to express my sincere gratitude to my advisor Prof. Isabella Mastroeni for encouraging me to begin the doctoral studies, for the continuous support of my PhD research, and for her guidance throughout this project. I also wish to thank her for providing me the opportunity to work on such an intriguing research topic.

I am deeply indebt to Prof. Roy L. Crole for introducing me in the field of categorical logic. I will always remember the long time we spent in his office by sketching diagrams and semantics on the whiteboard, the fruitful discussions that we had on various mathematical topics, and his impressive knowledge in theoretical aspects of computer science. Part of the work presented in this thesis would not have been possible without his expertise on the subject.

Finally, I would like to thank the reviewers, Roberto Bruni and Peter Müller, for their careful reading and thoughtful comments that helped to improve the thesis. They both provided valuable research ideas for ongoing and future works, along with interesting related works that have been included in the current bibliography.

Abstract

Modern software development rarely takes place within a single programming language. Often, programmers appeal to *cross-language interoperability*. Benefits are two-fold: exploitation of novel features of one language within another, and cross-language code reuse. For instance, 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).

Whilst this approach enables developers to benefit from the strengths of each base language, it comes at the price of a lack of clarity of formal properties of the new multi-language, mainly semantic specifications. Developing such properties is a key focus of this thesis. Indeed, while there has been some research exploring the interoperability mechanisms, there is little development of theoretical foundations.

In this thesis, we broaden the *boundary functions*-based approach à la Matthews and Findler to propose an algebraic framework that provides systematic and more general ways to define *multi-languages*, regardless of the inherent nature of the underlying languages. The aim of this strand of research is to overcome the lack of a formal model in which to design the combination of languages. Main contributions are an *initial algebra semantics* and a *categorical semantics* for multi-languages.

We then give ways in which interoperability can be 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 part of the thesis is devoted to exploring the properties of these theories. This is accomplished by way of both *universal algebra* and *category theory*, giving us a very general and flexible semantics, and hence a wide collection of models. *Classifying categories* are constructed, and hence equational theories furnish each categorical model with an internal language. From this we establish *soundness* and *completeness* of the multi-language equational logic.

As regards static analysis, the heterogeneity of the multi-language context opens up new and unexplored scenarios. In this thesis, we provide a general theory for the *combination of abstract interpretations* of existing languages in order to gain an abstract semantics of *multi-language programs*. As a part of this general theory, we show that formal properties of interest of multi-language abstractions (e.g., *soundness* and *completeness*) boil down to the features of the interoperability mechanism that binds the underlying languages together. We extend many of the standard concepts of abstract interpretation to the framework of multi-languages.

Finally, a minor contribution of the thesis concerns language specification formalisms. We prove that longstanding syntactical transformations between context-free grammars and algebraic signatures give rise to adjoint equivalences that preserve the abstract syntax of the generated terms. Thus, we have methods to move from context-free languages to the algebraic signature formalisms employed in the thesis.

Publications

[BCM20a] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. Equational logic and categorical semantics for multi-languages. In Patricia Johann, editor, *Proceedings of the 36th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2020, Online, October 1, 2020*, volume 352 of *Electronic Notes in Theoretical Computer Science*, pages 79–103. Elsevier, 2020.

[BCM20b] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. Equational logic and set-theoretic models for multi-languages. In Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno, editors, *Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14-16, 2020*, volume 2756 of *CEUR Workshop Proceedings*, pages 236–249. CEUR-WS.org, 2020.

[BCM20c] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. On multi-language abstraction — Towards a static analysis of multi-language programs. In David Pichardie and Mihaela Sighireanu, editors, *Static Analysis - 27th International Symposium, SAS 2020, Virtual Event, November 18-20, 2020, Proceedings*, volume 12389 of *Lecture Notes in Computer Science*, pages 310–332. Springer, 2020.

[BM19a] Samuele Buro and Isabella Mastroeni. On the multi-language construction. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 293–321. Springer, 2019.

[BM19b] Samuele Buro and Isabella Mastroeni. On the semantic equivalence of language syntax formalisms. In Alessandra Cherubini, Nicoletta Sabadini, and Simone Tini, editors, *Proceedings of the 20th Italian Conference on Theoretical Computer Science, ICTCS 2019, Como, Italy, September 9-11, 2019*, volume 2504 of *CEUR Workshop Proceedings*, pages 34–51. CEUR-WS.org, 2019.

[BM20] Samuele Buro and Isabella Mastroeni. On the semantic equivalence of language syntax formalisms. *Theoretical Computer Science*, 840:234–248, 2020.

Contents

CHAPTER 1

Introduction

Interoperability between languages has been a problem since the second programming language was invented.

— David Chisnall

There is currently a myriad of programming languages, many of which have extensive library support. With programs becoming larger and increasingly complex, *interoperability mechanisms* streamline program development by enabling the interplay between pieces of code written in different languages. Examples are *embedded interpreters* [Ram06], consisting of a runtime engine implemented in the host language (such as Jython [JBW+10] that lets Java interoperate with Python), or the *foreign function interface* system that allows one language to call routines written in another (e.g., the Java Native Interface [Lia99] enables Java code to call C++ functions). On one hand these mechanisms are essential tools, but on the other hand they hamper our understanding of the resulting programs. Indeed, multi-language programs do not obey any of the semantics of the base languages but a (non-trivial) combination of them. As a consequence, any method of formal reasoning (such as static program analysis or verification) is neutralised by the lack of a semantic specification.

1.1 An Informal Summary

Language interoperability is the ability of two or more programming languages to interact as a unique, integrated system. A multi-language is a programming language arising from a combination of two or more programming languages. But how can we formally combine programming languages? What is the meaning (semantics) of multi-language programs? We illustrate the problem in simple terms using some analogies with spoken language. Consider the following scenario based on natural languages rather than programming languages. English and Spanish are human languages usually spoken by people with a different cultural heritage. Geographical areas in which these cultures have met and mixed, have seen the birth of a new dialect referred to as *Spanglish*, in which new lexical and grammatical rules result from the blending of these languages. For instance, in the sentence "Keep tranquilo and

habla Spanglish", English words "calm" and "speak" have been substituted by their Spanish counterparts. The semantics of the whole sentence is given by translating the meaning of the mixed foreign words in a compositional way. When we look at programming languages, the scenario is similar: Multi-language programs are obtained by mixing code fragments. However, if translating words between two natural languages is relatively easy, converting the meaning of code fragments in one programming language to another is not, mainly due to the substantial differences between the features that each language provides (e.g., a high-level language that interoperates with a low-level one). Now, although arising from a combination of languages, the real gain made by Spanglish is that we can use it as a single language in itself. But it is much more difficult to construct a single useful programming language from others. Most existing techniques allow two languages to communicate, to exchange data, but without providing a clear view of a single new, fully-fledged programming language (cf. Spanglish). This clear single view would be much more useful to programmers, who would then have a complete understanding of this single combined language.

We would like to create a system in which starting from two (or more) programming languages specified according to some given design principles, we obtain a single combination of these languages with its own syntax and semantics and satisfying the same design principles: a multi-language. Why could this be useful? Here is one example. Static analysers are tools that provide analyses of program behaviour at compile time. They usually rely on mathematical frameworks such as abstract interpretation or data-flow equations. But it is very difficult to apply this technique to situations where two separate languages only interoperate, that is, communicate and exchange data. What would be far more useful and robust is to work with a (single) multi-language. The same goes for static or dynamic analysers, interpreters, compilers and so on, and these are all fundamental tools used everyday by computer scientists and engineers. We believe there will be considerable interest in multi-languages since practitioners find it useful to be able to blend key features from different languages. However, as noted, building multi-languages is not easy. There has been some practical work in recent years, but there is no theory of multi-languages.

1.2 The Lack of a Multi-Language Framework

The notion of *multi-language* is employed naively in several works in literature [PA14, AB11, TM07, FF08, Gra08, PPDA17, OSZ12, MF09] to indicate a combination 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 able to regulate both control flow and value conversion between the underlying languages [MF09], thus providing *cross-language interoperability* [Chi13]. We show below some of the most common ones.

Embedded Interpreters A very popular method to execute external code within a programming language is the use of *embedded interpreters* [Ben05, Ram11, Ram06]. Roughly, an embedded interpreter is an interpreter for the *external language* written in the *host language*. For instance, Nashorn [Orab] is a JavaScript interpreter written in Java released with Java 8, Jython [JBW+10] is a Java implementation of Python, and the Lua programming language [IdFF96] has been conceived to be an embedded language rather than a stand-alone one. Below is reported a simple multi-language

`Hello World` based on Nashorn, in order to give a taste of this cross-language interoperability mechanism:

```
ScriptEngine e = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval("print('Hello World!');");
```

In the first line the Nashorn interpreter is retrieved. Then, JavaScript code is passed as a string and executed by the interpreter in the second line. ♢

Foreign Function Interfaces One of the first mechanism developed to achieve cross-language interoperation are the *foreign function interfaces* [FF08]. They provide methods that allows one language to call functions written in another language. For instance, the Java Native Interface (JNI) is a programming interface enabling Java to call native methods (such as, written in C/C++ and assembly). The following code snippet is an example:

```
private native void helloWorld(); // Java

JNIEXPORT void JNICALL Java_HelloJNI_helloWorld(JNIEnv *env, jobject thisObj) {
  printf("Hello World!\n");
  return;
} // C or C++
```

The first part of the code is the (Java) declaration of a native method which is then implemented in C or C++ in the second part. The JNI acts as a bridge between the function calls in Java and the function implementation in C/C++. ♢

Boundary Functions Theoretical works follow the more formal approach of *boundary functions* (or, for short, *boundaries*) in the style of Matthews and Findler's multi-language semantics [MF09]. Simply put, boundaries are syntactical constructs that act as a gate between single-languages, and when a value needs to flow to the other language, they perform a conversion so that it complies to the other language semantic specifications. In [MF09], the authors use boundary functions to design a multi-language obtained by mixing ML and Scheme. ♢

Regardless of the chosen mechanism, the full construction is usually carried out manually by language designers, which define the multi-language by reusing the formal specifications of the base languages [PPDA17, PA14, AB11, MF09]. 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-languages which does not provide neither a method for language designers to conceive new multi-languages nor any guarantee on the correctness of the resulting constructions.

1.3 The Problem of Static Analysis on Multi-Language Code

Despite the wide range of frameworks for interoperability, there is a lack of techniques for combining *static analyses* of different languages. Static analysis consists of a range of well-established and widely used techniques for automatically extracting dynamic (i.e., runtime) behaviours statically (i.e., without executing the code). When it comes to multi-languages, two new challenges need to be tackled. Firstly, single-language analysers are not conceived for inspecting external code, and secondly the

combination of analyses is not straightforward, since the interoperability mechanism that blends the underlying languages adds a new semantic layer. For instance, consider the following Java code snippet analysed with SonarQube Scanner [CP13]:[1]

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

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

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

The method `eval` evaluates the JavaScript code `null` via the Nashorn engine (see previous section) and returns the equivalent Java value `null`. This trivial example underlines how easy it is to deceive an analysis when writing multi-language programs.

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

1.4 Structure and Contributions

We begin by introducing background notions. In Chapter 2, we provide preliminary definitions of the most recurring mathematical concepts on which the thesis is based. In particular, we introduce the reader to basic category theory, abstract interpretation, and fixpoint theory. Then, in Chapter 3, we present the framework of order-sorted algebra in detail along with order-sorted equational logic. Although Chapter 2 may be skipped at a first reading, we recommend the reader to have a closer look to Chapter 3, since order-sorted algebras will be pervasive throughout the thesis.

Contributions start from Chapter 4 and following. In Chapter 4 we define three different multi-language constructions, each of which refines the previous one. Rather than working with two fixed languages, we shall base the multi-language framework on order-sorted algebra specifications, in a way that we abstract the process of combining languages from the concrete definition of languages themselves. Main theorems of this chapter concern the *initiality* of the multi-language term model, which ensure a unique *semantic function* (that is, a *homomorphism*) providing multi-language terms with a meaning. Then, in Chapter 5, we lift order-sorted equational logic to the just defined multi-language world. We will prove that the resulting deduction system is *sound* and *complete* with respect to the algebraic semantics defined in the previous chapter.

In Chapter 6 we replay the results of Chapter 4 and 5, but in a categorical setting. 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. The main contribution is a deductive system for multi-languages

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

with a *sound* and *complete* categorical semantics. We also show that combining languages is a closed operation.

We start to investigate *abstract semantics* of multi-language programs in Chapter 7. The main contribution is a general technique for abstracting multi-language semantics given the interoperation of the underlying languages and of their abstract semantics. We study standard notion of abstract interpretation in the context of the algebraic framework of order-sorted algebras, and we then define the notion of *multi-language collecting semantics*.

Finally, in Chapter 8, we provide the categorical description of several syntax transformation methods across different formalisms. In particular, we prove that some longstanding syntactical transformations between context-free grammars and many-sorted signatures and between many-sorted signatures and order-sorted signatures give rise to adjoint functors and/or adjoint equivalences that preserve the abstract syntax of the generated terms. The conclusion is twofold: Every categorical property and construction can be shifted between these frameworks, and all these formalisms are essentially the same from a semantic perspective.

CHAPTER 2

Basic Mathematical Notions

Chapter reference(s): [ML13, BW95, Pie91]

In this chapter, we introduce the notation and the basics of the mathematics used throughout the thesis. The reader may want to skim through it and return to it when needed. In particular, we provide basic definitions and theorems on *fixpoints*, *abstract interpretation*, and *category theory*.

Structure We begin by providing some preliminary notions. Then, we introduce the basics of order theory and fixpoints theorems. We continue with some elements of abstract interpretation, and finally we present basic notions of category accompanied by simple examples.

2.1 Preliminary Notions

Definition 2.1 (Indexed Family of Sets)**.** Let I and A be sets. An *indexed family of sets* is a function $f\colon I \to \wp(A)$. We call A the *underlying set* and I the *index set*. The elements $i \in I$ are referred to as *indices*, and we define $A_i = f(i) \subseteq A$ the *component* of A indexed by i. We abuse notation and always write $(A_i \mid i \in I)$ to denote the indexed family of sets $f\colon I \to \wp(A)$, tacitly assuming that the underlying set is exactly the union of all components, that is $A = \bigcup_{i\in I} A_i$. ♢

Indexed family of sets

We usually define an indexed family simply by delivering its components, leaving both I and A implicit. For instance, we might write $A_0 = \{0, 2, 4, \ldots\}$ and $A_1 = \{1, 3, 5, \ldots\}$ for the indexed family $(A_i \mid i \in I)$ with $I = \{0, 1\}$ and $A = \mathbb{N}$. From now on, we refer to an indexed family of sets $(A_i \mid i \in I)$ as A, even though A already denotes the underlying set of such a family. In practice, this will cause no confusion: The underlying set of an indexed family will always be kept implicit. In the rare cases (if any) where we need to refer to both the indexed family *and* its underlying set, the latter will be denoted by $\bigcup A$.

Definition 2.2 (Kleene Closure)**.** Let A be a set (typically referred to as an *alphabet*). The *Kleene closure* A^* of A is defined as

Kleene closure

$$A^* = \bigcup_{n\in\mathbb{N}} A^n \quad \text{where} \quad \begin{cases} A^0 = \{\varepsilon\} \\ A^{n+1} = \{\, wa \mid w \in A^n \text{ and } a \in A \,\} \end{cases}$$

where the *empty string* ε is always assumed not to be part of the alphabet, that is $\varepsilon \notin A$. Moreover, we denote by $A^+ = A^* \setminus \{\varepsilon\}$ the set of non-empty strings. In the following, we will use juxtaposition of strings for their concatenation, with ε the unit element. ♢

2.2 Orderings and Fixpoints

Poset

A *partially ordered set (poset)* $(A, \leqslant)$ is a set A equipped with a binary relation $\leqslant \subseteq A \times A$ such that $\leqslant$ is *reflexive*, that is $x \leqslant x$ for each $x \in A$, *antisymmetric*, that is $x \leqslant y$ and $y \leqslant x$ imply $x = y$ for each pair of elements $x, y \in A$, and *transitive*, that is $x \leqslant y$ and $y \leqslant z$ imply $x \leqslant z$ for each $x, y, z \in A$. We shall write $x < y$ meaning that $x \leqslant y$ and $x \neq y$, and $x \geqslant y$ for $y \leqslant x$. A *total order* is a poset $(A, \leqslant)$ with all the elements comparable, that is $x \leqslant y$ or $y \leqslant x$ for each $x, y \in A$.

Total order

Maximal, Minimal

Maximum, Minimum

We say that an element $x \in A$ is *maximal (minimal)* if there is no element $y \in A$ different from x such that $y \geqslant x$ ($y \leqslant x$). Moreover, $x \in A$ is the *maximum (minimum)* element if for each $y \in A$ holds that $x \geqslant y$. We will usually denote maximum and minimum elements of a poset by $\top$ and $\bot$, respectively. Given a poset $(A, \leqslant)$ and $B \subseteq A$, we call $x \in A$ an *upper bound* of B if $y \leqslant x$ for each $y \in B$. Let U be the set of upper bounds of B. The *least upper bound (lub)* of B is the minimum element of U, if it exists. Vice versa, a *lower bound* of B is an element $x \in A$ such that $x \leqslant y$ for each $y \in B$. Let L be the set of lower bounds of B. The *greatest lower bound (glb)* of B is the maximum element of L, if it exists.

Upper bound

Least upper bound

Lower bound

Greatest lower bound

Notation 2.1. We usually denote by $\bigvee B$ and $\bigwedge B$ the lub and the glb of a subset B of a poset $(A, \leqslant)$. In general, we try to be consistent between the lub/glb notation and the partial ordering. For instance, if the partial ordering is denoted by $\sqsubseteq$, we will use $\sqcup$ and $\sqcap$ to denote the lub and the glb operator, if the ordering is denoted by $\preccurlyeq$, we will use $\curlyvee$ and $\curlywedge$, and so on. ♢

Increasing chain

Directed set

An *increasing chain* B in a poset $(A, \leqslant)$ is subset of A with a minimum element and total with respect to the ordering $\leqslant$ on A. A subset B of a poset $(A, \leqslant)$ is *directed* if $B \neq \varnothing$ and for each $x, y \in B$ there is an element $z \in A$ such that $z \geqslant x, y$ (that is, the set $\{x, y\}$ has at least an upper bound in A). Note that an increasing chain is obviously directed.

Dcpo, ω-cpo, $\bot$-cpo

A *directed complete partial order (dcpo)* is a poset such that each of its directed subsets has lub. A *ω-complete partial order (ω-cpo)* is a poset such that each of its increasing chains has lub. A *$\bot$-cpo* (or, pointed cpo) is an ω-cpo with a minimum. A *lattice* is a poset in which every two elements has both lub and glb. A *complete lattice* is a poset in which every subset has both lub and glb.

Lattice

Complete lattice

Monotone function

Continuous function

Let $(A, \leqslant)$ be a poset. A function $f\colon A \to A$ is *monotone* if for each $x, y \in A$ such that $x \leqslant y$ implies $f(x) \leqslant f(y)$; and *continuous* if it preserves all directed lub, that is, for every directed subset $B \subseteq A$ with lub in A, holds that $\bigvee f(B) = f(\bigvee B)$, where $f(B) = \{\, f(x) \mid x \in B \,\}$.

Fixpoint

Least fixpoint

Let $f\colon A \to A$ be a function on a poset $(A, \leqslant)$. A *fixpoint* of f is an element $x \in A$ such that $f(x) = x$. The *least fixpoint* $\operatorname{lfp} f$ of f is the smallest fixpoint of f. Note that if $\operatorname{lfp} f$ exists, then it is unique.

Theorem 2.1 (Knaster-Tarski Theorem). *Let* $(A, \leqslant)$ *be a complete lattice and* $f\colon A \to A$ *a monotone function on* A*. Then, the set of fixpoints of* f *is a complete lattice.*

Corollary 2.1. *Let* $(A, \leqslant)$ *be a complete lattice and* $f\colon A \to A$ *a monotone function on* A*. Then,* $\operatorname{lfp} f = \bigwedge\{ x \in A \mid f(x) \leqslant x \}$.

Knaster-Tarski theorem ensures the existence of a least fixpoint of a monotone function over a poset. However, it does not provide a constructive method for computing such a fixpoint. The following theorem gives an iterative approach to calculate the least fixpoint of a continuous function on a dcpo with a bottom element.

Theorem 2.2 (Kleene's Theorem). *Let* $(A, \leqslant)$ *be a dcpo with a bottom element* $\bot$ *and* $f\colon A \to A$ *a continuous function on* A*. Then,*

$$\operatorname{lfp} f = \bigvee_{n \in \mathbb{N}} f^n(\bot)$$

where $f^{n+1} = \overbrace{f \circ \cdots \circ f}^{n+1}$ *and* f^0 *is the identity on* A.

We shall sometimes write $\operatorname{lfp}_{\bot}^{\leqslant} f$ instead of $\operatorname{lfp} f$ to highlight the ordering and the minimum element we are working with.

2.3 Elements of Abstract Interpretation

We introduce the basics of abstract interpretation, namely *concretisation and abstraction functions*, *Galois connections*, *soundness* of the abstraction, and *best abstraction*. We mainly follow [Min17, RY20].

Definition 2.3 (Concretisation Function). Let $(C, \leqslant)$ and $(A, \sqsubseteq)$ be two posets. A *concretisation function* $\gamma\colon (A, \sqsubseteq) \to (C, \leqslant)$ is a monotone function that maps each abstract element $a \in A$ to a concrete object $\gamma(a) \in C$. ♢

Concretisation Function

Example 2.1. Let $(\wp(\mathbb{N}), \subseteq)$ and $(\wp(\mathbb{B}), \subseteq)$ be the posets encoding concrete and abstract worlds, where $\mathbb{B} = \{0, 1\}$ is a boolean representation of the parity property of natural numbers. Then, $\gamma\colon (\wp(\mathbb{B}), \subseteq) \to (\wp(\mathbb{N}), \subseteq)$ is defined by

$$\begin{aligned} \varnothing &\mapsto \varnothing \\ \{0\} &\mapsto \{ n \in \mathbb{N} \mid n \text{ is even} \} \\ \{1\} &\mapsto \{ n \in \mathbb{N} \mid n \text{ is odd} \} \\ \mathbb{B} &\mapsto \mathbb{N} \end{aligned}$$

♢

We say that an abstract element $a \in A$ is a *sound abstraction* of a concrete element $c \in C$ if and only if $c \leqslant \gamma(a)$.

Soundness

Galois connections provide a stronger relation between the concrete and abstract worlds by means of both a concretisation and an abstraction function.

Definition 2.4 (Galois Connection). Let $(C, \leqslant)$ and $(A, \sqsubseteq)$ be two posets. A *Galois connection* is given by a pair of (monotone) functions $\gamma\colon (A, \sqsubseteq) \to (C, \leqslant)$ and $\alpha\colon (C, \leqslant) \to (A, \sqsubseteq)$ such that for each $a \in A$ and $c \in C$

$$c \leqslant \gamma(a) \iff \alpha(c) \sqsubseteq a$$

and it is denoted by $(C, \leqslant) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$. ♢

Galois connection

Example 2.2. Let $2\mathbb{N}$ and $2\mathbb{N}+1$ be the sets of even and odd natural numbers, respectively. Consider the previous example and let $\alpha\colon (\wp(\mathbb{N}),\subseteq) \to (\wp(\mathbb{B}),\subseteq)$ be defined as

$$\alpha(X) = \begin{cases} \varnothing & \text{if } X = \varnothing \\ \{0\} & \text{if } X \subseteq 2\mathbb{N} \\ \{1\} & \text{if } X \subseteq 2\mathbb{N}+1 \\ \mathbb{B} & \text{otherwise} \end{cases}$$

Then, $(\wp(\mathbb{N}),\subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (\wp(\mathbb{B}),\subseteq)$ form a Galois connection. The proof is by cases on the element in $\wp(\mathbb{B})$. For instance, if we pick the element $\{0\}$, then $X \subseteq \gamma(\{0\})$ implies that $X \subseteq 2\mathbb{N}$, and therefore either $X = \varnothing$ and $\alpha(X) = \varnothing \subseteq \{0\}$ or X is a non-emtpy set of even numbers and $\alpha(X) = \{0\} \subseteq \{0\}$. The proof in the opposite direction is similar, as well as the other cases. ♢

Galois connections may be alternative characterised by the following proposition:

Proposition 2.1. $(C,\leqslant) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A,\sqsubseteq)$ *is a Galois connection if and only if*

1. α *and* γ *are monotone functions;*
2. $\gamma \circ \alpha$ *is* extensive*, that is* $c \leqslant \gamma(\alpha(c))$ *for each concrete element* $c \in C$*; and*
3. $\alpha \circ \gamma$ *is* reductive*, that is* $\alpha(\gamma(a)) \sqsubseteq a$ *for each abstract element* $a \in A$*.*

The next theorem establishes some useful properties of Galois connections that we will extensively use in proofs involving (α,γ):

Theorem 2.3 (Properties of Galois Connections)**.** *Let* $(C,\leqslant) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A,\sqsubseteq)$ *be a Galois connection. Then,*

1. $\gamma \circ \alpha \circ \gamma = \gamma$ *and* $\alpha \circ \gamma \circ \alpha = \alpha$*;*
2. $\alpha \circ \gamma$ *and* $\gamma \circ \alpha$ *are* idempotent*;*
3. α *maps concrete elements to their most precise sound approximation, that is*
$$\alpha(c) = \sqcap\{\, a \in A \mid c \leqslant \gamma(a) \,\}$$
for each $c \in C$*;*
4. *vice versa,* γ *maps abstract properties to their best concretisation in* C*, that is*
$$\gamma(a) = \vee\{\, c \in C \mid \alpha(c) \sqsubseteq a \,\}$$
for each $a \in A$*;*
5. α *preserves least upper bounds, that is*
$$\alpha(\vee X) = \sqcup\{\, \alpha(x) \in A \mid x \in X \,\}$$
for each $X \subseteq C$ *and whenever* $\vee X$ *exists; and*
6. γ *preserves greatest lower bounds, that is*
$$\gamma(\sqcap Y) = \wedge\{\gamma(y) \in C \mid y \in Y\}$$
for each $Y \subseteq A$ *and whenever* $\sqcap Y$ *exists.*

Best abstraction

By the third property, we say that $\alpha(c)$ is the *best abstraction* of the concrete element $c \in C$, that is the smallest abstract element soundly approximating c.

2.4 Basic Category Theory

We introduce the basics of category theory. The reader may want to consult a book for a more thorough presentation of these concepts (e.g., [BW95]). The exposition of the material in this chapter follows the presentation in [Pie91].

Definition 2.5 (Category)**.** A *category* C is given by

Category

- a collection of *objects*, usually denoted with letters $X, Y, Z, \ldots$;

Object

- a collection of *morphisms* (or, *arrows*) between the objects, denoted by $f\colon X \to Y$ or $X \xrightarrow{f} Y$, where X and Y are referred to as the *source* and *target* of the morphism f, respectively. We will denote the collection of morphisms between two objects X and Y in the category C by $C(X, Y)$;

Morphism, Arrow

Source, Target

- a way of composing consecutive morphisms, that is an operator

$$\circ\colon C(Y, Z) \times C(X, Y) \to C(X, Z)$$

assigning to each pair of morphisms $Y \xrightarrow{g} Z$ and $X \xrightarrow{f} Y$ a composite morphism $X \xrightarrow{g \circ f} Z$, for each triple of objects X, Y, and Z (we will always use infix notation for $\circ$). The composition operator of the category C must satisfy the following *associative law*: Given any morphisms $W \xrightarrow{f} X$, $X \xrightarrow{g} Y$, and $Y \xrightarrow{h} Z$, then

$$h \circ (g \circ f) = (h \circ g) \circ f$$

- an *identity morphism* $X \xrightarrow{id_X} X$ for each object X satisfying the following *identity laws*: Given any morphism $X \xrightarrow{f} Y$, then

Identity

$$id_Y \circ f = f \qquad \text{and} \qquad f \circ id_X = f \qquad \diamondsuit$$

Example 2.3. We give some examples of both categories of structures and structures as categories:

1. The category of sets *Set* has sets as objects and functions as morphisms. The composition of morphisms is given by ordinary set-theoretic composition (which is associative), and for each set X, the identity morphism id_X is the identity function on X.

2. A single set itself can be regarded as a category, where objects are the elements of the sets and without any arrows except identities.

3. The category of partial orders *Poset* is given by taking posets as objects and monotone functions as morphisms. The composition of morphisms is set-theoretic composition of functions and identity morphisms are identity functions (trivially monotone). Note that this is well-defined, since the composition of monotone functions is monotone.

4. A single poset $(P, \leqslant)$ can be regarded as a category:

   - objects are elements $x \in P$;

- there is a morphism (x, y) between two objects x and y just in case that $x \leqslant y$;
- the identity on x is (x, x); and
- $(y, z) \circ (x, y) = (x, z)$.

One can check that the categorical laws are satisfied. ♢

We shall shortly define different categorical notions (such as *monomorphisms*, *initial objects*, *product*, and many others). The following definition provides a simple tool to obtain a dual notion of these concepts.

Dual category

Definition 2.6 (Dual Category)**.** Let C be a category. The *dual category* C^{op} is defined as follows:

- the objects of C^{op} are the objects of C;
- $X \xrightarrow{f^{op}} Y$ is a morphism in C^{op} if and only if $Y \xrightarrow{f} X$ is a morphism in C;
- the identity morphism on an object X in C^{op} is $id_X = id_X^{op}$; and
- the composition of two morphisms $X \xrightarrow{f^{op}} Y$ and $Y \xrightarrow{g^{op}} Z$ in C^{op} is defined as
$$g^{op} \circ_{C^{op}} f^{op} = (f \circ_C g)^{op}$$
where $\circ_{C^{op}}$ and $\circ_C$ are the composition operators in C^{op} and C, respectively. ♢

For instance, as we shall see in Sections 2.4.1 and 2.4.2, *epimorphisms* can be defined as *monomorphisms* in the dual category, *terminal objects* of a category C are initial objects in C^{op}. In general, the dual of a categorical definition can be thought as the same definition except that morphisms are "turned around". Category theorists often use the prefix *co* to refer to such dual constructions (e.g., product and coproduct).

Before introducing some basic categorical concepts, we define the notion of *subcategory* and *diagram*.

Subcategory

Definition 2.7 (Subcategory)**.** Let C be a category. D is a *subcategory* of C if

- each object in D is also an object in C;
- each morphism in D is also a morphism in C, that is $D(X, Y) \subseteq C(X, Y)$ for each pair of objects X and Y in D; and
- identities and composition in D match those of C. ♢

Full subcategory

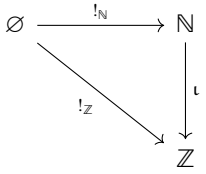
Luff subcategory

Example 2.4. The category *FSet* of finite sets is a subcategory of *Set*. In particular, we say that it is a *full subcategory*, since $FSet(X, Y) = Set(X, Y)$ for each pair of finite sets X and Y. On the other hand, if we define the category *Pfn* of sets where morphisms are partial functions, then *Set* is a subcategory of *Pfn*. Since the objects of these two categories are the same, we say that *Set* is a *luff subcategory* of *Pfn*. ♢

Diagram

Informally, a *diagram* in a category C is a collection of objects connected by some

morphisms of C. For instance, the following



is a diagram in *Set*, where $!_A$ denotes the empty function from the empty set to any set A, and ι is the inclusion map. We say that a diagram *commutes* if for every pair of vertices X and Y, the composition of the morphisms along every path from X to Y is the same. For instance, the previous diagram commutes since $\iota \circ !_{\mathbb{N}} = !_{\mathbb{Z}}$. We will use dashed morphism in diagram to assert that such a morphism exists whenever the rest of the diagram has been correctly filled (e.g., see Definition 2.13). We will provide a more formal definition of diagram after introducing *functors*.

2.4.1 Some Classes of Morphisms

We describe some classes of morphisms that are commonly found in the various categories.

Definition 2.8 (Monomorphism)**.** A morphism $X \xrightarrow{m} Y$ in a category C is a *monomorphism* (or, *monic*, *mono*) if for any pair of morphisms $g, h \colon W \to X$ we have that

Monomorphism

$$m \circ g = m \circ h \implies g = h$$ ♢

Example 2.5. Monomorphisms in *Set* are injective functions, and vice versa. ♢

Definition 2.9 (Epimorphism)**.** A morphism $X \xrightarrow{e} Y$ in a category C is an *epimorphism* (or, *epic*, *epi*) if for any pair of morphisms $g, h \colon Y \to Z$ we have that

Epimorphism

$$g \circ e = h \circ e \implies g = h$$ ♢

Example 2.6. Epimorphisms in *Set* are surjective functions, and vice versa. ♢

Definition 2.10 (Isomorphims)**.** A morphism $X \xrightarrow{f} Y$ in a category C is an *isomorphism* if there is an *inverse* morphism $Y \xrightarrow{g} X$ such that

Isomorphims
Inverse

$$g \circ f = \mathrm{id}_X \qquad \text{and} \qquad f \circ g = \mathrm{id}_Y$$

Since the inverse of a morphism is unique, we will denote g as f^{-1}. ♢

Two objects are said to be *isomorphic* provided that there is an isomorphism between them.

2.4.2 Universal Constructions

Universal constructions describe objects and arrows that satisfy a certain property. Here we mention some of the most common universal constructions that are found later on in the thesis.

Initial object

Definition 2.11 (Initial Object)**.** An object 0 in a category C is *initial* if for every object X of C there is a unique morphism, usually denoted by $!_X$, from 0 to X, that is $0 \xrightarrow{!_X} X$. ♢

The dual definition of an initial object is a *terminal object*.

Terminal object

Definition 2.12 (Terminal Object)**.** An object 1 in a category C is *terminal* if for every object X of C there is a unique morphism, usually denoted by $!_X$, from X to 1, that is $X \xrightarrow{!_X} 1$. ♢

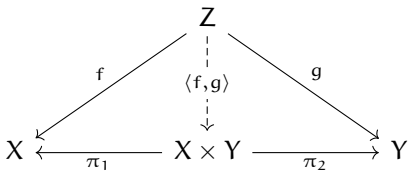
Note that we use the same notation both for terminal and initial object. The context, and in particular the source and the target of $!_X$, will make the notation unambiguous.

Example 2.7. The initial object in *Set* is the empty set, and the terminal object is the singleton set. Note that the initial object is unique, whereas there are infinite singleton sets. However, it is easy to prove that both initial and terminal objects (and, in general, every object determined by a universal construction) are *unique up to a unique isomorphism.* ♢

Example 2.8. The initial object in the category given by a poset $(P, \leqslant)$ is the minimum element; vice versa, the terminal object is the maximum. Of course, in general, they may not exist. ♢

Product
Projection
Mediating morphism

Definition 2.13 (Product)**.** Let C be a category. The *product* of two objects X and Y is an object $X \times Y$ along with two morphisms (called *projections*) $X \times Y \xrightarrow{\pi_1} X$ and $X \times Y \xrightarrow{\pi_2} Y$ such that for any object Z and morphisms $Z \xrightarrow{f} X$ and $Z \xrightarrow{g} Y$ there is a unique morphism (called *mediating morphism*) $Z \xrightarrow{\langle f,g\rangle} X \times Y$ making the following diagram commute:

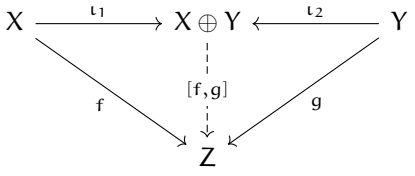


♢

Example 2.9. The (categorical) product of two objects X and Y in *Set* is the ordinary Cartesian product. In the category arising from a poset, the product of two elements is their greatest lower bound. ♢

Coproduct
Injections

Definition 2.14 (Coproduct)**.** Let C be a category. The *coproduct* of two objects X and Y is an object $X \oplus Y$ along with two morphisms (called *injections*) $X \xrightarrow{\iota_1} X \oplus Y$ and $Y \xrightarrow{\iota_2} X \oplus Y$ such that for any object Z and morphisms $X \xrightarrow{f} Z$ and $Y \xrightarrow{g} Z$ there is a unique morphism $X \oplus Y \xrightarrow{[f,g]} Z$ making the following diagram commute:



♢

Example 2.10. The coproduct in *Set* coincides with the disjoint union. In the category arising from a poset, the coproduct of two elements is their least upper bound. ♢

2.4.3 Functors, Natural Transformations, and Adjoints

A *functor* is a mapping between categories such that preserves the inner structure of the source category into the target category.

Functor

Definition 2.15 (Functor)**.** Let C and D be two categories. A *functor* $F\colon C \to D$ is a map taking each object X in C to an object FX in D and each morphism $X \xrightarrow{f} Y$ in C to a morphism $FX \xrightarrow{Ff} FY$ in D, such that identities and composition are preserved, that is

- $F\,\mathrm{id}_X = \mathrm{id}_{FX}$ for each object X in C; and
- $F(g \circ f) = Fg \circ Ff$ for each pair of composable morphisms in C. ♢

Example 2.11. The powerset operator which takes a set A to its powerset $\wp(A)$ and each function $f\colon A \to B$ to its direct image map $\wp(f)\colon \wp(A) \to \wp(B)$ defined by $X \subseteq A \mapsto \{f(x) \mid x \in X\}$ is an *(endo)*-functor $\wp\colon \mathit{Set} \to \mathit{Set}$. ♢

Endofunctor

We can now give a more formal definition of *diagrams*: A diagram X of shape $\mathcal{J}$ in a category C is a functor $X\colon \mathcal{J} \to C$. The category J is mostly irrelevant and serves the purpose of defining the shape of the diagram X. Usually, J will be a finite category.

A *natural transformation* is a mapping between functors which preserves the composition of morphisms.

Natural transformation

Definition 2.16 (Natural Transformation)**.** Let $F, G\colon C \to D$ be two functors. A *natural transformation* $\eta\colon F \Rightarrow G$ maps every object X in C to a morphism $\eta_X\colon FX \to GX$ in D such that the following diagram commutes for each arrow $X \xrightarrow{f} Y$ in C:

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle \eta_X}\downarrow & & \downarrow{\scriptstyle \eta_Y} \\ GX & \xrightarrow[Gf]{} & GY \end{array}$$

♢

Example 2.12. Let $\wp\colon \mathit{Set} \to \mathit{Set}$ be the powerset functor and $\mathrm{id}_{\mathit{Set}}\colon \mathit{Set} \to \mathit{Set}$ the *identity functor* on *Set*, i.e., the functor that maps each object and arrow to itself. There is a natural transformation $\eta\colon \mathrm{id}_{\mathit{Set}} \Rightarrow \wp$ defined by taking each component $\eta_A\colon A \to \wp(A)$ the function $\eta_A(a) = \{a\}$ for each set A. ♢

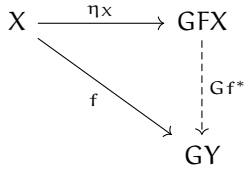
Identity functor

A natural transformation η is said to be a *natural isomorphism* if each component η_X is an isomorphism.

Natural isomorphism

Adjunction

Definition 2.17 (Adjunction)**.** An *adjunction* between two categories C and D is given by a pair of functors $F\colon C \to D$ and $G\colon D \to C$ and a natural transformation $\eta\colon \mathrm{id}_C \Rightarrow G \circ F$ such that for each object X in C and for each morphism $X \xrightarrow{f} GY$ there is a unique morphism $FX \xrightarrow{f^*} Y$ such that the following diagram commutes:

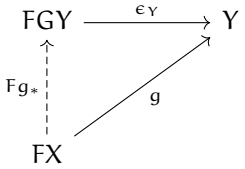


Left and right adjoint
Unit

Then, F is said to be the *left adjoint* of G, and vice versa G is the *right adjoint* of F. Moreover, the natural transformation η is called the *unit* of the adjunction (F, G). ♢

Counit

For each adjunction (F, G) with a unit η, there is another natural transformation $\epsilon\colon F \circ G \Rightarrow \mathrm{id}_D$, called the *counit* of the adjunction such that satisfy the dual property of the unit, that is for each morphism $FX \xrightarrow{g} Y$ in D there is a unique morphism $X \xrightarrow{g_*} GY$ such that the following diagram commutes



We sometimes write (F, G, ϵ, η) for an adjunction (F, G) with unit η and counit ϵ.

Example 2.13. Let $(P, \leqslant)$ and $(Q, \sqsubseteq)$ be two posets and $F\colon (P, \leqslant) \to (Q, \sqsubseteq)$ and $G\colon (Q, \sqsubseteq) \to (P, \leqslant)$ two functors between them when regarded as categories. Then, F is the left adjoint of G if and only if (F, G) is a Galois connection. ♢

Adjoint equivalence

Definition 2.18 (Adjoint Equivalence)**.** An adjoint (F, G, ϵ, η) is an *adjoint equivalence* if both η and ϵ are natural isomorphisms. ♢

It is possible to prove that an adjoint equivalence is actually an equivalence of categories according to the following definition:

Equivalence of categories

Definition 2.19 (Equivalence of Categories)**.** Let C and D be two categories. They are *equivalent* provided that there are two functors $F\colon C \to D$ and $G\colon D \to C$ and two natural isomorphisms $\eta\colon \mathrm{id}_C \Rightarrow G \circ F$ and $\epsilon\colon F \circ G \Rightarrow \mathrm{id}_D$. ♢

CHAPTER 3

Order-Sorted Algebras

Chapter reference(s): [GTWW77, GM92]

Order-sorted algebras [GM92] provide a mathematical tool for modeling formal systems as algebraic structures. The main concepts on which they rely are the notion of *sort*, to ensure a correct terms construction, and the notion of *subsort*, to provide several forms of *polymorphism*.

Order-sorted algebras generalise *many-sorted algebras* [Hig63]. The core new feature of order-sorted specifications is the addition of a partial ordering between sorts (the so-called *subsort relation*), enabling the definition of languages with polymorphic operators. Many-sorted algebras are themselves based on *universal algebra* [Bir35] to which they add sorts for avoiding meaningless terms.

In this chapter, we introduce the theory of order-sorted algebras. All the definitions, propositions, and theorems that follow were originally stated in [GM92]. Although the chapter does not provide new contributions, the way these concepts are presented, along with the examples and proofs, is rather new (unless otherwise stated) and designed to fit the material in subsequent chapters.

Structure The structure of the chapter is as follows: We begin by providing the basic notions of *sorted sets and functions*. Next we define the syntax of an *order-sorted signature*, that is the collection of *sorts* and *operators* from which well-formed *ground terms* are built. We then introduce *algebras* to provide signatures with a meaning. Algebras define *interpretation sets and functions* for each sort and operator in the signature, respectively. We prove that, under reasonable conditions, each algebra induces a unique *semantic function* (namely, a *homomorphism*). We proceed by defining the notion of *free algebra* over a set of *variables*, which leads to *term with variables*. Finally, we introduce *equational logic* in order to derive theorems from a given *theory*, that is a signature equipped with a set of *axioms*. We show that the process of deduction is both *sound* and *complete*.

3.1 Sorted Sets and Functions

Let S be a set whose members are understood as *sorts*. An S-*sorted set* A is a family

Sorted set

of sets indexed by S, and therefore denoted by

$$A = (\, A_s \mid s \in S \,)$$

We implicitly extend ordinary set-theoretic operators and predicates to S-sorted sets in a componentwise fashion. For instance, if A and B are two S-sorted sets, we write $A \subseteq B$ if $A_s \subseteq B_s$ for each $s \in S$, we define the cartesian product $A \times B$ by taking each indexed component $(A \times B)_s = A_s \times B_s$, etc. Moreover, we write $\bigcup A$ to denote the underlying set of the family A, that is $\bigcup A = \{\, a \in A_s \mid s \in S \,\}$.

Functions between sorted sets can be regarded as traditional functions that preserve the sort of elements. Formally, given two S-sorted sets A and B, an S-*sorted function* $h\colon A \to B$ is an S-sorted set $h \subseteq A \times B$ such that $h_s\colon A_s \to B_s$ is a set-theoretic function for each $s \in S$. If $f\colon A \to B$ and $g\colon B \to C$ are two S-sorted functions, their composition

Sorted function

$$g \circ f = \{\, (g \circ f)_s \mid s \in S \,\}$$

is a well-defined S-sorted function from A to C, where $(g \circ f)_s = g_s \circ f_s$.

We adopt some notational conventions that will come in handy when sequences of sorts are involved. Let A be an S-sorted set and $w = s_1 \dots s_n \in S^+$ a non-empty sequence of sorts. We set $A_w = A_{s_1} \times \cdots \times A_{s_n}$ the cartesian product of the components indexed by w. If $f\colon A \to B$ is an S-sorted function and $a_i \in A_{s_i}$ for $i = 1, \dots, n$, then $f_w\colon A_w \to B_w$ is defined by

$$f_w(a_1, \dots, a_n) = (f_{s_1}(a_1), \dots, f_{s_n}(a_n))$$

Finally, if the set of sorts S is equipped with a binary relation R, we often consider the extension of R — and denoted by the same symbol — to sequences *of the same length* in S^*. That is,

$$s_1 \dots s_n \; R \; s'_1 \dots s'_n \iff (\forall i = 1, \dots, n) \;\; s_i \; R \; s'_i$$

(and of course $\varepsilon \; R \; \varepsilon$). It is trivial to see that if R is a partial order over S, then so is its extension to S^*.

3.2 Order-Sorted Signatures

An *order-sorted signature* defines the symbols of the language (that is, the syntax). It specifies the *sorts*, the *operators*, and the *subsort relation* between sorts.

Order-sorted signature

Definition 3.1 (Order-Sorted Signature)**.** An *order-sorted signature Sg* is specified by

(i) a poset $(S, \leqslant)$ of *sorts*;

Function symbol

(ii) a set of *function symbols* $f\colon s_1, \dots, s_n \to s$ each with *arity* $n \geqslant 1$ and $(w, s) \in S^+ \times S$ the *rank* of f, where $w = s_1 \dots s_n$;

Constant

(iii) a set of *constants* $k\colon s$, each of a *unique rank* s (just a single sort); and

(iv) a *monotonicity requirement* that whenever $f\colon w_1 \to s$ and $f\colon w_2 \to r$ with $w_1 \leqslant w_2$, then $s \leqslant r$ (see Figure 3.1).

Operator

By an *operator* $\sigma\colon w \to s$ we mean either a function symbol $f\colon s_1, \dots, s_n \to s$ (when $w = s_1 \dots s_n \in S^+$) or a constant $k\colon s$ (when $w = \varepsilon$). ♢

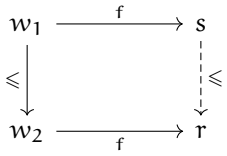


Figure 3.1: The graphical representation of the monotonicity requirement enforced by the order-sorted signature definition.

We shall see that one can think of sorts as syntactic categories, of subsort as inclusion, and of operators as building blocks of terms generated by the signature Sg.

Remark 3.1. The monotonicity constraint (3.1:iv), along with the forthcoming notion of *regularity* of an order-sorted signature (Definition 3.5), will ensure the existence of a unique least sort for each Sg-term. ♢

It sometimes turns out to be useful to consider all the operators in a given signature Sg as a single $(S^* \times S)$-sorted set $\Sigma = (\Sigma_{w,s} \mid w \in S^* \text{ and } s \in S)$, where each component is defined by

$$\begin{cases} \Sigma_{\varepsilon,s} = \{k \mid k\colon s \text{ in } Sg\} \\ \Sigma_{w,s} = \{f \mid f\colon w \to s \text{ in } Sg\} \qquad \text{with } w \in S^+ \end{cases}$$

Thus an order-sorted signature Sg can be also regarded as a triple $(S, \leqslant, \Sigma)$.

Polymorphism

Polymorphic function symbols can now be formally defined as those whose appear in more than one component of Σ. Furthermore, we can distinguish at least between two kinds of polymorphism: Let $f\colon w_1 \to s_1$ and $f\colon w_2 \to s_2$ be two distinct function symbols with the same name in Sg. We say they are

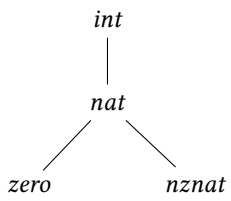
- *ad-hoc polymorphic*, if w_1 and w_2 are not related by $\leqslant$; and
- *subsort polymorphic*, if $w_1 \leqslant w_2$ or vice versa.

Ad-hoc polymorphism

Subsort polymorphism

Remark 3.2. Note that we have explictly ruled out polymorphic constants in the signature by requiring their rank to be unique (condition (3.1:iii)). Alternative but equivalent presentations (e.g., [GM92]) drop such a uniqueness constraint and extend the monotonicity requirement (condition (3.1:iv)) to also apply to constants. ♢

Example 3.1. We set up a simple order-sorted signature $Sg(Num)$ in order to specify elementary algebraic operations on natural and integer numbers (inspired from the Maude library [CDE+19, Sections 7.2 and 7.4]). Let $S(Num) = \{zero, nznat, nat, int\}$ be the set of sorts of $Sg(Num)$, where *nat* and *int* represent the sort of natural and integer numbers, while *zero* and *nznat* partition the set of naturals between zero and non-zero numbers, respectively. The subsort ordering on $S(Num)$ is given by the Hasse diagram on the left, whereas the operators are listed on the right:



$0\colon zero$	zero
$s\colon nat \to nznat$	(natural) successor
$p\colon nznat \to nat$	(natural) predecessor
$-\colon nznat \to int$	unary minus
$+\colon nat, nat \to nat$	natural numbers addition
$+\colon int, int \to int$	integer numbers addition

$$\text{(Const)}\ \frac{}{k\colon s}\ (k\colon s \text{ in } Sg) \qquad \text{(Subsort)}\ \frac{t\colon s}{t\colon r}\ (s \leqslant r \text{ in } Sg)$$

$$\text{(Fun)}\ \frac{(\forall 1 \leqslant i \leqslant n)\ \ t_i\colon s_i}{f(t_1,\ldots,t_n)\colon s}\ (f\colon s_1,\ldots,s_n \to s \text{ in } Sg)$$

Figure 3.2: Well-formed ground terms generated by an order-sorted signature Sg.

The symbol $+$ is subsort polymorphic in $Sg(Num)$ since $(nat, nat) \leqslant (int, int)$. However, we should pay attention when considering polymorphic symbols without explictly mentioning their rank: Suppose we slightly change $Sg(Num)$ to the new signature $Sg(Num + Bool)$, where the set of sorts is $S(Num + Bool) = S(Num) \cup \{bool\}$ and the operators are those of $Sg(Num)$ with the addition of the exclusive disjunction operator $+\colon bool, bool \to bool$. Now, the symbols $+\colon nat, nat \to nat$ and $+\colon int, int \to int$ are still subsort polymorphic but $+\colon nat, nat \to nat$ and $+\colon bool, bool \to bool$ are ad-hoc polymorphic. ♢

Example 3.2. In this example, we build the signature $Sg(\lambda)$ to model the syntax of the untyped lambda-calculus. The set of sorts $S(\lambda) = \{exp\}$ contains only the sort for expressions (that is, λ-terms), and the operators of $Sg(\lambda)$ are

$x\colon exp$	variable (for each $x \in Var$)
$\lambda_x\colon exp \to exp$	abstraction over x (for each $x \in Var$)
$\circ\colon exp, exp \to exp$	application

where Var is a countably infinite set of variables. ♢

3.2.1 Ground Terms

(Ground) Term

Given a signature Sg, we can build terms formed by the operators specified in it. In particular, well-formed *(ground) terms* generated by a signature Sg are defined by the inference rules depicted in Figure 3.2. A judgement of the form $t\colon s$ means that t is a well-formed term of sort s built out of Sg. Rule (Const) states that each constant $k\colon s$ is itself a term of sort s. They are the basic blocks of ground terms. Then, inductively, if we have valid terms $t_1\colon s_1, \ldots, t_n\colon s_n$ each of which of sort s_i, we can build compound terms $f(t_1,\ldots,t_n)\colon s$ by Rule (Fun), provided that $f\colon s_1,\ldots,s_n \to s$ is a function symbol in Sg. Finally, if s is a subsort of r and $t\colon s$, then t has also sort r by Rule (Subsort).

Example 3.3. Consider the signature $Sg(Num)$ of Example 3.1. Instances of terms generated by $Sg(Num)$ are

$$0\colon zero \quad s(s(0))\colon int \quad s(0)\colon nznat \quad +(s(0), -(s(0)))\colon int \quad \ldots$$

Note that the constant $0\colon zero$ cannot be polymorphic (see Remark 3.2) but the term 0 has any sort except $nznat$. ♢

Notation 3.1. Hereafter, when we refer to a signature Sg, we will leave implicit that $(S, \leqslant)$ is its poset of sorts. Moreover, we take it for granted that all the operators mentioned in the course of the exposition belong to Sg (or, equivalently, to the set of operators Σ associated to Sg), unless otherwise stated. ♢

3.3 Order-Sorted Algebras

Algebras are mathematical structures which give signatures an interpretation. They model sorts as sets (in which terms are provided with a meaning) and operators as functions between them.

Definition 3.2 (Order-Sorted Algebra)**.** An *order-sorted Sg-algebra* $\mathscr{A}$ over a signature *Sg* is specified by

Order-sorted algbera

(i) an *interpretation set* $[\![s]\!]_{\mathscr{A}}$ for each sort s and a set $[\![w]\!]_{\mathscr{A}} = [\![s_1]\!]_{\mathscr{A}} \times \cdots \times [\![s_n]\!]_{\mathscr{A}}$ for each $w = s_1 \dots s_n \in S^+$;

Interpretation set

(ii) an *interpretation function* $[\![f\colon w \to s]\!]_{\mathscr{A}}\colon [\![w]\!]_{\mathscr{A}} \to [\![s]\!]_{\mathscr{A}}$ for each $f\colon w \to s$ and an element $[\![k]\!]_{\mathscr{A}} \in [\![s]\!]_{\mathscr{A}}$ for each constant $k\colon s$;

Interpretation function

and by two monotonicity conditions requiring that

(iv) if $s \leqslant r$, then $[\![s]\!]_{\mathscr{A}} \subseteq [\![r]\!]_{\mathscr{A}}$; and

(v) if f appears with more than one rank $f\colon w_1 \to s$ and $f\colon w_2 \to r$ in *Sg* with $w_1 \leqslant w_2$ (that is, f is subsort polymorphic), then, for each $x \in [\![w_1]\!]_{\mathscr{A}}$,

$$[\![f\colon w_1 \to s]\!]_{\mathscr{A}}(x) = [\![f\colon w_2 \to r]\!]_{\mathscr{A}}(x) \qquad \diamondsuit$$

We often refer to the *carrier set* of an *Sg*-algebra $\mathscr{A}$, meaning the S-sorted family of interpretation sets $A = (\, A_s \mid s \in S \,)$, where $A_s = [\![s]\!]_{\mathscr{A}}$.

Carrier set

Notation 3.2. Please note that we use the letter A to denote the carrier set of the algebra $\mathscr{A}$, B that of $\mathscr{B}$, C that of $\mathscr{C}$, etc. $\diamondsuit$

Example 3.4. Consider the signature in Example 3.1. We can define an *Sg*(*Num*)-algebra $\mathscr{A}$ with interpretation sets $[\![\mathit{int}]\!]_{\mathscr{A}} = \mathbb{Z}$, $[\![\mathit{nat}]\!]_{\mathscr{A}} = \mathbb{N}$, $[\![\mathit{zero}]\!]_{\mathscr{A}} = \{0\}$, and $[\![\mathit{nznat}]\!]_{\mathscr{A}} = \mathbb{N} \setminus \{0\}$, and interpretation functions

$$\begin{aligned}
[\![0]\!]_{\mathscr{A}} &= 0 \\
[\![s\colon \mathit{nat} \to \mathit{nznat}]\!]_{\mathscr{A}}(n) &= n + 1 \\
[\![p\colon \mathit{nznat} \to \mathit{nat}]\!]_{\mathscr{A}}(m) &= m - 1 \\
[\![-\colon \mathit{nznat} \to \mathit{int}]\!]_{\mathscr{A}}(m) &= -m \\
[\![+\colon \mathit{nat}, \mathit{nat} \to \mathit{nat}]\!]_{\mathscr{A}}(n_1, n_2) &= n_1 + n_2 \\
[\![+\colon \mathit{int}, \mathit{int} \to \mathit{int}]\!]_{\mathscr{A}}(z_1, z_2) &= z_1 + z_2
\end{aligned}$$

where $n, n_1, n_2 \in \mathbb{N}$, $m \in \mathbb{N} \setminus \{0\}$, and $z, z_1, z_2 \in \mathbb{Z}$. $\diamondsuit$

The conditions (3.2:iv) and (3.2:v) that an *Sg*-algebra $\mathscr{A}$ has to meet formalise the intuitive requirements for the semantics of polymorphism:

- If s is a subsort of r in *Sg*, then by Rule (Subsort) every term of sort s has also sort r. Therefore, if we aim to give meaning to terms of sort s in $[\![s]\!]_{\mathscr{A}}$, then it is natural to require that $[\![s]\!]_{\mathscr{A}} \subseteq [\![r]\!]_{\mathscr{A}}$.
- If $f\colon w_1 \to s$ and $f\colon w_2 \to r$ with $w_1 \leqslant w_2$ in *Sg*, then $[\![w_1]\!]_{\mathscr{A}} \subseteq [\![w_2]\!]_{\mathscr{A}}$ by condition (3.2:iv). Therefore, we want to ensure the meaning of f to be consistent on the smaller interpretation set $[\![w_1]\!]_{\mathscr{A}}$.

Among all the algebras interpreting a signature *Sg*, the term construction illustrated in Section 3.2.1 gives rise to the so-called *term algebra*. Intuitively, the interpretation provided by the term algebra is the syntactical one:

Order-sorted term algebra

Definition 3.3 (Order-Sorted Term Algebra)**.** Let *Sg* be an order-sorted signature. The *order-sorted term algebra* $\mathscr{T}_{Sg}$ over *Sg* is defined as follows:

(i) $[\![s]\!]_{\mathscr{T}_{Sg}} = \{\, t \mid t\colon s \,\}$ for each sort s;

(ii) $[\![k]\!]_{\mathscr{T}_{Sg}} = k$ for each constant $k\colon s$; and

(iii) for each function symbol $f\colon s_1, \ldots, s_n \to s$, the corresponding interpretation function is

$$[\![f\colon w \to s]\!]_{\mathscr{T}_{Sg}}\colon [\![s_1]\!]_{\mathscr{T}_{Sg}} \times \cdots \times [\![s_n]\!]_{\mathscr{T}_{Sg}} \to [\![s]\!]_{\mathscr{T}_{Sg}}$$
$$(t_1, \ldots, t_n) \mapsto f(t_1, \ldots, t_n)$$

where $w = s_1 \ldots s_n$. ♢

The previous definition states that a term t is a member of $[\![s]\!]_{\mathscr{T}_{Sg}}$ if and only if t has sort s. Following Notation 3.2, we denote the carrier set of $\mathscr{T}_{Sg}$ by

$$T_{Sg} = (\, (T_{Sg})_s \mid s \in S \,)$$

Moreover, "term semantics" of operators in *Sg* is the (prefix) syntactic application of the operator names.

Proposition 3.1. *$\mathscr{T}_{Sg}$ is a proper order-sorted Sg-algebra.*

3.4 Algebraic Semantics

Semantic function

The fundamental reason for defining an *Sg*-algebra $\mathscr{A}$ is to gain a *semantic function* $[\![-]\!]_{\mathscr{A}}$ able to assign a meaning $[\![t]\!]_{\mathscr{A}} \in [\![s]\!]_{\mathscr{A}}$ to each term t of sort s. Since t is inductively built up from constants and function symbols of the signature *Sg* (according to the rules in Figure 3.2), we would like its meaning to be computed by recursively applying the interpretation functions provided by $\mathscr{A}$ to each operator forming t. Such a property is known as *compositionality*, and it is summed up by the next quotation from [SS71]:

> *The semantical definition is "syntax-directed" in that it follows the same order of clauses and transforms each language construct into the intended operations on the meanings of the parts.* Scott and Strachey

It is therefore natural to regard $[\![-]\!]_{\mathscr{A}}$ as an S-sorted function from the carrier set of $\mathscr{T}_{Sg}$ to that of $\mathscr{A}$. The compositionality nature of the semantic function is obtained by requiring $[\![-]\!]_{\mathscr{A}}$ to behave like a homomorphism with respect to the interpretation functions:

Order-sorted homomorphism

Definition 3.4 (Order-Sorted Homomorphism)**.** Let *Sg* be an order-sorted signature. An *order-sorted Sg-homomorphism* $h\colon \mathscr{A} \to \mathscr{B}$ between two *Sg*-algebras $\mathscr{A}$ and $\mathscr{B}$ is an S-sorted function $h\colon A \to B$ between their carrier sets satisfying the following:

(i) $h_s([\![k]\!]_{\mathscr{A}}) = [\![k]\!]_{\mathscr{B}}$ for each $k\colon s$ and $h_s \circ [\![f\colon w \to s]\!]_{\mathscr{A}} = [\![f\colon w \to s]\!]_{\mathscr{B}} \circ h_w$ (see Figure 3.3) for each $f\colon w \to s$; and

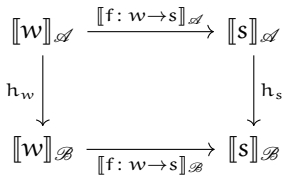


Figure 3.3: The commutativity diagram enforced by the order-sorted homomorphism definition.

(ii) if $s \leqslant r$, then $h_s(x) = h_r(x)$ for each $x \in [\![s]\!]_{\mathscr{A}}$. ♢

Before commenting on the previous definition, it is worth noting that *Sg*-algebras and homomorphisms form a category:

Proposition 3.2. *Let Sg be an order-sorted signature. The class of all Sg-algebras and the class of all Sg-homomorphisms form a category Alg(Sg).*

Category of algebras over *Sg*

The first requirements (3.4:i) yields exactly the compositionality of the semantic function: Let $h\colon \mathscr{T}_{Sg} \to \mathscr{A}$ be an *Sg*-homomorphism out of the term algebra (that is, h provides terms with a meaning). Consider now an arbitrary compound term $t = f(t_1, \ldots, t_n)$, with $f\colon w \to s$ and $w = s_1 \ldots s_n$ in *Sg*. The meaning of t given by h is *compositional*, in the following sense:

$$\begin{aligned} & h_s(f(t_1, \ldots, t_n)) \\ & \quad \overset{\lightning}{=} h_s([\![f\colon w \to s]\!]_{\mathscr{T}_{Sg}}(t_1, \ldots, t_n)) && \text{by cond. (3.3:iii)} \\ & \quad = [\![f\colon w \to s]\!]_{\mathscr{A}}(h_{s_1}(t_1), \ldots, h_{s_n}(t_n)) && \text{by (3.4:i)} \end{aligned} \tag{3.1}$$

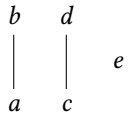
that reads as "the meaning of $f(t_1, \ldots, t_n)$ is the meaning of f applied to the meaning of the constituents $t_1, \ldots, t_n$".

Unfortunately, Equation 3.1 carries an issue ($\lightning$). Due to polymorphism, the function symbol f may have multiple ranks, therefore, in general, we cannot simply pick one as we erroneously did. However, if $f(t_1, \ldots, t_n)$ and each subterm $t_1, \ldots, t_n$ have a least sort, then condition (3.2:v) ensures the existence of a unique interpretation of $f(t_1, \ldots, t_n)$ under h regardless of the chosen rank (w, s). In the following section, we restrict the family of signatures in order to get such a least sort for each well-formed term, which in turn ensures the uniqueness of the semantic function 3.1.

3.4.1 Initial Algebra Semantics

Regularity of order-sorted signature is a property which rules out some inherently ambiguous signatures. The next example helps to clarify.

Example 3.5. Consider the following (non-regular) order-sorted signature $Sg(\mathbb{R})$. Let the poset of sorts $S(\mathbb{R})$ be given by the Hasse diagram on the left and the operators listed on the right:



$k\colon a$	a constant of sort a
$l\colon c$	a constant of sort c
$f\colon a, d \to e$	a polymorphic function symbol
$f\colon b, c \to e$	

Suppose that $\mathscr{A}$ is an $Sg(\mathfrak{R})$-algebra and consider the term $f(k, l)$. There are multiple ways to compute its semantics in a compositional way:

$$[\![f(k,l)]\!]_{\mathscr{A}} = \begin{cases} [\![f\colon a, d \to e]\!]_{\mathscr{A}}([\![k]\!]_{\mathscr{A}}, [\![l]\!]_{\mathscr{A}}) \\ [\![f\colon b, c \to e]\!]_{\mathscr{A}}([\![k]\!]_{\mathscr{A}}, [\![l]\!]_{\mathscr{A}}) \end{cases}$$

Unfortunately, since these symbols are not subsort polymorphic (hence (3.2:v) does not apply), there is no guarantee for these interpretations to be the same. Such an issue can be linked to the absence of a least rank for the operator f when applied to a pair of terms with sort a and c, respectively. ♢

Regularity

Definition 3.5 (Regularity)**.** An order-sorted signature Sg is *regular* if for each function symbol $f\colon w \to s$ and for each lower bound $w_0 \leqslant w$ the set

$$\{(w', s') \in S^+ \times S \mid f\colon w' \to s' \wedge w_0 \leqslant w'\}$$

has a minimum (the ordering we are considering on $S^+ \times S$ is the product order), called the *least rank* of f with respect to w_0. ♢

Least rank

Note that if $f\colon w \to s$ has least rank $(\overline{w}, \overline{s})$ with respect to some lower bound $w_0 \leqslant w$, then $f\colon \overline{w} \to \overline{s}$ appears in Sg.

The main reason to introduce regularity is the following proposition:

Proposition 3.3. *Let Sg be a regular order-sorted signature. Then, for each well-formed ground term $t\colon s$ there is a least sort $ls(t)$ such that $t\colon ls(t)$ and $ls(t) \leqslant s$.*

Proof. The proof is by rule induction on $t\colon s$.

1. (Const) Suppose that $t\colon s$ has been derived by Rule (Const). Therefore $t = k$ for some constant $k\colon s$ in Sg. Since the rank of constants is unique, we set $ls(k) = s$. Now suppose that $k\colon s'$ is a well-formed ground term. By a second rule induction on $k\colon s'$, we show that $s \leqslant s'$:

   a) (Const) If $k\colon s'$ has been derived by (Const), then $k\colon s'$ appears in Sg and, by uniqueness of rank of constants, we have $s = s'$.

   b) (Subsort) Suppose that $k\colon s'$ has been derived by Rule (Subsort), that is

   $$\frac{k\colon r}{k\colon s'}\ (r \leqslant s' \text{ in } Sg)$$

   Then, by induction hypothesis, $s \leqslant r$ and therefore $s \leqslant s'$.

2. (Fun) If $t\colon s$ has been derived by (Fun), then

   $$\frac{(\forall 1 \leqslant i \leqslant n)\ \ t_i\colon s_i}{f(t_1, \ldots, t_n)\colon s}\ (f\colon s_1, \ldots, s_n \to s \text{ in } Sg)$$

   where $t = f(t_1, \ldots, t_n)$. By induction hypothesis, there is $ls(t_i) \leqslant s_i$ such that $t_i\colon ls(t_i)$ is a well-formed ground term, for each $i = 1, \ldots, n$. Let $w_0 = ls(t_1) \ldots ls(t_n)$ and $w = s_1 \ldots s_n$. Then, $w_0 \leqslant w$ and, by the regularity of Sg, $f\colon w \to s$ has least rank $(\overline{w}, \overline{s})$ with respect to w_0, for some $(\overline{w}, \overline{s}) \in S^+ \times S$

with $\overline{w} = \overline{s_1} \dots \overline{s_n}$ and $w_0 \leqslant \overline{w}$. Therefore, $f\colon \overline{s_1}, \dots, \overline{s_n} \to \overline{s}$ is in Sg, and we can derive

$$\dfrac{\dfrac{t_1\colon ls(t_1)}{t_1\colon \overline{s_1}}\ (\text{Subsort}) \quad \dots \quad \dfrac{t_n\colon ls(t_n)}{t_n\colon \overline{s_n}}\ (\text{Subsort})}{f(t_1,\dots,t_n)\colon \overline{s}}\ (\text{Fun})$$

and set $ls(f(t_1,\dots,t_n)) = \overline{s}$. Now suppose that $f(t_1,\dots,t_n)\colon s'$ is a well-formed ground term. By a second rule induction on $f(t_1,\dots,t_n)\colon s'$ we show that $\overline{s} \leqslant s'$:

a) (Fun) Suppose that $f(t_1,\dots,t_n)\colon s'$ has been derived by (Fun), that is

$$\dfrac{(\forall 1 \leqslant i \leqslant n)\ \ t_i\colon s_i'}{f(t_1,\dots,t_n)\colon s'}\ (f\colon s_1',\dots,s_n' \to s' \text{ in } Sg)$$

Let $w' = s_1' \dots s_n'$. Then, $w_0 \leqslant w'$ and therefore $(\overline{w}, \overline{s}) \leqslant (w', s')$ implying that $\overline{s} \leqslant s'$.

b) (Subsort) The proof is analogous to Case 1b.

3. (Subsort) Again, the proof is analogous to Case 1b. □

The following theorem ensures that cases such as the one presented in Example 3.5 no longer appear. That is, given any order-sorted algebra $\mathscr{A}$ over a *regular* signature Sg, there is exactly one single way (namely, a homomorphism) to provide Sg-terms with a meaning induced by $\mathscr{A}$.

Theorem 3.1. *Let Sg be a regular signature. Then, $\mathscr{T}_{Sg}$ is the initial object of $Alg(Sg)$.*

Proof. Recall the initial object definition, and let $\mathscr{A}$ be an order-sorted algebra over Sg. (1) We first define an S-sorted function $h\colon T_{Sg} \to A$ from the carrier set T_{Sg} of $\mathscr{T}_{Sg}$ to the carrier set A of $\mathscr{A}$. (2) Then, we prove that h is an Sg-homomorphism $h\colon \mathscr{T}_{Sg} \to \mathscr{A}$ in the category of algebras $Alg(Sg)$. (3) Finally, we show that for any other Sg-homomorphism $l\colon \mathscr{T}_{Sg} \to \mathscr{A}$ holds that $h = l$.

(1) Let $t \in [\![r]\!]_{\mathscr{T}_{Sg}}$ be an Sg-term of sort r, that is, $t\colon r$. We define $h_r(t)$ by structural induction on t:

- $t = k$ for some constant $k\colon ls(k)$ in Sg. Then, we define

$$h_r(k) = [\![k]\!]_{\mathscr{A}} \tag{3.2}$$

Note that $[\![k]\!]_{\mathscr{A}} \in [\![r]\!]_{\mathscr{A}}$, because $ls(k) \leqslant r$ and therefore, by condition (3.2:iv), $[\![ls(k)]\!]_{\mathscr{A}} \subseteq [\![r]\!]_{\mathscr{A}}$.

- $t = f(t_1,\dots,t_n)$ for some function symbol $f\colon s_1,\dots,s_n \to s$ in Sg with $t_i\colon s_i$ for $i = 1,\dots,n$. Let $w = s_1 \dots s_n$. Then, $w_0 = ls(t_1) \dots ls(t_n) \leqslant w$ and by regularity there is $f\colon \overline{w} \to \overline{s}$ in Sg with $(\overline{w}, \overline{s})$ the least rank of $f\colon w \to s$ with respect to w_0. Let $\overline{w} = \overline{s_1} \dots \overline{s_n}$. We recursively define

$$h_r(f(t_1,\dots,t_n)) = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_{\overline{s_1}}(t_1),\dots,h_{\overline{s_n}}(t_n)) \tag{3.3}$$

and we prove that it is well-defined. First, we observe that $t_i\colon \overline{s_i}$ since $ls(t_i) \leqslant \overline{s_i}$. Second, $f(t_1,\dots,t_n)\colon r$ can be derived

- either by applying Rule (Fun), and therefore $\overline{s} \leqslant r$ since the definition of $h_r(f(t_1, \ldots, t_n))$ does not depend on the choice of the rank (w, s) of f, or
- by applying Rule (Subsort) for some $s' \leqslant r$ and therefore, by induction hypothesis, $\overline{s} \leqslant s' \leqslant r$.

In both cases we conclude that $[\![\overline{s}]\!]_{\mathscr{A}} \subseteq [\![r]\!]_{\mathscr{A}}$.

(2) We now prove that h is an *Sg*-homomorphism. We first show that $h_s(t) = h_r(t)$ if $s \leqslant r$ for each $t \in [\![s]\!]_{\mathscr{T}_{Sg}}$. Suppose that $t = k$. Then, $h_s(k) = [\![k]\!]_{\mathscr{A}} = h_r(k)$. Let now $t = f(t_1, \ldots, t_n)$. Regardless of the choice of the rank of f (see the previous point), we have that $h_s(f(t_1, \ldots, t_n)) = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_{\overline{s_1}}(t_1), \ldots, h_{\overline{s_n}}(t_n)) = h_r(f(t_1, \ldots, t_n))$, where $(\overline{w}, \overline{s})$ is the least rank of f with respect to $w_0 = ls(t_1) \ldots ls(t_n)$. We now show the compositionality of h:

- Let $k\colon s$ be a constant in *Sg*. Then,

$$\begin{aligned} h_s([\![k]\!]_{\mathscr{T}_{Sg}}) &= h_s(k) && \text{by cond. (3.3:ii)} \\ &= [\![k]\!]_{\mathscr{A}} && \text{by Eq. 3.2} \end{aligned}$$

- Let $f\colon w \to s$ be a function symbol in *Sg* with $w = s_1 \ldots s_n$ and $(t_1, \ldots, t_n) \in [\![w]\!]_{\mathscr{T}_{Sg}}$, that is $t_i\colon s_i$ for each $i = 1, \ldots, n$. Let $(\overline{w}, \overline{s})$ be the least rank of f with respect to $w_0 = ls(t_1) \ldots ls(t_n)$. Then,

$$\begin{aligned} &h_s([\![f\colon w \to s]\!]_{\mathscr{T}_{Sg}}(t_1, \ldots, t_n)) \\ &\quad = h_s(f(t_1, \ldots, t_n)) && \text{by cond. (3.3:iii)} \\ &\quad = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_{\overline{s_1}}(t_1), \ldots, h_{\overline{s_n}}(t_n)) && \text{by Eq. 3.3} \\ &\quad = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_{\overline{w}}(t_1, \ldots, t_n)) && \overline{w} = \overline{s_1} \ldots \overline{s_n} \\ &\quad = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_w(t_1, \ldots, t_n)) && \text{by (3.4:ii) since } \overline{w} \leqslant w \end{aligned}$$

(3) Finally, we prove the uniqueness of h. Let $l\colon \mathscr{T}_{Sg} \to \mathscr{A}$ be an order-sorted *Sg*-homomorphism. We show that $h_r(t) = l_r(t)$ for each $t\colon r$ by structural induction on t:

- Let $t = k$ for some constant $k\colon ls(k)$ in *Sg*. Since l is an order-sorted homomorphism, it satisfies (3.4:i). Then,

$$l_r(k) = [\![k]\!]_{\mathscr{A}} = h_r(k)$$

- Let $t = f(t_1, \ldots, t_n)$ for some function symbol $f\colon s_1, \ldots, s_n \to s$ in *Sg* with $t_i\colon s_i$ for each $i = 1, \ldots, n$. Let $w = s_1 \ldots s_n$ and, by regularity, $f\colon w \to s$ admits a least rank $(\overline{w}, \overline{s})$ with respect to $w_0 = ls(t_1) \ldots ls(t_n)$, where $\overline{w} = \overline{s_1} \ldots \overline{s_n}$. Following the same argument in (1), we have that

$\overline{s} \leqslant r$. Then,

$$
\begin{aligned}
&l_r(f(t_1,\ldots,t_n)) \\
&\quad = l_{\overline{s}}(f(t_1,\ldots,t_n)) && \text{by (3.4:ii) on } l \\
&\quad = l_{\overline{s}}([\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{T}_{Sg}}(t_1,\ldots,t_n)) && \text{by cond. (3.3:iii)} \\
&\quad = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(l_{\overline{s_1}}(t_1),\ldots,l_{\overline{s_n}}(t_n)) && \text{by (3.4:i) on } l \\
&\quad = [\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{A}}(h_{\overline{s_1}}(t_1),\ldots,h_{\overline{s_n}}(t_n)) && \text{by ind. hyp.} \\
&\quad = h_{\overline{s}}([\![f\colon \overline{w} \to \overline{s}]\!]_{\mathscr{T}_{Sg}}(t_1,\ldots,t_n)) && \text{by (3.4:i) on } h \\
&\quad = h_{\overline{s}}(f(t_1,\ldots,t_n)) && \text{by cond. (3.3:iii)} \\
&\quad = h_r(f(t_1,\ldots,t_n)) && \text{by (3.4:ii) on } h
\end{aligned}
$$

Hence $h\colon \mathscr{T}_{Sg} \to \mathscr{A}$ is the unique Sg-homomorphism from $\mathscr{T}_{Sg}$ to $\mathscr{A}$. □

The previous theorem establishes a one-one correspondence between algebras over a regular signature Sg and homomorphisms out of $\mathscr{T}_{Sg}$. It is therefore natural to extend the notation $[\![-]\!]_{\mathscr{A}}$ from operators in Sg to ground terms:

Term semantics

$$[\![t]\!]_{\mathscr{A}} = h_{ls(t)}(t) \tag{3.4}$$

where t is a term built out of Sg, $\mathscr{A}$ an order-sorted Sg-algebra, and $h\colon \mathscr{T}_{Sg} \to \mathscr{A}$ the unique homomorphism.

It is important to emphasise that the initial algebra approach to semantics does not rely on the "internal" definition of $\mathscr{T}_{Sg}$ but only on its property of being initial. This fact is captured by the following proposition:

Proposition 3.4. *Let $\mathscr{T}_1$ and $\mathscr{T}_2$ be two Sg-algebras.*

(i) If $\mathscr{T}_1$ and $\mathscr{T}_2$ are initial in $Alg(Sg)$, then they are isomorphic.

(ii) If $\mathscr{T}_1$ is initial and $\mathscr{T}_2$ is isomorphic to $\mathscr{T}_1$, then $\mathscr{T}_2$ is initial.

In particular, (3.4:ii) ensures that if we favour a different but isomorphic term representation (such as syntax trees, infix terms, labelled graphs, etc.) in place of that provided by $\mathscr{T}_{Sg}$, we can still prove the same theorems and propositions. For this reasons, it is customary to identify the *abstract syntax* (in the sense of [McC62a]) with the initial algebra, since it is independent of notational variation [GTWW77].

Abstract syntax

Example 3.6. Recall the signature $Sg(\lambda)$ of the untyped lambda-calculus shown in Example 3.2. We now define an $Sg(\lambda)$-algebra $\mathscr{E}$ (inspired by [GTWW77]) to provide meaning to λ-terms. Let $\mathbb{N}_\perp$ be the set natural numbers with $\perp$ adjoined, that is $\mathbb{N}_\perp = \mathbb{N} \cup \{\perp\}$. By [Sco76], there is a domain V of *values* satisfying the isomorphism

$$V \underset{\phi}{\overset{\psi}{\rightleftharpoons}} \mathbb{N}_\perp \oplus [V \to V]$$

where $[V \to V]$ is the set of continuous functions from V to V and $\oplus$ the disjoint union with bottom elements identified (the so-called *coalesced sum*). Let $X \in \{\mathbb{N}_\perp, [V \to V]\}$ and $\kappa_X\colon X \to \mathbb{N}_\perp \oplus [V \to V]$ the injection from X to $\mathbb{N}_\perp \oplus [V \to V]$. Since an injection over a non-empty domain always admits a left inverse, we denote by $\kappa_X^{-1}\colon \mathbb{N}_\perp \oplus [V \to V] \to X$ the left inverse of κ_X. Then we define a second injection $\iota_X\colon X \to V$ into V by $\iota_X = \psi \circ \kappa_X$ and the projection $\pi_X\colon V \to X$ onto X by $\pi_X = \kappa_X^{-1} \circ \phi$.

Let $Env = Var \to V$ be the set of *environments*. Meanings of λ-terms are continuous functions in $[Env \to V]$, and therefore we set $[\![exp]\!]_{\mathscr{E}} = [Env \to V]$. In order to define interpretation functions of $Sg(\lambda)$ operators, we will make use of the following functions:

- the function $val_x \colon Env \to V$ provides the value of the variable $x \in Var$ in the environment $\eta \in Env$, that is $val_x(\eta) = \eta(x)$;
- $assign_x \colon Env \times V \to Env$ updates the given environment by assigning a value v to x:
$$assign_x(\eta, v) = x' \in Var \mapsto \begin{cases} v & \text{if } x' = x \\ \eta(x') & \text{otherwise} \end{cases}$$
- the function $apply \colon V^2 \to V$ projects the first parameter into the function space $[V \to V]$, then applies the resulting function to the second parameter, i.e., $apply(v_1, v_2) = \pi_{[V \to V]}(v_1)(v_2)$; and
- the *curry* operator $curry_{D_1, D_2, D_3} \colon [D_1 \times D_2 \to D_3] \to [D_1 \to [D_2 \to D_3]]$ defined by $curry_{D_1, D_2, D_3}(f) = x \in D_1 \mapsto \big(y \in D_2 \mapsto f(x, y)\big)$.

All these functions can be proven to be continuous. Moreover, the composition of continuous functions is continuous. Finally, given a finite number of continuous functions $e_i \colon Env \to V$ with $1 \leqslant i \leqslant n$ for some $n \in \mathbb{N}$, the *tupling* $\langle e_1, \ldots, e_n \rangle \colon Env \to V^n$ of the e_i defined on an environment η by $(e_1(\eta), \ldots, e_n(\eta))$ is continuous. We can now provide the interpretation functions of the operators:

$$\begin{aligned} [\![x]\!]_{\mathscr{E}} &= val_x \\ [\![\lambda_x \colon exp \to exp]\!]_{\mathscr{E}}(e) &= \iota_{[V \to V]} \circ curry(e \circ assign_x) \\ [\![\circ \colon exp, exp \to exp]\!]_{\mathscr{E}}(e_1, e_2) &= apply \circ \langle e_1, e_2 \rangle \end{aligned}$$

where the domain of definition of the *curry* operator is clear by the context. Now, the unique $Sg(\lambda)$-homomorphism $eval \colon \mathscr{T}_{Sg(\lambda)} \to \mathscr{E}$ provide λ-terms with a meaning. As an example of application of *eval*, let num_n be the λ-term corresponding to the n-th Church numeral defined by[1]

$$\mathsf{num}_n = \lambda f \,.\, \lambda x \,.\, \underbrace{f\,(f\,(\cdots f}_{n\text{-times}}\, x)\cdots))$$

and $\mathsf{succ} = \lambda n \,.\, \lambda f \,.\, \lambda x \,.\, f\,(n\,f\,x)$ the successor operator. Please note that f, x, and n are concrete instances of $x \in Var$. We want to check that $[\![\circ(\mathsf{succ}, \mathsf{num}_0)]\!]_{\mathscr{E}} = [\![\mathsf{num}_1]\!]_{\mathscr{E}}$. According to Equations 3.4 and 3.3, we have

$$\begin{aligned} [\![\circ(\mathsf{succ}, \mathsf{num}_0)]\!]_{\mathscr{E}} &= eval_{exp}(\circ(\mathsf{succ}, \mathsf{num}_0)) \\ &= [\![\circ \colon exp, exp \to exp]\!]_{\mathscr{E}}(eval_{exp}(\mathsf{succ}), eval_{exp}(\mathsf{num}_0)) \\ &= apply \circ \langle eval_{exp}(\mathsf{succ}), eval_{exp}(\mathsf{num}_0) \rangle \\ &\overset{\star}{=} \eta \in Env \mapsto (\pi_{[V \to V]}(eval_{exp}(\mathsf{succ})(\eta))(eval_{exp}(\mathsf{num}_0)(\eta)) \end{aligned}$$

[1] We use the standard λ-calculus notation instead of the (more cumbersome) prefix one. In order to make the correspondence between the two notations clear, we assume that the application is left associative and that the body of an abstraction extends on the right as far as possible.

where

$$eval_{exp}(\mathsf{succ}) = \eta \in Env \mapsto \iota_{[V\to V]}(n \in V \mapsto \iota_{[V\to V]}(f \in V \mapsto \iota_{[V\to V]}(x \in V \mapsto \pi_{[V\to V]}(f)(\pi_{[V\to V]}(\pi_{[V\to V]}(n)(f))(x)))))$$

and

$$eval_{exp}(\mathsf{num}_0) = \eta \in Env \mapsto \iota_{[V\to V]}(f \in V \mapsto \iota_{[V\to V]}(v \mapsto v))$$

and therefore

$$\begin{aligned}[\![\circ(\mathsf{succ}, \mathsf{num}_0)]\!]_{\mathscr{E}} &\stackrel{\star}{=} \eta \in Env \mapsto (\pi_{[V\to V]}(eval_{exp}(\mathsf{succ})(\eta))(eval_{exp}(\mathsf{num}_0)(\eta)) \\ &= \eta \in Env \mapsto \iota_{[V\to V]}(f \in V \mapsto \iota_{[V\to V]}(x \in V \mapsto \\ &\qquad \pi_{[V\to V]}(f)(x))) \\ &= [\![\mathsf{num}_1]\!]_{\mathscr{E}} \end{aligned}$$ ♢

3.5 Free Algebra Construction

A free algebra over a signature *Sg* is the loosest algebra generated starting from a set X whose elements are understood as *variables*. The underlying set of a free algebra over X carries *terms with variables* in X and its operations are defined as free as possible [RT91], namely by interpreting all the operators in a purely syntactic way. In this section, we shall define the free algebra $\mathscr{T}_{Sg}(X)$ which yields *terms with variables*, as opposed to *ground terms* carried by the term algebra $\mathscr{T}_{Sg}$.

Let *Sg* be an order-sorted signature and X an S-sorted set of variables termed *variable set*. In the rest of this chapter, we assume that $X_s \cap X_{s'} = \varnothing$ for each $s, s' \in S$ such that $s \neq s'$ and $\bigcup S \cap \bigcup \Sigma = \varnothing$ (recall the notation in Section 3.1 and the definition of Σ). We usually refer to this fact by saying that X is a disjoint set of variables also disjoint from *Sg*. Moreover, if $\mathscr{A}$ is an algebra, we call *assignment* any S-sorted function $\alpha\colon X \to A$, where A is the carrier set of the algebra $\mathscr{A}$.

Variable set

Assignment

Terms over Sg with variables in X are generated by the rules in Figure 3.2 with the addition of the following Rule (Var):

Terms with variables

$$\text{(Var)}\ \frac{}{x\colon s}\ (x \in X_s)$$

Therefore, by treating variables in X_s as terms of sort s, we can inductively build compound terms containing variables by applying the other rules in Figure 3.2.

Example 3.7. Let *Sg*(*Num*) be the signature of Example 3.1. Examples of terms over *Sg*(*Num*) with variables in a set X are (assuming $n \in X_{nat}$ and $z \in X_{int}$)

$$n\colon nat \quad n\colon int \quad s(0)\colon nznat \quad +(s(n), z)\colon int \quad \ldots$$ ♢

It is easy to see that terms over *Sg* with variables in X coincide with ground terms over a new signature *Sg*(X) which contains variables in X as constants:

Definition 3.6. Let *Sg* be an order-sorted signature and X an S-sorted set of variables. Then, the *order-sorted signature Sg*(X) is defined by taking the same set of sorts S of *Sg* with the same subsort ordering ⩽, and

- if $f\colon w \to s$ in *Sg*, then $f\colon w \to s$ in *Sg*(X);

- if $k\colon s$ in Sg, then $k\colon s$ in $Sg(X)$; and
- if x is a variable in X_s, then $x\colon s$ is a constant in $Sg(X)$. ♢

Note that $Sg(X)$ is a proper order-sorted signature by the hypothesis we made on X (that is, X is a disjoint set of variables disjoint from Sg). Moreover, $Sg(X)$ is regular if and only if Sg is regular, since we did not add any polymorphic operators.

Remark 3.3. Such an hypothesis is not strictly required. If X is not a disjoint set of variables disjoint from Sg, we can make it so by considering the new set $\eta(X)$ defined by $\eta(X)_s = \{ (Sg, x_s) \mid x \in X_s \}$ and then proceed with $\eta(X)$ instead of X. ♢

Proposition 3.5. *$t\colon s$ is a term over Sg with variables in X if and only if $t\colon s$ is a ground term in $Sg(X)$.*

We are now interested in making $\mathscr{T}_{Sg(X)}$ into an Sg-algebra $\mathscr{T}_{Sg}(X)$, the so-called *free algebra* on X.

Order-sorted free algebra

Definition 3.7 (Order-Sorted Free Algebra)**.** Let Sg be an order-sorted signature and X an S-sorted set of variables. The *order-sorted free Sg-algebra* $\mathscr{T}_{Sg}(X)$ is defined simply by forgetting about variables in $Sg(X)$ and by taking the same definition of $\mathscr{T}_{Sg(X)}$, that is

(i) $[\![s]\!]_{\mathscr{T}_{Sg}(X)} = [\![s]\!]_{\mathscr{T}_{Sg}}$ for each sort $s \in S$;

(ii) $[\![k]\!]_{\mathscr{T}_{Sg}(X)} = [\![k]\!]_{\mathscr{T}_{Sg}}$ for each $k\colon s$; and

(iii) $[\![f\colon w \to s]\!]_{\mathscr{T}_{Sg}(X)} = [\![f\colon w \to s]\!]_{\mathscr{T}_{Sg}}$ for each $f\colon w \to s$. ♢

Note that the free algebra $\mathscr{T}_{Sg}(X)$ is defined over the original signature Sg, whereas the term algebra $\mathscr{T}_{Sg(X)}$ is defined over $Sg(X)$. Indeed, while $\mathscr{T}_{Sg(X)}$ carries the variables as constants of the signature, the free algebra keeps only their term representation in its carrier set, allowing homomorphisms to freely move their meaning to any another algebra in the same category. Therefore, we have the following:

Theorem 3.2. *Let Sg be a regular order-sorted signature, X an S-sorted set of variables, and $\mathscr{A}$ an order-sorted Sg-algebra. Then, given an assignment $a\colon X \to A$ (that is, an S-sorted function from X into the carrier set A of the algebra $\mathscr{A}$), there is a unique Sg-homomorphism $a^*\colon \mathscr{T}_{Sg}(X) \to \mathscr{A}$ extending a, i.e., $a^*_s(x) = a_s(x)$ for each $x \in X_s$.*

Proof. We make $\mathscr{A}$ into an $Sg(X)$-algebra $\mathscr{A}(X)$ by setting $[\![x]\!]_{\mathscr{A}(X)} = a_s(x)$ for each $x\colon s$. Therefore, by Theorem 3.1, there is a unique $Sg(X)$-homomorphism $h\colon \mathscr{T}_{Sg(X)} \to \mathscr{A}(X)$ in the category $Alg(Sg(X))$. It is easy to see that h is also a valid Sg-homormorphism from $\mathscr{T}_{Sg}(X)$ to $\mathscr{A}$ in $Alg(Sg)$ extending a:

$$\begin{aligned} h_s(x) &= h_s([\![x]\!]_{\mathscr{T}_{Sg(X)}}) && \text{☞ by Def. 3.3 on } \mathscr{T}_{Sg(X)} \\ &= [\![x]\!]_{\mathscr{A}(X)} && \text{☞ by (3.4:i)} \\ &= a_s(x) && \text{☞ by def. of } [\![x]\!]_{\mathscr{A}(X)} \end{aligned}$$

Now, suppose that $l\colon \mathscr{T}_{Sg}(X) \to \mathscr{A}$ is an Sg-homormophism extending a. Then,

$$l_s([\![x]\!]_{\mathscr{T}_{Sg}(X)}) = l_s(x) = a_s(x) = [\![x]\!]_{\mathscr{A}(X)}$$

and therefore l is also an $Sg(X)$-homomorphism $l\colon \mathscr{T}_{Sg(X)} \to \mathscr{A}(X)$ and by uniqueness of h we get $l = h$. Let $a^* = h$ and hence the thesis. □

Remark 3.4. When $X = \varnothing$, then Theorem 3.2 collapses to Theorem 3.1. ♢

From the previous theorem, it follows the next lemma which will be very useful for the proofs of the next section.

Lemma 3.1. *Let* $f\colon \mathscr{A} \to \mathscr{B}$ *be an Sg-homomorphism and* $a\colon X \to A$ *an assignment of the variables in* X*. Then,* $(f \circ a)^* = f \circ a^*$.

Proof. Both $(f \circ a)^*$ and $f \circ a^*$ are *Sg*-homomorphisms from $\mathscr{T}_{Sg}$ to $\mathscr{B}$. Moreover, for each $x \in X_s$ (for some sort $s \in S$)

$$\begin{aligned}(f \circ a)^*_s(x) &= (f \circ a)_s(x) && \text{☞ } (f \circ a)^* \text{ extends } f \circ a\\ &= f_s(a_s(x))\\ &= f_s(a^*_s(x)) && \text{☞ } a^* \text{ extends } a\\ &= (f \circ a^*)_s(x)\end{aligned}$$

therefore they both extend the same assignment function $f \circ a\colon X \to B$, hence they are unique (Theorem 3.2). □

3.6 Basic Algebraic Constructions

In this section, we provide familiar results and algebraic constructions in the order-sorted context, useful for proving important theorems hereafter in the chapter.

Proposition 3.6. *Let* $h\colon \mathscr{A} \to \mathscr{B}$ *be an Sg-homomorphism. Then,* $h\colon \mathscr{A} \to \mathscr{B}$ *is an isomorphism in Alg*(Sg) *if and only if* $h\colon A \to B$ *is a bijection (where* $h\colon A \to B$ *is the S-sorted function which underlies the homomorphism* h*).*

Proof. The "only if" part is trivial, and we only prove the converse, namely, that the inverse $h^{-1}\colon B \to A$ of a bijective homomorphism is still an *Sg*-homomorphism $h^{-1}\colon \mathscr{B} \to \mathscr{A}$. Let $k\colon s$ be a constant in *Sg*. Then,

$$h^{-1}_s([\![k]\!]_{\mathscr{B}}) = (h^{-1} \circ h)_s([\![k]\!]_{\mathscr{A}}) = [\![k]\!]_{\mathscr{A}}$$

and the proof for function symbols is similar. It only remains to prove that $h^{-1}_s(b) = h^{-1}_r(b)$ whenever $s \leqslant r$ in *Sg* and $b \in [\![s]\!]_{\mathscr{B}}$. Since h_s is a bijection by hypothesis, there is a unique $a \in [\![s]\!]_{\mathscr{A}}$ such that $h_s(a) = b$. Therefore, $h^{-1}_s(b) = h^{-1}_r(b)$ if and only if $h^{-1}_s(h_s(a)) = h^{-1}_r(h_s(a))$. But since h is a homomorphism, then by (3.4:ii) $h_s(a) = h_r(a)$ and thus $h^{-1}_s(h_s(a)) = h^{-1}_r(h_r(a))$ is trivially satisfied. □

Definition 3.8 (Closed Subset)**.** Let $\mathscr{A}$ be an *Sg*-algebra. A *closed subset* B of $\mathscr{A}$ is an S-sorted set such that

Closed subset

(i) $B \subseteq A$, that is $B_s \subseteq A_s$ with $A_s = [\![s]\!]_{\mathscr{A}}$ for each sort $s \in S$;

(ii) $[\![f\colon w \to s]\!]_{\mathscr{A}}(b_1, \ldots, b_n) \in B_s$ for each function symbol $f\colon w \to s$ with $w = s_1 \ldots s_n$ and $b_i \in B_{s_i}$ for $i = 1, \ldots, n$; and

(iii) $[\![k]\!]_{\mathscr{A}} \in B_s$ for each constant $k\colon s$. ♢

Definition 3.9 (Subalgebra)**.** Let $\mathscr{A}$ be an *Sg*-algebra. An *Sg*-algebra $\mathscr{B}$ is a *subalgebra* of $\mathscr{A}$ if

Order-sorted subalgebra

(i) $[\![s]\!]_{\mathscr{B}} \subseteq [\![s]\!]_{\mathscr{A}}$ for each $s \in S$ (or, equivalently, $B \subseteq A$);

(ii) $[\![f\colon w \to s]\!]_{\mathscr{B}}(b_1, \ldots, b_n) = [\![f\colon w \to s]\!]_{\mathscr{A}}(b_1, \ldots, b_n)$ for each function symbol $f\colon w \to s$ with $w = s_1 \ldots s_n$ and $b_i \in [\![s_i]\!]_{\mathscr{B}}$ for $i = 1, \ldots, n$; and

(iii) $[\![k]\!]_{\mathscr{B}} = [\![k]\!]_{\mathscr{A}}$ for each constant $k\colon s$. ◊

Note that a subalgebra $\mathscr{B}$ always arises from a closed subset B of $\mathscr{A}$, and, vice versa, a closed subset B can always be made into a subalgebra $\mathscr{B}$ of $\mathscr{A}$:

Proposition 3.7. *Let $\mathscr{A}$ be an order-sorted Sg-algebra. Then,*

(i) given an order-sorted Sg-subalgebra $\mathscr{B}$ of $\mathscr{A}$, the carrier set B of $\mathscr{B}$ is a closed subset of $\mathscr{A}$; and

(ii) given a closed subset B of $\mathscr{A}$, it can be made into an Sg-subalgebra $\mathscr{B}$ of $\mathscr{A}$.

Proof. We first prove (3.7:i). Condition (3.8:i) follows by (3.9:i), whereas (3.8:ii) and (3.8:iii) follow by the fact that $\mathscr{B}$ is both an order-sorted algebra and a subalgebra of $\mathscr{A}$. Vice versa, a closed subset B of $\mathscr{A}$ can be made into an *Sg*-subalgebra $\mathscr{B}$ by defining

$$[\![s]\!]_{\mathscr{B}} = B_s$$

for each $s \in S$,

$$[\![f\colon w \to s]\!]_{\mathscr{B}}(b_1, \ldots, b_n) = [\![f\colon w \to s]\!]_{\mathscr{A}}(b_1, \ldots, b_n)$$

for each function symbol $f\colon w \to s$ with $w = s_1 \ldots s_n$ and $b_i \in [\![s_i]\!]_{\mathscr{B}}$ for $i = 1, \ldots, n$, and

$$[\![k]\!]_{\mathscr{B}} = [\![k]\!]_{\mathscr{A}}$$

for each constant $k\colon s$. □

The next results show that each order-sorted *Sg*-homomorphism from an *Sg*-algebra $\mathscr{A}$ to an *Sg*-algebra $\mathscr{B}$ gives rise to an *Sg*-subalgebra of $\mathscr{B}$.

Proposition 3.8. *The image of an order-sorted Sg-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ is a closed subset of $\mathscr{B}$.*

Proof. We prove that the image of h gives rise to a closed subset of $\mathscr{B}$ denoted by H. We define $H_s = h_s([\![s]\!]_{\mathscr{A}})$. Then, given a constant $k\colon s$, we have that $[\![k]\!]_{\mathscr{B}} = h_s([\![k]\!]_{\mathscr{A}})$ by (3.4:i), and therefore $h_s([\![k]\!]_{\mathscr{A}}) \in H_s$. Finally, let $f\colon w \to s$ be a function symbol with $w = s_1 \ldots s_n$ and $x_i \in H_{s_i}$ for $i = 1, \ldots, n$. Then, there are $a_1, \ldots, a_n$ such that $h_{s_i}(a_i) = x_i$, and therefore

$$\begin{aligned} [\![f\colon w \to s]\!]_{\mathscr{B}}(x_1, \ldots, x_n) &= [\![f\colon w \to s]\!]_{\mathscr{B}}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n)) \\ &= h_s([\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \ldots, a_n)) \\ &\in H_s \end{aligned}$$

□

Corollary 3.1. *The image of an order-sorted Sg-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ is a subalgebra of $\mathscr{B}$.*

Proof. It immediately follows by Proposition 3.7. □

We denote such a subalgebra by $h(\mathscr{A})$. In the following, we say that any algebra $\mathscr{B}$ is a *homomorphic image* of an algebra $\mathscr{A}$ if there is a homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ such that $\mathscr{B}$ arises from the image of h, that is $\mathscr{B} = h(\mathscr{A})$.

Homomorphic image

A *congruence (relation)* is an equivalence relation on the carrier set of a given algebra which is compatible with its structure.

Definition 3.10 (Order-Sorted Congruence)**.** Let $\mathscr{A}$ be an *Sg*-algebra. An *order-sorted Sg-congruence* on $\mathscr{A}$ is an S-sorted set $\equiv = \{\equiv_s \mid s \in S\}$ of *equivalence relations* $\equiv_s$ on $[\![s]\!]_{\mathscr{A}}$ such that

Order-sorted congruence

(i) given $f\colon w \to s$ in *Sg* with $w = s_1 \dots s_n$ and $a_i \equiv_{s_i} a_i'$ with $a_i, a_i' \in [\![s_i]\!]_{\mathscr{A}}$ for $i = 1, \dots, n$, then

$$[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n) \equiv_s [\![f\colon w \to s]\!]_{\mathscr{A}}(a_1', \dots, a_n')$$

and

(ii) if $s \leqslant r$, then $\equiv_s \subseteq \equiv_r$ (that is, given $a, a' \in [\![s]\!]_{\mathscr{A}}$, then $a \equiv_s a'$ if and only if $a \equiv_r a'$). ♢

Definition 3.11 (Kernel)**.** Let $h\colon \mathscr{A} \to \mathscr{B}$ be an *Sg*-homomorphism. The *kernel* of h, denoted by $\ker(h)$, is the S-sorted family $\equiv_h$ of equivalence relations defined by $a\ (\equiv_h)_s\ a'$ if and only if $h_s(a) = h_s(a')$. ♢

Kernel of a homomorphism

Like subalgebras, congruences naturally arise from homomorphisms, as shown in the next proposition.

Proposition 3.9. *The kernel of an Sg-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ is a congruence on the order-sorted algebra $\mathscr{A}$.*

Proof. We need to prove that

$$[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n) \equiv_s [\![f\colon w \to s]\!]_{\mathscr{A}}(a_1', \dots, a_n')$$

given $f\colon w \to s$ in *Sg* with $w = s_1 \dots s_n$ and $a_i \equiv_{s_i} a_i'$ with $a_i, a_i' \in [\![s_i]\!]_{\mathscr{A}}$ for $i = 1, \dots, n$:

$$\begin{aligned}
&[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n) \equiv_s [\![f\colon w \to s]\!]_{\mathscr{A}}(a_1', \dots, a_n') \\
&\qquad \iff \qquad \text{by Def. 3.10} \\
&h_s([\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n)) = h_s([\![f\colon w \to s]\!]_{\mathscr{A}}(a_1', \dots, a_n')) \\
&\qquad \iff \qquad \text{by (3.4:i)} \\
&[\![f\colon w \to s]\!]_{\mathscr{A}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)) = [\![f\colon w \to s]\!]_{\mathscr{A}}(h_{s_1}(a_1'), \dots, h_{s_n}(a_n'))
\end{aligned}$$

and the last equation holds since $a_i \equiv_{s_i} a_i'$ for $i = 1, \dots, n$ by hypothesis, and therefore $h_{s_i}(a_i) = h_{s_i}(a_i')$. Finally, condition (3.10:ii) follows by h being a homomorphism (see condition (3.4:ii)). □

It is worth noting that subalgebras and congruences gives rise to complete lattices.

Lemma 3.2. *A class $\mathscr{X}$ of subsets of a set X is a complete lattice under the inclusion ordering if it is closed under arbitrary set-theoretic intersections.*

Proof. Let $\mathscr{Y}$ be a subset of $\mathscr{X}$. Then, by hypothesis, $\bigcap \mathscr{Y}$ is well-defined, belongs to $\mathscr{X}$, and is easily seen to be the greatest lower bound of $\mathscr{Y}$. On the other hand, let $\mathscr{Z}$ be the class of all subsets of X such that each $Z \in \mathscr{Z}$ (i) is also a member of $\mathscr{X}$ (that is, $\mathscr{Z} \subseteq \mathscr{X}$), and (ii) contains $\bigcup \mathscr{Y}$. Then, $\bigcap \mathscr{Z}$ is well-defined, belongs to $\mathscr{X}$, and is an upper bound of $\mathscr{Y}$. Now, suppose that $\mathscr{W}$ is an upper bound of $\mathscr{Y}$. Then $\mathscr{W}$ contains $\bigcup \mathscr{Y}$ and therefore $\mathscr{W} \in \mathscr{Z}$, hence $\bigcap \mathscr{Z} \subseteq \mathscr{W}$. □

Proposition 3.10. *Let Sg be an order-sorted signature and $\mathscr{A}$ an Sg-algebra.*

(i) The set of order-sorted Sg-subalgebras of $\mathscr{A}$ is a complete lattice under the inclusion ordering; and

(ii) The set of order-sorted Sg-congruences on $\mathscr{A}$ is a complete lattice under the inclusion ordering.

Proof. Both proofs are easy and follow by applying Lemma 3.2 after proving that arbitrary intersections of subalgebras are subalgebras and arbitrary intersections of congruences are congruences. □

The next definition will introduce a new class of order-sorted signatures for reasons that will be clarified in the next section, when we define order-sorted equations. Therefore, we state the next definition for what it is, and we demand the proper motivation to Example 3.11 and Proposition 3.13.

(Upward) filtered

Locally filtered

Coherence

Definition 3.12 (Coherence)**.** A poset $(S, \leqslant)$ is *(upward) filtered* if for any two elements $s, s' \in S$ there is $s'' \in S$ such that $s'' \geqslant s, s'$. Moreover, we say that $(S, \leqslant)$ is *locally filtered* if each of its connected component is filtered. Finally, we define an order-sorted signature *Sg* to be (i) *locally filtered* if its poset of sorts $(S, \leqslant)$ is locally filtered, and (ii) *coherent* if it is both locally filtered and regular. ♢

A *quotient algebra* is the outcome of partitioning an order-sorted algebra by a congruence relation.

Definition 3.13 (Quotient Algebra)**.** Let *Sg* be a locally filtered signature, $\mathscr{A}$ an *Sg*-algebra, and $\equiv$ a congruence on $\mathscr{A}$. For each connected component C of the poset of sorts $(S, \leqslant)$ of the signature *Sg*, let

$$A_C = \bigcup_{s \in C} [\![s]\!]_{\mathscr{A}}$$

and, given $a, a' \in A_C$, we define the equivalence relation

$$a \equiv_C a' \iff \exists s \in C \text{ such that } a \equiv_s a'$$

Natural projection

Quotient algebra

Moreover, let $q_C \colon A_C \to A_C/\equiv_C$ be the *natural projection* mapping each a into its equivalence class $[a]$. We define the quotient of the algebra $\mathscr{A}$ by the congruence $\equiv$, usually called *quotient algebra* and denoted by $\mathscr{A}/\equiv$, by setting

(i) $[\![s]\!]_{\mathscr{A}/\equiv} = q_C([\![s]\!]_{\mathscr{A}})$ for each sort $s \in S$ with $s \in C$;

(ii) $[\![f \colon w \to s]\!]_{\mathscr{A}/\equiv}([a_1], \ldots, [a_n]) = \left[[\![f \colon w \to s]\!]_{\mathscr{A}}(a_1, \ldots, a_n)\right]$ for each function symbol $f \colon w \to s$ with $w = s_1 \ldots s_n$ and $a_i \in [\![s_i]\!]_{\mathscr{A}}$ for $i = 1, \ldots, n$; and

(iii) $[\![k]\!]_{\mathscr{A}/\equiv} = \big[[\![k]\!]_{\mathscr{A}}\big]$ for each constant $k\colon s$. ♢

Remark 3.5 (Well-Definedness of Quotient Algebra). We first show that $\equiv_C$ is an equivalence relation on A_C. Reflexivity and symmetriy are trivial to prove. As regards the transitivity, suppose that $a \equiv_C a'$ and $a' \equiv_C a''$, with $a, a', a'' \in A_C$. Then, there are $s, s' \in C$ such that $a \equiv_s a'$ and $a' \equiv_{s'} a''$. Since the signature *Sg* is locally filtered, there is a sort $s'' \in C$ such that $s'' \geqslant s, s'$, and, since $\equiv$ is a congruence on $\mathscr{A}$, by condition (3.10:ii) we have that $a \equiv_{s''} a'$ and $a' \equiv_{s''} a''$. Therefore, by the transitivity of $\equiv_{s''}$, we get $a \equiv_{s''} a''$, hence $a \equiv_C a''$.

Finally, we show that (3.13:ii) is well-defined, that is the definition of the interpretation function $[\![f\colon w \to s]\!]_{\mathscr{A}/\equiv}([a_1], \dots, [a_n])$ does not depend on the choice of the representatives $a_1, \dots, a_n$ of their respective equivalence classes. First, we prove that $a \equiv_s a'$ if and only if $a \equiv_C a'$, where C is the connected component of s: The "only if" direction holds by construction; Conversely, suppose that $a \equiv_C a'$. Then, there is $s' \in C$ such that $a \equiv_{s'} a'$. But since *Sg* is locally filtered, there is also $s'' \geqslant s, s'$ such that $a \equiv_{s''} a'$ and therefore $a \equiv_s a'$ (both these facts follow by (3.10:ii)). Let $a'_i \in [\![s_i]\!]_{\mathscr{A}/\equiv}$ such that $a'_i \equiv_C a_i$ with $a'_i \in [\![s_i]\!]_{\mathscr{A}}$ for $i = 1, \dots, n$. We have to show that

$$
\begin{aligned}
&[\![f\colon w \to s]\!]_{\mathscr{A}/\equiv}([a_1, \dots, a_n]) = [\![f\colon w \to s]\!]_{\mathscr{A}/\equiv}([a'_1, \dots, a'_n]) \\
&\qquad \iff \text{⚭ by (3.13:ii)} \\
&\big[[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n)\big] = \big[[\![f\colon w \to s]\!]_{\mathscr{A}}(a'_1, \dots, a'_n)\big] \\
&\qquad \iff \text{⚭ by def. of equivalence classes} \\
&[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n) \equiv_C [\![f\colon w \to s]\!]_{\mathscr{A}}(a'_1, \dots, a'_n) \\
&\qquad \iff \text{⚭ since } a \equiv_s a' \text{ if and only if } a \equiv_C a' \text{ (see above)} \\
&[\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n) \equiv_s [\![f\colon w \to s]\!]_{\mathscr{A}}(a'_1, \dots, a'_n)
\end{aligned}
$$

and this follows by (3.10:i) since $a'_i \equiv_C a_i$ and therefore $a'_i \equiv_{s_i} a_i$. ♢

Given the quotient $\mathscr{A}/\equiv$ of the *Sg*-algebra $\mathscr{A}$ by the congruence $\equiv$, there is an obvious *Sg*-homomorphism $q\colon \mathscr{A} \to \mathscr{A}/\equiv$ defined by

$$a \in [\![s]\!]_{\mathscr{A}} \overset{q_s}{\longmapsto} [a] \in [\![s]\!]_{\mathscr{A}/\equiv}$$

We call such a homomorphism q the *quotient map* with respect to $\equiv$. Moreover, if R is an S-sorted binary relation on the carrier set A of an *Sg*-algebra $\mathscr{A}$, we denote by $\mathscr{A}/\check{R}$ the *Sg*-algebra obtained by quotienting $\mathscr{A}$ by the smallest congruence on $\mathscr{A}$ containing R, that is

Quotient map

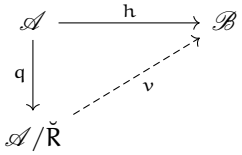
$$\check{R} = \bigcap \{\, R' \in \mathrm{Cong}(\mathscr{A}) \mid R \subseteq R' \,\}$$

where $\mathrm{Cong}(\mathscr{A})$ is the complete lattice of congruences on $\mathscr{A}$ (see Proposition 3.10).

Proposition 3.11 (Universal Property of Quotient). *Let $\mathscr{A}$ be an Sg-algebra with Sg a locally filtered signature. Given an S-sorted binary relation R on the carrier set A of $\mathscr{A}$, then the quotient map $q\colon \mathscr{A} \to \mathscr{A}/\check{R}$ satisfies*

(i) $R \subseteq \ker(q)$; *and*

(ii) given any Sg-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ *such that* $R \subseteq \ker(h)$*, then there is a unique Sg-homomorphism* $v\colon \mathscr{A}/\check{R} \to \mathscr{B}$ *which factorises* h *through* q*, that is* $v \circ q = h$*, or, diagrammatically,*



Proof. Since $\ker(q) = \check{R}$ and $\check{R}$ is the smallest congruence relation containing R by definition, then $R \subseteq \ker(q)$. Now, let $h\colon \mathscr{A} \to \mathscr{B}$ be an *Sg*-homomorphism such that $R \subseteq \ker(h)$, and recall that $[\![s]\!]_{\mathscr{A}/\check{R}} = \{\, [a] \in A_C/\check{R}_C \mid a \in [\![s]\!]_{\mathscr{A}} \,\}$, where C is the connected component of s (see both Definition 3.13 and Remark 3.5). There is only a unique S-sorted function $v\colon A/\check{R} \to B$ between the carrier sets of $\mathscr{A}/\check{R}$ and $\mathscr{B}$ such that $v \circ q = h$ defined by $v_s([a]) = h_s(a)$. Note that v is well-defined: Suppose that $a\ \check{R}_C\ a'$ for some $a, a' \in [\![s]\!]_{\mathscr{A}}$. Therefore, since $a\ \check{R}_C\ a'$ if and only if $a\ \check{R}_s\ a'$, we have that

$$v_s([a]) = v_s([a']) \iff h_s(a) = h_s(a')$$

and since $a\ \check{R}_C\ a'$ and $\check{R} \subseteq \ker(h)$, then $a\ (\equiv_h)_s\ a'$ and therefore $h_s(a) = h_s(a')$.

We prove that v is also an *Sg*-homomorphism. Let $k\colon s$ be a constant in *Sg*. Then,

$$\begin{aligned} v_s([\![k]\!]_{\mathscr{A}/\check{R}}) &= v_s\big([\![k]\!]_{\mathscr{A}}\big) && \text{by (3.13:iii)} \\ &= h_s([\![k]\!]_{\mathscr{A}}) && \text{by def. of } v \\ &= [\![k]\!]_{\mathscr{B}} && h \text{ is an } Sg\text{-hom.} \end{aligned}$$

Let $f\colon w \to s$ be a function symbol in *Sg* with $w = s_1 \dots s_n$ and $a_i \in [\![s_i]\!]_{\mathscr{A}}$ for $i = 1, \dots, n$. Then,

$$\begin{aligned} &v_s\big([\![f\colon w \to s]\!]_{\mathscr{A}/\check{R}}([a_1], \dots, [a_n])\big) \\ &\quad = v_s\big([\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n)]\big) && \text{by (3.13:ii)} \\ &\quad = h_s([\![f\colon w \to s]\!]_{\mathscr{A}}(a_1, \dots, a_n)) && \text{by def. of } v \\ &\quad = [\![f\colon w \to s]\!]_{\mathscr{B}}(h_{s_1}(a_1), \dots, h_{s_n}(a_n)) && h \text{ is an } Sg\text{-hom.} \\ &\quad = [\![f\colon w \to s]\!]_{\mathscr{B}}(v_{s_1}([a_1]), \dots, v_{s_n}([a_n])) && \text{by def. of } v \end{aligned}$$

Finally, if $s \leqslant r$ in *Sg* and $a \in [\![s]\!]_{\mathscr{A}}$, then

$$v_s([a]) = h_s(a) = h_r(a) = v_r([a])$$

and hence the thesis. □

Example 3.8. Various presentations of lambda-calculus provide semantics to λ-terms "up to α-equivalence", meaning that λ-terms are identified by equivalence classes of terms which only differ by a renaming of bound variables. In this example, we apply the universal property of quotient to formalise this practice.

First, recall the signature $Sg(\lambda)$ of the untyped lambda-calculus (Example 3.2). Let t be a λ-term, that is $t \in [\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}}$ or, equivalently, $t\colon exp$. The *free variables* $\mathrm{FV}(t)$

Free variables

of a term t are defined as all the variables appearing in t which are not bound to an abstraction operator:

$$\mathrm{FV}\colon [\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}} \to \wp(Var)$$

$$\begin{aligned}\mathrm{FV}(x) &= \{x\}\\ \mathrm{FV}(\lambda_x(t)) &= \mathrm{FV}(t) \setminus \{x\}\\ \mathrm{FV}(\circ(t_1,t_2)) &= \mathrm{FV}(t_1) \cup \mathrm{FV}(t_2)\end{aligned}$$

Substitution

Then, we define the *substitution* of t' for free occurrences of x in t, denoted by $t[t'/x]$, by structural induction on t. Please note that the following definition is not a function, but merely a total relation. We shall write $t[t'/x] \propto t''$ to indicate that t'' is one among the many outcomes of the substitution $t[t'/x]$:

$$\propto \subseteq \left([\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}} \times [\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}} \times Var\right) \times [\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}}$$

$$x[t/y] \propto \begin{cases} t & \text{if } x = y\\ x & \text{otherwise}\end{cases}$$

$$\lambda_x(t)[t'/y] \propto \begin{cases} \lambda_x(t) & \text{if } x = y\\ \lambda_x(u) & \text{if } x \notin \mathrm{FV}(t') \text{ where } t[t'/y] \propto u\\ \lambda_z(u) & \text{with } z \neq x \text{ and } z \notin \mathrm{FV}(t) \cup \mathrm{FV}(t'), \text{ and where } t[z/x] \propto u' \text{ and } u'[t'/y] \propto u\end{cases}$$

$$\circ(t_1,t_2)[t/x] \propto \circ(u_1,u_2) \quad \text{where } t_1[t/x] \propto u_1 \text{ and } t_2[t/x] \propto u_2$$

α-equivalence

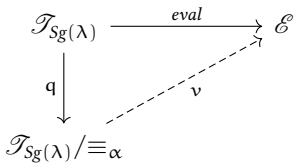
Now, we define *α-equivalence* as a congruence relation $\equiv_\alpha$ on $\mathscr{T}_{Sg(\lambda)}$ according to the following axioms (we write $\equiv_\alpha$ instead of $(\equiv_\alpha)_{exp}$ since there is just a single sort in the signature $Sg(\lambda)$):

$$(\mathrm{Ref}_\alpha)\ \frac{-}{t \equiv_\alpha t} \qquad (\mathrm{Sym}_\alpha)\ \frac{t \equiv_\alpha u}{u \equiv_\alpha t} \qquad (\mathrm{Trans}_\alpha)\ \frac{t \equiv_\alpha u \quad u \equiv_\alpha v}{t \equiv_\alpha v}$$

$$(\mathrm{Abs}_\alpha)\ \frac{t \equiv_\alpha u}{\lambda_x(t) \equiv_\alpha \lambda_x(u)}\ (x \in Var) \qquad (\mathrm{App}_\alpha)\ \frac{t_1 \equiv_\alpha t_2 \quad u_1 \equiv_\alpha u_2}{\circ(t_1,t_2) \equiv_\alpha \circ(u_1,u_2)}$$

$$(\alpha)\ \frac{-}{\lambda_x(t) \equiv_\alpha \lambda_y(t[y/x])}\ x \in Var \text{ and } y \notin \mathrm{FV}(t)$$

It is easy to prove that $\equiv_\alpha$ is an equivalence relation on $[\![exp]\!]_{\mathscr{T}_{Sg(\lambda)}}$ by Rules (Ref_α), (Sym_α), and (Trans_α) and a congruence by Rules (Abs_α) and (App_α). Therefore, we can quotient $\mathscr{T}_{Sg(\lambda)}$ by $\equiv_\alpha$ and then apply the universal property of quotient to get



where $eval\colon \mathscr{T}_{Sg(\lambda)} \to \mathscr{E}$ is the unique $Sg(\lambda)$-homomorphism defined in Example 3.6. Note that the commutativity property of the previous diagram ensures that the semantics of λ-terms is well-defined up to α-equivalence. ♢

Theorem 3.3 (First Homomorphism Theorem)**.** *Let Sg be a locally filtered order-sorted signature and* $h\colon \mathscr{A} \to \mathscr{B}$ *an Sg-homomorphism. Then,* $\mathscr{A}/\ker(h) \cong h(\mathscr{A})$.

Proof. Let $h'\colon \mathscr{A} \to h(\mathscr{A})$ be the corestriction of h. Then, by Proposition 3.11, there is a unique *Sg*-homomorphism $\nu\colon \mathscr{A}/\ker(h) \to h(\mathscr{A})$ such that $\nu \circ q = h'$. Moreover, ν is surjective because so is h, and given $a, a' \in [\![s]\!]_{\mathscr{A}}$, then

$$\nu_s([a]) = \nu_s([a']) \implies h_s(a) = h_s(a') \implies a\ (\equiv_h)_s\ a'$$

and therefore $[a] = [a']$, hence the injectivity of the homomorphism ν. The thesis now follows by Proposition 3.6. □

3.7 Order-Sorted Equations

Equational logic is nothing more than a collection of rules for manipulating equations. An *order-sorted equation* is simply an assertion of equality between two terms (with variables). In particular, the use of terms with variables allows a single equation to assert a (potentially infinite) class of term equalities. For instance, consider the signature *Sg*(*Num*) in Example 3.1. We might want to assert that the predecessor of the successor of each natural number is the natural number itself. If n is a variable of sort *nat*, we can write the single equation

$$(\forall n\colon nat)\ \ p(s(n)) = n$$

to state that such an equation holds for each term of sort *nat* that can be substituted into the variable n.

In the classical presentation of *many-sorted logics* [CG06], one requires terms appearing in an equation to have the same sort. However, in the order-sorted case we can retain much more flexibility with sorts. Indeed, since subsort relations are encoded in the term algebra by subset inclusion, we can be more general and allow equations with terms of "comparable sorts", that is terms whose *least sorts* resides in the same connected component.

Notation 3.3. In the rest of the chapter, we will adopt a lighter notation when applying homomorphisms to terms: Let *Sg* be a regular order-sorted signature and $a\colon \mathscr{T}_{Sg} \to \mathscr{A}$ an *Sg*-homomorphsim. We will write $a(t)$ instead of $a_{ls(t)}(t)$ for each term t built out of *Sg*. ♢

Order-sorted equation

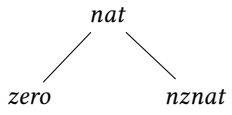
Definition 3.14 (Order-Sorted Equation)**.** Let *Sg* be a regular order-sorted signature. An *(unconditional) order-sorted Sg-equation* $(\forall X)\ \ t_1 = t_2$ is given by an S-sorted set of variables X and two terms t_1 and t_2 in $\mathscr{T}_{Sg}(X)$ such that their least sorts $ls(t_1)$ and $ls(t_2)$ belong to the same connected component of $(S, \leqslant)$. ♢

For the sake of simplicity, when X is finite we might write

$$(\forall x_1\colon s_1, \dots, x_n\colon s_n)\ \ t_1 = t_2$$

in place of $(\forall X)\ \ t_1 = t_2$, where $x_i \in X_{s_i}$ for each $i = 1, \dots, n$. Moreover, if $X = \varnothing$ (and therefore t_1 and t_2 are ground terms), we just write $t_1 = t_2$.

Example 3.9. Let *Sg*(*Nat*) be the order-sorted signature obtained by restricting the signature *Sg*(*Num*) in Example 3.1 to sorts $S(Nat) = \{nat, nznat, zero\}$ and operators



$0\colon \mathit{zero}$	zero
$s\colon \mathit{nat} \to \mathit{nznat}$	successor
$+\colon \mathit{nat}, \mathit{nat} \to \mathit{nat}$	addition

We can axiomatise the behaviour of the addition operator by adding the following order-sorted equations:

$$\begin{aligned}
&(\forall n\colon \mathit{nat}) && +(0, n) = n \\
&(\forall n\colon \mathit{nat}, m\colon \mathit{nat}) && +(s(n), m) = s(+(n, m)) \\
&(\forall n\colon \mathit{nat}, m\colon \mathit{nat}) && +(n, m) = +(m, n)
\end{aligned}$$

which assert the additive properties of addition over natural numbers. ♢

Definition 3.15 (Order-Sorted Satisfaction)**.** An order-sorted equation $(\forall X)\ t_1 = t_2$ over a regular signature Sg is *satisfied* by an Sg-algebra $\mathscr{A}$ if and only if $a^*(t_1) = a^*(t_2)$ for each assignment $a\colon X \to A$, where $a^*\colon \mathscr{T}_{Sg}(X) \to \mathscr{A}$ is the unique homomorphism extending a (Theorem 3.2). ♢

Example 3.10. Consider the signature $Sg(Nat)$ in Example 3.9 and let $\mathscr{N}$ be the $Sg(Nat)$-algebra obtained by restricting the $Sg(Num)$-algebra $\mathscr{A}$ of Example 3.4 only to operators and sorts of $Sg(Nat)$, that is interpretation sets $[\![\mathit{nat}]\!]_{\mathscr{A}} = \mathbb{N}$, $[\![\mathit{zero}]\!]_{\mathscr{A}} = \{0\}$, and $[\![\mathit{nznat}]\!]_{\mathscr{A}} = \mathbb{N} \setminus \{0\}$, and interpretation functions

$$\begin{aligned}
[\![0]\!]_{\mathscr{N}} &= 0 \\
[\![s\colon \mathit{nat} \to \mathit{nznat}]\!]_{\mathscr{N}}(n) &= n + 1 \\
[\![+\colon \mathit{nat}, \mathit{nat} \to \mathit{nat}]\!]_{\mathscr{N}}(n_1, n_2) &= n_1 + n_2
\end{aligned}$$

where $n \in \mathbb{N}$. It is easy to see that the algebra $\mathscr{N}$ satisfies all the $Sg(Nat)$-equations provided in Example 3.9. ♢

Definition 3.16 (Conditional Equation and Satisfaction)**.** Let Sg be a regular order-sorted signature. An *order-sorted conditional Sg-equation* $(\forall X)\ t = t' \Leftarrow C$ is given by an unconditional Sg-equation $(\forall X)\ t = t'$ and a *finite* set C of unconditional Sg-equations involving only variables in X, that is

Order-sorted conditional equation

$$C = \{(\forall X)\ t_1 = t_1', \ldots, (\forall X)\ t_n = t_n'\}$$

for some $n \in \mathbb{N}$.

An Sg-algebra $\mathscr{A}$ satisfies a conditional Sg-equation $(\forall X)\ t = t' \Leftarrow C$ if and only if for each assignment $a\colon X \to A$ satisfying each equation in C, it also satisfies $(\forall X)\ t = t'$, that is

$$(\forall 1 \leqslant i \leqslant n)\ a^*(t_i) = a^*(t_i') \implies a^*(t) = a^*(t')$$

where $C = \{(\forall X)\ t_1 = t_1', \ldots, (\forall X)\ t_n = t_n'\}$. ♢

Since C is always finite and equations in C involve only variables in X, we usually denote conditional equations by writing

$$(\forall X)\ t = t' \Leftarrow \left(t_1 = t_2 \wedge \cdots \wedge t_n = t_n'\right)$$

or, when t_i and t'_i are clear from the context,

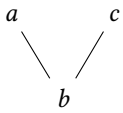
$$(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t'_i$$

Note that conditional equations proper generalises ordinary equations, in the following sense:

Proposition 3.12. *An Sg-equation* $(\forall X)\ \ t = t'$ *is satisfied by an Sg-algebra* $\mathscr{A}$ *if and only if* $\mathscr{A}$ *satisfies* $(\forall X)\ \ t = t' \Leftarrow \varnothing$.

With all these elements put together, we can provide a suitable justification for the notion of *coherence* introduced in Definition 3.12. Consider the following example taken from [GM92]:

Example 3.11. Let $Sg(\mathbb{C})$ be the following, *non-coherent*, order-sorted signature with poset of sorts $S(\mathbb{C})$ given by the Hasse diagram on the left and operators on the right:



$k\colon a$	a constant of sort a
$j\colon b$	a constant of sort b
$l\colon c$	a constant of sort c

Consider now the equation $k = l$ involving no variables. The term algebra $\mathscr{T}_{Sg(\mathbb{C})}$ surely does not satisfy such an equation. On the other hand, the $Sg(\mathbb{C})$-algebra $\mathscr{A}$ given by $[\![a]\!]_{\mathscr{A}} = [\![c]\!]_{\mathscr{A}} = \{0, 1\}$ and $[\![b]\!]_{\mathscr{A}} = \{0\}$ and interpretation functions $[\![k]\!]_{\mathscr{A}} = [\![l]\!]_{\mathscr{A}} = 1$ and $[\![j]\!]_{\mathscr{A}} = 0$ does satisfy $k = l$. Unfortunately, the unique $Sg(\mathbb{C})$-homormophism $h\colon \mathscr{T}_{Sg(\mathbb{C})} \to \mathscr{A}$ is actually an isomorphism. Therefore, under the present conditions, the notion of satisfaction is not closed under isomorphism. ♢

The issue with the previous example boils down to the absence of a common supersort of a and c. Indeed, if we just add a new sort $d > a, c$ to the signature, then the interpretation domains $[\![d]\!]_{\mathscr{T}_{Sg(\mathbb{C})}} = \{k, j, l\}$ and $[\![d]\!]_{\mathscr{A}} = \{0, 1\}$ preclude h to be an isomorphism. Note that this is exactly what the definition of coherence enforces, and allows us to prove the following:

Proposition 3.13. *Let Sg be a coherent order-sorted signature,* $\mathscr{A}$ *and* $\mathscr{B}$ *two isomorphic Sg-algebras, and* $(\forall X)\ \ t = t' \Leftarrow C$ *a conditional Sg-equation. Then,* $\mathscr{A}$ *satisfies* $(\forall X)\ \ t = t' \Leftarrow C$ *if and only if* $\mathscr{B}$ *does.*

Proof. ($\Longrightarrow$) Suppose that $\mathscr{A}$ satisfies $(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t'_i$. We want to prove that for each assignment $b\colon X \to B$, then

$$(\forall 1 \leqslant i \leqslant n)\ \ b^*(t_i) = b^*(t'_i) \implies b^*(t) = b^*(t')$$

Let $f\colon \mathscr{A} \to \mathscr{B}$ be the *Sg*-isomorphism between these algebras. Then, each assignment $b\colon X \to B$ can be factorised into $b = f \circ a$ with $a\colon X \to A$ defined by $a_s(x) = f_s^{-1}(b_s(x))$ for each variable $x \in X_s$. Now, let $s \geqslant ls(t), ls(t')$. Then,

$$\begin{aligned} b^*_s(t) &= f_s(a^*_s(t)) && \text{by Lem. 3.1} \\ &= f_s(a^*_s(t')) && \mathscr{A} \text{ satisfies } (\forall X)\ \ t = t' \Leftarrow C \\ &= b^*_s(t') && \text{by Lem. 3.1} \end{aligned}$$

hence, by (3.4:ii) (and Remark 3.3), $b^*(t) = b^*_s(t) = b^*_s(t') = b^*(t')$. ($\Longleftarrow$) It follows from the previous case: Isomorphism relation is symmetric. □

3.8 Order-Sorted Deduction

We shall now present the rules of *equational logic* to derive new equations starting from a signature and a given set of equations. In particular, we employ the following terminology: If $Ax(X)$ is a set of Sg-equations over the same set of variables X, henceforth referred to as *axioms*, we call

Axioms

$$Th = (Sg, Ax(X))$$

a *theory*. Moreover, we say that an Sg-algebra $\mathscr{A}$ is a *model* of a theory $Th = (Sg, Ax(X))$ if and only if $\mathscr{A}$ satisfies each equation in $Ax(X)$. We often abbreviate this fact by saying that $\mathscr{A}$ is a Th-model.

Theory, Model of a theory

Notation 3.4. From now on, when we refer to a theory Th we shall leave implicit that $Th = (Sg, Ax(X))$, unless stated otherwise. ♢

Proposition 3.14. *Let Th be an order-sorted theory. The class of all models of Th and the class of all Sg-homomorphisms between these models form a category Mod(Th).*

Category of models over *Th*

As we shall see, equational logic allows deducing *theorems* from a theory, that is new equations satisfied by any Th-model. We refer to this property as *soundness* of order-sorted deduction. Moreover, we shall prove that such a derivation process is *complete*, meaning that any equation satisfied by every model of the theory is also deducible from Th. In the following, we call any assignment of the form $\theta\colon X \to T_{Sg}(Y)$ a *substitution* (of terms in $\mathscr{T}_{Sg}(Y)$ for variables in X).

Theorem

Substitution

The formal rules of equational logic are given in Figure 3.4. The judgement of this deductive system are order-sorted unconditional equations (theorems) over the signature Sg of an order-sorted theory Th:

- Rule (Ax) states that every *unconditional* axiom in $Ax(X)$ is deducible from Th.
- Rules (Ref), (Sym), and (Trans) are self-explanatory.
- Rule (Cong) says that given any two substitutions θ and ϕ of terms in the free algebra $\mathscr{T}_{Sg}(Y)$ for variables in X, if every equation $(\forall Y)\ \theta_s(x) = \phi_s(x)$ is derivable from Th (for each $x \in X_s$ of some sort $s \in S$), then the substitution of variables in t via θ or ϕ produces two derivably equal terms $\theta^*(t)$ and $\phi^*(t)$ with variables in Y.
- Finally, Rule (Sub) allows deducing *unconditional equations* from *both conditional and unconditional axioms*. Let $(\forall X)\ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i'$ be a conditional axiom (if $n = 0$, then it can be regarded as unconditional) and $\theta\colon X \to T_{Sg}(Y)$ a substitution. If θ equalises each premise of such an equation, that is $(\forall Y)\ \theta^*(t_i) = \theta^*(t_i')$ is deducible from Th for each $i = 1, \dots, n$, then so is the unconditional equation $(\forall Y)\ \theta^*(t) = \theta^*(t')$.

The reader may notice that the usual rule of *generalised substitution* is admissible by a routine rule induction:

Lemma 3.3 (Generalised Substitution)**.** *The following rule is admissible:*

Generalised Substitution

$$\text{(GSub)}\ \frac{(\forall X)\ t = t'}{(\forall Y)\ \theta^*(t) = \theta^*(t')}\ \theta\colon X \to T_{Sg}(Y)$$

$$\text{(Ax)}\ \frac{(\forall X)\ \ t = t' \in Ax(X)}{(\forall X)\ \ t = t'} \qquad \text{(Ref)}\ \frac{t\colon s}{(\forall X)\ \ t = t} \qquad \text{(Sym)}\ \frac{(\forall X)\ \ t' = t}{(\forall X)\ \ t = t'}$$

$$\text{(Trans)}\ \frac{(\forall X)\ \ t = t' \qquad (\forall X)\ \ t' = t''}{(\forall X)\ \ t = t''}$$

$$\text{(Cong)}\ \frac{\begin{array}{c}\theta, \phi\colon X \to T_{Sg}(Y)\\ (\forall s \in S, \forall x \in X_s)\ \ (\forall Y)\ \ \theta_s(x) = \phi_s(x)\end{array}}{(\forall Y)\ \ \theta^*(t) = \phi^*(t)}$$

$$\text{(Sub)}\ \frac{\begin{array}{c}\theta\colon X \to T_{Sg}(Y)\\ (\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i' \in Ax(X)\\ (\forall 1 \leqslant i \leqslant n)\ \ (\forall Y)\ \ \theta^*(t_i) = \theta^*(t_i')\end{array}}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')}$$

Figure 3.4: Theorems deduced from an order-sorted theory $Th = (Sg, Ax(X))$.

Proof. The proof is by rule induction on the derivation of the premise $(\forall X)\ \ t = t'$:

1. (Ax) Suppose the premise has been derived by

   $$\text{(Ax)}\ \frac{(\forall X)\ \ t = t' \in Ax(X)}{(\forall X)\ \ t = t'}$$

   Therefore, we can apply Rule (Sub) to such an unconditional axiom and get

   $$\text{(Sub)}\ \frac{(\forall X)\ \ t = t' \Leftarrow \varnothing \in Ax(X)}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')}\ \theta\colon X \to T_{Sg}(Y)$$

2. (Ref) Suppose the premise has been derived by

   $$\text{(Ref)}\ \frac{-}{(\forall X)\ \ t = t}\ t\colon s$$

   Then, by the following derivation tree we can prove the conclusion:

   $$\text{(Cong)}\ \frac{(\forall s \in S, \forall x \in X_s)\ \ \text{(Ref)}\ \dfrac{-}{(\forall Y)\ \ \theta_s(x) = \theta_s(x)}\ \theta_s(x)\colon s}{(\forall Y)\ \ \theta^*(t) = \theta^*(t)}$$

   where $\theta\colon X \to T_{Sg}(Y)$.

3. (Sym) Suppose the premise has been derived by

   $$\text{(Sym)}\ \frac{(\forall X)\ \ t' = t}{(\forall X)\ \ t = t'}$$

   By induction hypothesis, we have that

   $$\text{(GSub)}\ \frac{(\forall X)\ \ t' = t}{(\forall Y)\ \ \theta^*(t') = \theta^*(t)}\ \theta\colon X \to T_{Sg}(Y)$$

   and hence the conclusion by a simple application of Rule (Sym).

4. (Trans) Suppose the premise has been derived by

$$\text{(Trans)}\ \frac{(\forall X)\ \ t = t' \qquad (\forall X)\ \ t' = t''}{(\forall X)\ \ t = t''}$$

Then, by induction hypothesis,

$$\text{(GSub)}\ \frac{(\forall X)\ \ t = t'}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')} \qquad \text{(GSub)}\ \frac{(\forall X)\ \ t' = t''}{(\forall Y)\ \ \theta^*(t') = \theta^*(t'')}$$

with $\theta\colon X \to T_{Sg}(Y)$, and therefore $(\forall Y)\ \ \theta^*(t) = \theta^*(t'')$ by Rule (Trans).

5. (Cong) Suppose the premise has been derived by

$$\text{(Cong)}\ \frac{(\forall s \in S, \forall x \in X_s)\ \ (\forall Y)\ \ \theta_s(x) = \phi_s(x)}{(\forall Y)\ \ \theta^*(t) = \phi^*(t)}\ \theta, \phi\colon X \to T_{Sg}(Y)$$

Then, by induction hypothesis, for each $s \in S$ and $x \in X_s$ we have

$$\text{(GSub)}\ \frac{(\forall Y)\ \ \theta_s(x) = \phi_s(x)}{(\forall Y)\ \ \psi^*(\theta_s(x)) = \psi^*(\phi_s(x))}\ \psi\colon Y \to T_{Sg}(Z)$$

and therefore

$$\text{(Cong)}\ \frac{(\forall s \in S, \forall x \in X_s)\ \ (\forall Z)\ \ \psi^*(\theta_s(x)) = \psi^*(\phi_s(x))}{(\forall Z)\ \ \psi^*(\theta^*(t)) = \psi^*(\phi^*(t))}$$

where $\psi^* \circ \theta, \psi^* \circ \phi\colon X \to T_{Sg}(Z)$.

6. (Sub) Suppose the premise has been derived by

$$\text{(Sub)}\ \frac{\begin{array}{c}(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i' \in Ax(X) \\ (\forall 1 \leqslant i \leqslant n)\ \ (\forall Y)\ \ \theta^*(t_i) = \theta^*(t_i')\end{array}}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')}\ \theta\colon X \to T_{Sg}(Y)$$

Then, by induction hypothesis,

$$\text{(GSub)}\ \frac{(\forall Y)\ \ \theta^*(t_i) = \theta^*(t_i')}{(\forall Z)\ \ \psi^*(\theta^*(t_i)) = \psi^*(\theta^*(t_i'))}\ \psi\colon Y \to T_{Sg}(Z)$$

for each $i = 1, \ldots, n$. Therefore, we can apply Rule (Sub) and get

$$\text{(Sub)}\ \frac{\begin{array}{c}(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i' \in Ax(X) \\ (\forall 1 \leqslant i \leqslant n)\ \ (\forall Z)\ \ \psi^*(\theta^*(t_i)) = \psi^*(\theta^*(t_i'))\end{array}}{(\forall Z)\ \ \psi^*(\theta^*(t)) = \psi^*(\theta^*(t'))}\ \psi^* \circ \theta\colon X \to T_{Sg}(Z)$$

hence the conclusion. □

We can now prove the soundness of the deduction system, namely that any unconditional equation derivable from a theory *Th* is satisfied by all the *Th*-model.

Theorem 3.4 (Soundness). *Let* $Th = (Sg, Ax(X))$ *be a theory over the coherent signature* Sg *and* $\mathscr{A}$ *an order-sorted* Th*-model. If* $(\forall Y)\ \ t = t'$ *is derivable from* Th *using rules in Figure 3.4, then* $\mathscr{A}$ *satisfies* $(\forall Y)\ \ t = t'$.

Proof. The proof is by rule induction on $(\forall Y)\ \ t = t'$. Most of the cases are easy to prove and simply follow by applying the definition of satisfaction. We only prove the case of Rule (Cong) which requires some calculation: Suppose the equation under discussion has been derived by

$$\text{(Cong)}\ \frac{(\forall s \in S, \forall x \in X_s)\ \ (\forall Y)\ \ \theta_s(x) = \phi_s(x)}{(\forall Y)\ \ \theta^*(t) = \phi^*(t)}\ \theta, \phi \colon X \to T_{Sg}(Y)$$

We have to prove that for each assignment $a \colon Y \to A$ (with A the carrier set of the algebra $\mathscr{A}$)

$$\begin{aligned} & a^*(\theta^*(t)) = a^*(\phi^*(t)) \\ \iff \quad & (a^* \circ \theta)^*(t) = (a^* \circ \phi)^*(t) \qquad \text{☞ by Lemma 3.1} \end{aligned}$$

We shall show that $a^* \circ \theta$ and $a^* \circ \phi$ are the same assignment. By induction hypothesis, the equation $(\forall Y)\ \ \theta_s(x) = \phi_s(x)$ is satisfied by $\mathscr{A}$, and therefore $(a^* \circ \theta)_s(x) = (a^* \circ \phi)_s(x)$ for each $s \in S$ and $x \in X_s$. Since by Theorem 3.2 there is a unique Sg-homomorphism extending $a^* \circ \theta = a^* \circ \phi$, then $(a^* \circ \theta)^* = (a^* \circ \phi)^*$ and hence $a^*(\theta^*(t)) = a^*(\phi^*(t))$. □

Proving the completeness of the order-sorted equational logic requires more mathematical tools and lemmas. The method is standard from the early days of universal algebra [Bir35] and requires the construction of a new algebra in which elements in the carrier set are equivalence classes of provably equal terms.

We start by defining a congruence relation on terms: Let $Th = (Sg, Ax(X))$ be an order-sorted theory, and let t and t' be two terms with variables in the free algebra $\mathscr{T}_{Sg}(Y)$. We shall write

$$t \sim_s t' \text{ if and only if } (\forall Y)\ \ t = t' \text{ is derivable from } Th \tag{3.5}$$

that is t and t' are related just in case they are provably equal.

Lemma 3.4. *Let Th be an order-sorted theory. Then, the binary relation $\sim$ is a congruence on the Sg-algebra $\mathscr{T}_{Sg}(Y)$.*

Proof. The binary relation $\sim$ is trivially an equivalence relation by Rules (Ref), (Sym), and (Trans). We now prove condition (3.10:i): Let $f \colon w \to s$ with $w = s_1 \dots s_n$ be a function symbol in Sg and $t_i \sim_{s_i} t_i'$ with t_i, t_i' terms of sort s_i in $\mathscr{T}_{Sg}(Y)$, for $i = 1, \dots, n$. We want to prove that

$$\begin{aligned} & [\![f \colon w \to s]\!]_{\mathscr{T}_{Sg}(Y)}(t_1, \dots, t_n) \sim_s [\![f \colon w \to s]\!]_{\mathscr{T}_{Sg}(Y)}(t_1', \dots, t_n') \\ & \qquad \iff \\ & f(t_1, \dots, t_n) \sim_s f(t_1', \dots, t_n') \\ & \qquad \iff \\ & (\forall Y)\ \ f(t_1, \dots, t_n) = f(t_1', \dots, t_n') \text{ is derivable from } Th \end{aligned}$$

Let Z be the set of variables defined by $Z_{s_i} = \{z_i\}$ for each s_i appearing in w and $Z_{s'} = \varnothing$ for any other sort s'. We define two substitutions $\theta, \phi \colon Z \to T_{Sg}(Y)$ by sending $z_i \mapsto t_i$ and $z_i \mapsto t_i'$, respectively. Since $t_i \sim_{s_i} t_i'$ by hypothesis, then

$(\forall Y)\ \theta_{s_i}(z_i) = \phi_{s_i}(z_i)$ is derivable (by definition of $\sim_{s_i}$). Therefore, we can apply Rule (Cong) to the term $f(z_1, \ldots, z_n)$ in $\mathscr{T}_{Sg}(Z)$, and we have that

$$\text{(Cong)}\ \frac{(\forall z_i \in \bigcup Z)\ (\forall Y)\ \theta_{s_i}(z_i) = \phi_{s_i}(z_i)}{(\forall Y)\ \theta^*(f(z_1, \ldots, z_n)) = \phi^*(f(z_1, \ldots, z_n))}\ \theta, \phi \colon Z \to T_{Sg}(Y)$$

which yields exactly $(\forall Y)\ f(t_1, \ldots, t_n) = f(t'_1, \ldots, t'_n)$. Finally, condition (3.10:i) holds because the rules of equational logic do not depend on the sort at which terms are considered. □

This lemma enables the quotienting of the free algebra $\mathscr{T}_{Sg}(Y)$ by the congruence $\sim$ which, by construction, has members the equivalence classes of provably equal terms. For the sake of clarity, we denote such a quotient algebra $\mathscr{T}_{Sg}(Y)/\sim$ by $\mathscr{T}_{Th}(Y)$.

Proposition 3.15. *Let $Th = (Sg, Ax(X))$ be an order-sorted theory and $(\forall Y)\ t = t'$ an unconditional Sg-equation. Then,*

(i) $[t] = [t']$ if and only if $(\forall Y)\ t = t'$ is derivable from Th.

(ii) If $(\forall Y)\ t = t'$ is satisfied by $\mathscr{T}_{Th}(Y)$, then $(\forall Y)\ t = t'$ is derivable from Th.

Proof. The first point trivially follows by the definition of $\sim$. With regard to the second, consider the assignment $q_Y \colon Y \to T_{Th}(Y)$ defined by $y \mapsto [y]$. Since $\mathscr{T}_{Th}(Y)$ satisfies $(\forall Y)\ t = t'$, then $q_Y^*(t) = q_Y^*(t')$. However, as discussed before Proposition 3.11, the quotient map $q \colon \mathscr{T}_{Sg}(Y) \to \mathscr{T}_{Th}(Y)$ embedding each term t into its equivalence class $[t]$ is an Sg-homomorphism extending q_Y, and by uniqueness we conclude that $q_Y^* = q$. Therefore, $q_Y^*(t) = q_Y^*(t')$ implies that $[t] = [t']$, and hence $(\forall Y)\ t = t'$ is derivable from Th. □

Proposition 3.16. *$\mathscr{T}_{Th}(Y)$ is a model of the order-sorted theory $Th = (Sg, Ax(X))$.*

Proof. Let $(\forall X)\ t = t' \Leftarrow \bigwedge_{i=1}^n t_i = t'_i$ be an axiom in $Ax(X)$ and $\theta \colon X \to T_{Th}(Y)$ an assignment satisfying each premise, that is

$$(\forall 1 \leqslant i \leqslant n)\ \theta^*(t_i) = \theta^*(t'_i)$$

We have to prove that $\theta^*(t) = \theta^*(t')$. For each $x \in X_s$ (for some $s \in S$), let t_x be any arbitrary representative of the equivalence class $\theta_s(x)$, namely $\theta_s(x) = [t_x]$, and consider the new assignment function $\phi \colon X \to \mathscr{T}_{Sg}(Y)$ defined by $\phi_s(x) = [t_x]$. Therefore, $\theta_s(x) = [\phi_s(x)]$ and thus $\theta = q \circ \phi$, where $q \colon \mathscr{T}_{Sg}(Y) \to \mathscr{T}_{Th}(Y)$ is the quotient map (and an Sg-homomorphism too). By Lemma 3.1, we have that

$$\theta^* = (q \circ \phi)^* = q \circ \phi^*$$

and hence $\theta^*(t) = [\phi^*(t)]$ for each term t in $\mathscr{T}_{Sg}(X)$. Then, $\theta^*(t_i) = \theta^*(t'_i)$ is equivalent to $[\phi^*(t_i)] = [\phi^*(t'_i)]$ and, by Proposition (3.15:i), the equation $(\forall Y)\ \phi^*(t_i) = \phi^*(t'_i)$ is derivable from Th, for each $i = 1, \ldots, n$. Thus, by applying the Rule (Sub) we can derive

$$\text{(Sub)}\ \frac{\begin{array}{c}(\forall X)\ t = t' \Leftarrow \bigwedge_{i=1}^n t_i = t'_i \in Ax(X) \\ (\forall 1 \leqslant i \leqslant n)\ (\forall Y)\ \phi^*(t_i) = \phi^*(t'_i)\end{array}}{(\forall Y)\ \phi^*(t) = \phi^*(t')}\ \phi \colon X \to T_{Sg}(Y)$$

and again by Proposition (3.15:i) we conclude that $[\phi^*(t)] = [\phi^*(t')]$ and therefore $\theta^*(t) = \theta^*(t')$. □

Theorem 3.5 (Completeness). *Let $Th = (Sg, Ax(X))$ be a theory over the coherent signature Sg. If $(\forall Y)\ t = t'$ is satisfied by any Th-model, then $(\forall Y)\ t = t'$ is derivable from Th using rules in Figure 3.4.*

Proof. Suppose that $(\forall Y)\ t = t'$ is satisfied by any Th-model, in particular it is satisfied by $\mathscr{T}_{Th}(Y)$ (by Proposition 3.16). Therefore, by Proposition (3.15:ii), $(\forall Y)\ t = t'$ is derivable from Th. □

Example 3.12. Consider the following theory which axiomatises βη-reduction of λ-terms. Let $Sg(\lambda^+)$ be the signature containing $Sg(\lambda)$ of Example 3.2 with the addition of the operators needed to handle α-conversion and substitution (detailed below). The following unconditional equation

$$(\forall m, n\colon exp) \qquad \circ(\lambda_x(m), n) = \mathrm{sub}_x(m, n) \tag{3.6}$$

axiomatises β-reduction, for each concrete instance of $x \in Var$, where we assume that $\mathrm{sub}_x\colon exp, exp \to exp$ is the substitution operator of n for free occurrences of x in m. Next, the conditional equation

$$(\forall m\colon exp) \qquad \lambda_x(\circ(m, x)) = m \iff \mathrm{not}(\mathrm{in}_x(\mathrm{fv}(m))) = \mathrm{true} \tag{3.7}$$

model the η-reduction for each $x \in Var$. In particular, $\mathrm{not}\colon bool \to bool$ is assumed to be the negation operator in the theory of booleans, $\mathrm{in}_x\colon varset \to bool$ is the operator returning true if x is contained in the set of variables modeled by the sort *varset*, and $\mathrm{fv}\colon exp \to varset$ provides the set of free variables in a given λ-term (the equational axiomatisation of all these new operators is simple but burdensome in terms of auxiliary definitions and specifications, and therefore omitted in this example; the reader may want to consult [VM06] to work out every detail). The theory $Th(\lambda^+) = (Sg(\lambda^+), Ax(\lambda^+))$ is now obtained by taking Equations 3.6 and 3.7 as axioms over the set of variables containing $m, n\colon exp$. ♢

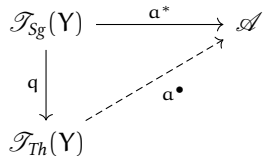
3.9 Initiality and Freeness Results

In previous sections we showed initiality and freeness results for $\mathscr{T}_{Sg}$ and $\mathscr{T}_{Sg}(X)$, respectively, in the category $Alg(Sg)$. Recall that $\mathscr{T}_{Sg}(\varnothing) = \mathscr{T}_{Sg}$ (see Remark 3.4). We now prove analogous results for the quotient algebra $\mathscr{T}_{Th}(Y)$ in the category $Mod(Th)$.

Theorem 3.6. *Let $Th = (Sg, Ax(X))$ be a coherent order-sorted theory, Y an S-sorted set of variables, and $\mathscr{A}$ an order-sorted Th-model. Then, given an assignment $a\colon Y \to A$ (that is, an S-sorted function from Y into the carrier set A of the algebra $\mathscr{A}$), there is a unique Sg-homomorphism $a^\bullet\colon \mathscr{T}_{Th}(Y) \to \mathscr{A}$ extending a, i.e., $a^\bullet_s([y]) = a_s(y)$ for each $y \in Y_s$.*

Proof. Since $\mathscr{A}$ is a Th-model, then it satisfies each equation derivable from Th (Theorem 3.4). Therefore, the congruence relation $\sim$ of Lemma 3.4 is contained in the kernel of the Sg-homomorphism $a^*\colon \mathscr{T}_{Sg}(Y) \to \mathscr{A}$ which exists by Theorem 3.2, that is $\sim \subseteq \ker(a^*)$. Then, by the universal property of quotient (Proposition 3.11), there is a unique Sg-homomorphism $a^\bullet\colon \mathscr{T}_{Th}(Y) \to \mathscr{A}$ such that the following diagram

commutes:



where q send each term t to its equivalence class $[t]$. Suppose now $h\colon \mathscr{T}_{Th}(Y) \to \mathscr{A}$ is an Sg-homomorphism such that $h_s([y]) = a_s(y)$. Since $a = h \circ [-]$, by the freeness of $\mathscr{T}_{Sg}(Y)$, we have that $a^* = (h \circ [-])^* = h \circ q$, hence $h = a^\bullet$. □

Corollary 3.2. *Let Th be a coherent order-sorted theory. Then, $\mathscr{T}_{Th}(\varnothing)$ is the initial model in Mod(Th).*

CHAPTER 4

Algebraic Multi-Language Constructions

Chapter reference(s): [BM19a]

Multi-languages arise by combining existing languages [PA14, AB11, TM07, FF08, Gra08, PPDA17, OSZ12, MF09]. For instance, the multi-language in [MF09] allows programmers to interchange ML expressions and Scheme expressions. Benefits are code reuse and software interoperability. Unfortunately, they come at the price of a lack of clarity of (formal) properties of the new multi-language, mainly semantic specifications. Developing such properties is a key focus of this chapter.

A first step in the formalisation of multi-language semantics has been achieved by Matthews and Findler. In [MF09], they propose *boundary functions* as a way to regulate the flow of values between languages. They show their approach on different variants of the same multi-language obtained by mixing ML [MTH90] and Scheme [Dyb09], representing two "syntactically sugared" versions of the simply-typed and untyped lambda calculi, respectively.

Rather than showing the embedding of two specific languages, we extend their approach to the much broader class of *order-sorted algebras* [GM92] with the aim of providing a framework that works regardless of the inherent nature of the combined languages.

Structure In this chapter, 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. We argue that when such generality is superfluous, we can achieve a neater approach where boundary functions do not need to carry the sorts between which they act as a bridge. Indeed, we show that when the cross-language conversion of a term does not depend on the sort at which the term is considered (that is, when boundaries are *subsort polymorphic*) the framework is able to apply the correct conversion. This last construction is an improvement of the original notion of boundaries in [MF09]. 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

$$e ::= n \mid e \texttt{ + } e \quad \text{where } n \in \mathbb{N} \qquad\qquad s ::= \texttt{-} \mid a \mid s \texttt{ + } s \quad \text{where } a \in \mathbb{A}$$

(a) BNF grammar of $\mathcal{L}(Exp)$. (b) BNF grammar of $\mathcal{L}(Str)$.

Figure 4.1: The BNF grammars of the languages $\mathcal{L}(Exp)$ and $\mathcal{L}(Str)$.

$$[\![n]\!] = n \qquad\qquad [\![\texttt{-}]\!] = \varepsilon$$
$$[\![e \texttt{ + } e']\!] = [\![e]\!] + [\![e']\!] \qquad\qquad [\![a]\!] = a$$
$$[\![s \texttt{ + -}]\!] = [\![\texttt{- + } s]\!] = [\![s]\!]$$
$$[\![s \texttt{ + - + } s']\!] = [\![s \texttt{ + } s']\!]$$
$$[\![a_0 \texttt{ + } \ldots \texttt{ + } a_n]\!] = a_0 \ldots a_n \quad n > 0$$

(a) Formal semantics of $\mathcal{L}(Exp)$. (b) Formal semantics of $\mathcal{L}(Str)$.

Figure 4.4: The formal semantics of the languages $\mathcal{L}(Exp)$ and $\mathcal{L}(Str)$.

languages. Finally, we provide a very specific notion of multi-language where no extra operator is added to the syntax. 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 *associated* order-sorted representation, namely it falls within the same formal model of the combined languages. Finally, we model the multi-language designed in [MF09] in order to show an instantiation of the framework. Please note that the whole chapter is accompanied by a running example, introduced below. This example is very simple but effective in showing our key ideas.

Running Example 4.1. Let $\mathcal{L}(Exp)$ and $\mathcal{L}(Str)$ be two formal languages (see Figure 4.1). The former is a language to construct simple mathematical expressions: $n \in \mathbb{N}$ ranges over natural numbers and $e \texttt{ + } e$ inductively generates all the possible additions (Figure 4.1(a)). The latter is a language to build strings over a finite alphabet of symbols $\mathbb{A} = \{a, b, \ldots, z\}$: $a \in \mathbb{A}$ denotes atoms (or, characters) of the language, whereas $s \texttt{ + } s$ concatenates them into strings (Figure 4.1(b)). A term in $\mathcal{L}(Exp)$ and $\mathcal{L}(Str)$ is interpreted as an element in the sets $\mathbb{N}$ and $\mathbb{A}^*$, accordingly to equations in Figure 4.2(a) and 4.3(b), respectively.

The syntax of the language $\mathcal{L}(Exp)$ can be given by the following order-sorted signature $Sg(Exp)$: the set of sorts $S(Exp) = \{exp, nat\}$ carries the sort exp of expressions and the sort nat of natural numbers; $\leqslant_{Exp}$ is the discrete order on $S(Exp)$ joined with $nat \leqslant_{Exp} exp$, that is natural numbers are expressions; and the operators in $Sg(Exp)$ are the constants $0, 1, 2, \ldots : nat$ and the function symbol $+ : exp, exp \to exp$.

Similarly, the signature $Sg(Str)$ models the syntax of the language $\mathcal{L}(Str)$: the set $S(Str) = \{str, chr\}$ provides the sorts of strings and characters; the subsort relation $\leqslant_{Str}$ is the discrete order on $S(Str)$ with the addition of the constraint $chr \leqslant_{Str}$

str (characters are one-symbol strings); and the operator symbols in $Sg(Str)$ are $\mathtt{a}, \ldots, \mathtt{z}\colon \mathit{chr}$, $\mathtt{-}\colon \mathit{str}$, and $+\colon \mathit{str}, \mathit{str} \to \mathit{str}$.

Algebraic semantics of $\mathcal{L}(\mathit{Exp})$ and $\mathcal{L}(\mathit{Str})$ are given by the algebras $\mathscr{E}$ and $\mathscr{S}$ over the signatures $Sg(\mathit{Exp})$ and $Sg(\mathit{Str})$, respectively. We set the interpretation domains of $\mathscr{E}$ to $[\![\mathit{nat}]\!]_{\mathscr{E}} = [\![\mathit{exp}]\!]_{\mathscr{E}} = \mathbb{N}$ and those of $\mathscr{S}$ to $[\![\mathit{chr}]\!]_{\mathscr{S}} = \mathbb{A} \subseteq \mathbb{A}^* = [\![\mathit{str}]\!]_{\mathscr{S}}$. Moreover, we define the interpretation functions of $\mathscr{E}$ by

$$\begin{aligned} [\![\mathtt{n}]\!]_{\mathscr{E}} &= \mathtt{n} && \forall \mathtt{n} \in \mathbb{N} \\ [\![+\colon \mathit{exp}, \mathit{exp} \to \mathit{exp}]\!]_{\mathscr{E}}(\mathtt{n}_1, \mathtt{n}_2) &= \mathtt{n}_1 + \mathtt{n}_2 \end{aligned}$$

and those of $\mathscr{S}$ as

$$\begin{aligned} [\![\mathtt{-}]\!]_{\mathscr{S}} &= \varepsilon \\ [\![\mathtt{a}]\!]_{\mathscr{S}} &= \mathtt{a} && \forall \mathtt{a} \in \mathbb{A} \\ [\![+\colon \mathit{str}, \mathit{str} \to \mathit{str}]\!]_{\mathscr{S}}(\mathtt{s}_1, \mathtt{s}_2) &= \mathtt{s}_1\mathtt{s}_2 \end{aligned}$$

Since $Sg(\mathit{Exp})$ and $Sg(\mathit{Str})$ are regular, the algebras $\mathscr{E}$ and $\mathscr{S}$ induce unique semantic functions $\mathfrak{h}_{\mathit{Exp}}\colon \mathscr{T}_{Sg(\mathit{Exp})} \to \mathscr{E}$ and $\mathfrak{h}_{\mathit{Str}}\colon \mathscr{T}_{Sg(\mathit{Str})} \to \mathscr{S}$, providing semantics to generated terms. ♢

4.1 Multi-Language Signatures and Algebras

The first step towards a multi-language construction is the specification of which terms of one language can be employed within another [PPDA17, MF09, OSZ12]. 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 [MF09]).

Definition 4.1 (Multi-Language Signature)**.** A *multi-language signature* is a triple $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ given by

Multi-language signature

(i) a pair of order-sorted signatures Sg_1 and Sg_2 with posets of sorts $(\mathtt{S}_1, \leqslant_1)$ and $(\mathtt{S}_2, \leqslant_2)$, respectively;

(ii) a (binary) relation *join* $\ltimes$ over $\mathtt{S}_1 \cup \mathtt{S}_2$ such that if $\mathtt{s} \ltimes \mathtt{s}'$, then $\mathtt{s} \in \mathtt{S}_\mathtt{i}$ and $\mathtt{s}' \in \mathtt{S}_\mathtt{j}$ with $\mathtt{i}, \mathtt{j} \in \{1, 2\}$ and $\mathtt{i} \neq \mathtt{j}$. ♢

Join relation

In the following, we always assume the sets of sorts $\mathtt{S}_1$ and $\mathtt{S}_2$ of the order-sorted signatures Sg_1 and Sg_2 to be disjoint (this hypothesis is non-restrictive: We can always perform a renaming of the sorts). The join relation $\ltimes$ specifies which syntactic categories can be blended in order to provide the syntax of the multi-language. More precisely, $\mathtt{S}_\mathtt{i} \ni \mathtt{s} \ltimes \mathtt{s}' \in \mathtt{S}_\mathtt{j}$ suggests that we want to use terms of sort $\mathtt{s}$ in $\mathscr{T}_{Sg_\mathtt{i}}$ in place of terms of sort $\mathtt{s}'$ in $\mathscr{T}_{Sg_\mathtt{j}}$. Since $\mathtt{i}$ is required to be different from $\mathtt{j}$, we will embed terms built out of a signature into another, *different*, one.

Notation 4.1. In the rest of the thesis, we shall leave implicit that the usual multi-language signature $\boldsymbol{Sg}$ is given by the triple $(Sg_1, Sg_2, \ltimes)$, unless otherwise stated, and we let $(\mathtt{S}_1, \leqslant_1)$ and $(\mathtt{S}_2, \leqslant_2)$ be the poset of sorts of Sg_1 and Sg_2, respectively. ♢

The role of an algebra is to provide an interpretation domain for each sort as well as the meaning of every operator symbol in the signature. When moving towards the multi-language context, we shall also provide meaning to the *interoperability*

constraints specified by the join relation $\ltimes$. Consequently, if $s \ltimes s'$ in the signature $\boldsymbol{Sg}$, a multi-language $\boldsymbol{Sg}$-algebra has to specify how values of sort s may be regarded as values of sort s'. These specifications are called *boundary functions*.

Multi-language algebra

Definition 4.2 (Multi-Language Algebra)**.** Let $\boldsymbol{Sg}$ be a multi-language signature. A *multi-language $\boldsymbol{Sg}$-algebra* $\mathscr{A}$ is given by

(i) a pair of order-sorted algebras $\mathscr{A}_1$ and $\mathscr{A}_2$ over Sg_1 and Sg_2, respectively; and

Boundary function

(ii) a *boundary function* $[\![s \ltimes s']\!]_{\mathscr{A}} \colon [\![s]\!]_{\mathscr{A}_i} \to [\![s']\!]_{\mathscr{A}_j}$ for each $s \ltimes s'$ in $\boldsymbol{Sg}$, with $s \in S_i$ and $s' \in S_j$ (recall that $i, j \in \{1, 2\}$ and $i \neq j$). ♢

An algebra sets out the meaning of a multi-language: The meaning of the underlying languages, and how terms of sort $s \in S_i$ can be interpreted as terms of sort $s' \in S_j$. Put differently, boundary functions regulate the flow of values across $\mathscr{A}_1$ and $\mathscr{A}_2$ [MF09].

Notation 4.2. If $\mathscr{A}$ is a multi-language $\boldsymbol{Sg}$-algebra, we denote by $\mathscr{A}_1$ and $\mathscr{A}_2$ the underlying order-sorted Sg_1- and Sg_2-algebras. ♢

Running Example 4.2. Suppose we are interested in a multi-language which blends the signatures $Sg(Exp)$ and $Sg(Str)$ of Running Example 4.1 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 $a^{(n \bmod |\mathbb{A}|)}$ (we are assuming a total lexicographical order $a^{(0)}, a^{(1)}, \ldots, a^{(|\mathbb{A}|-1)}$ 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 of natural numbers $\{0, \ldots, |\mathbb{A}| - 1\}$.

We can build such a multi-language by providing a join relation $\ltimes$ on $S(Exp) \cup S(Str)$ (that is, the set of sorts of $Sg(Exp)$ and $Sg(Str)$, respectively) and a boundary function for each interoperability constraint $s \ltimes s'$ introduced by $\ltimes$. These specifications give rise to a multi-language signature $\boldsymbol{Sg}(Exp + Str) = (Sg(Exp), Sg(Str), \ltimes)$ and a multi-language $\boldsymbol{Sg}(Exp + Str)$-algebra $\mathscr{M}$ (with underlying order-sorted algebras $\mathscr{M}_1 = \mathscr{E}$ and $\mathscr{M}_2 = \mathscr{S}$ over $Sg(Exp)$ and $Sg(Str)$, respectively). We define the interoperability constraints and the boundary functions as follows:

$$
\begin{aligned}
&exp \ltimes chr && [\![exp \ltimes chr]\!]_{\mathscr{M}} = \mathrm{chr} \\
&nat \ltimes chr && [\![nat \ltimes chr]\!]_{\mathscr{M}} = \mathrm{chr} \\
&chr \ltimes nat && [\![chr \ltimes nat]\!]_{\mathscr{M}} = \mathrm{ord} \\
&str \ltimes nat && [\![str \ltimes nat]\!]_{\mathscr{M}}(a_0 \ldots a_n) = \sum_{k=0}^{n} [\![chr \ltimes nat]\!]_{\mathscr{M}}(a_k) \cdot 10^{n-k}
\end{aligned}
$$
♢

Given two multi-language algebras over the same signature $\boldsymbol{Sg}$, we can define morphisms between them preserving their strucutre.

Multi-language homomorphism

Definition 4.3 (Multi-Language Homomorphism)**.** Let $\boldsymbol{Sg}$ be a multi-language signature, and let $\mathscr{A}$ and $\mathscr{B}$ be two $\boldsymbol{Sg}$-algebras. A *multi-language $\boldsymbol{Sg}$-homomorphism* $h \colon \mathscr{A} \to \mathscr{B}$ is given by

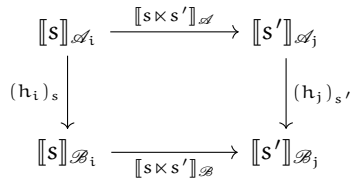


Figure 4.5: The commutativity diagram enforced by the multi-language homomorphism definition.

(i) a pair of order-sorted homomorphisms $h_1 : \mathscr{A}_1 \to \mathscr{B}_1$ and $h_2 : \mathscr{A}_2 \to \mathscr{B}_2$ such that they commute with boundary functions (Figure 4.5), namely

(ii) if $s \ltimes s'$ with $s \in S_i$ and $s' \in S_j$, then

$$(h_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}} = [\![s \ltimes s']\!]_{\mathscr{B}} \circ (h_i)_s \qquad \diamondsuit$$

Here commutativity makes multi-language homomorphisms (and therefore the subsequent definition of multi-language semantics) compositional with respect to boundary functions. This will enable us to consider interoperability constraints $s \ltimes s'$ as fully-fledged operators of the multi-language.

Notation 4.3. In the same spirit of Notation 4.2, if $h : \mathscr{A} \to \mathscr{B}$ is a multi-language ***Sg***-homomorphism, we shall denote by $h_1 : \mathscr{A}_1 \to \mathscr{B}_1$ and $h_2 : \mathscr{A}_2 \to \mathscr{B}_2$ the underlying order-sorted Sg_1- and Sg_2-homomorphisms. $\diamondsuit$

The composition of multi-language homomorphisms is defined componentwise: Let $f : \mathscr{A} \to \mathscr{B}$ and $g : \mathscr{B} \to \mathscr{C}$ be two ***Sg***-homomorphisms. Then, $g \circ f$ is given by taking $g_i \circ f_i : \mathscr{A}_i \to \mathscr{C}_i$ as the order-sorted Sg_i-homomorphism $(g \circ f)_i$, for $i = 1, 2$. Moreover, the identity homomorphism $\mathrm{id}_{\mathscr{A}}$ over a multi-language ***Sg***-algebra $\mathscr{A}$ is given by picking the identity order-sorted homomorphisms over the underlying algebras $\mathscr{A}_1$ and $\mathscr{A}_2$.

Proposition 4.1. *Multi-language homomorphisms are closed under composition.*

Proof. We prove that the composition $g \circ f$ of two ***Sg***-homomorphisms $f : \mathscr{A} \to \mathscr{B}$ and $g : \mathscr{B} \to \mathscr{C}$ commutes with boundary functions (condition (4.3:ii)). Let $s \ltimes s'$ and recall that $s \in S_i$ and $s' \in S_j$ with $i, j \in \{1, 2\}$ and $i \neq j$. Then,

$$\begin{aligned}
&\big((g \circ f)_j\big)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}} \\
&\quad = \big((g_j)_{s'} \circ (f_j)_{s'}\big) \circ [\![s \ltimes s']\!]_{\mathscr{A}} \\
&\quad = (g_j)_{s'} \circ \big((f_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}}\big) \\
&\quad = (g_j)_{s'} \circ \big([\![s \ltimes s']\!]_{\mathscr{B}} \circ (f_i)_s\big) && \text{f is an } \textbf{\textit{Sg}}\text{-homomorphism} \\
&\quad = \big((g_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{B}}\big) \circ (f_i)_s \\
&\quad = \big([\![s \ltimes s']\!]_{\mathscr{C}} \circ (g_i)_s\big) \circ (f_i)_s && \text{g is an } \textbf{\textit{Sg}}\text{-homomorphism} \\
&\quad = [\![s \ltimes s']\!]_{\mathscr{C}} \circ \big((g \circ f)_i\big)_s
\end{aligned}$$

$\square$

$$\text{(mlConst)}\ \frac{-}{k_j : s}\ (k : s \text{ in } Sg_j) \qquad \text{(mlSubsort)}\ \frac{t : s}{t : r}\ (s \leqslant_j r \text{ in } Sg_j)$$

$$\text{(mlFun)}\ \frac{(\forall 1 \leqslant i \leqslant n)\ \ t_i : s_i}{f_j(t_1, \ldots, t_n) : s}\ (f : s_1, \ldots, s_n \to s \text{ in } Sg_j)$$

$$\text{(mlInter)}\ \frac{t : s}{\hookrightarrow_{s,s'}(t) : s'}\ (s \ltimes s' \text{ in } \boldsymbol{Sg})$$

Figure 4.6: Well-formed ground terms generated by a multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

Hence, as in the many-sorted and order-sorted case [GTWW77, GM92], we immediately have the category of all the multi-language algebras over a multi-language signature:

Category of algebras over $\boldsymbol{Sg}$

Proposition 4.2. *Let $\boldsymbol{Sg}$ be a multi-language signature. The class of all $\boldsymbol{Sg}$-algebras and the class of all $\boldsymbol{Sg}$-homomorphisms form a category $Alg(\boldsymbol{Sg})$.*

4.2 Terms of a Multi-Language

Multi-language (ground) term

In this section, we define the notion of *multi-language (ground) terms* over a multi-language signature $\boldsymbol{Sg}$. Intuitively, we want multi-language terms to comprise all those built out of the underlying order-sorted signatures Sg_1 and Sg_2, as well as those obtained by the blending the two languages together according to the interoperability constraints given by 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 $s \ltimes s'$ (recall that s and s' are syntactic categories of different languages due to condition (4.1:ii)). Nonetheless, we must be careful not to add ambiguities during this process: A term t may be generated from both Sg_1 and Sg_2. When t is included in the multi-language, we lose the information to track the original language, thus making the (multi-language) semantics of t ambiguous. The simplest solution, explored in this section and refined in Section 4.4, to avoid such issues is to add syntactical notations to make explicit the context of the language in which we are operating.

Terms built out of a multi-language signature $\boldsymbol{Sg}$ are given by the rules in Figure 4.6. Except for (mlInter), the other rules are similar to the order-sorted case (*cf.* Figure 3.2). However, the reader may notice that both function symbols (Rule (mlFun)) and constants (Rule (mlConst)) carry the information of the source language as a subscript. For instance, this formalises some ad hoc multi-language specifications found in literature: [PA14, AB11, PPDA17] exploit distinct colors to disambiguate the source language of the operators, whereas [MF09] use different font styles for different languages. Finally, the key aspect of multi-language terms is encoded in Rule (mlInter): If t is a multi-language term of sort s and $s \ltimes s'$ (that is, s can interoperate with s'), then the new term $\hookrightarrow_{s,s'}(t)$ has sort s' and can be seen as the embedding of t in a different language.

Running Example 4.3. Recall the multi-language signature of Running Example 4.2. Well-formed multi-language terms built out of $\boldsymbol{Sg}(Exp + Str)$ according to rules in

Figure 4.6 are

$$\hookrightarrow_{str,nat}(+_2(\mathsf{s}_2, \hookrightarrow_{nat,chr}(1_1))) \quad +_1(2_1, 1_1) \quad +_2(\mathsf{m}_2, 1_2) \quad \hookrightarrow_{chr,nat}(\hookrightarrow_{nat,chr}(0_1))$$

In order to simplify the notation, we shall use colours in place of numerical subscript (blue for the first signature, red for the second one). For instance, the previous terms are re-written as

$$\hookrightarrow_{str,nat}(+(\mathsf{s}, \hookrightarrow_{nat,chr}(1))) \quad +(2, 1) \quad +(\mathsf{m}, 1) \quad \hookrightarrow_{chr,nat}(\hookrightarrow_{nat,chr}(0)) \qquad \diamondsuit$$

4.2.1 Multi-Language Term Algebra

As is the case with order-sorted terms, the construction of multi-language terms over a signature $\boldsymbol{Sg}$ gives rise to a (multi-language) term algebra $\mathscr{T}_{\boldsymbol{Sg}}$. Its definition goes through the concept of *associated signature* (recall that $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ and $(S_i, \leqslant_i)$ is the poset of sorts of Sg_i with $i = 1, 2$):

Associated signature

Definition 4.4 (Associated Signature)**.** The *associated signature* of a multi-language signature $\boldsymbol{Sg}$ is the order-sorted signature $\boldsymbol{Sg}^*$ defined by

(i) $S = S_1 \cup S_2$ as set of sorts, ordered by $\leqslant = \leqslant_1 \cup \leqslant_2$;

(ii) $k_i : s$ is a constant in $\boldsymbol{Sg}^*$ if $k : s$ is a constant in Sg_i;

(iii) $f_i : w \to s$ is a function symbol in $\boldsymbol{Sg}^*$ if $f : w \to s$ is a function symbol in Sg_i; and

(iv) $\hookrightarrow_{s,s'}$ is a function symbol in $\boldsymbol{Sg}^*$, usually referred to as *conversion operator*, if $s \ltimes s'$ in $\boldsymbol{Sg}$. $\diamondsuit$

Conversion operator

Notation 4.4. Whenever we talk about the associated signature $\boldsymbol{Sg}^*$ of $\boldsymbol{Sg}$, we leave implicit that $(S, \leqslant)$ is the poset of sorts defined as in the previous definition. $\diamondsuit$

It is easy to see that an associated signature is indeed an order-sorted signature, therefore admitting a term algebra $\mathscr{T}_{\boldsymbol{Sg}^*}$. Although it may seem a suitable definition of the multi-language term algebra, it does not meet Definition 4.2. Fortunately, we can base the definition of $\mathscr{T}_{\boldsymbol{Sg}}$ on the construction of $\mathscr{T}_{\boldsymbol{Sg}^*}$, with the aim of providing a fully-fledged multi-language term algebra.

Multi-language term algebra

Definition 4.5 (Multi-Language Term Algebra)**.** The *multi-language term algebra* $\mathscr{T}_{\boldsymbol{Sg}}$ over a signature $\boldsymbol{Sg}$ is defined as follows: Let $i = 1, 2$, then

(i) the order-sorted Sg_i-algebra $(\mathscr{T}_{\boldsymbol{Sg}})_i$ is given by

  (a) $[\![s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}})_i} = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}^*}}$ for each $s \in Sg_i$;

  (b) $[\![k]\!]_{(\mathscr{T}_{\boldsymbol{Sg}})_i} = [\![k_i]\!]_{\mathscr{T}_{\boldsymbol{Sg}^*}}$ for each constant k in Sg_i;

  (c) $[\![f : w \to s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}})_i} = [\![f_i : w \to s]\!]_{\mathscr{T}_{\boldsymbol{Sg}^*}}$ for each $f : w \to s$ in Sg_i; and

(ii) boundary functions are given by $[\![s \ltimes s']\!]_{\mathscr{T}_{\boldsymbol{Sg}}} = [\![\hookrightarrow_{s,s'} : s \to s']\!]_{\mathscr{T}_{\boldsymbol{Sg}^*}}$ for each interoperability constraint $s \ltimes s'$ in $\boldsymbol{Sg}$. $\diamondsuit$

Please note that $(\mathscr{T}_{\boldsymbol{Sg}})_i$ differs from the term algebra over Sg_i (which is denoted by $\mathscr{T}_{Sg_i}$). Indeed, $\mathscr{T}_{Sg_i}$ is a *subobject* of $(\mathscr{T}_{\boldsymbol{Sg}})_i$, where the latter also carries multi-language terms with sorts in Sg_i. The following example clarifies the distinction.

Running Example 4.4. Consider again the order-sorted signature $Sg(Exp)$ along with the multi-language specified by $\boldsymbol{Sg}(Exp + Str)$. For instance, the term $+(0,1)$ appears in $\mathscr{T}_{Sg(Exp)}$ and — in its annotated version $+(0,1)$ — in $(\mathscr{T}_{\boldsymbol{Sg}(Exp+Str)})_1$ (due to this distinction, we refer to $\mathscr{T}_{Sg(Exp)}$ as a suboject of $(\mathscr{T}_{\boldsymbol{Sg}(Exp+Str)})_1$ rather than a subalgebra). Conversely, the "proper" multi-language term $\hookrightarrow_{exp,chr}(+(0,1))$ only resides in the algebra $(\mathscr{T}_{\boldsymbol{Sg}(Exp+Str)})_1$. ♢

4.3 Multi-Language Algebraic Semantics

The last step in order to finalise the framework is to provide multi-language terms with a meaning. As with the order-sorted case, we need a notion of regularity in order to prove the initiality of the multi-language term algebra in its category, which in turn ensures a single eligible (initial algebra) semantics.

Multi-language regularity

Definition 4.6 (Regularity)**.** A multi-language signature $\boldsymbol{Sg}$ is *regular* if its associated signature $\boldsymbol{Sg}^*$ is regular. ♢

Proposition 4.3. *A multi-language signature $\boldsymbol{Sg}$ is regular if and only if Sg_1 and Sg_2 are regular.*

Proof. ($\Longrightarrow$) Let $\boldsymbol{Sg}$ be a regular multi-language signature. Therefore, its associated signature $\boldsymbol{Sg}^*$ is regular. Let $f\colon w \to s$ be a function symbol in Sg_i (for some $i = 1,2$) and $w_0 \leqslant w$ a lower bound of w. We need to prove that the set

$$A = \{\,(w',s') \in S_i^+ \times S_i \mid f\colon w' \to s' \wedge w_0 \leqslant w'\,\}$$

admits a minimum. Since f is a function symbol in Sg_i, then $f_i\colon w \to s$ is a function symbol in $\boldsymbol{Sg}^*$ by condition (4.4:iii). By regularity of $\boldsymbol{Sg}^*$ it follows that

$$A^* = \{\,(w',s') \in S^+ \times S \mid f_i\colon w' \to s' \wedge w_0 \leqslant w'\,\}$$

admits a minimum. Moreover,

$$\begin{aligned}(w',s') \in A &\iff \exists f\colon w' \to s' \text{ in } Sg_i \\ &\iff \exists f_i\colon w' \to s' \text{ in } \boldsymbol{Sg}^* \\ &\iff (w',s') \in A^*\end{aligned} \tag{4.1}$$

and therefore $A = A^*$. Since $\leqslant_i$ is contained into $\leqslant$, then they share the same minimum.

($\Longleftarrow$) Let Sg_1 and Sg_2 be regular signatures. Let $f_i\colon w \to s$ be a function symbol in $\boldsymbol{Sg}^*$ and $w_0 \leqslant w$ a lower bound of w. Therefore, there is some $i = 1,2$ such that $f\colon w \to s$ is in Sg_i. We need to prove that

$$A^* = \{\,(w',s') \in S^+ \times S \mid f_i\colon w' \to s' \wedge w_0 \leqslant w'\,\}$$

has a minimum, and the argument is analogous to that of the previous case. □

We are now in position to prove that $\mathscr{T}_{\boldsymbol{Sg}}$ is the initial object in $Alg(\boldsymbol{Sg})$. Initiality of $\mathscr{T}_{\boldsymbol{Sg}}$ is essential to assign a unique mathematical meaning to each term, as in the order-sorted case: Given a multi-language algebra $\mathscr{A}$, we have to show that there is a single interpretation of each multi-language term t in $\mathscr{A}$.

Theorem 4.1. *Let $\boldsymbol{Sg}$ be a regular multi-language signature. Then, $\mathscr{T}_{\boldsymbol{Sg}}$ is the initial object of $Alg(\boldsymbol{Sg})$.*

Proof. We have to show that for each multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ there is a unique multi-language homomorphism $h\colon \mathscr{T}_{\boldsymbol{Sg}} \to \mathscr{A}$. Let $\mathscr{A}$ be such an algebra. Consider the new order-sorted $\boldsymbol{Sg}^*$-algebra $\mathscr{B}$ defined as follows:

- $[\![s]\!]_{\mathscr{B}} = [\![s]\!]_{\mathscr{A}_i}$ if $s \in S_i$ for some $i = 1, 2$;
- $[\![k_i]\!]_{\mathscr{B}} = [\![k]\!]_{\mathscr{A}_i}$ for each constant $k_i\colon s$ in $\boldsymbol{Sg}^*$ and for some $i = 1, 2$;
- $[\![f_i\colon w \to s]\!]_{\mathscr{B}} = [\![f\colon w \to s]\!]_{\mathscr{A}_i}$ for each function symbol $f\colon w \to s$ in Sg_i and for some $i = 1, 2$; and
- $[\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} = [\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$ in $\boldsymbol{Sg}$.

Proving that $\mathscr{B}$ is an order-sorted $\boldsymbol{Sg}^*$-algebra is immediate. Since $\boldsymbol{Sg}$ is regular by hypothesis, then $\boldsymbol{Sg}^*$ is regular by Definition 4.6 and therefore $\mathscr{T}_{\boldsymbol{Sg}^*}$ is initial in its category. Thus, there is a unique order-sorted $\boldsymbol{Sg}^*$-homomorphism $f\colon \mathscr{T}_{\boldsymbol{Sg}^*} \to \mathscr{B}$. Moreover, we can regard f as a multi-language $\boldsymbol{Sg}$-homomorphism h from $\mathscr{T}_{\boldsymbol{Sg}}$ to $\mathscr{A}$ by considering $h_i\colon (\mathscr{T}_{\boldsymbol{Sg}})_i \to \mathscr{A}_i$ the restriction of f to S_i, that is $h_i = f|_{S_i}$. Checking that h_i is an order-sorted Sg_i-homomorphism is trivial and omitted, and we only prove that h satisfies condition (4.3:ii): Suppose that $s \ltimes s'$ with $s \in S_i$ and $s' \in S_j$ where $i, j \in \{1, 2\}$ and $i \neq j$, then

$$
\begin{aligned}
(h_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}} &= (h_j)_{s'} \circ [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} && \text{by def. of } [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} \\
&= f_{s'} \circ [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} && h_j = f|_{S_j} \\
&= [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} \circ f_s && f \text{ is an } \boldsymbol{Sg}^*\text{-homomorphism} \\
&= [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}} \circ (h_i)_s && h_i = f|_{S_i} \\
&= [\![s \ltimes s']\!]_{\mathscr{A}} \circ (h_i)_s && \text{by def. of } [\![\hookrightarrow_{s,s'}\colon s \to s']\!]_{\mathscr{B}}
\end{aligned}
$$

For the uniqueness part, suppose that $h'\colon \mathscr{T}_{\boldsymbol{Sg}} \to \mathscr{A}$ is a multi-language homomorphism. Now let g be the S-sorted function $g = h'_1 \cup h'_2$, that is

$$
g_s = \begin{cases} (h'_1)_s & \text{if } s \in S_1 \\ (h'_2)_s & \text{if } s \in S_2 \end{cases}
$$

(this is well-defined, since S_1 and S_2 are disjoint). It is immediate to see that g is also an order-sorted $\boldsymbol{Sg}^*$-homomorphism from $\mathscr{T}_{\boldsymbol{Sg}^*}$ to $\mathscr{B}$, and by uniqueness we have $g = f$. Hence $h' = h$ and the proof is concluded. □

Multi-language term semantics

We can now provide the semantics of a multi-language $\boldsymbol{Sg}$-term t induced by an $\boldsymbol{Sg}$-algebra $\mathscr{A}$. Let $h\colon \mathscr{T}_{\boldsymbol{Sg}} \to \mathscr{A}$ be the unique multi-language homomorphism, and let $ls(t)$ in Sg_i. Then,

$$
[\![t]\!]_{\mathscr{A}} = (h_i)_{ls(t)}(t) \tag{4.2}
$$

is the meaning of t induced by $\mathscr{A}$. Note that the least sort of t is given with respect to the associated signature $\boldsymbol{Sg}^*$ (which is well-defined, since $S = S_1 \cup S_2$).

Running Example 4.5. Consider the multi-language $\boldsymbol{Sg}(Exp + Str)$-algebra $\mathscr{M}$ in Running Example 4.2. The term algebra $\mathscr{T}_{\boldsymbol{Sg}(Exp+Str)}$ over $\boldsymbol{Sg}(Exp + Str)$ provides all

the multi-language terms, and Theorems 4.1 ensures a unique denotation of each t in $\mathscr{T}_{Sg}$ in $\mathscr{M}$. For instance, the term

$$t = \overbrace{\hookrightarrow_{str,nat}(\overbrace{+_2(f_2, +_2(o_2, \underbrace{\hookrightarrow_{exp,chr}(\overbrace{+_1(10_1, 5_1)}^{t_4})}_{t_3}))}^{t_2})}^{t_1} \tag{4.3}$$

which, according to the notation introduced in Example 4.3, can be rewritten as

$$\hookrightarrow_{str,nat}(+(f, +(o, \hookrightarrow_{exp,chr}(+(10, 5)))))$$

denotes the natural numbers 765:

$$\begin{aligned}
[\![t_4]\!]_{\mathscr{A}} &= (h_1)_{exp}(t_4) = [\![+\colon exp, exp \to exp]\!]_{\mathscr{A}}([\![10]\!]_{\mathscr{A}}, [\![5]\!]_{\mathscr{A}}) \\
&= [\![+\colon exp, exp \to exp]\!]_{\mathscr{A}}(10, 5) = 15 \\
[\![t_3]\!]_{\mathscr{A}} &= (h_2)_{chr}(t_3) = [\![\hookrightarrow_{exp,chr}]\!]_{\mathscr{A}}([\![t_4]\!]_{\mathscr{A}}) = [\![\hookrightarrow_{exp,chr}]\!]_{\mathscr{A}}(15) = o \\
[\![t_2]\!]_{\mathscr{A}} &= (h_2)_{str}(t_2) = [\![+\colon str, str \to str]\!]_{\mathscr{A}}([\![o]\!]_{\mathscr{A}}, [\![t_3]\!]_{\mathscr{A}}) \\
&= [\![+\colon str, str \to str]\!]_{\mathscr{A}}(o, o) = oo \\
[\![t_1]\!]_{\mathscr{A}} &= (h_2)_{str}(t_1) = [\![+\colon str, str \to str]\!]_{\mathscr{A}}([\![f]\!]_{\mathscr{A}}, [\![t_2]\!]_{\mathscr{A}}) \\
&= [\![+\colon str, str \to str]\!]_{\mathscr{A}}(f, oo) = foo \\
[\![t]\!]_{\mathscr{A}} &= (h_1)_{nat} = [\![\hookrightarrow_{str,nat}]\!]_{\mathscr{A}}([\![t_1]\!]_{\mathscr{A}}) = [\![\hookrightarrow_{str,nat}]\!]_{\mathscr{A}}(foo) = 765
\end{aligned}$$ ♢

4.4 Refining the Multi-Language Construction

The multi-language construction presented in Section 4.1 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 between sorts and subsorts: According to the running example, it would have been possible to define $[\![nat \ltimes chr]\!]_{\mathscr{M}}$ differently from $[\![exp \ltimes chr]\!]_{\mathscr{M}}$ although $nat \leqslant_{Exp} exp$, in order to employ distinct conversion specifications based on whether terms are used as natural numbers (*nat*) or as expressions (*exp*). However, when this amount of flexibility is not needed, we can refine the previous construction by simplifying the definition of associated signature. In the following sections we examine

- the case where boundary functions satisfy the monotonicity conditions of order-sorted algebra operators (Section 4.5); and
- the case where boundary functions commutes with the semantics of operator symbols (Section 4.6).

In the first case, we shall see that we can handle the embedding of terms from one language to another with a single polymorphic symbol $\hookrightarrow$ rather than a family of parametrised conversion operators $\{\hookrightarrow_{s,s'} \mid s \ltimes s'\}$. This has a major benefit in avoiding type cast bugs in real world implementations. Indeed, in this new kind of construction, developers do not have to explicitly write the sorts when moving terms across languages.

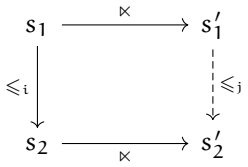


Figure 4.7: The commutativity diagram enforced by the SP signature definition.

Then, in the second case, we will define a new multi-language construction in which no extra syntax is added when building multi-language terms. This means that both conversion operators and operators subscript are not present in this new formalisation. Despite the requirements for this last construction are more restrictive, we will see that it turns out particularly useful for modelling the extension of a language with a new fragment. In this case, boundary functions are likely to be identity functions which trivially satisfy the constraints imposed by the framework.

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

4.5 Subsort Polymorphic Boundary Functions

In the multi-language construction presented in Section 4.1, any interoperability constraint $s \ltimes s'$ turned into an operator $\hookrightarrow_{s,s'}$ in the associated signature (see condition (4.4:iv)). Here, we show how to handle all this syntactical overhead with a single polymorphic operator $\hookrightarrow$ whenever interoperability constraints satisfy the monotonicity condition of order-sorted signatures, that is conditions (3.1:iv).

Definition 4.7 (Subsort Polymorphic Multi-Language Signature)**.** A *subsort polymorphic (SP) multi-language signature* is a multi-language signature $\boldsymbol{Sg}$ satisfying the following additional constraint (see Figure 4.7):

Subsort polymorphic (SP) multi-language signaure

(i) $s_1 \ltimes s'_1$, $s_2 \ltimes s'_2$, and $s_1 \leqslant_i s_2$ imply $s'_1 \leqslant_j s'_2$ ($i, j \in \{1, 2\}$ and $j \neq i$). ♢

Such a constraint has the following meaning: If we can use terms of sort s_1 and s'_1 in place of terms of sort s_2 and s'_2, respectively, and terms of sort s_1 have also sort s_2, then the conversion of s_1-terms has also to be s'_2-terms. Note that if $s \ltimes s'_1$ and $s \ltimes s'_2$, then $s'_1 = s'_2$.

Furthermore, order-sorted algebras demand consistency on the smaller domain when interpreting subsort polymorphic operators, which results in the following condition (4.8:i) on boundary functions:

Definition 4.8 (Subsort Polymorphic Multi-Language Algebra)**.** Let $\boldsymbol{Sg}$ be an SP multi-language signature. A *subsort polymorphic (SP) multi-language $\boldsymbol{Sg}$-algebra* is a multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ such that

Subsort polymorphic (SP) multi-language algebra

(i) $s_1 \ltimes s'_1$, $s_2 \ltimes s'_2$, and $s_1 \leqslant_i s_2$ imply that $[\![s_1 \ltimes s'_1]\!]_{\mathscr{A}}(a) = [\![s_2 \ltimes s'_2]\!]_{\mathscr{A}}(a)$ for each $a \in [\![s_1]\!]_{\mathscr{A}}$, with $i = 1, 2$. ♢

The notion of homomorphism in this new context does not change. That is, a homomorphism between two SP algebras is still an S-sorted function decomposable in two order-sorted homomorphisms that commutes with boundary functions.

$$\text{(spmlConst)}\ \frac{-}{k_j : s}\ (k : s \text{ in } Sg_j) \qquad \text{(spmlSubsort)}\ \frac{t : s}{t : r}\ (s \leqslant_j r \text{ in } Sg_j)$$

$$\text{(spmlFun)}\ \frac{(\forall 1 \leqslant i \leqslant n)\ \ t_i : s_i}{f_j(t_1, \ldots, t_n) : s}\ (f : s_1, \ldots, s_n \to s \text{ in } Sg_j)$$

$$\text{(spmlInter)}\ \frac{t : s}{\hookrightarrow(t) : s'}\ (s \ltimes s' \text{ in } \boldsymbol{Sg})$$

Figure 4.8: Well-formed ground terms generated by a subsort polymorphic multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

4.5.1 Subsort Polymorphic Term Algebra and Initial Semantics

Terms built out of a subsort polymorphic signature follow the same rationale as those defined by the rules in Figure 4.6, with the single exception that conversion operators do not require any kind of type annotation in their syntax. This shall lead to a single polymorphic conversion operator. The new rules for term generation are depicted in Figure 4.8.

Running Example 4.6. The multi-language signature $\boldsymbol{Sg}(Exp + Str)$ is subsort polymorphic, indeed

- $nat \ltimes chr$, $exp \ltimes chr$, $nat \leqslant_1 exp$, and $chr \leqslant_2 chr$; and
- $chr \ltimes nat$, $str \ltimes nat$, $chr \leqslant_2 str$, and $nat \leqslant_2 nat$.

Thus, the "equivalent" of multi-language terms shown in Running Example 4.3 are

$$\hookrightarrow(+_2(s_2, \hookrightarrow(1_1))) \quad +_1(2_1, 1_1) \quad +_2(m_2, 1_2) \quad \hookrightarrow(\hookrightarrow(0_1))$$

or, according to the previous notation,

$$\hookrightarrow(+(s, \hookrightarrow(1))) \quad +(2, 1) \quad +(m, 1) \quad \hookrightarrow(\hookrightarrow(0)) \qquad \diamondsuit$$

The associated signature to an SP multi-language signature merely differs from Definition 4.4 for carrying a unique polymorphic operator $\hookrightarrow$ instead of a family of parametrised symbols $\{\hookrightarrow_{s,s'} \mid s \ltimes s'\}$.

Subsort polymorphic (SP) associated signature

Definition 4.9 (Subsort Polymorphic Associated Signature)**.** The *subsort polymorphic (SP) associated signature* to the SP multi-language signature $\boldsymbol{Sg}$ is the order-sorted signature $\boldsymbol{Sg}^\circ$ defined by

(i) $S = S_1 \cup S_2$ as set of sorts, ordered by $\leqslant = \leqslant_1 \cup \leqslant_2$;

(ii) $k_i : s$ is a constant in $\boldsymbol{Sg}^\circ$ if $k : s$ is a constant in Sg_i;

(iii) $f_i : w \to s$ is a function symbol in $\boldsymbol{Sg}^\circ$ if $f : w \to s$ is a function symbol in Sg_i; and

(iv) a function symbol $\hookrightarrow : s \to s'$ (the *conversion operator*) for each $s \ltimes s'$. $\diamondsuit$

We now prove the well-definedness nature of the new signature $\boldsymbol{Sg}^\circ$. We shall check the monotonicity of each operator in $\boldsymbol{Sg}^\circ$, notably that of the polymorphic conversion operator $\hookrightarrow$.

Proposition 4.4. *Let* $\boldsymbol{Sg}$ *be an SP multi-language signature. Then,* $\boldsymbol{Sg}^\circ$ *is a proper order-sorted signature.*

Proof. We need to check condition (3.1:iv). Let $f\colon w_1 \to s$ and $f\colon w_2 \to r$ be function symbols in $\boldsymbol{Sg}^\circ$ with $w_1 \leqslant w_2$. There are two cases:

- $f = g_i$ for some function symbols $g\colon w_1 \to s$ and $g\colon w_2 \to r$ in Sg_i, for some $i = 1, 2$. Since Sg_i is an order-sorted signature, then $s \leqslant_i r$ and therefore $s \leqslant r$ by (4.9:i).
- $f = \hookrightarrow$ where $\hookrightarrow\colon s_1 \to s_1'$ and $\hookrightarrow\colon s_2 \to s_2'$. Then, $s_1 \ltimes s_1'$ and $s_2 \ltimes s_2'$. Since $s_1 = w_1 \leqslant w_2 = s_2$, by condition (4.7:i) follows that $s_1' = s \leqslant r = s_2'$. □

Again, we can show that subsort polymorphic term construction is purely algebraic by providing a proper term algebra. We will build such an algebra by exploiting the order-sorted term algebra over the SP associated signature.

Definition 4.10 (Subsort Polymorphic Multi-Language Term Algebra)**.** The *subsort polymorphic (SP) multi-language term algebra* $\mathscr{T}^\circ_{\boldsymbol{Sg}}$ over an SP signature $\boldsymbol{Sg}$ is defined as follows: Let $i = 1, 2$, then

Subsort polymorphic (SP) multi-language term algebra

(i) the order-sorted Sg_i-algebra $(\mathscr{T}^\circ_{\boldsymbol{Sg}})_i$ is given by

  (a) $[\![s]\!]_{(\mathscr{T}^\circ_{\boldsymbol{Sg}})_i} = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}^\circ}}$ for each $s \in Sg_i$;

  (b) $[\![k]\!]_{(\mathscr{T}^\circ_{\boldsymbol{Sg}})_i} = [\![k_i]\!]_{\mathscr{T}_{\boldsymbol{Sg}^\circ}}$ for each constant k in Sg_i;

  (c) $[\![f\colon w \to s]\!]_{(\mathscr{T}^\circ_{\boldsymbol{Sg}})_i} = [\![f_i\colon w \to s]\!]_{\mathscr{T}_{\boldsymbol{Sg}^\circ}}$ for each $f\colon w \to s$ in Sg_i; and

(ii) boundary functions are given by $[\![s \ltimes s']\!]_{\mathscr{T}^\circ_{\boldsymbol{Sg}}} = [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{T}_{\boldsymbol{Sg}^\circ}}$ for each interoperability constraint $s \ltimes s'$ in $\boldsymbol{Sg}$. ♢

With the aim of defining an SP multi-language semantics, we need a notion of regularity for SP signatures. As with ordinary multi-language signatures, SP regularity shall be based on the associated signature, in an analogous way to Definition 4.6.

Definition 4.11 (Regularity)**.** A subsort polymorphic multi-language signature $\boldsymbol{Sg}$ is *regular* if its associated signature $\boldsymbol{Sg}^\circ$ is regular. ♢

SP signature regularity

It is easy to prove that an equivalent of Proposition 4.3 still holds for subsort polymorphic multi-language signatures.

Proposition 4.5. *A subsort polymorphic multi-language signature* $\boldsymbol{Sg}$ *is regular if and only if* Sg_1 *and* Sg_2 *are regular.*

We can now define the category $Alg^\circ(\boldsymbol{Sg})$ of SP multi-language algebras over a given signature $\boldsymbol{Sg}$, and we shall prove that the SP term algebra $\mathscr{T}^\circ_{\boldsymbol{Sg}}$ is initial in this category.

Proposition 4.6. *Let* $\boldsymbol{Sg}$ *be an SP multi-language signature. The class of all SP* $\boldsymbol{Sg}$*-algebras and the class of all* $\boldsymbol{Sg}$*-homomorphisms form a category* $Alg^\circ(\boldsymbol{Sg})$.

Category of SP algebras over $\boldsymbol{Sg}$

Theorem 4.2. *Let* $\boldsymbol{Sg}$ *be a regular SP multi-language signature. Then,* $\mathscr{T}^\circ_{\boldsymbol{Sg}}$ *is the initial object of* $Alg^\circ(\boldsymbol{Sg})$.

Proof. We have to show that for each SP multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ there is a unique multi-language homomorphism $h\colon \mathscr{T}^{\circ}_{\boldsymbol{Sg}} \to \mathscr{A}$. Let $\mathscr{A}$ be such an algebra. Consider the new order-sorted $\boldsymbol{Sg}^{\circ}$-algebra $\mathscr{B}$ defined as follows:

- $[\![s]\!]_{\mathscr{B}} = [\![s]\!]_{\mathscr{A}_i}$ if $s \in S_i$ for some $i = 1, 2$;
- $[\![k_i]\!]_{\mathscr{B}} = [\![k]\!]_{\mathscr{A}_i}$ for each constant $k_i\colon s$ in $\boldsymbol{Sg}^{\circ}$ and for some $i = 1, 2$;
- $[\![f_i\colon w \to s]\!]_{\mathscr{B}} = [\![f\colon w \to s]\!]_{\mathscr{A}_i}$ for each function symbol $f\colon w \to s$ in Sg_i and for some $i = 1, 2$; and
- $[\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} = [\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$ in $\boldsymbol{Sg}$.

Proving that $\mathscr{B}$ is an order-sorted $\boldsymbol{Sg}^{\circ}$-algebra is easy, and we just prove the monotonicity of the conversion operator (that is, condition (3.2:v)). Let $\hookrightarrow\colon s_1 \to s_1'$ and $\hookrightarrow\colon s_2 \to s_2'$ with $s_1 \leqslant_i s_2$ for some $i = 1, 2$. Then,

$$\begin{aligned}
[\![\hookrightarrow\colon s_1 \to s_1']\!]_{\mathscr{B}}(x) &= [\![s_1 \ltimes s_1']\!]_{\mathscr{A}}(x) \\
&= [\![s_2 \ltimes s_2']\!]_{\mathscr{A}}(x) && \text{by cond. (4.8:i)} \\
&= [\![\hookrightarrow\colon s_2 \to s_2']\!]_{\mathscr{B}}(x)
\end{aligned}$$

Since $\boldsymbol{Sg}$ is regular by hypothesis, then $\boldsymbol{Sg}^{\circ}$ is regular by Definition 4.11, and therefore $\mathscr{T}_{\boldsymbol{Sg}^{\circ}}$ is initial in its category. Thus, there is a unique order-sorted $\boldsymbol{Sg}^{\circ}$-homomorphism $f\colon \mathscr{T}_{\boldsymbol{Sg}^{\circ}} \to \mathscr{B}$. Moreover, we can regard f as a multi-language $\boldsymbol{Sg}$-homomorphism h from $\mathscr{T}^{\circ}_{\boldsymbol{Sg}}$ to $\mathscr{A}$ by considering $h_i\colon (\mathscr{T}^{\circ}_{\boldsymbol{Sg}})_i \to \mathscr{A}_i$ the restriction of f to S_i, that is $h_i = f|_{S_i}$. Checking that h_i is an order-sorted Sg_i-homomorphism is trivial and omitted, and we only prove that h satisfies condition (4.3:ii): Suppose that $s \ltimes s'$ with $s \in S_i$ and $s' \in S_j$ where $i, j \in \{1, 2\}$ and $i \neq j$, then

$$\begin{aligned}
(h_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}} &= (h_j)_{s'} \circ [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} && \text{by def. of } [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} \\
&= f_{s'} \circ [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} && h_j = f|_{S_j} \\
&= [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} \circ f_s && f \text{ is an } \boldsymbol{Sg}^{\circ}\text{-homomorphism} \\
&= [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}} \circ (h_i)_s && h_i = f|_{S_i} \\
&= [\![s \ltimes s']\!]_{\mathscr{A}} \circ (h_i)_s && \text{by def. of } [\![\hookrightarrow\colon s \to s']\!]_{\mathscr{B}}
\end{aligned}$$

For the uniqueness part, suppose that $h'\colon \mathscr{T}^{\circ}_{\boldsymbol{Sg}} \to \mathscr{A}$ is a multi-language homomorphism. Now let g be the S-sorted function $g = h'_1 \cup h'_2$, that is

$$g_s = \begin{cases} (h'_1)_s & \text{if } s \in S_1 \\ (h'_2)_s & \text{if } s \in S_2 \end{cases}$$

(this is well-defined, since S_1 and S_2 are disjoint). It is immediate to see that g is also an order-sorted $\boldsymbol{Sg}^{\circ}$-homomorphism from $\mathscr{T}_{\boldsymbol{Sg}^{\circ}}$ to $\mathscr{B}$, and by uniqueness we have $g = f$. Hence $h' = h$ and the proof is concluded. □

The semantics of a term t induced by an SP multi-language algebra $\mathscr{A}$ is defined in the same way of ordinary multi-language algebras, thanks to the initiality result:

$$[\![t]\!]_{\mathscr{A}} = (h_i)_{ls(t)}(t) \tag{4.4}$$

where $h\colon \mathscr{T}^{\circ}_{\boldsymbol{Sg}} \to \mathscr{A}$ is the unique multi-language semantic function with $\boldsymbol{Sg}$ the SP signature. Moreover, the least sort of t is computed with respect to the associated signature $\boldsymbol{Sg}^{\circ}$.

Example 4.1. The boundary functions of the example 4.2 are subsort polymorphic:

$$[\![chr \ltimes nat]\!]_{\mathscr{A}}(\mathfrak{a}) = \mathrm{ord}(\mathfrak{a}) = [\![str \ltimes nat]\!]_{\mathscr{A}}(\mathfrak{a})$$

for each character $\mathfrak{a} \in \mathbb{A}$, and

$$[\![nat \ltimes chr]\!]_{\mathscr{A}} = [\![exp \ltimes chr]\!]_{\mathscr{A}}$$

by definition. Thus, the equivalent of the term t (see Equation 4.3) in the SP term algebra is

$$\hookrightarrow(+_2(\mathsf{f}_2, +_2(\mathsf{o}_2, \hookrightarrow(+_1(1\mathsf{0}_1, 5_1)))))$$

or, according to the previous notation,

$$\hookrightarrow(+(\mathsf{f}, +(\mathsf{o}, \hookrightarrow(+(1\mathsf{0}, 5)))))$$

and denoting the same natural number 765. ♢

4.6 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 the original languages Sg_1 and Sg_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 it is necessary, without the need of an explicit syntactical operator $\hookrightarrow$.
- Requiring a new form of *cross-language polymorphism* able to cope with shared operators among languages. Indeed, the main issue here is that the same operator might be present in both languages, but with different interpretation functions. The initiality of term algebras is then 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 Sg_1 models the syntax of a simple functional language (e.g., see [GTWW77, p. 77]) without an explicit encoding for string values, and Sg_2 is a language for manipulating strings (similar to the language $\mathcal{L}(Str)$ of the running example of this chapter), we can exploit the construction presented below in order to embed Sg_2 into Sg_1 without adding extra syntax.

4.6.1 Semantic-Only Multi-Language Signatures

A shared operator is an operator symbol which appears both in Sg_1 and Sg_2. For instance, in regards to the running example, the function symbol $+$ is a shared operator. Contrary to the previous cases where such ambiguity is removed by adding subscripts in the associated signature, we introduce a monotonicity requirements that shall lead to a notion of subsort polymorphism across signatures.

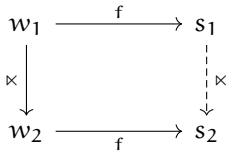


Figure 4.9: The graphical representation of the monotonicity requirement enforced by the semantic-only multi-language signature definition.

Semantic-only (SO) multi-language signature

Definition 4.12 (Semantic-Only Multi-Language Signature)**.** A *semantic-only (SO) multi-language signature* is a multi-language signature $\boldsymbol{Sg}$ satisfying the following additional constraints:

(i) $(S, \leqslant)$ is a poset, where $S = S_1 \cup S_2$ and $\leqslant = \leqslant_1 \cup \ltimes \cup \leqslant_2$; and

(ii) $f\colon w_1 \to s_1$ in Sg_i, $f\colon w_2 \to s_2$ in Sg_j, and $w_1 \ltimes w_2$ imply $s_1 \ltimes s_2$, with $i, j \in \{1, 2\}$ and $i \neq j$ (see Figure 4.9). ♢

Condition (4.12:i) forces the subsort relation to be directed, avoiding symmetricity of syntactic categories (which is typical when modeling language extensions), while condition (4.12:ii) shifts the monotonicity condition of order-sorted signature to shared operators.

We shall now define a proper notion of associated signature that does not carry extra symbols, except those in the underlying signatures. The key idea is to treat the interoperability constraints introduced by the join relation in the same way of ordinary subsort constraints.

Semantic-only (SO) associated signature

Definition 4.13 (Semantic-Only Associated Signature)**.** The *semantic-only (SO) associated signature* to the SO multi-language signature $\boldsymbol{Sg}$ is the order-sorted signature $\boldsymbol{Sg}^\diamond$ defined by

(i) $S = S_1 \cup S_2$ as set of sorts, ordered by $\leqslant = \leqslant_1 \cup \ltimes \cup \leqslant_2$;

(ii) $k\colon s$ is a constant in $\boldsymbol{Sg}^\diamond$ if $k\colon s$ is a constant in Sg_i; and

(iii) $f\colon w \to s$ is a function symbol in $\boldsymbol{Sg}^\diamond$ if $f\colon w \to s$ is a function symbol in Sg_i. ♢

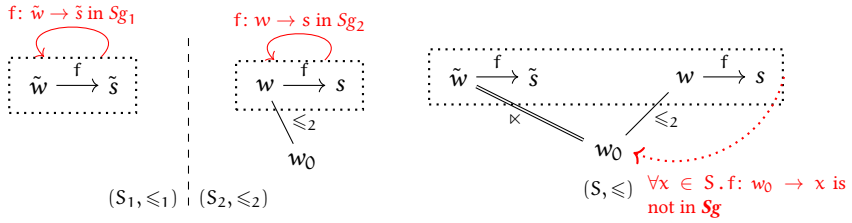
Consider now the order-sorted term algebra $\mathscr{T}_{\boldsymbol{Sg}^\diamond}$. If $s \ltimes s'$ in $\boldsymbol{Sg}$, then $s \leqslant s'$ in $\boldsymbol{Sg}^\diamond$. Then, by Rule (Subsort), $[\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}^\diamond}} \subseteq [\![s']\!]_{\mathscr{T}_{\boldsymbol{Sg}^\diamond}}$, and therefore we do not need an explicit operator $\hookrightarrow$ that acts as a bridge between syntactic categories.

Regularity of SO signatures

Consider the ordinary definition of regularity:

SO signature regularity

Definition 4.14 (Regularity)**.** A semantic-only multi-language signature $\boldsymbol{Sg}$ is *regular* if its associated signature $\boldsymbol{Sg}^\diamond$ is regular. ♢



(a) The Hasse-like diagrams of regular signatures Sg_1 (left) and Sg_2 (right).

(b) The Hasse-like diagram of the non-regular signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

Figure 4.10: A non-regular multi-language signature comprising two regular order-sorted signatures.

Unfortunately, multi-language regularity does not follow anymore from the regularity of the underlying languages and vice versa (see Figures 4.10 and 4.11).[1] More formally, Proposition 4.3 does not hold in this new context:

- Suppose $S_1 = \{\tilde{w}, \tilde{s}\}$, $S_2 = \{w_0, w, s\}$, $\leqslant_1$ and $\leqslant_2$ to be the reflexive relations on S_1 and S_2, respectively, plus $w_0 \leqslant_2 w$. Suppose also that $f\colon \tilde{w} \to \tilde{s}$ is a function symbol in Sg_1 and $f\colon w \to s$ a function symbol in Sg_2. If the cross-language interoperability relation $\ltimes$ is defined as $w_0 \ltimes \tilde{w}$ and $s \ltimes \tilde{s}$, the resulting associated signature is no longer regular (Figure 4.10(b)), although Sg_1 and Sg_2 are regular (Figure 4.10(a)). In particular, it is easy to see that $f\colon \tilde{w} \to \tilde{s}$ and $w_0 \leqslant w$ but the set $\{(w', s') \mid f\colon w' \to s' \wedge w_0 \leqslant w'\} = \{(\tilde{w}, \tilde{s}), (w, s)\}$ does not have a least element with respect to w_0.

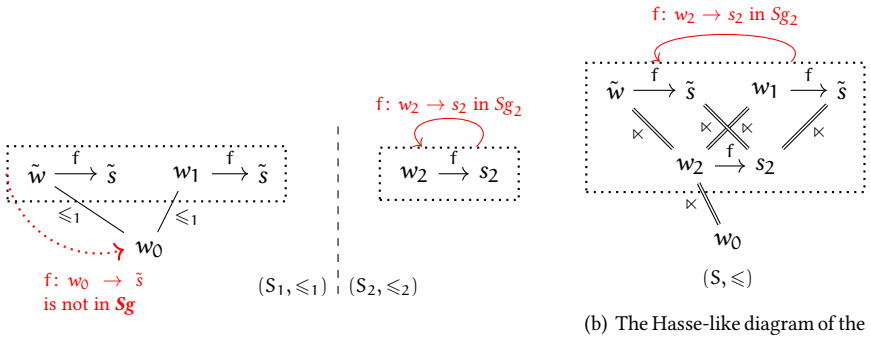
- On the other hand, let $S_1 = \{\tilde{w}, w_0, w_1, \tilde{s}\}$, $S_2 = \{w_2, s_2\}$, $\leqslant_1$ and $\leqslant_2$ be the reflexive relations on S_1 and S_2, respectively, plus $w_0 \leqslant_1 \tilde{w}$ and $w_0 \leqslant_1 w_1$, and $f\colon \tilde{w} \to \tilde{s}$ and $f\colon w_1 \to \tilde{s}$ in Sg_1. If the join relation $\ltimes$ is defined as $w_2 \ltimes \tilde{w}$, w_1, $w_0 \ltimes w_2$, and $s_2 \ltimes \tilde{s}$, the resulting associated signature is regular (Figure 4.11(b)), although Sg_1 is not (Figure 4.11(a)): Given $f\colon \tilde{w} \to \tilde{s}$ in $\boldsymbol{Sg}$ and $w_0 \leqslant \tilde{w}$, the set $\{(w', s') \mid f\colon w' \to s' \text{ in } \boldsymbol{Sg} \wedge w_0 \leqslant w'\} = \{(\tilde{w}, \tilde{s}), (w_1, \tilde{s}), (w_2, s_2)\}$ has least element (w_2, s_2) with respect to w_0 (Figure 4.11(b)).

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

Lemma 4.1 (Regularity over DCC poset [GM92])**.** *An order-sorted signature Sg over a DCC poset $(S, \leqslant)$ is regular if and only if whenever $f\colon w_1 \to s_1$ and $f\colon w_2 \to s_2$ in Sg and there is some $w_0 \leqslant w_1, w_2$, then there is some $w \leqslant w_1, w_2$ such that $f\colon w \to s$ and $w_0 \leqslant w$.*

At this point, we can relate the DCC of the poset $(S, \leqslant)$ in the associated signature of $\boldsymbol{Sg}$ to the DCC of $(S_1, \leqslant_1)$ and $(S_2, \leqslant_2)$:

[1] An (horizontal) arrow from a sequence of sorts w to a sort s labelled with a function symbol f is an alternative shorthand for $f\colon w \to s$. A (vertical) single line between two sorts s below s' labelled with a binary relation $\leqslant$ means that $s \leqslant s'$ (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 in order for the signature to be regular.



(a) The Hasse-like diagrams of signatures Sg_1 (non-regular, left) and Sg_2 (regular, right).

(b) The Hasse-like diagram of the regular multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

Figure 4.11: A regular multi-language signature comprising a non-regular order-sorted signature.

Proposition 4.7. *Let $\boldsymbol{Sg}^\diamond$ be the associated signature of the SO signature $\boldsymbol{Sg}$. Then, $(S, \leqslant)$ is DCC if and only if $(S_1, \leqslant_1)$ and $(S_2, \leqslant_2)$ are DCC.*

As a result, whenever we know that $(S_1, \leqslant_1)$ and $(S_2, \leqslant_2)$ are DCC, we can check the regularity of $\boldsymbol{Sg}$ by employing the Lemma 4.1 without checking whether $(S, \leqslant)$ is DCC.

Example 4.2. Consider the example illustrated in Figure 4.10(a). Both $(S_1, \leqslant_1)$ and $(S_2, \leqslant_2)$ satisfy the DCC, and therefore we can apply Lemma 4.1 to prove the non-regularity of the multi-language signature in Figure 4.10(b). Consider the function symbols $f\colon \tilde{w} \to \tilde{s}$ and $f\colon w \to s$ and the lower bound $w_0 \leqslant \tilde{w}, w$. Note that for each $w_0 \leqslant w' \leqslant \tilde{w}, w$ there is no $f\colon w' \to s'$ for any sort s'. ♢

4.6.2 Semantic-Only Multi-Language Algebras

In the multi-language construction presented in this section, the behaviour of boundary functions is no more bounded to syntactical operators as in the previous sections (*cf.* conditions (4.10:ii) and (4.5:ii)), but it is inherited by homomorphisms. A necessary condition to accomplish this aim is the commutativity of interpretation functions with boundary functions:

Semantic-only (SO) multi-language algebra

Definition 4.15 (Semantic-Only Multi-Language Algebra)**.** Let $\boldsymbol{Sg}$ be an SO multi-language signature. A *semantic-only (SO) multi-language $\boldsymbol{Sg}$-algebra* is an SP multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ such that

(i) $f\colon w_1 \to s_1$ in Sg_i, $f\colon w_2 \to s_2$ in Sg_j, and $w_1 \ltimes w_2$ imply

$$[\![s_1 \ltimes s_2]\!]_{\mathscr{A}} \circ [\![f\colon w_1 \to s_1]\!]_{\mathscr{A}_i} = [\![f\colon w_2 \to s_2]\!]_{\mathscr{A}_j} \circ [\![w_1 \ltimes w_2]\!]_{\mathscr{A}}$$

with $i, j \in \{1, 2\}$ and $i \neq j$. ♢

Note that $f\colon w_1 \to s_1$ in Sg_i, $f\colon w_2 \to s_2$ in Sg_j, and $w_1 \ltimes w_2$ imply $s_1 \ltimes s_2$ by condition (4.12:ii).

$$\text{(somlConst)}\ \frac{-}{k\colon s}\ (k\colon s \text{ in } Sg_j) \qquad \text{(somlSubsort)}\ \frac{t\colon s}{t\colon r}\ (s \leqslant_j r \text{ in } Sg_j)$$

$$\text{(somlFun)}\ \frac{(\forall 1 \leqslant i \leqslant n)\ \ t_i\colon s_i}{f(t_1,\ldots,t_n)\colon s}\ (f\colon s_1,\ldots,s_n \to s \text{ in } Sg_j)$$

$$\text{(somlInter)}\ \frac{t\colon s}{t\colon s'}\ (s \ltimes s' \text{ in } \boldsymbol{Sg})$$

Figure 4.12: Well-formed ground terms generated by a semantic-only multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

Remark 4.1. Please note that an SO multi-language algebra $\mathscr{A}$ is also an SP algebra by definition. That is, if $s_1 \ltimes s_1'$, $s_2 \ltimes s_2'$, and $s_1 \leqslant_i s_2$ for some $i = 1, 2$, then $[\![s_1 \ltimes s_1']\!]_{\mathscr{A}}(a) = [\![s_2 \ltimes s_2']\!]_{\mathscr{A}}(a)$. ♢

The notion of homomorphism between SO multi-language algebras remains unchanged from Definition 4.3.

4.6.3 Semantic-Only Term Algebra and Initial Semantics

Terms built out of an SO multi-language signature are those resulting from the rules in Figure 4.12. The reader may notice that constant terms and compound terms are no longer annotated with subscripts denoting the original language. Moreover, Rule (somlInter) makes a term t of sort s into a term of sort s', whenever $s \ltimes s'$. The term algebra over the current construction is defined similarly to Definition 4.5, except for boundary functions:

Definition 4.16 (Semantic-Only Multi-Language Term Algebra)**.** The *semantic-only (SO) multi-language term algebra* $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ over an SO signature $\boldsymbol{Sg}$ is defined as follows: Let $i = 1, 2$, then

Semantic-only (SO) multi-language term algebra

(i) the order-sorted Sg_i-algebra $(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i$ is given by

  (a) $[\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i} = [\![s]\!]_{\mathscr{T}_{Sg^{\diamond}}}$ for each $s \in Sg_i$;

  (b) $[\![k]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i} = [\![k]\!]_{\mathscr{T}_{Sg^{\diamond}}}$ for each constant k in Sg_i;

  (c) $[\![f\colon w \to s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i} = [\![f\colon w \to s]\!]_{\mathscr{T}_{Sg^{\diamond}}}$ for each $f\colon w \to s$ in Sg_i; and

(ii) boundary functions are given by $[\![s \ltimes s']\!]_{\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}} = \mathrm{id}_{[\![s]\!]_{\mathscr{T}_{Sg^{\diamond}}}}$ for each interoperability constraint $s \ltimes s'$ in $\boldsymbol{Sg}$. ♢

The definition of the boundary function $[\![s \ltimes s']\!]_{\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}}$ as the identity on the carrier set $\mathrm{id}_{[\![s]\!]_{\mathscr{T}_{Sg^{\diamond}}}}$ permits us to treat terms of sort s as if they had sort s' without converting them in a different syntactical form. Note that $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ trivially satisfies the commutativity condition (4.15:i), since boundary functions are defined as identities in the term algebra.

Proposition 4.8. *Let $\boldsymbol{Sg}$ be an SO multi-language signature. Then, the SO multi-language term $\boldsymbol{Sg}$-algebra is a proper SO multi-language algebra.*

Again, we can show that SO $\boldsymbol{Sg}$-algebras and $\boldsymbol{Sg}$-homomorphisms between them form a category denoted by $Alg^{\diamond}(\boldsymbol{Sg})$:

Category of SO algebras over $\boldsymbol{Sg}$

Proposition 4.9. *Let $\boldsymbol{Sg}$ be an SO multi-language signature. The class of all SO $\boldsymbol{Sg}$-algebras and the class of all $\boldsymbol{Sg}$-homomorphisms form a category $Alg^{\diamond}(\boldsymbol{Sg})$.*

We can now prove the initiality of $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ in its category.

Theorem 4.3. *Let $\boldsymbol{Sg}$ be a regular SO multi-language signature. Then, $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ is the initial object of $Alg^{\diamond}(\boldsymbol{Sg})$.*

Proof. Let $\mathscr{A}$ be an SO multi-language $\boldsymbol{Sg}$-algebra. Recall that $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ and $(S_i, \leqslant_i)$ is the poset of sorts of Sg_i, with $i = 1, 2$. Recall also that $(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i$ and $\mathscr{A}_i$ are the order-sorted Sg_i-algebra underlying $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ and $\mathscr{A}$, with carrier sets $(T^{\diamond}_{\boldsymbol{Sg}})_i$ and A_i, respectively. We first construct a pair of S_i-sorted function $h_i \colon (T^{\diamond}_{\boldsymbol{Sg}})_i \to A_i$ for any given multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$, then we show that h_i is a proper order-sorted Sg_i-homomorphism, and finally we prove its uniqueness.

1. (Construction) Let t be an order-sorted term in the carrier set $(T^{\diamond}_{\boldsymbol{Sg}})_i$ with sort s, that is, $t \in [\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}$. We define $(h_i)_s$ by rule induction on $t \colon s$:

   a) (Const) If t has been derived by Rule (Const), then there is $k \colon s$ in Sg_i with s the least sort of t. Thus, we define

   $$(h_i)_s(k) = [\![k]\!]_{\mathscr{A}_i} \tag{4.5}$$

   b) (Fun) If t has been derived by Rule (Fun), then $t = f(t_1, \ldots, t_n)$ for some function symbol $f \colon s_1, \ldots, s_n \to s$ in Sg_i. We define

   $$\begin{aligned}&(h_i)_s(f(t_1, \ldots, t_n))\\ &\qquad = [\![f \colon s_1, \ldots, s_n \to s]\!]_{\mathscr{A}_i}((h_i)_{s_1}(t_1), \ldots, (h_i)_{s_n}(t_n))\end{aligned} \tag{4.6}$$

   c) (Sub) If t has been derived by Rule (Sub), then there is a sort s' such that $s' \leqslant s$. Thus, we define

   $$(h_i)_s(t) = \begin{cases} (h_i)_{s'}(t) & s' \leqslant_i s \\ [\![s' \ltimes s]\!]_{\mathscr{A}}((h_i)_{s'}(t)) & s' \ltimes s \end{cases} \tag{4.7}$$

   Note that by definition of $\leqslant$ in the associated signature (condition (4.13:i)), if $s' \leqslant s$, then either $s' \leqslant_j s$ for some $j = 1, 2$ or $s' \ltimes s$. Since $t \in [\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}$ by hypothesis, it follows that $i = j$ and then $(h_i)_s$ is fully defined.

2. (Correctness) We first prove that the S_i-sorted function $h_i \colon (T^{\diamond}_{\boldsymbol{Sg}})_i \to A_i$ is indeed an Sg_i-homomorphism $h_i \colon (\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i \to \mathscr{A}_i$.

   a) (Condition (3.4:i)) Let $k \colon s$ be a constant in Sg_i. Then,

   $$\begin{aligned}(h_i)_s([\![k]\!]_{\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}) &= (h_i)_s(k)\\ &= [\![k]\!]_{\mathscr{A}_i} \qquad \text{by Eq. 4.5}\end{aligned}$$

Let $f\colon w \to s$ be a function symbol in Sg_i with $w = s_1 \dots s_n$. Then, for each tuple of ground terms $t_1 : s_1, \dots, t_n : s_n$,

$$\begin{aligned}
&\big((h_i)_s \circ [\![f\colon w \to s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}\big)(t_1, \dots, t_n) \\
&\quad = (h_i)_s(f(t_1, \dots, t_n)) \\
&\quad = [\![f\colon w \to s]\!]_{\mathscr{A}_i}((h_i)_{s_1}(t_1), \dots, (h_i)_{s_n}(t_n)) \qquad \text{by Eq. 4.6} \\
&\quad = \big([\![f\colon w \to s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i} \circ (h_i)_w\big)(t_1, \dots, t_n)
\end{aligned}$$

b) (Condition (3.4:ii)) Let $s \leqslant_i r$ in Sg_i. Then, by Equations 4.7 it is immediate that $(h_i)_s(t) = (h_i)_r(t)$ for each term t in $[\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}$.

In order to complete the correctness part of the proof, we need to check that h is a multi-language homomorphism by proving condition (4.3:ii). Suppose $s \ltimes s'$ with $s \in S_i$ and $s' \in S_j$ where $i, j \in \{1, 2\}$ and $i \neq j$. Let t in $[\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}$. By construction,

$$\begin{aligned}
\big((h_j)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}\big)(t) &= (h_j)_{s'}(t) && \text{by cond. (4.16:ii)} \\
&= [\![s \ltimes s']\!]_{\mathscr{A}}((h_i)_s(t)) && \text{by Eq. 4.7} \\
&= \big([\![s \ltimes s']\!]_{\mathscr{A}} \circ (h_i)_s\big)(t)
\end{aligned}$$

3. (Uniqueness) Suppose that $h'\colon \mathscr{T}^{\diamond}_{\boldsymbol{Sg}} \to \mathscr{A}$ is an $\boldsymbol{Sg}$-homomorphism. We shall show that $h' = h$. Let $t \in [\![s]\!]_{(\mathscr{T}^{\diamond}_{\boldsymbol{Sg}})_i}$. The proof is an easy rule induction on $t\colon s$ to prove that $(h'_i)_s(t) = (h_i)_s(t)$.

□

Thanks to the initiality of the term algebra, the semantics of a term t is defined as usual: Let $t\colon s$ be a term in $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$, $\mathscr{A}$ an SO multi-language algebra, and $h\colon \mathscr{T}^{\diamond}_{\boldsymbol{Sg}} \to \mathscr{A}$ the unique homomorphism defined in the proof of Theorem 4.3. Then,

$$[\![t]\!]_{\mathscr{A}} = (h_i)_{\mathrm{ls}(t)}(t) \tag{4.8}$$

with $\mathrm{ls}(t)$ the least sort of t in the associated signature $\boldsymbol{Sg}^{\diamond}$.

Running Example 4.7. Let $\mathscr{E}$ and $\mathscr{S}$ be the order-sorted algebras over the signatures $Sg(Exp)$ and $Sg(Str)$, respectively, defined in Running Example 4.1. Suppose we are interested in a new multi-language signature $\boldsymbol{Sg}$ over $Sg(Exp)$ and $Sg(Str)$ and a multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ such that any string expressions t with sort *str* in $Sg(Str)$ can denote the natural number $\mathrm{len}([\![t]\!]_{\mathscr{S}})$, where len gives the length of the string, when used in place of $Sg(Exp)$ terms. For instance, we require that $[\![+(10, 5)]\!]_{\mathscr{A}} = [\![+(10, 5)]\!]_{\mathscr{E}} = 15$ and $[\![+(\mathtt{f}, \mathtt{o})]\!]_{\mathscr{A}} = [\![+(\mathtt{f}, \mathtt{o})]\!]_{\mathscr{S}} = \mathtt{fo}$, but $[\![+(+(\mathtt{f}, \mathtt{o}), +(10, 5))]\!]_{\mathscr{A}} = [\![+(\mathtt{fo}, 15)]\!]_{\mathscr{A}} = 17$.

Since the requirements demand to use string expressions in place of natural numbers, the join relation $\ltimes$ shall define *str* $\ltimes$ *nat* and ensure transitivity, hence

$$\mathit{str} \ltimes \mathit{exp} \qquad \mathit{chr} \ltimes \mathit{nat} \qquad \mathit{chr} \ltimes \mathrm{exp}$$

Therefore, the new multi-language signature $\boldsymbol{Sg} = (Sg(Exp), Sg(Str), \ltimes)$ is trivially an SO signature.

Furthermore, the signatures $Sg(Exp)$ and $Sg(Str)$ are trivially regular. However, by blending them, we are generating subsort polymorphism on the function symbol

$+$, which is used as sum operator in $Sg(Exp)$ and as concatenation operator in $Sg(Str)$, and therefore we have to check the regularity of the SO multi-language signature $\boldsymbol{Sg}$. This amounts to employ Lemma 4.1 (thanks to Proposition 4.7) by setting the lower bound w_0 to $chr\,chr$. Now let $w_1 = exp\,exp$ and $w_2 = str\,str$. Since $w_0 \leqslant w_1, w_2$, we need to check the existence (w, s) such that $w_0 \leqslant w \leqslant w_1$ and $w_0 \leqslant w \leqslant w_2$ with $+\colon w \to s$ and $s \leqslant exp, chr$. Such a rank (w, s) exists and it is given by $w = str\,str$ and $s = str$. Other cases with different lower bounds are treated in a similar way. Therefore, $\boldsymbol{Sg}$ is a regular SO signature.

The multi-language $\boldsymbol{Sg}$-algebra $\mathscr{A}$ is now defined by taking $\mathscr{E}$ and $\mathscr{S}$ as underlying algebras, and by defining boundary functions $[\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$:

$$[\![chr \ltimes nat]\!]_{\mathscr{A}}(a) = [\![chr \ltimes exp]\!]_{\mathscr{A}}(a) = 1$$
$$[\![str \ltimes nat]\!]_{\mathscr{A}}(a_1 \dots a_n) = [\![str \ltimes exp]\!]_{\mathscr{A}}(a_1 \dots a_n) = \mathrm{len}(a_1 \dots a_n) = n$$

The above definition of boundary functions satisfies both conditions (4.8:i) and (4.15:i), thus $\mathscr{A}$ is a proper SO $\boldsymbol{Sg}$-algebra.

The initiality theorem (Theorem 4.3) yields the semantic homomorphism from $\mathscr{T}^{\diamond}_{\boldsymbol{Sg}}$ to $\mathscr{A}$. For instance, suppose we want to compute the multi-language semantics of the term

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

The least sorts of t, t_1, and t_2 are exp, str, and exp, respectively. The operator $+$ belongs to both $Sg(Exp)$, the addition $+\colon exp, exp \to exp$, and $Sg(Str)$, the concatenation $+\colon str, str \to str$. Its least rank with respect to the lower bound $\mathrm{ls}(t_1)\,\mathrm{ls}(t_2) = str\,exp$ is the pair $(exp\,exp, exp)$. By Equation 4.8 we have

$$[\![t]\!]_{\mathscr{A}} = (h_1)_{exp}(t) = [\![+\colon exp, exp \to exp]\!]_{\mathscr{E}}((h_1)_{exp}(t_1), (h_1)_{exp}(t_2))$$

Since $\mathrm{ls}(t_1) = str$ and $\mathrm{ls}(f) = \mathrm{ls}(o) = chr$, then the least rank of $+$ in t_1 with respect to the lower bound $\mathrm{ls}(f)\,\mathrm{ls}(o) = chr\,chr$ is precisely $(str\,str, str)$, thus

$$\begin{aligned}(h_1)_{exp}(t_1) &= [\![str \ltimes exp]\!]_{\mathscr{A}}((h_2)_{str}(t_1)) \\ &= [\![str \ltimes exp]\!]_{\mathscr{A}}([\![+\colon str, str \to str]\!]_{\mathscr{S}}((h_2)_{str}(f), (h_2)_{str}(o))) \\ &= [\![str \ltimes exp]\!]_{\mathscr{A}}([\![+\colon str, str \to str]\!]_{\mathscr{S}}(f, o)) \\ &= [\![str \ltimes exp]\!]_{\mathscr{A}}(fo) \\ &= 2\end{aligned}$$

Similarly, $\mathrm{ls}(t_2) = exp$ and $\mathrm{ls}(10) = \mathrm{ls}(5) = nat$. Then the least rank of $+$ in t_1 with respect to the lower bound $\mathrm{ls}(10)\,\mathrm{ls}(5) = nat\,nat$ is $(exp\,exp, exp)$, and therefore we have

$$\begin{aligned}(h_1)_{exp}(t_2) &= [\![+\colon exp, exp \to exp]\!]_{\mathscr{E}}((h_1)_{exp}(10), (h_1)_{exp}(5)) \\ &= [\![+\colon exp, exp \to exp]\!]_{\mathscr{E}}(10, 5) \\ &= 15\end{aligned}$$

Finally,

$$\begin{aligned}[\![t]\!]_{\mathscr{A}} &= [\![+\colon exp, exp \to exp]\!]_{\mathscr{E}}((h_1)_{exp}(t_1), (h_1)_{exp}(t_2)) \\ &= [\![+\colon exp, exp \to exp]\!]_{\mathscr{E}}(2, 15) \\ &= 17\end{aligned}$$

as desired. We can notice that without any conversion operator the framework is still able to apply the correct boundary functions to move values across languages. ♢

4.7 Combining Untyped and Simply-Typed Lambda-Calculi

The first theoretical paper addressing the challenge of multi-language construction is [MF09]. The authors study the so-called *natural embedding*, in which they 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. 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 formalisation of the untyped and simply-typed lambda-calculi, then we provide suitable boundary functions able to move values across languages.

In Example 3.6 we have formalised the untyped lambda-calculus in detail. Here, we do the same with the simply-typed lambda-calculus.

Example 4.3. We begin by defining types. Let *nat* be the sort of natural numbers. We inductively define the set of *simple types* T as the smallest set satisfying

- $nat \in \mathrm{T}$, and
- $\tau_1, \tau_2 \in \mathrm{T}$ implies $\tau_1 \to \tau_2 \in \mathrm{T}$.

The order-sorted signature of simply-typed lambda-calculus $Sg(\lambda^{\to})$ is defined by taking $\mathrm{S}(\lambda^{\to}) = \mathrm{T}$ as the set of sorts, equipped with the discrete order.

Recall that *Var* is a countably infinite set of variables (see Example 3.6). The set of (typed) variables of simply-typed lambda-calculus is $Var^{\to} = Var \times \mathrm{T}$ with instances denoted by $x^{\tau} \in Var^{\to}$. Then, the operators in $Sg(\lambda^{\to})$ are

$n \colon nat$	constant (for each $n \in \mathbb{N}$)
$x^{\tau} \colon \tau$	variable (for each $x^{\tau} \in Var^{\to}$)
$\lambda_{x^{\tau}} \colon \tau' \to (\tau \to \tau')$	abstraction over x^{τ} (for each $x^{\tau} \in Var^{\to}$)
$\circ \colon (\tau \to \tau'), \tau \to \tau'$	application (for each $\tau, \tau' \in \mathrm{T}$)

Since the order on the set of sorts $\mathrm{S}(\lambda^{\to})$ is discrete, the signature $Sg(\lambda^{\to})$ is trivially order-sorted.

We proceed now to define an order-sorted algebra $\mathscr{S}$ providing $Sg(\lambda^{\to})$-terms with a meaning. Let $V^{\to}$ be the T-sorted family

$$V^{\to} = (\, V^{\to}_{\tau} \mid \tau \in \mathrm{T} \,) \qquad \text{with} \qquad \begin{cases} V^{\to}_{nat} = \mathbb{N} \\ V^{\to}_{\tau_1 \to \tau_2} = V^{\to}_{\tau_1} \to V^{\to}_{\tau_2} \end{cases}$$

of *values*, and $Env^{\to} = \{\, \rho \colon Var^{\to} \to \bigcup V^{\to} \mid (\forall x^{\tau} \in Var^{\to})(\rho(x^{\tau}) \in V^{\to}_{\tau}) \,\}$ the set of *well-typed environments*. We define the carrier set of $\mathscr{S}$ by taking

$$[\![\tau]\!]_{\mathscr{S}} = Env^{\to} \to V^{\to}_{\tau}$$

and we define interpretation functions

$$\begin{aligned}
[\![n]\!]_{\mathscr{S}} &= \rho \mapsto n \\
[\![x^{\tau}]\!]_{\mathscr{S}} &= \rho \mapsto \rho(x^{\tau}) \\
[\![\lambda_{x^{\tau}} \colon \tau' \to (\tau \to \tau')]\!]_{\mathscr{S}}(m) &= \rho \mapsto v \in V^{\to}_{\tau} \mapsto m(\rho[x^{\tau} \leftarrow\!\shortmid v]) \\
[\![\circ \colon (\tau \to \tau'), \tau \to \tau']\!]_{\mathscr{S}}(m_1, m_2) &= \rho \mapsto (m_1(\rho))(m_2(\rho))
\end{aligned}$$

Therefore we have a unique semantic function $eval^{\to} \colon \mathscr{T}_{Sg(\lambda^{\to})} \to \mathscr{S}$ providing $Sg(\lambda^{\to})$-terms with a meaning in $\mathscr{S}$. ♢

Note that the definition of the interpretation sets $[\![\tau]\!]_{\mathscr{S}}$ is very loose, in the sense that there are elements $m \in [\![\tau]\!]_{\mathscr{S}}$ which are not the image of any $Sg(\lambda^{\to})$-term. In the following we will make use of the homomorphic image $eval^{\to}(\mathscr{T}_{Sg(\lambda^{\to})})$ given by $eval^{\to}$ (see Proposition 3.8), as formalised in the next example.

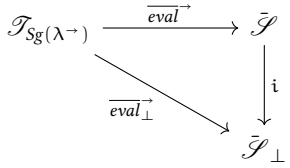
Example 4.4. We denote by $\bar{\mathscr{S}}$ the homomorphic image of $\mathscr{T}_{Sg(\lambda^{\to})}$ given by $eval^{\to}$, that is $\bar{\mathscr{S}} = eval^{\to}(\mathscr{T}_{Sg(\lambda^{\to})})$. We recall the inner definition of such an algebra. The interpretation sets $[\![\tau]\!]_{\bar{\mathscr{S}}}$ are given by the only elements in $[\![\tau]\!]_{\mathscr{S}}$ reachable by at least one $\lambda^{\to}$-term, that is

$$[\![\tau]\!]_{\bar{\mathscr{S}}} = \{\, m \in [\![\tau]\!]_{\mathscr{S}} \mid (\exists t \in [\![\tau]\!]_{\mathscr{T}_{Sg(\lambda^{\to})}})([\![t]\!]_{\mathscr{S}} = m) \,\}$$

Then, interpretation functions of $\bar{\mathscr{S}}$ are those of $\mathscr{S}$ restricted to the above interpretations sets. The resulting (unique) homomorphism out of the term algebra is denoted by $\overrightarrow{eval} \colon \mathscr{T}_{Sg(\lambda^{\to})} \to \bar{\mathscr{S}}$. ♢

Suppose we want to define a multi-language $\boldsymbol{Sg}(Sg(\lambda) + Sg(\lambda^{\to}))$ over the order-sorted signatures $Sg(\lambda)$ and $Sg(\lambda^{\to})$ interpreted by algebras $\mathscr{E}$ and $\bar{\mathscr{S}}$, respectively. The main issues here is that there is no guarantee that type soundness is preserved in the $\boldsymbol{Sg}(Sg(\lambda) + Sg(\lambda^{\to}))$-calculus, since untyped lambda-calculus is Turing-complete, whereas the simply-typed lambda calculus is not. Therefore, in order to proper handle the conversion of a value in $[\![exp]\!]_{\mathscr{E}}$ into $[\![\tau]\!]_{\bar{\mathscr{S}}}$ we need to extend the carrier sets of the simply-typed lambda-calculus with a new value $\bot$, representing the notion of *stuck state* or *non-termination*.

Example 4.5. Let $[\![\tau]\!]_{\bar{\mathscr{S}}_\bot} = [\![\tau]\!]_{\bar{\mathscr{S}}} \cup \{\bot\}$, and let the new interpretation functions $[\![-]\!]_{\bar{\mathscr{S}}_\bot}$ to output $\bot$ if one of their arguments is $\bot$ and the result of $[\![-]\!]_{\bar{\mathscr{S}}}$ otherwise, for each operator symbol. Thus we have a new semantic function $\overrightarrow{eval_\bot} \colon \mathscr{T}_{Sg(\lambda^{\to})} \to \bar{\mathscr{S}}_\bot$. It is trivial to check the commutativity of the following diagram, where $i \colon \bar{\mathscr{S}} \to \bar{\mathscr{S}}_\bot$ is the set-inclusion:



Moreover, we define

$$\bar{V}^{\rightarrow}_{\perp} = (\,(\bar{V}^{\rightarrow}_{\perp})_{\tau} \mid \tau \in T\,) \qquad \text{with} \qquad \begin{cases} (\bar{V}^{\rightarrow}_{\perp})_{nat} = [\![nat]\!]_{\bar{\mathscr{S}}_{\perp}} = \mathbb{N}_{\perp} = \mathbb{N} \cup \{\perp\} \\ (\bar{V}^{\rightarrow}_{\perp})_{\tau_1 \to \tau_2} = [\![\tau_1]\!]_{\bar{\mathscr{S}}_{\perp}} \to [\![\tau_2]\!]_{\bar{\mathscr{S}}_{\perp}} \end{cases}$$

and $\overline{Env}^{\rightarrow}_{\perp} = \left\{\, \rho\colon Var^{\rightarrow} \to \bigcup \bar{V}^{\rightarrow}_{\perp} \;\middle|\; (\forall x^{\tau} \in Var^{\rightarrow})(\rho(x^{\tau}) \in (\bar{V}^{\rightarrow}_{\perp})_{\tau}) \,\right\}$. ♢

We can now blend the two lambda-calculi in the new multi-language signature $\boldsymbol{Sg}(Sg(\lambda) + Sg(\lambda^{\rightarrow}))$, and provide meaning to interoperability constraint by defining a multi-language $\boldsymbol{Sg}(Sg(\lambda) + Sg(\lambda^{\rightarrow}))$-algebra $\mathscr{M}$.

Example 4.6. We would like to use λ-terms in place of $\lambda^{\rightarrow}$-terms and vice versa. Therefore, we define the following interoperability constraints:

$$(\forall \tau \in T)(exp \ltimes \tau) \qquad \text{and} \qquad (\forall \tau \in T)(\tau \ltimes exp)$$

In order to define the corresponding boundary functions, we need ways to convert $\lambda^{\rightarrow}$-environments to λ-environment and back. Let

$$Var \underset{\alpha}{\overset{\beta}{\leftrightarrows}} Var^{\rightarrow}$$

be any isomorphisms between these sets (it trivially exists, since both $Var^{\rightarrow} = Var \times T$ and Var are countable). We shall extend such functions α and β to $\dot{\alpha}$ and $\dot{\beta}$ between environments:

$$\begin{array}{ccc} Env & \underset{\dot{\alpha}}{\overset{\dot{\beta}}{\leftrightarrows}} & \overline{Env}^{\rightarrow}_{\perp} \\ \| & & \| \\ (Var \to V) & \underset{\dot{\alpha}}{\overset{\dot{\beta}}{\leftrightarrows}} & (Var^{\rightarrow} \to \bigcup \bar{V}^{\rightarrow}_{\perp}) \end{array}$$

In order to define $\dot{\alpha}$ and $\dot{\beta}$ we introduce the following restriction operator

$$-|_{\tau_1 \to \tau_2} : [V \to V] \to (\bar{V}^{\rightarrow}_{\perp})_{\tau_1 \to \tau_2}$$

which moves continuous functions in $[V \to V]$ to functions in $(\bar{V}^{\rightarrow}_{\perp})_{\tau_1 \to \tau_2} = (\bar{V}^{\rightarrow}_{\perp})_{\tau_1} \to (\bar{V}^{\rightarrow}_{\perp})_{\tau_2}$ whenever there are suitable translations. The definition is recursive on the sort $\tau_1 \to \tau_2$, and we write function composition as arrow sequence to

highlight corresponding domains and codomains:

$$\mathsf{f}|_{nat\to nat} = \left((\bar{V}^{\to}_{\perp})_{nat} = \mathbb{N}_\perp\right) \xrightarrow{\iota_{\mathbb{N}_\perp}} V \xrightarrow{\mathsf{f}} V \xrightarrow{\pi_{\mathbb{N}_\perp}} \left(\mathbb{N}_\perp = (\bar{V}^{\to}_{\perp})_{nat}\right)$$

$$\mathsf{f}|_{nat\to(\tau_1\to\tau_2)} = \left(\left((\bar{V}^{\to}_{\perp})_{nat} = \mathbb{N}_\perp\right) \xrightarrow{\iota_{\mathbb{N}_\perp}} V \xrightarrow{\mathsf{f}} V \xrightarrow{\pi_{[V\to V]}} [V \to V]\right)\Big|_{\tau_1\to\tau_2}$$

$$\mathsf{f}|_{(\tau_1\to\tau_2)\to nat} = \left((\bar{V}^{\to}_{\perp})_{\tau_1} \to (\bar{V}^{\to}_{\perp})_{\tau_2} \subseteq [V \to V]\right) \xrightarrow{\iota_{[V\to V]}} V \xrightarrow{\mathsf{f}} V \xrightarrow{\pi_{\mathbb{N}_\perp}} \left(\mathbb{N}_\perp = (\bar{V}^{\to}_{\perp})_{nat}\right)$$

$$\mathsf{f}|_{(\tau_1\to\tau_2)\to(\tau_3\to\tau_4)} = \left(\left((\bar{V}^{\to}_{\perp})_{\tau_1} \to (\bar{V}^{\to}_{\perp})_{\tau_2} \subseteq [V \to V]\right) \xrightarrow{\iota_{[V\to V]}} V \xrightarrow{\mathsf{f}} V \xrightarrow{\pi_{[V\to V]}} [V \to V]\right)\Big|_{\tau_3\to\tau_4}$$

Let $Var^{\to}_{\tau} = \{x^{\tau} \mid x \in Var\}$. We can now define the transformation of a $Sg(\lambda)$-environment to a $Sg(\lambda^{\to})$-environment by letting $\dot{\alpha}(\eta) = x^{\tau} \mapsto \dot{\alpha}^{\eta}_{\tau}(x^{\tau})$ where $\dot{\alpha}^{\eta}_{\tau} \colon Var^{\to}_{\tau} \to (\bar{V}^{\to}_{\perp})_{\tau}$ is

$$\dot{\alpha}^{\eta}_{nat} = \left(Var^{\to}_{\tau} \subseteq Var^{\to}\right) \xrightarrow{\beta} Var \xrightarrow{\eta} V \xrightarrow{\pi_{\mathbb{N}_\perp}} \left(\mathbb{N}_\perp = (\bar{V}^{\to}_{\perp})_{nat}\right)$$

$$\dot{\alpha}^{\eta}_{\tau_1\to\tau_2} = \left(\left(Var^{\to}_{\tau} \subseteq Var^{\to}\right) \xrightarrow{\beta} Var \xrightarrow{\eta} V \xrightarrow{\pi_{[V\to V]}} [V \to V]\right)\Big|_{\tau_1\to\tau_2}$$

Conversely, we define $\dot{\beta} \colon \overline{Env}^{\to}_{\perp} \to Env$ by

$$\dot{\beta} = \rho \in \overline{Env}^{\to}_{\perp} \mapsto x \in Var \mapsto \iota_X(\rho(\alpha(x)))$$

for the unique $X \in \{\mathbb{N}_\perp, [V \to V]\}$ such that $\rho(\alpha(x)) \in X$. Note that the definition of $\dot{\beta}$ is simpler than $\dot{\alpha}$. Indeed, moving values from the simply-typed to the untyped lambda calculus is far easier, since every $Sg(\lambda^{\to})$-value admits a denotation in the semantic domain of $Sg(\lambda)$.

Finally, we can define the boundary functions of the multi-language. Suppose we need to move a $Sg(\lambda)$-denotation e at a sort τ in $Sg(\lambda^{\to})$. Then,

$$[\![exp \ltimes nat]\!]_{\mathscr{M}}(e) = \dot{\beta} \circ e \circ \pi_{\mathbb{N}_\perp}$$
$$[\![exp \ltimes \tau_1 \to \tau_2]\!]_{\mathscr{M}}(e) = \left(\dot{\beta} \circ e \circ \pi_{[V\to V]}\right)\Big|_{\tau_1\to\tau_2}$$

that is, in order to move a $Sg(\lambda)$-value e into the simply-typed lambda-calculus $Sg(\lambda^{\to})$ at a type τ, we first run e on the conversion $\dot{\beta}(\rho)$ of the environment ρ, then we convert the resulting value to a valid $Sg(\lambda^{\to})$-value (that would be $\perp$, in the worst case scenario).

On the other hand, we define

$$[\![\tau \ltimes exp]\!]_{\mathscr{M}}(m) = \eta \in Env \mapsto \iota_X(m(\dot{\alpha}(\eta)))$$

for the unique $X \in \{\mathbb{N}_\perp, [V \to V]\}$ such that $m(\dot{\alpha}(\eta)) \in X$. ♢

CHAPTER 5

Multi-Language Equational Logic

Chapter reference(s): [BCM20b]

In the previous chapter, we introduced *multi-language algebras* as an extension of *order-sorted algebras* for modeling the combination of already existing languages, the so-called *multi-languages*. On the other hand, *equational logic* is the natural logic in the field of universal algebra since its early days. In this chapter, we address the problem of *equational reasoning* in a multi-language context, where equations are no longer defined between terms of the same *(single-) language*, but between *multi-language terms* generated by different signatures.

In particular, we focus on equational algebras [Tar68, Bir35, GM82, Hig63, GM92]. Inspired by the classic results of [Bir35], we develop in detail a number of analogues of standard concepts such as *algebra congruences* and *quotients*. These are put to use in a completeness proof for a system of equational reasoning in the multi-language setting. In general, we try to follow the same path of Sections 3.5 *et seq.* in a slightly more compact presentation.

Summarising, the purpose of this section is to provide an equational logic in a multi-language context. In particular, (i) we define equations between multi-language terms; then, (ii) we prove the rules of order-sorted equational deduction are still sound and complete for deriving all unconditional multi-language equations; and (iii) we show the existence of a free algebra in each class of models over a coherent signature.

Structure We introduce *free algebras* for a multi-language signature, in order to gain terms with variables and substitution homomorphisms. Then, we define the standard concepts of *congruences*, *kernel* of a homomorphisms, and *subalgebras* in the multi-language context. We proceed by defining the quotienting of a multi-language algebra by a congruence relation. Finally, we introduce the equational reasoning for multi-language theories, and in the last section we prove its *soundness* and *completeness*.

5.1 Multi-Language Free Algebra

To formally define the terms with variables of our multi-languages we shall make use of the concept of *free algebras*. First we review the abstract concept, and then we show that "multi-language terms built using variables" are a concrete instance of a free algebra, as is already the case in the order-sorted world (see Section 3.5). The universal property of a free algebra provides a convenient tool for defining and working with term substitutions.

Let S be the set of sorts of the multi-language, that is $S = S_1 \cup S_2$ (recall Notation 4.1). A *free algebra* is the loosest algebra generated from an S-sorted X (whose elements are understood as *variables*). As usual [RT91, Pow85], the free algebra yields *terms with variables* (as opposed to *ground terms* in the term algebra).

Notation 5.1. We first introduce some notation for multi-language homomorphisms, in order to simplify the exposition. Let $h\colon \mathscr{A} \to \mathscr{B}$ be a multi-language $\boldsymbol{Sg}$-homomorphisms, with $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ some multi-language signature. In the following, we set $S = S_1 \cup S_2$ the union of the sorts of the order-sorted signatures Sg_1 and Sg_2. We define the S-sorted function $\overline{h}\colon A \to B$, where $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$ (we recall that A_i and B_i are the carrier sets of the order-sorted algebras $\mathscr{A}_i$ and $\mathscr{B}_i$ underlying $\mathscr{A}$ and $\mathscr{B}$, respectively), such that for each $s \in S_i$

$$\begin{aligned} \overline{h}_s\colon [\![s]\!]_{\mathscr{A}_i} &\to [\![s]\!]_{\mathscr{B}_i} \\ a &\mapsto (h_i)_s(a) \end{aligned}$$

Moreover, we write A_s or $[\![s]\!]_{\mathscr{A}}$ for $[\![s]\!]_{\mathscr{A}_i}$ if $s \in S_i$. ♢

Lemma 5.1. *The mapping* $h \mapsto \overline{h}$ *is well-defined, a bijection, and functorial in that* $\overline{h \circ h'} = \overline{h} \circ \overline{h'}$.

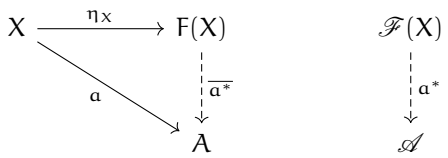
Proof. Immediate from the definitions. □

Notation 5.2. Note that we will usually write h for $\overline{h}$, thus identifying the two concepts, and regard $h\colon \mathscr{A} \to \mathscr{B}$ and $\overline{h}\colon A \to B$ as inter-changeable throughout the chapter. ♢

Free multi-language algebra

Definition 5.1 (Free Multi-Language Algebra)**.** Let $\boldsymbol{Sg}$ be a multi-language signature and X an S-sorted set of variables. A *free multi-language* $\boldsymbol{Sg}$*-algebra* $\mathscr{F}(X)$ over X is an $\boldsymbol{Sg}$-algebra $\mathscr{F}(X)$ together with an S-sorted function $\eta_X\colon X \to F(X)$, with $F(X)$ the carrier set of $\mathscr{F}(X)$, such that the following universal property is satisfied:

(i) for each $\boldsymbol{Sg}$-algebra $\mathscr{A}$, if $a\colon X \to A$ is an S-sorted function, then there exists a unique multi-language $\boldsymbol{Sg}$-homomorphism $a^*\colon \mathscr{F}(X) \to \mathscr{A}$ making the following diagram commute:



(5.1)

♢

The previous definition does not guarantee the existence of such a free $\boldsymbol{Sg}$-algebra over X. However, if it does exist it is unique up to a unique isomorphism. We now provide a two-step syntactic construction, establishing existence. See, for instance, [DW09, Definition 1.3.2], [GTWW77, p. 72], and [GM92, p. 14] for a free algebra construction in the one-sorted, many-sorted, and order-sorted worlds. We now define a specific instance of $\mathscr{F}(X)$, denoted by $\mathscr{T}_{\boldsymbol{Sg}}(X)$, and constructed out of syntax.

Definition 5.2 ((Canonical) Free Multi-Language Algebra)**.** Let $\boldsymbol{Sg}$ be a signature and X an S-sorted family of variables $(X_s \mid s \in S)$, each component also disjoint from the symbols in $\boldsymbol{Sg}$.

1. By setting $(X_i)_s = X_s$ and $X_i = \big((X_i)_s \;\big|\; s \in S_i\big)$ we obtain an S_i-sorted set of variables, for $i = 1, 2$. We define the new multi-language signature $\boldsymbol{Sg}(X) = (Sg_1(X_1), Sg_2(X_2), \ltimes)$ over X with the same interoperability relation $\ltimes$ as $\boldsymbol{Sg}$, along with order-sorted signatures $Sg_i(X_i)$ defined with

   - the same poset $(S_i, \leqslant_i)$ of sorts of Sg_i;
   - if the function symbol $f\colon w \to s$ is in Sg_i, then $f\colon w \to s$ is also in $Sg_i(X_i)$;
   - if the constant $k\colon s$ is in Sg_i, then $k\colon s$ is also in $Sg_i(X_i)$; and
   - if $x \in (X_i)_s$, then $x\colon s$ is in $Sg_i(X_i)$.

   Informally, $Sg_i(X_i)$ consists of all Sg_i operators and, for all $s \in S_i$, all variables of sort s_i.

2. The *(canonical) free multi-language $\boldsymbol{Sg}$-algebra* $\mathscr{T}_{\boldsymbol{Sg}}(X)$ over X is defined by making direct use of the term $\boldsymbol{Sg}(X)$-algebra $\mathscr{T}_{\boldsymbol{Sg}(X)}$. The order-sorted Sg_i-algebras $\mathscr{T}_{\boldsymbol{Sg}}(X)_i$, for $i = 1, 2$, are defined by

   (Canonical) Free multi-language algebra

   - $[\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} = [\![s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_i}$ for each $s \in S_i$;
   - $[\![k]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} = [\![k]\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_i}$ for each constant $k\colon s$ in Sg_i; and
   - $[\![f\colon w \to s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} = [\![f\colon w \to s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_i}$ for each $f\colon w \to s$ in Sg_i.

   And

   $$[\![s \ltimes s']\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)}\colon [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} \to [\![s']\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_j}$$

   where $i, j \in \{1, 2\}$ and $i \neq j$, is defined by

   $$[\![s \ltimes s']\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)}\colon [\![s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_i} \to [\![s']\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_j} \qquad \diamondsuit$$

Given X and $\boldsymbol{Sg}$, a *multi-language term with variable* t is any element of the set

Multi-language term with variables

$$(\mathsf{T}_{\boldsymbol{Sg}}(X))_s = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)} = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)^*}$$

for some sort $s \in S_i$, with $i = 1, 2$. $\mathscr{T}_{\boldsymbol{Sg}(X)^*}$ is the "standard" term order-sorted algebra over the associated signature $\boldsymbol{Sg}(X)^*$, and we will say that t has sort s.

Proposition 5.1. *We prove the previous equation, that is*

$$(\mathsf{T}_{\boldsymbol{Sg}}(X))_s = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)} = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)^*}$$

Proof.

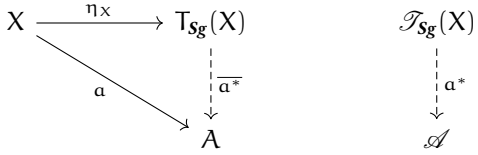
$$\begin{aligned}(\mathsf{T}_{\boldsymbol{Sg}}(X))_s &= [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)} && \text{by Not. 5.2}\\ &= [\![s]\!]_{(\mathscr{T}_{\boldsymbol{Sg}}(X))_i} && s \in S_i, \text{ for some } i = 1,2\\ &= [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)^*} && \text{by Def. 4.5}\end{aligned}$$

□

We can now show the freeness of $\mathscr{T}_{\boldsymbol{Sg}}(X)$ in $Alg(\boldsymbol{Sg})$.

Theorem 5.1. *Let* $\boldsymbol{Sg}$ *be a regular multi-language signature. Then, the* $\boldsymbol{Sg}$*-algebra* $\mathscr{T}_{\boldsymbol{Sg}}(X)$ *is free over* X *in* $Alg(\boldsymbol{Sg})$.

Proof. Take $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$. Let $\mathscr{A}$ be an $\boldsymbol{Sg}$-algebra and $a\colon X \to A$ an S-sorted function. We need to define the data in the following diagram in such a way that the $\boldsymbol{Sg}$-homomorphism $a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{A}$ is unique and makes the diagram commute.



(5.2)

The S-sorted function $\eta_X\colon X \to \mathsf{T}_{\boldsymbol{Sg}}(X)$ has components $(\eta_X)_s\colon X_s \to (\mathsf{T}_{\boldsymbol{Sg}}(X))_s$ given by set inclusions $X_s \subseteq (\mathsf{T}_{\boldsymbol{Sg}}(X))_s$. Informally, the definition of $a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{A}$ is straightforward: it is specified by structural recursion, and is defined in the unique way such that the definition yields a homomorphism. However, we provide a few more details. We have to give two Sg_i-homomorphisms $a_i^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X)_i \to \mathscr{A}_i$, or, by Lemma 5.1, equivalently giving an S-sorted function $\overline{a^*}\colon \mathsf{T}_{\boldsymbol{Sg}}(X) \to A$ with components $\overline{a^*}_s\colon (\mathsf{T}_{\boldsymbol{Sg}}(X))_s \to A_s$), which act as homomorphisms and commute with the boundary functions. Suppose that $s \in S_i$, for some $i = 1,2$. Whichever way you prefer to work, the definitions require that we define S_i (or, S) sorted functions with components

$$(a_i^*)_s\colon [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} \to [\![s]\!]_{\mathscr{A}_i} \qquad (\text{or, } \overline{a^*}_s\colon (\mathsf{T}_{\boldsymbol{Sg}}(X))_s \to A_s)$$

The source set is a set of inductively defined terms; hence a^* exists and can be defined by a structural recursion. Once again, the requirement to be an $\boldsymbol{Sg}$-homomorphism forces $a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{A}$ to be unique, bar the action on variable terms. However, the commutivity also forces us to define

$$(a_i^*)_s(x \in X_s) = a(x)$$

We provide indicative details for an element of $[\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} = [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)^*}$ (by Proposition 5.1) of just one structural form, built from a function symbol and sub-terms. Suppose that $f\colon w \to s$ in Sg_i, so that $f_i\colon w \to s$ in $\boldsymbol{Sg}(X)$. We show by example some of the details of the recursive definition for the Sg_i-homomorphism a_i^*. To be a homomorphism, picking any $s \in S_i$, we require

$$\begin{aligned}(a_i^*)_s \circ [\![f\colon w \to s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} &= [\![f\colon w \to s]\!]_{\mathscr{A}_i} \circ (a_i^*)_w\\ &\colon [\![w]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(X)_i} \to [\![s]\!]_{\mathscr{A}_i}\end{aligned}$$

Note that $[\![f\colon w \to s]\!]_{\mathcal{T}_{Sg}(X)_i} = [\![f_i\colon w \to s]\!]_{\mathcal{T}_{Sg}(X)^*}$. Suppose that $w = s_1 \dots s_n$ where each $s_k \in S_i$. If we pick any

$$(t_1, \dots, t_n) \in [\![w]\!]_{\mathcal{T}_{Sg}(X)_i} = [\![s_1]\!]_{\mathcal{T}_{Sg}(X)_i} \times \cdots \times [\![s_n]\!]_{\mathcal{T}_{Sg}(X)_i}$$

then we *recursively define*

$$(a_i^*)_s(f_i(t_1, \dots, t_n)) = \Phi((a_i^*)_{s_1}(t_1), \dots, (a_i^*)_{s_n}(t_n))$$

The only recursive definition which yields a homomorphism is the one where

$$\Phi = [\![f\colon w \to s]\!]_{\mathcal{A}_i}$$

(noting $[\![f_i\colon w \to s]\!]_{\mathcal{T}_{Sg}(X)^*} = f_i$). To ensure commutativity with boundary functions, and for constants, we also *recursively define*

$$(a_j^*)_{s'}(\hookrightarrow_{s,s'}(t)) = [\![s \ltimes s']\!]_{\mathcal{A}}((a_i^*)_s(t))$$

and

$$(a_i^*)_s(k_i) = [\![k]\!]_{\mathcal{A}_i}$$

respectively. The four recursive clauses complete the definition of a^*. □

Example 5.1. Let $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}$ be the multi-language term algebra over the signature $\mathbf{Sg}(Exp + Str)$ of Example 4.2, and let X be the S-sorted set of variables defined by $X_{nat} = \{n\}$ and $X_{chr} = \{c\}$. Then, the free algebra $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}(X)$ carries the terms in $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}$ enriched with variables in X. For instance, the multi-language term

$$t = +(\hookrightarrow_{chr,nat}(c), n)$$

is in $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}(X)$ but not in $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}$.

Since the freeness of $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}(X)$, given an assignment function $f\colon X \to A$, where A is the carrier set of the algebra $\mathcal{M}$ of Example 4.2, there is a unique homomorphism $f^*\colon \mathcal{T}_{\mathbf{Sg}(Exp+Str)}(X) \to \mathcal{M}$ able to compute the semantics of each term in $\mathcal{T}_{\mathbf{Sg}(Exp+Str)}(X)$. For instance, let $f_{nat}(n) = 0$ and $f_{chr}(c) = a$. Then,

$$\begin{aligned} f^*_{ls(t)}(t) &= f^*_{nat}(t) \\ &= [\![+\colon nat, nat \to nat]\!]_{\mathcal{E}}(f^*_{nat}(\hookrightarrow_{chr,nat}(c)), f^*_{nat}(n)) \\ &= [\![+\colon nat, nat \to nat]\!]_{\mathcal{E}}([\![chr \ltimes nat]\!]_{\mathcal{M}}(f_{chr}(c)), f_{nat}(n)) \\ &= [\![+\colon nat, nat \to nat]\!]_{\mathcal{E}}(1, 0) = 1 \end{aligned}$$

Steps of this calculation are based on the (constructive) proof of Theorem 5.1. ♢

5.2 Subalgebras, Kernels, and Congruences in a Multi-Language Context

The aim of this section is to introduce and extend well-known constructions of universal algebra to multi-languages. Apart from this being interesting in its own right, quotient algebras are central to our completeness proof. In particular, we define the notions of *congruence* and *quotient algebra*. In order to better understand the section, we recall that the same concepts in the order-sorted context have been presented in Section 3.6.

Proposition 5.2. *Let* $h\colon \mathscr{A} \to \mathscr{B}$ *be an* ***Sg****-homomorphism. Then,* $h\colon \mathscr{A} \to \mathscr{B}$ *is an isomorphism in* $Alg(\boldsymbol{Sg})$ *if and only if* $h\colon A \to B$ *is a bijection.*

Locally filtered, Coherence

Definition 5.3 (Coherence)**.** A multi-language signature ***Sg*** is *locally filtered* (respectively, *coherent*) if its associated signature $\boldsymbol{Sg}^*$ is locally filtered (respectively, coherent). ♢

We shall sometimes need these properties for results to hold: and we will require coherence for a well-defined notion of equation later in the chapter.

The notion of *subalgebra* is a generalization of the concept of subset in set theory.

Multi-language subalgebra

Definition 5.4 (Multi-Language Subalgebra)**.** Let $\mathscr{A}$ and $\mathscr{B}$ be two multi-language ***Sg***-algebras. Then, $\mathscr{B}$ is a *multi-language* ***Sg****-subalgebra* of $\mathscr{A}$ if

(i) $\mathscr{B}_i$ is an order-sorted Sg_i-subalgebra of $\mathscr{A}_i$, for $i = 1, 2$; and

(ii) $s \ltimes s'$ implies $[\![s \ltimes s']\!]_{\mathscr{A}}(b) = [\![s \ltimes s']\!]_{\mathscr{B}}(b)$ for each $b \in [\![s]\!]_{\mathscr{B}_i}$ provided that $s \in S_i$ for some $i = 1, 2$. ♢

The next definition provides a convenient tool to build subalgebras.

Closed subset

Definition 5.5 (Closed Subset)**.** Let $\mathscr{A}$ be a multi-language ***Sg***-algebra. An S-sorted subset $B \subseteq A$ is *closed* in $\mathscr{A}$ if

(i) $B_i = (\, B_s \mid s \in S_i \,)$ is a closed subset of A_i (see Definition 3.8);

(ii) $s \ltimes s'$ implies $[\![s \ltimes s']\!]_{\mathscr{A}}(b) \in (B_j)_{s'}$ for each $b \in (B_i)_s$ with $i, j = 1, 2$ and $i \neq j$. ♢

A subalgebra $\mathscr{B}$ of $\mathscr{A}$ is usually defined starting from a closed subset $B \subseteq A$. The interpretation domains of $\mathscr{B}$ are exactly the components of B, while the interpretation and the boundary functions of $\mathscr{B}$ are those of $\mathscr{A}$ restricted to B.

A very general way that subalgebras arise is from homomorphisms.

Lemma 5.2. *Let* $h\colon \mathscr{A} \to \mathscr{B}$ *be a multi-language* ***Sg****-homomorphism. Then, the image* $h(A)$ *of* h *gives rise to a multi-language* ***Sg****-subalgebra of* $\mathscr{B}$*, and denoted by* $h(\mathscr{A})$*.*

Proof. It is sufficient to show that $h(A)$ is a closed subset in $\mathscr{B}$, and then applying the construction described above this lemma. We first prove that

$$h(A)_i = (\, h_s(A_s) \mid s \in S_i \,)$$

is a closed subset in $\mathscr{B}_i$. Since $h(A)_i = h_i(A_i)$ and $h_i\colon \mathscr{A}_i \to \mathscr{B}_i$ is an Sg_i-homomorphism, then $h(A)_i$ is closed in $\mathscr{B}_i$ by Proposition 3.8. We now need to check condition (5.5:ii), that is, given $s \ltimes s'$ in ***Sg***, the conversion $[\![s \ltimes s']\!]_{\mathscr{B}}(x)$ of x have to reside in $(h(A)_j)_{s'}$, for each $x \in (h(A)_i)_s$ with $i, j = 1, 2$ and $i \neq j$. From $x \in (h(A)_i)_s$, it follows that there is y in $[\![s]\!]_{\mathscr{A}_i}$ such that

$$[\![s \ltimes s']\!]_{\mathscr{B}}(x) = [\![s \ltimes s']\!]_{\mathscr{B}}(h_s(y)) = h_{s'}\big([\![s \ltimes s']\!]_{\mathscr{A}}(y)\big) \in h(A)_{s'}$$

and hence the thesis. □

This lemma asserts that the category of multi-language algebras is closed under the taking of *homomorphic images*. Moreover, it is easy to prove that the corestriction $h' \colon A \to h(A)$ of h to its image $h(A)$ is indeed a homomorphism from $\mathscr{A}$ to the subalgebra of $\mathscr{B}$ generated by h, that is $h' \colon \mathscr{A} \to h(\mathscr{A})$.

A *congruence (relation)* is an equivalence relation on the carrier sets of a given algebra which is compatible with its structure. We extend this notion to the multi-language world.

Definition 5.6 (Multi-Language Congruence)**.** Let $\mathscr{A}$ be a multi-language ***Sg***-algebra. A *multi-language **Sg**-congruence*

Multi-language congruence

$$\equiv = (\, \equiv_s \subseteq [\![s]\!]_{\mathscr{A}_i} \times [\![s]\!]_{\mathscr{A}_i} \mid s \in S_i \text{ for } i = 1, 2\,)$$

is an S-sorted family of equivalence relations such that

(i) the S_i-restriction $\equiv|_{S_i} = (\, \equiv_s \mid s \in S_i\,)$ is an Sg_i*-congruence* on $\mathscr{A}_i$ for $i = 1, 2$; and

(ii) $s \ltimes s'$ and $a \equiv_s a'$ implies $[\![s \ltimes s']\!]_{\mathscr{A}}(a) \equiv_{s'} [\![s \ltimes s']\!]_{\mathscr{A}}(a')$ for each $a, a' \in [\![s]\!]_{\mathscr{A}}$. ♢

One general way congruences arise is from homomorphisms:

Definition 5.7 (Kernel)**.** Let $f \colon \mathscr{A} \to \mathscr{B}$ be an ***Sg***-homomorphism. The *kernel* of f is the S-sorted family of equivalence relations $\equiv_f$ defined by $a (\equiv_f)_s a'$ if and only if $f_s(a) = f_s(a')$. ♢

Kernel

Proposition 5.3. *Let $f \colon \mathscr{A} \to \mathscr{B}$ be a multi-language **Sg**-homomorphism. Then, the kernel of f is a multi-language **Sg**-congruence.*

Proof. Given a homomorphism $f \colon \mathscr{A} \to \mathscr{B}$ one can check that

$$(\equiv_f)|_{S_i} = \equiv_{f_i}$$

Moreover, $f_i \colon \mathscr{A}_i \to \mathscr{B}_i$ is an order-sorted Sg_i-homomorphism for $i = 1, 2$. Since the kernel of an order-sorted Sg_i-homomorphism is an order-sorted Sg_i-congruence (Proposition 3.9), the condition (5.6:i) is satisfied. As regards condition (5.6:ii), suppose that $s \ltimes s'$ and $a (\equiv_f)_s a'$. Then, $f_s(a) = f_s(a')$ and therefore

$$[\![s \ltimes s']\!]_{\mathscr{B}}(f_s(a)) = [\![s \ltimes s']\!]_{\mathscr{B}}(f_s(a'))$$

Since f is an ***Sg***-homomorphism, it commutes with boundary functions (condition (4.3:ii)), thus

$$f_{s'}([\![s \ltimes s']\!]_{\mathscr{A}}(a)) = f_{s'}([\![s \ltimes s']\!]_{\mathscr{A}}(a'))$$

and therefore

$$[\![s \ltimes s']\!]_{\mathscr{A}}(a)(\equiv_f)_{s'}[\![s \ltimes s']\!]_{\mathscr{A}}(a')$$ □

Let us denote by $\mathrm{Cong}(\mathscr{A})$ the set of all congruences on an algebra $\mathscr{A}$. Then, we have the following proposition:

Proposition 5.4. *Let $\mathscr{A}$ be a multi-language **Sg**-algebra. Then, the set of multi-language congruences on $\mathscr{A}$ is a complete lattice under the inclusion ordering, and it is denoted by* $\mathrm{Cong}(\mathscr{A})$.

Proof. Let $\mathscr{X}$ be a set of congruences on $\mathscr{A}$. It is well-known that the intersection of arbitrary equivalence relations is still an equivalence relation. Moreover, let

$$\widehat{\equiv} = \bigcap \mathscr{X} = \bigcap \{ \equiv \subseteq A \times A \mid \equiv \in \mathscr{X} \} = \left\{ \bigcap \equiv_s \subseteq A_s \times A_s \;\middle|\; \equiv \in \mathscr{X} \wedge s \in S \right\}$$

Then,

$$\widehat{\equiv}|_{S_i} = \left\{ \bigcap \equiv_s \subseteq A_s \times A_s \;\middle|\; \equiv \in \mathscr{X} \wedge s \in S_i \right\} = \bigcap \{ \equiv|_{S_i} \subseteq A_i \times A_i \mid \equiv \in \mathscr{X} \}$$

Since every $\equiv \in \mathscr{X}$ is a multi-language congruence, then every restriction $\equiv|_{S_i}$ is an order-sorted Sg_i-congruence by condition (5.6:i). By Proposition 3.10, $\widehat{\equiv}|_{S_i}$ is an order-sorted Sg_i-congruence, hence $\widehat{\equiv}$ satisfies condition (5.6:i). Finally, proving that $\widehat{\equiv}$ satisfies condition (5.6:ii) is trivial. Hence, the infimum of $\mathscr{X}$ is exactly $\widehat{\equiv} = \bigcap \mathscr{X}$.

Now, let

$$\check{\equiv} = \bigcap \left\{ X \in \mathrm{Cong}(\mathscr{A}) \;\middle|\; \bigcup \mathscr{X} \subseteq X \right\}$$

Note that $\check{\equiv}$ is an upper bound of $\mathscr{X}$, and, by the first part of this proof, it is also a congruence on $\mathscr{A}$. Suppose that $\tilde{\equiv}$ is another upper bound of $\mathscr{X}$, namely $\tilde{\equiv} \in \mathrm{Cong}(\mathscr{A})$ and $\equiv \subseteq \tilde{\equiv}$ for all $\equiv \in \mathscr{X}$. Therefore, $\bigcup \mathscr{X} \subseteq \tilde{\equiv}$ and, by the definition of $\check{\equiv}$, it follows that $\check{\equiv} \subseteq \tilde{\equiv}$. Hence, the supremum of $\mathscr{X}$ is $\check{\equiv}$. □

5.3 Quotienting of Multi-Language Algebras

The next definition lifts the notion of *quotient algebra* to multi-languages. Intuitively, a quotient algebra is the outcome of the partitioning of an algebra through a congruence relation.

Definition 5.8 (Quotient Multi-Language Algebra)**.** Let $\mathscr{A}$ be a multi-language algebra over a locally filtered multi-language signature $\boldsymbol{Sg}$. Given an $\boldsymbol{Sg}$-congruence $\equiv$, the *quotient* of $\mathscr{A}$ induced by $\equiv$ is the $\boldsymbol{Sg}$-algebra $\mathscr{A}/\equiv$ defined as follows:

Quotient multi-language algebra

(i) $(\mathscr{A}/\equiv)_i$ is the order-sorted quotient Sg_i-algebra $\mathscr{A}_i/(\equiv|_{S_i})$; and

(ii) $s \ltimes s'$ implies $[\![s \ltimes s']\!]_{\mathscr{A}/\equiv}([a]) = \big[[\![s \ltimes s']\!]_{\mathscr{A}}(a) \big]$. ♢

For each quotient algebra $\mathscr{A}/\equiv$ of the multi-language algebra $\mathscr{A}$, the homomorphic *quotient map* $q \colon A \to A/\equiv$ is defined by $q(a) = [a]$. Moreover, the kernel of q is exactly $\equiv$. Following Definition 4.3 and Lemma 5.1 we shall also write q for the order-sorted homomorphism $q \colon \mathscr{A} \to \mathscr{A}/\equiv$.

Quotient map

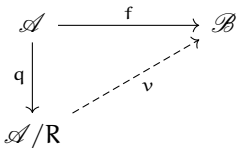
In the following, if R is an S-sorted family of binary relations on A, we denote by $\mathscr{A}/R$ the quotient multi-language algebra induced by the smallest congruence relation $\equiv_R$ that contains R. Such congruence always exists due to Proposition 5.4 and it is exactly

$$\equiv_R = \bigcap \{ \equiv \in \mathrm{Cong}(\mathscr{A}) \mid \equiv \subseteq R \}$$

Proposition 5.5. *Let $\mathscr{A}$ be a multi-language algebra over a locally filtered multi-language signature $\boldsymbol{Sg}$. If R is an S-sorted family of binary relations on A, then the quotient map $q \colon \mathscr{A} \to \mathscr{A}/R$ satisfies the following:*

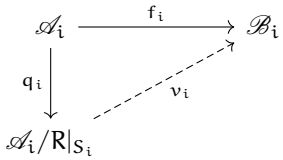
1. $R \subseteq \equiv_q$; *and*

2. *for each* $f\colon \mathscr{A} \to \mathscr{B}$ *such that* $R \subseteq \equiv_f$ *there is a unique multi-language* ***Sg****-homomorphism* $v\colon \mathscr{A}/R \to \mathscr{B}$ *such that the following diagram commute:*



(5.3)

Proof. Point (1) follows from $\equiv_q = \bigcap\{\equiv \in \mathrm{Cong}(\mathscr{A}) \mid \equiv \subseteq R\}$. As regards point (2), let $f\colon \mathscr{A} \to \mathscr{B}$ be a multi-language homomorphism such that $R \subseteq \equiv_f$. Then, $f_i\colon \mathscr{A}_i \to \mathscr{B}_i$ is an order-sorted Sg_i-homomorphism for $i = 1, 2$. Moreover, since $(\mathscr{A}/R)_i$ is the order-sorted quotient Sg_i-algebra $\mathscr{A}_i/R|_{S_i}$ (condition (5.8:i)), then $q_i\colon \mathscr{A}_i \to \mathscr{A}_i/R|_{S_i}$ is exactly the order-sorted quotient map. Hence, by Proposition 3.11 there is an order-sorted Sg_i-homomorphism $v_i\colon \mathscr{A}_i/R|_{S_i} \to \mathscr{B}_i$ making the following parametrised diagram commute:



(5.4)

Let $\overline{v}$ be the S-sorted function from the carrier of $\mathscr{A}/R$ to the carrier of $\mathscr{B}$ obtained by joining v_1 and v_2. Since the domains of v_1 and v_2 are disjoint, then $\overline{v}$ is well-defined and $v_i = v|_{S_i}$. We now prove that v is a multi-language homomorphism. We have to check condition (4.3:ii): Let $s \ltimes s'$. Then, $s \in S_j$ and $s' \in S_k$ with $j, k = 1, 2$ and $j \neq k$. Thus,

$$
\begin{aligned}
\big((v_k)_{s'} \circ [\![s \ltimes s']\!]_{\mathscr{A}/R}\big)([a]) &= (v_k)_{s'}\Big(\big[[\![s \ltimes s']\!]_{\mathscr{A}}(a)\big]\Big) && \text{by cond. (5.8:ii)}\\
&= (f_j)_{s'}([\![s \ltimes s']\!]_{\mathscr{A}}(a)) && \text{by Eq. 5.4}\\
&= [\![s \ltimes s']\!]_{\mathscr{B}}((f_k)_s(a)) && \text{by cond. (4.3:ii)}\\
&= [\![s \ltimes s']\!]_{\mathscr{B}}((v_j)_s([a])) && \text{by Eq. 5.4}\\
&= \big([\![s \ltimes s']\!]_{\mathscr{B}} \circ (v_j)_s\big)([a])
\end{aligned}
$$

Finally, suppose that $w\colon \mathscr{A}/R \to \mathscr{B}$ makes Diagram 5.3 commutes. Then, $w|_{S_i} = v_i$ for $i = 1, 2$, and hence the uniqueness. □

By setting $R = \equiv_f$ in this proposition, we get a multi-language version of the *first homomorphism theorem* which gives an isomorphism between the homomorphic image induced by f and the quotient algebra induced by $\equiv_f$.

Theorem 5.2 (First Isomorphism Theorem)**.** *Let* ***Sg*** *be a locally filtered signature. If* $f\colon \mathscr{A} \to \mathscr{B}$ *is an* ***Sg****-homomorphism, then* $\mathscr{A}/\equiv_f \cong f(\mathscr{A})$.

Proof. Let $f'\colon A \to f(A)$ be the corestriction of f and $R = \equiv_{f'}$ in Proposition 5.5 (note that $f(\mathscr{A})$ is a multi-language algebra and f' a multi-language homomorphism by Lemma 5.2). Then, there is a unique $v\colon \mathscr{A}/\equiv_{f'} \to \mathscr{B}$ making Diagram 5.3 commutes, and, by Theorem 5.2, v_i is an order-sorted Sg_i-isomorphism, for $i = 1, 2$. Thus, by Proposition 5.2, v is a multi-language isomorphism and hence the thesis. □

5.4 Equational Logic for Multi-Language Theories

We now give an equational logic for multi-language terms. We define equations between multi-language terms and show that our system is sound and complete for ***Sg***-algebras. We also show the existence of an initial algebra. Through conversion operators $\hookrightarrow$, multi-language equations can usefully axiomatize boundary functions.

Definition 5.9 (Multi-Language Equations)**.** Let ***Sg*** be a coherent multi-language signature, X be an S-sorted set of variables, and $\mathscr{T}_{Sg}(X)$ the free ***Sg***-algebra over X. An *(unconditional) multi-language **Sg**-equation* over X is a judgement of the form

Multi-language equation

$$(\forall X)\ \ t = t'$$

such that $t \in [\![s]\!]_{\mathscr{T}_{Sg}(X)}$ and $t' \in [\![s']\!]_{\mathscr{T}_{Sg}(X)}$ for some $s, s' \in S_i$ and some $i = 1, 2$, where s and s' are connected via $\leqslant_i$ (equivalently, least sorts $\mathrm{ls}(t)$ and $\mathrm{ls}(t')$ falls in the same connected component of S_i). ♢

Note that $(\forall X)\ \ t = t'$ is an ***Sg***-equation. The connectedness of $\mathrm{ls}(t)$ and $\mathrm{ls}(t')$, and coherence which ensures both terms have a super-sort, is necessary to ensure that forthcoming definitions of equation satisfaction are well-defined.

Definition 5.10 (Conditional Multi-Language Equations)**.** Let ***Sg*** be a coherent multi-language signature, X be an S-sorted set of variables, and $\mathscr{T}_{Sg}(X)$ the free ***Sg***-algebra over X. A *multi-language conditional **Sg**-equation* over X is a judgement of the form

Multi-language conditional equation

$$(\forall X)\ \ t = t' \Leftarrow C$$

such that $(\forall X)\ \ t = t'$ is an ***Sg***-equation and C is a *finite set* of ***Sg***-equations over X. Whenever we want to be explicit about the equations in C, we rewrite the previous equation as

$$(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t'_i$$

where $(\forall X)\ \ t_i = t'_i$ belongs to C for each $i = 1, \ldots, n$. ♢

As is the case for order-sorted equations, any unconditional multi-language equation $(\forall X)\ \ t = t'$ can be regarded as $(\forall X)\ \ t = t' \Leftarrow \varnothing$. Moreover, we write $Ax(X)$ for any set of conditional and/or unconditional equations over a set of variables X and we call $Ax(X)$ a set of ***Sg***-axioms over X.

The next step is to show how new equations can be inductively generated (derived) from a set of axioms. The rules require the concept of substitution, namely a syntactic transformation that replaces variables with terms. Let X and Y be two sets of variables. A *variable substitution* is an S-sorted function $\theta\colon X \to \mathsf{T}_{\mathbf{Sg}}(Y)$: for each variable $x \in X_s$ we supply a term $\theta_s(x)$, of sort s, involving variables from Y. Then, by the freeness of $\mathscr{T}_{\mathbf{Sg}}(X)$, there is a unique *substitution homomorphism* $\theta^*\colon \mathscr{T}_{\mathbf{Sg}}(X) \to \mathscr{T}_{\mathbf{Sg}}(Y)$ induced by θ such that $\theta^* \circ \eta_X = \theta$: The idea is that for any $t \in [\![s]\!]_{\mathscr{T}_{\mathbf{Sg}}(X)}$, θ^* recurses over t and then finally substitutes $\theta_s(x)$ for each x occuring in t.

Variable substitution

Substitution homomorphism

Notation 5.3. We shall adopt some further notation, used in the rules. If $a\colon X \to A$, where X is a set of variables, then any $x \in X_s$ has the unique sort s and we write $a(x)$ for $a_s(x)$. Further, for any S-sorted homomorphism $h\colon \mathscr{T}_{\mathbf{Sg}}(X) \to \mathscr{A}$, if $t \in [\![s]\!]_{\mathscr{T}_{\mathbf{Sg}}(X)}$ then of course $h_s(t) \in [\![s]\!]_{\mathscr{A}}$. But, since there is a least sort $\mathrm{ls}(t)$, also

$$t \in [\![\mathrm{ls}(t)]\!]_{\mathscr{T}_{\mathbf{Sg}}(X)} \subseteq [\![s]\!]_{\mathscr{T}_{\mathbf{Sg}}(X)}$$

$$\text{(mlAx)}\ \frac{(\forall X)\ \ t = t' \in Ax(X)}{(\forall X)\ \ t = t'} \qquad \text{(mlRef)}\ \frac{-}{(\forall X)\ \ t = t} \qquad \text{(mlSym)}\ \frac{(\forall X)\ \ t = t'}{(\forall X)\ \ t' = t}$$

$$\text{(mlTrans)}\ \frac{(\forall X)\ \ t = t' \qquad (\forall X)\ \ t' = t''}{(\forall X)\ \ t = t''}$$

$$\text{(mlCong)}\ \frac{\begin{array}{c}\theta, \phi \colon X \to T_{\mathbf{Sg}}(Y) \\ (\forall s \in S, \forall x \in X_s)\ \ (\forall Y)\ \ \theta_s(x) = \phi_s(x)\end{array}}{(\forall Y)\ \ \theta^*(t) = \phi^*(t)}$$

$$\text{(mlSub)}\ \frac{\begin{array}{c}\theta \colon X \to T_{\mathbf{Sg}}(Y) \\ (\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i' \in Ax(X) \\ (\forall 1 \leqslant i \leqslant n)\ \ (\forall Y)\ \ \theta^*(t_i) = \theta^*(t_i')\end{array}}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')}$$

Figure 5.1: Inference rules for inductively defining equational logic.

and since h is a homomorphism we have

$$h_{ls(t)}(t) = h_s(t) \in [\![ls(t)]\!]_{\mathscr{A}} \subseteq [\![s]\!]_{\mathscr{A}}$$

Thus, from now on we write $h(t)$ for $h_{ls(t)}(t)$ (that is, $h(t) = h_s(t) \in [\![s]\!]_{\mathscr{A}}$) to reduce subscript notation. ♢

Definition 5.11 (Equations Derived from Axioms)**.** Given a set $Ax(X)$ of axioms over a set X of variables, the inductive rules of equational logic allow the derivation of new *unconditional* equations. Our rules in Figure 5.1 are a modification of those in [GM92], and they make use of the following notation: t, t', and t'' are multi-language terms in $\mathscr{T}_{\mathbf{Sg}}(X)$, with $\mathbf{Sg}$ a multi-language signature, and $\theta, \phi \colon X \to T_{\mathbf{Sg}}(Y)$ are two variable substitutions from X to $T_{\mathbf{Sg}}(Y)$ over a set Y of variables. Thus one also has $\mathbf{Sg}$-homomorphisms $\theta^*, \phi^* \colon \mathscr{T}_{\mathbf{Sg}}(X) \to \mathscr{T}_{\mathbf{Sg}}(Y)$ by freeness. ♢

In the following, if $Ax(X)$ is a set of $\mathbf{Sg}$-axioms, we call $\mathbf{Th} = (\mathbf{Sg}, Ax(X))$ a *multi-language theory*. If $(\forall X)\ \ t = t'$ is generated from $Ax(X)$, then we say that $(\forall X)\ \ t = t'$ is a *theorem* derivable (or, generated) from $\mathbf{Th}$.

Multi-language theory, Theorem

The meaning of the rules of multi-language equational logic is analogous to the order-sorted case (Section 3.8); despite the form of the rules is the same, terms appearing in equations are (potentially) built from a combination of both underlying signatures Sg_1 and Sg_2, that is they are multi-language terms. Therefore, the main theorems of this chapter (soundness and completeness) have to be proven with respect the multi-language definitions of terms, equations, and satisfaction.

The rule (mlCong) intuitively says that replacing equals for equals yields new equalities, that is, term formation is a congruence. Thus we may build new equations using the "usual/informal" rules of equational reasoning. One easily proves admissible rule

$$\text{(mlGSub)}\ \frac{(\forall X)\ \ t = t'}{(\forall Y)\ \ \theta^*(t) = \theta^*(t')}\ \ \theta \colon X \to T_{Sg}(Y)$$

namely that if $(\forall X)\ \ t = t'$ has been derived from the theory, then $(\forall Y)\ \ \theta^*(t) = \theta^*(t')$ is also derivable; but note that the proof requires the following fact: Let

X, Y, and Z be variable sets, the composition of two substitution homomorphisms $\theta_1^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{T}_{\boldsymbol{Sg}}(Y)$ and $\theta_2^*\colon \mathscr{T}_{\boldsymbol{Sg}}(Y) \to \mathscr{T}_{\boldsymbol{Sg}}(Z)$ is still a substitution homomorphism, since it is induced by $\theta_2^* \circ \theta_1$. To see that

$$\theta_2^* \circ \theta_1^* = (\theta_2^* \circ \theta_1)^* \tag{5.5}$$

note that

$$(\theta_2^* \circ \theta_1)^* \circ \eta_X = \theta_2^* \circ \theta_1 = (\theta_2^* \circ \theta_1^*) \circ \eta_X$$

Notation 5.4. In the following, whenever we introduce a multi-language theory $\boldsymbol{Th}$ we leave implicit that $\boldsymbol{Th} = (\boldsymbol{Sg}, Ax(X))$. ♢

5.5 Soundness, Completeness, and Freeness Results

We now prove the main results of the chapter, namely soundness and completeness of the deduction and the freeness of the algebra obtained by quotienting the multi-language free algebra by the usual congruence relation that links provably equal multi-language terms.

Definition 5.12 (Satisfaction)**.** Let $\mathscr{A}$ be a multi-language algebra over a regular multi-language signature $\boldsymbol{Sg}$, and X a set of variables. The algebra $\mathscr{A}$ *satisfies* an unconditional equation

$$(\forall X)\ \ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t_i'$$

if for each assignment function $a\colon X \to A$ such that $a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{A}$ equalizes each premise $(\forall X)\ \ t_i = t_i'$, that is,

$$a^*(t_i) = a^*(t_i')$$

then a also equalizes t and t', namely

$$a^*(t) = a^*(t')$$

Also, $\mathscr{A}$ satisfies an unconditional $\boldsymbol{Sg}$-equation $(\forall X)\ \ t = t'$ just in case each assignment function $a\colon X \to A$ equalizes t and t'. ♢

Multi-language model

Let $Ax(X)$ be a set of conditional and/or unconditional equations. If an $\boldsymbol{Sg}$-algebra $\mathscr{A}$ satisfies each equation in $Ax(X)$ we say that $\mathscr{A}$ is a *model* of the multi-language theory $\boldsymbol{Th} = (\boldsymbol{Sg}, Ax(X))$. Think of $Ax(X)$ as a set of *axioms* satisfied (modelled) by the $\boldsymbol{Sg}$-algebra $\mathscr{A}$.

One may generalize the above fact about substitution homomorphisms (see Equation 5.5) to arbitrary assignment functions and $\boldsymbol{Sg}$-homomorphisms. We do so in the next lemma, which is used in the proof of soundeness and completeness.

Lemma 5.3. *Let $\mathscr{A}$ and $\mathscr{B}$ be two $\boldsymbol{Sg}$-algebras over a regular multi-language signature $\boldsymbol{Sg}$. If $a\colon X \to A$ is an assignment function and $h\colon \mathscr{A} \to \mathscr{B}$ is an $\boldsymbol{Sg}$-homomorphism, then*

$$(h \circ a)^* = h \circ a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(X) \to \mathscr{B}$$

Proof. The proof is easy. We have

$$(h \circ a^*) \circ \eta = h \circ (a^* \circ \eta) = h \circ a$$

and we are done by uniqueness. □

Theorem 5.3 (Soundness and Completeness)**.** *Let* ***Th*** *be a theory over a coherent multi-language signature* ***Sg*** *and* $(\forall Y)\ t = t'$ *any equation. Then,* $(\forall Y)\ t = t'$ *is derivable from the theory just in case, for every* ***Sg****-model* $\mathscr{A}$ *of the theory* ***Th****, we have that* $\mathscr{A}$ *satisfies the equation* $(\forall Y)\ t = t'$.

Proof. (Soundness) The proof of soundness is by rule induction, using the rules in Figure 5.1. We give only the cases below:

- (mlCong) Suppose an equation has been derived by Rule (mlCong). By induction, we know that each equation

  $$(\forall Y)\ \theta_s(x) = \phi_s(x)$$

  for every $s \in S$ and $x \in X_s$ is satisfied by the algebra $\mathscr{A}$. Pick any assignment function $a\colon Y \to A$ and any term t in $\mathscr{T}_{\mathbf{Sg}}(X)$. We need to show that $(\forall Y)\ \theta^*(t) = \phi^*(t)$ is satisfied by $\mathscr{A}$, that is, $a^*(\theta^*(t)) = a^*(\phi^*(t))$. For any $x \in X_s$ we have that $a^*(\theta_s(x)) = a^*(\phi_s(x))$ which implies that

  $$a^* \circ \theta = a^* \circ \phi\colon X \to A$$

  By the freeness of $\mathscr{T}_{\mathbf{Sg}}(X)$ we have

  $$(a^* \circ \theta)^* = (a^* \circ \phi)^*\colon \mathscr{T}_{\mathbf{Sg}}(X) \to \mathscr{A}$$

  and hence $a^* \circ \theta^* = a^* \circ \phi^*$ by Lemma 5.3.

- (mlSub) Suppose an equation has been derived by Rule (mlSub). We have to prove that $a^*(\theta^*(t)) = a^*(\theta^*(t'))$ for any assignment function $a\colon Y \to A$. Now $\mathscr{A}$ satisfies

  $$(\forall X)\ t = t' \Leftarrow \bigwedge_{i=1}^{n} t_i = t'_i$$

  since all axioms are satisfied by Rule (mlAx), which means that for each $i = 1, \ldots, n$

  $$a^*(t_i) = a^*(t'_i) \implies a^*(t) = a^*(t')$$

  Using the induction hypothesis, we have

  $$(a^* \circ \theta)^*(t_i) = (a^* \circ \theta)^*(t'_i)$$

  for each $i = 1, \ldots, n$, where $a^* \circ \theta\colon X \to A$, and so we deduce that $(a^* \circ \theta)^*(t) = (a^* \circ \theta)^*(t')$. But $(a^* \circ \theta)^* = a^* \circ \theta^*$ so we have $a^*(\theta^*(t)) = a^*(\theta^*(t'))$ as required. □

Soundness has been straightforward to prove. In order to prove completeness, we build a quotient algebra (see [Bir35]) $\mathscr{T}_{\mathbf{Th}}(Y)$ on $\mathscr{T}_{\mathbf{Sg}}(Y)$ such that given terms t and t' in the same equivalence class, the equation $(\forall Y)\ t = t'$ can be derived from ***Th***. The S-sorted binary relation $\sim$ is defined on the carrier sets of $\mathscr{T}_{\mathbf{Sg}}(Y)$ as

$$t \sim_s t' \text{ if and only if } (\forall Y)\ t = t' \text{ is derivable from } \mathbf{Th}$$

where $t, t' \in [\![s]\!]_{\mathscr{T}_{\mathbf{Sg}}(Y)}$. Trivially $\sim$ is an equivalence relation and a congruence on $\mathscr{T}_{\mathbf{Sg}}(Y)$. Hence, we can build the quotient algebra $\mathscr{T}_{\mathbf{Sg}}(Y)/\sim$ denoted by $\mathscr{T}_{\mathbf{Th}}(Y)$. The elements of $\mathscr{T}_{\mathbf{Th}}(Y)$ are exactly equivalence classes of terms, provably equal from ***Th***. Indeed, it is easy to prove that $[t] = [t']$ if and only if $(\forall Y)\ t = t'$ is derivable from the theory. Moreover we have the following proposition

Proposition 5.6. *If* $(\forall Y)\ \ t = t'$ *is satisfied by* $\mathscr{T}_{\boldsymbol{Th}}(Y)$*, then* $(\forall Y)\ \ t = t'$ *is derivable from* ***Th****.*

Proof. Suppose that $\mathscr{T}_{\boldsymbol{Th}}(Y)$ satisfies an equation $(\forall Y)\ \ t = t'$. Then we know for any $a\colon Y \to T_{\boldsymbol{Th}}(Y)$ we have $a^*(t) = a^*(t')$ in $\mathscr{T}_{\boldsymbol{Th}}(Y)$, that is $a^*(t) = a^*(t')$ in $\mathscr{T}_{\boldsymbol{Sg}}(Y)/{\sim}$. Taking a to be the quotient map on variables, that is $a(x) = [x]$, we have

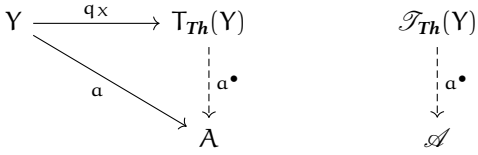
$$a^*(t) = [t] = [t'] = a^*(t')$$

by calculating with Definition 3.13. Hence $t \sim t'$ which implies that $(\forall Y)\ \ t = t'$ is derivable from ***Th***. □

The proof of completeness is then immediate by observing that $\mathscr{T}_{\boldsymbol{Th}}(Y)$ is a model of ***Th*** itself, for then that $\mathscr{T}_{\boldsymbol{Th}}(Y)$ satisfies any conditional axioms $(\forall X)\ \ t = t' \Leftarrow C$ follows by the assumption in the theorem. The class of the multi-language ***Sg***-algebras satisfying a theory ***Th*** is denoted by $Mod(\boldsymbol{Th})$, namely the class of *models* satisfying ***Th***. If we take all the ***Sg***-homomorphisms between them, we have that $Mod(\boldsymbol{Th})$ is a *full subcategory* of $Alg(\boldsymbol{Sg})$. In fact $\mathscr{T}_{\boldsymbol{Th}}(Y)$ is free in $Mod(\boldsymbol{Th})$, where the freeness is defined along the lines of Definition 5.1.

Theorem 5.4. *Let* ***Th*** *be a theory over a coherent signature* ***Sg****. Then, the quotient algebra* $\mathscr{T}_{\boldsymbol{Th}}(Y)$ *is the free algebra over* Y *in* $Mod(\boldsymbol{Th})$*.*

Proof. Let $\mathscr{A}$ satisfy each axiom and let $a\colon Y \to A$ be an assignment function. We prove there is a unique multi-language homomorphism $a^\bullet\colon \mathscr{T}_{\boldsymbol{Th}}(Y) \to \mathscr{A}$ making the following diagram commute (where $q_Y\colon Y \to T_{\boldsymbol{Th}}(Y)$ is defined as $q_Y(y) = [\eta_Y(y)]$, that is, $q_Y = q \circ \eta_Y$ with q the quotient map $q\colon \mathscr{T}_{\boldsymbol{Sg}}(Y) \to \mathscr{T}_{\boldsymbol{Th}}(Y)$):



(5.6)

Let $a^*\colon \mathscr{T}_{\boldsymbol{Sg}}(Y) \to \mathscr{A}$ be the unique homomorphism such that $a^* \circ \eta_Y = a$ (due to the freeness of $\mathscr{T}_{\boldsymbol{Sg}}(Y)$): Suppose $t \sim_s t'$ for some $t, t' \in [\![s]\!]_{\mathscr{T}_{\boldsymbol{Sg}}(Y)}$. Then, by definition of $\sim$, the equation $(\forall Y)\ \ t = t'$ is derivable from ***Th***, and, by Theorem 5.3, the algebra $\mathscr{A}$ satisfies $(\forall Y)\ \ t = t'$ implying that $a^*(t) = a^*(t')$. Hence, $t\ (\equiv_{a^*})_s\ t'$, and therefore $\sim\ \subseteq\ \equiv_{a^*}$. Let us define $a^\bullet[t] = a^*(t)$. This is well-defined since we just proved that if $t \sim t'$ then $a^*(t) = a^*(t')$. Further,

$$a^\bullet \circ q_Y(y) = a^\bullet([\eta_Y(y)]) = a^*(\eta_Y(y)) = a(y)$$ □

Since initiality is a special case of the freeness property and $\mathscr{T}_{\boldsymbol{Sg}}(\varnothing) = \mathscr{T}_{\boldsymbol{Sg}}$, then $\mathscr{T}_{\boldsymbol{Th}}(\varnothing)$ is an initial object in $Mod(\boldsymbol{Th})$.

Example 5.2. The following example is a variation of Example 4.2. Here, we show an application of the just developed theory for equational reasoning in a multi-language context (an example involving programming languages will be shown in Chapter 6 in the context of categorical logic).

Consider the following order-sorted signatures:

- The order-sorted signature Sg_1 defines the symbols of a language for constructing simple mathematical expressions over natural numbers in Peano's notation: Let the set of sort $S_1 = \{nat\}$ be a set with a single sort *nat* denoting the type of natural numbers (and therefore, the relation $\leqslant_1$ must be the reflexive one on S_1). Then, the operators are specified by the symbols

$$\begin{aligned} 0 &: nat && \text{zero} \\ s &: nat \to nat && \text{successor} \\ + &: nat, nat \to nat && \text{addition} \end{aligned}$$

- Let $c \in \mathbb{A} = \{a, b, \ldots, z\}$ be the metavariable ranging over a finite set $\mathbb{A}$ of characters. The order-sorted signature Sg_2 defines a language to build strings over a finite alphabet of symbols $\mathbb{A}$. The set of sort $S_2 = \{chr, str\}$ carries the sort *str* for strings and the sort *c* for characters. The subsort relation $\leqslant_2$ is the reflexive relation on S_2 plus $chr \leqslant_2 str$ (i.e., characters are one-symbol strings), and the operators are

$$\begin{aligned} c &: chr && \text{character (for all } c \in \mathbb{A}) \\ next &: chr \to chr && \text{next character} \\ + &: str, str \to str && \text{string concatenation} \end{aligned}$$

Interpretations of Sg_1 and Sg_2 are given by the order-sorted algebras $\mathscr{A}_1$ and $\mathscr{A}_2$, respectively. Let the interpretation domain of $\mathscr{A}_1$ be $[\![nat]\!]_{\mathscr{A}_1} = \mathbb{N}$ and those of $\mathscr{A}_2$ be $[\![chr]\!]_{\mathscr{A}_2} = \mathbb{A} \subseteq \mathbb{A}^* = [\![str]\!]_{\mathscr{A}_2}$. Moreover, we define the interpretation functions as follows (the juxtaposition of two or more strings denotes their concatenation):

$$\begin{aligned} [\![0]\!]_{\mathscr{A}_1} &= 0 \\ [\![s : nat \to nat]\!]_{\mathscr{A}_1}(n) &= n + 1 \\ [\![+ : nat, nat \to nat]\!]_{\mathscr{A}_1}(n_1, n_2) &= n_1 + n_2 \end{aligned}$$

$$\begin{aligned} [\![c]\!]_{\mathscr{A}_2} &= c \\ [\![next : chr \to chr]\!]_{\mathscr{A}_2}(c) &= c' \\ [\![+ : str, str \to str]\!]_{\mathscr{A}_2}(s_1, s_2) &= s_1 s_2 \end{aligned}$$

where c' is the character following c in $\mathbb{A}$ (we are assuming the standard alphabetical order on $\mathbb{A}$). Since Sg_1 and Sg_2 are regular order-sorted signatures, then $\mathscr{A}_1$ and $\mathscr{A}_2$ induce unique semantic functions $h_1 : \mathscr{T}_{Sg_1} \to \mathscr{A}_1$ and $h_2 : \mathscr{T}_{Sg_2} \to \mathscr{A}_2$, providing the (initial algebra) semantics of the languages.

We now provide order-sorted equations for $Sg(Exp)$ and $Sg(Str)$ separately, and then we axiomatize the behaviour of boundary functions of the multi-language described in Example 4.2 by the use of multi-language equations. Finally, we apply the rules of deduction and the soundness theorem to prove that the multi-language semantics of $+(\hookrightarrow_{chr,nat}(c), n)$ and $s(n)$ is the same under any assignment (where c and n are variables of sort *chr* and *nat*, respectively). Let $Ax(X_{Sg(Exp)})$ be the set containing the following order-sorted $Sg(Exp)$-equations, where $X_{Sg(Exp)} = \{n : nat, m : nat\}$

is the set of variables:

$$(\forall X_{Sg(Exp)}) \qquad +(0,n) = n \tag{5.7}$$
$$(\forall X_{Sg(Exp)}) \qquad +(n,0) = n \tag{5.8}$$
$$(\forall X_{Sg(Exp)}) \qquad +(s(n),m) = s(+(n,m)) \tag{5.9}$$
$$(\forall X_{Sg(Exp)}) \qquad +(n,m) = +(m,n) \tag{5.10}$$

Similarly, let $Ax(X_{Sg(Str)})$ be the set of axioms containing the following order-sorted Sg_2-equations:

$$(\forall X_{Sg(Str)}) \qquad next(a) = b \tag{5.11}$$
$$(\forall X_{Sg(Str)}) \qquad next(b) = c \tag{5.12}$$
$$(\forall X_{Sg(Str)}) \qquad \ldots = \ldots \tag{5.13}$$
$$(\forall X_{Sg(Str)}) \qquad next(z) = a \tag{5.14}$$

Let us recall the definition of the multi-language signature $\boldsymbol{Sg}(Exp + Str) = (Sg(Exp), Sg(Str), \ltimes)$ and the multi-language $\boldsymbol{Sg}(Exp + Str)$-algebra $\mathscr{M}$ (with underlying order-sorted algebras $\mathscr{A}_1$ and $\mathscr{A}_2$ defined above, respectively). We define the interoperability constraints and the boundary functions as follows:

$$exp \ltimes chr \qquad [\![exp \ltimes chr]\!]_{\mathscr{M}} = \mathrm{chr}$$
$$nat \ltimes chr \qquad [\![nat \ltimes chr]\!]_{\mathscr{M}} = \mathrm{chr}$$
$$chr \ltimes nat \qquad [\![chr \ltimes nat]\!]_{\mathscr{M}} = \mathrm{ord}$$
$$str \ltimes nat \qquad [\![str \ltimes nat]\!]_{\mathscr{M}}(a_0 \ldots a_n) = \sum_{k=0}^{n} [\![chr \ltimes nat]\!]_{\mathscr{M}}(a_k) \cdot 10^{n-k}$$

We can make $Ax(X) = Ax(X_{Sg(Exp)}) \cup Ax(X_{Sg(Str)})$ with $X = X_{Sg(Exp)} \cup X_{Sg(Str)}$ to a set of multi-language $\boldsymbol{Sg}$-equations simply by annotating the order-sorted equations. For instance, $(\forall X_{Sg(Exp)})\ +(0,n) = n$ becomes $(\forall X)\ +(0,n) = n$ (in a real implementation, this process can be automatized). Note that a substantial difference arises when lifting an order-sorted equation to a multi-language context. Consider again $(\forall X_{Sg(Exp)})\ +(0,n) = n$. An order-sorted substitution can transform the variable $n\colon nat$ to an arbitrary term in $[\![nat]\!]_{\mathscr{T}_{Sg(Exp)}}$, whereas, in the lifted equation $(\forall X)\ +(0,n) = n$, a multi-language substitution can assign to n a value in $[\![nat]\!]_{\mathscr{T}_{\boldsymbol{Sg}(Exp+Str)}}$, which also contains multi-language terms of sort *nat*.

The behavior of boundary functions can be axiomatized by adding the following equations to $Ax(X)$ and the new variables $s, v\colon str$:

$$(\forall X) \qquad \hookrightarrow_{nat,chr}(0) = a$$
$$(\forall X) \qquad \hookrightarrow_{nat,str}(0) = a$$
$$(\forall X) \qquad \hookrightarrow_{chr,nat}(c) = s(0)$$
$$(\forall X) \qquad \hookrightarrow_{str,nat}(c) = s(0)$$
$$(\forall X) \qquad \hookrightarrow_{nat,chr}(s(n)) = next(\hookrightarrow_{nat,chr}(n))$$
$$(\forall X) \qquad \hookrightarrow_{nat,str}(s(n)) = next(\hookrightarrow_{nat,chr}(n))$$
$$(\forall X) \qquad \hookrightarrow_{str,nat}(+(s,v)) = +(\hookrightarrow_{str,nat}(s), \hookrightarrow_{str,nat}(n))$$

It is immediate that $\mathscr{M}$ satisfies all the equations in AxX.

We now prove a simple property of the multi-language by exploiting the equational logic. We want to show that the (multi-language) semantics of $+(\hookrightarrow_{chr,nat}(c), n)$ is the same of $s(n)$ under any possible assignment of c and n. More formally, given an arbitrary assignment function $a\colon X \to A$, where X contains c and n, we need to prove that

$$a^*(+(\hookrightarrow_{chr,nat}(c), n)) = a^*(s(n))$$

Thanks to the soundness theorem, it is sufficient to show that

$$(\forall X)\ \ +(\hookrightarrow_{chr,nat}(c), n) = s(n) \tag{5.15}$$

In order to prove Derivation 5.15, we actually prove that

$$+(\hookrightarrow_{chr,nat}(c), n) = +(s(0), n) \tag{5.16}$$

and

$$+(s(0), n) = s(n) \tag{5.17}$$

then, by Rule (mlTrans) we conclude 5.15.

- 5.16: It follows by applying Rule (mlCong) to the multi-language term $+(m, n)$ with substitutions θ and θ' given by $\theta_{nat}(n) = n = \theta'_{nat}(n)$, $\theta_{nat}(m) = \hookrightarrow_{chr,nat}(c)$, and $\theta'_{nat}(m) = s(0)$.
- 5.17: It follows by observing that

  $$+(s(0), n) = s(+(0, n))$$

  is derivable by Rule (mlSub) directly from Equation 5.9, and

  $$s(+(0, n)) = s(n)$$

  is derivable by applying Rule (mlCong) to the multi-language term $s(n)$, where the premise of the rule follows by Equation 5.9. Hence, by transitivity we obtain Derivation 5.17. ♢

CHAPTER 6

Categorical Logic for Multi-Languages

☞ Chapter reference(s): [BCM20a]

We lift the basic syntactic theories of *order-sorted equational logic*, and models of the theories, to the algebraic multi-language framework. The models in for order-sorted algebras (see Chapter 3) are built from sets, but we adapt the categorical approach in [MM96]. 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 chapter, and an elaborate application in the last section combines an imperative language and a lambda calculus. Our account of order-sorted equational theories builds on and refines [MM96], 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.

Structure 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. Then, we present a similar set of results for multi-languages. We follow the same course of exposition of Chapters 4 and 5: We begin by illustrating categorical version of multi-language signatures, algebras, and homomorphisms. Then, we define equations, theories, and a deduction system which shall be proved sound and complete. Finally, we lift the same results for all the multi-language constructions presented in Chapter 4 and we conclude with an example involving programming languages.

6.1 Order-Sorted Equational Logic

We review order-sorted equational theories (see for example [GM92, MM96]). 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 [MM96], 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.

6.1.1 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 $\leqslant$, and then $S^n = S \times \cdots \times S$ (n-times cartesian product for $n \geqslant 1$) inherits the pointwise order, with typical instances written $w \leqslant w'$. If $w \in S^n$, we usually make explicit its components by writing $w = s_1 \dots s_n$, sometimes referring to a *sequence* of sorts.

We write $(A_s \mid s \in S)$ for a *family indexed by* S where each A_s is sometimes a set, but more generally an object in a category C. We sometimes refer to the family as an S-*sorted set* (*resp.*, an S-*sorted object*). Such indexed families are simply functors A in the presheaf category Set^S (*resp.*, C^S). As such, an S-*sorted function* (*resp.*, S-*sorted morphism*) $h\colon A \to B$ is simply a morphism (that is, natural transformation) in Set^S (*resp.*, C^S) where S is a set or poset.

In this chapter all categories have finite products, and functors preserve them up to *isomorphism*. If A, B, and A_i ($1 \leqslant i \leqslant n$) are objects in a category C, we write $A \times B$ for the binary product of A and B, $A_1 \times \cdots \times A_n$ or $\prod_1^n A_i$ for the finite product of the A_i, and 1 for the terminal object. Mediating morphisms for binary product are written $\langle f, f' \rangle$, and as usual $f \times f' = \langle f \circ \pi, f' \circ \pi' \rangle$ (for suitable f and f' and the usual projections). We adopt the obvious extension of notation for finite products. If A is an S-sorted object and $w = s_1 \dots s_n$, we denote by A_w the product $A_{s_1} \times \cdots \times A_{s_n}$. Likewise, if f is an S-sorted morphism, then the morphism f_w is defined by $f_{s_1} \times \cdots \times f_{s_n}$. The coproduct object of A and B is written $A + B$. We write $l_1 \dots l_n$ or $l_1, \dots, l_n$ for a typical finite list, and we may abbreviate just to $\vec{l}$. In the special case of a list of sorts $s_1, \dots, s_n$ we usually abbreviate to w.

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 [GM92, MM96].

Order-sorted signature

Definition 6.1 (Order-Sorted Signature)**.** An *order-sorted signature Sg* is specified by

(i) a poset $(S, \leqslant)$ of *sorts*;

Function symbol

(ii) a *collection* of *function symbols* $f\colon s_1, \dots, s_n \to s$ each with *arity* $n \geqslant 1$ and $(w, s) \in S^+ \times S$ the *rank* of f, where $w = s_1 \dots s_n$;

Constant

(iii) a *collection* of *constants* $k\colon s$, each of a *unique rank* s (just a single sort); and

(iv) a *monotonicity requirement* that whenever $f\colon w_1 \to s$ and $f\colon w_2 \to r$ with $w_1 \leqslant w_2$, then $s \leqslant r$.

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

Remark 6.1. Compare the previous definition with Definition 3.1 of (set-theoretic) order-sorted signatures: Operators can now form a collection rather than a set. ♢

Notation 6.1. When we define order-sorted signatures Sg_1, Sg_2, Sg, Sg', etc., we shall implicitly assume that their posets of sorts are denoted by $(S_1, \leqslant_1)$, $(S_2, \leqslant_2)$, $(S, \leqslant)$, $(S', \leqslant')$, etc., respectively. ♢

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. (All signatures in this chapter are assumed regular).

We recall the definition of regularity, which remains unchanged from Chapter 3.

Regularity

Definition 6.2 (Regularity)**.** An order-sorted signature Sg is *regular* if for each function symbol $f\colon w \to s$ and for each lower bound $w_0 \leqslant w$ the set

$$\{(w', s') \in S^+ \times S \mid f\colon w' \to s' \wedge w_0 \leqslant w'\}$$

has a minimum, called the *least rank* of f with respect to w_0. ♢

Least rank

Raw terms

Raw terms over a signature Sg are defined by the context-free grammar

$$t ::= x \mid k \mid f(t_1, \ldots, t_n)$$

with $x \in \mathit{Var}$ (a countably infinite set of variables), k a constant, and f a function symbol with arity n.

Context

A *context* is a finite list of ordered pairs $x\colon s$ formed by a variable x and a sort s in Sg. We usually define a context by writing $\Gamma = [x_1\colon s_1, \ldots, x_n\colon s_n]$. We denote context concatenation of Γ and Γ' by Γ, Γ'.

Proved term

We work with sorting judgements of the form $\Gamma \vdash t\colon s$. Those that are generated by the sorting rules in Figure 6.1 are called *proved terms*. Note that a term t may have more than one sort s for which $\Gamma \vdash t\colon s$ is a proved term. However there is always a unique least sort.

Lemma 6.1 (Terms Have A Least Sort)**.** *Suppose that Γ is a context and t a raw term for a given regular signature Sg. If there is any sort s for which $\Gamma \vdash t\colon s$ is a proved term, then there is a least such sort,* $\mathrm{ls}(t)$.

Proof. The proof is analogous to the one of Proposition 3.3. One uses rule induction. The proof is easy, though in the literature a key step is often omitted. By induction, for the rule (Γ-Fun), one easily uses regularity to obtain a sort, say $\tilde{s}$, such that $\Gamma \vdash f(t_1, \ldots, t_n)\colon \tilde{s}$. Now $\tilde{s}$ is a candidate for the least sort of $f(t_1, \ldots, t_n)$. Most authors state that such a sort $\tilde{s}$ is least. This is true, but proving it so requires a separate (though trivial) rule induction. □

Substitution

We denote by $t[u/x]$ the *substitution* of the raw term u for the variable x in t, and by $t[\vec{u}/\vec{x}]$ the *simultaneous substitution* of raw terms $\vec{u} = u_1, \ldots, u_n$ for variables $\vec{x} = x_1, \ldots, x_n$.

Inclusion structure

Definition 6.3 (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 $A \rightarrowtail B$ 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);

$$(\Gamma\text{-Var})\ \frac{-}{\Gamma, x\colon s, \Gamma' \vdash x\colon s} \qquad (\Gamma\text{-Const})\ \frac{-}{\Gamma \vdash k\colon s}\ (k\colon s \text{ in } Sg)$$

$$(\Gamma\text{-Fun})\ \frac{(\forall 1 \leqslant i \leqslant n)\ \ \Gamma \vdash t_i\colon s_i}{\Gamma \vdash f(t_1, \ldots, t_n)\colon s}\ (f\colon s_1, \ldots, s_n \to s \text{ in } Sg)$$

$$(\Gamma\text{-Sub})\ \frac{\Gamma \vdash t\colon s}{\Gamma \vdash t\colon r}\ (s \leqslant r \text{ in } Sg)$$

Figure 6.1: Proved terms generated by an order-sorted signature *Sg*.

- if $\iota_1\colon A_1 \rightarrowtail B_1$ and $\iota_2\colon A_2 \rightarrowtail B_2$ are morphisms in I, then so to is $\iota_1 \times \iota_2\colon A_1 \times A_2 \rightarrowtail B_1 \times B_2$.

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

Order-sorted algebra

Definition 6.4 (Order-Sorted Algebra)**.** Given an order-sorted signature *Sg*, an *Sg-algebra* $\mathscr{A}$ in an FPI-category (C, I) is specified by

(i) an object $[\![s]\!]_{\mathscr{A}}$ in C for each sort s and object $[\![w]\!]_{\mathscr{A}} = [\![s_1]\!]_{\mathscr{A}} \times \cdots \times [\![s_n]\!]_{\mathscr{A}}$ for each $w = s_1 \ldots s_n \in S^n$;

(ii) morphisms $[\![f\colon w \to s]\!]_{\mathscr{A}}\colon [\![w]\!]_{\mathscr{A}} \to [\![s]\!]_{\mathscr{A}}$ and $[\![k]\!]_{\mathscr{A}}\colon 1 \to [\![s]\!]_{\mathscr{A}}$ for each $f\colon w \to s$ and $k\colon s$; and

(iii) a morphism $[\![s \leqslant r]\!]_{\mathscr{A}}\colon [\![s]\!]_{\mathscr{A}} \rightarrowtail [\![r]\!]_{\mathscr{A}}$ in I for each $s \leqslant r$ in S, where we set $[\![s \leqslant s]\!]_{\mathscr{A}} = \mathrm{id}_{[\![s]\!]_{\mathscr{A}}}$

such that if the function symbol f appears with more than one rank $f\colon w_1 \to s$ and $f\colon w_2 \to r$ in *Sg* with $s_1 \ldots s_n = w_1 \leqslant w_2 = r_1 \ldots r_n$, then the following diagram commutes:

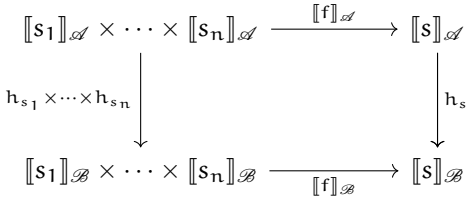
$$\begin{array}{ccc} [\![s_1]\!]_{\mathscr{A}} \times \cdots \times [\![s_n]\!]_{\mathscr{A}} & \xrightarrow{[\![f\colon w_1 \to s]\!]_{\mathscr{A}}} & [\![s]\!]_{\mathscr{A}} \\ \downarrow{\scriptstyle [\![s_1 \leqslant r_1]\!]_{\mathscr{A}} \times \cdots \times [\![s_n \leqslant r_n]\!]_{\mathscr{A}}} & & \downarrow{\scriptstyle [\![s \leqslant r]\!]_{\mathscr{A}}} \\ [\![r_1]\!]_{\mathscr{A}} \times \cdots \times [\![r_n]\!]_{\mathscr{A}} & \xrightarrow[[\![f\colon w_2 \to r]\!]_{\mathscr{A}}]{} & [\![r]\!]_{\mathscr{A}} \end{array}$$

♢

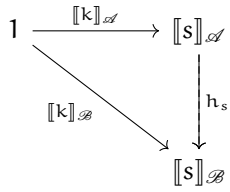
Notation 6.2. From now on, we might drop the algebra subscript and the ranks of function symbols in the semantic brackets whenever they are clear by context. ♢

Definition 6.5 (Order-Sorted Homomorphism)**.** Let *Sg* be an order-sorted signature and let $\mathscr{A}$ and $\mathscr{B}$ be *Sg*-algebras. An *Sg-homomorphism* $h\colon \mathscr{A} \to \mathscr{B}$ is an S-sorted morphism $(\, h_s\colon [\![s]\!]_{\mathscr{A}} \to [\![s]\!]_{\mathscr{B}} \mid s \in S \,)$ such that given $f\colon s_1, \ldots, s_n \to s$, $k\colon s$, and $s \leqslant r$ in *Sg* the following diagrams commute:

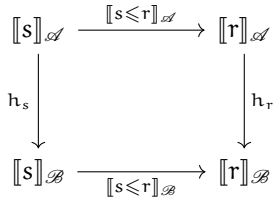
(i)



(ii)



(iii)



We define $h_w = h_{s_1} \times \cdots \times h_{s_n}$ provided that $w = s_1 \dots s_n$. ♢

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 $Alg(C, I)_{Sg}$.

If $\Gamma \vdash t : s$ is a proved term in a *regular* signature *Sg* and $\Gamma = [x_1 : s_1, \dots, x_n : s_n]$, any *Sg*-algebra $\mathscr{A}$ induces a (unique) morphism from $[\![\Gamma]\!]_{\mathscr{A}} = [\![s_1]\!]_{\mathscr{A}} \times \cdots \times [\![s_n]\!]_{\mathscr{A}}$ to $[\![s]\!]_{\mathscr{A}}$ in *C* according to the inductive definition that appears in Figure 6.2. We denote such an arrow by $[\![\Gamma \vdash t : s]\!]_{\mathscr{A}}$ and we refer to it as the *semantics of* $\Gamma \vdash t : s$.

Semantics of proved terms

Since terms can be assigned different types in one given context, we should consider whether the definition in Figure 6.2 is a sensible one. As such, we have the following lemma, where one sees that semantics of substitutions of terms is given as usual by morphism composition:

Lemma 6.2 (Well-Defined Semantics). *Given a proved term* $\Gamma \vdash t : s$ *and an algebra* $\mathscr{A}$ *over a regular signature Sg:*

- *The semantic morphism* $[\![\Gamma \vdash t : s]\!]$ *is unique; that is, the assignment* $\xi \mapsto [\![\xi]\!]$ *is a total function.*
- *The algebra induces a functor* $(S, \leqslant) \to I$ *between posetal categories, where* $s \leqslant s' \mapsto [\![s \leqslant s']\!] : [\![s]\!] \rightarrowtail [\![s']\!]$*, and so as a consequence the semantics can be factored through the morphism* $[\![\Gamma \vdash t : ls(t)]\!]$*, that is to say,*

$$[\![\Gamma \vdash t : s]\!] = [\![ls(t) \leqslant s]\!] \circ [\![\Gamma \vdash t : ls(t)]\!]$$

$$\frac{-}{[\![\Gamma, x\colon s, \Gamma' \vdash x\colon s]\!] = \pi\colon [\![\Gamma]\!] \times [\![s]\!] \times [\![\Gamma']\!] \to [\![s]\!]}$$

$$\frac{-}{[\![\Gamma \vdash k\colon s]\!] = [\![k]\!] \circ !\colon [\![\Gamma]\!] \to 1 \to [\![s]\!]}\ (k\colon s \text{ in } Sg)$$

$$\frac{\begin{array}{c}{}^{\dagger}(f\colon s_1, \dots, s_n \to s \text{ in } Sg) \\ (\forall 1 \leqslant i \leqslant n)\ [\![\Gamma \vdash t_i\colon s_i]\!] = m_i\colon [\![\Gamma]\!] \to [\![s_i]\!]\end{array}}{[\![\Gamma \vdash f(t_1, \dots, t_n)\colon s]\!] = [\![f]\!] \circ \langle m_1, \dots, m_n\rangle\colon [\![\Gamma]\!] \to (\prod_1^n [\![s_i]\!]) \to [\![s]\!]}\ \dagger$$

$$\frac{[\![\Gamma \vdash t\colon s]\!] = m\colon [\![\Gamma]\!] \to [\![s]\!]}{[\![\Gamma \vdash t\colon r]\!] = [\![s \leqslant r]\!] \circ m\colon [\![\Gamma]\!] \to [\![s]\!] \to [\![r]\!]}\ (s \leqslant r \text{ in } Sg)$$

Figure 6.2: Categorical semantics for proved terms.

- *Let* $\Gamma = [x_1\colon s_1, \dots, x_n\colon s_n]$ *be a context. If we have proved terms* $\Gamma \vdash t\colon s$ *and* $\Gamma' \vdash u_i\colon s_i$ *where* $1 \leqslant i \leqslant n$*, then* $\Gamma' \vdash t[\vec{u}/\vec{x}]\colon s$ *and*

$$[\![\Gamma' \vdash t[\vec{u}/\vec{x}]\colon s]\!] = [\![\Gamma \vdash t\colon s]\!] \circ \langle [\![\Gamma' \vdash u_1\colon s_1]\!], \dots, [\![\Gamma' \vdash u_n\colon s_n]\!]\rangle$$

Proof. Uniqueness follows by a simple induction over proved terms, as does the factorisation. However, note that the proof relies on the functoriality: Since I is posetal, $\xi \mapsto [\![\xi]\!]$ must be functorial on compositions $s \leqslant s' \leqslant s'' = s \leqslant s''$ in the poset of sorts $(S, \leqslant)$. □

Coherence

Equations between proved terms are defined only for *coherent* signatures. To define coherence, first let $\equiv$ be the symmetric and transitive closure of $\leqslant$. The equivalence classes induced by $\equiv$ on S are the *connected components* of $(S, \leqslant)$; and $(S, \leqslant)$ is *locally filtered*, if for every two sorts s' and s'' in the same *connected component* there is a sort s such that $s', s'' \leqslant s$. 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 s' and s'' respectively could potentially be judged equal if they have a "common supersort" s. That is, coherence is defined analogously to Definition 3.12.

Equation (in-context)

Definition 6.6 (Order-Sorted Equation)**.** Let *Sg* be a coherent signature. An *equation (in-context)* in *Sg* is denoted by $\Gamma \vdash t = t'\colon s$, where

- there are sorts s', s'' such that $\Gamma \vdash t\colon s'$ and $\Gamma \vdash t'\colon s''$ are proved terms in *Sg*;
- s' and s'' fall in the same connected component of $(S, \leqslant)$; and
- s is a common supersort of s' and s'' (which exists by the coherence condition).

Conditional equation (in-context)

A *conditional equation (in-context)* in *Sg* is a list of $m + 1$ equations-in-context (where $m \geqslant 1$) suggestively denoted by

$$\Gamma \vdash t = t'\colon s \iff \bigwedge_{\alpha=1}^{m} \Gamma \vdash t_\alpha = t'_\alpha\colon s_\alpha \qquad \diamondsuit$$

Definition 6.7 (Satisfaction of Order-Sorted Equations)**.** Let $\mathscr{A}$ be an Sg-algebra in an FPI-category (C, I).

- We say that $\mathscr{A}$ *satisfies* an Sg-equation if

$$[\![\Gamma \vdash t\colon s]\!] = [\![\Gamma \vdash t'\colon s]\!]$$

- We say that $\mathscr{A}$ *satisfies* an Sg-conditional equation if for all morphisms $u\colon U \to [\![\Gamma]\!]$ in the category C,

$$[\![\Gamma \vdash t_\alpha\colon s_\alpha]\!] \circ u = [\![\Gamma \vdash t'_\alpha\colon s_\alpha]\!] \circ u \implies [\![\Gamma \vdash t\colon s]\!] \circ u = [\![\Gamma \vdash t'\colon s]\!] \circ u$$

♢

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 6.3.

Order-sorted theory

Lemma 6.3 (Generalised Substitution)**.** *The following rule is admissible by a routine rule induction*

$$(\Gamma\text{-GSub})\ \frac{\Gamma \vdash t = t'\colon s \qquad (\forall 1 \leqslant i \leqslant n)\ \ \Gamma' \vdash u_i\colon s_i}{\Gamma' \vdash t[\vec{u}/\vec{x}] = t'[\vec{u}/\vec{x}]\colon s}\ (\Gamma = [x_1\colon s_1, \ldots, x_n\colon s_n])$$

Let $Th = (Sg, Ax)$ be an order-sorted theory. If an Sg-algebra $\mathscr{A}$ satisfies all the axioms in Ax, we call $\mathscr{A}$ a *model* of Th. The *category of models* $Mod(C, I)_{Th}$ is the full subcategory of $Alg(C, I)_{Sg}$ given by all the models of Th in (C, I).

Model of a theory, Category of models

Lemma 6.4 (Satisfaction is Well-Defined)**.** *As a consequence of Lemma 6.2, satisfaction is well-defined up to subsort-polymorphic equality, as follows: Suppose that we have a theorem $\Gamma \vdash t = t'\colon s$ satisfied in a model $\mathscr{A}$. If $\Gamma \vdash t = t'\colon \tilde{s}$ is also a theorem, then it too is satisfied.*

Proof. The existence of least sorts $\mathrm{ls}(t)$ and $\mathrm{ls}(t')$ means that s and $\tilde{s}$ are connected, and so have a supersort $s' \geqslant s, \tilde{s}$. Thus each term has this type s', and the result follows by using factorisation from Lemma 6.2 and the left-cancellation properties of monomorphisms. □

The category $FPI\big((C, I), (\mathcal{D}, \mathcal{J})\big)$ is defined by having objects functors $F\colon C \to \mathcal{D}$ such that finite products are preserved and F restricts to a functor $F|_I\colon I \to \mathcal{J}$ (that is monics are also preserved). Suppose that we have a model $\mathscr{A}$ in $Mod(C, I)_{Th}$. Then there is a model $F_*\mathscr{A}$ in $Mod(\mathcal{D}, \mathcal{J})_{Th}$ that is, roughly speaking, defined by "taking the image of Sg in (C, I) induced by the model $\mathscr{A}$, 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 [Cro93, J$^+$02]) leads to a functor

$$Ap\colon FPI\big((C, I), (\mathcal{D}, \mathcal{J})\big) \to Mod(\mathcal{D}, \mathcal{J})_{Th}$$

A *classifying category* $Cl(Th)$ for a theory Th is an FPI-category such that there is an equivalence of categories

Classifying category

$$Ap_{\mathscr{G}}\colon FPI\big(Cl(Th), (\mathcal{D}, \mathcal{J})\big) \simeq Mod(\mathcal{D}, \mathcal{J})_{Th}$$

$$\text{(Γ-AxSub)}\ \frac{\Gamma \vdash t = t' : s \in Ax \quad (\forall 1 \leqslant i \leqslant n)\ \ \Gamma' \vdash u_i : s'_i}{\Gamma' \vdash t[\vec{u}/\vec{x}] = t'[\vec{u}/\vec{x}] : s}$$

$$\text{(Γ-AxCSub)}\ \frac{\Gamma \vdash t = t' : s \iff \bigwedge_1^m\ \Gamma \vdash t_\alpha = t'_\alpha : s_\alpha \in Ax \quad (\forall 1 \leqslant i \leqslant n)\ \ \Gamma' \vdash u_i : s'_i \quad (\forall 1 \leqslant \alpha \leqslant m)\ \ \Gamma' \vdash t_\alpha[\vec{u}/\vec{x}] = t'_\alpha[\vec{u}/\vec{x}] : s_\alpha}{\Gamma' \vdash t[\vec{u}/\vec{x}] = t'[\vec{u}/\vec{x}] : s}$$

Let $\Gamma = [x_1 : s'_1, \ldots, x_n : s'_n]$ be a context in (Γ-AxSub) and (Γ-AxCSub).

$$\text{(Γ-Ref)}\ \frac{\Gamma \vdash t : s}{\Gamma \vdash t = t : s} \qquad \text{(Γ-Sym)}\ \frac{\Gamma \vdash t = t' : s}{\Gamma \vdash t' = t : s}$$

$$\text{(Γ-Trans)}\ \frac{\Gamma \vdash t = t' : s \qquad \Gamma \vdash t' = t'' : s}{\Gamma \vdash t = t'' : s}$$

$$\text{(Γ-Sub)}\ \frac{\Gamma \vdash t = t' : s}{\Gamma \vdash t = t' : r}\ (s \leqslant r)$$

$$\text{(Γ-Cong)}\ \frac{\Gamma \vdash t : s \qquad (\forall 1 \leqslant i \leqslant n)\ \ \Gamma' \vdash u_i = u'_i : s_i}{\Gamma' \vdash t[\vec{u}/\vec{x}] = t[\vec{u'}/\vec{x}] : s}\ (\Gamma = [x_1 : s_1, \ldots, x_n : s_n])$$

Figure 6.3: Theorems generated by an order-sorted theory $Th = (Sg, Ax)$.

Generic model

where $\mathscr{G}$ is a model of the theory Th in $Cl(Th)$. Such a model $\mathscr{G}$ will be called *generic model*. Thus, models of Th in $(\mathcal{D}, \mathcal{J})$ correspond to such structure preserving functors with source $Cl(Th)$.

At an abstract level this notion is standard in categorical type theory/logic. Nevertheless, we feel that our concrete construction is simpler than that found in [MM96], 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 [Cro12, Pit16] 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).

Theorem 6.1 (Existence of Classifying Category)**.** *There is an FPI-category $Cl(Th)$ constructed out of the syntax of $Th = (Sg, Ax)$, in which there is a* generic *model of Th with the property that equality of morphisms corresponds to derivability of term equations.*

Proof (Construction). The objects are finite lists of sorts, with a typical object $\vec{s}$ or s. Given such an object a context Γ *matches* an object $\vec{s} = s_1 \ldots s_n$ if $\Gamma = [x_1 : s_1, \ldots, x_n : s_n]$ for any x_i. Now fix arbitrary $\vec{s}$ and s'. Consider the set

$$\{\, (\Gamma, t) \mid \Gamma \text{ matches } \vec{s} \text{ and } \Gamma \vdash t : s' \,\}$$

We can define an equivalence relation on this set by $(\Gamma, t) \sim (\Gamma', t')$ just in case $\Gamma \vdash t = \pi t' : s'$ where the permutation π on *Var* swaps the variables in Γ' to those in Γ, preserving matching. Write $(\Gamma \mid t)$ for an equivalence class. Then a morphism $\vec{s} \to \vec{s}'$ is a list of equivalence classes $(\Gamma \mid t_1) \dots (\Gamma \mid t_m)$ where the terms have types matching $\vec{s}'$. We sometimes write $(\Gamma \mid \vec{t})$ for such a morphism.

This is a category where composition is given in the usual way by substitution of terms:

$$(\Gamma \mid \vec{t}') \circ (\Gamma \mid \vec{t}) = (\Gamma \mid \vec{t}'[\vec{t}/\vec{x}])$$

and identities are lists of the form

$$(\Gamma \mid \vec{x}) = (\Gamma \mid x_1) \dots (\Gamma \mid x_n)$$

Note that one needs to verify in detail that this construction is well-defined; for composition one appeals to rules (Γ-GSub) and (Γ-Cong).

The terminal object is ε. The binary product of $\vec{s}$ and $\vec{s}'$ is $\vec{s}\vec{s}'$, with projections $(\Gamma, \Gamma' \mid \vec{x})$ and $(\Gamma, \Gamma' \mid \vec{x}')$, and if we are given morphisms $(\Gamma' \mid \vec{t}) : \vec{r} \to \vec{s}$ and $(\Gamma' \mid \vec{t}') : \vec{r} \to \vec{s}$ then the mediating morphism is $(\Gamma \mid \vec{t}\vec{t}')$.

The objects of the inclusion category are all those of the main category $Cl(Th)$. Obviously every identity of $Cl(Th)$ is in the inclusion category. More generally there is a morphism $\vec{s} \to \vec{r}$ just in case the objects are the same length n and moreover $\vec{s} \leqslant \vec{r}$ in S^n. When this is so, the (single) morphism is $(\Gamma \mid \vec{x}) : \vec{s} \to \vec{r}$. The fact that this is a monomorphism in $Cl(Th)$ follows from (Γ-Sub).

Let $\Gamma = [x_1 : s_1, \dots, x_n : s_n]$. The so-called *generic model* $\mathscr{G}$ is defined by

- objects $[\![s]\!]_{\mathscr{G}} = s$ for each sort s and objects $[\![w]\!]_{\mathscr{G}} = \vec{w}$ for each $w = s_1 \dots s_n \in S^n$;
- morphisms $[\![f : w \to s]\!]_{\mathscr{G}} = (\Gamma \mid f(x_1, \dots, x_n)) : \vec{w} \to s$ and $[\![k : s]\!]_{\mathscr{G}} = (\varepsilon \mid k) : 1 \to s$; and
- morphisms $[\![s \leqslant r]\!]_{\mathscr{G}} = (x : s \mid x) : s \rightarrowtail r$ in the inclusion category for each $s \leqslant r$ in S; we set $[\![s \leqslant s]\!]_{\mathscr{G}} = \mathrm{id}_s$.

One can then show that $[\![\Gamma \vdash t : s]\!]_{\mathscr{G}} = (\Gamma \mid t)$. □

Theorem 6.2 (Soundness and Completeness)**.** *Let Th be a coherent order-sorted theory. $\Gamma \vdash t = t' : s$ is a theorem generated by Th if and only if $\Gamma \vdash t = t' : s$ is satisfied by every model of Th.*

Proof. Soundness follows by rule induction for Figure 6.3. For completeness, suppose that $\Gamma \vdash t = t' : s$ is satisfied in any model. Then in particular it is satisfied in the generic model $\mathscr{G}$ in the classifying category $Cl(Th)$. Thus we have $[\![\Gamma \vdash t : s]\!]_{\mathscr{G}} = [\![\Gamma \vdash t' : s]\!]_{\mathscr{G}}$ and so we have $(\Gamma \mid t) = (\Gamma \mid t')$ which holds precisely when $\Gamma \vdash t = t' : s$ is a theorem. □

We conclude this section with a new result, although it is motivated by analogous theorems [Pit00]. The proof also makes use of matching contexts and permutations of variables.

Theorem 6.3 (Relationship to Free Algebras)**.** *There is an equivalence between FPI-categories $Cl(Th)$ and $(FreeAlg^{op}, \mathcal{J})$ where FreeAlg is the category of free order-sorted algebras over finite sets of variables, and order-sorted homomorphisms. Moreover the equivalence is given by an FPI-functor $\Phi : Cl(Th) \simeq (FreeAlg^{op}, \mathcal{J})$.*

Proof. Let *Var* be a (countable) fixed set of variables $\{V_1, V_2, \ldots\}$. We call the context

$$\Gamma_{\vec{s}} = [V_1 \colon s_1, \ldots, V_n \colon s_n]$$

Primary context

the *primary context* for any sorts $s_1, \ldots, s_n$. In general, below, the metavariables x, y, z etc., possibly subscripted, *range over Var*. Thus $\Gamma = [x_1 \colon s_1, \ldots, x_n \colon s_n]$ is a typical context as before, and we say that any such context *matches* $\vec{s} = s_1, \ldots, s_n$.

First we define the objects $\mathscr{T}_{Th}(\Gamma)$ of *FreeAlg*. Recall the definition of the free algebra $\mathscr{T}_{Sg}(X)$ over an S-sorted set X of variables (see Section 3.5). Then, we define

$$\mathscr{T}_{Th}(\Gamma) = \mathscr{T}_{Th}(X_\Gamma) = \mathscr{T}_{Sg}(X_\Gamma)/{\sim}$$

where $(X_\Gamma)_s = \{\, x_i \mid x_i \colon s_i \in \Gamma \text{ and } s_i = s \,\}$. In particular, the interpretation sets of $\mathscr{T}_{Th}(\Gamma)$ are sets of equivalence classes

$$[\![s]\!]_{\mathscr{T}_{Th}(\Gamma)} = [\![s]\!]_{\mathscr{T}_{Sg}(X_\Gamma)/\sim} \qquad \text{where} \qquad [\![s]\!]_{\mathscr{T}_{Sg}(X_\Gamma)} = \{\, t \mid \Gamma \vdash t \colon s \text{ in } Sg \,\}$$

and $\sim$ is the usual congruence defined in 3.5, that is $t \sim_s t'$ just in case we can derive $\Gamma \vdash t = t' \colon s$ in *Th*, and write $[\,t\,]$ for a typical equivalence class.

Let Γ and Γ' be two context matching $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_m$, respectively. The morphisms $h \colon \mathscr{T}_{Th}(\Gamma) \to \mathscr{T}_{Th}(\Gamma')$ must be S-sorted functions

$$(\, h_s \colon [\![s]\!]_{\mathscr{T}_{Th}(\Gamma)} \to [\![s]\!]_{\mathscr{T}_{Th}(\Gamma')} \mid s \in S \,)$$

These are specified by lists $h_s = \{\Gamma' \vdash t_1, \ldots, t_n\}$ where $t_i \in [\![s_i]\!]_{\mathscr{T}_{Sg}(X_{\Gamma'})}$ and where

$$h_s([\,t\,] \in [\![s]\!]_{\mathscr{T}_{Th}(\Gamma)}) = [\,t[t_1, \ldots, t_m/x_1, \ldots, x_m]\,] \in [\![s]\!]_{\mathscr{T}_{Sg}(X_{\Gamma'})}$$

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

$$h = \{\Gamma' \vdash t_1, \ldots, t_n\} \colon \mathscr{T}_{Th}(\Gamma) \to \mathscr{T}_{Th}(\Gamma')$$

and

$$h' = \{\Gamma'' \vdash t'_1, \ldots, t'_m\} \colon \mathscr{T}_{Th}(\Gamma') \to \mathscr{T}_{Th}(\Gamma'')$$

then we have $h \circ h'$ defined by

$$\{\Gamma'' \vdash t_1[t'_1, \ldots, t'_m/x_1, \ldots, x_m], \ldots, t_n[t'_1, \ldots, t'_m/x_1, \ldots, x_m]\}$$

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

With a view to showing that $(\mathit{FreeAlg}^{\mathrm{op}}, \mathcal{J})$ is an FPI-category, we shall show that *FreeAlg* is has finite coproducts, and then define $\mathcal{J}$. Given objects $\mathscr{T}_{Th}(\Gamma)$ and $\mathscr{T}_{Th}(\Gamma')$ with Γ and Γ' matching $\vec{s} = s_1, \ldots, s_n$ and $\vec{s}' = s'_1, \ldots, s'_m$, respectively, then the binary coproduct object is given by $\mathscr{T}_{Th}(\Delta)$ with $\Delta = \Gamma, \Gamma'$. The coproduct insertions are given by $\{\Delta \vdash v_1, \ldots, v_n\}$ and $\{\Delta \vdash v_{n+1}, \ldots, v_{n+m}\}$. Given morphisms

$$\{\Gamma'' \vdash t_1, \ldots, t_n\} \colon \mathscr{T}_{Th}(\Gamma) \to \mathscr{T}_{Th}(\Gamma'')$$

and

$$\{\Gamma'' \vdash t'_1, \ldots, t'_m\} \colon \mathscr{T}_{Th}(\Gamma') \to \mathscr{T}_{Th}(\Gamma'')$$

then the mediating morphism is $\{\Gamma'' \vdash t_1, \ldots, t_n, t'_1, \ldots, t'_m\}$. Note that $\mathscr{T}_{Th}(\varepsilon)$ is the initial object (*cfr.* Corollary 3.2).

Now, let $\Gamma_{\vec{s}}$ and $\Gamma_{\vec{r}}$ be two context matching $\vec{s} = s_1, \ldots, s_n$ and $\vec{r} = r_1, \ldots, r_n$. Suppose that $s_i \leqslant r_i$ for each $1 \leqslant i \leqslant n$. Then there is an epic morphism

$$i = \{\Gamma_{\vec{s}} \vdash x_1, \ldots, x_n\} \colon \mathscr{T}_{Th}(\Gamma_{\vec{r}}) \to \mathscr{T}_{Th}(\Gamma_{\vec{s}}))$$

It is easy to verify that this is an epimorphism, and hence yields a monomorphism in $\mathit{FreeAlg}^{\mathrm{op}}$. The luff subcategory $\mathcal{J}$ has all of its morphisms the monomorphisms

$$i^{\mathrm{op}} = \{\Gamma_{\vec{s}} \vdash x_1, \ldots, x_n\} \colon \mathscr{T}_{Th}(\Gamma_{\vec{s}}) \to \mathscr{T}_{Th}(\Gamma_{\vec{r}})$$

This is certainly an inclusion category.

Now we prove the equivalence. We define a functor $\Phi \colon \mathit{Cl}(\mathit{Th}) \to (\mathit{FreeAlg}^{\mathrm{op}}, \mathcal{J})$ as follows. Given a morphism $(\Gamma \mid t_1) \ldots (\Gamma \mid t_m) \colon \vec{s} \to \vec{r}$ then Φ sends this to

$$\{\Gamma_{\vec{s}} \vdash \pi\, t_1, \ldots, \pi\, t_m\} \colon \mathscr{T}_{Th}(\Gamma_{\vec{r}}) \to \mathscr{T}_{Th}(\Gamma_{\vec{s}})$$

where permutation π is specified by $\pi \colon x_i \mapsto V_i$. We check this is well defined. Suppose that

$$(\Gamma \mid t_1) \ldots (\Gamma \mid t_m) = (\Gamma' \mid t'_1) \ldots (\Gamma' \mid t'_m).$$

We need to check that

$$\begin{aligned} \{\Gamma_{\vec{s}} \vdash \pi\, t_1, \ldots, \pi\, t_m\} &= \Phi((\Gamma \mid t_1) \ldots (\Gamma \mid t_m)) \\ &= \Phi((\Gamma' \mid t'_1) \ldots (\Gamma \mid t'_m)) \\ &= \{\Gamma_{\vec{s}} \vdash \pi'\, t'_1, \ldots, \pi'\, t'_m\} \end{aligned}$$

By definition we have $\Gamma \vdash t_j = \rho\, t'_j \colon r_j$ where ρ is specified by $\rho \colon x'_i \mapsto x_i$. We can deduce, using substitution rules for equations, that $\pi\,\Gamma \vdash \pi\, t_j = \pi(\rho\, t'_j) \colon r_j$ and this is exactly $\Gamma_{\vec{s}} \vdash \pi\, t_j = \pi'\, t'_j \colon r_j$ as required, since $\pi' = \pi \circ \rho$. 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.

- Φ *is essentially surjective.* For any object $\vec{s}$ in $\mathit{Cl}(\mathit{Th})$ we have $\Phi(\vec{s}) = \mathscr{T}_{Th}(\Gamma_{\vec{s}})$. But one easily shows that for any $\mathscr{T}_{Th}(\Gamma)$ where Γ matches $\vec{s}$, we have $\mathscr{T}_{Th}(\Gamma) \cong \mathscr{T}_{Th}(\Gamma_{\vec{s}})$ where the inverse homomorphisms "swap variables" x_i and V_i.
- Φ *is faithful.* Let

  $$\Phi((\Gamma \mid t_1) \ldots (\Gamma \mid t_m)) = \Phi((\Gamma' \mid t'_1) \ldots (\Gamma \mid t'_m)).$$

  We need to check that $\Gamma \vdash t_j = \rho\, t'_j \colon r_j$. By the assumption we have

  $$\{\Gamma_{\vec{s}} \vdash \pi\, t_1, \ldots, \pi\, t_m\} = \{\Gamma_{\vec{s}} \vdash \pi'\, t'_1, \ldots, \pi'\, t'_m\}$$

  Hence $\Gamma_{\vec{s}} \vdash \pi\, t_j = \pi\, t'_j \colon r_j$. Therefore we can deduce that $\Gamma \vdash t_j = (\pi^{-1} \circ \pi')\, t'_j \colon r_j$. We are done since $\pi^{-1} \circ \pi' = \rho$.
- Φ *is full.* Let

  $$\begin{aligned} \{\Gamma_{\vec{s}} \vdash t_1, \ldots, t_m\} &\colon \Phi(\vec{r}) \to \Phi(\vec{s}) \\ &\colon \mathscr{T}_{Th}(\Gamma_{\vec{r}}) \to \mathscr{T}_{Th}(\Gamma_{\vec{s}}) \end{aligned}$$

  Then $(\Gamma_{\vec{s}} \mid t_1, \ldots, t_m)$ is the appropriate $\mathit{Cl}(\mathit{Th})$ morphism.
- Φ *is an object of* $\mathit{FPI}\big(\mathit{Cl}(\mathit{Th}), (\mathit{FreeAlg}^{\mathrm{op}}, \mathcal{J})\big)$. We need to check that $\Phi|_I \colon I \to \mathcal{J}$ where I is the inclusion category of $\mathit{Cl}(\mathit{Th})$. Let $\vec{s} \leqslant \vec{r}$. Note that

  $$\Phi((\Gamma \mid x_1) \ldots (\Gamma \mid x_n) \colon \vec{s} \to \vec{r})$$

  is the monomorphism $\{\Gamma_{\vec{s}} \vdash x_1, \ldots, x_n\} \colon \Gamma_{\vec{r}} \to \Gamma_{\vec{s}}$. □

$0\colon nat$	zero constant
$s\colon nat \to nat$	successor function
$+\colon nat, nat \to nat$	addition operator

(a) Sg_1 operators.

$c\colon chr$	character constant (for each $c \in \mathbb{A}$)
$next\colon chr \to chr$	next character function
$+\colon str, str \to str$	string concatenation

(b) Sg_2 operators.

Figure 6.6: Operators of order-sorted signatures Sg_1 and Sg_2.

6.2 Multi-Language Equational Logic

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. Note that this example is the categorical equivalent of the running example in Chapter 5. See Section 6.4 for the outline of a more complex example, involving programming languages.

Running Example 6.1. Our example is defined using the following order-sorted signatures:

- 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 $(S_1, \leqslant_1)$ 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 6.4(a).
- Let $c \in \mathbb{A} = \{a, b, \ldots, z\}$ be the metavariable ranging over a finite set $\mathbb{A}$ of characters. The signature Sg_2 defines a language to build strings over $\mathbb{A}$. The set of sorts S_2 of Sg_2 carries the sort *str* for strings and the sort *chr* for characters. The subsort relation $\leqslant_2$ is the reflexive relation on S_2 plus $chr \leqslant_2 str$ (i.e., characters are one-symbol strings), and the operator symbols in Sg_2 appear in Figure 6.5(b).

We model Sg_1 and Sg_2 by the order-sorted algebras $\mathscr{A}_1$ and $\mathscr{A}_2$ (see Figure 6.9) in $(Set, Incl)$, the FPI-category of sets with inclusion functions forming the inclusion structure. The symbol c' in the definition of $[\![next\colon chr \to chr]\!]_{\mathscr{A}_2}$ denotes the character that follows c in $\mathbb{A}$ (assuming the standard alphabetical order). ♢

6.2.1 Fundamentals of Multi-Languages

The forthcoming definitions and results gradually define and illustrate the theory of multi-languages, and give relationships between multi-languages and order-sorted languages. A multi-language signature 6.8 is specified as two order-sorted signatures (as in the running example) together with *an interoperability relation* between the two

Sorts
$$[\![nat]\!]_{\mathscr{A}_1} = \mathbb{N}$$

Operators
$$[\![0]\!]_{\mathscr{A}_1} = * \mapsto 0 \colon 1 \to \mathbb{N}$$
$$[\![s \colon nat \to nat]\!]_{\mathscr{A}_1} = n \mapsto n + 1 \colon \mathbb{N} \to \mathbb{N}$$
$$[\![+ \colon nat, nat \to nat]\!]_{\mathscr{A}_1} = (n_1, n_2) \mapsto n_1 + n_2 \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

(a) Sg_1 semantics.

Sorts
$$[\![chr]\!]_{\mathscr{A}_2} = \mathbb{A}$$
$$[\![str]\!]_{\mathscr{A}_2} = \mathbb{A}^*$$

Operators
$$(\forall c \in \mathbb{A})\ [\![c]\!]_{\mathscr{A}_2} = * \mapsto c \colon 1 \to \mathbb{A}$$
$$[\![next \colon chr \to chr]\!]_{\mathscr{A}_2} = c \mapsto c' \colon \mathbb{A} \to \mathbb{A}$$
$$[\![+ \colon str, str \to str]\!]_{\mathscr{A}_2} = (s_1, s_2) \mapsto s_1 s_2 \colon \mathbb{A}^* \times \mathbb{A}^* \to \mathbb{A}^*$$

Subsorts
$$[\![chr \leqslant_2 str]\!]_{\mathscr{A}_2} = c \mapsto c \colon \mathbb{A} \to \mathbb{A}^*$$

(b) Sg_2 semantics.

Figure 6.9: Categorical semantics of Sg_1 and Sg_2.

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. 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).

Notation 6.3. In the following, if S_1 and S_2 are two sets of sorts and $s \in S_i$ with $i = 1, 2$, we write s_i for the element $\iota_i(s) \in S_1 + S_2$ where

$$\iota_i(s) = (s, i) \in S_1 \times \{1\} \cup S_2 \times \{2\} \qquad \diamondsuit$$

Remark 6.2. Among the many benefits of a categorical semantics, one retain more generality than set-theoretic semantics. In this spirit, note that we dropped the assumption that S_1 and S_2 are disjoint sets and instead we use disjoint union. $\diamondsuit$

Thus, in the following relationships $s_i \ltimes s'_j$ we have $s \in S_i$ and $s' \in S_j$. This is a very useful notation but perhaps requires care on first reading. Moreover, if $w = s_1 \dots s_n \in S_i^n$, then we write w_i for $(s_1)_i \dots (s_n)_i$.

Definition 6.8 (Multi-Language Signature)**.** A *multi-language signature* $\boldsymbol{Sg}$ is given by a triple $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$, that is

Multi-language signature

(i) a pair of order-sorted signatures Sg_1 and Sg_2 with posets of sorts $(S_1, \leqslant_1)$ and $(S_2, \leqslant_2)$, respectively; and

(ii) a *(binary) relation join* $\ltimes$ over $S_1 + S_2$ such that $s_i \ltimes s'_j$ with $i, j \in \{1, 2\}$ and $i \neq j$. $\diamondsuit$

Join relation

The idea is that if $s_i \ltimes s'_j$, and $\Gamma \vdash t : s$ is a proved term in *one* language, then t can be used in place of a term t' such that $\Gamma' \vdash t' : s'$ in the *other* language: as in [MF09], "ML code can be used in place of Scheme code". This is made precise in due course.

We show in 6.9 and 6.10 that there are nice notions of categories of signatures, both order-sorted and multi-language.

Category of order-sorted signatures

Definition 6.9 (Category of Order-Sorted Signature)**.** *OSSg* is the *category of order-sorted signatures* with morphisms $h : Sg_1 \to Sg_2$ given by

(i) a monotone function $h : (S_1, \leqslant_1) \to (S_2, \leqslant_2)$, where we will write $h(w) = h(s_1) \dots h(s_n)$ for $w = s_1 \dots s_n \in S_1^n$; and

(ii) 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 $f : w \to s$ in Sg_1, then $h(f) : h(w) \to h(s)$ in Sg_2. ♢

Category of multi-language signatures

Definition 6.10 (Category of Multi-Language Signature)**.** *MLSg* is the *category of multi-language signatures* in which a morphism

$$h = (h_1, h_2) : (Sg_1, Sg_2, \ltimes) \to (Sg'_1, Sg'_2, \ltimes')$$

is defined by two morphisms $h_1 : Sg_1 \to Sg'_1$ and $h_2 : Sg_2 \to Sg'_2$ in *OSSg* such that they preserve the join relation, namely $s_i \ltimes s'_j$ in $(Sg_1, Sg_2, \ltimes)$ implies $(h_i(s)_i \ltimes' (h_j(s'))_j$ in $(Sg'_1, Sg'_2, \ltimes')$. ♢

We shall also see that we can exhibit a functor that maps a multi-language signature to an order-sorted signature (the *associated signature* 6.11), blending the two original signatures into one.

Associated signature

Definition 6.11 (Associated Signature)**.** Let $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ be a multi-language signature. The *associated signature* $\boldsymbol{Sg}^*$ of $\boldsymbol{Sg}$ is the order-sorted signature defined as follows:

(i) the poset of sorts is given by $(S_1 + S_2, \leqslant)$, where $s_i \leqslant r_j$ if $i = j$ and $s \leqslant_i r$;

(ii) if $f : w \to s$ is a function symbol in Sg_i for some $i = 1, 2$, then $f_i : w_i \to s_i$ is a function symbol in $\boldsymbol{Sg}^*$;

(iii) if $k : s$ is a constant in Sg_i for some $i = 1, 2$, then $k_i : s_i$ is a constant in $\boldsymbol{Sg}^*$; and

Conversion operator

(iv) a *conversion operator* $\hookrightarrow_{s_i, s'_j} : s_i \to s'_j$ for each constraint $s_i \ltimes s'_j$. ♢

Associated signature functor

The *associated signature functor* $(-)^* : \mathit{MLSg} \to \mathit{OSSg}$ maps each multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ to its associated signature $\boldsymbol{Sg}^*$, and each multi-language signature morphism $h : \boldsymbol{Sg} \to \boldsymbol{Sg}'$ to the order-sorted signature morphism $h^* : \boldsymbol{Sg}^* \to \boldsymbol{Sg}'^*$ given by $h^*(s_i) = (h_i(s))_i$ for each $s \in S_i$ (hence $s_i \in S_1 + S_2$) and $h^*(f_i) = (h_i(f))_i$ for each $f \in Sg_i$ (hence f_i in $\boldsymbol{Sg}$).

Running Example 6.2. $(-)^*$ embeds the multi-language signature $\boldsymbol{Sg}$ into *OSSg*, providing the order-sorted version $\boldsymbol{Sg}^*$ of the multi-language. $\boldsymbol{Sg}^*$ generates Sg_i-terms (see Section 6.2.2) as well as hybrid multi-language terms involving conversion operators such as

$$[c : \mathit{chr}_2, n : \mathit{nat}_1] \vdash +_1(\hookrightarrow_{\mathit{chr}_2, \mathit{nat}_1}(c), n) : \mathit{nat}_1$$

From now on, we use colours in the examples for disambiguating the left and the right inclusion in place of subscripts $_1$ and $_2$ (as we did in previous chapters). Moreover, we use an infix notation whenever the operators lend themselves well to do so. That is, the previous term is represented by

$$[\mathtt{c}\colon \mathit{chr}, \mathtt{n}\colon \mathit{nat}] \vdash +(\hookrightarrow_{\mathit{chr},\mathit{nat}}(\mathtt{c}), \mathtt{n})\colon \mathit{nat} \qquad \diamondsuit$$

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 6.12 (Multi-Language Algebra)**.** Let $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ be a multi-language signature. An ***Sg*-algebra** $\mathscr{A}$ in an FPI-category (C, I) is given by

Multi-language algebra

(i) a pair of order-sorted algebras $\mathscr{A}_1$ and $\mathscr{A}_2$ in (C, I) over Sg_1 and Sg_2, respectively; and

(ii) a *boundary morphism* $[\![\mathtt{s}_i \ltimes \mathtt{s}'_j]\!]_{\mathscr{A}}\colon [\![\mathtt{s}]\!]_{\mathscr{A}_i} \to [\![\mathtt{s}']\!]_{\mathscr{A}_j}$ in C for each constraint $\mathtt{s}_i \ltimes \mathtt{s}'_j$. $\diamondsuit$

Boundary morphism

An algebra sets out the meaning of a multi-language: The meaning of the underlying languages, and *how* terms of sort $\mathtt{s} \in \mathtt{S}_i$ can be interpreted as terms of sort $\mathtt{s}' \in \mathtt{S}_j$. Put differently, "boundary morphisms regulate the flow of values across $\mathscr{A}_1$ and $\mathscr{A}_2$" [MF09].

Definition 6.13 (Multi-Language Homomorphism)**.** Let $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ be a multi-language signature, and let $\mathscr{A}$ and $\mathscr{B}$ be two $\boldsymbol{Sg}$-algebras. An ***Sg*-homomorphism** $\mathtt{h}\colon \mathscr{A} \to \mathscr{B}$ is given by a pair of order-sorted homomorphisms $\mathtt{h}_1\colon \mathscr{A}_1 \to \mathscr{B}_1$ and $\mathtt{h}_2\colon \mathscr{A}_2 \to \mathscr{B}_2$ such that they commute with boundary functions, namely, if $\mathtt{s}_i \ltimes \mathtt{s}'_j$, then the following diagram commutes:

Multi-language homomorphism

$$\begin{array}{ccc} [\![\mathtt{s}]\!]_{\mathscr{A}_i} & \xrightarrow{[\![\mathtt{s}_i \ltimes \mathtt{s}'_j]\!]_{\mathscr{A}}} & [\![\mathtt{s}']\!]_{\mathscr{A}_j} \\ \downarrow{\scriptstyle (\mathtt{h}_i)_{\mathtt{s}}} & & \downarrow{\scriptstyle (\mathtt{h}_j)_{\mathtt{s}'}} \\ [\![\mathtt{s}]\!]_{\mathscr{B}_i} & \xrightarrow[[\![\mathtt{s}_i \ltimes \mathtt{s}'_j]\!]_{\mathscr{B}}]{} & [\![\mathtt{s}']\!]_{\mathscr{B}_j} \end{array}$$

$\diamondsuit$

Given a multi-language signature $\boldsymbol{Sg}$, the class of all $\boldsymbol{Sg}$-algebras and homomorphisms form a category denoted by $Alg(C, I)_{\boldsymbol{Sg}}$. We have a simple connection between this category and $Alg(C, I)_{\boldsymbol{Sg}^*}$, outlined in Theorem 6.4, after more of the running example.

Running Example 6.3. Suppose we are interested in a multi-language $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ according to the specifications of Running Example 4.2, that is

- terms denoting natural numbers can be used in place of characters according to the function $\mathrm{chr}\colon \mathbb{N} \to \mathbb{A}$ that maps the natural number $\mathtt{n}$ to the $\mathtt{n}$-th character symbol modulo $|\mathbb{A}|$; and
- terms denoting strings can be used in place of natural numbers according to the function $\mathrm{len}\colon \mathbb{A}^* \to \mathbb{N}$, namely the length of the string.

In order to get such a multi-language, we provide the join relation $\ltimes$ on $S_1 + S_2$ and a boundary morphism $[\![s_i \ltimes s'_j]\!]_{\mathscr{A}} \colon [\![s]\!]_{\mathscr{A}_i} \to [\![s']\!]_{\mathscr{A}_j}$ for each constraint $s_i \ltimes s'_j$ introduced by $\ltimes$:

- $nat_1 \ltimes chr_2$ and $nat_1 \ltimes str_2$ with boundaries

$$[\![nat_1 \ltimes chr_2]\!]_{\mathscr{A}}(n) = [\![nat_1 \ltimes str_2]\!]_{\mathscr{A}}(n) = \mathrm{chr}(n)$$

  and

- $chr_2 \ltimes nat_1$ and $str_2 \ltimes nat_1$ with boundaries

$$[\![chr_2 \ltimes nat_1]\!]_{\mathscr{A}}(c) = \mathrm{len}(c) = 1$$
$$[\![str_2 \ltimes nat_1]\!]_{\mathscr{A}}(s) = \mathrm{len}(s) \qquad \diamondsuit$$

The next theorem yields a formal correspondence between multi-languages and order-sorted languages: We can make a multi-language signature $\boldsymbol{Sg}$ 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 $\boldsymbol{Sg}$ and $\boldsymbol{Sg}^*$.

Theorem 6.4. *There is a natural isomorphism between the category of multi-language algebras over $\boldsymbol{Sg}$ and the category of order-sorted algebras over the associated signature $\boldsymbol{Sg}^*$ denoted by*

$$\eta \colon MLAlg(C, I) \Rightarrow OSAlg(C, I) \circ (-)^*$$

which induces

$$Alg(C, I)_{\boldsymbol{Sg}} \cong Alg(C, I)_{\boldsymbol{Sg}^*}$$

where there are functors $MLAlg(C, I) \colon MLSg \to Cat^{\mathrm{op}}$ and $OSAlg(C, I) \colon OSSg \to Cat^{\mathrm{op}}$ that map signatures to their category of algebras in (C, I).

Proof. The functors are defined on objects by

$$MLAlg(C, I)(\boldsymbol{Sg}) = Alg(C, I)_{\boldsymbol{Sg}}$$
$$OSAlg(C, I)(Sg) = Alg(C, I)_{Sg}$$

Now, let $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ and $\boldsymbol{Sg}' = (Sg'_1, Sg'_2, \ltimes')$, and let

$$h = (h_1, h_2) \colon \boldsymbol{Sg} \to \boldsymbol{Sg}'$$

be a signature morphism between them. We shall define a functor

$$H \colon Alg(C, I)_{\boldsymbol{Sg}'} \to Alg(C, I)_{\boldsymbol{Sg}}$$

and let $MLAlg(C, I)(h) = H$. Let $\mathscr{A}$ be an $\boldsymbol{Sg}'$-algebra in $Alg(C, I)_{\boldsymbol{Sg}'}$. Then, the multi-language $\boldsymbol{Sg}$-algebra $H\mathscr{A}$ is defined as follows:

- We define its order-sorted components $(H\mathscr{A})_1$ and $(H\mathscr{A})_2$. Let $i = 1, 2$:
  - the interpretation of sorts is given by $[\![s]\!]_{(H\mathscr{A})_i} = [\![h_i(s)]\!]_{\mathscr{A}_i}$ for each $s \in S_i$;
  - given the function symbol $f \colon w \to s$ in Sg_i, we let $[\![f \colon w \to s]\!]_{(H\mathscr{A})_i} = [\![h_i(f) \colon h_i(w) \to h_i(s)]\!]_{\mathscr{A}_i}$;

- the constant symbol $k\colon s$ in Sg_i is interpreted by letting $[\![k]\!]_{(H\mathscr{A})_i} = [\![h_i(k)]\!]_{\mathscr{A}_i}$; and
- $[\![s \leqslant_i r]\!]_{(H\mathscr{A})_i} = [\![h_i(s) \leqslant'_i h_i(r)]\!]_{\mathscr{A}_i}$ for each subsort constraint $s \leqslant_i r$ in Sg_i.

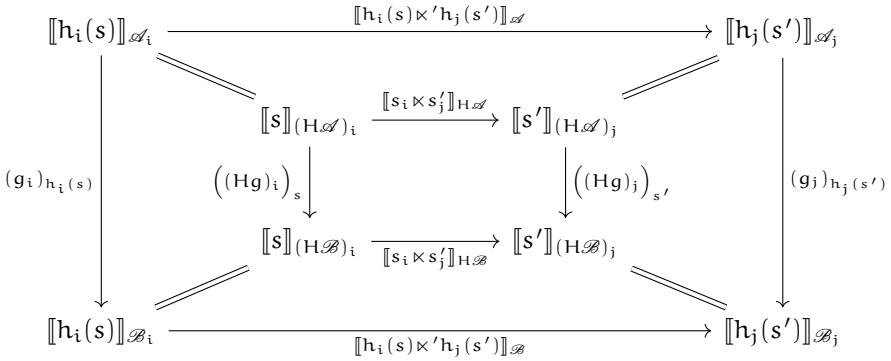
The fact that $(H\mathscr{A})_i$ is a proper order-sorted Sg_i-algebra is ensured by the properties of the (multi-language) signature morphism h.

- Boundary morphisms are defined by $[\![s_i \ltimes s'_j]\!]_{H\mathscr{A}} = [\![h_i(s) \ltimes' h_j(s')]\!]_{\mathscr{A}}$ for each constraint $s_i \ltimes s'_j$ in $\boldsymbol{Sg}$.

In order to define the action of H on homomorphisms, suppose that $g\colon \mathscr{A} \to \mathscr{B}$ is a multi-language $\boldsymbol{Sg}'$-homomorphism in $Alg(C, I)_{\boldsymbol{Sg}'}$. Then, $(Hg)_i\colon (H\mathscr{A})_i \to (H\mathscr{B})_i$ is defined by the following S_i-sorted morphisms:

- $\big((Hg)_i\big)_s = (g_i)_{h_i(s)}$, which is well-defined since $g_i\colon \mathscr{A}_i \to \mathscr{B}_i$ is an order-sorted Sg'_i-homomorphism and $[\![h_i(s)]\!]_{\mathscr{A}_i} = [\![s]\!]_{(H\mathscr{A})_i}$.

The commutativity of the diagram in Definition 6.13 is given by a tedious but simple diagram chase:



We next define a functor $H\colon Alg(C, I)_{Sg_2} \to Alg(C, I)_{Sg_1}$ and set

$$OSAlg(C, I)(h) = H$$

where $h\colon Sg_1 \to Sg_2$ is an order-sorted signature morphism. This is similar to the definition of H above. First pick any (order-sorted) Sg_2-algebra $\mathscr{A}$. We need to define the order-sorted Sg_1-algebra $H\mathscr{A}$. We define

- objects $[\![s \in S_1]\!]_{H\mathscr{A}} = [\![h(s)]\!]_{\mathscr{A}}$ in C and hence

$$[\![w]\!]_{H\mathscr{A}} = [\![h(s_1)]\!]_{\mathscr{A}} \times \cdots \times [\![h(s_n)]\!]_{\mathscr{A}}$$

for $w = s_1 \dots s_n \in S_1^n$;

- morphisms

$$[\![f\colon w \to s \in Sg_1]\!]_{H\mathscr{A}} = [\![h(f)\colon h(w) \to h(s) \in Sg_2]\!]_{\mathscr{A}}\colon [\![w]\!]_{H\mathscr{A}} \to [\![s]\!]_{H\mathscr{A}}$$

and morphisms $[\![k \in Sg_1]\!]_{H\mathscr{A}} = [\![h(k)]\!]_{\mathscr{A}}\colon 1 \to [\![s]\!]_{H\mathscr{A}}$; and

- a morphism $[\![s \leqslant_1 r \in S_1]\!]_{H\mathscr{A}} = [\![h(s) \leqslant_2 h(r)]\!]_{\mathscr{A}}\colon [\![s]\!]_{H\mathscr{A}} \rightarrowtail [\![r]\!]_{H\mathscr{A}}$ in I.

We omit the verification that semantics of operators commutes with the semantics of subsorting, although this is essentially immediate since $\mathscr{A}$ is an Sg_2-algebra. Now let $g\colon \mathscr{A} \to \mathscr{B}$ be an order-sorted Sg_2-homomorphism. We define the Sg_1-homomorphism $H(g)\colon H\mathscr{A} \to H\mathscr{B}$ by setting the components to be

$$(H(g))_{s\in S_1} = g_{h(s)}\colon [\![h(s)]\!]_{\mathscr{A}} \to [\![h(s)]\!]_{\mathscr{B}}$$

Now we define the natural transformation η by specifying the components $\eta_{\boldsymbol{Sg}}\colon Alg(C,I)_{\boldsymbol{Sg}} \to Alg(C,I)_{\boldsymbol{Sg}^*}$. Pick any $\boldsymbol{Sg}$-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$. First we define the (order-sorted) $\boldsymbol{Sg}^*$-algebra $\eta_{\boldsymbol{Sg}}\mathscr{A}$ by setting

- $[\![s_i \in S_1 + S_2]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![s]\!]_{\mathscr{A}_i}$ in C and $[\![w]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![s_1]\!]_{\mathscr{A}_i} \times \cdots \times [\![s_n]\!]_{\mathscr{A}_i}$ for each $w = s_1 \ldots s_n \in (S_1 + S_2)^n$;
- morphisms $[\![f_i\colon w_i \to s_i]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![f\colon w \to s]\!]_{\mathscr{A}_i}$ and $[\![k_i]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![k]\!]_{\mathscr{A}_i}\colon 1 \to [\![s]\!]_{\mathscr{A}_i}$; and
- $[\![\hookrightarrow_{s_i,s'_j}\colon s_i \to s'_j]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![s_i \ltimes s'_j]\!]_{\mathscr{A}}\colon [\![s]\!]_{\mathscr{A}_i} \to [\![s']\!]_{\mathscr{A}_j}$
- $[\![s_i \leqslant r_j]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![s \leqslant_i r]\!]_{\mathscr{A}_i}\colon [\![s]\!]_{\mathscr{A}_i} \rightarrowtail [\![r]\!]_{\mathscr{A}_i}$ in I for each $s_i \leqslant r_j$ in $S_1 + S_2$ (recall that $i = j$).

where the required commutation properties follow immediately since the $\mathscr{A}_i$ are Sg_i-algebras. Second we define $\boldsymbol{Sg}^*$-homomorphism $\eta_{\boldsymbol{Sg}}(h)\colon \eta_{\boldsymbol{Sg}}\mathscr{A} \to \eta_{\boldsymbol{Sg}}\mathscr{B}$. Since by the definition of h there are Sg_i-homomorphisms $h_i\colon \mathscr{A}_i \to \mathscr{B}_i$, we can define

$$\eta_{\boldsymbol{Sg}}(h)_{s_i\in S_1+S_2} = (h_i)_s\colon [\![s]\!]_{\mathscr{A}_i} \to [\![s]\!]_{\mathscr{B}_i}$$

and $\eta_{\boldsymbol{Sg}}(h)_{s_i}$ inherits the required commuting properties from h (commuting with conversion operators $\hookrightarrow_{s_i,s'_j}$), and from the h_i (commuting with the function symbols f_i).

One can see that $\eta_{\boldsymbol{Sg}}$ is an isomorphism by reversing the construction and by noting there is a one-one correspondence between the sorts, operators, and subsort constraints in $\boldsymbol{Sg}$ and those in Sg_1 and Sg_2. This is straightforward. More difficult is naturality of η which we show next.

Let

$$h = (h_1, h_2)\colon \boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes) \to \boldsymbol{Sg}' = (Sg'_1, Sg'_2, \ltimes')$$

Then we need to show that the diagram below commutes.

$$\begin{array}{ccc} Alg(C,I)_{\boldsymbol{Sg}} & \xrightarrow{\eta_{\boldsymbol{Sg}}} & Alg(C,I)_{\boldsymbol{Sg}^*} \\ {\scriptstyle H}\uparrow & & \uparrow{\scriptstyle H^*} \\ Alg(C,I)_{\boldsymbol{Sg}'} & \xrightarrow{\eta_{\boldsymbol{Sg}'}} & Alg(C,I)_{\boldsymbol{Sg}'^*} \end{array}$$

where of course $H^* = MLAlg(C,I)(h^*)$. Pick any morphism $g\colon \mathscr{A} \to \mathscr{B}$ in $Alg(C,I)_{\boldsymbol{Sg}'}$. First we need to show that $\eta_{\boldsymbol{Sg}}(H\mathscr{A}) = H^*(\eta_{\boldsymbol{Sg}'}\mathscr{A})$. Let us check only that these $\boldsymbol{Sg}^*$-algebras provide equal meaning to sorts $s_i \in S_1 + S_2$ where we have $(h_i(s))_i \in S'_1 + S'_2$:

$$\begin{aligned} [\![s_i]\!]_{\eta_{\boldsymbol{Sg}}(H\mathscr{A})} &= [\![s \in S_i]\!]_{(H\mathscr{A})_i} \\ &= [\![h_i(s) \in S'_i]\!]_{\mathscr{A}_i} \\ &= [\![(h_i(s))_i \in S'_1 + S'_2]\!]_{\eta_{\boldsymbol{Sg}'}\mathscr{A}} \\ &= [\![h^*(s_i)]\!]_{\eta_{\boldsymbol{Sg}'}\mathscr{A}} = [\![s_i]\!]_{H^*(\eta_{\boldsymbol{Sg}'}\mathscr{A})} \end{aligned}$$

Now g furnishes us with $\boldsymbol{Sg}'$-homomorphisms $g_i : \mathscr{A}_i \to \mathscr{B}_i$, and we need to show that

$$\eta_{\boldsymbol{Sg}}(Hg) = H^*(\eta_{\boldsymbol{Sg}'}g) : [\![h_i(s) \in S'_i]\!]_{\mathscr{A}_i} \to [\![h_i(s) \in S'_i]\!]_{\mathscr{B}_i}$$

is an equality of $\boldsymbol{Sg}^*$-homomorphisms. But this follows from the following calculation on components of $S_1 + S_2$-sorted morphisms

$$\begin{aligned}(\eta_{\boldsymbol{Sg}}(Hg))_{s_i} &= ((Hg)_i)_s \\ &= (g_i)_{h_i(s)} \\ &= (\eta_{\boldsymbol{Sg}'}g)_{(h_i(s))_i} \\ &= (\eta_{\boldsymbol{Sg}'}g)_{(h^*(s_i))} \\ &= (H^*(\eta_{\boldsymbol{Sg}'}g))_{s_i}\end{aligned}$$

and the proof is completed. □

Running Example 6.4. The multi-language semantics of the term introduced in Running Example 6.2 is given by the algebra $\eta_{\boldsymbol{Sg}}\mathscr{A}$ which leads to

$$[\![[c : chr, n : nat] \vdash +(\hookrightarrow_{chr,nat}(c), n) : nat]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = (c, n) \mapsto n + 1 : \mathbb{A} \times \mathbb{N} \to \mathbb{N}$$

♢

6.2.2 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 $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ be a multi-language signature. A *(multi-language) proved term* $\Gamma \vdash t : s_i$ is a proved term over the associated signature $\boldsymbol{Sg}^*$. It follows that if $\Gamma \vdash t : s$ is a proved term over Sg_i, then $\underline{\Gamma \vdash t : s}$ is a proved term in $\boldsymbol{Sg}^*$, where $\underline{\Gamma \vdash t : s} = \underline{\Gamma} \vdash \underline{t} : \underline{s}$ and

Multi-language proved term

- $\underline{s} = s_i$ for each $s \in S_i$; and $\underline{\Gamma} = [x_1 : \underline{s_1}, \ldots, x_n : \underline{s_n}]$ for each context $\Gamma = [x_1 : s_1, \ldots, x_n : s_n]$ over Sg_i;
- $\underline{t}$ is recursively defined over the syntax of raw terms generated by Sg_i:
  - $\underline{x} = x$;
  - $\underline{k} = k_i$; and
  - $\underline{f(t_1, \ldots, t_a)} = f_i(\underline{t_1}, \ldots \underline{t_n})$

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 $\boldsymbol{Sg}$-algebra $\mathscr{A}$, the categorical *semantics of a (multi-language) term* $\Gamma \vdash t : s_i$ is the order-sorted semantics of $\Gamma \vdash t : s_i$ induced by

Categorical semantics of multi-language terms

$\eta_{\boldsymbol{Sg}}\mathscr{A}$, namely

$$[\![\Gamma \vdash t\colon s_i]\!]_{\mathscr{A}} = [\![\Gamma \vdash t\colon s_i]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} \tag{6.1}$$

As expected, a multi-language preserves the semantics of the underlying terms:

Proposition 6.1. *Let $\mathscr{A}$ be a multi-language $\boldsymbol{Sg}$-algebra over $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$. If $\Gamma \vdash t\colon s$ is a proved term over Sg_i, then*

$$[\![\underline{\Gamma \vdash t\colon s}]\!]_{\mathscr{A}} = [\![\underline{\Gamma \vdash t\colon s}]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}} = [\![\Gamma \vdash t\colon s]\!]_{\mathscr{A}_i}$$

Regularity and coherence for a multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$ are defined with respect to its associated signature. That is, $\boldsymbol{Sg}$ is *regular* (resp., *coherent*) if $\boldsymbol{Sg}^*$ is regular (resp., coherent). It is immediate that $\boldsymbol{Sg}$ is regular (resp., coherent) if and only if Sg_1 and Sg_2 are regular (resp., coherent).

(Conditional) equation

Definition 6.14 (Multi-Language Equation and Satisfaction)**.** Let $\boldsymbol{Sg}$ be a coherent multi-language signature. A (conditional) *equation* for $\boldsymbol{Sg}$ is an order-sorted (conditional) equation over $\boldsymbol{Sg}^*$. A multi-language algebra $\mathscr{A}$ *satisfies* any such (conditional) equation if the (conditional) equation is satisfied by $\eta_{\boldsymbol{Sg}}\mathscr{A}$. ♢

An immediate consequence of Proposition 6.1 is that every Sg_i-equation satisfied by $\mathscr{A}_i$ is also satisfied by the multi-language algebra $\mathscr{A}$ (in its inclusion form provided by the mapping $\underline{(-)}$).

Multi-language theory, Axioms, Theorems

Model of a theory

A *multi-language theory* $\boldsymbol{Th} = (\boldsymbol{Sg}, Ax)$ is a pair of a multi-language signature $\boldsymbol{Sg}$ and a set of (conditional) multi-language equations Ax over $\boldsymbol{Sg}$, namely the *axioms* of the theory. The *theorems* of $\boldsymbol{Th}$ are the equations $\Gamma \vdash t = t'\colon s_i$ derivable from $(\boldsymbol{Sg}^*, Ax)$. A multi-language $\boldsymbol{Sg}$-algebra that satisfies all the axioms in Ax is said a *model* of $\boldsymbol{Th}$, and $MLMod(C, I)_{\boldsymbol{Th}}$ denotes the full subcategory of models of $Alg(C, I)_{\boldsymbol{Th}}$.

We now introduce the categories of theories in order to define the *associated theory* of a multi-language theory.

Notation 6.4. From now on, when we write order-sorted theories Th_1, Th_2, Th, Th', etc., we assume they are defined as $Th_1 = (Sg_1, Ax_1)$, $Th_2 = (Sg_2, Ax_2)$, $Th = (Sg, Ax)$, $Th' = (Sg', Ax')$, etc., respectively. ♢

Running Example 6.5. 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 6.12. We can generate from Ax_1 and Ax_2 a set Ax of multi-language equations by applying $\underline{(-)}$ to each equation. For instance,

$$(eq_{1,1}) = [n\colon nat] \vdash 0 + n = n\colon nat$$

becomes

$$\underline{(eq_{1,1})} = [n\colon nat] \vdash 0 + n = n\colon nat$$

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 $\Gamma \vdash t\colon nat$ in Sg_1, into the variable $n\colon nat$. However, a multi-language substitution can substitute any t', where $\Gamma' \vdash t'\colon nat$ in $\boldsymbol{Sg}^*$, for $n\colon nat$ in the lifted equation $\underline{(eq_{1,1})}$, including, crucially, the possibility that t' is a hybrid multi-language term.

$$(eq_{1,1}) \quad [n\colon nat] \vdash 0 + n = n\colon nat$$
$$(eq_{1,2}) \quad [n\colon nat, m\colon nat] \vdash s(n) + m = s(n + m)\colon nat$$
$$(eq_{1,3}) \quad [n\colon nat, m\colon nat] \vdash n + m = m + n\colon nat$$

(a) Th_1 axioms.

$$(eq_{2,1}) \quad \vdash next(a) = b\colon chr$$
$$(eq_{2,\ldots}) \quad \vdash \ldots = \ldots\colon chr$$
$$(eq_{2,26}) \quad \vdash next(z) = a\colon chr$$

(b) Th_2 axioms.

Figure 6.12: Axioms of order-sorted theories Th_1 and Th_2.

The behaviour of boundary morphisms can be axiomatized by adding the following equations to Ax:

$$(EQ_1) \quad \vdash \hookrightarrow_{nat,chr}(0) = a\colon chr$$
$$(EQ_2) \quad \vdash \hookrightarrow_{nat,str}(0) = a\colon str$$
$$(EQ_3) \quad [c\colon chr] \vdash \hookrightarrow_{chr,nat}(c) = s(0)\colon nat$$
$$(EQ_4) \quad [c\colon chr] \vdash \hookrightarrow_{str,nat}(c) = s(0)\colon nat$$
$$(EQ_5) \quad [n\colon nat] \vdash \hookrightarrow_{nat,chr}(s(n)) = next(\hookrightarrow_{nat,chr}(n))\colon chr$$
$$(EQ_6) \quad [n\colon nat] \vdash \hookrightarrow_{nat,str}(s(n)) = next(\hookrightarrow_{nat,str}(n))\colon str$$
$$(EQ_7) \quad [w\colon str, v\colon str] \vdash \hookrightarrow_{str,nat}(w + v) = \hookrightarrow_{str,nat}(s) + \hookrightarrow_{str,nat}(v)\colon nat$$

♢

Definition 6.15 (Category of Order-Sorted Theories)**.** Let $OSTh$ be the *category of order-sorted theories* whose morphisms $h\colon Th_1 \to Th_2$ are signature morphisms $h\colon Sg_1 \to Sg_2$ in $OSSg$ that preserve theorems, that is, if $\Gamma \vdash t = t'\colon s_i$ is a theorem of Th_1 with $\Gamma = [x_1\colon s_1, \ldots, x_n\colon s_n]$, then $\Gamma' \vdash h(t) = h(t')\colon h(s_i)$ is a theorem of Th_2, where $\Gamma' = [x_1\colon h(s_1), \ldots, x_n\colon h(s_n)]$ and $h(t)$ and $h(t')$ are inductively defined over the syntax according to the action of h on operators. ♢

Category of order-sorted theories

Definition 6.16 (Category of Multi-Language Theories)**.** The *category of multi-language theories* is denoted by $MLTh$ and a theory morphism $h\colon (\boldsymbol{Sg}_1, Ax_1) \to (\boldsymbol{Sg}_2, Ax_2)$ is a signature morphism $h\colon \boldsymbol{Sg}_1 \to \boldsymbol{Sg}_2$ in the category of multi-language signatures $MLSg$ such that if $\Gamma \vdash t = t'\colon s_i$ is a theorem of $(\boldsymbol{Sg}_1, Ax_1)$ with $\Gamma = [x_1\colon s_1, \ldots, x_n\colon s_n]$, then $\Gamma' \vdash h^*(t) = h^*(t')\colon h^*(s_i)$ is a theorem of $(\boldsymbol{Sg}_2, Ax_2)$, where $\Gamma' = [x_1\colon h^*(s_1), \ldots, x_n\colon h^*(s_n)]$. ♢

Category of Multi-Language Theories

Functors $MLAlg(C, I)$ and $OSAlg(C, I)$ can be easily extended to

$$MLMod(C, I)\colon MLTh \to Cat^{op}$$

and

$$OSMod(C, I)\colon OSTh \to Cat^{op}$$

respectively, such that they associate to each signature its corresponding category of models. Then, $(-)^*\colon \mathit{MLTh} \to \mathit{OSTh}$ is defined by $\boldsymbol{Th}^* = (\boldsymbol{Sg}^*, Ax)$ on objects and by h^* on morphisms $h\colon \boldsymbol{Th}_1 \to \boldsymbol{Th}_2$.

Proposition 6.2. *$\mathit{MLMod}(C, I)$ and $\mathit{OSMod}(C, I) \circ (-)^*$ are isomorphic functors. Let η be the natural isomorphism between them and $\boldsymbol{Th}$ a multi-language theory. Then, $\eta_{\boldsymbol{Th}}$ is the isomorphism between categories $\mathit{MLMod}(C, I)_{\boldsymbol{Th}}$ and $\mathit{Mod}(C, I)_{\boldsymbol{Th}}$.*

Theorem 6.5 (Soundness and Completeness)**.** *Let $\boldsymbol{Th}$ be a multi-language theory. $\Gamma \vdash t = t' : s$ is a theorem of $\boldsymbol{Th}$ if and only if $\Gamma \vdash t = t' : s$ is satisfied by every model of $\boldsymbol{Th}$.*

6.3 Further Multi-Language Constructions

Chapter 4 provides three different multi-language constructions based on boundary morphism properties (although in that chapter, morphisms are only set-theoretic functions). In Section 6.2, 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 $\hookrightarrow_{s_i, s'_j}\colon s_i \to s'_j$ in the associated signature to be replaced by subsort polymorphic operators $\hookrightarrow\colon s_i \to s'_j$ that *do not carry* any sort information. One can check that any associated signature $\boldsymbol{Sg}^*$ defined in this way remains an order-sorted signature if and only if the following additional constraint holds for $\boldsymbol{Sg}$:

$$s_i \ltimes s'_j,\ r_i \ltimes r'_j, \text{ and } s_i \leqslant_i r_i \text{ imply } s'_j \leqslant_j r'_j \tag{6.2}$$

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

$$\begin{aligned} &s_i \ltimes s'_j,\ r_i \ltimes r'_j, \text{ and } s_i \leqslant_i r_i \text{ imply} \\ &\qquad [\![s' \leqslant_j r']\!]_{\mathscr{A}_j} \circ [\![s_i \ltimes s'_j]\!]_{\mathscr{A}} = [\![r_i \ltimes r'_j]\!]_{\mathscr{A}} \circ [\![s' \leqslant_j r']\!]_{\mathscr{A}_j} \end{aligned} \tag{6.3}$$

In this new multi-language construction, we can prove the following version of Theorem 6.4:

Theorem 6.6. *Assume (6.2) and (6.3) for multi-language signatures and algebras, respectively. There is a natural isomorphism $\eta\colon \mathit{MLAlg}(C, I) \Rightarrow \mathit{OSAlg}(C, I) \circ (-)^*$ inducing $\mathit{Alg}(C, I)_{\boldsymbol{Sg}} \cong \mathit{Alg}(C, I)_{\boldsymbol{Sg}^*}$, where there are functors $\mathit{MLAlg}(C, I)\colon \mathit{MLSg} \to \mathit{Cat}^{\mathrm{op}}$ and $\mathit{OSAlg}(C, I)\colon \mathit{OSSg} \to \mathit{Cat}^{\mathrm{op}}$ that map signatures to their category of algebras in (C, I).*

Proof. The proof is almost identical to the proof of Theorem 6.4. That each $\eta_{\boldsymbol{Sg}\mathscr{A}}$ is a proper order-sorted algebra boils down to the fact that each $[\![\hookrightarrow\colon s_i \to s'_j]\!]_{\eta_{\boldsymbol{Sg}}\mathscr{A}}$ commutes with the desired morphisms in I; but this commutativity follows immediately from (6.3). □

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

- $(S_1 + S_2, \ltimes)$ is a poset; and
- $f\colon w \to s$ in Sg_i and $f\colon w' \to s'$ in Sg_j with $w_i \ltimes w'_j$, then $s_i \ltimes s'_j$.

and the associated signature $\boldsymbol{Sg}^*$ is defined as follows:

- the poset of sorts is given by $(S_1 + S_2, \leqslant)$, where $s_i \leqslant r_j$ if $i = j$ and $s \leqslant_i r$ or $i \neq j$ and $s_i \ltimes r_j$;
- if $f\colon w \to s$ is a function symbol in Sg_i, then $f\colon w_i \to s_i$ is a function symbol in $\boldsymbol{Sg}^*$, and similarly for constants.

Multi-language algebras now force boundary morphisms to act as subsort morphisms. This means that if the function symbol f appears with more than one rank $f\colon w \to s$ and $f\colon w' \to r$ in Sg_i and Sg_j, respectively, with $(s_1)_i, \ldots, (s_n)_i = w \ltimes w' = (r_1)_j, \ldots, (r_n)_j$, then the following diagram commutes:

$$\begin{array}{ccc} [\![s_1]\!]_{\mathscr{A}_i} \times \cdots \times [\![s_n]\!]_{\mathscr{A}_i} & \xrightarrow{[\![f\colon w_1 \to s]\!]_{\mathscr{A}_i}} & [\![s]\!]_{\mathscr{A}_i} \\ \downarrow{\scriptstyle [\![(s_1)_i \ltimes (r_1)_j]\!]_{\mathscr{A}} \times \cdots \times [\![(s_n)_i \ltimes (r_n)_j]\!]_{\mathscr{A}}} & & \downarrow{\scriptstyle [\![s_i \ltimes r_j]\!]_{\mathscr{A}}} \\ [\![r_1]\!]_{\mathscr{A}_j} \times \cdots \times [\![r_n]\!]_{\mathscr{A}_j} & \xrightarrow[[\![f\colon w_2 \to r]\!]_{\mathscr{A}_j}]{} & [\![r]\!]_{\mathscr{A}_j} \end{array}$$

Theorem 6.7. *Assume these new hypotheses for multi-language signatures and algebras, respectively. There is a natural isomorphism* $\eta\colon MLAlg(C, I) \Rightarrow OSAlg(C, I) \circ (-)^*$ *inducing* $Alg(C, I)_{\boldsymbol{Sg}} \cong Alg(C, I)_{\boldsymbol{Sg}^*}$*, where there are functors* $MLAlg(C, I)\colon MLSg \to Cat^{\mathrm{op}}$ *and* $OSAlg(C, I)\colon OSSg \to Cat^{\mathrm{op}}$.

6.4 The Lambda-Imp Multi-Language

This section is dedicated to showing the construction of a 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 straightforward 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 interoperability between the functional and imperative paradigms. Although the example we give is very small, it shows how the resulting multi-language can enrich both paradigms. More complex examples can be built along the lines of the one presented here. However, the presentation of elaborated theories requires a non-trivial effort in terms of length and mathematical details. Hence the choice of providing an example over two simple but fundamental theories.

For instance, suppose the imperative language does not come equipped with a conditional (C-style) operator

$$\texttt{predicate ? exp}_1\texttt{ : exp}_2$$

that returns the value denoted by $\texttt{exp}_1$ if the `predicate` holds and $\texttt{exp}_2$ otherwise. In such a case, a program like `y = x ? y + 1 : y - 1` must be encoded through an `if` statement. However, for each type τ, we can provide a lambda term $\texttt{cond(b, e}_1\texttt{, e}_2\texttt{)}$ that acts as a conditional operator, where `b` is interpreted as a boolean, and $\texttt{e}_1$ and $\texttt{e}_2$

have the same sort τ. Therefore, we might wish to build a theory in which lambda expressions can be used as expressions in the imperative language, and vice versa. In the example below, blue indicates imperative commands/expressions. The red expression is a lambda expression (containing further imperative expressions, such as `x > 0`), but playing the role of the right hand side of an imperative assignment command.

```
res := cond(x > 0, true, false)
```

Summarising, we will now show how we can

- provide a categorical semantics to such multi-language programs; and
- reason about them using multi-language equations.

The reader may wish to skim Sections 6.4.1 and 6.4.2 which are standard formalisations of the underlying languages to get a feel for our notation, and then read in more detail Section 6.4.3 for the multi-language construction itself.

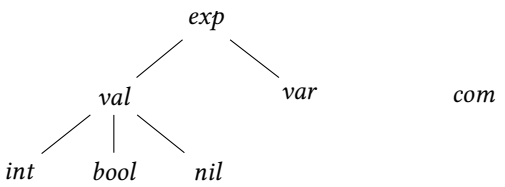
6.4.1 The Simple Imperative Language

The syntax of the simple imperative programming language Imp is given by the following order-sorted signature $Sg(\mathsf{Imp})$:

Integers	$\mathtt{i} : \mathit{int}$	$\forall \mathtt{i} \in \{\ldots, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \ldots\}$
Booleans	$\mathtt{true} : \mathit{bool}$	
	$\mathtt{false} : \mathit{bool}$	
Null value	$\mathtt{nil} : \mathit{nil}$	
Binary operators	$\otimes : \mathit{exp}, \mathit{exp} \to \mathit{exp}$	$\forall \otimes \in \mathit{BinOp}$
Unary operators	$\diamond : \mathit{exp} \to \mathit{exp}$	$\forall \diamond \in \mathit{UnOp}$
Variables	$\mathtt{x} : \mathit{var}$	$\forall \mathtt{x} \in \mathit{Var}$
Commands	$\mathtt{skip} : \mathit{com}$	
	$\mathtt{if} : \mathit{exp}, \mathit{com}, \mathit{com} \to \mathit{com}$	
	$\mathtt{comp} : \mathit{com}, \mathit{com} \to \mathit{com}$	
	$\mathtt{assign} : \mathit{var}, \mathit{exp} \to \mathit{com}$	
	$\mathtt{while} : \mathit{exp}, \mathit{com} \to \mathit{com}$	

where $\mathit{BinOp} = \{\mathtt{+}, \mathtt{*}, \mathtt{<=}, \mathtt{==}, \mathtt{and}, \ldots\}$ is a set of binary operators that are commonly found in imperative languages, and $\mathit{UnOp} = \{\mathtt{not}, \mathtt{-}\}$ contains the logical negation and unary minus operators. Finally, $\mathit{Var} = \{\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots\}$ is a countable set of variables.

The collection of sorts in $Sg(\mathsf{Imp})$ is specified by the following poset (depicted as a Hasse diagram):



Note that the signature $Sg(\mathsf{Imp})$ is not finite due to specifications of integers and variables. However, it is easy to recover finiteness by providing an inductive definition of these two sorts (see, for instance, [VM06]).

We provide semantics to $Sg(\mathsf{Imp})$ in the FPI-category $(\mathit{Dcpo}, \mathit{Incl})$, whose Dcpo objects are directed-complete partial orders (dcpo) and morphisms are Scott-continuous functions, and inclusion structure Incl is inclusions on dcpos (sub-dcpos).

- For each sort of $Sg(\mathsf{Imp})$ we assign a dcpo according to the following definitions:

$$\begin{array}{ll} [\![\mathit{int}]\!] = \mathbb{Z} & [\![\mathit{val}]\!] = \mathit{Val}_\perp = \mathit{Val} \cup \{\perp\} \\ [\![\mathit{bool}]\!] = \mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\} & \quad \text{where } \mathit{Val} = \mathbb{Z} \cup \mathbb{B} \\ [\![\mathit{nil}]\!] = \{\perp\} & [\![\mathit{exp}]\!] = \mathit{Env} \to \mathit{Val}_\perp \\ [\![\mathit{var}]\!] = \mathit{Var} & [\![\mathit{com}]\!] = (\mathit{Env}_\perp \xrightarrow{\sqcup} \mathit{Env}_\perp, \sqsubseteq) \end{array}$$

  where $\mathit{Env} = \mathit{Var} \to \mathit{Val}_\perp$ and $\mathit{Env}_\perp = \mathit{Env} \cup \{\perp\}$ (we assume that $\mathit{Env}_\perp$ is a flat poset in which $\perp \sqsubseteq \rho$ for each $\rho \in \mathit{Env}_\perp$). All these dcpos have the discrete order except for $[\![\mathit{com}]\!]$ which is ordered by the *information order* on continuous functions between stores, namely

$$f \sqsubseteq g \quad \text{if} \quad \forall \rho \in \mathit{Env}_\perp \,.\, f(\rho) \sqsubseteq g(\rho)$$

  The least upper bound $\bigsqcup$ of a directed subset $S \subseteq [\![\mathit{com}]\!]$ is defined by

$$(\sqcup S)\rho = \sqcup\{\, f(\rho) \mid f \in S \,\}$$

  Since S is directed, $\bigsqcup S$ is guaranteed to exist.

- For each subsort constraint $s \leqslant r$ in $Sg(\mathsf{Imp})$, we define a monomorphism $[\![s \leqslant r]\!] : [\![s]\!] \to [\![r]\!]$ in Incl:

  - if $r = \mathit{val}$ and $s \in \{\mathit{int}, \mathit{bool}, \mathit{nil}\}$, we define $[\![s \leqslant r]\!]$ to be the inclusion function;
  - $[\![\mathit{val} \leqslant \mathit{exp}]\!] = v \in \mathit{Val}_\perp \mapsto (\rho \in \mathit{Env} \mapsto v)$;
  - $[\![\mathit{var} \leqslant \mathit{exp}]\!] = x \in \mathit{Var} \mapsto (\rho \in \mathit{Env} \mapsto \rho(x))$; and
  - the remaining monomorphisms are generated by composition or they are the identity function on their respective domains.

  Since Dcpo is a *concrete category*, proving that $[\![s \leqslant r]\!]$ is monic amounts to show it is an injective function in the category of sets (the (faithful) forgetful functor reflects monomorphisms).

- For each operator $f : w \to s$ in $Sg(\mathsf{Imp})$, we choose a morphism $[\![f]\!] : [\![w]\!] \to [\![s]\!]$ in Dcpo. We just provide semantics of commands, and in the following definitions we assume $\rho \neq \perp$; in the other case, we define $[\![f]\!](x \in [\![w]\!]) =$

$\bot \mapsto \bot$ for all command operators:

$$\begin{aligned}
[\![\texttt{comp}]\!](c_1, c_2) &= \rho \mapsto c_2(c_1(\rho)) \\
[\![\texttt{skip}]\!] &= \rho \mapsto \rho \\
[\![\texttt{assign}]\!](x, e) &= \rho \mapsto \rho[x \leftarrow e(\rho)] \\
[\![\texttt{if}]\!](e, c_1, c_2) &= \rho \mapsto \begin{cases} c_1(\rho) & \text{if } e(\rho) = \mathsf{tt} \\ c_2(\rho) & \text{if } e(\rho) = \mathsf{ff} \\ \bot & \text{otherwise} \end{cases} \\
[\![\texttt{while}]\!](e, c) &= \mathrm{lfp}\, \mathsf{F}_{e,c} = \bigsqcup_{n \in \mathbb{N}} \mathsf{F}^n_{e,c}(\rho \mapsto \bot)
\end{aligned}$$

where $\mathsf{F}_{e,c} \colon (Env_\bot \to Env_\bot) \xrightarrow{\sqcup} (Env_\bot \to Env_\bot)$ is defined as

$$\mathsf{F}_{e,c}(f)(\rho) = \begin{cases} \rho & \text{if } e(\rho) = \mathsf{ff} \\ f(c(\rho)) & \text{if } e(\rho) = \mathsf{tt} \\ \bot & \text{otherwise} \end{cases}$$

In order to provide an equational axiomatization of Imp, we need extra (syntactical) structures in the signature. For instance, there is no way to express "expressions `if E then C else C'` and `C` are provably equal if the expression `E` evaluates to `true` in a given store `S`", since $Sg(\mathsf{Imp})$ does not yet have the *syntactical* notion of store. Therefore, we extend $Sg(\mathsf{Imp})$ by adding the following sorts and operators:

Store operations	`[]`: *store*
	`get`: *store*, *var* $\to$ *val*
	`put`: *store*, *var*, *val* $\to$ *store*
Configurations	`<>`: *store*, *exp* $\to$ *config*
	`<>`: *store*, *com* $\to$ *config*

The axiomatization of the language is given by the equations below (we give only the axiomatization of commands, see [VM06] for equations involving expressions and store operations). Moreover, note that we use an infix notation instead of a prefix one (and sometimes we change the name of operators) in order to avoid a cumbersome exposition. For instance, we write `C; C'` in place of `comp(C, C')`, or `S[X := V]` in place of `put(S, X, V)`, etc. (this is possible thanks to the algebraic equivalence between signatures and grammars, as we will show in Chapter 8):

```
<S, if E then C else C'> = <S, C>                  if  <S, E> = <S, true>
<S, if E then C else C'> = <S, C'>                 if  <S, E> = <S, false>
            <S, C; C'> = <S', C'>                  if  <S, C> = <S', skip>
           <S, X := E> = <S[X := V], skip>         if  <S, E> = <S, V>
      <S, while E do C> = <S, skip>                if  <S, E> = <S, false>
      <S, while E do C> = <S, C; while E do C>     if  <S, E> = <S, true>
```

where uppercase constants are variables whose sort is easily deducible from the signature. Note that the algebra we gave satisfies all these equations provided that we give meaning to new sorts and operators (such an extension is trivial and omitted).

6.4.2 The simply-typed lambda calculus

The syntax of the simply-typed lambda calculus is given by the following signature $Sg(\lambda^{\rightarrow})$, where *single* letters such as e rather than three such as *exp* for the imperative language, denote sorts):

Types	$\tau\colon t$	$\forall \tau \in \mathrm{T}$
Variables	$\mathtt{x}\colon x$	$\forall \mathtt{x} \in \mathit{Var}$
Abstraction	$\lambda\colon x, t, e \to e$	
Application	$\circ\colon e, e \to e$	
Integer constants	$\mathtt{i}\colon e$	$\forall \mathtt{i} \in \{\ldots, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \ldots\}$
Conditional operator	$\mathtt{cond}\colon e, e, e \to e$	

where T is inductively defined as the smallest set (1) containing `int` and (2) given $\tau, \tau' \in \mathrm{T}$, then $\tau \to \tau' \in \mathrm{T}$. We assume a flat ordering between sorts, except for $x \leqslant e$, thus enabling the use of variables as expressions. Here too, we can easily recover the finiteness property of the signature. We provide both a static and dynamic semantics for $Sg(\lambda^{\rightarrow})$ using sets. (Note that, in the following, we stick to the standard notation for lambda expressions, writing, for instance, `(λx:int.x) 1` in place of $\circ(\lambda(\mathtt{x}, \mathtt{int}, \mathtt{x}), \mathtt{1})$.)

Static Semantics

The static semantics assigns a unique type $\tau \in \mathrm{T}$ to each well-defined lambda term, and $\bot$ otherwise. We can model such a semantics as an algebra in the FPI-category $(\mathit{Set}, \mathit{Incl})$.

- For each sort in $Sg(\lambda^{\rightarrow})$ we pick a set in *Set*:

$$[\![t]\!] = \mathrm{T} \qquad [\![x]\!] = \mathit{Var} \qquad [\![e]\!] = \Delta \to \mathrm{T}_{\bot}$$

  where $\Delta = \mathit{Var} \to \mathrm{T}_{\bot}$ and $\mathrm{T}_{\bot} = \mathrm{T} \cup \{\bot\}$.

- The only non-trivial monomorphism to define is

$$[\![x \leqslant e]\!] = \mathtt{x} \in \mathit{Var} \mapsto (\delta \in \Delta \mapsto \delta(\mathtt{x}))$$

- The static semantics of operators is defined as follows:

$$[\![\mathtt{cond}]\!](e_1, e_2, e_3) = \delta \mapsto \begin{cases} \tau & \text{if } e_1(\delta) = \mathtt{nat} \text{ and } e_2(\delta) = \tau = e_3(\delta) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\lambda]\!](\mathtt{x}, \tau, e) = \delta \mapsto \big(\tau \to e(\delta[\mathtt{x} \leftarrow\!\shortmid\, \tau])\big)$$

$$[\![\tau]\!] = \tau \qquad [\![\mathtt{x}]\!] = \mathtt{x} \qquad [\![\mathtt{i}]\!] = \delta \mapsto \mathtt{int}$$

$$[\![\circ]\!](e_1, e_2) = \delta \mapsto \begin{cases} \tau' & \text{if } e_1(\delta) = \tau \to \tau' \text{ and } e_2(\delta) = \tau \\ \bot & \text{otherwise} \end{cases}$$

  Note that for a neater semantics of $[\![\mathtt{i}]\!]$ we could have added a subsort $c \leqslant e$ for constants and define $[\![\mathtt{i}]\!] = \mathtt{int}$ and the monomorphism $[\![c \leqslant e]\!](\mathtt{int}) = \delta \mapsto \mathtt{int}$.

Dynamic Semantics

The dynamic semantics provides expressions with a meaning. It is defined by an algebra in $(\mathit{Set}, \mathit{Incl})$, and in the next section we make explicit the connection with the static semantics.

- For each sort in $Sg(\lambda^{\to})$ we pick a set in *Set*:

$$[\![t]\!] = \mathrm{T} \qquad [\![x]\!] = \mathit{Var} \qquad [\![e]\!] = \mathrm{H} \to (\!|\, \mathrm{T} \,|\!)_\perp$$

  where $\mathrm{H} = \mathit{Var} \to (\!|\, \mathrm{T} \,|\!)_\perp$, $(\!|\, \mathrm{T} \,|\!)_\perp = (\!|\, \mathrm{T} \,|\!) \cup \{\perp\}$, and

$$(\!|\, \mathrm{T} \,|\!) = \bigcup_{\tau \in \mathrm{T}} (\!|\, \tau \,|\!) \qquad \text{with} \qquad \begin{cases} (\!|\, \mathtt{int} \,|\!) = \mathbb{Z} \\ (\!|\, \tau \to \tau' \,|\!) = (\!|\, \tau \,|\!) \to (\!|\, \tau' \,|\!) \end{cases}$$

- The only non-trivial monomorphism to define is

$$[\![x \leqslant e]\!] = \mathrm{x} \in \mathit{Var} \mapsto (\eta \in \mathrm{H} \mapsto \eta(\mathrm{x}))$$

- The dynamic semantics of operators is defined as follows:

$$[\![\tau]\!] = \tau \qquad [\![\mathrm{x}]\!] = \mathrm{x} \qquad [\![\mathrm{i}]\!] = \eta \mapsto \mathrm{int}(\mathrm{i})$$

$$[\![\mathsf{cond}]\!](e_1, e_2, e_3) = \eta \mapsto \begin{cases} e_2(\eta) & \text{if } e_1(\eta) \in \mathbb{Z} \setminus \{0\} \\ e_3(\eta) & \text{if } e_1(\eta) = 0 \\ \perp & \text{otherwise} \end{cases}$$

$$[\![\lambda]\!](\mathrm{x}, \tau, e) = \eta \mapsto \begin{cases} v \in (\!|\, \tau \,|\!) \mapsto e(\eta[\mathrm{x} \leftarrow\!\shortmid v]) & \text{if } e(\eta[\mathrm{x} \leftarrow\!\shortmid v]) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

$$[\![\circ]\!](e_1, e_2) = \eta \mapsto \begin{cases} \big(e_1(\eta)\big)(e_2(\eta)) & \text{if } e_2(\eta) \in \mathrm{dom}(e_1(\eta)) \\ \perp & \text{otherwise} \end{cases}$$

  where int is the obvious bijection between $\{\ldots, \mathtt{-1}, \mathtt{0}, \mathtt{1}, \ldots\}$ and $\mathbb{Z}$. A similar argument for achieving a neater dynamic semantics for $[\![\mathrm{i}]\!]$ can be done by following the observation made in the previous section.

Basic Theorems

Let $\eta \in \mathrm{H}$ and $\delta \in \Delta$. We write $\delta \vdash \eta$ if for every $\mathrm{x} \in \mathit{Var}$ we have that $\eta(\mathrm{x}) \in (\!|\delta(\mathrm{x})|\!)$, where $(\!|\, \perp \,|\!) = \{\perp\}$. Then, for each closed lambda term $\vdash \mathrm{M} \colon e$, the following propositions hold:

Proposition 6.3. *If* $[\![\mathrm{M}]\!]\delta = \tau$ *and* $\delta \vdash \eta$, *then* $[\![\mathrm{M}]\!]\eta \in (\!|\, \tau \,|\!)$.

Proposition 6.4. *If* $[\![\mathrm{M}]\!]\eta = \perp$, *then* $[\![\mathrm{M}]\!]\delta = \perp$.

Equational Theory

The simply-typed lambda calculus can be regarded as an equational theory by axiomatizing the α, β, and η-equivalence rules. For instance, β and η-equivalence rules are given by the following equations:

$$\mathtt{(\lambda X\colon T\,.\,M)\ M'} = \mathtt{M[M'/X]}$$
$$\mathtt{\lambda X\colon T\,.\,(M\ X)} = \mathtt{M} \quad \text{if} \quad \mathtt{X} \text{ is not free in } \mathtt{M}$$

However, in order to fully formalise these rules we need the axiomatization of free variables and term substitution, which can be found in [VM06].

6.4.3 The Multi-Language Construction

In this section, we build the multi-language $\boldsymbol{Sg}(\lambda^{\rightarrow}\text{-Imp})$ that blends $Sg(\mathsf{Imp})$ and $Sg(\lambda^{\rightarrow})$ together. The first step is to specify which syntactic categories of one language can be used in the other. Since we would like to use $Sg(\lambda^{\rightarrow})$ expressions in place of $Sg(\mathsf{Imp})$ expressions and vice versa, we define the join relation by requiring $e \ltimes exp$ and $exp \ltimes e$. This allow us to write programs such as the one in the introduction, namely

$$\texttt{res := } \hookrightarrow_{e,exp}(\texttt{cond}(\hookrightarrow_{exp,e}(\texttt{x > 0}),\hookrightarrow_{exp,e}(\texttt{true}),\hookrightarrow_{exp,e}(\texttt{false})))$$

In order for these programs to have meaning, we must exhibit two boundary functions that specify how lambda expressions flow from $Sg(\lambda^{\rightarrow})$ (in which they have a meaning) to $Sg(\mathsf{Imp})$ (in which they don't), and vice versa.

- In the former case, we have to define

  $$[\![e \ltimes exp]\!] : (\mathrm{H} \to (\!|\, \mathrm{T} \,|\!)_{\perp}) \to (Env \to Val_{\perp})$$

  that is, given a lambda expression e, we need to provide (the denotation of) an expression of the imperative language representing the conversion of e. The approach we follow here is to "run" e over the conversion $\phi(\rho)$ of a store $\rho \in Env$, and to convert the result of $e(\phi(\rho)) \in (\!|\, \mathrm{T} \,|\!)_{\perp}$ to a proper value in the Imp language through a function $\psi : (\!|\, \mathrm{T} \,|\!)_{\perp} \to Val_{\perp}$. Diagrammatically, this can be depicted as

  $$\begin{array}{ccc} \mathrm{H} & \xrightarrow{\;e\;} & (\!|\, \mathrm{T} \,|\!)_{\perp} \\ {\scriptstyle\phi}\uparrow & & \downarrow{\scriptstyle\psi} \\ Env & \xrightarrow{[\![e \ltimes exp]\!}(e) & Val_{\perp} \end{array}$$

  The function $\psi(v)$ is the identity on integers (which exist as values in both languages) and the undefined value $\perp$ whenever we want to move the denotation of a function from $\lambda^{\rightarrow}$ to Imp. Indeed, in this case there are no suitable denotations for functions in the imperative language. We set:

  $$\psi(v) = \begin{cases} v & \text{if } v \in (\!|\, \mathtt{int} \,|\!) = \mathbb{Z} \\ \perp & \text{otherwise} \end{cases}$$

  The conversion of a store $\rho \in Env$ to a lambda environment follows a similar approach:

  $$\phi(\rho) = x \in Var \mapsto \begin{cases} \rho(x) & \text{if } \rho(x) \in \mathbb{Z} = (\!|\, \mathtt{int} \,|\!) \\ 1 & \text{if } \rho(x) = \mathrm{tt} \\ 0 & \text{if } \rho(x) = \mathrm{ff} \\ \perp & \text{otherwise} \end{cases}$$

Note that ϕ establishes a match between the variable's name in the imperative language and in the lambda calculus, thus sharing their values between them. Consider again the small program provided at the beginning of this section. The variable `x` of Imp is used in the lambda calculus in order to perform a conditional choice.

- In the latter case, we have to provide the conversion of an expression e of the imperative language Imp to a lambda expression $[\![exp \ltimes e]\!](e)$. The intuition is along the lines of the previous construction: we start with the conversion of a lambda environment η to a valid store $\beta(\eta)$ in Imp. Then, we let the result of $e(\beta(\eta))$ flow to $(\!|\, T \,|\!)_\perp$ through a function α:

$$\begin{array}{ccc} H & \xrightarrow{[\![exp \ltimes e]\!](e)} & (\!|\, T \,|\!)_\perp \\ {\scriptstyle\beta}\downarrow & & \uparrow{\scriptstyle\alpha} \\ Env & \xrightarrow{\quad e \quad} & Val_\perp \end{array}$$

The conversion of a value in $Val_\perp$ is defined by keeping integers and $\perp$ untouched, and by converting tt and ff to 1 and 0, respectively. Such an approach allows us to employ boolean expressions of Imp (e.g., `x > 0`) as parameters in the `cond` operator in $\lambda^{\to}$:

$$\alpha(v) = \begin{cases} v & \text{if } v \in \mathbb{Z} \\ 1 & \text{if } v = tt \\ 0 & \text{if } v = ff \\ \perp & \text{otherwise} \end{cases}$$

On the other hand, the conversion of a lambda environment to a store collapses everything that is not an integer to $\perp$, since there are no suitable values for functions:

$$\beta(\eta) = x \in Var \mapsto \begin{cases} \eta(x) & \text{if } \eta(x) \in \mathbb{Z} \\ \perp & \text{otherwise} \end{cases}$$

Finally, the equational axiomatization of the behavior of boundary functions reflects the approach outlined in the definitions of $[\![e \ltimes exp]\!]$ and $[\![exp \ltimes e]\!]$:

$$\begin{aligned} \hookrightarrow_{e,exp}(i) = i = \hookrightarrow_{exp,e}(i) \qquad & \forall i \in \{\ldots, -1, 0, 1, \ldots\} \\ \hookrightarrow_{e,exp}(x) = x = \hookrightarrow_{exp,e}(x) \qquad & \forall x \in Var \\ \hookrightarrow_{exp,e}(\texttt{true}) = 1 & \\ \hookrightarrow_{exp,e}(\texttt{false}) = 0 & \end{aligned}$$

One can now compute the multi-language semantics of

$$[\![\texttt{res := } \hookrightarrow_{e,exp}(\texttt{cond}(\hookrightarrow_{exp,e}(\texttt{x > 0}), \hookrightarrow_{exp,e}(\texttt{true}), \hookrightarrow_{exp,e}(\texttt{false}))]\!] =$$

$$= \rho \mapsto \rho \left[x \leftarrow \begin{cases} 1 & \text{if } \rho(x) > 0 \\ 0 & \text{if } \rho(x) \leqslant 0 \\ \perp & \text{otherwise} \end{cases} \right]$$

CHAPTER 7

Abstract Semantics of Multi-Language Programs

☞ Chapter reference(s): [BCM20c]

In this chapter, we design a general technique for abstracting multi-language semantics given the interoperation of the underlying languages and of their abstract semantics. We exploit abstract interpretation theory [CC77] for retaining independency from the underlying analyses, and the algebraic framework of multi-languages (see Chapter 4) for generality of the blended languages.

Abstract interpretation has allowed a disparate collection of (practical) methods and algorithms proposed along the years for static analysis to evolve into a mature discipline, founded on a robust theoretical framework. This provides a good environment for designing static analysis methods within a language, semantics, and approximation independent way [CC92]. It has broad scope and wide applicability. Our aim is to retain such broad scope, but to work with multi-language programs: Instead of fixing two programming languages and combining their respective analyses, we model abstract interpretation itself, within the algebraic framework of multi-language semantics. Such an approach allows us to lay down the first steps of a general technique for designing static analyses of multi-language programs, in a way that (1) is independent of both underlying languages and analyses and (2) preserves the design and properties of the single-language abstract semantics.

Structure We begin by giving a general, algebraic, and fixpoint construction of the *collecting semantics*, namely the reference semantics for defining and proving the correctness of approximated properties. Then, we instantiate the abstract interpretation-based semantics approximation in the algebraic framework, in order to fill the gap between the algebraic approach to program semantics and static analysis. Finally, we combine all these concepts, obtaining an algebraic framework for modelling *abstract interpretation of multi-language programs*.

7.1 Algebraic Perspective on Collecting Semantics

We give a general construction of *collecting semantics*. First we set up notation. We let Sg be a regular order-sorted signature and $\mathscr{C}$ an Sg-algebra. Theorem 3.1 guarantees the existence of a homomorphism $[\![-]\!]_{\mathscr{C}} \colon \mathscr{T}_{Sg} \to \mathscr{C}$ providing Sg-terms P, called *programs*, with a meaning $[\![\mathsf{P}]\!]_{\mathscr{C}}$ (see Equation 3.4).

Program

Remark 7.1. Signatures are of course completely general. Sg might specify a lambda-calculus with $[\![-]\!]_{\mathscr{C}}$ its denotational semantics, as in Example 3.2, or Sg might specify the syntax of an imperative language with $[\![-]\!]_{\mathscr{C}}$ its small-step operational semantics (as in the following example). ♢

Example 7.1. We illustrate a simple imperative language Imp on which we define various kinds of semantics in order to show the generality of the algebraic framework. Let *Var* be a set of variables and *Val* a set of scalar values with metavariables $\mathtt{x}$ and v, respectively. Variables and values occur in the language as terminal symbols, and for each production defining the syntax of the language (on the right), we introduce a corresponding algebraic operator (on the left), or a family of operators when they are parametric on a subscript:

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

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

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

In the examples in the following sections, we often use the correspondence between algebraic and context-free terms. For instance, we may write the algebraic term

$$cond(bop_{>}(\mathtt{x}, \mathtt{0}), skip, assign_{\mathtt{x}}(bop_{-}(\mathtt{0}, \mathtt{x})))$$

in the less cumbersome context-free form

```
if x > 0 then skip else x = 0 - x
```

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

Expressions. ▷ We treat expressions E as "atomic" terms that are fully evaluable into a scalar value in a single-step. Let

$$\mathsf{S}_{exp} = \{ \langle \mathsf{E}, \rho \rangle \mid \mathsf{E} \in [\![exp]\!]_{\mathscr{T}_{\mathsf{Imp}}} \wedge \rho \in Env \}$$

$$[\![exp]\!]_{\mathscr{S}} = \wp(S_{exp} \times Val)$$

$$\begin{array}{rcl} [\![\mathtt{v}]\!]_{\mathscr{S}} & = & \{\, \langle \mathtt{v}, \rho\rangle \rightarrowtriangle \mathtt{v} \mid \rho \in Env \,\} \\ [\![\mathtt{x}]\!]_{\mathscr{S}} & = & \{\, \langle \mathtt{x}, \rho\rangle \rightarrowtriangle \rho(\mathtt{x}) \mid \rho \in Env \,\} \\ [\![bop_{\odot}]\!]_{\mathscr{S}}([\![\mathtt{E}_1]\!]_{\mathscr{S}}, [\![\mathtt{E}_2]\!]_{\mathscr{S}}) & = & \{\, \langle bop_{\odot}(\mathtt{E}_1, \mathtt{E}_2), \rho\rangle \rightarrowtriangle [\![\mathtt{E}_1]\!]^{\rho}_{\mathscr{S}} \odot [\![\mathtt{E}_2]\!]^{\rho}_{\mathscr{S}} \mid \rho \in Env \,\} \end{array}$$

Figure 7.1: Small-step operational semantics of Imp expressions.

be the set of configurations where $\mathtt{E}$ is an expression and ρ an environment in $Env = Var \to Val$. The small-step semantics of expressions is given in Figure 7.1. Intuitively, starting from an expression $\mathtt{E}$, we build a set of pairs in $\wp(S_{exp} \times Val)$ representing the one-step evaluation of $\mathtt{E}$ in each environment ρ. More precisely, $\langle \mathtt{E}, \rho\rangle \rightarrowtriangle \mathtt{v} \in [\![\mathtt{E}]\!]_{\mathscr{S}}$ simply means that $\mathtt{E}$ is evaluated into $\mathtt{v}$ in ρ. We write $[\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}}$ for denoting such $\mathtt{v}$ (unique by construction).

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

There are some missing cases in the definition of the interpretation functions for the operators in Figure 7.1. For instance, we have defined $[\![bop_{\odot}]\!]_{\mathscr{S}}$ on arguments $[\![\mathtt{E}_1]\!]_{\mathscr{S}}$ and $[\![\mathtt{E}_2]\!]_{\mathscr{S}}$. However, there are semantic elements in $\wp(S_{exp} \times Val)$ that are not the image of any expressions $\mathtt{E}$ (e.g., the empty set $\varnothing$). We shall leave implicit that $[\![bop_{\odot}]\!]_{\mathscr{S}}(e_1, e_2) = \varnothing$ whenever there are no $\mathtt{E}_1$ or $\mathtt{E}_2$ such that $e_1 = [\![\mathtt{E}_1]\!]_{\mathscr{S}}$ and $e_2 = [\![\mathtt{E}_2]\!]_{\mathscr{S}}$. ◁

Commands. ▷ Let

$$S_{com} = \{\, \langle \mathtt{C}, \rho\rangle \mid \mathtt{C} \in [\![com]\!]_{\mathscr{T}_{\mathsf{Imp}}} \cup \{\bot\} \wedge \rho \in Env \,\}$$

where $\mathtt{C}$ is a command (or $\bot$, denoting the end of a computation) and ρ an environment. For each command operator of Imp we define its semantics by specifying exactly the pairs of configurations which are related by the action of such an operator (Figure 7.2). We write $[\![\mathtt{C}]\!]^{\rho}_{\mathscr{S}}$ for the unique $\langle \mathtt{C}', \rho'\rangle$ such that $\langle \mathtt{C}, \rho\rangle \rightarrowtriangle \langle \mathtt{C}', \rho'\rangle \in [\![\mathtt{C}]\!]_{\mathscr{S}}$ (the previous discussions on recovering an expression $\mathtt{E}$ from $[\![\mathtt{E}]\!]_{\mathscr{S}}$ and the partial definition of interpretation functions shall also apply to commands). ◁

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

```
if x > 0 then skip else x = 0 - x
```

Then,

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

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

$$\begin{array}{rcl}
[\![skip]\!]_{\mathscr{S}} & = & \{\, \langle skip, \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \mid \rho \in Env \,\} \\
[\![assign_{\mathtt{x}}]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}) & = & \{\, \langle assign_{\mathtt{x}}(\mathtt{E}), \rho\rangle \rightarrowtriangle \langle \bot, \rho[\mathtt{x} \leftarrowtail [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}}]\rangle \\
& & \quad \mid \rho \in Env \,\} \\
[\![cond]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}, [\![\mathtt{C}_1]\!]_{\mathscr{S}}, [\![\mathtt{C}_2]\!]_{\mathscr{S}}) & = & \{\, \langle cond(\mathtt{E}, \mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_1, \rho\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} \neq \mathtt{0} \,\} \\
& \cup & \{\, \langle cond(\mathtt{E}, \mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_2, \rho\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} = \mathtt{0} \,\} \\
[\![loop]\!]_{\mathscr{S}}([\![\mathtt{E}]\!]_{\mathscr{S}}, [\![\mathtt{C}]\!]_{\mathscr{S}}) & = & \{\, \langle loop(\mathtt{E}, \mathtt{C}), \rho\rangle \rightarrowtriangle \langle \bot, \rho\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} = \mathtt{0} \,\} \\
& \cup & \{\, \langle loop(\mathtt{E}, \mathtt{C}), \rho\rangle \rightarrowtriangle \langle seq(\mathtt{C}, loop(\mathtt{E}, \mathtt{C})), \rho\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{E}]\!]^{\rho}_{\mathscr{S}} \neq \mathtt{0} \,\} \\
[\![seq]\!]_{\mathscr{S}}([\![\mathtt{C}_1]\!]_{\mathscr{S}}, [\![\mathtt{C}_2]\!]_{\mathscr{S}}) & = & \{\, \langle seq(\mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle \mathtt{C}_2, \rho'\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{C}_1]\!]^{\rho}_{\mathscr{S}} = \langle \bot, \rho'\rangle \,\} \\
& \cup & \{\, \langle seq(\mathtt{C}_1, \mathtt{C}_2), \rho\rangle \rightarrowtriangle \langle seq(\mathtt{C}'_1, \mathtt{C}_2), \rho'\rangle \\
& & \quad \mid \rho \in Env \wedge [\![\mathtt{C}_1]\!]^{\rho}_{\mathscr{S}} = \langle \mathtt{C}'_1, \rho'\rangle \,\}
\end{array}$$

Figure 7.2: Small-step operational semantics of Imp commands.

where the semantics of the condition is

$$\begin{array}{rcl}
[\![\mathtt{x > 0}]\!] = [\![bop_{>}]\!]([\![\mathtt{x}]\!], [\![\mathtt{0}]\!]) & = & \{\, \langle \mathtt{x > 0}, \rho\rangle \rightarrowtriangle 1 \mid \rho \in Env \wedge (\rho(\mathtt{x}) \mathtt{ > 0}) = 1 \,\} \\
& \cup & \{\, \langle \mathtt{x > 0}, \rho\rangle \rightarrowtriangle 0 \mid \rho \in Env \wedge (\rho(\mathtt{x}) \mathtt{ > 0}) = 0 \,\}
\end{array}$$

and therefore,

$$\begin{array}{l}
[\![cond]\!] \quad ([\![\mathtt{x > 0}]\!], [\![\mathtt{skip}]\!], [\![\mathtt{x = 0 - x}]\!]) = \\
\quad = \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \mathtt{skip}, \rho\rangle \\
\qquad \mid \rho \in Env \wedge (\rho(\mathtt{x}) \mathtt{ > 0}) = 1 \,\} \\
\quad \cup \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho\rangle \rightarrowtriangle \langle \mathtt{x = 0 - x}, \rho\rangle \\
\qquad \mid \rho \in Env \wedge (\rho(\mathtt{x}) \mathtt{ > 0}) = 0 \,\}
\end{array}$$

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

A property of a set is any subset. By *semantic* properties of programs, we mean properties of the (components of) the carrier set C of an algebra $\mathscr{C}$. In this section, we provide a systematic construction of an algebra $\mathscr{C}^*$ able to compute the *strongest property of programs*, that is

Strongest program property

$$[\![\mathtt{P}]\!]_{\mathscr{C}^*} = \{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$$

for each program P (Proposition 7.2). The induced semantic function $[\![-]\!]_{\mathscr{C}^*}$ is usually called the *(standard) collecting semantics* [CGR19]. Henceforth, we distinguish the semantics $[\![-]\!]_{\mathscr{C}}$ from the collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ by referring to the former as the *standard semantics*, and we shall link them algebraically in the category of algebras $Alg(Sg)$ via Proposition 7.3. We conclude this section by providing a general fixpoint calculation of $[\![-]\!]_{\mathscr{C}^*}$ whenever $[\![-]\!]_{\mathscr{C}}$ is a fixpoint semantics (Theorem 7.2).

(Standard) collecting semantics

Standard semantics

We introduce the product operator $\times$ used in the following definition. Let A be a family of sets indexed by I. The product operator

$$\times\colon \prod_{i\in I} \wp(A_i) \to \wp\Big(\prod_{i\in I} A_i\Big)$$

defines the mapping $(X_1, \ldots, X_n) \mapsto X_1 \times \cdots \times X_n$.

Definition 7.1 (Collecting Semantics)**.** Let $\mathscr{C}$ be an *Sg*-algebra. The *collecting semantics* $\mathscr{C}^*$ over $\mathscr{C}$ is defined as follows:

(i) the carrier sets are $[\![s]\!]_{\mathscr{C}^*} = \wp[\![s]\!]_{\mathscr{C}}$ for each sort s; and

(ii) the semantics of the operators is $[\![k]\!]_{\mathscr{C}^*} = \{[\![k]\!]_{\mathscr{C}}\}$ for each constant $k\colon s$, and

$$[\![f\colon w \to s]\!]_{\mathscr{C}^*} = \wp[\![f\colon w \to s]\!]_{\mathscr{C}} \circ \times$$

for each function symbol $f\colon w \to s$, where for any function $\theta\colon A \to B$ the function $\wp(\theta)\colon \wp(A) \to \wp(B)$ computes the image of θ. ♢

Proposition 7.1. *$\mathscr{C}^*$ is a proper order-sorted Sg-algebra in the category $Alg(Sg)$.*

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

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

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

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

Proposition 7.2. *$[\![-]\!]_{\mathscr{C}^*}\colon \mathscr{T}_{Sg} \to \mathscr{C}^*$ computes the strongest program property for each program P generated from Sg, that is $[\![P]\!]_{\mathscr{C}^*} = \{[\![P]\!]_{\mathscr{C}}\}$.*

Proof. By structural induction on P. □

Remark 7.2. Other forms of collecting semantics found in literature are abstractions of the collecting semantics provided here. For instance, [AMS20] defines a collecting semantics for functional programs interpreted on $D_\perp \to D_\perp$ by taking $f\colon D_\perp \to D_\perp$ to $\wp(f)$ on the lifted domain $\wp(D_\perp) \to_{cjm} \wp(D_\perp)$ of *complete-join morphisms* (cjm). Such a collecting semantics computes all the possible results of a functional program with respect to a set of input values, and it can be obtained as an abstraction of the standard collecting semantics:

$$\begin{aligned}\alpha\big(S \in \wp(D_\perp \to D_\perp)\big) &= X \in \wp(D_\perp) \mapsto \{\, f(x) \in D_\perp \mid x \in X \wedge f \in S \,\}\\ \gamma\big(F \in \wp(D_\perp) \to_{cjm} \wp(D_\perp)\big) &= \{\, x \in D_\perp \mapsto f(x) \in D_\perp \mid f(x) \in F(\{x\}) \,\}\end{aligned}$$

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

The *(forward) reachability semantics* is widely used for invariance analyses or, in general, for discovering state properties of programs [SJ03, BG15, KO11]. For each program P, it collects the set of states that are reachable by running P from a set of initial states. It can be shown that it is an abstraction of the standard collecting semantics over, for instance, a *trace semantics* (such a construction is formalised in the following example). ♢

Example 7.2. *Prefix trace semantics* associates each program $\mathtt{P}$ with the set of all finite traces obtained by iterating an arbitrarily large number of times the small-step semantics $\mathscr{S}$ from $\langle \mathtt{P}, \rho \rangle$, for each environment ρ.

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

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

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

$$\begin{array}{rcl} X & \mapsto & \{\varepsilon\} \cup \{\, \langle \mathtt{f}(\mathtt{P}_1, \ldots, \mathtt{P}_n), \rho \rangle \mid \rho \in \mathit{Env} \,\} \\ & \cup & \{\, \tau \rightarrowtriangle \langle \mathtt{C}, \rho \rangle \rightarrowtriangle \langle \mathtt{C}', \rho' \rangle \in S^*_{com} \mid \tau \rightarrowtriangle \langle \mathtt{C}, \rho \rangle \in X \wedge \langle \mathtt{C}', \rho' \rangle = [\![\mathtt{C}]\!]^{\rho}_{\mathscr{S}} \,\} \end{array}$$

The trace semantics of the constant *skip* is trivially defined by

$$[\![\mathit{skip}]\!]_{\mathscr{P}} = \{\varepsilon\} \cup \{\, \langle \mathit{skip}, \rho \rangle \mid \rho \in \mathit{Env} \,\} \cup \{\, \langle \mathit{skip}, \rho \rangle \rightarrowtriangle \langle \bot, \rho \rangle \mid \rho \in \mathit{Env} \,\}$$

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

We show an application of the prefix trace semantics $\mathscr{P}$ applied to the same term $\mathtt{P} = \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}$ of the previous example:

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

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

$$\begin{array}{rcl} F^0_{\mathtt{P}} & = & \varnothing \\ F^1_{\mathtt{P}} & = & \{\varepsilon\} \cup \{\langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho \rangle\} \\ F^2_{\mathtt{P}} & = & \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho \rangle \rightarrowtriangle \langle \mathtt{skip}, \rho \rangle \\ & & \quad \mid \rho \in \mathit{Env} \wedge (\rho(\mathtt{x}) > 0) = 1 \,\} \\ & \cup & \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho \rangle \rightarrowtriangle \langle \mathtt{x = 0 - x}, \rho \rangle \\ & & \quad \mid \rho \in \mathit{Env} \wedge (\rho(\mathtt{x}) > 0) = 0 \,\} \\ & \cup & F^1_{\mathtt{P}} \\ F^3_{\mathtt{P}} & = & \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho \rangle \rightarrowtriangle \langle \mathtt{skip}, \rho \rangle \rightarrowtriangle \langle \bot, \rho \rangle \\ & & \quad \mid \rho \in \mathit{Env} \wedge (\rho(\mathtt{x}) > 0) = 1 \,\} \\ & \cup & \{\, \langle \mathtt{if\ x > 0\ then\ skip\ else\ x = 0 - x}, \rho \rangle \rightarrowtriangle \langle \mathtt{x = 0 - x}, \rho \rangle \rightarrowtriangle \\ & & \quad \rightarrowtriangle \langle \bot, \rho[\mathtt{x} \leftarrowtail \mathtt{-x}] \rangle \mid \rho \in \mathit{Env} \wedge (\rho(\mathtt{x}) > 0) = 0 \,\} \\ & \cup & F^1_{\mathtt{P}} \\ F^{\delta > 3}_{\mathtt{P}} & = & F^3_{\mathtt{P}} \end{array}$$

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

Standard and collecting semantics are related by the property established in Proposition 7.2. Moreover, the singleton function $\{-\}\colon C \to \wp(C)$ that maps standard semantics $[\![\mathtt{P}]\!]_{\mathscr{C}}$ of programs to their strongest property $\{[\![\mathtt{P}]\!]_{\mathscr{C}}\}$ is an *Sg*-homomorphism $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$. From the abstract interpretation perspective, it means that $\{-\}$ acts as a *complete abstraction*, hence with no loss of precision [GR97].

Proposition 7.3. *The singleton function* $\{-\}\colon C \to \wp(C)$ *defined by* $c \mapsto \{c\}$ *is an Sg-homomorphism* $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$, *and therefore* $\{-\} \circ [\![-]\!]_{\mathscr{C}} = [\![-]\!]_{\mathscr{C}^*}$.

Remark 7.3. Readers familiar with category theory may notice that $\{-\}$ is the unit of the *powerset monad* on $\mathit{Alg}(\mathit{Sg})$. However, we do not pursue this here. ♢

7.2 Fixpoint Calculation of Collecting Semantics

We begin by introducing some preliminary notions.

Preliminary Notions. We call $[\![-]\!]_{\mathscr{C}}$ a *fixpoint semantics* if $[\![\mathsf{P}]\!]_{\mathscr{C}} = \mathrm{lfp}^{\preccurlyeq}_{\bot}\, \mathsf{F}$ for some *semantic transformer* $\mathsf{F}\colon \mathsf{C} \to \mathsf{C}$ (depending on *Sg*) on a *semantic domain* $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$. More precisely, we follow [Cou02] and we assume the following:

Fixpoint semantics

- The *semantic domain* $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$ is a poset $(\mathsf{C}, \preccurlyeq)$ with a smallest element $\bot$ and a *partially* defined least upper bound (lub) operator $\curlyvee$.

Semantic domain

- The *semantic transformer* $\mathsf{F}\colon \mathsf{C} \to \mathsf{C}$ is a monotone function such that its *transfinite iterates* $\mathsf{F}^0 = \bot$, $\mathsf{F}^{\delta+1} = \mathsf{F}(\mathsf{F}^\delta)$ for successor ordinals $\delta + 1$, and $\mathsf{F}^\lambda = \curlyvee_{\delta<\lambda} \mathsf{F}^\delta$ for limit ordinals λ are well-defined.

Semantic transformer

Under these assumptions, the fixpoint semantics is exactly

$$[\![\mathsf{P}]\!]_{\mathscr{C}} = \mathrm{lfp}^{\preccurlyeq}_{\bot}\, \mathsf{F} = \mathsf{F}^{\epsilon}$$

where ϵ is the least ordinal such that $\mathsf{F}(\mathsf{F}^\epsilon) = \mathsf{F}^\epsilon$.

The goal of this section is to provide a fixpoint calculation of the collecting semantics. We assume the standard semantics $[\![\mathsf{P}]\!]_{\mathscr{C}} = \mathrm{lfp}^{\preccurlyeq}_{\bot}\, \mathsf{F}$ to be a fixpoint semantics over a generic semantic domain $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$, and we define a new *computational order* $\preccurlyeq^*$ on $\wp(\mathsf{C})$ that makes the collecting semantics $[\![\mathsf{P}]\!]_{\mathscr{C}^*}$ the least fixpoint of $\mathsf{F}^* = \wp(\mathsf{F})$ (see Theorem 7.2).

Remark 7.4. The problem of finding the right partial order $\preccurlyeq^*$ on $\wp(\mathsf{C})$ for achieving such a fixpoint definition of the collecting semantics has been recently addressed in [CGR19, Sect. 7.2], by considering a preorder on $\wp(\mathsf{C})$ that is partial *only* along the iterates. A similar approach has been previously taken in [MP17] and in the thesis of Pasqua [Pas19]. Here, we show that it can be extended to a fully-fledged partial order $\preccurlyeq^*$ over the whole set $\wp(\mathsf{C})$. ♢

Definition 7.2 (Collecting Semantics Domain)**.** Let $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$ be a (non-trivial) semantic domain. The *collecting semantics domain* $\langle \wp(\mathsf{C}), \preccurlyeq^*, \bot^*, \curlyvee^* \rangle$ with respect to $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$ is defined as follows:

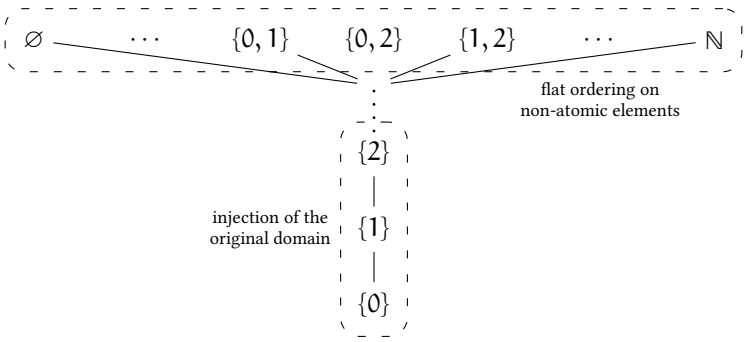
Collecting semantics domain

- Let $\mathsf{X}, \mathsf{Y} \in \wp(\mathsf{C})$. Then,

$$\mathsf{X} \preccurlyeq^* \mathsf{Y} \text{ iff } \begin{cases} \mathsf{X} = \{\mathsf{x}\} \text{ and } \mathsf{Y} = \{\mathsf{y}\} \text{ and } \mathsf{x} \preccurlyeq \mathsf{y} \text{ for some } \mathsf{x}, \mathsf{y} \in \mathsf{C} & (7.1) \\ \mathsf{X} = \mathsf{Y} & (7.2) \\ \mathsf{X} = \{\mathsf{x}\} \text{ and } \mathsf{Y} \neq \{\mathsf{y}\} \text{ for some } \mathsf{x} \in \mathsf{C} \text{ and for all } \mathsf{y} \in \mathsf{C} & (7.3) \end{cases}$$

♢

Example 7.3. Let $\mathsf{C} = \mathbb{N}$ and $\preccurlyeq = \leqslant$. Then, $(\wp(\mathbb{N}), \leqslant^*)$ is



♢

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

Lemma 7.1. *Let $\mathcal{D} \subseteq \wp(C)$ be a non-empty set of atoms of the form $\{x\}$ for some $x \in C$. Then, $\curlyvee^*\mathcal{D}$ exists if and only if $\curlyvee \cup \mathcal{D}$ exists, and when either one exists $\curlyvee^*\mathcal{D} = \{\curlyvee \cup \mathcal{D}\}$.*

Proof. ($\Longleftarrow$) Suppose $\hat{c} = \curlyvee \cup \mathcal{D}$ is exists. Then, $x \preccurlyeq \hat{c}$ for each $\{x\} \in \mathcal{D}$, and therefore $\{x\} \preccurlyeq^* \{\hat{c}\}$ by (7.1). Now, let $Y \in \wp(C)$ be such that $\{x\} \preccurlyeq^* Y$ for each $\{x\} \in \mathcal{D}$. If $Y = \{y\}$, we conclude $\hat{c} \preccurlyeq y$ and therefore $\{\hat{c}\} \preccurlyeq^* \{y\}$ by (7.1). Otherwise, if $Y \neq \{y\}$, then by (7.3) every atom is smaller than Y, including $\{\hat{c}\}$. Hence, $\curlyvee^*\mathcal{D} = \{\hat{c}\}$. ($\Longrightarrow$) Suppose $\curlyvee^*\mathcal{D}$ is exists, and let $X, Y \in \wp(C)$ be two distinct non-atomic sets (which exist since C is assumed not to be $\{\bot\}$). Note that by (7.3) they both are greater than every atom $\{x\} \in \mathcal{D}$. Since they are non-comparable according to $\preccurlyeq^*$, then $\curlyvee^*\mathcal{D}$ must be atomic for some $\hat{c} \in C$, and therefore $x \preccurlyeq \hat{c}$ by (7.1). Now, let $y \in C$ be such that $x \preccurlyeq y$ for every $\{x\} \in \mathcal{D}$. Then, $\{x\} \preccurlyeq^* \{y\}$ and therefore $\{\hat{c}\} \preccurlyeq^* \{y\}$ by (7.1). Hence, we conclude that $\hat{c} \preccurlyeq y$ and $\curlyvee \cup \mathcal{D} = \hat{c}$. □

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

Proposition 7.4. *For each ordinal δ, $(F^*)^\delta = \{F^\delta\}$.*

Proof. By transfinite induction:

- Let $\delta = 0$. Then, $(F^*)^0 = \bot^* = \{\bot\} = \{F^0\}$.
- Let $\delta + 1$ be a successor ordinal. Then,

$$(F^*)^{\delta+1} = F^*((F^*)^\delta) \overset{\text{IH}}{=} F^*(\{F^\delta\}) = \{F(F^\delta)\} = \{F^{\delta+1}\}$$

- Let λ be a limit ordinal. Then, $(F^*)^\lambda = \curlyvee^*_{\delta<\lambda}(F^*)^\delta \overset{\text{IH}}{=} \curlyvee^*_{\delta<\lambda}\{F^\delta\}$. Since $(\{F^\delta\} \mid \delta < \lambda)$ is a set of atoms, by Lemma 7.1 we conclude

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

□

By Proposition 7.4, F^* is a proper semantic transformer over the previously defined collecting semantics domain. The fixpoint definition of the collecting semantics now follows from the application of the Kleenian fixpoint transfer theorem in its most general formulation (that we now recall).

Theorem 7.1 (Kleenian Fixpoint Transfer Theorem [Cou02]). *Let* $(\mathsf{D}, \leqslant, \bot, \vee)$ *and* $(\mathsf{D}^\natural, \leqslant^\natural, \bot^\natural, \vee^\natural)$ *be two semantic domains and* $\mathsf{F}\colon \mathsf{D} \to \mathsf{D}$ *and* $\mathsf{F}^\natural\colon \mathsf{D}^\natural \to \mathsf{D}^\natural$ *two semantic transformer over them. Let* $\alpha\colon \mathsf{D} \to \mathsf{D}^\natural$ *be a function such that*

(i) $\alpha(\bot) = \bot^\natural$;

(ii) $\mathsf{F}^\natural \circ \alpha = \alpha \circ \mathsf{F}$; *and*

(iii) α *preserves the lub of the iterates, that is*

$$\alpha(\vee_{\delta<\lambda} \mathsf{F}^\delta) = \vee^\natural_{\delta<\lambda} \alpha(\mathsf{F}^\delta)$$

for each limit ordinal λ.

Then, $\alpha(\mathrm{lfp}^{\leqslant}_{\bot} \mathsf{F}) = \mathrm{lfp}^{\leqslant^\natural}_{\bot^\natural} \mathsf{F}^\natural$.

The singleton function $\{-\}\colon \mathsf{C} \to \wp(\mathsf{C})$ introduced in Proposition 7.3 satisfies the hypotheses for α in Theorem 7.1 for domains $\langle \mathsf{C}, \preccurlyeq, \bot, \curlyvee \rangle$ and $\langle \wp(\mathsf{C}), \preccurlyeq^*, \bot^*, \curlyvee^* \rangle$ and transformers F and F^*, respectively. Therefore, given the above,

Theorem 7.2 (Fixpoint Collecting Semantics). *The function* $\{-\}\colon \mathsf{C} \to \wp(\mathsf{C})$ *satisfies the hypotheses (i), (ii), and (iii) of Theorem 7.1, hence*

$$[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \mathrm{lfp}^{\preccurlyeq^*}_{\bot^*} \mathsf{F}^*$$

Proof. (i) follows from $\bot^* = \{\bot\}$, (iii) is a consequence of Proposition 7.4, and (ii) holds by $(\mathsf{F}^* \circ \{-\})(\mathsf{c}) = \{\mathsf{F}(\mathsf{c})\} = (\{-\} \circ \mathsf{F})(\mathsf{c})$ for each $\mathsf{c} \in \mathsf{C}$. Moreover,

$$\begin{aligned}
[\![\mathsf{P}]\!]_{\mathscr{C}^*} &= \{[\![\mathsf{P}]\!]_{\mathscr{C}}\} && \text{by Prop. 7.2} \\
&= \{\mathrm{lfp}^{\preccurlyeq}_{\bot} \mathsf{F}\} && \text{by hypothesis on } \mathscr{C} \\
&= \mathrm{lfp}^{\preccurlyeq^*}_{\bot^*} \mathsf{F}^* && \text{by Thm. 7.1}
\end{aligned}$$

and hence the thesis. □

7.3 Basic Notions of Algebraic Abstract Semantics

We proceed to characterise *abstract interpretations* of the standard collecting semantics in the algebraic setting. There are several frameworks in which to design sound approximations of program semantics [CC92]. Here, we study both the ideal case in which a *Galois connection* (gc) ties the concrete and the abstract domain, and the more general scenario characterised by the absence of a best approximation function. Although it is not the most general abstract interpretation framework to work with, it meets the usual setting in which static analyses are designed [RY20].

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

Abstract algebra

Concretisation function

Abstract domain

Abstract semantics

Definition 7.3 (Soundness)**.** Let $\mathscr{A}$ be an abstract *Sg*-algebra (with carrier set A) and $\gamma\colon A \to \wp(C)$ a monotone concretisation function. The algebra $\mathscr{A}$ is *sound* if its operators soundly approximate the concrete ones, i.e.,

(i) $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ for each $f\colon w \to s$; and

(ii) $[\![k]\!]_{\mathscr{C}^*} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ for each constant $k\colon s$. ♢

A straightforward consequence of this definition is that sound algebras induce sound abstract semantic functions:

Proposition 7.5. *If $\mathscr{A}$ is sound, then $[\![P]\!]_{\mathscr{C}^*} \subseteq \gamma[\![P]\!]_{\mathscr{A}}$ for each program P.*

Proof. By structural induction on P:

- Let $P = k$ for some $k\colon s$ in *Sg*. Then, it follows directly by Definition 7.3.
- Let $P = f(P_1, \ldots, P_n)$. By regularity, there are $w = s_1, \ldots, s_n$ and s for which $f\colon w \to s$ and such that (w, s) is the small rank with respect to $w_0 = ls(P_1), \ldots, ls(P_n)$. Since P is well-formed, it follows that $P_i\colon s_i$ for each $1 \leqslant i \leqslant n$ and $P\colon s$. Then,

$$\begin{aligned}\gamma[\![f(P_1, \ldots, P_n)]\!]_{\mathscr{A}} &= (\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}})([\![P_1]\!]_{\mathscr{A}}, \ldots, [\![P_n]\!]_{\mathscr{A}}) \\ &\supseteq ([\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma)([\![P_1]\!]_{\mathscr{A}}, \ldots, [\![P_n]\!]_{\mathscr{A}})\end{aligned}$$

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

$$\begin{aligned}&\supseteq [\![f\colon w \to s]\!]_{\mathscr{C}^*}([\![P_1]\!]_{\mathscr{C}^*}, \ldots, [\![P_n]\!]_{\mathscr{C}^*}) \\ &= [\![f(P_1, \ldots, P_n)]\!]_{\mathscr{C}^*}\end{aligned}$$ □

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

Definition 7.4 (Most Precise Algebra)**.** Let $(\wp(C), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ be a gc between the concrete and the abstract domain. The *most precise algebra* $\mathscr{A}^\diamond$ approximating $\mathscr{C}$ with respect to (α, γ) is defined as follows:

(i) carrier sets are $[\![s]\!]_{\mathscr{A}^\diamond} = A_s$ (recall the notation for sorted sets); and

(ii) the semantics of the operators is $[\![k]\!]_{\mathscr{A}^\diamond} = \alpha[\![k]\!]_{\mathscr{C}^*}$ for each constant $k\colon s$, $[\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} = \alpha \circ [\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma$ for each function symbol $f\colon w \to s$. ♢

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

Proposition 7.6. *$\mathscr{A}^\diamond$ soundly approximates the concrete semantics. Moreover, $\mathscr{A}^\diamond$ is the most precise abstraction with respect to (α, γ), that is, for any other sound algebra $\mathscr{A}$, $\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ and $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ for each operator in Sg. Therefore, by Proposition 7.5, $\gamma[\![P]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![P]\!]_{\mathscr{A}}$ for each program P.*

Proof. We first prove that $\mathscr{A}^\diamond$ is sound. First condition of Definition 7.3 follows by extensivity of $\gamma \circ \alpha$ and monotonicity of $[\![f\colon w \to s]\!]_{\mathscr{C}^*}$, and the second one follows by (α, γ) being a Galois connection. Proving that $\mathscr{A}^\diamond$ is the most precise abstraction with respect to (α, γ) simply means that (1) $\gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}^\diamond} \subseteq \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$ and (2) $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ for each operator in *Sg*.

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

(2) (The proof of $\gamma[\![k]\!]_{\mathscr{A}^\diamond} \subseteq \gamma[\![k]\!]_{\mathscr{A}}$ is similar.)

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

In general, abstraction and concretisation functions α and γ are not homomorphisms between $\mathscr{A}$ and $\mathscr{C}^*$. However, if they are homomorphisms, then $\mathscr{A}$ is *backward* and *forward complete*, respectively (Propositions 7.7 and 7.8).

Definition 7.5 ((Backward) Completeness)**.** Let $\mathscr{A}$ be an *Sg*-algebra and $(\wp(C), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ a gc. The left adjoint α encodes concrete properties in the abstract domain. The algebra $\mathscr{A}$ is *complete* with respect to

(i) a function symbol $f\colon w \to s$ if $\alpha \circ [\![f\colon w \to s]\!]_{\mathscr{C}^*} = [\![f\colon w \to s]\!]_{\mathscr{A}} \circ \alpha$; and

(ii) a constant $k\colon s$ if $\alpha[\![k]\!]_{\mathscr{C}^*} = [\![k]\!]_{\mathscr{A}}$. ♢

Proposition 7.7. *Let $\alpha\colon \mathscr{C}^* \to \mathscr{A}$ be an Sg-homomorphism. Then, $\mathscr{A}$ is complete with respect to each operator in Sg, and therefore $\alpha[\![P]\!]_{\mathscr{C}^*} = [\![P]\!]_{\mathscr{A}}$ for each program P.*

Proof. A simple structural induction on P. □

Note that, in general, the existence of a best abstract approximation is not guaranteed (e.g., for convex polyhedra [CH78] or the final state automata domain [AM19]). In such cases, a dual notion of completeness (*forward completeness* [GRS00]) with respect to the concretisation function is investigated.

Definition 7.6 (Forward Completeness)**.** Let $\mathscr{A}$ be an *Sg*-algebra and $\gamma\colon A \to \wp(C)$ a monotone concretisation function. The algebra $\mathscr{A}$ is *forward complete* with respect to

(i) a function symbol $f\colon w \to s$ if $[\![f\colon w \to s]\!]_{\mathscr{C}^*} \circ \gamma = \gamma \circ [\![f\colon w \to s]\!]_{\mathscr{A}}$; and

(ii) a constant $k\colon s$ if $[\![k]\!]_{\mathscr{C}^*} = \gamma[\![k]\!]_{\mathscr{A}}$. ♢

Proposition 7.8. *Let $\gamma\colon \mathscr{A} \to \mathscr{C}^*$ be an Sg-homomorphism. Then, $\mathscr{A}$ is forward complete with respect to each operator in Sg, and therefore $[\![P]\!]_{\mathscr{C}^*} = \gamma[\![P]\!]_{\mathscr{A}}$ for each program P.*

Proof. By structural induction on P. □

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

Figure 7.3: Syntax of the imperative language While.

7.4 The Multi-Language Abstraction

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

In Section 7.4.1 we set up our example. We present the core of an imperative language While, and by recalling the multi-language construction of Definition 4.1, we apply the construction so that While interoperates with a very simple mathematical language Num (in the spirit of Octave). In Section 7.4.2 we define the combination of abstract interpretations of different languages, a key contribution of our work. By example we apply our theory to two different sign abstract semantics (one for each language), and we show how to derive the sign semantics for the multi-language NumWhile obtained by blending While and Num.

7.4.1 The Multi-Language Construction: A Running Example

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

Running Example 7.1. Let While be (the syntax of) the imperative programming language in Figure 7.3, similar to the language introduced in Example 7.1. Variables $\mathtt{x} \in Var$ and values $\mathtt{i} \in \mathbb{Z}_\perp$ (where $\mathbb{Z}_\perp = \mathbb{Z} \cup \{\perp\}$) occur in the language as terminal symbols, and for each production defining the syntax of While (on the right), we introduce a corresponding algebraic operator (on the left), or a family of operators when they are parametric on a subscript. The rank of each algebraic operator can be inferred by the non-terminals appearing in the production rules; for instance, the operator *cond* is sorted as $cond\colon exp_1, com_1, com_1 \to com_1$, where com_1 and exp_1 denote the sort of commands and expressions of While. The denotational semantics $[\![-]\!]_{\mathscr{D}_1}$ is provided by the While-algebra $\mathscr{D}_1$ in Figure 7.5. As usual, we let $Env_1 = Var \to \mathbb{Z}_\perp$ be the set of environments of While, and we set $Env_1^\perp = Env_1 \cup \{\perp\}$.

$$
\begin{array}{rcl}
[\![exp_1]\!]_{\mathscr{D}_1} & = & Env_1 \to \mathbb{Z}_\perp \\
[\![i]\!]_{\mathscr{D}_1} & = & \rho_1 \mapsto i \\
[\![x]\!]_{\mathscr{D}_1} & = & \rho_1 \mapsto \rho_1(x) \\
[\![bop_\odot]\!]_{\mathscr{D}_1}(e_1, e_2) & = & \rho_1 \mapsto e_1(\rho_1) \odot e_2(\rho_1)
\end{array}
$$

(a) Denotational semantics of While expressions.

$$
\begin{array}{rcl}
[\![com_1]\!]_{\mathscr{D}_1} & = & Env_1^\perp \xrightarrow{\perp} Env_i^\perp \\
[\![skip]\!]_{\mathscr{D}_1} & = & \rho \mapsto \rho \\
[\![assign_x]\!]_{\mathscr{D}_1}(e) & = & \rho \overset{\perp}{\longmapsto} \rho[x \leftarrow e(\rho)] \\
[\![seq]\!]_{\mathscr{D}_1}(c_1, c_2) & = & \rho \mapsto (c_2 \circ c_1)(\rho) \\
[\![cond]\!]_{\mathscr{D}_1}(e, c_1, c_2) & = & \rho \overset{\perp}{\longmapsto} \begin{cases} c_1(\rho) & e(\rho) \neq 0 \\ c_2(\rho) & e(\rho) = 0 \end{cases} \\
[\![loop]\!]_{\mathscr{D}_1}(e, c) & = & \mathrm{lfp}^{\sqsubseteq}_{\perp} F_{e,c} \\
\text{where} \quad F_{e,c}(f) & = & \rho \overset{\perp}{\longmapsto} \begin{cases} \rho & e(\rho) = 0 \\ f(c(\rho)) & e(\rho) \neq 0 \end{cases}
\end{array}
$$

(b) Denotational semantics of While commands.

Figure 7.5: Denotational semantics of While.

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

Figure 7.6: Syntax of the mathematical language Num.

The carrier sets of commands and expressions are $[\![com_1]\!]_{\mathscr{D}_1} = Env_1^\perp \to Env_1^\perp$ and $[\![exp_1]\!]_{\mathscr{D}_1} = Env_1 \to \mathbb{Z}_\perp$. Moreover, we assume that While provides users with very basic operators, i.e., $\odot \in \{+, -, <, >, ==, !=\}$.

We let Num be a mathematical language with more advanced numerical functions, such as modulo and bitwise operators, rational numbers, trigonometric functions, etc. Its syntax is depicted in Figure 7.6. We consider variables $x \in Var$ and values $q \in \mathbb{Q}_\perp = \mathbb{Q} \cup \{\perp\}$ terminal symbols. We denote by f_n mathematical functions of the language with arity n, such as the modulo binary operator `%`, the unary `sin` function, etc. Denotational semantics $[\![-]\!]_{\mathscr{D}_2}$ induced by the Num-algebra $\mathscr{D}_2$, where $[\![exp_2]\!]_{\mathscr{D}_2} = Env_2 \to \mathbb{Q}_\perp$ and $[\![com_2]\!]_{\mathscr{D}_2} = Env_2 \to Env_2$ with $Env_2 = Var \to \mathbb{Q}_\perp$ is given in Figure 7.7. ♢

Recall (Definition 4.1) that the signature of a multi-language, which we shall

$$
\begin{aligned}
[\![exp_2]\!]_{\mathscr{D}_2} &= Env_2 \to \mathbb{Q}_\perp \\
[\![q]\!]_{\mathscr{D}_2} &= \rho_2 \mapsto q \\
[\![x]\!]_{\mathscr{D}_2} &= \rho_2 \mapsto \rho_2(x) \\
[\![f_n]\!]_{\mathscr{D}_2}(e_1, e_2) &= \rho_2 \mapsto f_n(e_1(\rho_2), e_2(\rho_2)) \\
[\![?]\!]_{\mathscr{D}_2}(e_1, e_2, e_3) = \rho_2 \mapsto &\begin{cases} \perp & e_1(\rho_2) = \perp \\ e_2(\rho_2) & e_1(\rho_2) \neq 0 \\ e_3(\rho_2) & e_1(\rho_2) = 0 \end{cases}
\end{aligned}
$$

Figure 7.7: Denotational semantics of Num expressions.

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

The multi-language program on the right implements the exponentiation by squaring [CFA+05] for efficiently computing the powers of an integer number: It stores in `res` the n-th power of `a`. The binary operator `&` is the bitwise and operator and `>>` is the right shift operation.

Figure 7.8: Multi-language exponentiation by squaring algorithm.

refer to as ***Sg***, is specified by blending the order-sorted signatures Sg_1 and Sg_2 of the single-languages through an *interoperability relation* $\ltimes$ on sorts. In particular, an interoperability constraint $s \ltimes s'$ implies that Sg_i-terms of sort s can be used in place of Sg_j-terms of sort s' (with $i, j \in \{1, 2\}$ and $i \neq j$). This determines the terms of the multi-language $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$.

Running Example 7.2. For instance, Num provides users with more advanced binary operators and values than those of While. However, the purpose of Num is limited to define handy mathematical functions (indeed, it is not even Turing-complete). We can take advantage of such mathematical expressiveness without sacrificing computational power by allowing the use of Num-expressions (that is, terms with sort exp_2) into While-programs, in place of While-expression of sort exp_1. Therefore, the interoperability relation shall simply specify $exp_2 \ltimes exp_1$, and we let NumWhile to be formally defined as $\mathsf{NumWhile} = (\mathsf{While}, \mathsf{Num}, \ltimes)$. As a result, programmers may write programs such as the one in Figure 7.8, where terms in magenta are Num-expressions used in place of While-expressions (that is, we use colours as we did in previous chapters rather than applying the conversion operator $\hookrightarrow_{exp_2, exp_1}$ for clarity reasons). ♢

The multi-language ***Sg***-semantics $\mathscr{C}$ is then obtained by pairing the algebras $\mathscr{C}_1$ and $\mathscr{C}_2$ with boundary functions $[\![s \ltimes s']\!]_{\mathscr{C}} : [\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j}$ that specify how terms of sort s in Sg_i can be interpreted as terms of sort s' in Sg_j. In other words, boundary functions regulate the flow of values between the underlying languages [MF09]. For instance, they can act as a bridge between different type representations in the underlying languages (e.g., to enable the interoperability of Java and JavaScript in Nashorn [Orab] or the interoperability of Java and Kotlin [Jet]), or deal between different machine-integers implementation (e.g., the mapping between Java primitive types and C types in JNI [Oraa]), etc.

Running Example 7.3. We denote by $\mathscr{D}$ the multi-language NumWhile-algebra defined by coupling $\mathscr{D}_1$ and $\mathscr{D}_2$ with the boundary function $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ defined below (recall Definition 4.2). Note that values of Num-expressions range over the set of rational numbers $\mathbb{Q}$, whereas While only handles integer values in $\mathbb{Z}$. A natural choice for the boundary function $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ of NumWhile that converts Num-expressions to While-expressions is to *truncate* the value of Num-expression to their nearest integer value. Since expressions only have values with respect to an environment, we shall specify a conversion from $Env_2 \to \mathbb{Q}_\perp$ to $Env_1 \to \mathbb{Z}_\perp$ (we use ρ_1 and ρ_2 as metavariables for Env_1 and Env_2, respectively):

$$[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}(e \in Env_2 \to \mathbb{Q}_\perp) = \rho_1 \in Env_1 \mapsto \psi(e(\phi(\rho_1))) \in \mathbb{Z}_\perp$$

where $\phi\colon Env_1 \to Env_2$ is the inclusion function and $\psi\colon \mathbb{Q}_\perp \to \mathbb{Z}_\perp$ truncates rational values, i.e., $\psi(q) = \text{truncate}(q)$ if $q \neq \perp$ and $\psi(\perp) = \perp$. For instance, the semantics of the Num-expression $[\![\mathsf{n}\ /\ 2]\!]_{\mathscr{D}_2} = \rho_2 \mapsto \rho_2(\mathsf{n})\ /\ 2$ is mapped by $[\![exp_2 \ltimes exp_1]\!]_{\mathscr{D}}$ to the function $\rho_1 \mapsto \rho_1(\mathsf{n})\ /_{\mathbb{Z}}\ 2$, where $/_{\mathbb{Z}}$ is the integer division (we ignore the case where $\rho_2(\mathsf{n}) = \perp$, as it is clearly trivial). ♢

7.4.2 Combining Abstractions of Different Languages

The first step towards an abstract interpretation theory for multi-languages is to find a suitable notion of *multi-language collecting semantics*. A satisfactory definition is obtained by pairing the collecting semantics $\mathscr{C}_1^*$ and $\mathscr{C}_2^*$ of the underlying languages with the lifting of boundary functions provided by $\wp$:

Definition 7.7 (Multi-Language Collecting Semantics)**.** Let $\mathscr{C}$ be a multi-language $\boldsymbol{Sg}$-algebra over the multi-language signature $\boldsymbol{Sg} = (Sg_1, Sg_2, \ltimes)$. The *multi-language collecting semantics* $\mathscr{C}^*$ over $\mathscr{C}$ is specified by:

(i) the collecting Sg_i-semantics $\mathscr{C}_i^*$ over $\mathscr{C}_i$, for $i = 1, 2$; and

(ii) boundary functions $[\![s \ltimes s']\!]_{\mathscr{C}^*} = \wp[\![s \ltimes s']\!]_{\mathscr{C}}$ for each $s \ltimes s'$. ♢

We can then lift Proposition 7.2 and 7.3 to the multi-language world in order to show that $\mathscr{C}^*$ has the desired properties:

Proposition 7.9. *Let $\mathscr{C}$ be a multi-language $\boldsymbol{Sg}$-algebra. The collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$ computes the strongest program property for each* multi-language *program P generated by $\boldsymbol{Sg}$, that is $[\![\mathsf{P}]\!]_{\mathscr{C}^*} = \{[\![\mathsf{P}]\!]_{\mathscr{C}}\}$. Moreover, the singleton function $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$ is a multi-language $\boldsymbol{Sg}$-homomorphism.*

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

- if $\mathsf{P} = \mathsf{k}_i$ for some constant symbol $\mathsf{k}\colon s$ in Sg_i, then

$$[\![\mathsf{k}_i]\!]_{\mathscr{C}^*} = [\![\mathsf{k}]\!]_{\mathscr{C}_i^*} \overset{\text{Prop. 7.2}}{=} \{[\![\mathsf{k}]\!]_{\mathscr{C}_i}\} = \{[\![\mathsf{k}_i]\!]_{\mathscr{C}}\}$$

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

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

- if $\mathsf{P} = \hookrightarrow_{s,s'}(\mathsf{P}')$ for some $s \ltimes s'$ in $\boldsymbol{Sg}$, then

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

Now, the homomorphic nature of $\{-\}\colon \mathscr{C} \to \mathscr{C}^*$ follows by Proposition 7.3 and by observing that $\{-\}$ trivially commutes with boundary functions. □

We are now interested in whether we can obtain a fixpoint definition of the multi-language collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$. At a minimum, a fixpoint definition of the two underlying language semantics is needed, since every single-language program is also a multi-language one. However, the semantics of the multi-language does not only depend on these specifications (that is, it is not a *universal property* of the underlying semantics) but is determined up to a family of boundary functions defining the interoperability of Sg_1 and Sg_2. The next proposition states that assuming the fixpoint definition of $[\![-]\!]_{\mathscr{C}_1}$ and $[\![-]\!]_{\mathscr{C}_2}$, the fixpoint calculation of the multi-language collecting semantics boils down to a constructive specification of the boundary functions in $\mathscr{C}$.

Theorem 7.3. *Let $\mathscr{C}$ be a multi-language $\boldsymbol{Sg}$-algebra whose boundary functions admit a constructive definition, that is $[\![s \ltimes s']\!]_{\mathscr{C}} = \operatorname{lfp} \mathsf{F}_{s \ltimes s'}$ for each $s \ltimes s'$ in $\boldsymbol{Sg}$ and for some $\mathsf{F}_{s \ltimes s'}\colon ([\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j}) \to ([\![s]\!]_{\mathscr{C}_i} \to [\![s']\!]_{\mathscr{C}_j})$. Then, the multi-language collecting semantics $[\![-]\!]_{\mathscr{C}^*}$ induced by $\mathscr{C}^*$ admits a fixpoint computation if and only if $\mathscr{C}_1$ and $\mathscr{C}_2$ does.*

Proof. Each operator in $\boldsymbol{Sg}$ admits a fixpoint definition. □

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

Running Example 7.4. Let $\langle A_{\mathcal{V}}, \sqsubseteq_{\mathcal{V}}, \sqcup_{\mathcal{V}}, \sqcap_{\mathcal{V}}, \bot_{\mathcal{V}}, \top_{\mathcal{V}}\rangle$ be the standard sign domain (Figure 7.9) and $\widetilde{\gamma}_i\colon A_{\mathcal{V}} \to \wp(Val_i)$ (where $Val_1 = \mathbb{Z}_\bot$ and $Val_2 = \mathbb{Q}_\bot$) the corresponding concretisation function (Figure 7.10). We let $\mathscr{A}_1$ and $\mathscr{A}_2$ be the abstract algebras defining a sign analysis for languages While and Num, respectively (that is, the computation induced by $\mathscr{A}_i$ is carried out using abstract values in $A_{\mathcal{V}}$ instead

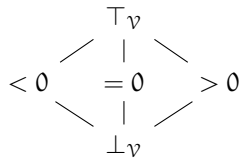


Figure 7.9: Sign abstract domain.

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

Figure 7.10: Concretisation functions $\widetilde{\gamma}_i \colon A_{\mathcal{V}} \to \wp(\mathit{Val}_i)$.

$$\begin{array}{rcl}
[\![\mathit{exp}_1]\!]_{\mathscr{A}_1} & = & \mathit{Env}^\sharp \to A_{\mathcal{V}} \\
[\![i]\!]_{\mathscr{A}_1} & = & \rho^\sharp \mapsto \widetilde{\alpha}_1(\{i\}) \\
[\![x]\!]_{\mathscr{A}_1} & = & \rho^\sharp \mapsto \rho^\sharp(x) \\
[\![\mathit{bop}_\odot]\!]_{\mathscr{A}_1}(e_1^\sharp, e_2^\sharp) & = & \rho^\sharp \mapsto e_1^\sharp(\rho^\sharp) \odot^\sharp e_2^\sharp(\rho^\sharp)
\end{array}$$

Figure 7.11: Sign semantics of While expressions.

of concrete ones in Val_i). The abstract semantics $\mathscr{A}_i$ are given in Figures 7.11, 7.12, and 7.13.

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

$$(\gamma_i)_{\mathit{exp}_i}(e^\sharp) = \{e_i \in [\![\mathit{exp}_i]\!]_{\mathscr{D}_i} \mid \forall \rho^\sharp \in \mathit{Env}^\sharp \,.\, \forall \rho_i \in \gamma_i(\rho^\sharp) \,.\, e_i(\rho_i) \in \widetilde{\gamma}_i(e^\sharp(\rho^\sharp))\}$$
$$(\gamma_i)_{\mathit{com}_i}(c^\sharp) = \{c_i \in [\![\mathit{com}_i]\!]_{\mathscr{D}_i} \mid \forall \rho^\sharp \in \mathit{Env}^\sharp \,.\, \forall \rho_i \in \gamma_i(\rho^\sharp) \,.\, c_i(\rho_i) \in \gamma_i(c^\sharp(\rho^\sharp))\}$$

♢

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

Theorem 7.4 (Soundness). *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be sound Sg_i-algebras with concretisation functions $\gamma_i \colon A_i \to \wp(C_i)$, for $i = 1, 2$. If $[\![s \ltimes s']\!]_{\mathscr{C}^*} \circ \gamma_i \subseteq \gamma_j \circ [\![s \ltimes s']\!]_{\mathscr{A}}$ for each $s \ltimes s'$, then the multi-language abstract semantics $\mathscr{A}$ is sound, that is $[\![P]\!]_{\mathscr{C}^*} \subseteq \gamma [\![P]\!]_{\mathscr{A}}$ for each multi-language program P generated by **Sg**.*

$$[\![com_1]\!]_{\mathscr{A}_1} = Env^\sharp \to Env^\sharp$$

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

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

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

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

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

Figure 7.12: Sign semantics of While commands.

$$\begin{aligned} [\![exp_2]\!]_{\mathscr{A}_2} &= Env^\sharp \to A_{\mathcal{V}} \\ [\![q]\!]_{\mathscr{A}_2} &= \rho^\sharp \mapsto \widetilde{\alpha}_2(\{q\}) \\ [\![x]\!]_{\mathscr{A}_2} &= \rho^\sharp \mapsto \rho^\sharp(x) \\ [\![f_n]\!]_{\mathscr{A}_2}(e_1^\sharp, e_2^\sharp) &= \rho^\sharp \mapsto f_n^\sharp(e_1^\sharp(\rho^\sharp), e_2^\sharp(\rho^\sharp)) \\ [\![?]\!]_{\mathscr{A}_2}(e_1^\sharp, e_2^\sharp, e_3^\sharp) &= \rho^\sharp \mapsto \begin{cases} \bot_{\mathcal{V}} & e_1^\sharp(\rho^\sharp) = \bot_{\mathcal{V}} \\ e_2^\sharp(\rho^\sharp) & e_1^\sharp(\rho^\sharp) \in \{>0, <0\} \\ e_3^\sharp(\rho^\sharp) & e_1^\sharp(\rho^\sharp) = (=0) \end{cases} \end{aligned}$$

Figure 7.13: Sign semantics of Num.

Proof. We first prove that for each constant $k_i : s$ and function symbol $f_i : w \to s$ in ***Sg*** we have $[\![k_i]\!]_{\mathscr{C}^*} \subseteq \gamma_s[\![k]\!]_{\mathscr{A}}$ and $[\![f_i : w \to s]\!]_{\mathscr{C}^*} \subseteq \gamma_s[\![f_i : w \to s]\!]_{\mathscr{A}}$. Then, $[\![P]\!]_{\mathscr{C}^*} \subseteq \gamma[\![P]\!]_{\mathscr{A}}$ for each multi-language program P generated by ***Sg*** follows by a simple structural induction, in the same style of the proof of Proposition 7.5. Let $\gamma_i : \wp(C_i) \to A_i$ be the concretisation function of $\mathscr{A}_i$,

- $[\![k_i]\!]_{\mathscr{C}^*} = [\![k]\!]_{\mathscr{C}_i^*} \subseteq (\gamma_i)_s[\![k]\!]_{\mathscr{A}_i} = \gamma_s[\![k_i]\!]_{\mathscr{A}}$; and
- $$\begin{aligned} [\![f_i : w \to s]\!]_{\mathscr{C}^*} \circ \gamma_w &= [\![f : w \to s]\!]_{\mathscr{C}_i^*} \circ (\gamma_i)_w \\ &\subseteq (\gamma_i)_s \circ [\![f : w \to s]\!]_{\mathscr{A}_i} \\ &= \gamma_s \circ [\![f_i : w \to s]\!]_{\mathscr{A}} \end{aligned}$$

□

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

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

Figure 7.14: The abstract iterates of the loop in Figure 7.8.

Theorem 7.5 (Completeness). *Let $\mathscr{A}$ be the multi-language abstract semantics. If the order-sorted Sg_i-algebra $\mathscr{A}_i$ is forward (resp., backward) complete with respect to $k : s$ and $f : w \to s$ in Sg_i, then $\mathscr{A}$ is forward (resp., backward) complete with respect $k_i : s$ and $f_i : w \to s$ in $\boldsymbol{Sg}$, respectively.*

Proof. We only prove the forward completeness of the function symbol $f_i : w \to s$ in $\boldsymbol{Sg}$, the other cases are similar.

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

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

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

Corollary 7.1. *Let $\mathscr{A}$ be the multi-language abstract semantics and $\llbracket s \ltimes s' \rrbracket_{\mathscr{A}}$ a forward (resp., backward) complete boundary functions. For each multi-language program P sorted by s, if P is forward (resp., backward) complete, then so too is the program $\hookrightarrow_{s,s'}(P)$.*

Equivalent multi-language versions of Propositions 7.7 and 7.8 also apply. The proof boils down to the fact that multi-language homomorphisms are pair of order-sorted homomorphisms that also commute with boundary functions.

Running Example 7.5. The multi-language abstract algebra $\mathscr{A}$ is obtained by combining $\mathscr{A}_1$ and $\mathscr{A}_2$ with an abstraction of the boundary function $\llbracket exp_2 \ltimes exp_1 \rrbracket_{\mathscr{D}}$ defined in Example 7.3. Since the underlying algebras share the same abstract domain for expressions, that is $\llbracket exp_1 \rrbracket_{\mathscr{A}_1} = \llbracket exp_2 \rrbracket_{\mathscr{A}_2}$, there is no need to convert abstract values between two identical domains, therefore we set $\llbracket exp_2 \ltimes exp_1 \rrbracket_{\mathscr{A}} = id$.

Let us show the computation of the abstract semantics of the multi-language program in Figure 7.8, starting from the set of initial states in which both a and n are greater than 0. The abstract precondition before entering the loop is given by the abstract environment $\{a \mapsto (>0), n \mapsto (>0), res \mapsto (>0)\}$, where res is positive due to the assignment on line 1. The abstract iterates of the loop converge in three steps, as shown in Figure 7.14. Note that the abstract interpretation of the program guarantees the result of a^n, with $a, n > 0$, to be positive. The imprecision on the final abstract value of the variable n is due to poor choice of the abstract domain (which can be easily refined). Since the example trivially satisfies the hypotheses of Theorem 7.4, the result is sound. ♢

CHAPTER 8

Semantic Equivalence of Language Syntax Formalisms

Chapter reference(s): [BM19b, BM20]

Several formalisms for language syntax specification exist in literature [Vis97]. Among them, *formal grammars* [Cho56, Ear83, Knu68] and *algebraic signatures* [CC81, Hig63, GM92] have played and still play a pivotal role. The former are widely used to define syntax of programming languages [Sco09], notably due to compelling results on context-free parsing techniques [Ear83, Val75, Lee02]. The latter provide an algebraic approach to syntax specification, and they are ubiquitous in the fields of universal algebra [CC81], model theory [CK92], and logics in general.

In the field of program analysis, the syntax of programming languages is usually presented by context-free grammars, whereas the multi-language formalisation provided in the previous chapters is based on order-sorted specifications. In this chapter, we bridge this gap by narrowing the focus to three different syntax formalisms: *context-free grammars* (*Grm*), *many-sorted signatures* (*MSSg*), and *order-sorted signatures* (*OSSg*). The aim is to provide mappings between these categories (see Figure 8.1) able to translate language syntax specifications from one formalism to another without altering their classes of semantics. Put differently, if $Alg(\mathscr{X})$ denotes the class of semantics of an object $\mathscr{X}$ (in *Grm*, *MSSg*, or *OSSg*) and $\Upsilon\mathscr{X}$ the conversion of $\mathscr{X}$ to another formalism, we shall prove that $Alg(\mathscr{X})$ and $Alg(\Upsilon\mathscr{X})$ are equivalent, meaning that $\mathscr{X}$ and $\Upsilon\mathscr{X}$ are essentially the same from a semantic point of view.

Structure We begin by reviewing some well-known results on the equivalence of language syntax formalisms, which will be generalised later on in the chapter. We then present the formalisms under investigation: context-free grammars, many-sorted signatures, and order-sorted signatures. We define the categories of grammars and signatures, and we provide transformations between them in terms of adjoint functors. Finally, we show that these transformations do not alter the semantics of the transformed objects.

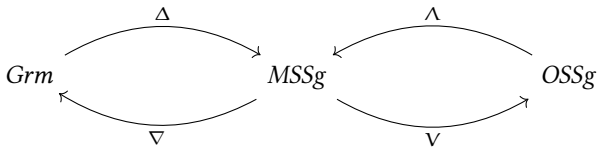


Figure 8.1: The mappings between the different syntax formalisms investigated in this chapter.

8.1 Old and New Results

The mathematical links between these different frameworks have already been partially studied in literature. Goguen et al. [GTWW77] provide a definition of $\Delta\colon \mathit{Grm} \to \mathit{MSSg}$ that yields an isomorphism between the sets of terms (that is, the *term algebras*) over the grammar G and its conversion to many-sorted signature ΔG, and conversely the definition of $\nabla\colon \mathit{MSSg} \to \mathit{Grm}$ that makes the term algebras over the many-sorted signature $\mathfrak{S}$ and the grammar $\nabla\mathfrak{S}$ isomorphic (the proofs are outlined in detail in [Vis98]).

Other results on the subject are given in [GM92]. The authors provide a definition of $\Lambda\colon \mathit{OSSg} \to \mathit{MSSg}$ that gives rise to an equivalence between the categories of algebras over an order-sorted signature *Sg* and its many-sorted conversion ΛSg. Both these results of [GTWW77, GM92] are an instance of the aim of this chapter, as we will prove later.

In the following sections, we unify and broaden such results in a more general setting. We model *Grm*, *MSSg*, and *OSSg* as the *categories* whose objects are grammars, many-sorted and order-sorted signatures, respectively. Arrows between objects in the same category are morphisms preserving the *abstract syntax* of the language [McC62b]. This is a fundamental point: According to [FPT99],

> *the essential syntactical structure of programming languages is not that given by their concrete or surface syntax [...]. Rather, the deep structure of a phrase should reflect its semantic import.* Marcelo Fiore et al. (1999)

This viewpoint is also made explicit in [GTWW77, Rus76], where the semantics of a language is defined by the unique homomorphism from the *initial algebra* (namely the abstract syntax — see the discussion following Proposition 3.4) to another algebra in the same category.

The mappings from one formalism to another are therefore defined in terms of *functors* between the respective categories. Since the naturality of such constructions, the *adjoint* nature of these functors is then investigated, discussing their semantic implications over the categories of algebras.

8.2 Formalisms for Language Syntax Specification

In this section, we provide a brief presentation of the three syntax formalisms discussed in the rest of the article. Their technical aspects are deferred to the next sections.

The most popular formalism to specify languages are *context-free grammars*. They enable language designers to easily handle both abstract and concrete aspects of the syntax by combining terminal symbols with syntactic constituents of the

language through *production rules*. Several definitions of context-free grammars exist in literature [Vis98, RJ98]. Here, we follow [GTWW77] (or, the so-called *algebraic grammars* in [RJ98]) and, for the sake of succintness, we usally refer to them simply as grammars.

Although grammars are an easy-to-use tool for syntax specification, *signatures* provide a more algebraic approach to language definition. The concept of *many-sorted signature* arose in [Hig63] in order to lift the theory of (full) abstract algebras in case of partially defined operations. From the language syntax perspective, signatures allow the specification of sorted operators, which in turn provide a basis for an algebraic construction of the language semantics. In the rest of the chapter, we follow the exposition of [GTWW77] and [BL70] on this subject.

The last formalism considered here are *order-sorted signatures* [GM92], already presented in Chapter 3. They are built upon many-sorted signatures to which they add an explicit treatment of polymorphic operators. Their main aim is to provide a basis on which to develop an algebraic theory to handle several types of polymorphism, multiple inheritance, left inverses of subsort inclusion (retracts), and complete equational deduction.

Notation 8.1. In the rest of the chapter, we will make use of the following notations: If A and B are two sets and $f\colon A \to B$ is a function, we denote by $f^*\colon A^* \to B^*$ the unique *monoid homomorphism* induced by the Kleene closure on the sets A and B extending the function f. That is,

$$\begin{cases} f^*(\varepsilon) = \varepsilon \\ f^*(a_1 \dots a_n) = f(a_1) \dots f(a_n) \quad n > 0 \end{cases}$$

Moreover, 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 $g(X) = \{\, g(a) \in B \mid a \in X \,\}$. ♢

8.3 Context-Free Algebras

A *context-free grammar* [GTWW77] (or, a *CF grammar*) is a triple $G = (N, T, P)$, where N is the set of *non-terminal symbols* (or, *non-terminals*), T is the set of *terminal symbols* (or, *terminals*) disjoint from N, and $P \subseteq N \times (N \cup T)^*$ is the set of *production rules* (or, *productions*). If (A, β) is a production in P, we stick to the standard notation $A \to \beta$ (although some authors [Vis98] reverse the order and write $\beta \to A$ to match the signature formalism).

Context-free grammar, Non-terminals, Terminals, Productions

If $\alpha, \gamma \in (N \cup T)^*$, $B \in N$, and $B \to \beta \in P$, we define $\alpha B \gamma \Rightarrow \alpha\beta\gamma$ the *one-step reduction relation* on the set $(N \cup T)^*$. The language $\mathcal{L}(G)$ generated by G is the union of the N-sorted family $\mathcal{L}_N(G) = \{\, \mathcal{L}_A(G) \mid A \in N \,\}$, that is, $\mathcal{L}(G) = \bigcup \mathcal{L}_N(G)$, where $\mathcal{L}_A(G) = \{\, t \in T^* \mid A \Rightarrow^* t \,\}$ and $\Rightarrow^*$ is the reflexive transitive closure of $\Rightarrow$.

The *non-terminals projection* $\mathrm{nt}\colon N \cup T \to N \cup \{\varepsilon\}$ on G is defined by

Non-terminals projection

$$\mathrm{nt}(x) = \begin{cases} x & x \in N \\ \varepsilon & \text{otherwise} \end{cases}$$

Notation 8.2. In the following, we implicitly characterise the function nt according to the subscript/superscript of G, namely, if G', G_1, etc. are grammars, we denote by nt', nt_1, etc. their non-terminals projections, respectively. ♢

An *abstract grammar morphism* (henceforth *morphism*, when this terminology does not lead to ambiguities) $f\colon G_1 \to G_2$ is a map between two grammars $G_1 = (N_1, T_1, P_1)$ and $G_2 = (N_2, T_2, P_2)$ that preserves the abstract structure of the generated strings.

Abstract grammar morphism

Definition 8.1 (Abstract Grammar Morphism)**.** Let $G_1 = (N_1, T_1, P_1)$ and $G_2 = (N_2, T_2, P_2)$ be two context-free grammars. An *abstract grammar morphism* $f\colon G_1 \to G_2$ is a pair of functions $f_0\colon N_1 \to N_2$ and $f_1\colon P_1 \to P_2$ such that

$$f_1(A \to \beta) = f_0(A) \to \beta'$$

where $nt_2^*(\beta') = (f_0^* \circ nt_1^*)(\beta)$ for each production $A \to \beta$ in P_1. ♢

The identity morphism on an object $G = (N, T, P)$ is denoted by id_G, with $(id_G)_0 = id_N$ and $(id_G)_1 = id_P$. The composition of two grammar morphism $f\colon G_1 \to G_2$ and $g\colon G_2 \to G_3$ is obtained by defining $(g \circ f)_0 = g_0 \circ f_0$ and $(g \circ f)_1 = g_1 \circ f_1$.

Proposition 8.1. *The class of all grammars and the class of all abstract grammar morphisms form the category Grm.*

The following subsection makes clear the semantic implications that a grammar morphism $f\colon G_1 \to G_2$ induces on the categories of algebras over G_1 and G_2. The insight is that preserving the abstract syntax of G_1 into G_2 ensures the possibility to employ G_2-algebras in order to provide meaning to G_1-terms.

8.3.1 Algebras over a Context-Free Grammar

The algebraic approach applied to context-free languages is introduced in [HR76, Rus76]. The authors exploit the theory of *heterogeneous algebras* [BL70] to provide semantics to context-free grammars (see also [GTWW77]). The algebraic notions that lead to the category of algebras over a context-free grammar are here summarised.

Remark 8.1. The following concepts are analogous in spirit to those introduced in Chapter 3. Therefore, we avoid to be didactic in the exposition, and we redirect the reader to the aforementioned chapter for a thorough presentation of the algebraic approach. ♢

Context-free algbera

Definition 8.2 (Context-Free Algebra)**.** A *context-free* G*-algebra* $\mathscr{A}$ over a CF grammar $G = (N, T, P)$ is specified by

Interpretation set

(i) an *interpretation set* $[\![C]\!]_{\mathscr{A}}$ for each non-terminal $C \in N$ and a set $[\![W]\!]_{\mathscr{A}} = [\![C_1]\!]_{\mathscr{A}} \times \cdots \times [\![C_n]\!]_{\mathscr{A}}$ for each $W = C_1 \dots C_n \in N^+$;

(ii) an element $[\![C \to \delta]\!]_{\mathscr{A}} \in [\![C]\!]_{\mathscr{A}}$ for each production rule $C \to \delta$ such that $nt^*(\delta) = \varepsilon$; and

Interpretation function

(iii) an *interpretation function* $[\![C \to \delta]\!]_{\mathscr{A}}\colon [\![nt^*(\delta)]\!]_{\mathscr{A}} \to [\![C]\!]_{\mathscr{A}}$ for each production $C \to \delta$ such that $nt^*(\delta) \neq \varepsilon$. ♢

Notation 8.3. As usual, if $\mathscr{A}$ is a G-algebra, we denote by $[\![C]\!]_{\mathscr{A}}$ and A_C the same interpretation (or, carrier) set, for each non-terminal C. ♢

It is well-known [GTWW77, HR76] that the class of all G-algebras and the class of all G-homomorphisms form a category, denoted by $Alg(G)$. The *initial object* in $Alg(G)$ is the *term algebra* (or, *initial algebra*) and it is denoted by $\mathscr{T}_G$. Specifically, the carrier sets $(T_G)_C$ of $\mathscr{T}_G$ are inductively defined as the smallest sets such that, for each $C \to \delta \in P$,

- if $nt^*(\delta) = \varepsilon$, then $C \to \delta \in (T_G)_C$; and
- if $nt^*(\delta) = C_1 \dots C_n$ and $t_i \in (T_G)_{C_i}$ for $i = 1, \dots, n$, then

$$C \to \delta(t_1, \dots, t_n) \in (T_G)_C$$

Then, the interpretation functions are obtained by defining

- $[\![C \to \delta]\!]_{\mathscr{T}_G} = C \to \delta$, if $nt^*(\delta) = \varepsilon$; and
- $[\![C \to \delta]\!]_{\mathscr{T}_G}(t_1, \dots, t_n) = C \to \delta(t_1, \dots, t_n)$, if $nt^*(\delta) = C_1 \dots C_n$ and $t_i \in (T_G)_{C_i}$ for $i = 1, \dots, n$.

We now show the semantic effects that grammar morphisms induce on the respective categories of algebras. Let $G_1 = (N_1, T_1, P_1)$ and $G_2 = (N_2, T_2, P_2)$ be two context-free grammars. Suppose that $f\colon G_1 \to G_2$ is a grammar morphism and let $\mathscr{A}$ be a G_2-algebra. We can make $\mathscr{A}$ into a G_1-algebra $\xi_f\mathscr{A}$ by defining

$$\begin{aligned} (\xi_f A)_C &= A_{f_0(C)} && \text{for each } C \in N_1, \text{ and} \\ [\![C \to \delta]\!]_{\xi_f\mathscr{A}} &= [\![f_1(C \to \delta)]\!]_{\mathscr{A}} && \text{for each } C \to \delta \in P_1 \end{aligned}$$

Moreover, if $h\colon \mathscr{A} \to \mathscr{B}$ is a G_2-homomorphism, then $(\xi_f h)_C = h_{f_0(C)}$ is G_1-homomorphism from $\xi_f\mathscr{A}$ to $\xi_f\mathscr{B}$.

Proposition 8.2. *The map $\xi_f\colon Alg(G_2) \to Alg(G_1)$ induced by the abstract grammar morphism $f\colon G_1 \to G_2$ is a functor.*

This last proposition suggests to investigate the functorial nature of $Alg(-)$.

Proposition 8.3. *Alg is a contravariant functor from the category of context-free grammar Grm to the category of small categories Cat, or, equivalently, $Alg\colon Grm \to Cat^{op}$ is a (covariant) functor.*

Proof. Let $f\colon G_1 \to G_2$ be a grammar morphism and let $Alg(f) = \xi_f$. Then, for each context-free grammar $G = (N, T, P)$ and for each algebra $\mathscr{A}$ in $Alg(G)$ holds that

$$\begin{aligned} (Alg(id_G)(A))_C = (\xi_{id_G} A)_C &= A_C && \text{for each } C \in N, \text{ and} \\ [\![C \to \delta]\!]_{Alg(id_G)(\mathscr{A})} = [\![C \to \delta]\!]_{\xi_{id_G}\mathscr{A}} &= [\![C \to \delta]\!]_{\mathscr{A}} && \text{for each } C \to \delta \in P \end{aligned}$$

which imply $Alg(id_G) = id_{Alg(G)}$. Moreover, given two grammar morphisms $f\colon G_1 \to G_2$ and $g\colon G_2 \to G_3$ we have that

$$\begin{aligned} \big(Alg(g \circ f)(A)\big)_C &= (\xi_{(g \circ f)} A)_C \\ &= A_{(g_0 \circ f_0)(C)} \\ &= (\xi_g A)_{f_0(C)} \\ &= \big((\xi_f \circ \xi_g)(A)\big)_C \\ &= \big((Alg(f) \circ Alg(g))(A)\big)_C \end{aligned}$$

for each $C \in N$, and

$$\begin{aligned}
[\![C \to \delta]\!]_{Alg(g\circ f)(\mathscr{A})} &= [\![C \to \delta]\!]_{\xi_{(g\circ f)}\mathscr{A}} \\
&= [\![(g_1 \circ f_1)(C \to \delta)]\!]_{\mathscr{A}} \\
&= [\![f_1(C \to \delta)]\!]_{\xi_g \mathscr{A}} \\
&= [\![C \to \delta]\!]_{(\xi_f \circ \xi_g)(\mathscr{A})} \\
&= [\![C \to \delta]\!]_{(Alg(f)\circ Alg(g))(\mathscr{A})}
\end{aligned}$$

for each $C \to \delta \in P$, and thus $Alg(g \circ f) = Alg(f) \circ Alg(g)$. □

Since functors preserve isomorphisms, then isomorphic grammars give rise to isomorphic categories of algebras, implying that f does not lose any (semantic relevant) information.

Corollary 8.1. *If* $f\colon G_1 \to G_2$ *is an abstract grammar isomorphism, then*

$$\xi_{f^{-1}} \circ \xi_f = \mathrm{id}_{Alg(G_1)} \qquad \textit{and} \qquad \xi_f \circ \xi_{f^{-1}} = \mathrm{id}_{Alg(G_2)}$$

Therefore, $\xi_{f^{-1}} = \xi_f^{-1}$ *and hence* $Alg(G_1)$ *and* $Alg(G_2)$ *are isomorphic.*

Example 8.1 (Deriving a Compiler)**.** In this example, we show how a grammar morphism $f\colon G_1 \to G_2$ induces a compiler with respect to any G_2-algebra in $Alg(G_2)$. Consider the following grammar specifications $G_1 = (N_1, T_1, P_1)$ (left) and $G_2 = (N_2, T_2, P_2)$ (right) in the Backus-Naur form:

$$n ::= +(n, n) \mid 0 \mid 1 \mid 2 \mid \cdots \qquad\qquad p ::= (p + p) \mid \text{even} \mid \text{odd}$$

(Here, we have just specified the productions; terminals and non-terminals can be easily recovered from such specifications assuming no useless symbols in both sets).

Let $f\colon G_1 \to G_2$ be the grammar morphism mapping

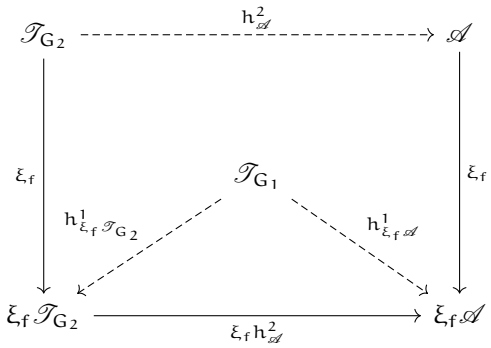
$$\begin{array}{rcll}
n & \overset{f_0}{\longmapsto} & p & \\
n \to +(n, n) & \overset{f_1}{\longmapsto} & p \to (p + p) & \\
n \to 2x & \overset{f_1}{\longmapsto} & p \to \text{even} & \text{for each } x \in \mathbb{N} \\
n \to 2x + 1 & \overset{f_1}{\longmapsto} & p \to \text{odd} & \text{for each } x \in \mathbb{N}
\end{array}$$

Suppose that $\mathscr{A}$ is the G_2-algebra such that

$$\begin{aligned}
[\![p]\!]_{\mathscr{A}} &= \{0, 1\} \\
[\![p \to \text{even}]\!]_{\mathscr{A}} &= 0 \\
[\![p \to \text{odd}]\!]_{\mathscr{A}} &= 1 \\
[\![p \to (p + p)]\!]_{\mathscr{A}}(x_1, x_2) &= (x_1 + x_2) \mod 2
\end{aligned}$$

Let $\mathscr{T}_{G_1}$ and $\mathscr{T}_{G_2}$ denote the G_1- and G_2-term algebras, respectively. Since $\mathscr{T}_{G_2}$ is initial, there is a unique homomorphism $h^2_{\mathscr{A}}\colon \mathscr{T}_{G_2} \to \mathscr{A}$, namely, the semantics of the language generated by G_2 induced by the algebra $\mathscr{A}$. Applying the functor ξ_f to

$h^2_{\mathscr{A}}$ yields the following commutative diagram:



where $h^1_{\xi_f \mathscr{T}_{G_2}}$ and $h^1_{\xi_f \mathscr{A}}$ are the unique homomorphisms out of $\mathscr{T}_{G_1}$ and into $\mathscr{T}_{G_2}$ and $\mathscr{A}$, respectively.

In this case, the commutativity has an interesting meaning: $h^1_{\xi_f \mathscr{T}_{G_2}}$ is the compiler translating G_1-terms into the language generated by G_2 with respect to the semantic function $h^2_{\mathscr{A}}$ induced by the morphism f. Indeed, it is easy to show that for all terms $t \in [\![p]\!]_{\mathscr{T}_{G_2}}$, holds that

$$\left(h^2_{\mathscr{A}}\right)_p(t) = \left(\xi_f h^2_{\mathscr{A}}\right)_n(t)$$

and therefore we have the following equality (where $\circ$ is the set-theoretic composition operator)

$$\xi_f h^2_{\mathscr{A}} \circ h^1_{\xi_f \mathscr{T}_{G_2}} = h^2_{\mathscr{A}} \circ h^1_{\xi_f \mathscr{T}_{G_2}} = h^1_{\xi_f \mathscr{A}}$$

which is the equation characterising a compiler [FNT91]. For instance, let $+(5,3)$ denotes the G_1-term formally defined by

$$n \to +(n,n)(n \to 5, n \to 3)$$

If we apply the compiler $h^1_{\xi_f \mathscr{T}_{G_2}}$ to $+(5,3)$, we obtain a G_2-term for which $h^2_{\mathscr{A}}$-semantics agrees with $h^1_{\xi_f \mathscr{A}}$, that is,

$$(h^1_{\xi_f \mathscr{T}_{G_2}})_n\big(+(5,3)\big) = (\mathrm{odd} + \mathrm{odd})$$

where $(\mathrm{odd} + \mathrm{odd})$ denotes the G_2-term

$$p \to (p + p)(p \to \mathrm{odd}, p \to \mathrm{odd})$$

hence

$$(h^2_{\mathscr{A}})_p\big((\mathrm{odd} + \mathrm{odd})\big) = 0 = (h^1_{\xi_f \mathscr{A}})_n\big(+(5,3)\big) \qquad \diamondsuit$$

8.4 Many-Sorted and Order-Sorted Signatures

We prove the same results developed in the previous section for the classes of algebras over a many-sorted and order-sorted signature.

Definition 8.3 (Many-Sorted Signature)**.** A *many-sorted signature* $\mathfrak{S}$ is an order-sorted signature (see Definition 3.1) such that

Many-sorted signature

(i) the subsort ordering is the discrete one; and

(ii) does not define (ad-hoc) polymorphic operators. ♢

Therefore, a *many-sorted signature* [GTWW77] (or, an *MS signature*) can be regarded as a pair $\mathfrak{S} = (S, \Sigma)$, where S is a *set of sorts* and Σ is a disjoint family of sets[1] $\Sigma_{w,s}$ such that $w \in S^*$ and $s \in S$ defined by

$$\begin{cases} \Sigma_{\varepsilon,s} = \{\, k \mid k\colon s \text{ in } \mathfrak{S} \,\} \\ \Sigma_{w,s} = \{\, f \mid f\colon w \to s \text{ in } \mathfrak{S} \,\} \qquad \text{with } w \in S^+ \end{cases}$$

We write $\sigma \in \Sigma_{w,s}$ or $\sigma\colon w \to s$ to denote any arbitrary operator of $\mathfrak{S}$, which could be either a function symbol (hence $w \neq \varepsilon$) or a constant (hence $w = \varepsilon$). If $\sigma \in \Sigma_{w,s}$ is an operator symbol, we define functions $\mathrm{ar}(\sigma) = w$ the *arity*, $\mathrm{srt}(\sigma) = s$ the *sort*, $\mathrm{rnk}(\sigma) = (w, s)$ the *rank* of σ. As in the case of context-free grammars, we suppose that $S \cap \bigcup \Sigma = \varnothing$.

Remark 8.2. In this chapter, we always consider a many-sorted signature $\mathfrak{S}$ as a pair (S, Σ). By definition, Σ is a *disjoint* family of sets, and thus there is no need to label constants and function symbols with ranks, since they can be recovered from the operator name. Moreover, it is convenient to consider a constant $k\colon s$ as a function symbols k with rank (ε, s). ♢

A *many-sorted signature morphism* $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ is a map between two many-sorted signatures $\mathfrak{S}_1 = (S_1, \Sigma_1)$ and $\mathfrak{S}_2 = (S_2, \Sigma_2)$ that preserves the structure of $\mathfrak{S}_1$ in $\mathfrak{S}_2$, into the following sense:

Many-sorted signature morphism

Definition 8.4 (Many-Sorted Signature Morphism)**.** Let $\mathfrak{S}_1 = (S_1, \Sigma_1)$ and $\mathfrak{S}_2 = (S_2, \Sigma_2)$ be two many-sorted signatures. A *many-sorted signature morphism* $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ is given by a pair of functions $f_0\colon S_1 \to S_2$ and $f_1\colon \bigcup \Sigma_1 \to \bigcup \Sigma_2$ such that $f_1(\sigma)\colon f_0^*(w) \to f_0(s)$ in $\mathfrak{S}_2$ for each operator $\sigma\colon w \to s$ in $\mathfrak{S}_1$. ♢

The identity arrow on $\mathfrak{S} = (S, \Sigma)$ is denoted by $\mathrm{id}_{\mathfrak{S}}$ and is such that $(\mathrm{id}_{\mathfrak{S}})_0$ and $(\mathrm{id}_{\mathfrak{S}})_1$ are the set identity functions on their domains, and the composition of two morphisms $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ and $g\colon \mathfrak{S}_2 \to \mathfrak{S}_3$ is obtained by defining $(g \circ f)_0 = g_0 \circ f_0$ and $(g \circ f)_1 = g_1 \circ f_1$, which is trivially a morphism from $\mathfrak{S}_1$ to $\mathfrak{S}_3$.

Proposition 8.4. *The class of all many-sorted signatures and the class of all many-sorted signature morphisms form the category MSSg.*

Remark 8.3. Order-sorted signatures have already been introduced in Chapter 3. Following Remark 8.2, we shall regard an order-sorted signature Sg as a triple $(S, \leqslant, \Sigma)$, with operators $\sigma \in \Sigma_{w,s}$ to be either constants or function symbols. ♢

Order-sorted signature morphism

We recall here Definition 6.9 of an order-sorted signature morphism presented in Chapter 6: An *order-sorted signature morphism* $f\colon Sg_1 \to Sg_2$ between two order-sorted signatures $Sg_1 = (S_1, \leqslant_1, \Sigma_1)$ and $Sg_2 = (S_2, \leqslant_2, \Sigma_2)$, is given by two components f_0 and f_1. The former component f_0 is a monotone function between S_1 and S_2 preserving subsort constraints, that is, if $s \leqslant_1 s'$ in the poset $(S_1, \leqslant_1)$, then $f_0(s) \leqslant_2 f_0(s')$ in $(S_2, \leqslant_2)$. The latter component

$$f_1 = \{\, f^1_{w,s}\colon \Sigma^1_{w,s} \to \Sigma^2_{f_0^*(w), f_0(s)} \mid w \in S_1^* \wedge s \in S_1 \,\}$$

[1] Such a condition is not necessary and may be omitted at the cost of complicating the subsequent exposition. We follow [GTWW77], and we adopt it to simplify the presentation.

is a set of functions preserving the rank of operators. Moreover, we also require the preservation of polymorphism, that is, if $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ is a polymorphic operator in Sg_1, then $f^1_{w_1,s_1}(\sigma) = f^1_{w_2,s_2}(\sigma)$.

The identity morphism id_{Sg} over an order-sorted signature Sg is defined by taking $(id_{Sg})_0$ and each component $(id_{Sg})^1_{w,s}$ of $(id_{Sg})_1$ the set-theoretic identities on their domains. The composition $g \circ f$ of two order-sorted signature morphisms $f\colon Sg_1 \to Sg_2$ and $g\colon Sg_2 \to Sg_3$ is obtained by defining $(g \circ f)_0 = g_0 \circ f_0$ and $(g \circ f)^1_{w,s} = g^1_{f^*_0(w),f_0(s)} \circ f^1_{w,s}$.

Proposition 8.5. *The class of all order-sorted signatures and the class of all order-sorted signature morphisms form the category OSSg.*

8.4.1 Algebras over a Signature

We now have all the elements to show the semantic effects induced by a many-sorted signature morphism $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$, where $\mathfrak{S}_1 = (S_1, \Sigma_1)$ and $\mathfrak{S}_2 = (S_2, \Sigma_2)$.

Definition 8.5 (Many-Sorted Algebra and Homomorphism)**.** Let $\mathfrak{S} = (S, \Sigma)$ be a many-sorted signature. A *many-sorted algebra* (resp., *homomorphism*) is just an order-sorted algebra (resp., homomorphism) over $\mathfrak{S}$ regarded as the order-sorted signature $Sg = (S, \equiv, \Sigma)$, with $\equiv$ the discrete order on S. ♢

Many-sorted algebra, Many-sorted homomorphism

As in the case of context-free grammars, we can build a mapping from the category of algebras $Alg(\mathfrak{S}_2)$ to $Alg(\mathfrak{S}_1)$, in order to employ $\mathfrak{S}_2$-algebras to provide meaning to $\mathfrak{S}_1$-terms: Let $\mathscr{A}$ be an $\mathfrak{S}_2$-algebra. We can make $\mathscr{A}$ to a $\mathfrak{S}_1$-algebra $\zeta_f\mathscr{A}$ by defining

$$(\zeta_f A)_s = A_{f_0(s)} \qquad \text{for each } s \in S_1, \text{ and}$$
$$[\![\sigma]\!]_{\zeta_f\mathscr{A}} = [\![f_1(\sigma)]\!]_{\mathscr{A}} \qquad \text{for each } \sigma \in \bigcup \Sigma_1$$

Then, given a $\mathfrak{S}_2$-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$, we define the $\mathfrak{S}_1$-homomorphism $\zeta_f h\colon \zeta_f\mathscr{A} \to \zeta_f\mathscr{B}$ such that $(\zeta_f h)_s = h_{f_0(s)}$. The very same construction can be applied to the order-sorted case, namely, if $g\colon Sg_1 \to Sg_2$ is an order-sorted signature morphism, the map $\psi_g\colon Alg(Sg_2) \to Alg(Sg_1)$ is defined analogously to ζ_f.

Proposition 8.6. *The maps $\zeta_f\colon Alg(\mathfrak{S}_2) \to Alg(\mathfrak{S}_1)$ and $\psi_g\colon Alg(Sg_2) \to Alg(Sg_1)$ induced by the signature morphisms $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ and $g\colon Sg_1 \to Sg_2$, respectively, are functors.*

Again, we can prove that *Alg* is a contravariant functor (we use the same name for the functor *Alg* both for categories of grammars and signatures):

Proposition 8.7. *Alg is a contravariant functor from the category of many-sorted signatures (order-sorted signatures) MSSg (resp., OSSg) to the category of small categories Cat, or, equivalently, $Alg\colon MSSg \to Cat^{op}$ (resp., $Alg\colon OSSg \to Cat^{op}$) is a (covariant) functor.*

Proof. The proof is analogous to the proof of Theorem 8.3. □

This last proposition leads to an equivalent of Corollary 8.1 for signature morphisms: Isomorphic signatures give rise to isomorphic categories of algebras, entailing that signature isomorphisms do not add or remove any semantic relevant information.

Proposition 8.8. *If* $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ *and* $g\colon Sg_1 \to Sg\mathfrak{S}_2$ *are isomorphism, then*

$$\zeta_{f^{-1}} \circ \zeta_f = \mathrm{id}_{Alg(\mathfrak{S}_1)} \qquad \psi_{g^{-1}} \circ \psi_g = \mathrm{id}_{Alg(Sg_1)}$$
$$\zeta_f \circ \zeta_{f^{-1}} = \mathrm{id}_{Alg(\mathfrak{S}_2)} \qquad \psi_g \circ \psi_{g^{-1}} = \mathrm{id}_{Alg(Sg_2)}$$

Therefore, $\zeta_{f^{-1}} = \zeta_f^{-1}$ *and* $\psi_{g^{-1}} = \psi_g^{-1}$*, and thus* ζ_f *and* ψ_g *are isomorphisms.*

8.5 Equivalence between MS Signatures and CF Grammars

In this section, we generalise the results of [GTWW77] by proving that the conversion of a grammar into a signature and vice versa can be extended to functors that give rise to an *adjoint equivalence* between *Grm* and *MSSg*. The major benefit of such a new development is the preservation of all the categorical properties (such as initiality, limits, colimits, etc.) from *Grm* to *MSSg*, and vice versa. A concrete example is provided at the end of the section.

The map $\Delta\colon Grm \to MSSg$ makes a grammar $G = (N, T, P)$ into a signature $\Delta G = (S_G, \Sigma_G)$, where $S_G = N$ and $\Sigma^G_{w,s} = \{A \to \beta \in P \mid A = s \wedge nt^*(\beta) = w\}$, and a grammar morphism $f\colon G_1 \to G_2$ into the signature morphism Δf such that $(\Delta f)_0 = f_0$ and $(\Delta f)_1 = f_1$.

Proposition 8.9. $\Delta\colon Grm \to MSSg$ *is a functor.*

Proof. The only non-trivial fact in the proof is checking that Δf satisfies the signature morphism condition: Let $G_1 = (N_1, T_1, P_1)$ and $G_2 = (N_2, T_2, P_2)$ be two context-free grammars, and let $f\colon G_1 \to G_2$ be a grammar morphism. If $\Delta G_1 = (S_{G_1}, \Sigma_{G_1})$ and $\Delta G_2 = (S_{G_2}, \Sigma_{G_2})$, then given $A \to \beta\colon nt_1^*(\beta) \to A$ in Σ_{G_1} holds that

$$(\Delta f)_1(A \to \beta) = f_1(A \to \beta) = f_0(A) \to \beta'$$

where $(f_0^* \circ nt_1^*)(\beta) = nt_2^*(\beta')$, and therefore

$$\begin{aligned}(\Delta f)_1(A \to \beta)\colon\ & nt_2^*(\beta') \to f_0(A)\\ & : (f_0^* \circ nt_1^*)(\beta) \to f_0(A)\\ & : ((\Delta f)_0^* \circ nt_1^*)(\beta) \to (\Delta f)_0(A)\end{aligned}$$

Hence Δf is a proper signature morphism from ΔG_1 to ΔG_2. □

Similarly, we define $\nabla\colon MSSg \to Grm$ that maps objects and arrows between the specified categories. The conversion of a signature $\mathfrak{S} = (S, \Sigma)$ to a grammar $\nabla\mathfrak{S} = (N_{\mathfrak{S}}, T_{\mathfrak{S}}, P_{\mathfrak{S}})$ is obtained by defining $N_{\mathfrak{S}} = S$, $T_{\mathfrak{S}} = \bigcup \Sigma$, and $P_{\mathfrak{S}} = \{s \to \sigma w \mid \sigma \in \Sigma_{w,s}\}$, while a signature morphism $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ is mapped to the grammar morphism ∇f such that $(\nabla f)_0 = f_0$ and $(\nabla f)_1(s \to \sigma w) = f_0(s) \to f_1(\sigma) f_0^*(w)$.

Note that the construction of $P_{\mathfrak{S}}$ follows by considering function symbols $\sigma \in \bigcup \Sigma$ in the signature as terminals in the grammar. For instance, in the context of algebraic semantics of programming languages (e.g., see [GM96]), the usual loop operator `while`: *exp com* → *com* is mapped by ∇ to the production rule *com* → `while` *exp com*. Of course, other (isomorphic) solutions are possible, such as the mapping of the previous operator to the new embellished rule *com* → `while (` *exp* `) {` *com* `}`.

Proposition 8.10. $\nabla\colon MSSg \to Grm$ *is a functor.*

Proof. We show that ∇f yields a proper grammar morphism (the remaining part of the proof is trivial): Let $\mathfrak{S}_1 = (S_1, \Sigma_1)$ and $\mathfrak{S}_2 = (S_2, \Sigma_2)$ be two many-sorted signatures, and let $f\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ be a signature morphism. Also, let $\nabla\mathfrak{S}_1 = (N_{\mathfrak{S}_1}, T_{\mathfrak{S}_1}, P_{\mathfrak{S}_1})$ and $\nabla\mathfrak{S}_2 = (N_{\mathfrak{S}_2}, T_{\mathfrak{S}_2}, P_{\mathfrak{S}_2})$, and let nt_{∇_1} and nt_{∇_2} denote the non-terminals projections on $\nabla\mathfrak{S}_1$ and $\nabla\mathfrak{S}_2$, respectively. Then,

$$(\nabla f)_1(s \to \sigma w) = f_0(s) \to f_1(\sigma) f_0^*(w)$$

for each $s \to \sigma w \in P_{\mathfrak{S}_1}$. Since the following chain of equalities holds

$$nt^*_{\nabla_2}(f_1(\sigma) f_0^*(w)) = f_0^*(w) = f_0^*(nt^*_{\nabla_1}(\sigma w)) = (f_0^* \circ nt^*_{\nabla_1})(\sigma w)$$

then ∇f is a grammar morphism from $\nabla\mathfrak{S}_1$ to $\nabla\mathfrak{S}_2$. □

As underlined in [Vis98] (and shown in the next example), Δ and ∇ are not isomorphisms. Indeed, in general, $\mathfrak{S} \neq \Delta\nabla\mathfrak{S}$ and $G \neq \nabla\Delta G$, and thus $\Delta\nabla \neq \mathrm{id}_{MSSg}$ and $\nabla\Delta \neq \mathrm{id}_{Grm}$. However, as we prove in the next two propositions, there are natural isomorphisms η and ϵ^{-1} that transform the identity functors id_{MSSg} and id_{Grm} to $\Delta\nabla$ and $\nabla\Delta$, respectively. It follows that $\mathfrak{S} \cong \Delta\nabla\mathfrak{S}$ and $G \cong \nabla\Delta G$ (where $\cong$ means *is isomorphic to*).

Example 8.2. Consider the following context-free grammar G (with terminal symbols underlined) for generating natural numbers in Peano's notation:

$$n ::= \underline{s}\, n \mid \underline{0}$$

Its conversion to a signature via Δ and way back to grammar via ∇ is

$$n ::= \underline{n \to s\, n}\, n \mid \underline{n \to 0}$$

Even though G and $\nabla\Delta G$ are different, there is a trivial grammar isomorphism $f\colon G \to \nabla\Delta G$. In particular, f is specified by taking $f_0 = \mathrm{id}_N$, where $N = \{n\}$ is the set of the unique non-terminal symbol of G, and f_1 that maps

$$\begin{array}{rcl} n \to \underline{s}\, n & \overset{f_1}{\longmapsto} & n \to \underline{n \to s\, n}\, n \\ n \to \underline{0} & \overset{f_1}{\longmapsto} & n \to \underline{n \to 0} \end{array}$$

One can check that f_1 satisfies the grammar morphism condition with respect to f_0 (that is, it preserves the non-terminals that appear in the production rules). ♢

Let $\mathfrak{S} = (S, \Sigma)$ be a many-sorted signature. We denote by $\eta_{\mathfrak{S}}\colon \mathfrak{S} \to \Delta\nabla\mathfrak{S}$ the signature morphism defined by $(\eta_{\mathfrak{S}})_0 = \mathrm{id}_S$ and $(\eta_{\mathfrak{S}})_1(\sigma) = (\sigma) \to \sigma\, \mathrm{ar}(\sigma)$. Since in the many-sorted case the arity and the rank are fully determined by the operator name (Σ is a disjoint family of sets) the previous function is well-defined (recall that $\mathrm{ar}(\sigma)$ is the arity of the operator σ).

Proposition 8.11. $\eta\colon \mathrm{id}_{MSSg} \Rightarrow \Delta\nabla$ *is a natural isomorphism.*

Proof. Let $\mathfrak{S} = (S, \Sigma)$ be a many-sorted signature and let $\sigma\colon w \to s$ in $\mathfrak{S}$. Then, $(\eta_{\mathfrak{S}})_1(\sigma) = s \to \sigma w$ has the same rank of σ. Since $(\eta_{\mathfrak{S}})_0$ is the identity on the set of sorts, $\eta_{\mathfrak{S}}$ satisfies the signature morphism condition. Moreover, it is easy to prove that each component $\eta_{\mathfrak{S}}$ is an isomorphism in *MSSg* by defining its inverse $\eta_{\mathfrak{S}}^{-1}$ as

$(\eta_{\mathfrak{S}}^{-1})_0 = \mathrm{id}_S$ and $(\eta_{\mathfrak{S}}^{-1})_1(s \to \sigma w) = \sigma$. We complete the proof by showing that the following diagram commutes for each signature morphism $f\colon \mathfrak{S} \to \mathfrak{S}'$:

$$\begin{array}{ccc} \mathfrak{S} & \xrightarrow{\ f\ } & \mathfrak{S}' \\ {\scriptstyle \eta_{\mathfrak{S}}}\downarrow & & \downarrow{\scriptstyle \eta_{\mathfrak{S}'}} \\ \Delta\nabla\mathfrak{S} & \xrightarrow[\Delta\nabla f]{} & \Delta\nabla\mathfrak{S}' \end{array}$$

The 0-th components of the morphisms in the diagram trivially commute. As regards the 1-th components, the morphisms in the diagram commute if and only if $(\eta_{\mathfrak{S}'})_1(f_1(\sigma)) = (\Delta\nabla f)_1\big((\eta_{\mathfrak{S}})_1(\sigma)\big)$ for each $\sigma \in \Sigma_{w,s}$:

$$\begin{aligned} (\eta_{\mathfrak{S}'})_1(f_1(\sigma)) &= f_0(s) \to f_1(\sigma) f_0^*(w) \\ &= (\Delta\nabla f)_1(s \to \sigma w) \\ &= (\Delta\nabla f)_1\big((\eta_{\mathfrak{S}})_1(\sigma)\big) \end{aligned}$$

and hence the thesis. □

Similarly, let $G = (N, T, P)$ be a grammar. We denote by $\epsilon_G\colon \nabla\Delta G \to G$ the grammar morphism defined by $(\epsilon_G)_0 = \mathrm{id}_N$ and $(\epsilon_G)_1\big(A \to (A, \beta)\,\mathrm{nt}^*(\beta)\big) = A \to \beta$.

Remark 8.4. Note that the productions in $P_{\Delta G}$ are formed from those in P, i.e., $P_{\Delta G} = \{\, A \to (A, \beta)\,\mathrm{nt}^*(\beta) \mid A \to \beta \in P \,\}$. Therefore, when considering a general production in $P_{\Delta G}$ derived from $A \to \beta$ in P, we write $A \to (A, \beta)\,\mathrm{nt}^*(\beta)$ (or, $A ::= \underline{A \to \beta}\,\mathrm{nt}^*(\beta)$ when considering a specific production in some example) instead of $\overline{A \to A} \to \beta\,\mathrm{nt}^*(\beta)$ to avoid any confusion. ♢

Proposition 8.12. *$\epsilon\colon \nabla\Delta \Rightarrow \mathrm{id}_{Grm}$ is a natural isomorphism.*

Proof. Let $G = (N, T, P)$ be a context-free grammar and let $A \to (A, \beta)\,\mathrm{nt}^*(\beta)$ be a production in $P_{\Delta G}$. Recall that

$$(\epsilon_G)_1(A \to (A, \beta)\,\mathrm{nt}^*(\beta)) = A \to \beta \qquad \text{and} \qquad (\epsilon_G)_0(A) = A$$

and therefore

$$((\epsilon_G)_0^* \circ \mathrm{nt}^*_{\nabla\Delta})((A, \beta)\,\mathrm{nt}^*(\beta)) = \mathrm{nt}^*(\beta)$$

where $\mathrm{nt}_{\nabla\Delta}$ is the non-terminals mapping on $\nabla\Delta G$. Thus, ϵ_G is a proper grammar morphism. Moreover, ϵ_G is an isomorphism in *Grm*: Let ϵ_G^{-1} denotes its inverse defined by

$$(\epsilon_G^{-1})_0 = \mathrm{id}_N \qquad \text{and} \qquad (\epsilon_G^{-1})_1(A \to \beta) = A \to (A, \beta)\,\mathrm{nt}^*(\beta)$$

Now one can check that $\epsilon_G \circ \epsilon_G^{-1} = \mathrm{id}_G$ and $\epsilon_G^{-1} \circ \epsilon_G = \mathrm{id}_{\nabla\Delta G}$. In order to prove the thesis, we show the commutativity of the following diagram for each grammar morphism $f\colon G \to G'$:

$$\begin{array}{ccc} \nabla\Delta G & \xrightarrow{\ \nabla\Delta f\ } & \nabla\Delta G' \\ {\scriptstyle \epsilon_G}\downarrow & & \downarrow{\scriptstyle \epsilon_{G'}} \\ G & \xrightarrow[f]{} & G' \end{array}$$

Since $\nabla\Delta f = f$ and $(\epsilon_G)_0$ and $(\epsilon_{G'})_0$ are the identity functions, we can conclude the commutativity of the 0-th components of the diagram. Moreover,

$$\begin{aligned}&(\epsilon_{G'})_1\big((\nabla\Delta f)_1(A \to (A,\beta)\,\mathrm{nt}^*(\beta))\big)\\ &\quad= (\epsilon_{G'})_1\big(f_0(A) \to f_1(A \to \beta)(f_0 \circ \mathrm{nt})^*(\beta)\big)\end{aligned}$$

for each production rule $A \to (A,\beta)\,\mathrm{nt}^*(\beta)$ in $P_{\Delta G}$. Let $G' = (N', T', P')$. Since f is a grammar morphism and $A \to \beta \in P$, then $f_0(A) \to \beta' \in P'$ for some β' where $(\mathrm{nt}')^*(\beta') = (f_0^* \circ \mathrm{nt}^*)(\beta)$. Therefore, we can continue the previous chain of equalities:

$$\begin{aligned}&= (\epsilon_{G'})_1\big(f_0(A) \to f_1(A \to \beta)(\mathrm{nt}')^*(\beta')\big)\\ &= f_0(A) \to \beta'\\ &= f_1(A \to \beta)\\ &= f_1\big((\epsilon_G)_1(A \to (A,\beta)\,\mathrm{nt}^*(\beta))\big)\end{aligned}$$

and the proof is complete. □

Example 8.3. Consider the context-free grammar G of the previous example. The grammar morphism ϵ_G transforms $\nabla\Delta G$ back to G. Indeed,

$$(\epsilon_G)_1(n \to \underline{n \to s\,n}\,n) = n \to \underline{s}\,n$$

and

$$(\epsilon_G)_1(n \to \underline{n \to 0}) = n \to \underline{0}$$ ♢

The previous results suggest to study if ∇ and Δ form an adjunction between the categories *Grm* and *MSSg*.

Theorem 8.1. *∇ is left adjoint to Δ and (ϵ,η) are the counit and the unit of the adjunction $(\nabla,\Delta,\epsilon,\eta)$.*

Proof. We need to prove the following triangle equalities:

The 0-th components of both diagrams trivially commutes. We only prove the commutativity of the 1-th components.

- For each $s \to \sigma w \in P_{\mathfrak{S}}$

$$\begin{aligned}(\epsilon_{\nabla\mathfrak{S}})_1\big((\nabla\eta_{\mathfrak{S}})_1(s \to \sigma w)\big) &= (\epsilon_{\nabla\mathfrak{S}})_1\big((\eta_{\mathfrak{S}})_0(s) \to (\eta_{\mathfrak{S}})_1(\sigma)(\eta_{\mathfrak{S}})_0^*(w)\big)\\ &= (\epsilon_{\nabla\mathfrak{S}})_1(s \to (s,\sigma w)w)\\ &= s \to \sigma w\end{aligned}$$

- For each $A \to \beta \in \Sigma^G_{\mathrm{nt}^*(\beta),A}$

$$\begin{aligned}(\Delta\epsilon_G)_1\big((\eta_{\Delta G})_1(A \to \beta)\big) &= (\Delta\epsilon_G)_1(A \to (A,\beta)\,\mathrm{nt}^*(\beta))\\ &= (\epsilon_G)_1(A \to (A,\beta)\,\mathrm{nt}^*(\beta))\\ &= A \to \beta\end{aligned}$$

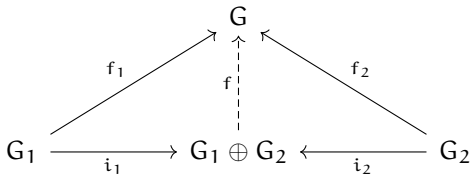
□

Since ∇ is left adjoint to Δ (Theorem 8.1) and η and ϵ are natural isomorphisms (Propositions 8.11 and 8.12), we get the following corollary.

Corollary 8.2. $(\nabla, \Delta, \epsilon, \eta)$ *is an adjoint equivalence.*

Theorem 8.1 implies that *Grm* and *MSSg* are identical except for the fact that each category may have different numbers of isomorphic copies of the same object. A consequence of this result is that we can move categorical limits between *Grm* and *MSSg*. The next example provides a definition of *coproduct* in *Grm* able to recognise the union of two context-free languages. As a consequence of Theorem 8.1, we achieve for free the same construction in *MSSg*.

Example 8.4 (Coproduct Preservation)**.** Suppose to have the following notion of categorical coproduct in *Grm*: Given two context-free grammars $G_1 = (N_1, T_1, P_1)$ and $G_2 = (N_2, T_2, P_2)$, the coproduct of G_1 and G_2 is defined by $G_1 \oplus G_2 = (N_1 \uplus N_2, T_1 \uplus T_2, P_1 \uplus P_2)$, where $\uplus$ is the disjoint union of sets. The inclusion morphism $i_k \colon G_k \to G_1 \oplus G_2$ for $k = 1, 2$ are defined by $(i_k)_0 = \mathrm{id}_{N_k}$ and $(i_k)_1 = \mathrm{id}_{P_k}$. Given two morphisms $f_1 \colon G_1 \to G$ and $f_2 \colon G_2 \to G$, where G is a context-free grammar, one can check that the unique morphism f that makes the following diagram commute



is obtained by defining $f_0(n) = (f_j)_0(n)$ with $n \in N_j$ and $f_1(A \to \beta) = (f_j)_1(A \to \beta)$ with $A \to \beta \in P_j$, for some $j = 1, 2$. The term algebra over $G_1 \oplus G_2$ carries terms both in G_1 and G_2 and recognises the (disjoint) union of the languages over G_1 and G_2. Since $(\nabla, \Delta, \epsilon, \eta)$ is an adjoint equivalence, then so is $(\Delta, \nabla, \eta^{-1}, \epsilon^{-1})$. Therefore, Δ is left adjoint to ∇ and hence it preserves colimits. Since a coproduct is a colimit, $\Delta(G_1 \oplus G_2)$ is the coproduct of ΔG_1 and ΔG_2 in *MSSg*. ♢

8.6 Adjointness between MS Signatures and OS Signatures

In this section, we show that similar results of those in Section 8.5 hold for many-sorted and order-sorted signature transformations Λ and V.

The map $\Lambda \colon \mathit{OSSg} \to \mathit{MSSg}$ converts an order-sorted signature $Sg = (S, \leqslant, \Sigma)$ to the many-sorted signature $\mathfrak{S}_{Sg} = (S_{Sg}, \Sigma_{Sg})$ defined by $S_{Sg} = S$ and $\Sigma^{Sg}_{w,s} = \{\, \sigma_{w,s} \mid \sigma \in \Sigma_{w,s} \,\}$ (such a construction is provided in [GM92]). The transformation of an order-sorted signature morphism $f \colon Sg_1 \to Sg_2$ to a many-sorted signature morphism $\Lambda f \colon \Lambda Sg_1 \to \Lambda Sg_2$ is obtained by defining $(\Lambda f)_0 = f_0$ and $(\Lambda f)_1(\sigma_{w,s}) = \left(f^1_{w,s}(\sigma)\right)_{f^*_0(w), f_0(s)}$.

Proposition 8.13. $\Lambda \colon \mathit{OSSg} \to \mathit{MSSg}$ *is a functor.*

Similarly, the map $V \colon \mathit{MSSg} \to \mathit{OSSg}$ maps the many-sorted signature $\mathfrak{S} = (S, \Sigma)$ to the order-sorted signature $Sg_{\mathfrak{S}} = (S_{\mathfrak{S}}, \leqslant_{\mathfrak{S}}, \Sigma_{\mathfrak{S}})$, where $S_{\mathfrak{S}} = S$, $\leqslant_{\mathfrak{S}}$ is the reflexive binary relation on S, and $\Sigma^{\mathfrak{S}}_{w,s} = \Sigma_{w,s}$. Moreover, if $f \colon \mathfrak{S}_1 \to \mathfrak{S}_2$ is a

many-sorted signature morphism, then $V\mathrm{f}\colon V\mathfrak{S}_1 \to V\mathfrak{S}_2$ defined by $(V\mathrm{f})_0 = \mathrm{f}_0$ and $(V\mathrm{f})^1_{w,s} = \mathrm{f}_1|_{\Sigma_{w,s}}$ is an order-sorted signature morphism.

Proposition 8.14. *$V\colon MSSg \to OSSg$ is a functor.*

As before, we can provide natural transformations

$$\varphi\colon \mathrm{id}_{MSSg} \Rightarrow \Lambda V \qquad \text{and} \qquad \vartheta\colon V\Lambda \Rightarrow \mathrm{id}_{OSSg}$$

Let $\mathfrak{S} = (S, \Sigma)$ be a many-sorted signature. Then, the $\mathfrak{S}$-component $\varphi_{\mathfrak{S}}\colon \mathfrak{S} \to \Lambda V\mathfrak{S}$ of φ is defined by taking $(\varphi_{\mathfrak{S}})_0 = \mathrm{id}_S$ and $(\varphi_{\mathfrak{S}})_1(\sigma) = \sigma_{\mathrm{ar}(\sigma),\mathrm{srt}(\sigma)}$.

Proposition 8.15. *$\varphi\colon \mathrm{id}_{MSSg} \Rightarrow \Lambda V$ is a natural isomorphism.*

Proof. The component $\varphi_{\mathfrak{S}}$ at $\mathfrak{S}$ of φ is trivially an invertible many-sorted signature morphism for each many-sorted signature $\mathfrak{S} = (S, \Sigma)$. Thus, we only prove the naturality of φ, i.e., that

$$\begin{array}{ccc} \mathfrak{S} & \xrightarrow{\quad \mathrm{f} \quad} & \mathfrak{S}' \\ {\scriptstyle \varphi_{\mathfrak{S}}}\downarrow & & \downarrow{\scriptstyle \varphi_{\mathfrak{S}'}} \\ \Lambda V\mathfrak{S} & \xrightarrow[\Lambda V\mathrm{f}]{} & \Lambda V\mathfrak{S}' \end{array}$$

commutes for each many-sorted signature morphism $\mathrm{f}\colon \mathfrak{S} \to \mathfrak{S}'$. The 0-th component of the diagram commutes because $(\Lambda V\mathrm{f})_0 = \mathrm{f}_0$ and $\varphi_{\mathfrak{S}}$ and $\varphi_{\mathfrak{S}'}$ are the right and left identities for f, respectively. As regards the 1-th component of the diagram, we have that

$$\begin{aligned} (\Lambda V\mathrm{f})_1(\sigma_{w,s}) &= \big((V\mathrm{f})_1(\sigma)\big)_{(V\mathrm{f})^*_0(w),(V\mathrm{f})_0(s)} \\ &= (\mathrm{f}_1(\sigma))_{\mathrm{f}^*_0(w),\mathrm{f}_0(s)} \\ &= (\varphi_{\mathfrak{S}'})_1(\mathrm{f}_1(\sigma)) \end{aligned}$$

and hence the thesis. □

Conversely, if $Sg = (S, \leqslant, \Sigma)$ is an order-sorted signature, the Sg-component $\vartheta_{Sg}\colon V\Lambda Sg \to Sg$ of ϑ is obtained by defining $(\vartheta_{Sg})_0 = \mathrm{id}_S$ and $(\vartheta_{Sg})^1_{w,s}(\sigma_{w,s}) = \sigma$.

Proposition 8.16. *$\vartheta\colon V\Lambda \Rightarrow \mathrm{id}_{OSSg}$ is a natural transformation.*

Proof. We prove the naturality of ϑ, i.e., that

$$\begin{array}{ccc} V\Lambda Sg & \xrightarrow{\quad V\Lambda\mathrm{f} \quad} & V\Lambda Sg' \\ {\scriptstyle \vartheta_{Sg}}\downarrow & & \downarrow{\scriptstyle \vartheta_{Sg'}} \\ Sg & \xrightarrow[\mathrm{f}]{} & Sg' \end{array}$$

commutes for each order-sorted signature morphism $f\colon Sg \to Sg'$. The 0-th component of the diagram trivially commutes. As regards the 1-th component, we have that

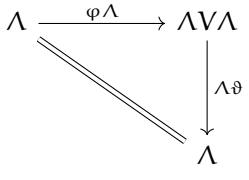
$$\begin{aligned}(\vartheta_{Sg'})^1_{f^*_0(w),f_0(s)}\big((V\Lambda f)^1_{w,s}(\sigma_{w,s})\big) &= (\vartheta_{Sg'})^1_{f^*_0(w),f_0(s)}\big((\Lambda f)_1(\sigma_{w,s})\big)\\ &= (\vartheta_{Sg'})^1_{f^*_0(w),f_0(s)}\big(f^1_{w,s}(\sigma)_{f^*_0(w),f_0(s)}\big)\\ &= f^1_{w,s}(\sigma)\\ &= f^1_{w,s}\big((\vartheta_{Sg})^1_{w,s}(\sigma_{w,s})\big)\end{aligned}$$

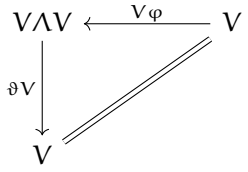
and hence the thesis. □

Again, Λ and V form an adjunction:

Theorem 8.2. *V is left adjoint to Λ and (ϑ, φ) are the counit and the unit of the adjunction $(V, \Lambda, \vartheta, \varphi)$.*

Proof. We prove the following triangle equalities (0-th component trivially commutes):





- For each $\sigma \in \Sigma^{\mathfrak{S}}_{w,s}$

$$\begin{aligned}(\vartheta_{V\mathfrak{S}})^1_{w,s}\big((V\varphi_{\mathfrak{S}})^1_{w,s}(\sigma)\big) &= (\vartheta_{V\mathfrak{S}})^1_{w,s}\big((\varphi_{\mathfrak{S}})_1(\sigma)\big)\\ &= (\vartheta_{V\mathfrak{S}})^1_{w,s}(\sigma_{w,s})\\ &= \sigma\end{aligned}$$

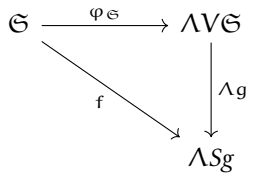
- For each $\sigma_{w,s} \in \Sigma^{Sg}_{w,s}$

$$\begin{aligned}(\Lambda\vartheta_{Sg})_1\big((\varphi_{\Lambda Sg})_1(\sigma_{w,s})\big) &= (\Lambda\vartheta_{Sg})_1((\sigma_{w,s})_{w,s})\\ &= \big((\vartheta_{Sg})^1_{w,s}(\sigma_{w,s})\big)_{w,s}\\ &= \sigma_{w,s}\end{aligned}$$

□

The results in this section can be rephrased in terms of *free constructions*. Indeed, the order-sorted signature $V\mathfrak{S}$ is actually a free object on $\mathfrak{S}$ (together with the morphism $\varphi_{\mathfrak{S}}\colon \mathfrak{S} \to \Lambda V\mathfrak{S}$). In this context, the functor $\Lambda\colon$ *OSSg* $\to$ *MSSg* acts as a *forgetful functor* which forgets the ordering between sorts of the signature, whereas the *free functor* $V\colon$ *MSSg* $\to$ *OSSg* adds the loosest ordering on the set of sorts of a many-sorted signature (i.e., the smallest reflexive relation). Therefore, it follows that, given a many-sorted signature $\mathfrak{S}$, for each order-sorted signature *Sg*

and (many-sorted) morphism $f\colon \mathfrak{S} \to \Lambda Sg$ there is a unique (order-sorted) morphism $g\colon V\mathfrak{S} \to Sg$ such that the following diagram commutes:



Example 8.5. In this example, we denote by $\equiv_X$ the smallest reflexive relation on a given set X, that is, $\equiv_X = \{(x, x) \mid x \in X\}$. Let $\mathfrak{S} = (S_1, \Sigma_1)$ be the many-sorted signature with only two sorts a and b, and two operators $k : a$ and $l : b$. The free order-sorted signature on $\mathfrak{S}$ is $V\mathfrak{S} = (S_1, \equiv_{S_1}, \Sigma_1)$. Now, let $Sg = (S_2, \leqslant_2, \Sigma_2)$ be the following order-sorted signature

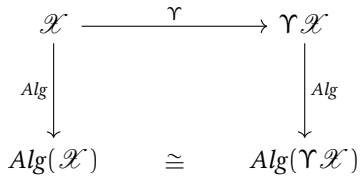
$$k : a \qquad l : b \qquad i : c \qquad \text{where} \qquad \leqslant_2 = \{(a, b), (a, c), (b, c)\} \cup \equiv_{S_2}$$

The forgetful functor Λ maps Sg to $\Lambda Sg = (S_2, \widehat{\Sigma}_2)$, where $\widehat{\Sigma}^2_{w,s} = \{k_{\varepsilon,a}, l_{\varepsilon,b}, i_{\varepsilon,c}\}$. Suppose that $f\colon \mathfrak{S} \to \Lambda Sg$ is the many-sorted morphism which its first component f_0 acts as the inclusion function from S_1 to S_2, and $f_1(k) = k_{\varepsilon,a}$ and $f_1(l) = l_{\varepsilon,b}$. The unique morphism $g\colon V\mathfrak{S} \to Sg$ which makes the previous diagram commute is the one who mimics the behavior of f, namely $g_0(s) = s$ for each $s \in S_1$ and $g^1_{w,s}(\sigma) = \sigma$ for each $\sigma \in \Sigma^1_{w,s}$. ♢

8.7 Semantic Equivalence

In this section, we show that the provided syntactical transformations between context-free grammars and many-sorted signatures (Section 8.5) and between many-sorted and order-sorted signatures (Section 8.6) give rise to equivalent categories of algebras over the transformed objects.

More specifically, if $\Upsilon \in \{\Delta, \nabla, \Lambda, V\}$ is a syntactical transformation and $\mathscr{X}$ is a language specification in the domain of Υ, then we prove that $Alg(\mathscr{X})$ and $Alg(\Upsilon\mathscr{X})$ are equivalent, namely



Some of these equivalences are presented as *isomorphisms of categories*. It is well-known that an isomorphism of categories is a strong notion of categorical equivalence where functors compose to the identity.

8.7.1 Context-Free Grammars and Many-Sorted Signatures

As mentioned in the introduction, [GTWW77] proves an equivalence between the many-sorted term ΔG-algebra $\mathscr{T}_{\Delta G}$ and the initial algebra $\mathscr{T}_G$ over each grammar G. We now extend this result to the whole categories of algebras $Alg(G)$ and $Alg(\acute{}G)$.

Let $\mathscr{A}$ be a G-algebra, and recall that the conversion of G to many-sorted signature is denoted by $\Delta G = (S_G, \Sigma_G)$. Then, we map $\mathscr{A}$ to the many-sorted ΔG-algebra $\mathscr{A}^{\uparrow}$ such that $A^{\uparrow}_N = A_s$ for each $N \in S_G$ and $[\![C \to \delta]\!]_{\mathscr{A}^{\uparrow}} = [\![C \to \delta]\!]_{\mathscr{A}}$ for each $C \to \delta \in \bigcup \Sigma_G$ (operators in ΔG are productions in G). Furthermore, given a G-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$, we define the ΔG-homomorphism $h^{\uparrow}\colon \mathscr{A}^{\uparrow} \to \mathscr{B}^{\uparrow}$ such that $h^{\uparrow}_N = h_N$.

Conversely, let $\mathscr{A}$ be a ΔG-algebra. Then, we define the inverse construction that maps $\mathscr{A}$ to the G-algebra $\mathscr{A}^{\downarrow}$ such that $A^{\downarrow}_N = A_N$ for each non-terminal N and $[\![C \to \delta]\!]_{\mathscr{A}^{\downarrow}} = [\![C \to \delta]\!]_{\mathscr{A}}$ for each production $C \to \delta$. Moreover, if $h\colon \mathscr{A} \to \mathscr{B}$ is a ΔG-homomorphism, then $h^{\downarrow}\colon \mathscr{A}^{\downarrow} \to \mathscr{B}^{\downarrow}$ such that $h^{\downarrow}_s = h_s$ is a proper G-homomorphism.

Theorem 8.3. *The inverse of $(\_)^{\uparrow}$ is $(\_)^{\downarrow}$, therefore they form an isomorphism of categories between $Alg(G)$ and $Alg(\Delta G)$ for each context-free grammar G.*

Since an isomorphism of categories is a strict notion of categorical equivalence, it preserves the initial objects, and thus, by applying $(\_)^{\uparrow}$ and $(\_)^{\downarrow}$ to the initial algebras, we have the exactly result of [GTWW77], that is, $\mathscr{T}^{\uparrow}_G = \mathscr{T}_{\Delta G}$ and $\mathscr{T}^{\downarrow}_{\Delta G} = \mathscr{T}_G$.

In a similar manner, given a many-sorted signature $\mathfrak{S}$, we can extend the equivalence between the initial algebras $\mathscr{T}_{\mathfrak{S}}$ and $\mathscr{T}_{\nabla\mathfrak{S}}$ to their whole categories of algebras $Alg(\mathfrak{S})$ and $Alg(\nabla\mathfrak{S})$.

Let $\mathscr{A}$ be a many-sorted $\mathfrak{S}$-algebra. We denote by $\nabla\mathfrak{S} = (N_{\mathfrak{S}}, T_{\mathfrak{S}}, P_{\mathfrak{S}})$ the context-free grammar obtained by converting the signature $\mathfrak{S}$. We define the $\nabla\mathfrak{S}$-algebra ${}^{\uparrow}\mathscr{A}$, where ${}^{\uparrow}A_s = A_s$ for each $s \in N_{\mathfrak{S}}$ and $[\![s \to \sigma w]\!]_{{}^{\uparrow}\mathscr{A}} = [\![\sigma]\!]_{\mathscr{A}}$ for each $s \to \sigma w \in P_{\mathfrak{S}}$. The conversion of a $\mathfrak{S}$-homomorphism $h\colon \mathscr{A} \to \mathscr{B}$ to a $\nabla\mathfrak{S}$-homomorphism ${}^{\uparrow}h\colon {}^{\uparrow}\mathscr{A} \to {}^{\uparrow}\mathscr{B}$ is analogous to the previous case.

On the contrary, if $\mathscr{A}$ is a $\nabla\mathfrak{S}$-algebra and $h\colon \mathscr{A} \to \mathscr{B}$ is a $\nabla\mathfrak{S}$-homomorphism, we can obtain a many-sorted $\mathfrak{S}$-algebra ${}^{\downarrow}\mathscr{A}$ and an $\mathfrak{S}$-homomorphism ${}^{\downarrow}h\colon {}^{\downarrow}\mathscr{A} \to {}^{\downarrow}\mathscr{B}$ by simply inverting the previous construction.

Theorem 8.4. *The inverse of ${}^{\uparrow}(\_)$ is ${}^{\downarrow}(\_)$, therefore they form an isomorphism of categories between $Alg(\mathfrak{S})$ and $Alg(\nabla\mathfrak{S})$ for each many-sorted signature $\mathfrak{S}$.*

Again, the result of [GTWW77] is a special case of this last theorem by noting that ${}^{\uparrow}\mathscr{T}_{\mathfrak{S}} = \mathscr{T}_{\nabla\mathfrak{S}}$ and ${}^{\downarrow}\mathscr{T}_{\nabla\mathfrak{S}} = \mathscr{T}_{\mathfrak{S}}$.

Example 8.6 (Example 8.4 Continued)**.** In the Example 8.4, we have shown how to preserve categorical constructions between *Grm* and *MSSg*. Theorems 8.3 and 8.4 can be applied on the top of Theorem 8.1 to ensure the semantic equivalence of the achieved constructions. For instance, if the $(G_1 \oplus G_2)$-algebra $\mathscr{A}$ provides the semantics of the disjoint union of languages over G_1 and G_2, then $\mathscr{A}^{\uparrow}$ provides the equivalent semantics in the category $Alg(\Delta(G_1 \oplus G_2))$, as a consequence of Theorem 8.3. ♢

8.7.2 Many-Sorted and Order-Sorted Signatures

The forgetful functor Λ transforms an order-sorted signature *Sg* to the many-sorted signature ΛSg by forgetting the ordering on the sorts. In [GM92], the authors prove the categorical equivalence between $Alg(Sg)$ and $Alg(\Lambda Sg)$. We now extend such a result to its left adjoint V.

Let $\mathscr{A}$ be a many-sorted $\mathfrak{S}$-algebra and let $V\mathfrak{S} = (S_{\mathfrak{S}}, \leqslant_{\mathfrak{S}}, \Sigma_{\mathfrak{S}})$. We define the order-sorted $V\mathfrak{S}$-algebra $\mathscr{A}_\uparrow$ such that $(A_\uparrow)_s = A_s$ for each $s \in S_{\mathfrak{S}}$ and $[\![\sigma\colon w \to s]\!]_{\mathscr{A}_\uparrow} = [\![\sigma]\!]_{\mathscr{A}}$ for each $\sigma \in \Sigma^{\mathfrak{S}}_{w,s}$. Moreover, if $h\colon \mathscr{A} \to \mathscr{B}$ is an $\mathfrak{S}$-homomorphism, then $h_\uparrow\colon \mathscr{A}_\uparrow \to \mathscr{B}_\uparrow$ is the $\Lambda\mathfrak{S}$-homomorphism defined by $(h_\uparrow)_s = h_s$. Furthermore, we denote by $(\_)_\downarrow$ the inverse functor that maps $\Lambda\mathfrak{S}$-algebras and $\Lambda\mathfrak{S}$-homomorphism to the category $Alg(\mathfrak{S})$.

Theorem 8.5. *The inverse of $(\_)_\uparrow$ is $(\_)_\downarrow$, therefore they form an isomorphism of categories between $Alg(\mathfrak{S})$ and $Alg(V\mathfrak{S})$ for each many-sorted signature $\mathfrak{S}$.*

8.8 Some Remarks

An obvious but important consequence of the underlying categorical model is the compositional nature of the proved results. Indeed, we can get an adjointness between the category of grammars *Grm* and the category of order-sorted signatures *OSSg* by simply composing $V\Delta$ and $\nabla\Lambda$. The algebraic counterpart of the same observation allows us to claim that the composition of the functors $(\_)_\downarrow \circ (\_)^\uparrow$ gives rise to an isomorphism between $Alg(G)$ and $Alg(V\Delta G)$ (and, of course, the dual result holds).

It may be worth to point out what is lost when moving (via Λ) from *OSSg* to *Grm* or *MSSg*, and why we can still get an equivalence between the categories of algebras. The functor Λ forgets the ordering between sorts. Therefore, two order-sorted signatures which only differ on the order relation are collapsed to the same many-sorted signature when Λ is applied. Moreover, it produces different (sorted) copies of the same polymorphic operator (e.g., $f\colon w_1 \to s_1$ and $f\colon w_2 \to s_2$ are mapped to $f_{w_1,s_1} : w_1 \to s_1$ and $f_{w_2,s_2} : w_2 \to s_2$). Thus, polymorphism is neutralised in this process. For these reasons, Λ yields an adjointness rather than an equivalence. However, note that no information has been lost from an algebraic perspective. Indeed we can set $[\![f_{w_1,s_1}]\!] = [\![f\colon w_1 \to s_1]\!]$ and $[\![f_{w_2,s_2}]\!] = [\![f\colon w_2 \to s_2]\!]$, obtaining a semantic equivalence between the categories of algebras.

CHAPTER 9

Conclusions

The thesis addresses the problem of providing a formal semantics to the combination of programming languages, the so-called *multi-languages*. We have introduced an algebraic framework able to model this new paradigm, and we have constructively shown how to attain a multi-language specification by only stipulate (i) how the syntactic categories of the base languages have to be combined and (ii) how the values may flow from one language to the other. We have proved the suitability of the framework to yield the (algebraic and categorical) semantics of multi-language terms, while preserving the semantics of single-languages, formalised as order-sorted specifications. We have also proved that combining languages is a closed operation, that is, 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.

We have then addressed the problem of equational deduction in a multi-language context, from both a set-theoretic and categorical perspective. We have lifted the order-sorted equational logic of [GM92] to the previously defined framework of multi-languages, and we have proved the soundness and the completeness of the resulting deduction system. The main benefit of the theoretical development pursued in the thesis is a solid mathematical foundation for reasoning about equalities in a multi-language context.

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

Finally, we have provided a categorical model of three different syntax formalisms (context-free grammars, many-sorted signatures, and order-sorted signatures). We have shown how the extension to functors of already existing syntactical transformations gives rise to adjoint constructions able to preserve the abstract syntax of the

generated terms. Then, we have proved that the categories of algebras over the objects in these formalisms are categorically equivalent up to the provided transformations.

9.1 Related Works

In this section we mention research works directly related to the content of this thesis, divided according to subtopics covered.

9.1.1 Multi-Language Formalisation and Implementation

Cross-language interoperability is a well-researched area from both theoretical and practical points of view. The most related work to our approach is undoubtedly [MF09], which provides operational semantics to a combined language obtained by embedding a Scheme-like language into an ML-like language. This 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. However, we have decoupled semantics of boundary functions from syntactical conversion operators. Such an approach allowed us to investigate several variants of multi-language constructions.

Other theoretical works on multi-languages [Ram11, Ben05, Gra08, WNL+10, PPDA17] take a similar line and combine typed and untyped languages (Lua and ML [Ram11], Java and PLT Scheme [Gra08], or Assembly and a typed functional language [PPDA17]), focusing on typing issues and values exchanging techniques. Instead of focusing on a specific issue, we have adopted a more 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 number of works have focused on multi-language runtime mechanisms: [GS01] 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, etc.) which are then translated to IL. [GSS+18] proposes a virtual machine that can execute the composition of dynamically typed programming languages (Ruby and JavaScript) and statically typed one (C). [BBT15, BBT13] describes a multi-language runtime mechanism achieved by combining single-language interpreters of (different versions of) Python and Prolog.

9.1.2 Equational Logic in Universal Algebra

Equational logic is a very wide field that have been studied since the 80s. Here, we mention only those works that are related to this thesis for extending equational logic to a new class of algebras, and we redirect the reader to [HO80, Tay79] for surveys about the general topic. In the field of universal algebra, equational logic emerges from the works of [Bir35, Tar68]. Ordinary (one-sorted) algebras have been extended to *many-sorted algebras* by [Hig63] by adding sorts to the operators in the signature. The naive extension of equational logic to the many-sorted world fails to be sound due to the possible simultaneous presence of empty and non-empty carrier interpretation sets. [GM82] solves this problem by explicitly quantifying the variables involved into equations, and provide a sound and complete equational deduction system. Finally, [GM92] extends many-sorted algebras to order-sorted algebras by

adding a subsort relation on the set of sorts of the signature, and they also lift the equational logic to the order-sorted world.

9.1.3 Analysis of Multi-Language Programs

For what concerns the verification of multi-language code, a few works concentrated on analysis-related aspects in a multi-language scenario. In the Java Native Interface context, [TM07] proposes a specification language which extends the Java Virtual Machine Language with primitives that approximate C code that cannot be compiled into Java. [LT14] introduces Pungi, a system that transforms Python/C interface code to affine programs with the aim of correctly handling Python's heap when it interoperates with C++. Similarly, CPyChecker [Mal] is static checker analysing C extension modules written for Python. It is able to flag null pointer dereferences and reference counting bugs. Finally, some researches focus on detecting type errors in multi-language code [LT09, FF08].

9.1.4 Equivalence of Syntax Formalisms

The work most directly related to ours is [Vis98], where the correspondence between context-free grammars and algebraic signatures is studied, both in the first-order and high-order setting. In particular, the author provides a proof (Proposition 2.15) of the isomorphism between the term algebra over a grammar and over its conversion to a many-sorted signature, and vice versa. Our work generalises these results (just for the first-order case, but as well including order-sorted specifications) into a categorical framework. Syntax formalisms and syntactical transformations are modeled by categories and functors between them, respectively. Then, the relationship between such formalisms is given by the nature of these functors (Corollary 8.2 and Theorem 8.2). Such an abstract perspective allowed us to extend the aforementioned isomorphisms to the whole category of algebras.

To the best of our knowledge, the first paper presenting the syntactical transformations studied in this work is [GTWW77]. More specifically, the authors provided the definitions of $\Delta\mathsf{G}$ and $\nabla\mathfrak{S}$ with the properties described in [Vis98]. A similar approach is taken in [GM92], where the definition of Λ is given, along with the proof (Theorem 4.2) that $Alg(Sg)$ is equivalent to $Alg(\Lambda Sg)$.

9.2 Future Works

We have investigated the combination of base languages defined through order-sorted specifications. Although we have made progress in generalising the multi-language construction in a way that is not dependent on the combined languages, there is still great deal to be done. For instance, current type specification is limited. We plan to replay the categorical methods illustrated in Chapter 6 for more complex type systems including *function types*, *algebraic and recursive data types*, *T-types*, and *session types* [Cro93, CDGP09]. Future research should investigate membership equational logic [BJM00] as an alternative to order-sorted theories. Specifications in this logic generalises order-sorted theories by adding membership constraints.

A second line of research will investigate *operational semantics for multi-languages*. *Rewriting logic* seems the most reasonable approach to link the algebraic semantics presented in this thesis to the operational one [Mes92, Mes20]. In particular, rewriting

logic is obtained from equational logic by removing the simmetry rule. Our goal is to perform multi-language rewriting modulo E by partitioning axioms into a set R of rewriting rules and a set E of multi-language equations. This research would be particularly useful in order to move towards an implementation of multi-languages or automatic tools able to combine the rewriting specifications of the base languages. Moreover, possible implementations may exploit the K framework [RS10], an executable semantic framework based on rewriting logic. Operational semantics for multi-languages may also benefit from the modular approach to structural operational semantics [Mos04]. Indeed, modularity is a crucial property when extending a language with new constructs, a similar scenario that occurs when two languages are combined.

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 equations 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 [MR77]. We are currently working on such a system, with a view to giving a sound and complete semantics. There are interesting questions concerning the appropriate category theory, and the answers will have connections to work such as [PV07]. And further, since equational theories give rise to *free algebra monads* [Plo06], further studies should investigate the possibility of extending/generalizing the results in this thesis to the notion of *monad* [Mog91].

As regards the analysis of multi-language code, future research should consider the *asymmetrical* lifting of a single-language analysis to the whole multi-language. In Chapter 7, we assumed the existence of two algebras, $\mathscr{A}_1$ and $\mathscr{A}_2$, providing the abstract semantics of the underlying languages. Then, the abstract semantics of boundary functions defines the flow of abstract values during the abstract computations. Even though our framework is general enough to allow such algebras to be different (e.g., $\mathscr{A}_1$ may define a sign semantics whereas $\mathscr{A}_2$ provides an interval analysis), we did not discuss the case in which there exists only one analysis. It may be fruitful to investigate this asymmetrical situation, for instance in the case where one of the underlying languages cannot alter the values flowing from the other (see the *lump embedding* construction of [MF09]).

Further research on the equivalence of language syntax formalisms concerns refinements of the syntactical transformations between the formalisms in order to preserve specific properties of the concrete syntax. Among them, *polymorphism* seems the most interesting. Unfortunately, the composition of functors $\mathsf{V}\Delta$ and $\nabla\Lambda$ yields non-polymorphic set of operators. Another future work goes in the direction of providing syntactical transformation from *Grm* to *OSSg* that yields only *regular* (see [GM92] for *regularity* definition) order-sorted signatures. Then, studying the adjoint of such a transformation could provide an interesting notion of regularity in the category of grammars that may be employed to weaken the standard notion of *ambiguity*.

Finally, we plan to instantiate the presented framework in order to model real-world multi-languages, such as Rust (or its core, e.g.RustBelt [JJKD18], which formalises Rust distinctive features), that can be thought as the combination of *Safe* Rust (which ensures memory safety), and *Unsafe* Rust (which allows programmers to

perform unsafe operations), or the integration of Assembly with C code, which has already been the target of static analysis-related research [Che20, MMS08].

Bibliography

[AB11] Amal Ahmed and Matthias Blume. An equivalence-preserving CPS translation via multi-language semantics. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming, ICFP 2011, Tokyo, Japan, September 19-21, 2011*, pages 431–444. ACM, 2011.

[AM19] Vincenzo Arceri and Isabella Mastroeni. Static program analysis for string manipulation languages. *Electronic Proceedings in Theoretical Computer Science*, 299:19–33, 2019.

[AMS20] Gianluca Amato, Maria Chiara Meo, and Francesca Scozzari. On collecting semantics for program analysis. *Theor. Comput. Sci.*, 823:1–25, 2020.

[BBT13] Edd Barrett, Carl Friedrich Bolz, and Laurence Tratt. Unipycation: A case study in cross-language tracing. In *Proceedings of the 7th ACM workshop on Virtual machines and intermediate languages*, pages 31–40, 2013.

[BBT15] Edd Barrett, Carl Friedrich Bolz, and Laurence Tratt. Approaches to interpreter composition. *Comput. Lang. Syst. Struct.*, 44:199–217, 2015.

[BCM20a] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. Equational logic and categorical semantics for multi-languages. In Patricia Johann, editor, *Proceedings of the Thirty-Sixth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2020, , June 2-6, 2020*, volume 0 of *Electronic Notes in Theoretical Computer Science*, pages 0–0. Elsevier, 2020.

[BCM20b] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. Equational logic and set-theoretic models for multi-languages. In Gennaro Cordasco, Luisa Gargano, and Adele A. Rescigno, editors, *Proceedings of the 21st Italian Conference on Theoretical Computer Science, Ischia, Italy, September 14-16, 2020*, volume 2756 of *CEUR Workshop Proceedings*, pages 236–249. CEUR-WS.org, 2020.

[BCM20c] Samuele Buro, Roy L. Crole, and Isabella Mastroeni. On multi-language abstraction — Towards a static analysis of multi-language programs. In David Pichardie and Mihaela Sighireanu, editors, *Static Analysis - 27th International Symposium, SAS 2020, Virtual Event, November 18-20, 2020, Proceedings*, volume 12389 of *Lecture Notes in Computer Science*, pages 310–332. Springer, 2020.

[Ben05] Nick Benton. Embedded interpreters. *J. Funct. Program.*, 15(4):503–542, 2005.

[BG15] Nikolaj Bjørner and Arie Gurfinkel. Property directed polyhedral abstraction. In Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, volume 8931 of *Lecture Notes in Computer Science*, pages 263–281. Springer, 2015.

[Bir35] Garrett Birkhoff. On the structure of abstract algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 31(4):433–454, 1935.

[BJM00] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000.

[BL70] Garrett Birkhoff and John D. Lipson. Heterogeneous algebras. *Journal of Combinatorial Theory*, 8(1):115–133, 1970.

[BM19a] Samuele Buro and Isabella Mastroeni. On the multi-language construction. In Luís Caires, editor, *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 293–321. Springer, 2019.

[BM19b] Samuele Buro and Isabella Mastroeni. On the semantic equivalence of language syntax formalisms. In Alessandra Cherubini, Nicoletta Sabadini, and Simone Tini, editors, *Proceedings of the 20th Italian Conference on Theoretical Computer Science, ICTCS 2019, Como, Italy, September 9-11, 2019*, volume 2504 of *CEUR Workshop Proceedings*, pages 34–51. CEUR-WS.org, 2019.

[BM20] Samuele Buro and Isabella Mastroeni. On the semantic equivalence of language syntax formalisms. *Theoretical Computer Science*, 840:234–248, 2020.

[BW95] Michael Barr and Charles Wells. *Category theory for computing science (2. ed.)*. Prentice Hall international series in computer science. Prentice Hall, 1995.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977.

[CC81] Paul Moritz Cohn and Paul Moritz Cohn. *Universal algebra*, volume 159. Reidel Dordrecht, 1981.

[CC92] Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.

[CDE+19] Manuel Clavel, Francisco Durán, Steven Eker, Santiago Escobar, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn Talcott. *Maude manual (version 3.0).* SRI International, 2019.

[CDGP09] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, September 7-9, 2009, Coimbra, Portugal*, pages 219–230. ACM, 2009.

[CFA+05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography.* CRC press, 2005.

[CG06] Carlos Caleiro and Ricardo Gonçalves. On the algebraization of many-sorted logics. In José Luiz Fiadeiro and Pierre-Yves Schobbens, editors, *Recent Trends in Algebraic Development Techniques, 18th International Workshop, WADT 2006, La Roche en Ardenne, Belgium, June 1-3, 2006, Revised Selected Papers*, volume 4409 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2006.

[CGR19] Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. A^2i: abstract^2 interpretation. *Proc. ACM Program. Lang.*, 3(POPL):42:1–42:31, 2019.

[CH78] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski, editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, pages 84–96. ACM Press, 1978.

[Che20] Marc Chevalier. *Proving the security of software-intensive embedded systems by abstract interpretation.* Theses, ENS Paris ; PSL University, November 2020.

[Chi13] David Chisnall. The challenge of cross-language interoperability. *Commun. ACM*, 56(12):50–56, 2013.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Trans. Inf. Theory*, 2(3):113–124, 1956.

[CK92] Chen C. Chang and H. Jerome Keisler. *Model theory, Third Edition*, volume 73 of *Studies in logic and the foundations of mathematics.* North-Holland, 1992.

[Cou02] Patrick Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.

[CP13] G Campbell and Patroklos P Papapetrou. *SonarQube in action.* Manning Publications Co., 2013.

[Cro93] Roy L. Crole. *Categories for Types.* Cambridge mathematical textbooks. Cambridge University Press, 1993.

[Cro12] Roy L. Crole. Alpha equivalence equalities. *Theor. Comput. Sci.*, 433:1–19, 2012.

[DW09] Klaus Denecke and Shelly L. Wismath. *Universal Algebra and Coalgebra.* World Scientific, 2009.

[Dyb09] R. Kent Dybvig. *The Scheme Programming Language, Fourth Edition.* MIT Press, 2009.

[Ear83] Jay Earley. An efficient context-free parsing algorithm (reprint). *Commun. ACM*, 26(1):57–61, 1983.

[FF08] Michael Furr and Jeffrey S. Foster. Checking type safety of foreign function calls. *ACM Trans. Program. Lang. Syst.*, 30(4):18:1–18:63, 2008.

[FNT91] Yoshihiko Futamura, Kenroku Nogi, and Akihiko Takano. Essence of generalized partial computation. *Theor. Comput. Sci.*, 90(1):61–79, 1991.

[FPT99] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202. IEEE Computer Society, 1999.

[GM82] Joseph A. Goguen and José Meseguer. Completeness of many-sorted equational logic. *ACM SIGPLAN Notices*, 17(1):9–17, 1982.

[GM92] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992.

[GM96] Joseph A. Goguen and Grant Malcolm. *Algebraic semantics of imperative programs.* Foundations of computing series. MIT Press, 1996.

[GR97] Roberto Giacobazzi and Francesco Ranzato. Completeness in abstract interpretation: A domain perspective. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings*, volume 1349 of *Lecture Notes in Computer Science*, pages 231–245. Springer, 1997.

[Gra08] Kathryn E. Gray. Safe cross-language inheritance. In Jan Vitek, editor, *ECOOP 2008 - Object-Oriented Programming, 22nd European Conference, Paphos, Cyprus, July 7-11, 2008, Proceedings*, volume 5142 of *Lecture Notes in Computer Science*, pages 52–75. Springer, 2008.

[GRS00] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000.

[GS01] Andrew D. Gordon and Don Syme. Typing a multi-language intermediate code. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 248–260. ACM, 2001.

[GSS+18] Matthias Grimmer, Roland Schatz, Chris Seaton, Thomas Würthinger, and Mikel Luján. Cross-language interoperability in a multi-language runtime. *ACM Trans. Program. Lang. Syst.*, 40(2):8:1–8:43, 2018.

[GTWW77] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24(1):68–95, 1977.

[Hig63] Philip J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27(1-2):115–132, 1963.

[HO80] Gérard Huet and Derek C Oppen. Equations and rewrite rules: A survey. *Formal Language Theory*, pages 349–405, 1980.

[HR76] William S. Hatcher and Teodor Rus. Context-free algebras. *Journal of Cybernetics*, 6(1-2):65–77, 1976.

[IdFF96] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua-an extensible extension language. *Softw. Pract. Exp.*, 26(6):635–652, 1996.

[J+02] Peter T Johnstone et al. *Sketches of an Elephant: A Topos Theory Compendium: Volume 2*, volume 2. Oxford University Press, 2002.

[JBW+10] Josh Juneau, Jim Baker, Frank Wierzbicki, Leo Soto, and Victor Ng. *The definitive guide to Jython: Python for the Java platform.* Apress, Berkely, CA, USA, 1st edition, 2010.

[Jet] JetBrains. Calling java code from kotlin. https://kotlinlang.org/docs/reference/java-interop.html.

[JJKD18] Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt: securing the foundations of the rust programming language. *Proc. ACM Program. Lang.*, 2(POPL):66:1–66:34, 2018.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Math. Syst. Theory*, 2(2):127–145, 1968.

[KO11] Jonathan Kochems and C.-H. Luke Ong. Improved functional flow and reachability analyses using indexed linear tree grammars. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPIcs*, pages 187–202. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011.

[Lee02] Lillian Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, 2002.

[Lia99] Sheng Liang. *Java Native Interface: Programmer's guide and reference.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[LT09] Siliang Li and Gang Tan. Finding bugs in exceptional situations of jni programs. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 442–452, 2009.

[LT14] Siliang Li and Gang Tan. Finding reference-counting errors in python/c programs with affine analysis. In Richard E. Jones, editor, *ECOOP 2014 - Object-Oriented Programming - 28th European Conference, Uppsala, Sweden, July 28 - August 1, 2014. Proceedings*, volume 8586 of *Lecture Notes in Computer Science*, pages 80–104. Springer, 2014.

[Mal] David Malcolm. Usage example: A static analysis tool for cpython extension code. https://gcc-python-plugin.readthedocs.io/en/latest/cpychecker.html.

[McC62a] John McCarthy. Towards a mathematical science of computation. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*, pages 21–28. North-Holland, 1962.

[McC62b] John McCarthy. Towards a mathematical science of computation. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*, pages 21–28. North-Holland, 1962.

[Mes92] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[Mes20] José Meseguer. Generalized rewrite theories, coherence completion, and symbolic methods. *J. Log. Algebraic Methods Program.*, 110, 2020.

[MF09] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *ACM Trans. Program. Lang. Syst.*, 31(3):12:1–12:44, 2009.

[Min17] Antoine Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017.

[ML13] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

[MM96] Narciso Martí-Oliet and José Meseguer. Inclusions and subtypes I: first-order case. *J. Log. Comput.*, 6(3):409–438, 1996.

[MMS08] Stefan Maus, Michal Moskal, and Wolfram Schulte. Vx86: x86 assembler simulated in C powered by automated theorem proving. In José Meseguer and Grigore Rosu, editors, *Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008, Urbana, IL, USA, July 28-31, 2008, Proceedings*, volume 5140 of *Lecture Notes in Computer Science*, pages 284–298. Springer, 2008.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

[Mos04] Peter D. Mosses. Modular structural operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:195–228, 2004.

[MP17] Isabella Mastroeni and Michele Pasqua. Hyperhierarchy of semantics - A formal framework for hyperproperties verification. In Francesco Ranzato, editor, *Static Analysis - 24th International Symposium, SAS 2017, New York, NY, USA, August 30 - September 1, 2017, Proceedings*, volume 10422 of *Lecture Notes in Computer Science*, pages 232–252. Springer, 2017.

[MR77] M. Makkai and G.E. Reyes. *First Order Categorical Logic*. Lecture Notes in Mathematics. Springer Verlag, 1977.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *Definition of standard ML*. MIT Press, 1990.

[Oraa] Oracle. Jni types and data structures. https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/types.html.

[Orab] Oracle. Nashorn user's guide. https://docs.oracle.com/en/java/javase/14/nashorn/introduction.html.

[OSZ12] Peter-Michael Osera, Vilhelm Sjöberg, and Steve Zdancewic. Dependent interoperability. In Koen Claessen and Nikhil Swamy, editors, *Proceedings of the sixth workshop on Programming Languages meets Program Verification, PLPV 2012, Philadelphia, PA, USA, January 24, 2012*, pages 3–14. ACM, 2012.

[PA14] James T. Perconti and Amal Ahmed. Verifying an open compiler using multi-language semantics. In Zhong Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8410 of *Lecture Notes in Computer Science*, pages 128–148. Springer, 2014.

[Pas19] Michele Pasqua. *Hyper Static Analysis of Programs – An Abstract Interpretation-Based Framework for Hyperproperties Verification*. PhD thesis, University of Verona, 2019.

[Pie91] Benjamin C. Pierce. *Basic category theory for computer scientists*. Foundations of computing. MIT Press, 1991.

[Pit00] 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.

[Pit16] Andrew M. Pitts. Nominal techniques. *ACM SIGLOG News*, 3(1):57–72, 2016.

[Plo06] Gordon Plotkin. Some varieties of equational logic. In *Algebra, Meaning, and Computation*, pages 150–156. Springer, 2006.

[Pow85] Wayne B. Powell. *Ordered Algebraic Structures*, volume 99. CRC Press, 1985.

[PPDA17] Daniel Patterson, Jamie Perconti, Christos Dimoulas, and Amal Ahmed. Funtal: reasonably mixing a functional language with assembly. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 495–509. ACM, 2017.

[PV07] E. Palmgren and S.J. Vickers. Partial Horn logic and cartesian categorie. *Annals of Pure and Applied Logic*, 145(3):314 – 353, 2007.

[Ram06] Norman Ramsey. ML module mania: A type-safe, separately compiled, extensible interpreter. *Electron. Notes Theor. Comput. Sci.*, 148(2):181–209, 2006.

[Ram11] Norman Ramsey. Embedding an interpreted language using higher-order functions and types. *J. Funct. Program.*, 21(6):585–615, 2011.

[RJ98] Teodor Rus and James S. Jones. Phrase parsers from multi-axiom grammars. *Theor. Comput. Sci.*, 199(1-2):199–229, 1998.

[RS10] Grigore Rosu and Traian-Florin Serbanuta. An overview of the K semantic framework. *J. Log. Algebraic Methods Program.*, 79(6):397–434, 2010.

[RT91] Jan Reiterman and Vera Trnkovà. Free structures. In Horst Herrlich and Hans-Eberhard Porst, editors, *Category Theory at Work*, Research and Exposition in Mathematics, pages 277–288. Heldermann Verlag, 1991.

[Rus76] Teodor Rus. Context-free algebra: A mathematical device for compiler specifications. In Antoni W. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science 1976, 5th Symposium, Gdansk, Poland, September 6-10, 1976, Proceedings*, volume 45 of *Lecture Notes in Computer Science*, pages 488–494. Springer, 1976.

[RY20] Xavier Rival and Kwangkeun Yi. *Introduction to Static Analysis: An Abstract Interpretation Perspective.* Mit Press, 2020.

[Sco76] Dana S. Scott. Data types as lattices. *SIAM J. Comput.*, 5(3):522–587, 1976.

[Sco09] Michael L. Scott. *Programming Language Pragmatics (3. ed.).* Academic Press, 2009.

[SJ03] Fausto Spoto and Thomas P. Jensen. Class analyses as abstract interpretations of trace semantics. *ACM Trans. Program. Lang. Syst.*, 25(5):578–630, 2003.

[SS71] Dana S. Scott and Christopher Strachey. *Toward a mathematical semantics for computer languages*, volume 1. Oxford University Computing Laboratory, Programming Research Group Oxford, 1971.

[Tar68] Alfred Tarski. Equational logic and equational theories of algebras. In *Studies in Logic and the Foundations of Mathematics*, volume 50, pages 275–288. Elsevier, 1968.

[Tay79] Walter Taylor. Equational logic. *Houston Journal of Mathematics*, 47(2), 1979.

[TM07] Gang Tan and Greg Morrisett. Ilea: inter-language analysis across java and c. In Richard P. Gabriel, David F. Bacon, Cristina Videira Lopes, and Guy L. Steele Jr., editors, *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2007, October 21-25, 2007, Montreal, Quebec, Canada*, pages 39–56. ACM, 2007.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975.

[Vis97] Eelco Visser. *Syntax definition for language prototyping*. Ponsen & Looijen, 1997.

[Vis98] Eelco Visser. Polymorphic syntax definition. *Theor. Comput. Sci.*, 199(1-2):57–86, 1998.

[VM06] Alberto Verdejo and Narciso Martí-Oliet. Executable structural operational semantics in maude. *J. Log. Algebraic Methods Program.*, 67(1-2):226–293, 2006.

[WNL+10] Tobias Wrigstad, Francesco Zappa Nardelli, Sylvain Lebresne, Johan Östlund, and Jan Vitek. Integrating typed and untyped code in a scripting language. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 377–388. ACM, 2010.

Index